Toward Asymptotically Optimal Motion Planning for Kinodynamic Systems using a Two-Point Boundary Value Problem Solver

Christopher Xie Jur van den Berg Sachin Patil Pieter Abbeel

Abstract— **We present an approach for asymptotically optimal motion planning for kinodynamic systems with arbitrary nonlinear dynamics amid obstacles. Optimal sampling-based planners like RRT*, FMT*, and BIT* when applied to kinodynamic systems require solving a two-point boundary value problem (BVP) to perform exact connections between nodes in the tree. Two-point BVPs are non-trivial to solve, hence the prevalence of alternative approaches that focus on specific instances of kinodynamic systems, use approximate solutions to the two-point BVP, or use random propagation of controls. In this work, we explore the feasibility of exploiting recent advances in numerical optimal control and optimization to solve these two-point BVPs for arbitrary kinodynamic systems and how they can be integrated with existing optimal planning algorithms. We combine BIT* with a two-point BVP solver that uses sequential quadratic programming (SQP). We consider the problem of computing minimum-time trajectories. Since the duration of trajectories is not known a-priori, we include the time-step as part of the optimization to allow SQP to optimize over the duration of the trajectory while keeping the number of discrete steps fixed for every connection attempted. Our experiments indicate that using a two-point BVP solver in the inner-loop of BIT* is competitive with the state-of-the-art in sampling-based optimal planning that explicitly avoids the use of two-point BVP solvers.**

I. Introduction

Kinodynamic motion planning has been extensively studied over the past two decades [5], [14]. Since the introduction of RRT* by Karaman and Frazzoli in 2010 [11], techniques for asymptotically optimal motion planning have been an active area of study. RRT* allows finding asymptotically optimal paths by combining the traditional RRT [13] approach, which is only probabilistically complete and does not guarantee optimality, with a tree rewiring procedure. For this rewiring, RRT* needs to solve two-point boundary value problems [14], i.e., two points in the state space need to be exactly and optimally connected by a feasible path. RRT, on the other hand, only involves single-boundary value problems, where the tree is grown by forward-integrating the dynamics from the states in the tree.

Finding an optimal trajectory between two states for differentially constrained systems is non-trivial in general. For holonomic robots, the states can be connected by a straight line in configuration space. Prior works on extending RRT* for kinodynamic systems have focused on specific instances of kinodynamic systems [10], [23], or have used approximate

Christopher Xie, Sachin Patil, and Pieter Abbeel are with the Department of Electrical Engineering and Computer Science, University of California, Berkeley. Jur van den Berg is with the School of Computing, University of Utah, Salt Lake City.

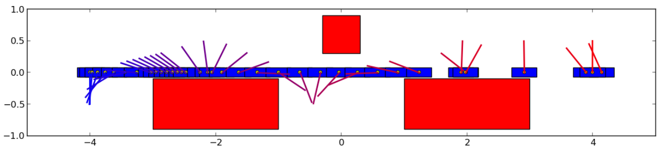


Fig. 1: A minimum-time collision-free trajectory for the cart-pole system found using our approach. The evolution of the trajectory is color coded according to time, with the start state shown in blue and the target state shown in red.

solutions to the two-point boundary value problem [9]. However, solving the two-point boundary value problem (BVP), for systems such as a cart-pole (Fig. 1) is challenging [1].

Therefore, there has been a particularly substantial research effort in designing asymptotically optimal algorithms that do not require solving two-point boundary value problems [17], [15]. These methods propagate the tree by shooting, i.e., randomly sampling controls and time durations. In addition, these methods have to consider a region around a goal state and cannot arrive at an exact goal state. A common assumption made in this line of work is that solving two-point boundary problems is impractical or not generally applicable to all robotics systems.

In this paper, we explore the feasibility of exploiting recent advances in numerical optimal control and optimization to solve these two-point BVP problems and how they can be integrated with existing asymptotically optimal planning algorithms such as RRT* [11] and recently proposed variants such as fast marching trees (FMT*) [8], and batch-informed trees (BIT*) [7] to plan optimal trajectories for kinodynamic systems with arbitrary, nonlinear dynamics. There has been considerable interest in the use of trajectory optimization for robotic motion planning [19], [22], [20], [25] but these works have been limited to kinematic motion planning. Recent work has explored the use of a nonlinear optimization method to solve the two-point BVP problem within a RRT* planner [21] but the proposed method requires additional trajectory smoothing steps to compute locally optimal solutions.

We use a variant of Sequential Quadratic Programming (SQP) to construct a two-point boundary value solver to (locally-) optimally connect nodes in the state space. SQP typically works on systems with discrete-time dynamics and a finite horizon – that is, only a pre-specified number of steps can be taken to solve the boundary-value problem. A key challenge here is that we do not know in advance what the duration of an optimal trajectory is. To this end, we keep the number of steps fixed, but include the duration of the discrete time steps to our optimization formulation. We consider the objective of computing minimum time trajectories but our formulation can also consider arbitrary state and control dependent objectives.

We present preliminary results on three kinodynamic systems: (i) double integrator, (ii) cart-pole system, and (iii) acrobot system. Our experiments indicate that our approach offers improvements in solution quality, solves a larger number of problems, and is comparable in performance as compared to state-of-the-art shooting-based approaches for asymptotically optimal kinodynamic planning [17], [15].

II. 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. The dynamics of the robot is defined by the following continuous-time system:

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

where $\mathbf{x}(t) \in \mathcal{X}$ is the state of the robot, and $\mathbf{u}(t) \in \mathcal{U}$ is the control input of the robot.

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

The *cost* $c(\pi)$ of a trajectory π is defined by:

$$c(\pi) = \int_0^\tau \ell(\mathbf{x}(t), \mathbf{u}(t))\, \mathrm{d}t, \tag{2}$$

where ℓ is an arbitrary cost functional. In this work, we compute the minimum-time trajectories, for which $c(\pi) = \tau$.

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. Here, we model $\mathcal{X}_{\text{free}}$ as the intersection of a number of nonlinear constraints:

$$\mathcal{X}_{\text{free}} = \bigcap_i \{\mathbf{x} \in \mathcal{X} \mid h_i(\mathbf{x}) > 0\}. \tag{3}$$

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. We assume this set is convex and modeled as the intersection of a number of linear constraints:

$$\mathcal{U}_{\text{free}} = \bigcap_i \{\mathbf{u} \in \mathcal{U} \mid \mathbf{a}_i^\top \mathbf{u} + b_i > 0\}. \tag{4}$$

The formal definition of the problem we discuss in this paper is then as follows: 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:

$$\begin{aligned}\pi^*_{\text{free}} = \operatorname{argmin}\{\pi \,|\, \mathbf{x}(0) = \mathbf{x}_{\text{start}} \wedge \mathbf{x}(\tau) = \mathbf{x}_{\text{goal}} \wedge \\ \forall \{t \in [0, \tau]\}\, (\mathbf{x}(t) \in \mathcal{X}_{\text{free}} \wedge \mathbf{u}(t) \in \mathcal{U}_{\text{free}})\}\; c(\pi).\end{aligned} \tag{5}$$

Note that the arrival time τ is not given, and optimizing it is part of the problem definition.

III. BIT*: Asymptotically Optimal Global Planning

We build on recent work by Gammell et al. [7] on efficient asymptotically optimal motion planning for holonomic robots. This work proposes the notion of Bellman Random Trees, which describe a class of planners that search implicit random geometric graphs (RGG) for explicit spanning trees. Examples of these planners are RRT* [11] and FMT* [8]. RRT* is an iterative, anytime planner that maintains a minimum cost tree over its samples and updates the tree in an anytime fashion in order to preserve its shortest path quality during its update. Fast marching trees (FMT*) is a bulk planner that performs a ”lazy” dynamic programming recursion that maintains an approximate minimum cost tree over a set of probabilistically drawn samples.

Batch-informed trees (BIT*) [7] combines the anytime nature of incremental RRT* and the efficiency of bulk sampling methods such as FMT* with admissible heuristics to focus the search. It is summarized as follows. An implicit RGG $\mathcal{G}_i$ is constructed by sampling m_i uniformly distributed samples from $\mathcal{X}_{\text{free}}$, and choosing a radius r_i such that it defines a neighbor relation on the nodes and is chosen such that it guarantees asymptotic optimality [11], [7]. Then a heuristically guided search builds a spanning tree until the entire graph has been searched. If a current solution exists, the cost of this solution provides a heuristically informed subset of $\mathcal{X}_{\text{free}}$ denoted $\mathcal{X}_{\hat{f}}$ such that any solution that is better than the current one lives in this subset. Then m_{i+1} new samples are added to construct a new RGG $\mathcal{G}_{i+1}$, and r_{i+1} is chosen to satisfy asymptotic optimality, and G_{i+1} is searched for an explicit spanning tree. This process repeats until a stopping condition is satisfied.

BIT* requires connecting two states exactly in state space. This leads to solving the two-point boundary value problem under differential constraints. Typically, this problem is considered very difficult. We remedy this with a generic solution to this problem which is applicable to any dynamics and cost functional in the following section.

In order for BIT* to perform an intelligent search, admissible heuristic functions must be provided. The examples provided in [7] use a heuristic that is equal to the true edge cost between two states. This is non-trivial for kinodynamic systems, and in general, heuristics for these systems are difficult to come up with. We used the Euclidean distance divided by maximum speed as a conservative heuristic in our implementation.

BIT* focuses its search using heuristics, which intelligently explores the state space. As a result, the number of calls to the true cost function that connects two states (i.e. two-point BVP solver) is much lower than RRT* [7]. This makes it preferable to use as a globally optimal planner as opposed to RRT*, since our BVP solver incurs computational overhead when invoked a large number of times.

IV. Two-Point Boundary Value Problem Solver

In this section we describe our two-point BVP solver to exactly connect a given pair of states, which is a key ingredient of BIT*. Typically, two-point BVP solvers are not full motion planners, meaning that they do not account for obstacles. The resulting trajectory is evaluated by the higher-level planning algorithm and discarded in case the trajectory is not collision-free.

We use a variant of sequential quadratic programming (SQP) to compute local plans. It computes a locally optimal solution to a full motion planning problem, but may fail if the problem proves to be too difficult. SQP typically works on systems with discrete-time dynamics and a finite horizon – that is, only a pre-specified number of steps can be taken to solve the planning problem. We do not know in advance what the duration of an optimal trajectory is; to this end, we keep the number of steps fixed, but make the time step that each of these steps comprise a variable, and include it in the optimization formulation.

A. Discretizing the Dynamics

Given an initial state $\mathbf{x}(t)$ and a constant control input $\mathbf{u}$, the evolution of the state over a given amount of time λ is given by the solution to the differential equation of Eq. (1), which we denote by a function $\tilde{\mathbf{f}}$:

$$\mathbf{x}(t+\lambda) = \tilde{\mathbf{f}}(\mathbf{x}(t), \mathbf{u}, \lambda). \tag{6}$$

We note that the function $\tilde{\mathbf{f}}$ is in general difficult to compute analytically, but it can be approximated closely with for instance Runge-Kutta integration (see Appendix A). Also, let the cost accumulated over an amount of time λ given an initial state $\mathbf{x}$ and constant control input $\mathbf{u}$ be given by function $\tilde{\ell}$:

$$\tilde{\ell}(\mathbf{x}, \mathbf{u}, \lambda) = \int_0^\lambda \ell(\tilde{\mathbf{f}}(\mathbf{x}, \mathbf{u}, t), \mathbf{u}) \, \mathrm{d}t. \tag{7}$$

The above functions constitute the discrete-time dynamics and the discrete-time cost function if we subdivide the time axis into discrete steps of equal duration λ.

B. Optimization Problem

With the definitions of the discrete-time dynamics and the discrete-time cost we can now reformulate the local motion planning problem as a non-convex program. Let $\mathbf{x}_k$ denote the state at stage k, and let $\mathbf{u}_k$ denote the constant control input applied between stage k and stage $k+1$, where λ is the amount of time between stages. We then get:

$$\text{find:} \quad \lambda, \mathbf{x}_0, \ldots, \mathbf{x}_s, \mathbf{u}_0, \ldots, \mathbf{u}_{s-1} \tag{8}$$

$$\text{that minimize:} \quad \sum_{k=0}^{s-1} \tilde{\ell}(\mathbf{x}_k, \mathbf{u}_k, \lambda) \tag{9}$$

$$\text{such that:} \quad \mathbf{x}_0 = \mathbf{x}_{\text{start}}, \tag{10}$$

$$\mathbf{x}_s = \mathbf{x}_{\text{goal}}, \tag{11}$$

$$\mathbf{x}_{k+1} = \tilde{\mathbf{f}}(\mathbf{x}_k, \mathbf{u}_k, \lambda), \ \forall k \in 0 \ldots s-1 \tag{12}$$

$$\mathbf{u}_k \in \mathcal{U}_{\text{free}}, \ \forall k \in 0 \ldots s-1 \tag{13}$$

$$\mathbf{x}_k \in \mathcal{X}_{\text{free}}, \ \forall k \in 1 \ldots s-1 \tag{14}$$

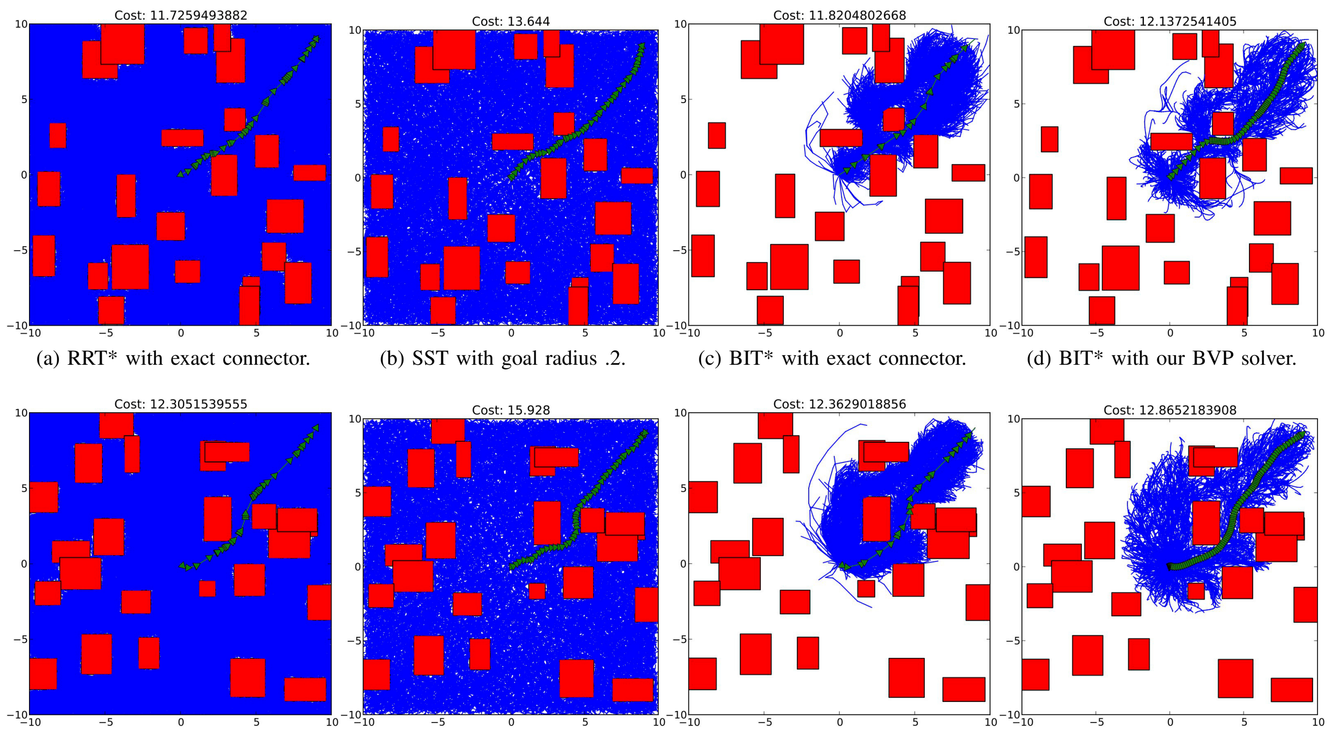
$$\lambda \geq 0 \tag{15}$$

The number of stages s comprising the motion plan is a preset constant. The duration of the trajectory is variable given that the duration λ of each step is part of the optimization.

We use sequential quadratic programming (SQP) to locally optimize the non-convex, constrained optimization problem that results from the two-point BVP formulation. SQP [24] optimizes problems in parameter $\mathbf{y}$ of the form $\min_{\mathbf{y}} \tilde{\ell}(\mathbf{y})$ subject to constraints. One repeatedly constructs a quadratic program (quadratic objective and linear constraints) that locally approximates the original problem around the current solution $\mathbf{y}$ by linearizing the nonlinear constraints and quadratizing the cost functional. Then one solves the quadratic program to compute a step $\Delta\mathbf{y}$ that make progress on the original problem. Two necessary ingredients in a SQP implementation are trust regions and merit functions. A trust region constrains $\mathbf{y}$ in each subproblem to the region where the approximation is valid. The trust region is adaptively changed based on the merit function, which has the form $\tilde{\ell}_\mu(\mathbf{y}) = \tilde{\ell}(\mathbf{y}) + \mu \cdot \text{ConstraintViolation}(\mathbf{y})$. Here, μ is a given penalty parameter that penalizes violations of nonlinear constraints, and it ensures that the steps taken by the algorithm make progress on both the cost function $\tilde{\ell}(\mathbf{y})$ and the constraints. The optimization algorithm solves a series of problems $\min_{\mathbf{y}} \tilde{\ell}_{\mu_0}(\mathbf{y}), \min_{\mathbf{y}} \tilde{\ell}_{\mu_1}(\mathbf{y}), \ldots, \min_{\mathbf{y}} \tilde{\ell}_{\mu_n}(\mathbf{y})$ for $\mu_0 < \mu_1 < \cdots < \mu_n$ where the penalty parameter μ is sequentially increased in an outer loop. We used sequential quadratic programming (SQP) with ℓ_1 penalties, also used by Schulman et al. [20] for kinematic motion planning.

At the core of the SQP method is a QP solver. We efficiently solve the underlying QPs using a numerical optimization code generation framework called FORCES [4]. FORCES generates code for solving QPs that is based on the interior-point method and is specialized for convex multistage problems such as trajectory optimization. Automatic code generation for convex solvers has gained popularity since it is able to exploit the fact that all problem dimensions and the structure of the problem are known a priori. This permits generation of highly customized and fast solver code that solves instances of a particular problem. We use this solver for all our experiments.

In order to make sure that each of the convex optimization problems that are solved in sequence have a solution, we must set the number of steps s sufficiently large. If we assume that the linearized dynamics of each step and in each iteration are *controllable*, then setting $s > n$ (recall that n is the dimension of the state space) guarantees the existence of a solution in absence of any constraints. However, setting $s = n+1$ results in only one possible solution, that is in general far from the optimum of the underlying continuous-time problem definition. In order to approach such optimum, we need $\lambda \to 0$ and therefore set $s \to \infty$, but that is not practical. Therefore, we choose a value in between and set s to be $3n$ in our implementation. Note that even with setting $s >> n$, the SQP procedure may not result in a solution because of the inequality constraints. In this case, we ignore the connection, and treat it in the same way as a connection that is not collision-free in traditional motion planners.



(a) RRT* with exact connector. (b) SST with goal radius .2. (c) BIT* with exact connector. (d) BIT* with our BVP solver.

(e) RRT* with exact connector. (f) SST with goal radius .2. (g) BIT* with exact connector. (h) BIT* with our BVP solver.

Fig. 2: Solution plots for double integrator model after running for 10 minutes. The top row has solutions from obstacle scene 1, and the bottom row has solutions from obstacle scene 2. The blue indicates the tree found by the algorithm, and the green lines and arrows denote the solution found. Note that for RRT* with the exact connector and SST, the trees are dense, while BIT* focuses its search.

V. Experimental Results

We tested our method on three differential systems; a simple double integrator system in 2D configuration space, a cart-pole system with obstacles, and an acrobot system with obstacles. All experiments were implemented to minimize time to get from a start state to a goal state. We compared SST [15], a state of the art shooting approach, with our approach. We used the SST library provided by Kostas and Littlefield [16]. Our implementation was in C++ and ran with on a single 3.2 Ghz Intel processor. Our simulation results are detailed in sections V-A to V-C.

A. Double Integrator

The double integrator model is a simple model described by position in a nD space (our configuration space is $n =$ 2D, our state space is 4D), and velocities for each coordinate axis. The linear dynamics of the system is described by:

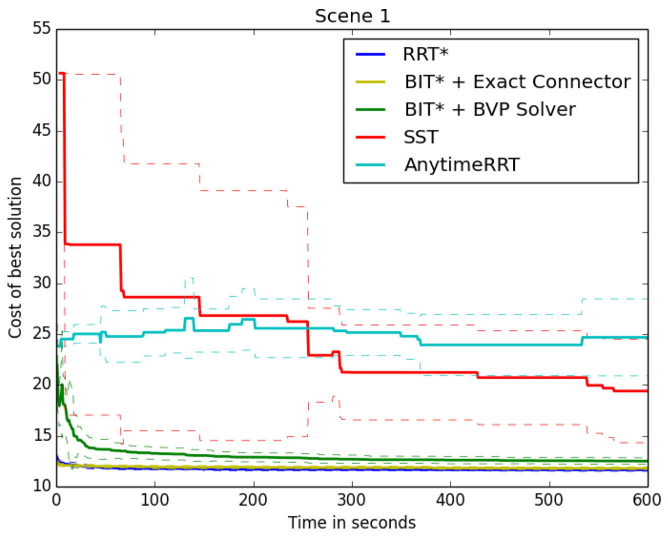
$$\mathbf{x} = \begin{bmatrix} \mathbf{p} \\ \dot{\mathbf{p}} \end{bmatrix}, \quad \dot{\mathbf{x}} = \begin{bmatrix} 0 & I_n \\ 0 & 0 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 \\ I_n \end{bmatrix} \mathbf{u}.$$

Here, the control $\mathbf{u} = \ddot{\mathbf{p}}$ is acceleration, where $\mathbf{p}$ is a vector that describes the position of the point robot and $\dot{\mathbf{p}}$ is the velocity. We restricted our variables in this way: $\mathbf{p} \in [-10, 10]^2(\mathrm{m}), \dot{\mathbf{p}} \in [-1, 1]^2(\mathrm{m/s}), \mathbf{u} \in [-1, 1]^2(\mathrm{m/s}^2)$. We ran our experiments with a point robot in two different randomly generated obstacle scenarios as depicted in Fig. 2. For each of these experiments, the start state was $[0, 0, 0, 0]^\top$ and the goal state was $[9, 9, 0, 0]^\top$.

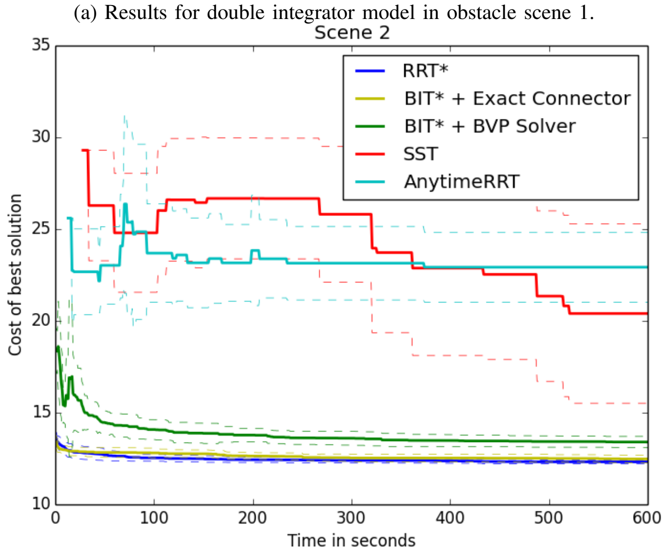
We ran RRT* using the implementation provided by the SMP library [12] which included the double integrator model and an exact connector for this system under bounded controls and states. We then took the exact connector and plugged it into our implementation of BIT*. For BIT*, we used a batch size of 150. SST requires a goal radius due to its random propagation methods which we set to .2, below which SST did not find a solution. For other SST parameters, we set $\delta_v = .6, \delta_s = .3$ so that it is able to find a solution through narrow passages in the scene. For this system, we also compared to our implementation of Anytime RRTs [6], using a goal radius of .2, $\epsilon_f = 0.05, \delta_d = \delta_c = 0.15$. Each experiment was run for 10 minutes each. Visual solutions of these algorithms can be found in Fig. 2. Our heuristic function for estimating edge cost in BIT* was difference in position divided by max speed. The speed of the point robot at time t is equal to the norm of the velocity vector $\dot{\mathbf{p}}(t)$. We used the same heuristic function for Anytime RRTs.

In an effort to speed up our implementation, we pruned calls to the BVP solver based on these criteria: the orientation (direction of the velocity vector $\dot{\mathbf{p}}$) of these two states differed by more than $\frac{\pi}{2}$, or the start state was in "front" of the goal state (e.g., the projection of the goal state on the line described by the position and orientation of the start state is negative). We initialized states to be straight line trajectories from start to goal, the controls to zero, and the duration of the time step to be the distance between the two states divided by the number of time steps.

The graphs in Fig. 3 show the best paths found by the



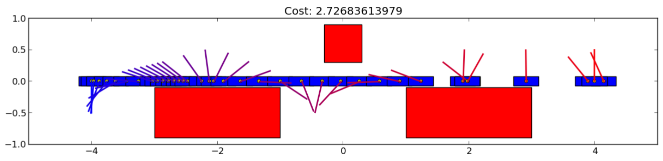
(a) Results for double integrator model in obstacle scene 1.



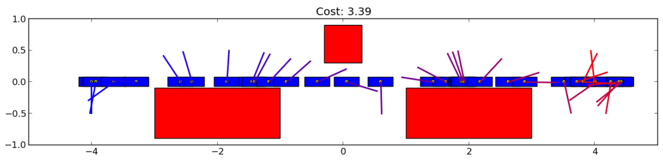
(b) Results for double integrator model in obstacle scene 2.

Fig. 3: Double integrator. The solid color coded lines indicate solution values averaged over 10 runs. The dashed lines indicate the standard deviations. The left graph corresponds to scene 1 and the right graph corresponds to scene 2. The averages were computed only over the runs that had found a solution at a given time, which explains the jumps and gaps in the best cost.

algorithms versus computation time. These are averaged over 10 runs each. Note that BIT* with the exact connector finds very similar values to RRT*. BIT* with our generic BVP solver does not fall far behind, while SST computes a trajectory that is approximately twice the cost of the others. 2 out of the 10 SST runs did not find a solution; they were left out of these calculations. Note that SST uses a goal radius of .2 while the other three algorithms connect the start and goal states exactly. This gives a slight advantage to SST in that the true optimum of the shortest path from the start state to the goal region (in this case, the region is a hypersphere) is less than or equal to the true optimum for the other three problems. Anytime RRTs seems to find a lower quality solution as compared to SST to begin with but plateaus towards the end and it is not clear if the path quality converges to the optimal value.



(a) BIT* with our BVP solver for obstacle scene 1.



(b) SST for obstacle scene 1. The final goal state found by this algorithm is $[-8.6 \times 10^{-3}, -.106, -3.03, 3.99]^{\top}$.

Fig. 4: Solution plots for cart-pole model after running for 10 minutes. The poles are color coded according to time. The start state has a blue pole, and evolves towards a red goal state.

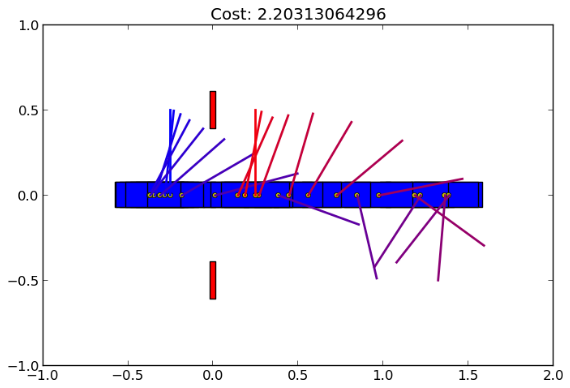


Fig. 5: BIT* with our BVP solver for obstacle scene 2. Solution plots for cart-pole model after running for 10 minutes. The poles are color coded according to time. The start state has a blue pole, and evolves towards a red goal state.

The main difference in running time between BIT* with the exact connector and BIT* with the BVP solver is the numerous calls to the BVP solver, which is the bottleneck of the algorithm. On average, it takes 0.013s to run the BVP solver with a standard deviation of 0.018s, and 2×10^{-7}s to run the exact connector with a standard deviation of 4.8×10^{-5}s, which shows that our BVP solver is a factor of 10^5 times slower than the exact connector. This motivated the need to speed up our implementation by pruning.

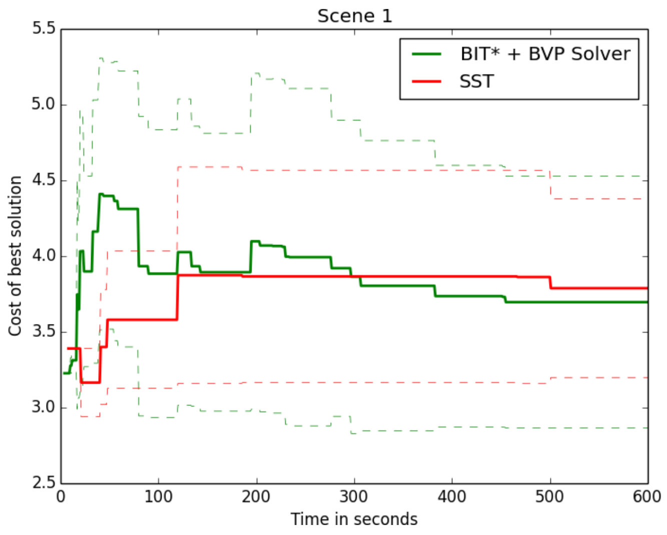
B. *Cart-pole*

The cart-pole system is a nonlinear system described by the following dynamics [3]:

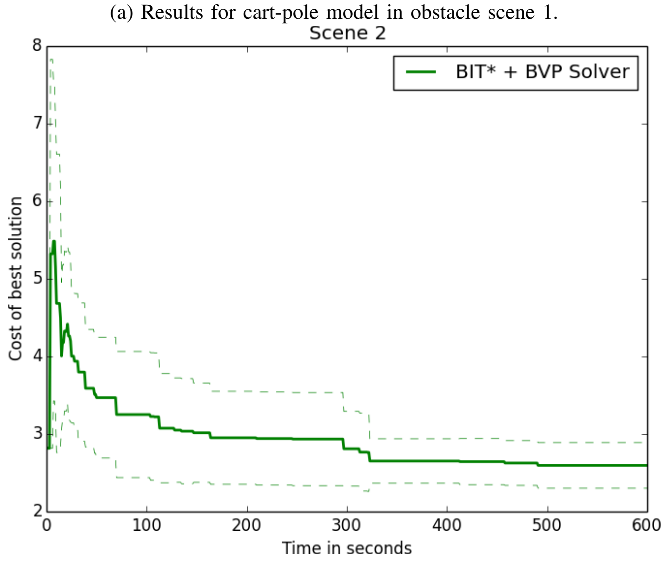
$$\mathbf{x} = [\dot{\theta} \quad \dot{p} \quad \theta \quad p]^{\intercal},$$

$$\dot{\mathbf{x}} = \begin{bmatrix} \frac{-3m_2 l\dot{\theta}^2 \sin\theta\cos\theta - 6(m_1+m_2)g\sin\theta - 6(u-b\dot{p})\cos\theta}{4l(m_1+m_2)-3m_2 l\cos^2\theta} \\ \frac{2m_2 l\dot{\theta}^2\sin\theta + 3m_2 g\sin\theta\cos\theta + 4u - 4b\dot{p}}{4(m_1+m_2)-3m_2 l\cos^2\theta} \\ \dot{\theta} \\ \dot{p} \end{bmatrix}.$$

The state space is 4D and the control is 1D, which is the external force applied to the cart. m_1 denotes the mass of the cart, m_2 denotes the mass of the pole, l denotes the length of the pole, θ denotes the angle of the pendulum, p denotes



(a) Results for cart-pole model in obstacle scene 1.



(b) Results for cart-pole model in obstacle scene 2.

Fig. 6: Results for cart-pole model. The solid color coded lines indicate solution values averaged over 10 runs. The dashed lines indicate the standard deviations. The left graph corresponds to scene 1 and the right graph corresponds to scene 2. The averages were computed only over the runs that had found a solution at a given time, which explains the jumps and gaps in the best cost. Note that for scene 2, SST found no solutions for a goal radius of 0.2, thus this graph only contains a plot for our method.

the position of the cart, b denotes the friction between the cart and the ground, and $g = 9.8$ (m/s^2) is acceleration due to gravity. We chose $m_1 = .5$ kg, $m_2 = .5$ kg, $l = .5$ m, and $b = .1$ N/m/s. For SST, we set $\delta_v = .8, \delta_s = .4$. A batch size of 25 was chosen for BIT*.

For scene 1, we constrained the problem to $p \in [-10, 10]$ (m), $\dot{p} \in [-10, 10]$ (m/s), $\dot{\theta} \in [-10, 10]$ (rad/s), $\mathbf{u} \in [-20, 20]$ (N). For each experiment, the start state was $[0, 0, 0, -4]^\top$ and the goal state was $[0, 0, \pi, 4]^\top$, e.g. the goal state must be at position 4 with the pole propped up and balanced. We set a goal radius of .4 for SST, below which SST did not find a solution.

For scene 2, as proposed by [18], we constrained the problem to $p \in [-5, 5]$ (m), $\dot{p} \in [-5, 5]$ (m/s), $\dot{\theta} \in [-10, 10]$ (rad/s), $\mathbf{u} \in [-10, 10]$ (N). For each experiment, the start state was $[0, 0, \pi, -.25]^\top$ and the goal state was $[0, 0, \pi, .25]^\top$. To accommodate the smaller scale of this problem, we had to set a goal radius of .2. Anything bigger and SST would deem a state where the cart is on the other side of the obstacle (with the pole propped up) a solution. The obstacle layouts can be seen in Fig. 4. The heuristic function we used for this system is very similar to the heuristic function we used for the double integrator. We take the difference in configurations and divide it by the maximum speed.

Of the 10 runs, 5 of the SST runs found solutions which are included in the averages in Fig. 6 for scene 1. BIT* with our BVP solver found solutions for every run. We re-initialized our duration of each time step if our BVP solver did not make progress after the first iteration. Calling this BVP solver took on average 0.0394s with a standard deviation of 0.063s. For scene 2, SST found no solutions over 10 runs for a goal radius of 0.2, while our method found solutions for every run. Note that the goal radius for scene 1 (.4) is quite large. This means that the goal state found by SST does not have to be propped exactly upright, or completely stopped (e.g. zero positional velocity) as shown in Fig. 4 (b). Again, the true optimum of this problem is less than or equal to the true optimum of the problem that BIT* is trying to solve.

C. Acrobot

The acrobot system is two-link nonlinear system with two joints. Let $\mathbf{x} = [\boldsymbol{\theta} \;\; \dot{\boldsymbol{\theta}}]^\intercal$, $\boldsymbol{\theta} = [\theta_1 \;\; \theta_2]^\intercal$ be the 4D state of the system. The system is described by dynamics given by [2]:

$$\ddot{\boldsymbol{\theta}} = \mathbf{D}^{-1}(\boldsymbol{\theta}) \left(\mathbf{T} - \mathbf{C}(\dot{\boldsymbol{\theta}}, \boldsymbol{\theta}) - \boldsymbol{\Phi}(\boldsymbol{\theta}) \right),$$

where $\mathbf{D}(\boldsymbol{\theta})$ is a 2×2 matrix with elements

$$d_{11} = m_1 l_{c1}^2 + m_2 \left(l_1^2 + l_{c2}^2 + 2 l_{c2} l_1 \cos(\theta_2) \right) + I_1 + I_2,$$
$$d_{22} = m_2 l_{c2}^2 + I_2,$$
$$d_{12} = d_{21} = m_2 \left(l_{c2}^2 + l_{c2} l_1 \cos(\theta_2) \right) + I_2,$$

$\mathbf{T}$ is a 2×1 matrix with elements

$$t_1 = -K\dot{\theta}_1, \quad t_2 = \tau - K\dot{\theta}_2,$$

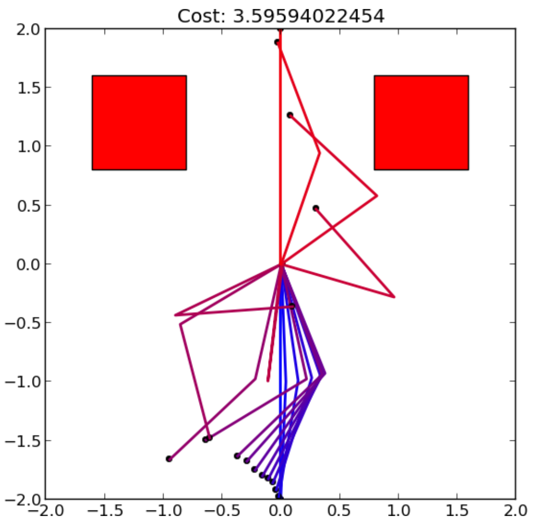
$\mathbf{C}(\dot{\boldsymbol{\theta}}, \boldsymbol{\theta})$ is a 2×1 matrix with elements

$$c_1 = -m_2 l_1 l_{c2} \dot{\theta}_2^2 \sin(\theta_2) - 2 m_2 l_1 l_{c2} \dot{\theta}_1 \dot{\theta}_2 \sin(\theta_2),$$
$$c_2 = m_2 l_1 l_{c2} \dot{\theta}_1^2 \sin(\theta_2),$$

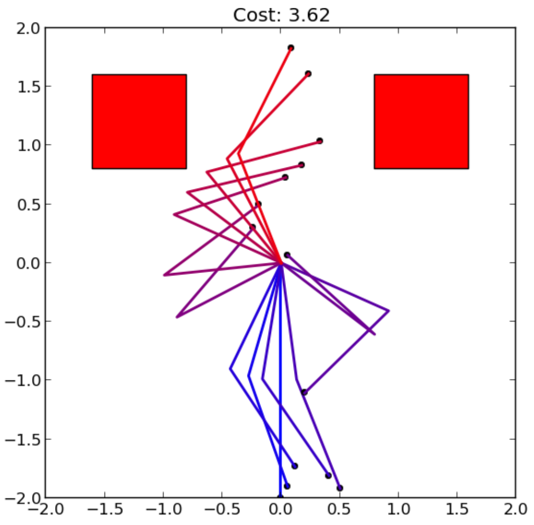
and $\boldsymbol{\Phi}(\boldsymbol{\theta})$ is a 2×1 matrix with elements

$$\phi_1 = (m_1 l_{c1} + m_2 l_1)\, g \cos(\theta_1) + m_2 l_{c2} g \cos(\theta_1 + \theta_2),$$
$$\phi_2 = m_2 l_{c2} g \cos(\theta_1 + \theta_2).$$

The control, τ is the external torque applied at the middle joint. θ_1 denotes the angle of the first link with the negative y-axis, θ_2 denotes the relative angle of the first link and the second link, m_1, m_2 denotes the masses of the two links, l_1, l_2 denotes the length of the two links, l_{c1}, l_{c2} denotes the length to the mass center of the two links, I_1, I_2 denotes the link inertias, K is a damping coefficient, and $g = 9.8$ m/s^2.



(a) BIT* with our BVP solver.



(b) SST with goal radius .2.

Fig. 7: Solution plots for the acrobot model after running for 10 minutes. The links are color coded according to time. The start state has a blue pole, and evolves towards a red goal state.

We chose $m_1 = m_2 = 1$ kg, $l_1 = l_2 = 1$ m, $l_{c1} = l_{c2} = .5$ m, $I_1 = .2$ kg$\cdot$m^2, $I_2 = 1$ kg$\cdot$m^2, and $K = 1$ Ns/m.

For our obstacle scene, we constrained the problem to $\dot{\theta_1}, \dot{\theta_2} \in [-6, 6]$ (rad/s), $\tau \in [-8, 8]$ N$\cdot$m. For each experiment, the start state was $[0, 0, 0, 0]^\intercal$ and the goal state was $[\pi, 0, 0, 0]^\intercal$, e.g. both links must be standing straight up with no angular velocity at the joints. We set a goal radius of .2 for SST. The obstacle layouts can be see in Fig. 7.

Our heuristic function to estimate the cost of an edge between two states $\mathbf{x}_1$ and $\mathbf{x}_2$ for this system is $\max\{\frac{|\theta_1(\mathbf{x}_1) - \theta_1(\mathbf{x}_2)|}{v_{\max}}, \frac{|\theta_2(\mathbf{x}_1) - \theta_2(\mathbf{x}_2)|}{v_{\max}}\}$, where $\theta_1(\mathbf{x})$ denotes the value of θ_1 in state $\mathbf{x}$, and $v_{\max}$ is the maximum angular velocity in the problem.

Of the 10 runs for SST, each run found a solution. This was the case as well for BIT* with our BVP solver. We reinitialized our duration of each time step if our acrobot BVP solver did not make progress after the first iteration. Calling this solver took on average 0.24s with a standard deviation of 0.347. In the SST implementation [16], a goal radius was specified in terms of the distance between the end effector positions at two acrobot states. In all cases of the SST runs, the goal state was not upright because of the goal radius (Fig. 7b).

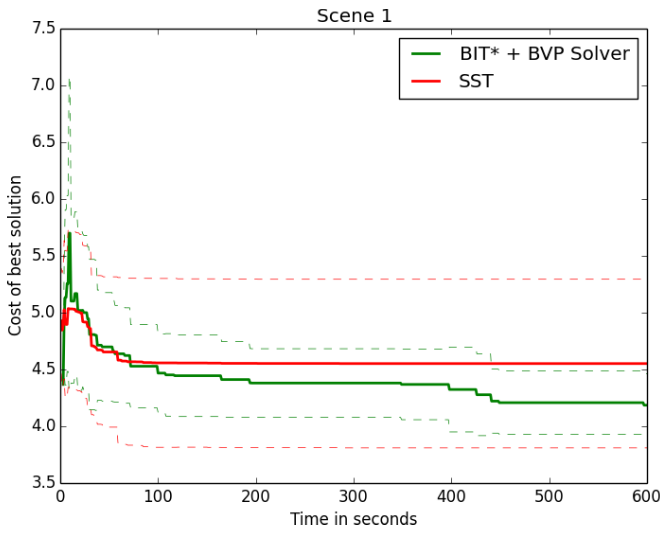


Fig. 8: Results for acrobot model. The solid color coded lines indicate solution values averaged over 10 runs. The dashed lines indicate the standard deviations. The left graph corresponds to scene 1 and the right graph corresponds to scene 2. The averages were computed only over the runs that had found a solution at a given time, which explains the jumps and gaps in the best cost.

VI. Discussion, Conclusion, and Further Work

In this work, we explore the feasibility of exploiting recent advances in numerical optimal control and optimization to solve two-point BVPs for arbitrary kinodynamic systems and how they can be integrated with BIT* [7], a sampling-based optimal planner. BIT* combines the anytime nature of RRT* with the bulk efficiency of FMT* with heuristic functions for intelligent search over a sequence of implicit RGGs. We construct discretized versions of the dynamics and cost by approximating solutions to differential equations and integrals by numerical methods such as Runge-Kutta integration. With this discretized form, we use sequential quadratic programming (SQP), which iteratively constructions quadratic program approximations to the original non-convex problem in order to find a local optimum. Since the duration of trajectories is not known a-priori, we include the time-step as part of the optimization to allow SQP to optimize over the duration of the trajectory while keeping the number of discrete steps fixed.

Our preliminary results on the double integrator, cart-pole system, and acrobot system are promising and indicate that using a two-point BVP solver in the inner-loop of BIT* is competitive with the state-of-the-art in sampling-based optimal planning that explicitly avoids the use of BVP solvers. With recent advances in optimization and optimal control, we believe that using two-point BVP solvers for kinodynamic motion planning will emerge as an active research area. We intend to explore the notion of optimality of planners such as RRT* and BIT* when using a discrete-time two-point BVP solver to exactly connect states in the tree.

Acknowledgments

We would like to thank Jonathan Gammell, Siddhartha Srinivasa, and Tim Barfoot for providing us with early access to the BIT* source code, a working draft of the paper [7], and for insightful discussions. We also thank Zakary Littlefield and Kostas Bekris for proving us with the SST source code [15] and helping us out with the comparison experiments and Sertac Karaman for providing access to the SMP library. This work is supported in part by the U.S. National Science Foundation under Award IIS-1227536: Multilateral Manipulation by Human-Robot Collaborative Systems, and by DARPA Young Faculty Award #D13AP00046.

References

[1] J. T. Betts. Survey of numerical methods for trajectory optimization. *Journal of guidance, control, and dynamics*, 21(2):193-207, 1998.

[2] G. Betts. Minimum-time control of the acrobot. *IEEE Int. Conf. on Robotics and Automation*, 3281-3287, 1997.

[3] M. Deisenroth, C. E. Rasmussen. PILCO: A model-based and data-efficient approach to policy search. *Proc. Int. Conf. on Machine Learning (ICML)*, 465-472, 2011.

[4] A. Domahidi. FORCES: Fast optimization for real-time control on embedded systems. Available at forces.ethz.ch (2012).

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

[6] D. Ferguson, A. Stentz. Anytime RRTs. *Int. Conf. on Intelligent Robots and Systems (IROS)*, 2006.

[7] J. Gammell, S. S. Srinivasa, T. D. Barfoot. BIT*: Batch Informed Trees for Optimal Sampling-based Planning via Dynamic Programming on Implicit Random Geometric Graphs. *Int. Conf. on Robotics and Automation (ICRA)*, 2015.

[8] L. Janson, and M. Pavone. Fast Marching Trees: a Fast Marching Sampling-Based Method for Optimal Motion Planning in Many Dimensions–Extended Version. *arXiv preprint arXiv:1306.3532*, 2013.

[9] 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.

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

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

[12] S. Karaman. Sampling-based Motion Planning (SMP) Library. 2014.

[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] Y. Li, Z. Littlefield, K. E. Bekris. Asymptotically Optimal Sampling-based Kinodynamic Planning. *arXiv preprint arXiv:1407.2896*, 2014.

[16] Z. Littlefield, K. E. Bekris. Stable Sparse RRT (SST): Efficient Asymptotically Near-Optimal Kinodynamic Motion Planning. Available at pracsyslab.org, 2014.

[17] G. Papadopoulos, H. Kurniawati, N. M. Patrikalakis. Analysis of Asymptotically Optimal Sampling-based Motion Planning Algorithms for Lipschitz Continuous Dynamical Systems. *arXiv preprint arXiv:1405.2872*, 2014.

[18] S. Ramasamy, G. Wu, K. Sreenath. Dynamically feasible motion planning through partial differential flatness. *Robotics: Science and Systems*, 2014.

[19] K. Rawlik, M. Toussaint, S. Vijayakumar. An approximate inference approach to temporal optimization in optimal control. *Advances in Neural Information Processing Systems*, 2010.

[20] J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, P. Abbeel. Finding locally optimal, collisi-on-free trajectories with sequential convex optimization. *Robotics: Science and Systems*, 2013.

[21] S. Stoneman, R. Lampariello. Embedding nonlinear optimization in RRT* for optimal kinodynamic planning. *IEEE Conf. on Decision and Control (CDC)*, 2014.

[22] J. van den Berg. Extended LQR: Locally-Optimal Feedback Control for Systems with Nonlinear Dynamics and Non-Quadratic Cost. *Proc. Int. Symp. on Robotics Research*, 2013.

[23] D. Webb, J. van den Berg. Kinodynamic RRT*: Asymptotically Optimal Motion Planning for Robots with Linear Dynamics. *Proc. IEEE Int. Conf. on Robotics and Automation*, 2013.

[24] S. J. Wright and J. Nocedal. Numerical optimization. Vol. 2. New York: Springer, 1999.

[25] M. Zucker, N. Ratliff, A. Dragan, M. Pivtoraiko, M. Klingensmith, C. Dellin, J. Bagnell, S. Srinivasa. CHOMP: Covariant Hamiltonian optimization for motion planning. *Int. Journal of Robotics Research*, 2013.

Appendix

Discretizing Dynamics and Cost

We approximate the discretized dynamics $\tilde{\mathbf{f}}(\mathbf{x}, \mathbf{u}, \lambda)$ and discretized cost $\tilde{\ell}(\mathbf{x}, \mathbf{u}, \lambda)$ given a state $\mathbf{x}$, a control input $\mathbf{u}$, and a time-step duration λ using Runge-Kutta-4 integration of the continuous-time dynamics $\mathbf{f}$ and cost ℓ. The values of these functions are best computed simultaneously:

$$\tilde{\mathbf{f}}(\mathbf{x}, \mathbf{u}, \lambda) = \mathbf{x} + (\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4)/6, \tag{16}$$

$$\tilde{\ell}(\mathbf{x}, \mathbf{u}, \lambda) = (k_1 + 2k_2 + 2k_3 + k_4)/6, \tag{17}$$

where

$$\mathbf{k}_1 = \lambda \mathbf{f}(\mathbf{x}, \mathbf{u}), \qquad k_1 = \lambda \ell(\mathbf{x}, \mathbf{u}), \tag{18}$$

$$\mathbf{k}_2 = \lambda \mathbf{f}(\mathbf{x} + \mathbf{k}_1/2, \mathbf{u}), \qquad k_2 = \lambda \ell(\mathbf{x} + \mathbf{k}_1/2, \mathbf{u}), \tag{19}$$

$$\mathbf{k}_3 = \lambda \mathbf{f}(\mathbf{x} + \mathbf{k}_2/2, \mathbf{u}), \qquad k_3 = \lambda \ell(\mathbf{x} + \mathbf{k}_2/2, \mathbf{u}), \tag{20}$$

$$\mathbf{k}_4 = \lambda \mathbf{f}(\mathbf{x} + \mathbf{k}_3, \mathbf{u}), \qquad k_4 = \lambda \ell(\mathbf{x} + \mathbf{k}_3, \mathbf{u}). \tag{21}$$