Derivative-Free Trajectory Optimization with Unscented Dynamic Programming

Zachary Manchester and Scott Kuindersma

***Abstract*— Trajectory optimization algorithms are a core technology behind many modern nonlinear control applications. However, with increasing system complexity, the computation of dynamics derivatives during optimization creates a computational bottleneck, particularly in second-order methods. In this paper, we present a modification of the classical Differential Dynamic Programming (DDP) algorithm that eliminates the computation of dynamics derivatives while maintaining similar convergence properties. Rather than relying on naive finite difference calculations, we propose a deterministic sampling scheme inspired by the Unscented Kalman Filter that propagates a quadratic approximation of the cost-to-go function through the nonlinear dynamics at each time step. Our algorithm takes larger steps than Iterative LQR—a DDP variant that approximates the cost-to-go Hessian using only first derivatives—while maintaining the same computational cost. We present results demonstrating its numerical performance in simulated balancing and aerobatic flight experiments.**

I. Introduction

Trajectory optimization algorithms are a powerful class of methods for generating goal-directed behavior in dynamical systems by computing admissible state and control sequences that minimize a cost functional subject to a set of constraints [1]. In many applications—such as robotics—computation time is a critical factor driving algorithm selection. Differential Dynamic Programming (DDP) [2] is a second-order method with favorable quadratic convergence properties for smooth discrete-time systems [3], [4]. The algorithm proceeds iteratively by simulating the dynamics forward and computing updates to the control inputs backwards in time using local quadratic models of the cost-to-go (see Section II-A). Importantly, DDP must compute second-order derivatives of the dynamics in the backwards phase, which are almost always the most expensive part of the computation. A closely related algorithm, called Iterative LQR (iLQR) [5], uses only the first-order derivatives of the dynamics to reduce computation time at the expense of slower convergence. The relationship between DDP and iLQR is similar to the relationship between Newton's method and the Gauss-Newton method in that the Hessian is approximated using only the Jacobian [6].

In this paper, we present a new variant of DDP that maintains the efficient computational properties of iLQR while achieving superior convergence rates. Our algorithm, called Unscented Dynamic Programming (UDP), avoids analytical computation of dynamics derivatives using a sampling scheme inspired by the Unscented Kalman Filter (UKF). The Unscented Transform has been used previously to propagate probability distributions in stochastic trajectory optimization algorithms such as Iterative Local Dynamic Programming (iLDP) [7]. Our approach is fundamentally different in dealing with the deterministic case and using the Unscented Transform to propagate the cost-to-go function, rather than a probability distribution.

DDP and its variants fall into the general family of *shooting methods* that are built on the theoretical foundation of Pontryagin's Minimum Principle [8]. By parameterizing only the controls and simulating the dynamics forward to evaluate the cost, they create compact optimization problems for which local minima can often be identified quickly. Among the limitations of these techniques are sensitivity to input trajectory initializations and numerical conditioning issues sometimes referred to as the "tail wagging the dog" phenomenon [1]. An alternative class of algorithms, called *direct methods*, parameterize both the state and control trajectories and solve large (often sparse) nonlinear programs using off-the-shelf sequential quadratic programming packages [1], [9]. These formulations are particularly useful for problems with many state and input constraints. Both classes of algorithms have demonstrated value in challenging optimization problems, including high-dimensional robot motion planning [10], [11], [12], [13].

Efficient implementations of iLQR have been used for Model-Predictive Control (MPC) of humanoid robots in simulation [14], [15] and hardware [11]. In the MPC setting, the algorithm is often terminated before convergence due to real-time requirements. We show that our approach generally takes larger steps than iLQR, making it particularly attractive for these applications. Recent work has explored LQR smoothing (inspired by Kalman smoothing) as an alternative to iLQR [16]. We anticipate that an analogous application of UKF ideas could be made for this class of algorithms, although we do not pursue such development in this paper.

In the remainder of the paper, we review basic concepts from the DDP algorithm and the Unscented Transform (Section II), introduce the UDP algorithm (Section III), and describe experimental results comparing UDP against the classic DDP and iLQR algorithms (Section IV).

II. Background

For completeness and to establish notation, we derive the classic DDP algorithm and the Unscented Transform in the following subsections.

John A. Paulson School of Engineering and Applied Sciences, Harvard University, Cambridge, MA, USA. {zmanchester,scottk}@seas.harvard.edu

A. Differential Dynamic Programming

The DDP algorithm assumes 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, $u \in \mathbb{R}^m$ is a control input, and an additive cost function,

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

where $X = \{x_1, \ldots, x_N\}$ and $U = \{u_1, \ldots, u_{N-1}\}$ are the state and input trajectories, respectively. In the development below, we further assume that $\ell_f(x_N)$ and $\ell(x_k, u_k)$ are quadratic and that the gradients and Hessians are therefore trivially defined. Second-order Taylor expansions of non-quadratic cost functions could also be substituted. Specifically, let

$$\begin{aligned} J = &\frac{1}{2} x_N^T W_N x_N + w_N^T x_N \\ &+ \sum_{k=1}^{N-1} \frac{1}{2} x_k^T W_k x_k + w_k^T x_k + \frac{1}{2} u_k^T R_k u_k + r_k^T u_k, \end{aligned} \tag{3}$$

where $W \in \mathbf{S}_+^n$ and $R \in \mathbf{S}_{++}^m$ are, respectively, positive-semidefinite and positive-definite cost weighting matrices, and $w \in \mathbb{R}^n$ and $r \in \mathbb{R}^m$ are cost weighting vectors.

The *optimal cost-to-go*, $V_k^*(x)$, gives the total cost that will be accumulated between time steps k and N, starting in state x, if the optimal control policy is followed. Using Bellman's principle of optimality [17], this function can be written as a simple recurrence relation,

$$\begin{aligned} V_N^*(x) &= \frac{1}{2} x^T W_N x + w_N^T x \\ V_k^*(x) &= \min_u \; Q_k^*(x, u) \\ &\equiv \min_u \; \Big(\frac{1}{2} x^T W_k x + w_k^T x \\ &\qquad + \frac{1}{2} u^T R_k u + r_k^T u + V_{k+1}^*(f(x, u)) \Big). \end{aligned} \tag{4}$$

When interpreted as an update procedure, this relationship leads to classical dynamic programming algorithms. However, while $V_N^*(x)$ is quadratic, $V_k^*(x)$ will not be in general due to the nonlinearity of the dynamics. Cost-to-go functions often have complex geometry that is difficult to represent and expensive to compute. DDP overcomes this difficulty by starting with an initial guess trajectory and approximating the cost-to-go function as locally quadratic near that trajectory,

$$V_k(x + \delta x) \approx V_k(x) + \frac{1}{2} \delta x^T H_k \delta x + g_k^T \delta x, \tag{5}$$

where H_k and g_k are the Hessian and gradient of V_k evaluated at x, respectively.

The algorithm proceeds by calculating the following approximate cost-to-go at each time step,

$$\begin{aligned} Q_k(x + \delta x, u + \delta u) &\approx Q_k(x, u) \\ &+ \frac{1}{2} \begin{bmatrix} \delta x \\ \delta u \end{bmatrix}^T \begin{bmatrix} A_k & C_k^T \\ C_k & B_k \end{bmatrix} \begin{bmatrix} \delta x \\ \delta u \end{bmatrix} + \begin{bmatrix} a_k \\ b_k \end{bmatrix}^T \begin{bmatrix} \delta x \\ \delta u \end{bmatrix}, \end{aligned} \tag{6}$$

where $A_k = \partial^2 Q_k / \partial x^2$, $B_k = \partial^2 Q_k / \partial u^2$, $C_k = \partial^2 Q_k / \partial u \partial x$ are block matrices of the Hessian of Q_k and $a_k = \partial Q_k / \partial x$, $b_k = \partial Q_k / \partial u$ are the gradient vectors. The block matrices in the Hessian and gradient are computed as

$$a_x = W_k x + w_k + \left(\frac{\partial f}{\partial x} \right)^T g_{k+1} \tag{7}$$

$$b_k = R_k u + r_k + \left(\frac{\partial f}{\partial u} \right)^T g_{k+1} \tag{8}$$

$$A_k = W_k + \left(\frac{\partial f}{\partial x} \right)^T H_{k+1} \left(\frac{\partial f}{\partial x} \right) + \left(\frac{\partial^2 f}{\partial x^2} \right) \cdot g_{k+1} \tag{9}$$

$$B_k = R_k + \left(\frac{\partial f}{\partial u} \right)^T H_{k+1} \left(\frac{\partial f}{\partial u} \right) + \left(\frac{\partial^2 f}{\partial u^2} \right) \cdot g_{k+1} \tag{10}$$

$$C_k = \left(\frac{\partial f}{\partial u} \right)^T H_{k+1} \left(\frac{\partial f}{\partial x} \right) + \left(\frac{\partial^2 f}{\partial u \partial x} \right) \cdot g_{k+1}. \tag{11}$$

Note that the second derivatives of the dynamics appearing in equations (9), (10), and (11) are rank-three tensors, and that their multiplication with the vector g_{k+1} produces matrices. These tensor calculations are relatively expensive and are often omitted, resulting in the iLQR algorithm [5].

Minimizing equation (6) with respect to δu results in the following correction to the control trajectory,

$$\delta u_k = -B_k^{-1}(C_k \delta x + b_k) = -K_k \delta x - l_k, \tag{12}$$

which consists of a constant term, l_k, and a linear feedback term, $K_k \delta x$. These terms can be substituted back into equation (6) to obtain H_k and g_k in equation (5):

$$H_k = W_k + K_k^T B_k K_k - K_k^T C_k - C_k^T K_k \tag{13}$$

$$g_k = w_k + W_k x + a_k + (K_k^T B_k - C_k^T) l_k - K_k^T b_k. \tag{14}$$

Additionally, the expected change in the cost-to-go at time k can be computed:

$$\delta V_k = -l_k^T B_k l_k - b_k^T l_k. \tag{15}$$

From here, the recursion can be continued backward until $k = 1$. At that point, a forward pass is performed with the new corrected feedback control $u_k(x)$ to compute a new state trajectory x_k. These alternating backward and forward passes are then repeated until convergence to a locally optimal trajectory is achieved, as summarized in Algorithm 1.

DDP is a second-order algorithm that, like Newton's method, can achieve quadratic convergence rates [3], [4]. Also like Newton's method, some additional care must be taken in practice to ensure good convergence properties. First, a regularization term must sometimes be added to B_k in equation (12) to ensure positive-definiteness. Second, a

Algorithm 1 Differential Dynamic Programming

procedure DDP(x, u, ϵ)
  repeat
    $K, l, \delta V \leftarrow$ backward pass using (7) – (15)
    $x, u, \delta J \leftarrow$ FORWARDPASS($x, u, K, l, \delta V$)
  until $|\delta J| < \epsilon$
  return x, u
end procedure
function FORWARDPASS($x, u, K, l, \delta V$)
  $\alpha = 1$
  repeat
    for $k = 1 \ldots N$ **do**
      $x_{k+1} \leftarrow f(x_k, u_k - \alpha l_k - K_k \delta x_k)$
    end for
    $J \leftarrow$ calculate using (3)
    $\alpha \leftarrow$ reduce according to line search update
  until Wolfe conditions are satisfied [6]
  return $x, u, \delta J$
end function

line search must be performed during the forward pass of the algorithm to ensure a sufficient decrease in cost is achieved. An approximate line search using the Wolfe conditions as the termination criteria is often used [6]. These implementation details are discussed in depth in [18].

B. Unscented Transform

The Unscented Transform [19] is a method used to approximate the mean and covariance of a probability distribution that has been mapped through a nonlinear coordinate transformation. It was originally developed for use in recursive estimation algorithms, most notably the UKF [20]. The basic idea is to sample the distribution at a small number of strategically chosen points, known as *sigma points*, S_i, which are then propagated through the coordinate transformation and used to calculate a new mean and covariance.

The Unscented Transform is typically motivated by considering a probability distribution with mean, $\mu \in \mathbb{R}^n$, and covariance, $P \in \mathbf{S}^n_{++}$, that is mapped through a nonlinear coordinate transformation, $x' = f(x)$. Consider the following first-order approximation of the mean and covariance of the transformed distribution that is used in many applications (e.g., the Extended Kalman Filter):

$$\mu' = f(\mu) \tag{16}$$

$$P' = \left(\frac{\partial f}{\partial x}\right) P \left(\frac{\partial f}{\partial x}\right)^T. \tag{17}$$

If the Cholesky factorization $P = LL^T$, where L is the unique lower triangular "square root" of P, is substituted into equation (17),

$$L'L'^T = \left(\frac{\partial f}{\partial x}\right) LL^T \left(\frac{\partial f}{\partial x}\right)^T, \tag{18}$$

a transformation equation for L can be written:

$$L' = \left(\frac{\partial f}{\partial x}\right) L. \tag{19}$$

Equation (19) can be interpreted as propagating the column vectors of L through the linearized mapping $\frac{\partial f}{\partial x}$. It is then natural to consider simply propagating the columns of L directly through the full nonlinear transformation $f(x)$ without linearizing. This basic intuition leads directly to Algorithm 2, where subscripts i indicate columns of a matrix and β is a scale factor chosen by the user.

Algorithm 2 Unscented Transform

procedure UT(μ, P, f)
  $L = \text{chol}(P)$
  for $i \leftarrow 1 \ldots n$ **do**
    $S_i = f(\mu + \beta L_i)$
    $S_{n+i} = f(\mu - \beta L_i)$
  end for
  $\mu' \leftarrow 0$
  $P' \leftarrow 0$
  for $i \leftarrow 1 \ldots 2n$ **do**
    $\mu' \leftarrow \mu' + \frac{1}{2n} S_i$
  end for
  for $i \leftarrow 1 \ldots 2n$ **do**
    $P' \leftarrow P' + \frac{1}{2\beta^2}(S_i - \mu')(S_i - \mu')^T$
  end for
  return μ', P'
end procedure

There are many variations of the Unscented Transform with a variety of weighting and sampling schemes. The simple version presented in this section has been shown to produce mean and covariance estimates that capture first and second order effects of the coordinate transformation [20].

III. Unscented Dynamic Programming

The main idea underlying the proposed UDP algorithm is to replace the gradient and Hessian calculations in equations (7)–(11) with approximations computed from a set of sample points. The choice of samples is motivated by analyzing the transformation of the Hessian in equation (9),

$$H^- = \left(\frac{\partial f}{\partial x}\right)^T H_{k+1} \left(\frac{\partial f}{\partial x}\right), \tag{20}$$

where we use a superscript "$-$" to denote backward propagation through the dynamics. Taking the inverse of both sides gives:

$$(H^-)^{-1} = \left(\frac{\partial f}{\partial x}\right)^{-1} H_{k+1}^{-1} \left(\frac{\partial f}{\partial x}\right)^{-T}. \tag{21}$$

If the Cholesky factorization $H_{k+1}^{-1} = LL^T$ is now substituted into (21), we arrive at the following transformation equation for L,

$$L^- = \left(\frac{\partial f}{\partial x}\right)^{-1} L, \tag{22}$$

where the columns of L are propagated through the dynamics backward in time from t_{k+1} to t_k. As in Section II-B, we are now in a position to eliminate the linearization and propagate

the columns of L directly through the backwards system dynamics:

$$x_k = f^-(x_{k+1}, u_k). \tag{23}$$

This 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.

To compute the full Hessian matrix in equation (6), a set of $2(n+m)$ sample points is generated using the following Cholesky factorization:

$$L = \text{chol}\left(\begin{bmatrix} H_{k+1} & 0 \\ 0 & R_k \end{bmatrix}^{-1}\right). \tag{24}$$

As in Algorithm 2, the sample points are constructed from the columns of $L = [L_1, \ldots, L_{n+m}]$,

$$\begin{bmatrix} x_i^s \\ u_i^s \end{bmatrix} = \begin{bmatrix} x_{k+1} \\ u_k \end{bmatrix} + \beta L_i \tag{25}$$

$$\begin{bmatrix} x_{i+n+m}^s \\ u_{i+n+m}^s \end{bmatrix} = \begin{bmatrix} x_{k+1} \\ u_k \end{bmatrix} - \beta L_i, \tag{26}$$

where each column and its negative is scaled and added to the vector $\begin{bmatrix} x_{k+1}^T & u_k^T \end{bmatrix}^T$. The samples are then propagated backward through the dynamics:

$$x_i^{s-} = f^-(x_i^s, u_i^s). \tag{27}$$

Using the back-propagated sample points, the Hessian in equation (6) becomes

$$\begin{bmatrix} A_k & C_k^T \\ C_k & B_k \end{bmatrix} = M^{-1} + \begin{bmatrix} W_k & 0 \\ 0 & 0 \end{bmatrix}, \tag{28}$$

where

$$M = \frac{1}{2\beta^2} \sum_{i=1}^{2(n+m)} \left(\begin{bmatrix} x_i^{s-} \\ u_i^s \end{bmatrix} - \begin{bmatrix} x_k \\ u_k \end{bmatrix}\right)\left(\begin{bmatrix} x_i^{s-} \\ u_i^s \end{bmatrix} - \begin{bmatrix} x_k \\ u_k \end{bmatrix}\right)^T \tag{29}$$

and the one-step cost, W_k, has been added to the A_k block.

The last ingredient needed to complete the DDP recursion of Section II-A is the gradient vector. This can be computed without any additional evaluations of the dynamics function by projecting g_{k+1} onto the sample points used to calculate the Hessian:

$$g_i^s = g_{k+1}^T x_i^s. \tag{30}$$

An $(n+m) \times (n+m)$ linear system is then solved,

$$D \begin{bmatrix} a^- \\ b^- \end{bmatrix} = d, \tag{31}$$

where the columns of D are differences of pairs of sample vectors,

$$D = \begin{bmatrix} x_1^{s-} - x_{n+m+1}^{s-} & \cdots & x_{n+m}^{s-} - x_{2(n+m)}^{s-} \\ u_1^s - u_{n+m+1}^s & \cdots & u_{n+m}^s - u_{2(n+m)}^s \end{bmatrix}, \tag{32}$$

and the entries of d are differences of pairs of elements from g^s:

$$d = \begin{bmatrix} g_1^s - g_{n+m+1}^s \\ \vdots \\ g_{n+m}^s - g_{2(n+m)}^s \end{bmatrix}. \tag{33}$$

This procedure is equivalent to calculating a centered finite-difference approximation of the terms

$$a^- = \left(\frac{\partial f}{\partial x}\right)^T g_{k+1} \tag{34}$$

and

$$b^- = \left(\frac{\partial f}{\partial u}\right)^T g_{k+1} \tag{35}$$

in equations (7) and (8). Finally, the one-step costs for step k are added:

$$a_k = a^- + W_k x + w_k \tag{36}$$

$$b_k = b^- + R_k u + r_k. \tag{37}$$

The sample-based Hessian and gradient calculations detailed in this section can be readily inserted into equations (12)–(15), creating a derivative-free variant of the DDP algorithm. The number of sample points and dynamics function evaluations per iteration are the same as would be required to calculate centered-difference first derivatives of the dynamics for use in the standard iLQR algorithm. As has been demonstrated in applications of the UKF, however, the Unscented Transform often produces better results than naive finite difference approaches.

IV. Examples

Three numerical examples are now presented to demonstrate the performance of the UDP algorithm. The new algorithm is compared to the standard DDP and iLQR algorithms with finite-difference derivatives. All implementation details (line search, regularization, etc.) are identical except for the substitution of the Unscented Transform procedure in lieu of equations (7)–(11) in the UDP implementation.

A. Pendulum

In the first test case, a simple pendulum with an input torque is considered. The goal is to swing the pendulum from its downward stable equilibrium at $\theta = 0$ to the upward unstable equilibrium at $\theta = \pi$.

The following cost function is used,

$$J = \frac{1}{2}(x_N - x_g)^T W_N (x_N - x_g) + \sum_{k=1}^{N-1} \frac{1}{2}(x_k - x_g)^T W (x_k - x_g) + \frac{1}{2} u_k^T R u_k \tag{38}$$

where

$$x_g = \begin{bmatrix} \theta_g \\ \dot{\theta}_g \end{bmatrix} = \begin{bmatrix} \pi \\ 0 \end{bmatrix} \tag{39}$$

$$W_N = \begin{bmatrix} 30 & 0 \\ 0 & 30 \end{bmatrix} \tag{40}$$

$$W = \begin{bmatrix} 0.3 & 0 \\ 0 & 0.3 \end{bmatrix} \tag{41}$$

and $R = 0.3$. The classic fourth-order Runge-Kutta method is used to discretize the dynamics with a step size of 0.1 and the horizon is set to $N = 50$ steps. The algorithms are initialized with all states and control inputs set to zero.

TABLE I

PENDULUM SWING-UP PERFORMANCE

Algorithm	Cost	Iterations	Time (s)
UDP	38.73	57	15.2
DDP	38.64	34	16.6
iLQR	38.65	79	19.5

UDP, DDP, and iLQR converge to essentially the same trajectory. There are, however, marked differences in their convergence behavior, as can be seen in Table I and Figure 1. The DDP algorithm takes the largest steps and converges with the fewest number of iterations. However, the computational cost of each iteration is much higher than the other two algorithms. The UDP algorithm approaches the convergence rate of DDP while requiring approximately the same amount of computation per iteration as iLQR.

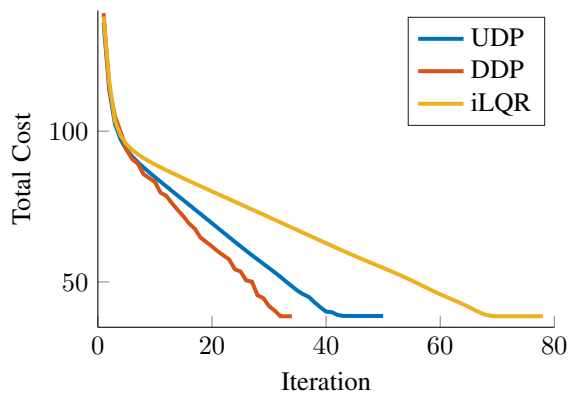

Fig. 1. Pendulum swing-up cost

B. Cart Pole

The second example deals with the cart pole system shown in Figure 2. The system has two degrees of freedom—translation of the cart and rotation of the pendulum—but only one control input consisting of a force applied to the cart. The goal is to swing the pendulum from its downward

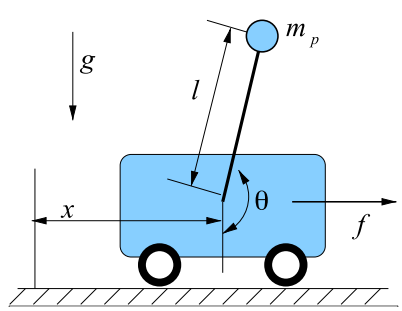

Fig. 2. Cart pole

equilibrium to its upward equilibrium.

A cost function of the same form as equation (38) is used, this time with the goal state $x_g = \begin{bmatrix} 0 & \pi & 0 & 0 \end{bmatrix}^T$ and the weighting matrices $W_N = 10^3\, I_{4\times4}$, $W = 0.1\, I_{4\times4}$, and $R = 0.01$, where $I_{4\times4}$ is the 4×4 identity matrix. As before, the algorithms are initialized with zeros and they are run with a step size of 0.1 and a horizon of $N = 50$.

TABLE II

CART POLE SWING-UP PERFORMANCE

Algorithm	Cost	Iterations	Time (s)
UDP	131.78	183	78.4
DDP	131.76	67	173.1
iLQR	135.40	54	26.6

Unlike in the pendulum example, the UDP and DDP algorithms converge to a slightly different trajectory than iLQR. The cost reduction at each iteration for UDP and iLQR is plotted in Figure 3 (DDP is omitted for visual clarity). The two algorithms take similar steps when they begin, with iLQR achieving a cost of 131.40 by iteration 54 compared to 131.25 for UDP. However, iLQR stops making significant progress at that point. UDP, on the other hand, continues converging until it closely matches the cost achieved by the full DDP algorithm.

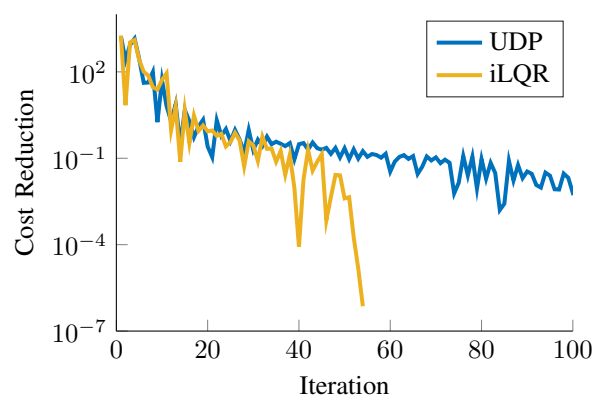

Fig. 3. Cart pole swing-up cost reduction

C. Airplane

The final example deals with the small airplane shown in Figure 4. A dynamics model for this airplane was constructed

Fig. 4. E-flite AS3Xtra model aircraft

based on a combination of flat plate aerodynamic theory and parameter fitting from motion capture experiments. The 12-dimensional state vector consists of the aircraft position,

attitude, velocity, and angular velocity. Attitude is represented using modified Rodrigues parameters which, unlike Euler angles, allow rotations of up to $360°$ before becoming singular [21]. The system has four control inputs consisting of the throttle setting and aileron, elevator, and rudder angles. Control responses are fast enough that actuator dynamics can be ignored.

The goal in this example is for the airplane to perform a $180°$ barrel roll within a 6 meter physical distance while minimizing control effort. A cost function of the same form as equation (38) is again used, this time with a running cost on the control effort and only a terminal cost on the state ($W = 0$). The goal state is inverted level flight. The algorithms are initialized with a trajectory consisting of a straight-and-level flight ending at the desired final position, but in an upright attitude.

UDP, DDP, and iLQR converge to essentially identical final trajectories, depicted in Figure 5. Of the three algorithms, UDP converges fastest in terms of both iteration count and computation time, as shown in Table III.

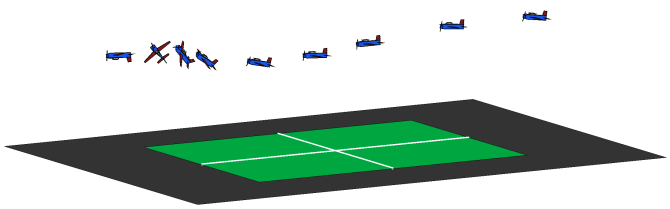

Fig. 5. Barrel roll trajectory

V. Discussion

We have presented a derivative-free variant of the DDP algorithm that uses the Unscented Transform to propagate quadratic cost-to-go estimates backwards along the trajectory. The new UDP algorithm has the same per-iteration computational requirements as iLQR, but due to the ability of the Unscented Transform to capture second-order dynamical information, it offers performance closer to that of the full DDP algorithm. A MATLAB implementation of the algorithm is available at `http://bit.ly/unscented-dp`.

Several directions for future research remain. First, a thorough investigation of the effect of the parameter β used to scale the sigma points is needed. Ultimately, a scheme for choosing β based on the system dynamics and cost function would greatly enhance the utility of the algorithm. Second, while the UDP algorithm presented here does not require derivatives of the dynamics, it does require derivatives of the cost function. Extending the sampling scheme to handle non-quadratic cost functions could avoid further derivative calculations. Lastly, the algorithms discussed in this paper do not account for input constraints. By solving quadratic programming problems at each time step, rather than computing the unconstrained LQR policy, these constraints can be handled efficiently [22], [23].

TABLE III

Airplane barrel roll performance

Algorithm	Cost	Iterations	Time (s)
UDP	37.80	30	11.6
DDP	37.80	31	100.2
iLQR	37.81	36	12.1

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

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

[4] 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.

[5] 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.

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

[7] E. Todorov and Y. Tassa, "Iterative Local Dynamic Programming," in *Adaptive Dynamic Programming and Reinforcement Learning*, 2009.

[8] L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mischenko, *The Mathematical Theory of Optimal Processes*. Interscience, 1962.

[9] C. R. Hargraves and S. W. Paris, "Direct Trajectory Optimization Using Nonlinear Programming and Collocation," *J. Guidance*, vol. 10, no. 4, pp. 338–342, 1987.

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

[11] 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.

[12] M. Posa, C. Cantu, and R. Tedrake, "A direct method for trajectory optimization of rigid bodies through contact," *International Journal of Robotics Research*, vol. 33, pp. 69–81, Jan. 2014.

[13] 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), IEEE, 2016.

[14] T. Erez, Y. Tassa, and E. Todorov, "Infinite-Horizon Model Predictive Control for Periodic Tasks with Contacts," in *Robotics Science and Systems VII*, pp. 73–80, 2012.

[15] 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.

[16] 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, 2014.

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

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

[19] J. Uhlmann, *Dynamic Map Building and Localization: New Theoretical Foundations*. PhD thesis, University of Oxford, 1995.

[20] E. A. Wan and R. V. D. Merwe, "The unscented Kalman filter for nonlinear estimation," in *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, pp. 153–158, 2000.

[21] H. Schaub and J. L. Junkins, *Analytical Mechanics of Space Systems*. Reston, VA: AIAA, 2nd ed., Oct 2009.

[22] 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.

[23] S. Kuindersma, F. Permenter, and R. Tedrake, "An Efficiently Solvable Quadratic Program for Stabilizing Dynamic Locomotion," in *Proceedings of the International Conference on Robotics and Automation (ICRA)*, (Hong Kong, China), pp. 2589–2594, IEEE, May 2014.