Control-Limited Differential Dynamic Programming

Paper-ID [148]

***Abstract*—We describe a generalization of the Differential Dynamic Programming trajectory optimization algorithm which accommodates box inequality constraints on the controls, without significantly sacrificing convergence quality or computational effort. To this effect we describe an efficient Quadratic Programming sub-algorithm which benefits from warm starts and provides explicit Hessian factors. We demonstrate our algorithm on three simulated problems, including a 28-DoF grasping problem. Simple cost terms were sufficient to generate highly dexterous and agile grasping behaviors. A movie of the grasping results can be found here goo.gl/GlM8h**

I. Introduction

Constraints on the control signal applied to an actuator are invariably present in robotic systems. Signal clamping alleviates the danger of frying one's robot by exceeding voltage limits, but will not help in finding the best signal given those limits.

Optimal control algorithms render such control constraints as inequality-constrained optimization problems, which are always harder than unconstrained ones. Classic Differential Dynamic Programming (DDP) – a trajectory-optimizer – is efficient precisely because it parameterizes unconstrained controls. Below we describe a generalization of DDP which accommodates box inequality constraints on the controls, without significantly sacrificing convergence quality or computational effort.

In section II we provide background on DDP and box constraints. In III we motivate and describe our proposed algorithm. In IV we describe experimental results obtained in simulation.

II. Background

Trajectory optimization is the process of finding a state-control sequence which locally minimizes a given cost function. *Shooting methods* – which trace their ancestry to the two-point boundary-value problem of the venerable Maximum Principle [1] – are an important sub-class of trajectory optimization methods. Unlike so-called *direct methods* which explicitly represent the state, these methods parameterize only the controls, and obtain the states from forward integration (hence "shooting"). Given the state-control trajectory, Dynamic Programming is used to find an improved control sequence. Because states are never explicitly represented in the optimization space, these methods are also known as *indirect* [2].

Because the dynamics are folded into the optimization, state-control trajectories are always feasible and "dynamic constraints" unnecessary. If additionally the controls are unconstrained, so is the optimization search-space, and shooting methods can enjoy the benefits of unconstrained optimization.

DDP is a second-order shooting method [3] which under mild assumptions admits quadratic convergence for any system with smooth dynamics [4]. It has been shown to posses convergence properties similar to or slightly better than Newton's method performed on the entire control sequence [5].

Classic DDP requires second order derivatives of the dynamics, which are usually the most expensive part of the computation. If these are ignored one obtains a Gauss-Newton approximation known as iterative-LQG [6], which is similar to Riccati iterations, but accounting for the regularization and line-search required to handle the nonlinearity. See section II-B below.

Control constraints can be naively imposed by applying a sigmoidal squashing function, and also by simple clamping. A more principled approach is to solve a Quadratic Program (QP). In the following we compare these three approaches and demonstrate that the last approach is superior.

A. Shooting methods

The discrete-time[1] dynamics

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

describe the evolution from time i to $i+1$ of the state $\mathbf{x} \in \mathbb{R}^n$, given the control $\mathbf{u} \in \mathbb{R}^m$. A trajectory is a sequence of states $\mathbf{X} \equiv \{\mathbf{x}_0, \mathbf{x}_1 \ldots, \mathbf{x}_N\}$ and controls $\mathbf{U} \equiv \{\mathbf{u}_0, \mathbf{u}_1 \ldots, \mathbf{u}_{N-1}\}$. The *total cost* J_0 is the sum of running costs ℓ and final cost ℓ_f, incurred when starting from $\mathbf{x}_0$ and applying $\mathbf{U}$ until the horizon N is reached:

$$J_0(\mathbf{x}_0, \mathbf{U}) = \sum_{i=0}^{N-1} \ell(\mathbf{x}_i, \mathbf{u}_i) + \ell_f(\mathbf{x}_N),$$

where the $\mathbf{x}_i$ for $i > 0$ are given by (1). The solution of the optimal control problem is the minimizing control sequence

$$\mathbf{U}^* \equiv \operatorname*{argmin}_{\mathbf{U}} J_0(\mathbf{x}_0, \mathbf{U}).$$

Letting $\mathbf{U}_i \equiv \{\mathbf{u}_i, \mathbf{u}_{i+1} \ldots, \mathbf{u}_{N-1}\}$ be the tail of the control sequence, we define the *cost-to-go* J_i as the partial sum of costs from i to N:

$$J_i(\mathbf{x}_i, \mathbf{U}_i) = \sum_{j=i}^{N-1} \ell(\mathbf{x}_j, \mathbf{u}_j) + \ell_f(\mathbf{x}_N).$$

The *Value* at time i is the optimal cost-to-go starting at $\mathbf{x}$:

$$V(\mathbf{x}, i) \equiv \min_{\mathbf{U}_i} J_i(\mathbf{x}, \mathbf{U}_i).$$

Setting $V(\mathbf{x}, N) \equiv \ell_f(\mathbf{x}_N)$, the Dynamic Programming Principle reduces the minimization over a sequence of controls

[1] We restrict ourselves to the discrete, but the continuous-time limit is straightforward.

$\mathbf{U}_i$, to a sequence of minimizations over a single control, proceeding backwards in time:

$$V(\mathbf{x}, i) = \min_{\mathbf{u}}[\ell(\mathbf{x}, \mathbf{u}) + V(\mathbf{f}(\mathbf{x}, \mathbf{u}), i{+}1)] \tag{2}$$

Shooting methods involve iterating a *forward pass* or *rollout* which integrates (1), followed by a *backward pass* which approximates a local solution to (2).

B. Differential Dynamic Programming

Let $Q(\delta\mathbf{x}, \delta\mathbf{u})$ be the change in the argument of the minimum in (2) as a function of small perturbations of the i-th nominal $(\mathbf{x}, \mathbf{u})$ pair:

$$\begin{aligned} Q(\delta\mathbf{x}, \delta\mathbf{u}) &= \ell(\mathbf{x} + \delta\mathbf{x}, \mathbf{u} + \delta\mathbf{u}, i) - \ell(\mathbf{x}, \mathbf{u}, i) \\ &\quad + V(\mathbf{f}(\mathbf{x} + \delta\mathbf{x}, \mathbf{u} + \delta\mathbf{u}), i{+}1) - V(\mathbf{f}(\mathbf{x}, \mathbf{u}), i{+}1) \end{aligned} \tag{3}$$

and expand to second order

$$\approx \frac{1}{2}\begin{bmatrix} 1 \\ \delta\mathbf{x} \\ \delta\mathbf{u} \end{bmatrix}^\mathsf{T} \begin{bmatrix} 0 & Q_\mathbf{x}^\mathsf{T} & Q_\mathbf{u}^\mathsf{T} \\ Q_\mathbf{x} & Q_{\mathbf{xx}} & Q_{\mathbf{xu}} \\ Q_\mathbf{u} & Q_{\mathbf{ux}} & Q_{\mathbf{uu}} \end{bmatrix} \begin{bmatrix} 1 \\ \delta\mathbf{x} \\ \delta\mathbf{u} \end{bmatrix} \tag{4}$$

The Q-function is the discrete-time analogue of the Hamiltonian and is sometimes known as the *pseudo-Hamiltonian*. Examining (3) we see that the expansion coefficients are[2]

$$Q_\mathbf{x} = \ell_\mathbf{x} + \mathbf{f}_\mathbf{x}^\mathsf{T} V'_\mathbf{x} \tag{5a}$$
$$Q_\mathbf{u} = \ell_\mathbf{u} + \mathbf{f}_\mathbf{u}^\mathsf{T} V'_\mathbf{x} \tag{5b}$$
$$Q_{\mathbf{xx}} = \ell_{\mathbf{xx}} + \mathbf{f}_\mathbf{x}^\mathsf{T} V'_{\mathbf{xx}} \mathbf{f}_\mathbf{x} + V'_\mathbf{x} \cdot \mathbf{f}_{\mathbf{xx}} \tag{5c}$$
$$Q_{\mathbf{uu}} = \ell_{\mathbf{uu}} + \mathbf{f}_\mathbf{u}^\mathsf{T} V'_{\mathbf{xx}} \mathbf{f}_\mathbf{u} + V'_\mathbf{x} \cdot \mathbf{f}_{\mathbf{uu}} \tag{5d}$$
$$Q_{\mathbf{ux}} = \ell_{\mathbf{ux}} + \mathbf{f}_\mathbf{u}^\mathsf{T} V'_{\mathbf{xx}} \mathbf{f}_\mathbf{x} + V'_\mathbf{x} \cdot \mathbf{f}_{\mathbf{ux}}. \tag{5e}$$

Minimizing (4) WRT $\delta\mathbf{u}$ we obtain

$$\delta\mathbf{u}^* = \operatorname*{argmin}_{\delta\mathbf{u}} Q(\delta\mathbf{x}, \delta\mathbf{u}) = -Q_{\mathbf{uu}}^{-1}(Q_\mathbf{u} + Q_{\mathbf{ux}}\delta\mathbf{x}), \tag{6}$$

giving us a locally-linear feedback policy of the form

$$\delta\mathbf{u}^*(\delta\mathbf{x}) = \mathbf{l} + \mathbf{L} \cdot \delta\mathbf{x} \tag{7}$$

with $\mathbf{l} \in \mathbb{R}^m$ the open-loop modification and $\mathbf{L} \in \mathbb{R}^{m \times n}$ the feedback gain. To ensure positive-definiteness, regularization is added

$$\underset{\delta\mathbf{u}}{\text{minimize}}\, Q(\delta\mathbf{x}, \delta\mathbf{u}) + \mu\tfrac{\|\delta\mathbf{u}\|^2}{2}, \tag{8}$$

amounting to a Levenberg-Marquardt Hessian modification $\widetilde{Q}_{\mathbf{uu}} = Q_{\mathbf{uu}} + \mu \cdot \mathbf{I}_m$. The open-loop and feedback terms are

$$\mathbf{l} = -\widetilde{Q}_{\mathbf{uu}}^{-1} \cdot Q_\mathbf{u} \tag{9a}$$
$$\mathbf{L} = -\widetilde{Q}_{\mathbf{uu}}^{-1} \cdot Q_{\mathbf{ux}}, \tag{9b}$$

Plugging the policy (7, 9) back into (4), we obtain a quadratic model of $V(i)$:

$$\Delta V(i) = +\tfrac{1}{2}\mathbf{l}^\mathsf{T} Q_{\mathbf{uu}} \mathbf{l} + \mathbf{l}^\mathsf{T} Q_\mathbf{u} \tag{10a}$$
$$V_\mathbf{x}(i) = Q_\mathbf{x} + \mathbf{L}^\mathsf{T} Q_{\mathbf{uu}} \mathbf{l} + \mathbf{L}^\mathsf{T} Q_\mathbf{u} + Q_{\mathbf{ux}}^\mathsf{T} \mathbf{l} \tag{10b}$$
$$V_{\mathbf{xx}}(i) = Q_{\mathbf{xx}} + \mathbf{L}^\mathsf{T} Q_{\mathbf{uu}} \mathbf{L} + \mathbf{L}^\mathsf{T} Q_{\mathbf{ux}} + Q_{\mathbf{ux}}^\mathsf{T} \mathbf{L}. \tag{10c}$$

[2]Here and elsewhere in the paper we drop the index i and use primes to denote the next time-step $V' \equiv V(i{+}1)$.

The backward pass is initialized with the final cost at $\mathbf{x}_N$

$$V(N) = \ell_f(\mathbf{x}_N) \tag{11a}$$
$$V_\mathbf{x}(N) = \ell_{f_\mathbf{x}}(\mathbf{x}_N) \tag{11b}$$
$$V_{\mathbf{xx}}(N) = \ell_{f_{\mathbf{xx}}}(\mathbf{x}_N), \tag{11c}$$

and then recursively computes the linear controllers (9) and the quadratic expansion (10). Once it is completed, a forward pass computes a new trajectory:

$$\hat{\mathbf{x}}_0 = \mathbf{x}_0 \tag{12a}$$
$$\hat{\mathbf{u}}_i = \mathbf{u}_i + \alpha\mathbf{l}_i + \mathbf{L}_i(\hat{\mathbf{x}}_i - \mathbf{x}_i) \tag{12b}$$
$$\hat{\mathbf{x}}_{i+1} = \mathbf{f}(\hat{\mathbf{x}}_i, \hat{\mathbf{u}}_i), \tag{12c}$$

where α is a line search parameter (see below).

The last terms in (5c, 5d, 5e), denote contraction with a tensor which scales cubically with the dimension and is often expensive to represent and compute. Ignoring these terms amounts to a Gauss-Newton approximation of the Hessian, and is known as *iterative Linear Quadratic Gaussian* or iterative-LQG optimization [6]. Though some of the convergence speed is lost, the speedup gain from discarding the cubic terms often more than compensates.

In addition to using only first derivatives, iterative-LQG includes several small improvements to classic DDP with regards to the Value update (10), the schedule of μ in (8), alternative regularization, and better improvement prediction using (10a). Details are provided in [7].

Algorithm I $\quad \mathbf{U}^* \longleftarrow \text{iLQG}(\mathbf{f}, \ell, \ell_f, \mathbf{x}_0, \mathbf{U})$

Integrate $\mathbf{U}$ to get the initial $(\mathbf{x}_i, \mathbf{u}_i)$ trajectory.
Repeat until convergence:

1) *Derivatives:* Compute the derivatives of ℓ and $\mathbf{f}$ in (5).
2) *Backward Pass:* Initialize with (11). Iterate (5, 9, 10) for decreasing $i = N{-}1, \ldots 0$. If $\widetilde{Q}_{\mathbf{uu}}$ is non-PD, increase μ and restart the backward pass. If successful, decrease μ.
3) *Convergence:* If a termination condition is met, return $\mathbf{U}$.
4) *Forward Pass:* Set $\alpha = 1$. Iterate (12) to get a new nominal sequence. If the cost was sufficiently reduced, accept $(\hat{\mathbf{x}}_i, \hat{\mathbf{u}}_i)$, otherwise decrease α and restart the forward pass.

C. Box Constraints

Rather than treat general inequality constraints we will focus on the box

$$\underline{\mathbf{b}} \leqslant \mathbf{u} \leqslant \bar{\mathbf{b}}$$

with elementwise inequality and $\underline{\mathbf{b}}, \bar{\mathbf{b}}$ the respective lower and upper bounds. Besides lending itself to our proposed solution, the box constraint accurately describes nearly any set of standard mechanical actuators.

One can consider several methods for imposing such control constraints. Below we outline two such methods, which are easy-to-implement heuristics but not do not work so well. In the following section we develop a more principled method which is the main contribution of this paper.

1) Naïve Clamping: A first attempt to enforce box constraints is to clamp the controls in the forward-pass. Letting double square brackets $[\![\cdot]\!]_{\mathbf{b}}$ denote the element-wise clamping, or projection operator

$$[\![\mathbf{u}]\!]_{\mathbf{b}} = \min(\max(\mathbf{u}, \underline{\mathbf{b}}), \bar{\mathbf{b}}),$$

the forward-pass is modified by replacing (12b) with

$$\hat{\mathbf{u}}_i = [\![\mathbf{u}_i + \alpha \mathbf{l}_i + \mathbf{L}_i(\hat{\mathbf{x}}_i - \mathbf{x}_i)]\!]_{\mathbf{b}}. \tag{13}$$

When attempted as such, this approach fails. Since the backpass is ignorant of the constraints, it repeatedly tries to violate them and eventually generates control steps which are not legitimate descent directions. Post-hoc clamping can be introduced to the control modification $\mathbf{l}$ in (9a)

$$\mathbf{l} \leftarrow [\![\mathbf{l} + \mathbf{u}]\!]_{\mathbf{b}} - \mathbf{u},$$

but this does not help much. It might also seem sensible that the rows of $\mathbf{L}$ corresponding to clamped controls should be set to 0, since the feedback is inactive in these dimensions. All this feels correct, it is not well-motivated, and besides does not significantly improve convergence.

2) Squashing Functions: Another way to enforce box constraints is to introduce a sigmoidal squashing function $\mathbf{s}(\mathbf{u})$ in the dynamics

$$\mathbf{x}_{i+1} = \mathbf{f}(\mathbf{x}_i, \mathbf{s}(\mathbf{u}_i)) \tag{14}$$

where $\mathbf{s}()$ is an elementwise sigmoid with the vector limits

$$\lim_{\mathbf{u}\to-\infty} \mathbf{s}(\mathbf{u}) = \underline{\mathbf{b}} \qquad \lim_{\mathbf{u}\to\infty} \mathbf{s}(\mathbf{u}) = \bar{\mathbf{b}}. \tag{15}$$

For example

$$\mathbf{s}(\mathbf{u}) = \frac{\bar{\mathbf{b}} - \underline{\mathbf{b}}}{2} \tanh(\mathbf{u}) + \frac{\bar{\mathbf{b}} + \underline{\mathbf{b}}}{2} \tag{16}$$

is such a function. If the control-cost term remains a function of the original $\mathbf{u}$, that amounts to changing the problem to be solved. If the input to the control-cost term is also the squashed $\mathbf{s}(\mathbf{u})$, the control problem remains the same, but $\mathbf{u}$ will tend to $\pm\infty$. One way to prevent this is to have control cost terms for both $\mathbf{u}$ and $\mathbf{s}(\mathbf{u})$, see section IV-B. While this trick works in the sense that the optimizer does not get stuck, it does not mitigate the sigmoidal non-linearity. Since the backward pass uses a locally linear approximation of the dynamics (quadratic in the case of full DDP), significant higher order terms always have a detrimental effect on convergence.

III. Proposed Algorithm

The Quadratic Program (QP) is a well understood problem with many methods of solution [8] which together form the backbone of the Sequential-QP approach to nonlinear optimization. However, the problem at hand is different from generic QPs in two important respects.

First, because we are solving the problem sequentially (backwards) in time, the current solution will often be similar to the subsequent one. We would therefore prefer an algorithm that can enjoy warm starts.

Second, the problem we wish to solve is small. The problem size m corresponds to the number of actuators and in reality does not exceeds several dozen, except for very specialized robots. Although DDP searches in the space of control trajectories $\mathbf{U} \in \mathbb{R}^{m\times N}$, we solve the m-dimensional problem N times, not a single problem of size mN. The difference is made stark when considering N Hessians of size $m\times m$ rather than a large $Nm\times Nm$ matrix, as in the direct representation. Since factorization complexity is cubic in the dimension, the respective complexities are $O(Nm^3)$ and $O(N^3m^3)$.

The warm-start requirement rules out some classes of algorithms, for example interior-point methods. Since these methods glide smoothly to the solution from the interior, they do not benefit from being initialized at the boundary. Active-set methods do traverse the boundary and can be warm-started, but need to explicitly account for constraint activation, separately for each constraint.

The Projected-Newton class of algorithms were developed for problems with simple constraints, where the projection operator is trivial – like clamping in the case of the box. Their key feature is the projected line-search, whereby the search-point is continuously clamped, allowing multiple constraints to form and break in each iteration. In [9], Bertsekas analyses these methods and proves convergence for a large class of approximate Hessians. In the following we describe a special case thereof, which uses the exact Hessian at all times. Its key feature, which we prove below, is that if the initial point has the same active constraint set as the optimum, the solution will be reached in a single iteration.

A. Box-Constrained Quadratic Programs

We wish to solve the constrained version of Eq. (8):

$$\underset{\delta\mathbf{u}}{\text{minimize}} \quad Q(\delta\mathbf{x}, \delta\mathbf{u}) + \mu\tfrac{\|\delta\mathbf{u}\|^2}{2} \tag{17a}$$

$$\text{subject to} \quad \underline{\mathbf{b}} \leqslant \mathbf{u} + \delta\mathbf{u} \leqslant \bar{\mathbf{b}} \tag{17b}$$

In generic[3] form, for $\mathbf{x} \in \mathbb{R}^n$ and a positive-definite Hessian $\mathbf{H} \succ 0$, the box-constrained QP is

$$\underset{\mathbf{x}}{\text{minimize}} \quad f(\mathbf{x}) = \mathbf{x}^\mathsf{T}\mathbf{q} + \tfrac{1}{2}\mathbf{x}^\mathsf{T}\mathbf{H}\mathbf{x} \tag{18a}$$

$$\text{subject to} \quad \underline{\mathbf{b}} \leqslant \mathbf{x} \leqslant \bar{\mathbf{b}} \tag{18b}$$

B. Projected-Newton Solution

The algorithm proceeds by iteratively identifying the active constraints, and then performing a projected Newton step using the reduced Hessian in the free sub-space. Begin at some feasible initial guess $\mathbf{x} = [\![\mathbf{x}]\!]_{\mathbf{b}}$ and define the gradient $\mathbf{g} = \nabla_{\mathbf{x}} f = \mathbf{q} + \mathbf{H}\mathbf{x}$. The complimentary sets of *clamped* and *free* indices c and f are

$$\mathsf{c}(\mathbf{x}) = \left\{ j \in 1\ldots n \;\middle|\; \begin{array}{c} \mathbf{x}_j = \underline{\mathbf{b}}_j\,, \quad \mathbf{g}_j > 0 \\ \text{or} \\ \mathbf{x}_j = \bar{\mathbf{b}}_j\,, \quad \mathbf{g}_j < 0 \end{array} \right\} \tag{19a}$$

$$\mathsf{f}(\mathbf{x}) = \left\{ j \in 1\ldots n \;\middle|\; j \notin \mathsf{c} \right\} \tag{19b}$$

[3] In section III we use $\mathbf{x}$ to denote a generic variable rather than the state.

Note that indices are clamped only when the descent direction at the boundary points outward. Repartitioning the problem[4]

$$\mathbf{x} \leftarrow \begin{bmatrix} \mathbf{x}_\mathsf{f} \\ \mathbf{x}_\mathsf{c} \end{bmatrix}, \quad \mathbf{q} \leftarrow \begin{bmatrix} \mathbf{q}_\mathsf{f} \\ \mathbf{q}_\mathsf{c} \end{bmatrix}, \quad \mathbf{H} \leftarrow \begin{bmatrix} \mathbf{H}_{\mathsf{ff}} & \mathbf{H}_{\mathsf{fc}} \\ \mathbf{H}_{\mathsf{cf}} & \mathbf{H}_{\mathsf{cc}} \end{bmatrix},$$

the gradient in the free subspace is

$$\mathbf{g}_\mathsf{f} = \nabla_{\mathbf{x}_\mathsf{f}} f = \mathbf{q}_\mathsf{f} + \mathbf{H}_{\mathsf{ff}}\mathbf{x}_\mathsf{f} + \mathbf{H}_{\mathsf{fc}}\mathbf{x}_\mathsf{c}, \tag{20a}$$

and the Newton step in the free subspace is

$$\Delta\mathbf{x}_\mathsf{f} = -\mathbf{H}_{\mathsf{ff}}^{-1}\mathbf{g}_\mathsf{f} = -\mathbf{H}_{\mathsf{ff}}^{-1}\left(\mathbf{q}_\mathsf{f} + \mathbf{H}_{\mathsf{fc}}\mathbf{x}_\mathsf{c}\right) - \mathbf{x}_\mathsf{f}. \tag{20b}$$

The full step is therefore

$$\Delta\mathbf{x} = \begin{bmatrix} \Delta\mathbf{x}_\mathsf{f} \\ \mathbf{0}_\mathsf{c} \end{bmatrix}. \tag{20c}$$

The projected Newton candidate point $\hat{\mathbf{x}}$ for a line-search parameter α is

$$\hat{\mathbf{x}}(\alpha) = [\![\mathbf{x} + \alpha\Delta\mathbf{x}]\!]_\mathbf{b}. \tag{21a}$$

A backtracking line-search reduces α until the Armijo condition [10] is satisfied

$$\frac{f(\mathbf{x}) - f(\hat{\mathbf{x}}(\alpha))}{\mathbf{g}^\mathsf{T}\left(\mathbf{x} - \hat{\mathbf{x}}(\alpha)\right)} > \gamma \tag{21b}$$

with $0 < \gamma < \frac{1}{2}$ the minimally acceptable reduction ratio[5].

Algorithm II $\quad \mathbf{x}^* \longleftarrow \mathrm{QP}[\mathbf{H}, \mathbf{q}, \underline{\mathbf{b}}, \bar{\mathbf{b}}, \mathbf{x}]$

Repeat until convergence:

1) Get indices: Equations (19).

2) Get Newton step: Equations (20).

3) Convergence: If $\|\mathbf{g}_\mathsf{f}\| < \epsilon \ll 1$, terminate.

4) Line search: Decrease α in (21a) until (21b) is satisfied. Accept the candidate $\mathbf{x} \leftarrow \hat{\mathbf{x}}(\alpha)$.

By design, the key feature of the algorithm is the following:

Lemma. *If the initial point* $\mathbf{x}$ *has the same clamped constraints as the optimum* $\mathsf{c}(\mathbf{x}) = \mathsf{c}(\mathbf{x}^*)$*, then the solution will be reached in a single iteration.*

Proof: Setting $\Delta\mathbf{x}_\mathsf{f} = 0$ at the optimum, we have from (20b) that $\mathbf{x}_\mathsf{f}^* = -\mathbf{H}_{\mathsf{ff}}^{-1}\left(\mathbf{q}_\mathsf{f} + \mathbf{H}_{\mathsf{fc}}\mathbf{x}_\mathsf{c}\right)$. If $\mathsf{c}(\mathbf{x}) = \mathsf{c}(\mathbf{x}^*)$ then $\mathbf{x}_\mathsf{c} = \mathbf{x}_\mathsf{c}^*$ and therefore $\Delta\mathbf{x}_\mathsf{f} = \mathbf{x}_\mathsf{f}^* - \mathbf{x}_\mathsf{f}$ taking us directly to the minimum in one step. ■

C. Box-DDP/iLQG

Using the notation of Algorithm II, the solution of (17) for $\delta\mathbf{x} = \mathbf{0}$ is given by

$$\mathbf{l}_\mathsf{f} = \mathrm{QP}[\widetilde{Q}_{\mathbf{uu}}, Q_\mathbf{u}, \underline{\mathbf{b}} - \mathbf{u}, \bar{\mathbf{b}} - \mathbf{u}, \mathbf{l}']. \tag{22a}$$

Note that we warm-start the solver with the solution at the subsequent time-step $\mathbf{l}'$. The subscript notation $\mathbf{l}_\mathsf{f}$ indicates that only the free dimensions are non-zero. Unlike in the naive clamping of Sec. II-C1, here the correct formula for the feedback gains emerges naturally. The solution of (17) with a non-zero $\delta\mathbf{x}$ remains linear in $\delta\mathbf{x}$ in the free subspace, leading to the gains

$$\mathbf{L}_\mathsf{f} = \widetilde{Q}_{\mathbf{uu},\mathsf{ff}}^{-1} \cdot Q_{\mathbf{ux},\mathsf{f}} \tag{22b}$$

Here $\widetilde{Q}_{\mathbf{uu},\mathsf{ff}}$ is the reduced Hessian which had already been factorized in (20b), and $\mathbf{L}_\mathsf{f}$ and $Q_{\mathbf{ux},\mathsf{f}}$ are "compressed" by ignoring the rows corresponding to the clamped subspace c. It follows that $\mathbf{L}_\mathsf{c}$, the rows of $\mathbf{L}$ corresponding to clamped controls, are identically zero.

To summarize, the box-constrained iterative-LQG algorithm replaces (9a) and (9b) with (22a) and (22b) in the backward-pass, and replaces (12b) with (13) in the forward-pass.

D. Complexity

In problems with elaborate dynamics, the effort required to compute the derivatives (5) is often significantly larger than that required for the backward-pass. In that case the extra effort required by the box-QP solver will go unnoticed. If however we ignore the time required for the derivatives, or make it very small by computing them in parallel, the leading complexity term comes from the Cholesky factorization in (20b), which is $O(m^3)$. Since standard DDP requires one factorization anyway in (9), the question is how many extra factorizations on average, does the box-QP solution impose. The algorithm performs a factorization whenever $\mathsf{c}(\mathbf{x})$ changes, which might not be often, depending on the problem. As reported below, in our experiments the average number of factorization was never larger than 2.

IV. Results

A. Linear-Quadratic problems

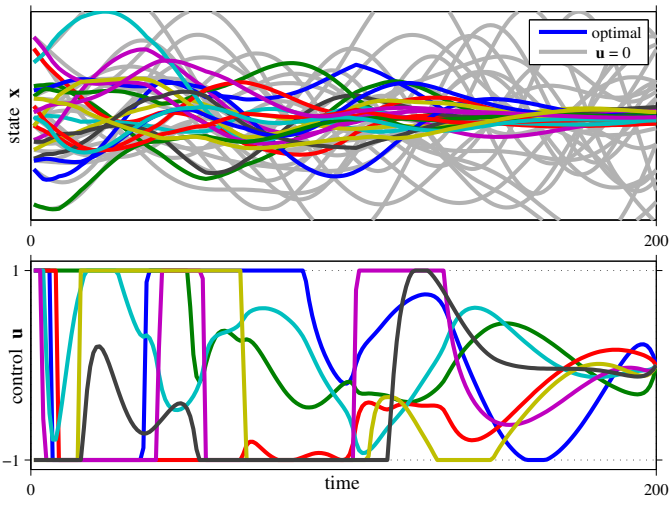

Fig. 1. A typical optimal trajectory of a random linear system. Here $h = 0.01$, $n = 20$, $m = 7$ and $N = 200$. ***Top:*** the state trajectory $\mathbf{X} \equiv \{\mathbf{x}_0 \ldots \mathbf{x}_{200}\}$, the passive dynamics ($\mathbf{u} = 0$) are shown in gray. ***Bottom:*** the control tape $\mathbf{U} \equiv \{\mathbf{u}_0 \ldots \mathbf{u}_{199}\}$. The limits $\underline{\mathbf{b}} = -\mathbf{1}$ and $\bar{\mathbf{b}} = \mathbf{1}$ are indicated.

[4] It is not strictly necessary to sort the indices $\{\mathsf{f}, \mathsf{c}\}$ when repartitioning. We do so here to enhance readability.

[5] We use the oft-quoted $\gamma = 0.1$.

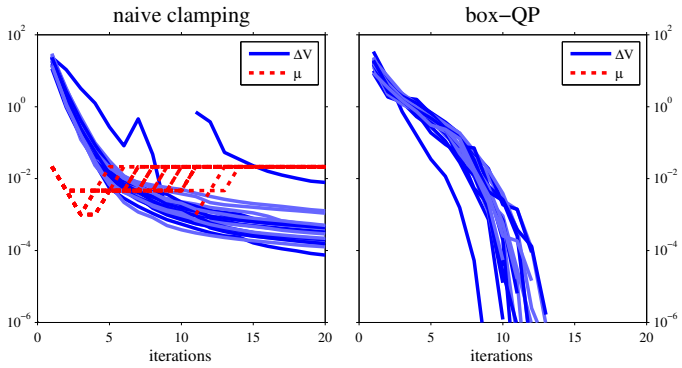

Fig. 2. ***Left:***, convergence trace of random LQ problems using the naïve clamping heuristic described in section II-C. The regularization parameter μ never vanishes, and full convergence ($\Delta V = 0$) is not achieved in the 20 iterations shown. ***Right:*** Convergence trace using the proposed algorithm. Note how the solution requires no regularization ($\mu = 0$), and clearly displays quadratic convergence traces.

The finite-horizon Linear-Quadratic (LQ) optimal control problem is solved by exactly one iteration of DDP with $\mu = 0$ (no regularization). It is described by linear dynamics

$$\mathbf{x}_{i+1} = \mathbf{A}_i\mathbf{x}_i + \mathbf{B}_i\mathbf{u}_i. \tag{23a}$$

and the quadratic optimization criterion

$$\underset{\mathbf{u}_0,\mathbf{u}_1,\ldots\mathbf{u}_{N-1}}{\text{minimize}} \quad \tfrac{1}{2}\mathbf{x}_N^\mathsf{T}\mathbf{Q}_f\mathbf{x}_N + \tfrac{1}{2}\sum_{i=0}^{N-1}\left(\mathbf{x}_i^\mathsf{T}\mathbf{Q}_i\mathbf{x}_i + \mathbf{u}_i^\mathsf{T}\mathbf{R}_i\mathbf{u}_i\right). \tag{23b}$$

Box-constrained LQ problems are the natural first test-case for our algorithm. We generated random LQ problems as follows. The state dimension n was drawn uniformly from $\{10\ldots100\}$. The control dimension m was drawn from $\{1\ldots\lfloor\frac{n}{2}\rfloor\}$. The matrices $\mathbf{A},\mathbf{B},\mathbf{Q},\mathbf{R}$ were all time independent. For a time-step h the random dynamics matrices were $\mathbf{A} = \mathbf{I}_n + h\,\mathbf{N}(n,n)$ and $\mathbf{B} = h\,\mathbf{N}(n,m)$, where $\mathbf{N}$ is a matrix with standard normally distributed elements $\mathbf{N}_{ij} \sim \mathcal{N}(0;1)$, and $\mathbf{I}$ is the identity. The cost matrices were $\mathbf{Q} = \mathbf{Q}_f = h\mathbf{I}_n$ and $\mathbf{R} = c_u h\mathbf{I}_m$ with c_u the control-cost coefficient. Control bounds were $\underline{\mathbf{b}} = -\mathbf{1}$ and $\overline{\mathbf{b}} = \mathbf{1}$. The initial state was drawn from the normal distribution $\mathbf{x}_0 = \mathbf{N}(n,1)$.

Figure 1 shows a typical state-control trajectory. Figure 2 shows a comparison between the naïve clamping heuristic described in section II-C, and the proposed algorithm. The box-QP solution shows quadratic convergence and requires no regularization. Quadratic convergence, which amounts to convergence like Newton's method means the doubling of correct significant bits in the solution with each iteration. This manifests as quadratic-looking traces on a log-plot of the convergence trace, as seen evidenced in Figure 2. The average number of factorizations per iteration was 1.5.

B. Car Parking

Next, we tested our algorithm on a planar car-parking problem. Here one of the control variables – the angle of the front wheels – was kinematic, rather than dynamic variable. When controls specify kinematic variables, bounds often arise naturally from the geometry of the problem rather than from actuator limits. This makes kinematic problems an important class for our proposed algorithm.

(x, y, θ, v) is the 4-dimensional state. x, y is the position of the point midway between the back wheels. θ is the angle of the car relative to the x-axis. v is the velocity of the front wheels. The two control signals are ω the wheel angle and a the acceleration.

For Euler dynamics with a time-step h and letting d denote the distance between the front and back axles, the rolling distance of the front and back wheels are respectively

$$f = hv \tag{24a}$$

$$b = f\cos(\omega) + d - \sqrt{d^2 - f^2\sin^2(\omega)}, \tag{24b}$$

and the h-step dynamics are

$$x' = x + b\cos(\theta) \tag{24c}$$

$$y' = y + b\sin(\theta) \tag{24d}$$

$$\theta' = \theta + \sin^{-1}(\sin(\omega)\frac{f}{d}) \tag{24e}$$

$$v' = v + ha. \tag{24f}$$

The "parking" task was encoded as a final-cost on the distance of the last state from (0,0,0,0), i.e. at the plane's origin, facing east and motionless. Distance was measured using the Huber-type function $z(x,p) = \sqrt{x^2+p^2} - p$. This function is roughly quadratic in a p-sized neighborhood of the origin and linear thereafter. The state cost was

$$\ell_f(\mathbf{x}) = z(x,p_x) + z(y,p_y) + z(\theta,p_\theta) + z(v,p_v)$$

We chose $p_x = p_y = 0.1m$, $p_\theta = 0.01rad$ and $p_v = 1m/s$ to compensate for the relative difficulty of changing each variable. Because it is easier to stop the car ($v = 0$) than to orient it ($\theta = 0$), we would like the optimizer to focus on the harder task once near enough to the goal-state.

In order to make the task more difficult, we added a running cost that penalizes cartesian distance from the origin

$$\ell(\mathbf{x}) = 0.01 \times \big(z(x,p_x) + z(y,p_y)\big).$$

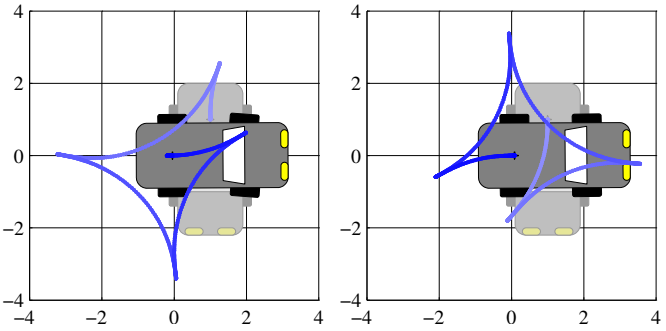

Fig. 3. Two solution trajectories, starting from $(1, 1, \frac{3\pi}{2}, 0)$ (gray car, background) and ending at the goal state (foreground). The blue line shows the trace of the (x, y) coordinates.

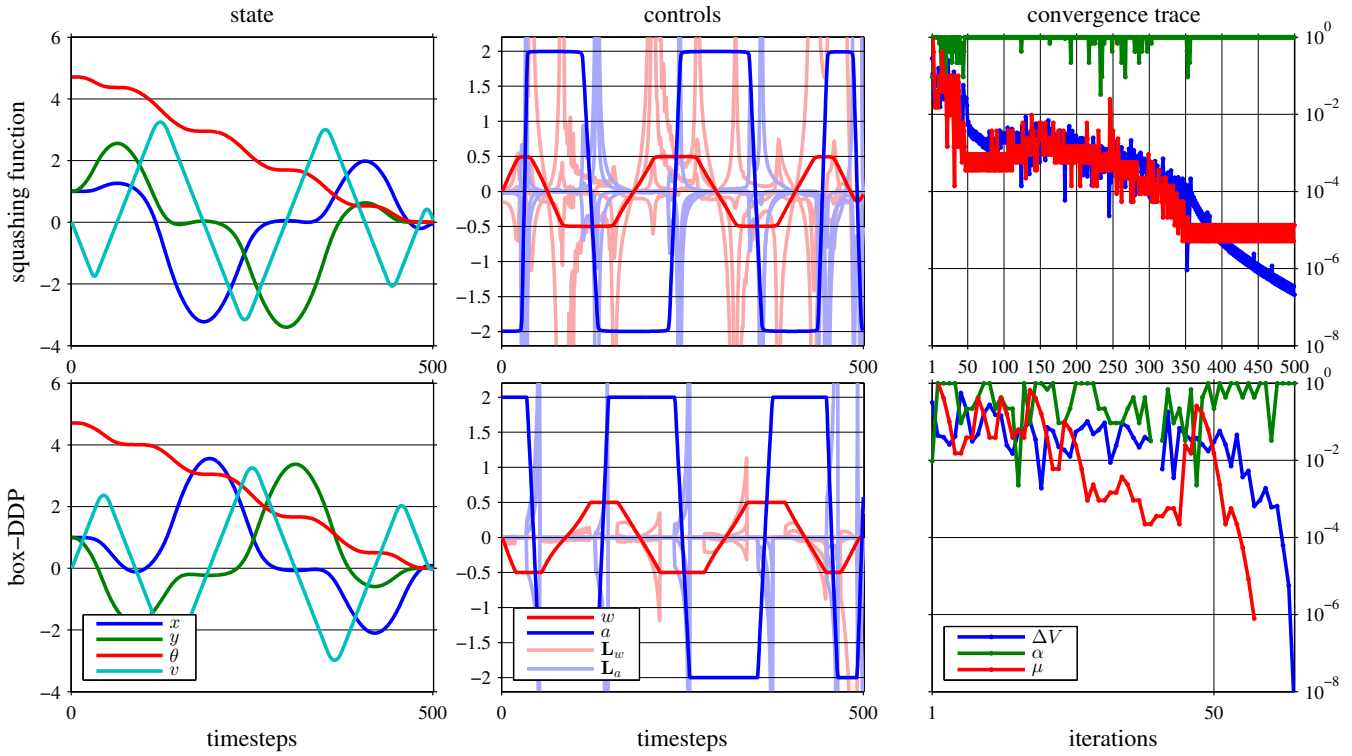

Fig. 4. Comparison between squashing-functions (***top row***) and BOX-DDP (***bottom row***). ***Left column:*** State trajectories $(x, y, \theta, v)_i$ at the optimum. The top and bottom trajectories correspond to the left and right ones in figure 3. ***Middle column:*** Controls (ω, a) (solid) and feedback gains $\mathbf{L}$ (light colors, not to scale). Note how the rows of the 2×4 matrices $\mathbf{L}$ vanish whenever the corresponding controls are clamped, in the BOX-DDP solution. ***Right column:*** Convergence of the two algorithms. BOX-DDP converges quadratically after 64 iterations, while the squashing-function solution barely converged by 500.

This term encourages parking maneuvers which do not take the car far from the origin. The control cost was $\ell(\mathbf{u}) = c_\omega \omega^2 + c_a a^2$ with $c_\omega = 0.01$ and $c_a = 0.0001$. Cost coefficients were chosen to be small in order to encourage the controller to hit the bounds $b_\omega = \pm 0.5 rad$ and $b_a = \pm 2 m/s^2$.

Figure 3 shows two solution trajectories, starting from $(1, 1, \frac{3\pi}{2}, 0)$ and ending at the goal state. Since our optimization is local, convergence to different local minima is to be expected.

1) Squashing Functions: We used the car parking domain to compare box-DDP to the "squashing" heuristic described in section II-C. The squashing function (16) is in this case

$$\omega(\tilde{\omega}) = 0.5 \times \tanh(\tilde{\omega})$$
$$a(\tilde{a}) = 2 \times \tanh(\tilde{a}).$$

In order to prevent the "pre-controls" $(\tilde{\omega}, \tilde{a})$ from diverging, a small explicit cost on these was added

$$\ell(\tilde{\omega}, \tilde{a}) = c_\omega \omega^2 + c_a a^2 + 10^{-6} \times (\tilde{\omega}^2 + \tilde{a}^2).$$

This term is small enough to not significantly modify the problem, but large enough to pull $(\tilde{\omega}, \tilde{a})$ back towards the origin when they are too large. The dimension of this problem is small enough that full DDP can be used, including the 2nd-order terms in (5c, 5d, 5e), allowing for true quadratic convergence.

C. Grasping

The results of this section can only be fully appreciated by watching the accompanying movie:

https://dl.dropbox.com/u/56715/boxDDPgrasping.mp4

Grasping is hard. It is a high high dimensional domain with many dynamics constraints arising from joint limits, contacts of the fingers with the object, and inter-finger contacts. No control method has yet approached the dexterity and agility of the human hand, or the robustness of its control mechanism.

The high-dimensionality allows many different solutions to a particular grasping problem, but also creates many local minima. An initial bad grasp will lead to loss of grip and require a recovery manoeuver.

1) MPC: Our approach to grasping, which to our knowledge is demonstrated here for the first time, is to use Model Predictive Control (MPC), also known as online trajectory-optimization. In MPC the current state of the system is measured, a single iteration of a trajectory-optimization algorithm is applied (BOX-ILQG in this case), and the initial part of the resulting policy is applied to the system until the process can be repeated and the policy updated again. For MPC to succeed, large updates to the policy must be made in each step so that the optimizer can "keep-up" with the changing state. This means that the regularization parameter μ, which slows the optimizer and makes it "careful" should be kept

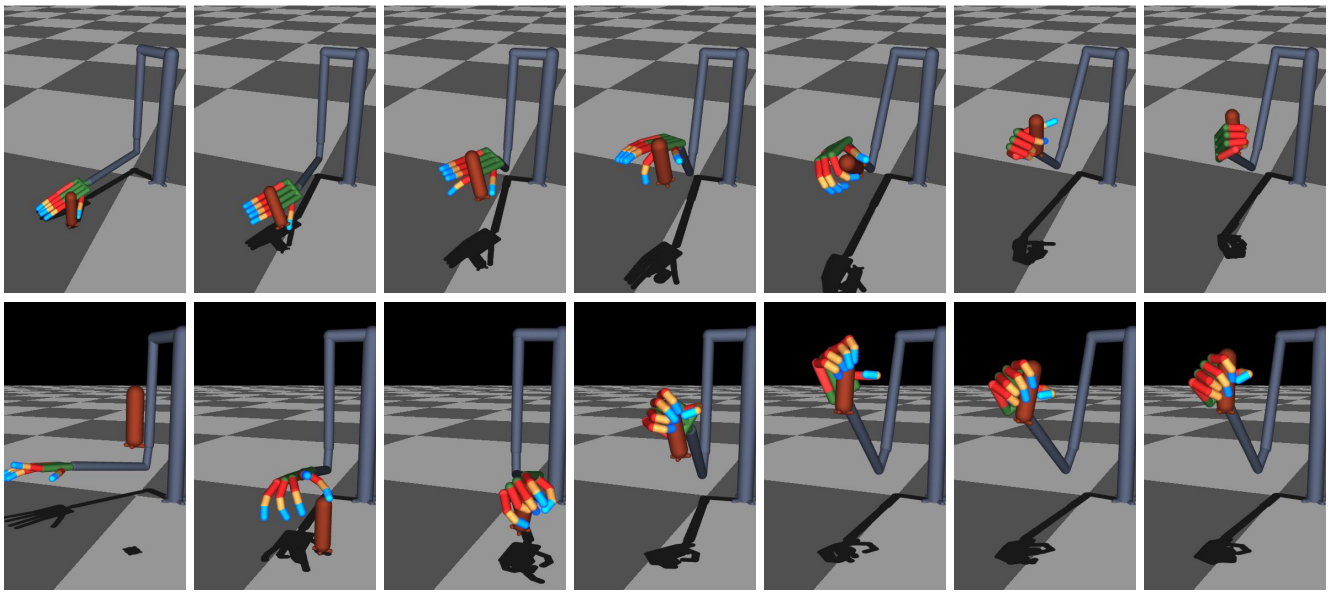

Fig. 5. Two sequences from the accompanying movie. The time between consecutive frames is 150ms. Both sequences show an agile recovery manoeuver. The top sequence begins with a back-handed fumble of the object, which is then caught and brought into the desired position. In the second sequence the initial grip on the object is not robust and the object begins to slip. The controller releases the object and re-grips it in mid-flight.

small.

This explains why a problem such as grasping is difficult for a trajectory optimizer with unbounded controls. Too high a control-cost and the hand gets stuck in local minima. Too low a cost and the back-pass encounters non-PD Hessians in Eq. (6), the regularization factor μ increases, and the optimization slows down too much. This results in wild, uncontrolled movements, due to the mismatch between the state and the policy.

Control limits help by letting us use a small control-cost within the limits, helping us jump out of local minima, but avoiding the divergences of large controls. Some of the divergence can also be attributed to divergence of the forward-pass due to the linear gains. Because the initial state changes on every MPC iteration the gains of the previous, possibly outdated policy can generate large senseless control signals. BOX-iLQG sets the rows of $\mathbf{L}$ to 0 when the limits are hit, reducing this danger.

2) Hand Model: We are currently in the process of designing, assembling, identifying and controlling a 28-DoF arm and hand. In this paper we will describe experiments with a simplified geometric model of this hand. Figure 6 shows a CAD model of our robot and the simplified model used here.

Along with the free object shown in the movie, out problem domain has 34 DoF.

The numerical details of the cost function are specific to our model so we will not

Our cost function was composed of three terms, all using the Huber-type function described above. The first term corresponds to a cost on the object position and orientation. The origin of this term was exposed as a parameter to the user, letting us move the object around once a grasp was achieved. The second term penalizes the object's positional and rotational velocities. The last cost term, with a small coefficient penalized the distance of the hand from the object. This "hint" term is needed for the hand to make initial contact. Once contact has been achieved, the first two terms dominate. Note that we used no grasp-specific costs promoting "force closure" or other supposedly desirable properties. All the behaviors seen in the movie emerged from these simple costs.

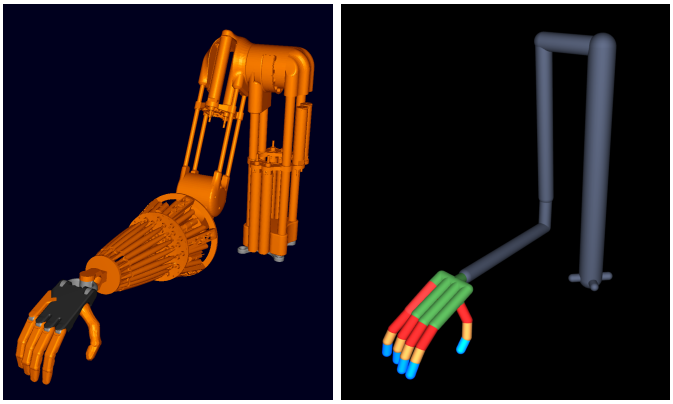

Fig. 6. ***Left:*** A full CAD model of our 28-DOF robotic hand. ***Right:*** A simplified model using capsules to simplify contact detection. The kinematic tree is the same in both models.

ACKNOWLEDGMENT

This work was supported by the National Science Foundation.

REFERENCES

[1] L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishchenko, *The mathematical theory of optimal processes*. Interscience New York, 1962.

[2] O. Stryk and R. Bulirsch, "Direct and indirect methods for trajectory optimization," *Annals of Operations Research*, vol. 37, no. 1, pp. 357–373, Dec. 1992.

[3] D. Q. Mayne, "A second-order gradient method of optimizing non-linear discrete time systems," *Int J Control*, vol. 3, p. 85–95, 1966.

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

[5] L. Z. Liao and C. A. Shoemaker, "Advantages of differential dynamic programming over newton's method for discrete-time optimal control problems," *Cornell University, Ithaca, NY*, 1992.

[6] E. Todorov and W. Li, "A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *Proceedings of the 2005, American Control Conference, 2005.*, Portland, OR, USA, 2005, pp. 300–306.

[7] 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*, 2012.

[8] J. Nocedal and S. J. Wright, *Numerical optimization*. New York: Springer, 2006.

[9] D. P. Bertsekas, "Projected newton methods for optimization problems with simple constraints," *SIAM Journal on Control and Optimization*, vol. 20, no. 2, pp. 221–246, Mar. 1982. [Online]. Available: http://epubs.siam.org/doi/abs/10.1137/0320018

[10] L. Armijo, "Minimization of functions having lipschitz continuous first partial derivatives." *Pacific Journal of Mathematics*, vol. 16, no. 1, pp. 1–3, 1966.