Differential Dynamic Programming with Nonlinear Constraints

Zhaoming Xie[1] C. Karen Liu[2] Kris Hauser[3]

***Abstract*— Differential dynamic programming (DDP) is a widely used trajectory optimization technique that addresses nonlinear optimal control problems, and can readily handle nonlinear cost functions. However, it does not handle either state or control constraints. This paper presents a novel formulation of DDP that is able to accommodate arbitrary nonlinear inequality constraints on both state and control. The main insight in standard DDP is that a quadratic approximation of the value function can be derived using a recursive backward pass, however the recursive formulae are only valid for unconstrained problems. The main technical contribution of the presented method is a derivation of the recursive quadratic approximation formula in the presence of nonlinear constraints, after a set of active constraints has been identified at each point in time. This formula is used in a new Constrained-DDP (CDDP) algorithm that iteratively determines these active set and is guaranteed to converge toward a local minimum. CDDP is demonstrated on several underactuated optimal control problems up to 12D with obstacle avoidance and control constraints and is shown to outperform other methods for accommodating constraints.**

I. INTRODUCTION

Nonlinear optimal control problems have seen numerous applications in science and engineering, and a variety of trajectory optimization techniques have been developed to solve them, including direct collocation methods [1], shooting methods, differential dynamic programming (DDP) [2], and the iterative linear quadratic regulator (iLQR) which is highly related to DDP [3]. Differential dynamic programming (DDP) is an iterative method that decomposes a large problem across a control sequence into a recursive series of small problems, each over an individual control at a single time instant, solved backwards in time. Its key insight is that the value function can be approximated by a quadratic fit around the current trajectory, and that this fit can be calculated recursively in analytical form. By iteratively moving toward the minima of the quadratic approximations, the trajectory is progressively improved toward a local optimum with superlinear convergence. The key advantage of DDP (and the closely related iLQG) approach over collocation methods is that the size of each smaller problem is time-independent, since no intermediate matrices are larger than $n \times n$ where n is the state-space dimension. However, collocation methods have the advantage that they can handle state and control constraints using nonlinear program solvers like sequential quadratic programming.

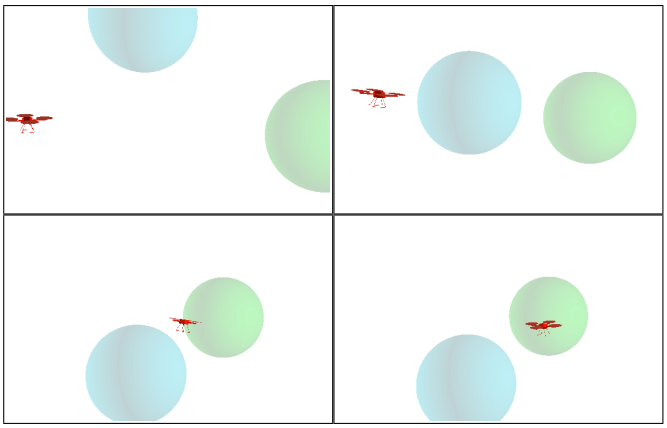

Fig. 1: CDDP computes a trajectory for a quadcopter flying toward the goal state while avoiding two moving spheres in the 3D space. In this camera view, the blue sphere moves from top to bottom while the green sphere moves away from the camera.

In this paper we present an extension to DDP that handles nonlinear constraints on both state and control. Prior research has indeed considered incorporating constraints in DDP, but those approaches have either been limited to linear systems, or linear constraints only on control, or fail to properly handle infeasible QP sub-problems. Another method sometimes used is to apply some barrier function that penalizes proximity to constraints, converting the constrained problem into an unconstrained one. However, our experiments find that this approach is more susceptible to local minima. To our knowledge, no DDP variant that handles nonlinear constraints has been successfully applied to higher-dimensional problems like those exhibited in underactuated robotics and vehicle control. Our method is demonstrated on simulated dynamic vehicle obstacle avoidance problems, including a quadcopter with control constraints avoiding multiple moving obstacles (Fig. 1).

II. RELATED WORK

Differential Dynamic Programming is a well established method for nonlinear trajectory optimization [2] that uses an analytical derivation of the optimal control at each point in time according to a second order fit to the value function. These problems are recursive in nature and solved backward in time, starting from a given time horizon. The iterative

*This work is partially supported by NSF grants IIS #1218534 and CAREER #1253553.

[1]Zhaoming Xie is with School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA zxie47@gatech.edu

[2]C. Karen Liu is with School of Interactive Computing, Georgia Institute of Technology, Atlanta, GA 30332, USA karenliu@cc.gatech.edu

[3]Kris Hauser is with the Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708, USA kris.hauser@duke.edu

linear quadratic regulator (iLQR) is a similar backwards recursion approach.

State- and control-limited optimal control problems have been solved in the linear quadratic case using multi-parametric programming to derive optimal gain coefficients for every initial point in state space [4], but the complexity of the piecewise-linear policy can grow sharply with dimension. Nonlinear constrained problems can be solved using collocation methods, which formulate a large nonlinear program across the entire trajectory and optimize using numerical methods like sequential quadratic programming [1]. However, these methods are expensive due to their need to formulate a large optimization problem over all control variables across the trajectory.

Several authors have incorporated constraints into DDP methods. Most similar to our work, Murray and Yakovitz (1979) [5] and Yakovitz (1986) [6] derive a constrained nonlinear DDP method for linearly constrained controls using a stagewise Karush-Kuhn-Tucker condition. However, these methods are only applicable to problems with linear control, and lack the ability to handle infeasible QPs in the forward pass. Shi et al [7] maintain iterates an admissible region, but work only with linear systems and constraints. Tassa et al develop a control-limited DDP method and apply it to high-dimensional humanoid characters [8], but only support box-bounded control constraints. Our method, by contrast, supports arbitrary nonlinear state and cost constraints.

Constraint penalties have also been used to address trajectory optimization with constraints. Manchester and Kuindersma (2016) mention the use of exponential barrier functions to account for constraints [9]. Chang et al [10] use a relaxation technique to handle constraints, in which constraint violations are penalized to produce an unconstrained objective function. Unlike our method, the iterates and final result are not guaranteed to be feasible.

III. Preliminaries

For completeness, we first formally define the optimal control problem and provide a short review on DDP to define mathematical notation used throughout this paper. Please refer to [2] for more detailed description of DDP.

A. Optimal Control Problem

Consider the discrete-time dynamical system,

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k), \tag{1}$$

where $\mathbf{x}_k \in \mathbb{R}^n$ is the state of the system at time step k, $\mathbf{u}_k \in \mathbb{R}^m$ is the control input at time step k, and f is the dynamic function that governs the state transition given the current state and control.

The cost of a sequence of states, $\mathbf{X} = \{\mathbf{x}_0, \cdots, \mathbf{x}_N\}$, and a sequence of control, $\mathbf{U} = \{\mathbf{u}_0, \cdots, \mathbf{u}_{N-1}\}$, is defined by the objective function,

$$J(\mathbf{X}, \mathbf{U}) = \sum_{k=0}^{N-1} l(\mathbf{x}_k, \mathbf{u}_k) + l^f(\mathbf{x}_N), \tag{2}$$

where $l(\mathbf{x}, \mathbf{u}) : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ is the running cost and $l^f : \mathbb{R}^n \to \mathbb{R}$ is the final cost.

Given the initial state $\mathbf{x}_0$ and a time horizon N, our goal is to find a sequence of control $\mathbf{U}$ that minimizes J subject to dynamic constraints (1). This optimization problem can be solved by dynamic programming because the optimality of future control from a particular state does not depend on the past control or state sequences. Therefore, we define an optimal value function at time step k as the optimal cost-to-go starting at a given state $\mathbf{x}$:

$$V_k(\mathbf{x}) = \min_{\mathbf{U}} \sum_{j=k}^{N-1} l(\mathbf{x}_j, \mathbf{u}_j) + l^f(\mathbf{x}_N).$$

By the recursive nature of Bellman Optimality Principle, the optimal value function can be rewritten as

$$V_k(\mathbf{x}) = \min_{\mathbf{u}} l(\mathbf{x}, \mathbf{u}) + V_{k+1}(f(\mathbf{x}, \mathbf{u})), \tag{3}$$

where the boundary condition is

$$V_N(\mathbf{x}) = l^f(\mathbf{x}).$$

B. Differential Dynamic Programming

DDP is an iterative method to numerically solve a nonlinear optimal control problem as described above. At each iteration, DDP performs a backward pass and a forward pass on the current estimate of state-control trajectory ($\mathbf{X}$, $\mathbf{U}$), i.e. nominal trajectory. In the backward pass, the algorithm approximates the value function as a quadratic function alone the nominal trajectory. In the forward pass, a forward integration is performed to produce a new nominal trajectory based on the value function computed in the backward pass. This process is repeated until a desired level of convergence.

a) Backward pass.: We first define an action-value function, $Q : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$, which evaluates the cost-to-go of taking a given action in a given state following the optimal policy thereafter. More conveniently, we define Q in terms of deviation from the nominal state and action $(\mathbf{x}_k, \mathbf{u}_k)$:

$$\begin{aligned} Q_k(\boldsymbol{\delta}_x, \boldsymbol{\delta}_u) = l(&\mathbf{x}_k + \boldsymbol{\delta}_x, \mathbf{u}_k + \boldsymbol{\delta}_u) \\ &+ V_{k+1}(f(\mathbf{x}_k + \boldsymbol{\delta}_x, \mathbf{u}_k + \boldsymbol{\delta}_u)). \end{aligned} \tag{4}$$

To approximate the value function as a quadratic function, we Taylor-expand Q_k about $(\mathbf{0}, \mathbf{0})$ and truncate the terms beyond second-order:

$$\begin{aligned} Q_k(\boldsymbol{\delta}_x, \boldsymbol{\delta}_u) = Q_k(\mathbf{0}, \mathbf{0}) &+ Q_{x,k}^T \boldsymbol{\delta}_x + Q_{u,k}^T \boldsymbol{\delta}_u \\ &+ \frac{1}{2}(\boldsymbol{\delta}_x^T Q_{xx,k} \boldsymbol{\delta}_x + \boldsymbol{\delta}_u^T Q_{uu,k} \boldsymbol{\delta}_u) \\ &+ \boldsymbol{\delta}_u^T Q_{ux,k} \boldsymbol{\delta}_x, \end{aligned} \tag{5}$$

where

$$\begin{aligned} Q_k(\mathbf{0}, \mathbf{0}) &= l(\mathbf{x}_k, \mathbf{u}_k) + V_{k+1}(\mathbf{x}_{k+1}), \\ Q_{x,k} &= l_{x,k} + f_x^T b_{k+1}, \\ Q_{u,k} &= l_{u,k} + f_u^T b_{k+1}, \\ Q_{xx,k} &= l_{xx,k} + f_x^T A_{k+1} f_x + b_{k+1}^T f_{xx}, \\ Q_{uu,k} &= l_{uu,k} + f_u^T A_{k+1} f_u + b_{k+1}^T f_{uu}, \\ Q_{ux,k} &= l_{ux,k} + f_u^T A_{k+1} f_x + b_{k+1}^T f_{ux}. \end{aligned}$$

Note that other than the time index, k, the subscript indicates the variable with respect to which the derivative is taken. To simplify the notations, we define A_{k+1} and b_{k+1} to be the Hessian and the gradient of $V_{k+1}(\mathbf{x})$ evaluated at $\mathbf{x}_{k+1}$.

We can then express the value function by optimizing the quadratic approximation of Q over $\boldsymbol{\delta}_u$:

$$V_k(\mathbf{x}) = \min_{\boldsymbol{\delta}_u} Q_k(\boldsymbol{\delta}_x, \boldsymbol{\delta}_u). \tag{6}$$

The solution to this optimization is a linear feedback relation between $\boldsymbol{\delta}_u$ and $\boldsymbol{\delta}_x$:

$$\boldsymbol{\delta}_u = K_k \boldsymbol{\delta}_x + j_k, \tag{7}$$

where, $K_k = -Q_{uu,k}^{-1} Q_{ux,k}$ and $j_k = -Q_{uu,k}^{-1} Q_{u,k}$.

Finally, we plug the optimal control (7) into the approximated Q function (5) to recover the value function:

$$V_k(\mathbf{x}) = \frac{1}{2}(\mathbf{x} - \mathbf{x}_k)^T A_k (\mathbf{x} - \mathbf{x}_k) + b_k^T (\mathbf{x} - \mathbf{x}_k) + c_k,$$

where

$$\begin{aligned} A_k &= Q_{xx} + K_k^T Q_{uu} K_k + Q_{ux}^T K_k + K_k^T Q_{ux}, \\ b_k &= Q_x + K_k^T Q_{uu} j_k + Q_{ux}^T j_k + K_k^T Q_u, \end{aligned} \tag{8}$$

and c_k is the constant term irrelevant to the algorithm. The Hessian (A_k) and the gradient (b_k) of the value function evaluated at $\mathbf{x}_k$ are then passed to the previous time step $k-1$ as the backward pass continues.

Starting with $A_N = l_{xx}^f$ and $b_N = l_x^f$, we can recursively solve all the K_k, j_k, A_k, and b_k from time step N to 0.

b) Forward pass.: During the forward pass, we update the nominal state-control trajectory using the optimal linear feedback (7) from the previous backward pass, starting with $\mathbf{x}_0^{new} = \mathbf{x}_0$:

$$\mathbf{u}_k^{new} = \mathbf{u}_k + K_k(\mathbf{x}_k^{new} - \mathbf{x}_k) + j_k \tag{9}$$

$$\mathbf{x}_{k+1}^{new} = f(\mathbf{x}_k^{new}, \mathbf{u}_k^{new}). \tag{10}$$

IV. Constrained DDP

Our CDDP method extends standard DDP to enforce general inequality constraints, which are expressed as differentiable vector functions of state or/and control variables:

$$g_k(\mathbf{x}, \mathbf{u}) \leq \mathbf{0}. \tag{11}$$

The constraint g_k are enforced on the state and control on k'th time step. Although we consider inequalities here the method could also easily be applied to equality constraints.

When a constraint is active, the quadratic approximation of the value function used in standard DDP is no longer accurate, because when the control at step $k-1$ is changed infinitesimally, the optimal state and/or control at time k still remain at the boundary of the feasible set. To properly handle this situation, the DDP backwards recursion should be modified with the knowledge that active constraints remain active under local perturbation. Using techniques from sensitivity analysis, this section derives the quadratic approximation of the value function in the presence of active constraints.

CDDP begins with a suboptimal feasible trajectory, and iteratively reduces the cost of the trajectory while keeping it feasible in the face of various approximations of the problem. The algorithm's backward pass calculates the constrained value function approximation to determine a step direction, and on the forward pass ensures the feasibility of the new nominal trajectory and decreasing cost using a trust region method. The algorithm must also discover when constraints should be added and removed from the active set.

A. Backward Pass

Algorithm 1 describes the procedures for our backward pass. Similar to the standard DDP, the backward pass quadratically approximates the value function about the nominal trajectory through a recursive process from time step N to time step 0. However, instead of the unconstrained optimization in (6), we should express the value function in terms of a constrained optimization:

$$\begin{aligned} \min_{\boldsymbol{\delta}_u} \; & \frac{1}{2}\boldsymbol{\delta}_u^T Q_{uu,k} \boldsymbol{\delta}_u + \boldsymbol{\delta}_u^T Q_{ux,k} \boldsymbol{\delta}_x + Q_{u,k}^T \boldsymbol{\delta}_u \\ \text{subject to} \quad & g_k(\mathbf{x}_k + \boldsymbol{\delta}_x, \mathbf{u}_k + \boldsymbol{\delta}_u) \leq \mathbf{0}. \end{aligned} \tag{12}$$

Unfortunately, the relationship between $\boldsymbol{\delta}_x$ and the optimal $\boldsymbol{\delta}_u$ can no longer be derived by a simple matrix inversion as in (7). Here we derive a local analytical, quadratic expression of the optimal solution to (12) as a function of $\boldsymbol{\delta}_x$ using sensitivity analysis. Near the nominal trajectory, the analytical approximation yields a good approximation of the value function under the assumption that the currently active constraints do not change.

To this end, we first create an active set that includes all the equality constraints and the inequality constraints whose boundary the current nominal trajectory lies on. Due to numerical issues, we select constraints that meet equality with 0 up to some tolerance $g_k(\mathbf{x}_k, \mathbf{u}_k) \geq -\epsilon$ (line 8). Let $\hat{g}_k$ denote the subset of active constraints. The linearization of the active set constraints indicates that the constraint:

$$C_k \boldsymbol{\delta}_u = D_k \boldsymbol{\delta}_x \tag{13}$$

must be met, where $C_k = \hat{g}_{u,k}(\mathbf{x}_k, \mathbf{u}_k)$ and $D_k = -\hat{g}_{x,k}(\mathbf{x}_k, \mathbf{u}_k)$ (line 9-10) are the derivatives of the active constraints in the $\mathbf{u}$ and $\mathbf{x}$ dimensions, respectively.

In a local neighborhood around the nominal trajectory, the solutions to the constrained optimization are linearly approximated by the equality-constrained optimization:

$$\begin{aligned} \min_{\boldsymbol{\delta}_u} \; & \frac{1}{2}\boldsymbol{\delta}_u^T Q_{uu,k} \boldsymbol{\delta}_u + \boldsymbol{\delta}_u^T Q_{ux,k} \boldsymbol{\delta}_x + Q_{u,k}^T \boldsymbol{\delta}_u \\ & \text{subject to} \quad C_k \boldsymbol{\delta}_u = D_k \boldsymbol{\delta}_x, \end{aligned} \tag{14}$$

where the solution can be expressed analytically through KKT conditions,

$$\begin{bmatrix} Q_{uu,k} & C_k^T \\ C_k & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\delta}_u \\ \boldsymbol{\lambda} \end{bmatrix} = - \begin{bmatrix} Q_{ux,k} \\ D_k \end{bmatrix} \boldsymbol{\delta}_x - \begin{bmatrix} Q_{u,k} \\ 0 \end{bmatrix}. \tag{15}$$

The equation can be used as-is using the Schur complement to determine $\boldsymbol{\delta}_u$ as a function of $\boldsymbol{\delta}_x$. However, we have found that this often fails to release constraints from the

Algorithm 1 CDDP Backward Pass

1: $A_N \leftarrow l_{xx}^f, b_N \leftarrow l_x^f$
2: **for** $k = N-1, N-2, \ldots, 0$ **do**
3: $\quad Q_x \leftarrow l_{x,k} + f_x^T b_{k+1}$
4: $\quad Q_u \leftarrow l_{u,k} + f_u^T b_{k+1}$
5: $\quad Q_{xx} \leftarrow l_{xx,k} + f_x^T (A_{k+1} + \mu_1 I_n) f_x + b_{k+1}^T f_{xx}$
6: $\quad Q_{uu} \leftarrow l_{uu,k} + f_u^T (A_{k+1} + \mu_1 I_n) f_u + b_{k+1}^T f_{uu} + \mu_2 I_m$
7: $\quad Q_{ux} \leftarrow l_{ux,k} + f_u^T (A_{k+1} + \mu_1 I_n) f_x + b_{k+1}^T f_{ux}$
8: $\quad \hat{g}_k \leftarrow$ all entries of g_k such that $g_k(\mathbf{x}_k, \mathbf{u}_k) \leq \epsilon$
9: $\quad C_k \leftarrow \hat{g}_{u,k}(\mathbf{x}_k, \mathbf{u}_k)$
10: $\quad D_k \leftarrow -\hat{g}_{x,k}(\mathbf{x}_k, \mathbf{u}_k)$
11: $\quad$ Solve $\frac{1}{2}\boldsymbol{\delta}_u^T Q_{uu} \boldsymbol{\delta}_u + \boldsymbol{\delta}_u^T Q_u$, subject to $C_k \boldsymbol{\delta}_u = 0$
12: $\quad$ Compute Lagrange multipliers λ
13: $\quad \hat{C}_k$, $\hat{D}_k \leftarrow$ rows from C_k, D_k corresponding to positive entries of λ
14: $\quad W \leftarrow (\hat{C}_k Q_{uu}^{-1} \hat{C}_k^T)^{-1} \hat{C}_k Q_{uu}^{-1}$
15: $\quad H \leftarrow Q_{uu}^{-1}(I - \hat{C}_k^T W)$
16: $\quad K_k \leftarrow -H Q_{ux} + (W)^T \hat{D}_k$
17: $\quad j_k \leftarrow -H Q_u$
18: $\quad A_k \leftarrow Q_{xx} + K_k^T Q_{uu} K_k + Q_{ux}^T K_k + K_k^T Q_{ux}$
19: $\quad b_k \leftarrow Q_x + K_k^T Q_{uu} j_k + Q_{ux}^T j_k + K_k^T Q_u$
20: $\quad$ Store Q_{uu}, Q_{ux}, Q_u as $Q_{uu,k}, Q_{ux,k}, Q_{u,k}$
21: **end for**

active set. Instead, we examine the dual solution (Lagrangian multipliers), $\boldsymbol{\lambda}$, to indicate which constraints remain active. We solve the equation once to determine the dual solution about the nominal trajectory, i.e., when $\boldsymbol{\delta}_x = \mathbf{0}$. Removing the constraints associated with the negative elements of $\boldsymbol{\lambda}$, we obtain a new active set: $\hat{C}_k \boldsymbol{\delta}_u = \hat{D}_k \boldsymbol{\delta}_x$ (line 11-13).

Now we can express the primal solution subject to the modified active set of constraints:

$$\boldsymbol{\delta}_u = K_k \boldsymbol{\delta}_x + j_k. \tag{16}$$

K_k and j_k are the feedback gain and the open-loop term defined as (line 14-17)

$$K_k = -H Q_{ux,k} + (W)^T \hat{D}_k, \tag{17}$$

$$j_k = -H Q_{u,k}, \tag{18}$$

where

$$W = (\hat{C}_k Q_{uu,k}^{-1} \hat{C}_k^T)^{-1} \hat{C}_k Q_{uu,k}^{-1},$$
$$H = Q_{uu,k}^{-1}(I - \hat{C}_k^T W).$$

After computing K_k and j_k, we can update the Hessian and the gradient of the approximated value function using (8) (line 18-19).

B. Forward Pass

Algorithm 2 lists pseudocode for the forward pass. During the forward pass, we ensure that the updated nominal trajectory remains feasible and continues to reduce the objective function. Although the approximated value function from the backward pass takes into account the estimated active constraints, it cannot guarantee that directly using the linear

Algorithm 2 CDDP Forward Pass

1: $feasible \leftarrow$ **False**
2: $J_{int} = J(\mathbf{X}, \mathbf{U})$
3: $\mathbf{e} \leftarrow$ BIG $\quad \triangleright$ set to initial large trust bound
4: **while** $feasible =$ **False do**
5: $\quad feasible \leftarrow$ **True**
6: $\quad \mathbf{x} \leftarrow \mathbf{x_0}$
7: $\quad \mathbf{X_{temp}} \leftarrow \mathbf{X}$, $\mathbf{U_{temp}} \leftarrow \mathbf{U}$
8: $\quad$ **for** $k = 0, 1, \ldots, N-1$ **do**
9: $\quad\quad \boldsymbol{\delta}_x \leftarrow \mathbf{x} - \mathbf{x}_k$
10: $\quad\quad \mathbf{x}_{temp,k} \leftarrow \mathbf{x}$
11: $\quad\quad$ Solve QP: $\boldsymbol{\delta}_u^* = \arg\min \frac{1}{2}\boldsymbol{\delta}_u^T Q_{uu,k} \boldsymbol{\delta}_u + \boldsymbol{\delta}_u^T Q_{ux,k} \boldsymbol{\delta}_x + \boldsymbol{\delta}_u^T Q_{u,k}$, subject to $|\boldsymbol{\delta}_u| \leq \mathbf{e}$ and $g_k(\mathbf{x}, \mathbf{u}_k) + g_{u,k} \boldsymbol{\delta}_u \leq 0$
12: $\quad\quad$ **if** QP is infeasible **then**
13: $\quad\quad\quad feasible \leftarrow$ **False**
14: $\quad\quad\quad \mathbf{e} \leftarrow \alpha \mathbf{e}$
15: $\quad\quad\quad$ **break**
16: $\quad\quad$ **end if**
17: $\quad\quad \mathbf{u}_{temp,k} \leftarrow \mathbf{u}_k + \boldsymbol{\delta}_u^*$
18: $\quad\quad \mathbf{x} \leftarrow f(\mathbf{x}, \mathbf{u}_{temp,k})$
19: $\quad$ **end for**
20: $\quad \mathbf{x}_{temp,N} \leftarrow \mathbf{x}$
21: $\quad J_{temp} \leftarrow J(\mathbf{X_{temp}}, \mathbf{U_{temp}})$
22: **end while**
23: **if** $J_{temp} < J_{int}$ **then**
24: $\quad \mathbf{X} \leftarrow \mathbf{X_{temp}}$, $\mathbf{U} \leftarrow \mathbf{U_{temp}}$
25: $\quad \mu_1 \leftarrow \beta_1 \mu_1$, $\mu_2 \leftarrow \beta_2 \mu_2 \quad \triangleright\ 0 < \beta_1, \beta_2 < 1$
26: **else**
27: $\quad \mu_1 \leftarrow \alpha_1 \mu_1$, $\mu_2 \leftarrow \alpha_2 \mu_2 \quad \triangleright\ \alpha_1, \alpha_2 > 1$
28: **end if**

feedback to update the nominal trajectory will result in a feasible trajectory. Consequently, we fully solve a QP (line 12-16) that takes into account all the constraints to guarantee the updated nominal trajectory is feasible:

$$\min_{\boldsymbol{\delta}_u} \frac{1}{2}\boldsymbol{\delta}_u^T Q_{uu,k} \boldsymbol{\delta}_u + \boldsymbol{\delta}_u^T Q_{ux,k} \boldsymbol{\delta}_x + Q_{u,k}^T \boldsymbol{\delta}_u$$
$$\text{subject to } g_k(\mathbf{x}_k, \mathbf{u}_k) + g_{x,k}(\mathbf{x}_k, \mathbf{u}_k) \boldsymbol{\delta}_x + g_{u,k}(\mathbf{x}_k, \mathbf{u}_k) \boldsymbol{\delta}_u \leq \mathbf{0}. \tag{19}$$

Note that $\boldsymbol{\delta}_x$ term in the linearzied constraint vanishes because the constraint is linearize about the newly updated nominal state:

$$\mathbf{u}_{k-1} = \mathbf{u}_{k-1} + \boldsymbol{\delta}_u^* \tag{20}$$

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}) \tag{21}$$

where $\boldsymbol{\delta}_u^*$ is the optimal solution of the QP from the previous time step.

C. Regularization

In practice, the convergence of the optimization problem highly depends on the choice of step size, since step sizes that are too large may lead to infeasibility or increases in the

objective. We adapt the regularization scheme proposed by [11]:

$$Q_{xx,k} = l_{xx,k} + f_x^T(A_{k+1} + \mu_1 I_n) f_x + b_{k+1}^T f_{xx} \quad (22)$$

$$Q_{uu,k} = l_{uu,k} + f_u^T(A_{k+1} + \mu_1 I_n) f_u + b_{k+1}^T f_{uu} + \mu_2 I_m \quad (23)$$

$$Q_{ux,k} = l_{ux,k} + f_u^T(A_{k+1} + \mu_1 I_n) f_x + b_{k+1}^T f_{ux}, \quad (24)$$

where the μ_1 term keeps the new state trajectory close to the old one while the μ_2 term regularizes the control trajectory.

We adjust the value of μ_1 and μ_2 after each forward pass. If the trajectory is improved, the values are reduced by the factor of β_1 and β_2 respectively (Algorithm 2 line 25), where $0 < \beta_1, \beta_2 < 1$. Otherwise, we increase the weight of regularization terms by the factor of α_1 and α_2 (Algorithm 2 line 27), where $\alpha_1, \alpha_2 > 1$.

In addition, we enforce a trust-region to limit the size of $\boldsymbol{\delta}_u$ such that the new nominal trajectory stays close to the feasible region. We add a box constraint on $\boldsymbol{\delta}_u$ during the QP solve (19):

$$-\mathbf{e} \leq \delta_u \leq \mathbf{e}, \quad (25)$$

where $\mathbf{e}$ is an adaptive bound to limit the distance between $\boldsymbol{\delta}_u$ and the current nominal trajectory. Initially, $\mathbf{e}$ is a vector of a large number, which is reduced when an attempt to solve the QP results in infeasible solution (Algorithm 2 line 12-15):

$$\mathbf{e} \leftarrow \alpha \mathbf{e} \quad (26)$$

where $0 < \alpha < 1$.

V. RESULT

We evaluated CDDP on three different dynamic systems: a 2D point mass, a 2D car, and a 3D quadcopter. Each dynamic system was tested with a nonlinear geometric constraint and a more complex scenario with multiple or/and moving constraints. CDDP was compared against two alternative methods. The first one replaces hard constraints with a log-barrier penalty term in the objective function and the second one uses sequential quadratic programming implemented by SNOPT software. For all examples, the regularization parameters are $\beta_1 = \beta_2 = 0.95$ and $\alpha_1 = \alpha_2 = 1.05$, and the trust region reduction rate is $\alpha = 0.5$. We run all the tests on a laptop computer with a 2.7 GHz Intel Core i5 processor and 8GB of RAM.

For all three methods, the optimization terminates when the same computation time budget is reached. We compare the total costs of the trajectory at the termination time shown in Table I. The computation time budget is set to 5 seconds for all examples.

A. 2D Point Mass

We begin with a simple 2D point mass system. The state includes the position and the velocity of the point mass, $\mathbf{x} = \{x, y, v^x, v^y\}$, while the control directly commands the acceleration of the point mass, $\mathbf{u} = \{a^x, a^y\}$. The dynamics of the system can be expressed by following linear equations:

$$\begin{aligned} x_{k+1} &= x_k + h v_k^x, \\ y_{k+1} &= y_k + h v_k^y, \\ v_{k+1}^x &= v_k^x + h a_k^x, \\ v_{k+1}^y &= v_k^y + h a_k^y. \end{aligned}$$

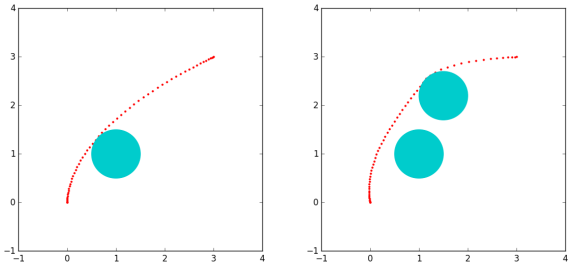

Fig. 2: Solutions to the point mass examples computed by CDDP.

The optimal trajectory of the point mass minimizes the control effort and the deviation from the goal state:

$$\begin{aligned} l(\mathbf{x}, \mathbf{u}) &= h \mathbf{u}^T R \mathbf{u}, \\ l^f(\mathbf{x}) &= (\mathbf{x} - \mathbf{x}^{goal})^T Q^f (\mathbf{x} - \mathbf{x}^{goal}), \end{aligned} \quad (27)$$

where R is an identity matrix giving the same weights to the cost of a_x and a_y. Q^f is a diagonal matrix with diagonal elements being 50, 50, 10, and 10.

We set the initial and the goal state to be $\mathbf{x}_0 = \begin{bmatrix} 0, 0, 0, 0 \end{bmatrix}$ and $\mathbf{x}^{goal} = \begin{bmatrix} 3, 3, 0, 0 \end{bmatrix}$. The initial nominal trajectory starts from $\mathbf{x}_0$ and ends at $\begin{bmatrix} 0, 3, 0, 0 \end{bmatrix}$. We use a time step of $h = 0.05$ with time horizon $N = 300$.

In the first experiment, we place a circular constraint centered at $\begin{bmatrix} 1, 1 \end{bmatrix}$ with radius 0.5:

$$(x-1)^2 + (y-1)^2 \geq 0.25.$$

Our method successfully generates a trajectory shown in Fig. 2 Left. Adding another circular constraint (centered at $\begin{bmatrix} 1.5, 2.2 \end{bmatrix}$ with radius 0.5) creates a more constrained problem, results in a different trajectory (Fig. 2 Right).

B. 2D Car

Consider a 2D circle following the simplified vehicle dynamics:

$$\begin{aligned} x_{k+1} &= x_k + h v_k \sin(\theta_k), \\ y_{k+1} &= y_k + h v_k \cos(\theta_k), \\ \theta_{k+1} &= \theta_k + h u^\theta v_k, \\ v_{k+1} &= v_k + h u^v, \end{aligned}$$

where the state, $\mathbf{x} = \{x, y, \theta, v\}$, includes the 2D position, the orientation, and the forward velocity of the vehicle. The control variable u^θ changes the steering angle and u^v changes the forward velocity.

Similar to the point mass example, the optimal trajectory of the 2D vehicle minimizes the control effort and the

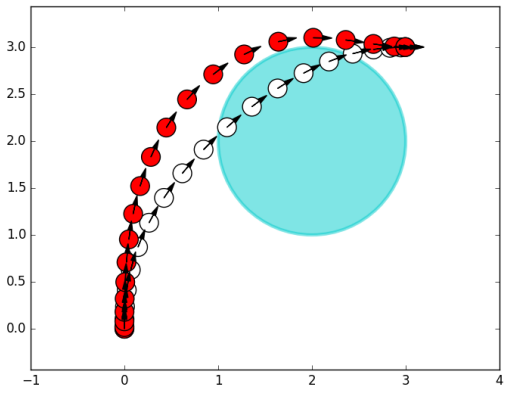

Fig. 3: The optimal trajectory of a 2D car (red circles) driving from $(0, 0, 0, 0)$ to $(3, 3, \frac{\pi}{2}, 0)$ while avoiding a circular obstacle centered at (2, 2). The black arrow indicates the heading direction of the car. For comparison, the white circles show the optimal trajectory without the circular constraint.

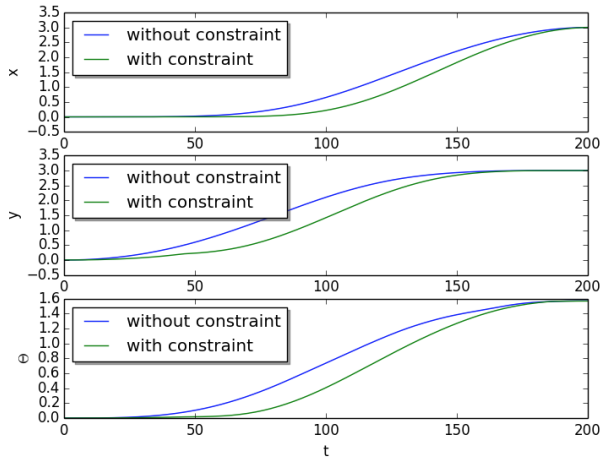

Fig. 4: The trajectory of position and orientation of the 2D car without constraints (green trajectories) and with the moving circular constraint (blue trajectories).

deviation from the goal state (27). The only difference is that the diagonal elements of R are 0.2 and 0.1 while those for Q^f are $50, 50, 50, 10$.

We set the initial state and the goal state to be $\mathbf{x}_0 = \left[0, 0, 0, 0\right]$ and $\mathbf{x}^{goal} = \left[3, 3, \frac{\pi}{2}, 0\right]$. The initial nominal trajectory starts from $\mathbf{x}_0$ and ends at $\left[2, 4, \frac{\pi}{2}, 0\right]$. We use a time step of $h = 0.05$ with a time horizon $N = 100$.

In addition to the circular constraint (centered at $\left[2, 2\right]$ with radius 1), we also set a constraint on the control variable: $u^\theta \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$. Our method successfully produces a trajectory shown in Fig. 3.

To make the problem more interesting, we let the circular obstacle move over time. Consider a circle of radius 1 with initial center at $\left[-1, 1.2\right]$ moving horizontally to the right at 0.5 per time unit. The initial state $\mathbf{x}_0$ and the goal state $\mathbf{x}^{goal}$ are the same as before, but we increase the time horizon to $N = 200$. Our method produces a solution shown in Fig. 4. To avoid the collision with the moving circle, the vehicle waits for awhile at the beginning and starts to accelerate. Fig. 5 visualizes a few frames of the vehicle motion.

C. 3D Quadcopter

We tested out algorithm on a 3D quadcopter, an under-actuated nonlinear dynamic system with following equations of motion [12]:

$$\begin{aligned}
\mathbf{x}_{k+1} &= \mathbf{x}_k + h\mathbf{v}_k \\
\mathbf{v}_{k+1} &= \mathbf{v}_k + h(\mathbf{g} + \frac{1}{m}(R_\theta \mathbf{f} - k_d \mathbf{v}_k)) \\
\boldsymbol{\theta}_{k+1} &= \boldsymbol{\theta}_k + hJ_\omega^{-1}\boldsymbol{\omega}_k \\
\boldsymbol{\omega}_{k+1} &= \boldsymbol{\omega}_k + hI_c^{-1}\boldsymbol{\tau}
\end{aligned}$$

where $\mathbf{x}$ and $\mathbf{v}$ are 3-vectors representing the position and the velocity of the quadcopter in the inertia frame. $\boldsymbol{\theta}$ is the 3-vector consists of the roll, pitch and yaw angles define in the body frame. $\boldsymbol{\omega}$ is the angular velocity in the body frame. R_θ is a rotation matrix that transforms a vector from the body frame to the inertia frame while J_ω is the Jacobian matrix that transforms the time derivative of $\boldsymbol{\theta}$ to angular velocity: $\boldsymbol{\omega} = J_\omega \dot{\boldsymbol{\theta}}$. k_d is the friction coefficient. I_c is the inertia matrix.

The quadcopter is actuated by four motors generating thrust force along the z-axis of the body frame. The angular velocity of each motor, u_1, u_2, u_3, u_4, can be adjusted independently. The total thrust force and torque on the quadcopter in the body frame is given by

$$\begin{aligned}
\mathbf{f} &= \left[0, 0, u_1^2 + u_2^2 + u_3^2 + u_4^2\right] \\
\boldsymbol{\tau} &= \left[u_1^2 - u_3^2, u_2^2 - u_4^2, u_1^2 - u_2^2 + u_3^2 - u_4^2\right].
\end{aligned}$$

Therefore, we define the control variables as $\mathbf{u} = \left[u_1^2, u_2^2, u_3^2, u_4^2\right]$ and enforce lower bounds on the control vector: $\mathbf{u} \geq \mathbf{0}$.

We use the same cost function as (27) with R being the identity matrix and the diagonal elements of Q^f being $50, 50, 50, 2, 2, 2, 1, 1, 1, 1, 1, 1$.

We first test the algorithm on a scene where the quadcopter has to fly around a sphere obstacle centered at the origin with radius 2. The initial and the goal location of quadcopter are $\mathbf{x}_0 = [-3.5, 0, 0]$ and $\mathbf{x}^{goal} = [2.8, 0, 0]$ respectively. Starting with a linear trajectory from $\mathbf{x}_0$ to $[-0.5, 3, 0]$ with time step $h = 0.02$ and time horizon $N = 200$. Fig. 6 illustrates a few samples along the final trajectory.

To demonstrate a more challenging scenario, we show that the quadcopter is able to reach the goal state while avoiding two moving spheres in the 3D space. The first sphere (radius 1) moves along the negative z-axis from $[-1.5, 0, 2]$ at the speed of 1 unit per second while the second one (radius 1) moving along the y-axis from $[1, -2.5, 0]$ at the speed of 1 unit per second. Starting with hovering controller at $\mathbf{x_0}$, with $\mathbf{x}^{goal} = [1, 0, 0]$, Fig. 1 visualizes a few frames of the quadcopter motion.

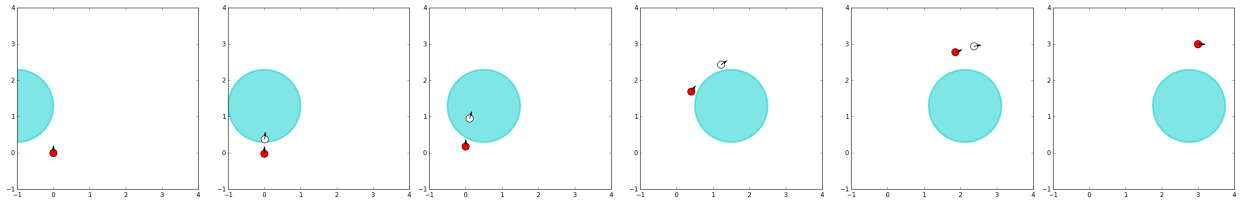

Fig. 5: Frames from optimal trajectory of a 2D car, shown as the red circle, driving from $(0, 0, 0, 0)$ to $(3, 3, \frac{\pi}{2}, 0)$ while avoiding a moving obstacle, shown as the cyan circle. The white circle shows the optimal trajectory without the moving constraint.

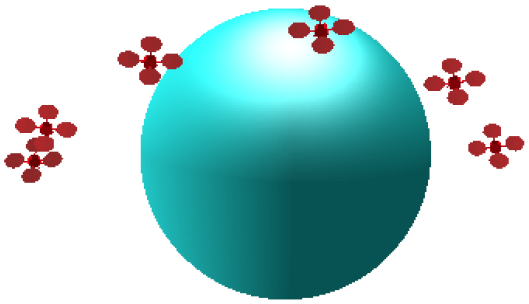

Fig. 6: The trajectory of a quadcopter avoiding a sphere.

D. Comparison with barrier methods and SQP

One common practice to handle inequality constraints in an optimization is to reformulate the constraints as log-barrier functions and include them in the objective function[13]. We compare our method against the log-barrier formulation of the optimization:

$$\min_{\mathbf{U}} \sum_{k=1}^{N-1} l(\mathbf{x}_k, \mathbf{u}_k) + l^f(\mathbf{x}_N) - \sum_{i=0}^{M} t \log(-g_i(\mathbf{x}, \mathbf{u})).$$

Note that as t approaches 0, $-t\log(-g)$ also approaches 0 if the constraint is satisfied ($g < 0$). Otherwise, $-t\log(-g)$ approaches ∞. Starting with a large t, the method gradually decreases t to improve the approximation of the original constrained problem. In each iteration of t, an unconstrained optimization is solved using Newton's method. To use log-barrier method for constraints in DDP, we add the additional log-barrier cost term at each time step during the backward pass and forward pass.

Another common practice to solve optimal control problem is to use sequential quadratic programming (SQP), which makes a local quadratic approximation of the objective function and local linear approximations of the constraints and solves a quadratic program on each iteration[14]. SNOPT[15] is a SQP solver that is heavily used in the control community[16].

We compare our method with log-barrier method and SNOPT in all the examples with a 5 s time budget. We also do experiments on the same examples with different discretization of the time step. The summary of the results is in Table I. CDDP and SNOPT outperform log-barrier methods in all cases except in the 2D car with fixed circle and $N = 100$. And even though SNOPT wins at examples with shorter horizon ($N = 100, 200$), CDDP outperforms SNOPT in longer horizon ($N >= 300$) except in the car with fixed circle and $N = 500$ and quadcopter with moving spheres and $N = 400$. Fig. 7 shows how the cost changes over time for the quadcopter example with fixed sphere constraints, with time horizon $N = 400$.

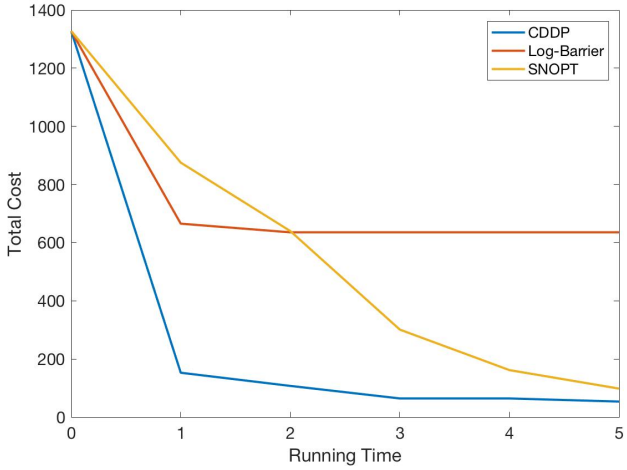

Fig. 7: Comparison between CDDP, log-barrier DDP and SNOPT on the quadcopter with fixed sphere constraint example, with time horizon $N = 400$.

VI. CONCLUSIONS

We presented an extension of DDP to problems with nonlinear state and control constraints. Experiments demonstrate that our method converges in fewer iterations, and can solve more complex dynamic problems than a log-barrier constraint penalty approach.

Our implementation is somewhat slower than DDP due to the need to solve quadratic programs rather than matrix inversion in inner steps, and this could be improved in future work. Also, like all nonconvex optimization, we need a good initial trajectory to ensure that the algorithm would not get stuck in a bad local minima. In future work we plan to use a sampling-based planning algorithm like RRT to find a good initial trajectory [17], and our method could be incorporated

Examples	CDDP	log-barrier	$SNOPT$
Point Mass one circle (h=0.05,N=300)	0.065	0.042	0.073
Point Mass two circles (h=0.05,N=300)	0.28	0.64	0.43
Car fixed circle (h=0.05,N=100)	0.33	0.28	0.28
Car moving circle (h=0.05,N=200)	0.10	0.32	0.08
Quadcopter fixed sphere (h=0.02,N=200)	62.07	478.00	81.79
Quadcopter moving spheres (h=0.02,N=200)	49.36	104.42	48.13
Point Mass one circle (h=0.03,N=500)	0.071	18.30	0.32
Point Mass two circles (h=0.03,N=500)	0.27	1.41	0.42
Car fixed circle (h=0.01,N=500)	0.49	9.86	0.30
Car moving circle (h=0.02,N=500)	0.21	4.80	86.00
Quadcopter fixed sphere (h=0.01,N=400)	53.69	635.53	98.28
Quadcopter moving spheres (h=0.01,N=400)	52.45	182.01	49.12

TABLE I: Comparison of different methods with a 5 s time budget.

into a kinodynamic planner to help it converge more quickly to a global optimum [18].

REFERENCES

[1] C. R. Hargraves and S. W. Paris, "Direct trajectory optimization using nonlinear programming and collocation," *J. Guidance, Control, and Dynamics*, vol. 10, no. 4, pp. 338–342, 1987.

[2] D. Mayne, "A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems," *International Journal of Control*, vol. 3, no. 1, pp. 85–95, 1966. [Online]. Available: http://dx.doi.org/10.1080/00207176608921369

[3] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems," in *1st Intl Conf. Informatics in Control, Automation and Robotics*, 2004.

[4] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3 – 20, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0005109801001741

[5] D. M. Murray and S. J. Yakowitz, "Constrained differential dynamic programming and its application to multireservoir control," *Water Resources Research*, vol. 15, no. 5, pp. 1017–1027, 1979.

[6] S. Yakowitz, "The stagewise kuhn-tucker condition and differential dynamic programming," *IEEE transactions on automatic control*, vol. 31, no. 1, pp. 25–30, 1986.

[7] J. Shi, P. B. Luh, S.-C. Chang, and T.-S. Chang, "A method for constrained dynamic optimization problems," in *American Control Conference, 1990*. IEEE, 1990, pp. 830–835.

[8] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 1168–1175.

[9] Z. Manchester and S. Kuindersma, "Derivative-free trajectory optimization with unscented dynamic programming," in *the Proceedings of the 55th Conference on Decision and Control (CDC)*, 2016.

[10] S.-C. Chang, C.-H. Chen, I.-K. Fong, and P. B. Luh, "Hydroelectric generation scheduling with an effective differential dynamic programming algorithm," *IEEE transactions on power systems*, vol. 5, no. 3, pp. 737–743, 1990.

[11] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2012, pp. 4906–4913.

[12] T. Luukkonen, "Modelling and control of quadcopter," *Independent research project in applied mathematics, Espoo*, 2011.

[13] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004.

[14] R. Tedrake, "Underactuated robotics: Algorithms for walking, running, swimming, flying, and manipulation (course notes for mit 6.832)." 2016.

[15] P. E. Gill, W. Murray, and M. A. Saunders, "Snopt: An sqp algorithm for large-scale constrained optimization," 2002.

[16] M. Posa, S. Kuindersma, and R. Tedrake, "Optimization and stabilization of trajectories for constrained dynamical systems," in *Proceedings of the International Conference on Robotics and Automation (ICRA)*, IEEE. Stockholm, Sweden: IEEE, 2016.

[17] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[18] 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, Dec 2016.