# Constrained Unscented Dynamic Programming

Brian Plancher, Zachary Manchester, and Scott Kuindersma

*Abstract*— **Differential Dynamic Programming (DDP) has become a popular approach to performing trajectory optimization for complex, underactuated robots. However, DDP presents two practical challenges. First, the evaluation of dynamics derivatives during optimization creates a computational bottleneck, particularly in implementations that capture second-order dynamic effects. Second, constraints on the states (e.g., boundary conditions, collision constraints, etc.) require additional care since the state trajectory is implicitly defined from the inputs and dynamics. This paper addresses both of these problems by building on recent work on Unscented Dynamic Programming (UDP)—which eliminates dynamics derivative computations in DDP—to support general nonlinear state and input constraints using an augmented Lagrangian. The resulting algorithm has the same computational cost as first-order penalty-based DDP variants, but can achieve constraint satisfaction to high precision without the numerical ill-conditioning associated with penalty methods. We present results demonstrating its favorable performance on several simulated robot systems including a quadrotor and 7-DoF robot arm.**

## I. Introduction

Trajectory optimization algorithms [1] are a powerful set of tools for synthesizing dynamic motions for complex robots [2], [3], [4], [5], [6]. Most robot tasks of interest include state constraints relating to, e.g., obstacle avoidance, reaching a desired goal state, and maintaining contact with the environment. Direct transcription methods for trajectory optimization parameterize both the state and input trajectories, which allows them to easily handle such constraints by formulating a large (and sparse) nonlinear program that can be solved using off-the-shelf Sequential Quadratic Programming (SQP) packages. In contrast, Differential Dynamic Programming (DDP) parameterizes only the input trajectory and uses Bellman's optimality principle to iteratively solve a sequence of much smaller optimization problems [7], [8]. Recently, DDP and its variants have received increased attention due to growing evidence that online planning is possible for high-dimensional robots [4], [9]. However, constraints are typically addressed approximately by augmenting the native cost with constraint penalty functions. This paper introduces a variant of DDP that captures nonlinear constraints on states and inputs with high accuracy while maintaining favorable convergence properties.

The classical DDP algorithm begins by simulating the dynamics forward from an initial state with an initial guess for the input trajectory. An affine control law is then computed in a backwards pass using local quadratic approximations of the cost-to-go. The forward pass is then performed again using this control law to update the input trajectory. This process is repeated until convergence.

Second derivatives of the dynamics (rank-three tensors) are required to compute the quadratic cost-to-go approximations used in DDP. This computational bottleneck led to the development of the iterative Linear Quadratic Regulator (iLQR) algorithm [10], which uses only first derivatives of the dynamics to reduce computation time at the expense of slower convergence. Recent work has proposed a completely derivative-free variant of DDP that uses a deterministic sampling scheme inspired by the unscented Kalman filter. The resulting algorithm, Unscented Dynamic Programming (UDP) [11], has the same computational complexity per-iteration as iLQR with finite-difference derivatives, but provides empirical convergence approaching that of the full second-order DDP algorithm. The primary contribution of this paper is an extension of UDP that supports nonlinear constraints on states and inputs.

Several authors have proposed methods for adding constraints to DDP methods. For the special case of box constraints on input variables, quadratic programs (QPs) can be solved in the backwards pass [12], [13]. In this setting, bounds on the inputs are enforced by projecting the feedback terms onto the constraint surface. Farshidian et al. [14] extends this to equality constraints on both the state and input through a similar projection framework while pure state constraints are still handled via penalty functions.

Designing effective penalty functions and continuation schedules can be difficult. One common approach is to increase penalty weighting coefficients in the cost function until convergence to a result that satisfies the constraints [14]. However, it is well known that these methods often lead to numerical ill-conditioning before reaching the desired constraint tolerance [15], [16]. In practice, this leads to infeasible trajectories or collisions when run on hardware. Despite this, penalty methods have seen broad application in robotics. For example, van den Berg [17] uses exponential barrier cost terms in the LQR Smoothing algorithm to prevent a quadrotor from colliding with cylindrical obstacles.

The augmented Lagrangian approach to solving constrained nonlinear optimization problems was conceived to overcome the conditioning problems of penalty methods by adding a linear Lagrange multiplier term to the objective [16]. As pointed out by other researchers [18], these methods may be particularly well-suited to trajectory optimization problems as they allow the solver to temporarily traverse infeasible regions and aggressively move towards local optima before making incremental adjustments to satisfy constraints. A theoretical formulation of this method

School of Engineering and Applied Sciences, Harvard University, Cambridge, MA. brian_plancher@g.harvard.edu, {zmanchester, scottk}@seas.harvard.edu

was proposed for use in the DDP context over two decades ago [19] and was later used to develop a hybrid-DDP algorithm [20], but this work appears to have received little attention in the robotics community. We compare our constrained UDP algorithm against this method.

In the remainder of the paper, we review key concepts from DDP, augmented Lagrangian methods, and the unscented transform (Section II), introduce the constrained UDP algorithm (Section III), and describe our experimental results on an inverted pendulum, a quadrotor flying through a virtual forest, and a manipulator avoiding obstacles (Section IV). Several practical considerations are also discussed.

## II. Background

In the following subsections, we summarize key ideas from DDP, augmented Lagrangian methods, and the unscented transform, all of which form the basis for the constrained UDP algorithm described in Section III.

### A. Differential Dynamic Programming

We assume a discrete-time nonlinear dynamical system of the form:

$$x_{k+1} = f(x_k, u_k), \tag{1}$$

where $x \in \mathbb{R}^n$ is the system state and $u \in \mathbb{R}^m$ is a control input. The goal is to find an input trajectory, $U = \{u_0, \ldots, u_{N-1}\}$, that minimizes an additive cost function,

$$J(x_0, U) = \ell_f(x_N) + \sum_{k=0}^{N-1} \ell(x_k, u_k), \tag{2}$$

where $x_0$ is the initial state and $x_1, \ldots, x_N$ are computed by integrating forward the dynamics (1).

Using Bellman's principle of optimality [21], we can define the *optimal cost-to-go*, $V_k(x)$, by the recurrence relation:

$$\begin{aligned} V_N(x) &= \ell_f(x_N) \\ V_k(x) &= \min_u \; \ell(x,u) + V_{k+1}(f(x,u)). \end{aligned} \tag{3}$$

When interpreted as an update procedure, this relationship leads to classical dynamic programming algorithms [21]. However, the curse of dimensionality prevents direct application of dynamic programming to most systems of interest to the robotics community. In addition, while $V_N(x) = \ell_f(x_N)$, and often has a simple analytical form, $V_k(x)$ will typically have complex geometry that is difficult to represent due to the nonlinearity of the dynamics (1). DDP avoids these difficulties by settling for *local* approximations to the cost-to-go along a trajectory.

We define $Q(\delta x, \delta u)$ as the local change in the minimization argument in (3) under perturbations, $\delta x, \delta u$:

$$\begin{aligned} Q(\delta x, \delta u) = {} & \ell(x + \delta x, u + \delta u) \\ & + V(f(x + \delta x, u + \delta u)) \\ & - \ell(x,u) - V(f(x,u)). \end{aligned} \tag{4}$$

Taking the second-order Taylor expansion of $Q$, we have:

$$Q(\delta x, \delta u) \approx \frac{1}{2} \begin{bmatrix} 1 \\ \delta x \\ \delta u \end{bmatrix}^T \begin{bmatrix} 0 & Q_x^T & Q_u^T \\ Q_x & Q_{xx} & Q_{xu} \\ Q_u & Q_{xu}^T & Q_{uu} \end{bmatrix} \begin{bmatrix} 1 \\ \delta x \\ \delta u \end{bmatrix}, \tag{5}$$

where the block matrices are computed as:

$$\begin{aligned} Q_{xx} &= \ell_{xx} + f_x^T V'_{xx} f_x + V'_x \cdot f_{xx} \\ Q_{uu} &= \ell_{uu} + f_u^T V'_{xx} f_u + V'_x \cdot f_{uu} \\ Q_{xu} &= \ell_{xu} + f_x^T V'_{xx} f_u + V'_x \cdot f_{xu} \\ Q_x &= \ell_x + f_x^T V'_x \\ Q_u &= \ell_u + f_u^T V'_x. \end{aligned} \tag{6}$$

Following the notation used elsewhere [4], we have dropped explicit time indices and used a prime to indicate the next timestep. Derivatives with respect to $x$ and $u$ are are denoted with subscripts. The rightmost terms in the equations for $Q_{xx}$, $Q_{uu}$, and $Q_{xu}$ involve second derivatives of the dynamics, which are rank-three tensors. As mentioned previously, these tensor calculations are relatively expensive and are often omitted, resulting in the iLQR algorithm [10].

Minimizing equation (5) with respect to $\delta u$ results in the following correction to the control trajectory:

$$\delta u = -Q_{uu}^{-1} \left( Q_{ux} \delta x + Q_u \right) \equiv K \delta x + d, \tag{7}$$

which consists of an affine term $d$ and a linear feedback term $K\delta x$. These terms can be substituted back into equation (5) to obtain an updated quadratic model of $V$:

$$\begin{aligned} \Delta V &= -\frac{1}{2} Q_u Q_{uu}^{-1} Q_u \\ V_x &= Q_x - Q_u Q_{uu}^{-1} Q_{ux} \\ V_{xx} &= Q_{xx} - Q_{xu} Q_{uu}^{-1} Q_{ux}. \end{aligned} \tag{8}$$

Therefore, a backward update pass can be performed starting at the final state, $x_N$, by setting $V_N = \ell_f(x_N)$, and iteratively applying the above computations. A forward simulation pass is then performed to compute a new state trajectory using the updated controls. This forward-backward process is repeated until the algorithm converges within a specified tolerance.

DDP, like other variants of Newton's method, can achieve quadratic convergence near a local optimum [8], [22]. However, care must be taken to ensure good convergence behavior from arbitrary initialization: A line search parameter, $\alpha$, must be added to the forward pass to ensure a satisfactory decrease in cost, and a regularization term, $\rho$, must be added to $Q_{uu}$ in equation (7) to ensure positive-definitiveness [23].

### B. Unscented Dynamic Programming

UDP replaces the gradient and Hessian calculations in equation (6) with approximations computed from a set of sample points [11]. Inspired by the Unscented Kalman Filter [24], points are sampled from a level set of the cost-to go function and propagated backward in time through the nonlinear dynamics. They are then used to compute a new cost-to-go to approximation at the earlier timestep.

To compute the derivatives of $V(f(x,u))$ appearing in the Hessian (6), a set of $2(n+m)$ sample points are generated from the columns of the following matrix:

$$L = chol\left(\begin{bmatrix} V'_{xx} & 0 \\ 0 & \ell_{uu} \end{bmatrix}^{-1}\right). \tag{9}$$

Each column, $L_i$, is scaled by a constant factor, $\beta$, and both added to and subtracted from the vector $[x'; u]$ (again, using the shorthand $x = x_k$, $u = u_k$, $x' = x_{k+1}$):

$$\begin{bmatrix} \tilde{x}'_i \\ \tilde{u}_i \end{bmatrix} = \begin{bmatrix} x' \\ u \end{bmatrix} + \beta L_i \qquad \begin{bmatrix} \tilde{x}'_{i+m+n} \\ \tilde{u}_{i+m+n} \end{bmatrix} = \begin{bmatrix} x' \\ u \end{bmatrix} - \beta L_i. \tag{10}$$

The samples are then propagated backwards through the dynamics such that $\tilde{x}_i = f^{-1}(\tilde{x}'_i, \tilde{u}_i)$. A backwards dynamics function can always be defined for a continuous-time dynamical system by simply integrating backwards in time using, for example, a Runge-Kutta method. Note that this problem is not well posed for dynamics that include rigid contact unless certain smoothing approximations are made [25].

Using these sample points, the Hessian in equation (6) becomes:

$$\begin{aligned} \begin{bmatrix} Q_{xx} & Q_{xu} \\ Q_{ux} & Q_{uu} \end{bmatrix} &= M^{-1} + \begin{bmatrix} \ell_{xx} & \ell_{xu} \\ \ell_{ux} & 0 \end{bmatrix} \\ M &= \frac{1}{2\beta^2} \sum_{i=1}^{2(n+m)} \left( \begin{bmatrix} \tilde{x}_i \\ \tilde{u}_i \end{bmatrix} - \begin{bmatrix} x \\ u \end{bmatrix} \right) \left( \begin{bmatrix} \tilde{x}_i \\ \tilde{u}_i \end{bmatrix} - \begin{bmatrix} x \\ u \end{bmatrix} \right)^T. \end{aligned} \tag{11}$$

The gradient terms in (6) can be computed from the same set of sample points by solving a linear system,

$$\begin{aligned} \begin{bmatrix} Q_x \\ Q_u \end{bmatrix} &= D^{-1} \begin{bmatrix} V'_x \tilde{x}_i - V'_x \tilde{x}_{i+m+n} \\ \vdots \\ V'_x \tilde{x}_{n+m} - V'_x \tilde{x}_{2(m+n)} \end{bmatrix} + \begin{bmatrix} \ell_x \\ \ell_u \end{bmatrix}, \\ D &= \begin{bmatrix} \tilde{x}_i - \tilde{x}_{i+m+n} & \dots & \tilde{x}_{m+n} - \tilde{x}_{2(m+n)} \\ \tilde{u}_i - \tilde{u}_{i+m+n} & \dots & \tilde{u}_{m+n} - \tilde{u}_{2(m+n)} \end{bmatrix}, \end{aligned} \tag{12}$$

which is equivalent to a centered finite difference.

### *C. Augmented Lagrangian Methods*

A natural approach to approximately enforcing constraints in optimization algorithms is to apply a quadratic penalty to constraint violations. Suppose that we wish to solve the generic minimization problem:

$$\begin{aligned} &\underset{z}{\text{minimize}} && g(z) \\ &\text{subject to} && c_i(z) = 0 \quad i \in \mathcal{E}, \end{aligned} \tag{13}$$

where $g(z)$ and $c_i(z)$ are smooth nonlinear functions. Penalty methods for solving constrained optimization problems start by defining a new cost function,

$$g_p(z;\mu) = g(z) + \frac{\mu}{2} \sum_{i \in \mathcal{E}} c_i(z)^T c_i(z), \tag{14}$$

where $\mu$ is a scalar weighting parameter. As $\mu \to \infty$, the minimizing value of $g_p(z;\mu)$ will converge toward satisfaction of the constraints [16]. While $\mu$ often does not have to grow unbounded for a solution to be found within a given tolerance, numerical issues are still prevalent since the condition number of the Hessian of $g_p$ grows with $\mu$ (see [16] for additional details). It is important to note that, while we focus on quadratic penalties in this paper, there are a wide variety of other penalty functions that can be used. For example, $L_1$ loss functions have also been used for collision-free path planning in robotic arms and humanoids [26].

To overcome the numerical issues associated with penalty methods, augmented Lagrangian solvers add a term to $g_p$ that estimates the Lagrange multipliers associated with the constraints:

$$\begin{aligned} \mathcal{L}_A(z;\mu,\lambda) &= \\ & g(z) + \frac{\mu}{2} \sum_{i \in \mathcal{E}} c_i(z)^T c_i(z) + \sum_{i \in \mathcal{E}} \lambda^i c_i(z) \cdot \end{aligned} \tag{15}$$

Given initial values for $\mu$ and $\lambda$, an unconstrained minimization is performed, after which $\mu$ and $\lambda$ are updated. As in penalty methods, $\mu$ is systematically increased across these "major iterations" using a predefined schedule. However, the presence of the $\lambda$ terms allows convergence with much smaller values of $\mu$.

The update for $\lambda$ at major iteration $j$ can be derived by considering the first-order necessary conditions evaluated at an approximate minimizer, $z_j$:

$$\begin{aligned} 0 \approx \nabla_z \mathcal{L}_A(z_j;\mu_j,\lambda^j) &= \\ & \nabla_z g(z_j) - \sum_{i \in \mathcal{E}} [\lambda_i^j - \mu_j c_i(z_j)] \nabla_z c_i(z_j) \cdot \end{aligned} \tag{16}$$

Recall that the first-order necessary conditions for a local solution, $z^*, \lambda^*$, of the original constrained optimization problem is given by differentiating the (true) Lagrangian [16]:

$$0 = \nabla_z g(z^*) - \sum_{i \in \mathcal{E}} \lambda_i^* \nabla_z c_i(z^*). \tag{17}$$

Comparing (16) and (17), a natural update rule for $\lambda$ arises:

$$\lambda_i^{j+1} \leftarrow \lambda_i^j - \mu_j c_i(z_j) \quad \forall i \in \mathcal{E}. \tag{18}$$

It can be shown that given $\lambda^*$, the solution, $z^*$, of (13) is a strict local minimizer of (15) for all $\mu$ above some minimum value [16]. Practically speaking, the aim is to quickly improve estimates of $\lambda^*$ so that reasonable approximations of $z^*$ can be computed by minimizing (15) without $\mu$ growing too large.

## III. Constrained Unscented Dynamic Programming

To develop the constrained UDP algorithm, we add quadratic penalty and Lagrange multiplier terms for each constraint and define an outer loop to update $\lambda$ and $\mu$ when the inner UDP solution converges. We use the following augmented Lagrangian function:

$$\begin{aligned} \mathcal{L}_A(x_0,U;\mu,\lambda) &= \ell_f(x_N) + \sum_{k=0}^{N-1} \ell(x_k,u_k) + \\ & \frac{1}{2} \sum_{k=0}^{N} c(x_k,u_k)^T I_{\mu_k} c(x_k,u_k) + \text{diag}(\lambda_k) c(x_k,u_k), \end{aligned} \tag{19}$$

where $c(x,u)$ is the vertical concatenation of all equality and inequality constraints:

$$c_i(x,u)=0,\;\; i\in\mathcal{E} \qquad c_i(x,u)\geq 0,\;\; i\in\mathcal{I}. \tag{20}$$

Note that we are using separate $\mu_k^i$ and $\lambda_k^i$ for each constraint at each timestep.

To handle inequality constraints, we follow [18] and define $I_{\mu_k}$ as a diagonal matrix that encodes the active constraints:

$$I_{\mu_k}(i,i)=\begin{cases}\mu_k^i & i\in\mathcal{E},\;\; c_i(x_k,u_k)<0,\;\; \lambda_k^i>0\\ 0 & \text{otherwise}\end{cases} \tag{21}$$

Including $I_{\mu_k}$ in (19) ensures that penalties are not incurred when inequality constraints are satisfied.

At each timestep during the backward pass of the UDP algorithm, the constraint functions and their gradients are evaluated. A modified version of $Q(\delta x,\delta u)$ from equations (5)–(6), which we denote $\hat{Q}$, is then defined to include a Gauss-Newton approximation of the constraint terms:

$$\begin{aligned}
\hat{Q}_{xx} &= Q_{xx}+\frac{\partial c(x,u)}{\partial x}I_\mu\frac{\partial c(x,u)}{\partial x}\\
\hat{Q}_{uu} &= Q_{uu}+\frac{\partial c(x,u)}{\partial u}I_\mu\frac{\partial c(x,u)}{\partial u}+\rho I\\
\hat{Q}_{xu} &= Q_{xu}+\frac{\partial c(x,u)}{\partial x}I_\mu\frac{\partial c(x,u)}{\partial u}\\
\hat{Q}_{x} &= Q_{x}+c(x,u)I_\mu\frac{\partial c(x,u)}{\partial x}+\mathrm{diag}(\lambda)\frac{\partial c(x,u)}{\partial x}\\
\hat{Q}_{u} &= Q_{u}+c(x,u)I_\mu\frac{\partial c(x,u)}{\partial u}+\mathrm{diag}(\lambda)\frac{\partial c(x,u)}{\partial u},
\end{aligned} \tag{22}$$

where we have included a regularization parameter $\rho>0$. Equations (7) and (8) are used to compute the feedback policy during the backwards pass as usual.

The algorithm proceeds by running the inner augmented UDP algorithm until convergence, and then updating $\mu$ and $\lambda$ according to a set schedule until the desired feasibility tolerance is achieved. The full constrained UDP procedure is summarized in Algorithm 1.

### *A. Updating $\mu$ and $\lambda$*

As is typically done in augmented Lagrangian methods, we update $\lambda$ only if the constraint violation of the local minimizer is less than a threshold value, $\phi$, and otherwise update $\mu$. The parameter $\phi$ is updated according to a predefined schedule to help guide the algorithm toward a solution while avoiding large increases in $\mu$ early on. While there are many different variations on this schedule in the literature, most suggest a monotonically decreasing schedule for $\phi$, which results in a monotonically increasing $\mu$ [27], [16].

## IV. Examples

In this section, three numerical examples are provided to demonstrate the performance of constrained UDP. We compare UDP and iLQR using both penalty and augmented Lagrangian formulations. All of the algorithms are implemented in MATLAB. Each uses the same scheduling of updates to $\mu$ and $\lambda$, and all integration is done with a 3rd-order Runge-Kutta method and a time horizon of 4 seconds.

**Algorithm 1** Constrained UDP

```
 1: Initialize $\mu$, $\lambda$, $\phi$, $\rho$, $U$, $x_0$
 2: Perform forward pass to compute $X=\{x_0,x_1,\ldots,x_N\}$
 3: while $\max(c)>\epsilon_c$ do
 4:     while cost not converged do
 5:         Compute $V_N$ and derivatives
 6:         for $k=N-1,\ldots,0$ do
 7:             (11)–(12), (22) $\rightarrow \hat{Q}^k$
 8:             if $\hat{Q}^k_{uu}$ is invertible then
 9:                 (7) $\rightarrow K_k, d_k$                      } Backward
10:                 (8) $\rightarrow V_k$ and derivatives           } Pass
11:             else
12:                 Increase $\rho$ go to line 6
13:             end if
14:         end for
15:         $\alpha=1$
16:         $\tilde{x}_0=x_0$
17:         for $k=0,\ldots,N-1$ do
18:             $\tilde{u}_k=u_k+\alpha d_k+K_k(\tilde{x}_k-x_k)$
19:             $\tilde{x}_{k+1}=f(\tilde{x}_k,\tilde{u}_k)$
20:         end for                                             } Forward
21:         Compute $\tilde{J}$ using (2) and $\tilde{X},\tilde{U}$   } Pass
22:         if $\tilde{J}$ satisfies line search criteria then
23:             Update $X\leftarrow\tilde{X}$, $U\leftarrow\tilde{U}$
24:         else
25:             Reduce $\alpha$ and go to line 17
26:         end if
27:     end while
28:     for $k=0,\ldots,N$ do
29:         for $i\in\mathcal{E}\cup\mathcal{I}$ do
30:             if $c_i^k<\phi_i^k$ then
31:                 Update $\lambda_i^k$ using (18)
32:                 Reduce $\phi_i^k$                                } Outer
33:             else                                                } Loop
34:                 Increase $\mu_k^i$                               } Updates
35:             end if
36:         end for
37:     end for
38: end while
```

### *A. Inverted Pendulum*

We first consider the classic inverted-pendulum system and swing-up task. We define the state vector to be $x=[y,\theta,\dot{y},\dot{\theta}]^T$, where $y$ is the translation of the cart and $\theta$ is the angle of the pendulum measured from the downward equilibrium. The initial state is $x_0=[0,0,0,0]^T$ and the goal state is $x_g=[0,\pi,0,0]^T$. We use a quadratic cost function of the form:

$$\begin{aligned}
J=&\frac{1}{2}(x_N-x_g)^TQ_N(x_N-x_g)+\\
&\sum_{k=0}^{N-1}\frac{1}{2}(x_k-x_g)^TQ(x_k-x_g)+\frac{1}{2}u_k^TRu_k,
\end{aligned} \tag{23}$$

where $Q=0.1\times I_{4x4}$, $R=0.01$, and $Q_N=1000\times I_{4x4}$. We set the number of knot points to $N=120$ and initialized the algorithms with all of the states and controls set to 0. We also enforced an input constraint of ±30 N, and a final state constraint of $x_N = x_g$. We ran optimizations at three different constraint tolerances: $1e^{-2}$ ("low precision"), $1e^{-4}$ ("medium precision"), and $5e^{-7}$ ("high precision"). In all cases, the intermediate cost convergence tolerance was set to $1e^{-2}$ and the final iteration cost convergence tolerance was set to $1e^{-6}$. We also set a maximum value of $1e^{30}$ for $\mu$, since allowing it to grow much larger led to poor numerical conditioning. For both penalty-based and augmented UDP, $\beta$ was set to $1e^{-2}$.

Figure 1 provides some intuition for how the constrained UDP algorithm solves the inverted pendulum swing-up problem. The vertical dashed black lines indicate outer loop updates. Intuitively, the first major iteration finds a local minimum of the primary objective despite large constraint violations. Later iterations reduce constraint violation while often increasing cost (unconstrained solutions give a lower bound on cost). Despite increasing values of $\mu$ in the final few iterations, the product of $c(x, u)$ and $\mu$ remains relatively constant due to the corresponding decrease in constraint violation. We also observed that the algorithm often violates input constraints temporarily to guide the state trajectory to feasible regions of state space. This behavior is qualitatively different from SQP methods, where linearized constraints are strictly satisfied during each major iteration.

In our initial experiments, we found that using a separate $\mu$ and $\lambda$ for each constraint at each timestep allows the algorithm more flexibility in adjusting the penalty parameters to allow for these temporary constraint violations. Unfortunately, this flexibility sometimes comes at the price of additional major iterations, where adjacent timesteps pass large inputs back-and-forth until both $\mu$ values are increased sufficiently.

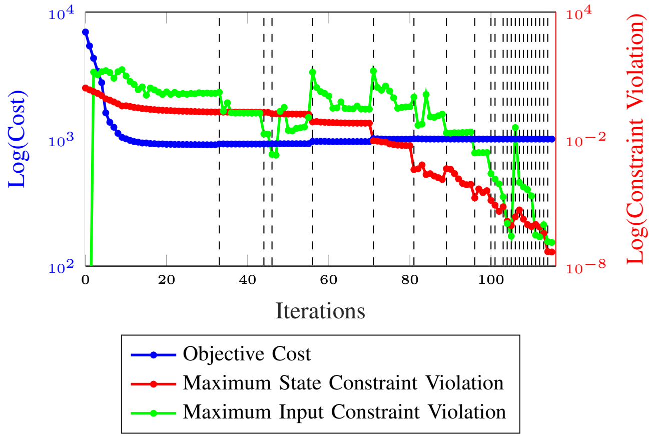

Fig. 1. Cost and constraint violation per iteration for the constrained UDP algorithm for the inverted pendulum.

Figure 2 shows a typical output from the augmented iLQR algorithm (iLQR-A), which fails in the high-precision case, exiting after reaching a limit of 100 major iterations. The algorithm makes progress until the upper bound of $\mu \leq 1e^{30}$ is reached on the final state constraint penalty. In later iterations, a trade-off occurs between improving satisfaction of the input constraints and the final state constraint, and the algorithm is unable to make further progress.

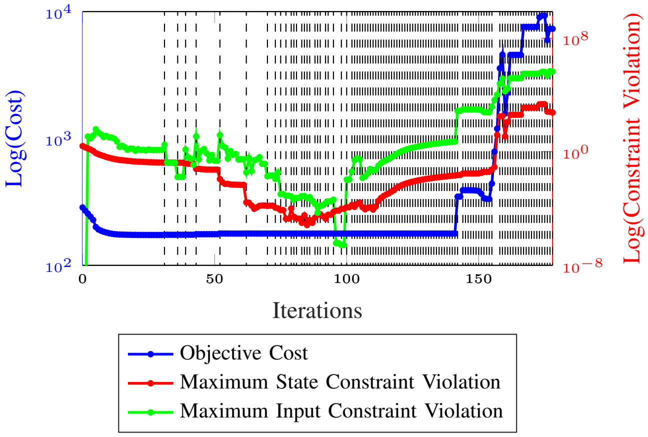

Fig. 2. Inverted pendulum cost and constraint violation per iteration for the augmented iLQR algorithm failing with $5e^{-7}$ constraint tolerance.

Complete results from our inverted pendulum experiments are given in Table I. Results in red indicate a failure to meet the required constraint tolerance. All of the algorithms were able to converge in the low-precision setting with very similar total cost, but the penalty methods required much larger maximum values for $\mu$. We also found that iLQR-A finished the fastest, although UDP-A found a lower final cost. In the medium-precision setting, both augmented Lagrangian methods succeeded, while the penalty methods failed to achieve state constraint tolerance and exited after reaching 100 major iterations. In the high-precision setting, the constrained UDP algorithm (UDP-A) was able to find a feasible solution while all other algorithms failed. This experiment and those that follow show that, for loose constraint tolerances, the use of both iLQR and penalty methods may be sufficient, but UDP-A is superior when high-precision is desired.

## B. *Quadrotor*

The objective for this example is to compute a trajectory that flies a quadrotor through a forest while avoiding collisions with trees. The quadrotor has four inputs, corresponding to the thrust of each rotor, and twelve states corresponding to the position and Euler angles, along with their respective first derivatives. A quadratic cost function with $Q = 0.1 \times I_{12x12}$, $R = 0.01 \times I_{4x4}$, and $Q_N = 1000 \times I_{12x12}$ was used, with $N = 120$ knot points. All control inputs were initialized to 0, an input constraint of $-10 \leq u \leq 10$ was applied, and a no-collision constraints with the trees and a final state constraint of $x_N = x_g$ were enforced. The algorithms were again tested at three different constraint tolerance values. For the unscented variants, $\beta$ was set to $1e - 4$. Figure 3 shows a final trajectory computed by the constrained UDP algorithm.

The full results are shown below in Table II. As in the previous example, all of the algorithms converged to the

**Low precision:** $\max(c) < 1e^{-2}$, initial $\phi = 1e^{-1}$

| | Iters | Cost | $c_x$ | $c_u$ | $\mu_x$ | $\mu_u$ |
|---|---|---|---|---|---|---|
| iLQR-P | 254 | 1006.1 | $2.3e^{-4}$ | $\mathbf{5.8e^{-7}}$ | $1e^{7}$ | $1e^{7}$ |
| UDP-P | 229 | 1022.4 | $\mathbf{1.2e^{-4}}$ | $1.1e^{-6}$ | $1e^{8}$ | $1e^{8}$ |
| iLQR-A | **93** | 1001.0 | $8.2e^{-3}$ | $1.8e^{-3}$ | $\mathbf{1e^{5}}$ | $\mathbf{1e^{3}}$ |
| UDP-A | 140 | **999.6** | $8.8e^{-3}$ | $4.8e^{-4}$ | $\mathbf{1e^{5}}$ | $\mathbf{1e^{3}}$ |

**Medium precision:** $\max(c) < 1e^{-4}$, initial $\phi = 1e^{-2}$

| | Iters | Cost | $c_x$ | $c_u$ | $\mu_x$ | $\mu_u$ |
|---|---|---|---|---|---|---|
| iLQR-P | 231 | 1011.9 | $1.7e^{-4}$ | $1.8e^{-4}$ | $1e^{11}$ | $1e^{11}$ |
| UDP-P | 258 | 1054.2 | $4.1e^{-4}$ | $3.4e^{-8}$ | $1e^{16}$ | $1e^{16}$ |
| iLQR-A | **98** | 1001.4 | $2.4e^{-5}$ | $7.7e^{-5}$ | $1e^{7}$ | $1e^{8}$ |
| UDP-A | 126 | **999.8** | $\mathbf{9.0^{e-6}}$ | $\mathbf{3.7e^{-5}}$ | $\mathbf{1e^{6}}$ | $\mathbf{1e^{4}}$ |

**High precision:** $\max(c) < 5e^{-7}$, initial $\phi = 5e^{-3}$

| | Iters | Cost | $c_x$ | $c_u$ | $\mu_x$ | $\mu_u$ |
|---|---|---|---|---|---|---|
| iLQR-A | 179 | 1001.7 | $3.7e^{-5}$ | $5.0e^{-5}$ | $1e^{17}$ | $1e^{13}$ |
| UDP-A | **117** | **999.8** | $\mathbf{4.6e^{-8}}$ | $\mathbf{1.3e^{-7}}$ | $\mathbf{1e^{8}}$ | $\mathbf{1e^{7}}$ |

TABLE I

INVERTED PENDULUM OPTIMIZATION RESULTS. RED INDICATES FAILURE TO MEET CONSTRAINT TOLERANCE. SEE TEXT FOR DETAILS.

**Low precision:** $\max(c) < 1e^{-2}$, initial $\phi = 1e^{-1}$

| | Iters | Cost | $c_x$ | $c_u$ | $\mu_x$ | $\mu_u$ |
|---|---|---|---|---|---|---|
| iLQR-P | 272 | 758.6 | $\mathbf{2.8e^{-4}}$ | $\mathbf{2.9e^{-6}}$ | $1e^{6}$ | $1e^{6}$ |
| UDP-P | 125 | 727.4 | $2.0e^{-3}$ | $7.8e^{-6}$ | $1e^{5}$ | $1e^{5}$ |
| iLQR-A | **109** | 712.4 | $3.7e^{-3}$ | $3.2e^{-3}$ | $1e^{5}$ | $\mathbf{1e^{2}}$ |
| UDP-A | 115 | **707.3** | $7.6e^{-3}$ | $2.3e^{-3}$ | $\mathbf{1e^{4}}$ | $\mathbf{1e^{2}}$ |

**Medium precision:** $\max(c) < 1e^{-4}$, initial $\phi = 1e^{-2}$

| | Iters | Cost | $c_x$ | $c_u$ | $\mu_x$ | $\mu_u$ |
|---|---|---|---|---|---|---|
| iLQR-P | 223 | 764.2 | $1.4e^{-3}$ | $3.2e^{-4}$ | $1e^{12}$ | $1e^{12}$ |
| UDP-P | **114** | 729.6 | $\mathbf{2.9e^{-5}}$ | $\mathbf{2.2e^{-7}}$ | $1e^{10}$ | $1e^{10}$ |
| iLQR-A | 253 | 712.6 | $3.3e^{-4}$ | $2.1e^{-4}$ | $1e^{8}$ | $1e^{5}$ |
| UDP-A | 158 | **708.8** | $9.1e^{-5}$ | $1.8e^{-6}$ | $\mathbf{1e^{8}}$ | $\mathbf{1e^{5}}$ |

**High precision:** $\max(c) < 1e^{-6}$, initial $\phi = 5e^{-3}$

| | Iters | Cost | $c_x$ | $c_u$ | $\mu_x$ | $\mu_u$ |
|---|---|---|---|---|---|---|
| UDP-P | 201 | 729.6 | $2.9e^{-5}$ | $2.2e^{-7}$ | $1e^{11}$ | $1e^{11}$ |
| UDP-A | **149** | **708.8** | $\mathbf{5.3e^{-7}}$ | $\mathbf{1.7e^{-7}}$ | $\mathbf{1e^{8}}$ | $\mathbf{1e^{4}}$ |

TABLE II

QUADROTOR OPTIMIZATION RESULTS. RED INDICATES FAILURE TO MEET CONSTRAINT TOLERANCE. SEE TEXT FOR DETAILS.

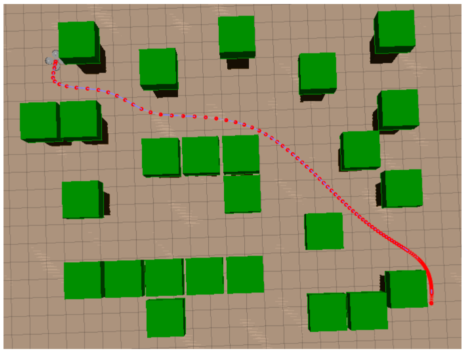

Fig. 3. Collision-free quadrotor trajectory computed by constrained UDP.

low-precision constraint tolerance, with the augmented Lagrangian variants requiring fewer iterations. For the medium-precision case, both iLQR methods failed to achieve the required tolerance. We hypothesize that the higher-order information provided by the UDP backup procedure is responsible for its improved convergence [11]. For tight constraint tolerances, only UDP-A is able to find a feasible trajectory.

### C. Robotic Arm

The objective for this example is to compute a trajectory for a Kuka LBR IIWA 14 robotic arm to place a rigid object onto a shelf while avoiding an obstacle in its workspace. The state vector is comprised of the 7 joint positions and velocities. We again used a quadratic cost function with $Q = I_{14x14}$, $R = 1e^{-4} \times I_{7x7}$, and $Q_N = 1000 \times I_{14x14}$, with $N = 400$, all controls initialized to 0, input constraints of $-200 \leq u \leq 200$ Nm on each joint, a no collision constraint with the obstacle, and a final state constraint of $x_N = x_g$. The algorithms were again tested at three different constraint tolerance values. For the unscented variants, $\beta$ was set to $1e^{-4}$. In all cases, we set our intermediate cost convergence tolerance to 10 and our final iteration cost tolerance to $1e^{-2}$. A screen shot of the trajectory computed by the constrained UDP algorithm in the high precision setting is shown in Figure 4.

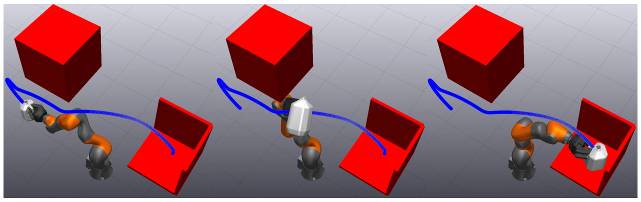

Fig. 4. Collision-free Kuka arm trajectory computed by constrained UDP.

Table III contains the results of all trials. As before, all of the algorithms handled the low-precision case, which produced similar overall costs. The penalty methods converged faster, with lower constraint violation, and larger $\mu$ values. Despite their success in the low precision case, however, both penalty methods failed in the medium precision case. Once again, in the high precision case, only the constrained UDP method succeeded. This example further demonstrates how penalty and iLQR based methods may succeed with loose constraint tolerances, but the constrained UDP algorithm can support much more precise constraint satisfaction.

## V. CONCLUSION AND FUTURE WORK

We have presented the constrained UDP algorithm, a DDP variant capable of satisfying nonlinear state and input constraints with high accuracy through the use of an augmented Lagrangian. Several directions for future research remain. Combining multiple constraint-handling approaches may prove beneficial. For example, box constraints on inputs were captured using cost terms in our experiments. It would be straightforward to instead use existing QP techniques in the backward pass to compute input constraints [12],

**Low precision:** $\max(c) < 1e^{-2}$, initial $\phi = 5e^{-1}$

| | Iters | Cost | $c_x$ | $c_u$ | $\mu_x$ | $\mu_u$ |
|---|---|---|---|---|---|---|
| iLQR-P | **50** | 2161.4 | $9.5e^{-3}$ | $6.5e^{-8}$ | $\mathbf{1e^{6}}$ | $1e^{6}$ |
| UDP-P | 53 | **2155.4** | $8.6e^{-3}$ | $\mathbf{5.1e^{-8}}$ | $\mathbf{1e^{6}}$ | $1e^{6}$ |
| iLQR-A | 89 | 2171.3 | $7.1e^{-3}$ | $6.1e^{-3}$ | $\mathbf{1e^{6}}$ | $\mathbf{1e^{2}}$ |
| UDP-A | 88 | 2174.9 | $\mathbf{6.3e^{-3}}$ | $5.4e^{-3}$ | $\mathbf{1e^{6}}$ | $1e^{3}$ |

**Medium precision:** $\max(c) < 1e^{-3}$, initial $\phi = 1e^{-2}$

| | Iters | Cost | $c_x$ | $c_u$ | $\mu_x$ | $\mu_u$ |
|---|---|---|---|---|---|---|
| iLQR-P | 155 | 2161.3 | $9.6e^{-3}$ | $6.2e^{-3}$ | $1e^{17}$ | $1e^{17}$ |
| UDP-P | 146 | 2226.8 | $7.0e^{-3}$ | $5.3e^{-3}$ | $1e^{12}$ | $1e^{12}$ |
| iLQR-A | 84 | 2688.4 | $5.7e^{-4}$ | $\mathbf{1.3e^{-5}}$ | $\mathbf{1e^{8}}$ | $\mathbf{1e^{4}}$ |
| UDP-A | **82** | **2674.4** | $\mathbf{4.2e^{-4}}$ | $1.8e^{-5}$ | $\mathbf{1e^{8}}$ | $\mathbf{1e^{4}}$ |

**High precision:** $\max(c) < 5e^{-5}$, initial $\phi = 5e^{-3}$

| | Iters | Cost | $c_x$ | $c_u$ | $\mu_x$ | $\mu_u$ |
|---|---|---|---|---|---|---|
| iLQR-A | 182 | 6471.1 | $5.9e^{-3}$ | $3.8e^{-5}$ | $1e^{10}$ | $1e^{3}$ |
| UDP-A | **71** | **2674.7** | $\mathbf{2.9e^{-5}}$ | $\mathbf{7.1e^{-6}}$ | $\mathbf{1e^{10}}$ | $\mathbf{1e^{6}}$ |

TABLE III

KUKA ARM OPTIMIZATION RESULTS. RED INDICATES FAILURE TO MEET CONSTRAINT TOLERANCE. SEE TEXT FOR DETAILS.

[13], while using augmented Lagrangian terms for state constraints. An empirical comparison including barrier methods in the DDP setting would also be interesting.

As highlighted in prior work on UDP [11], the sigma point scaling parameter, $\beta$, must be chosen ahead of time for each example. Automatic approaches to setting $\beta$ remain an interesting open problem. Similarly, more work is needed to determine optimal—or even good suboptimal—schedules for $\phi$ and $\mu$. We believe our results could be significantly improved if more effort was spent exploring update schemes. Finally, we have not yet optimized our implementation of constrained UDP to minimize computation time per iteration. In future work, we will evaluate the suitability of this algorithm for realtime model predictive control (MPC).

## ACKNOWLEDGMENTS

This work was supported by an Internal Research and Development grant from Draper, Inc. The authors would like to thank Patrick Varin and the members of the Harvard Agile Robotics Lab for their useful feedback and insights.

## REFERENCES

[1] J. T. Betts, *Practical Methods for Optimal Control Using Nonlinear Programming*, vol. 3 of *Advances in Design and Control*. Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM), 2001.

[2] K. D. Mombaur, "Using optimization to create self-stable human-like running," *Robotica*, vol. 27, no. 3, pp. 321–330, 2009.

[3] M. Posa, M. Tobenkin, and R. Tedrake, "Lyapunov analysis of rigid body systems with impacts and friction via sums-of-squares," in *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control (HSCC 2013)*, pp. 63–72, 2013.

[4] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and Stabilization of Complex Behaviors through Online Trajectory Optimization," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[5] W. Xi and C. D. Remy, "Optimal Gaits and Motions for Legged Robots," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2014.

[6] 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)*, (Stockholm, Sweden), pp. 1366–1373, IEEE, 2016.

[7] D. Q. Mayne, "A second-order gradient method of optimizing nonlinear discrete time systems," *Int J Control*, vol. 3, p. 8595, 1966.

[8] D. H. Jacobson and D. Q. Mayne, *Differential Dynamic Programming*. Elsevier, 1970.

[9] J. Koenemann, A. Del Prete, Y. Tassa, E. Todorov, O. Stasse, M. Bennewitz, and N. Mansard, "Whole-body Model-Predictive Control applied to the HRP-2 Humanoid," in *Proceedings of the IEEERAS Conference on Intelligent Robots*, 2015.

[10] W. Li and E. Todorov, "Iterative Linear Quadratic Regulator Design for Nonlinear Biological Movement Systems," in *Proceedings of the 1st International Conference on Informatics in Control, Automation and Robotics*, (Setubal, Portugal), 2004.

[11] Z. Manchester and S. Kuindersma, "Derivative-Free Trajectory Optimization with Unscented Dynamic Programming," in *Proceedings of the 55th Conference on Decision and Control (CDC)*, (Las Vegas, NV), 2016.

[12] J. F. O. D. O. Pantoja and D. Q. Mayne, "A sequential quadratic programming algorithm for discrete optimal control problems with control inequality constraints," in *Proceedings of the 28th IEEE Conference on Decision and Control,*, pp. 353–357 vol.1, Dec. 1989.

[13] Y. Tassa, T. Erez, and E. Todorov, "Control-Limited Differential Dynamic Programming," in *Proceedings of the International Conference on Robotics and Automation (ICRA)*, May 2014.

[14] F. Farshidian, M. Neunert, A. W. Winkler, G. Rey, and J. Buchli, "An Efficient Optimal Planning and Control Framework For Quadrupedal Locomotion," *arXiv:1609.09861 [cs]*, Sept. 2016.

[15] A. Forsgren, P. Gill, and M. Wright, "Interior Methods for Nonlinear Optimization," *SIAM Review*, vol. 44, pp. 525–597, Jan. 2002.

[16] J. Nocedal and S. J. Wright, *Numerical Optimization*. Springer, 2nd ed., 2006.

[17] J. van den Berg, "Iterated LQR smoothing for locally-optimal feedback control of systems with non-linear dynamics and non-quadratic cost," in *American Control Conference (ACC), 2014*, pp. 1912–1918, June 2014.

[18] M. Toussaint, "A Novel Augmented Lagrangian Approach for Inequalities and Convergent Any-Time Non-Central Updates," *arXiv:1412.4329 [math]*, Dec. 2014.

[19] T. C. Lin and J. S. Arora, "Differential dynamic programming technique for constrained optimal control," *Computational Mechanics*, vol. 9, pp. 27–40, Jan. 1991.

[20] G. Lantoine and R. P. Russell, "A Hybrid Differential Dynamic Programming Algorithm for Constrained Optimal Control Problems. Part 1: Theory," *Journal of Optimization Theory and Applications*, vol. 154, pp. 382–417, Aug. 2012.

[21] R. Bellman, *Dynamic Programming*. Dover, 1957.

[22] L.-z. Liao and C. A. Shoemaker, "Advantages of Differential Dynamic Programming Over Newton's Method for Discrete-time Optimal Control Problems," technical report, Cornell University, July 1992.

[23] Y. Tassa, *Theory and Implementation of Biomimetic Motor Controllers*. PhD thesis, Feb. 2011.

[24] S. J. Julier and J. K. Uhlmann, "Unscented filtering and nonlinear estimation," *Proceedings of the IEEE*, vol. 92, no. 3, pp. 401–422, 2004.

[25] E. Todorov, "A convex, smooth and invertible contact model for trajectory optimization," in *Proceedings of the International Conference on Robotics and Automation (ICRA)*, Feb. 2011.

[26] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *The International Journal of Robotics Research*, vol. 33, pp. 1251–1270, Aug. 2014.

[27] D. P. Bertsekas, "Multiplier Methods: A Survey," *Automatica*, vol. 12, pp. 133–145, Mar. 1976.