2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 7-12, 2012. Vilamoura, Algarve, Portugal

Synthesis and Stabilization of Complex Behaviors through Online Trajectory Optimization

Yuval Tassa, Tom Erez and Emanuel Todorov
University of Washington

***Abstract*— We present an online trajectory optimization method and software platform applicable to complex humanoid robots performing challenging tasks such as getting up from an arbitrary pose on the ground and recovering from large disturbances using dexterous acrobatic maneuvers. The resulting behaviors, illustrated in the attached video, are computed only 7 x slower than real time, on a standard PC. The video also shows results on the acrobot problem, planar swimming and one-legged hopping. These simpler problems can already be solved in real time, without pre-computing anything.**

I. INTRODUCTION

Online trajectory optimization, also known as Model-Predictive Control (MPC), is among the most powerful methods for automatic control. It retains the key benefit of the optimal control framework: the ability to specify high-level task goals through simple cost functions, and synthesize all details of the behavior and control law automatically. At the same time MPC side-steps the main drawback of dynamic programming – the *curse of dimensionality*. This drawback is particularly problematic for humanoid robots, whose state space is so large that no control scheme (optimal or not) can explore all of it in advance and prepare suitable responses for every conceivable situation.

MPC avoids the need for extensive exploration by postponing the design of the policy until the last minute, and thereby finding controls only for the states that are actually visited. This is done by re-optimizing the movement trajectory and associated control sequence at each time step of the control loop, always starting at the current state estimate. The first control signal is applied to the system, the next state is measured/estimated, and the procedure is then repeated. The trajectory optimizer is warm-started with the solution from the previous time step, which greatly speeds up the method and often yields convergence after a single (re)optimization step. The trajectory being optimized extends to some pre-defined horizon; thus this approach is also known as receding-horizon control. A short horizon reduces the amount of computation but results in myopic behaviors.

In domains such as chemical process control where the dynamics are sufficiently slow and smooth – and thus online trajectory optimization is already feasible – MPC is the method of choice [1]. In robotics, however, the typical timescales of the dynamics are orders of magnitude faster. Furthermore, many robotic tasks involve contact phenomena that present a serious challenge to optimization-based approaches. As a result, MPC is rarely used to control dexterous robots. This is not because robotics researchers are unaware of it or unwilling to use it, but simply because they lack the tools to make it work. While recent examples demonstrate the power of MPC applied to robotics [2], [3], much work remains to be done before it becomes a standard off-the-shelf tool. When it does, we believe it will revolutionize the field and enable control of complex behaviors currently only seen in movies.

A. Specific contributions

The results presented here are enabled by advances on multiple fronts. Our new physics simulator, called MuJoCo, was used to speed up the computation of dynamics derivatives. MuJoCo is a C-based, platform-independent, multi-threaded simulator tailored to control applications. We detail several improvements to the iterative LQG method for trajectory optimization [4] that increase its efficiency and robustness. We present a simplified model of contact dynamics which yields a favorable trade-off between physical realism and speed of simulation/optimization. We introduce cost functions that result in better-behaved energy landscapes and are more amenable to trajectory optimization. Finally, we describe a MATLAB-based environment where the user can modify the dynamics model, cost function or algorithm parameters, while interacting in real time with the controlled system. We have found that hands-on familiarity with the various strengths and weaknesses of the MPC machinery is invaluable for proper control design.

These advances have enabled us to synthesize complex humanoid behaviors, like getting up from the ground from an arbitrary initial pose and recovering from large disturbances, in near real-time. As we show in the attached video, we can easily solve commonly-studied problems like the “acrobot”, and also more challenging ones like swimming and one-legged hopping. These simpler problems can already be solved in real time, without pre-computing anything and without specifying heuristic approximations to the value function. We show both robustness to state perturbations, a generic feature of the online approach, since it always optimizes the trajectory starting at the present (possibly perturbed) state, and robustness to modeling errors – by optimizing trajectories with respect to one model and applying the resulting controls to a different model

II. ONLINE TRAJECTORY OPTIMIZATION

MPC is based on repeatedly solving a finite-horizon optimal control problem. This is done here using a trajectory optimization method (iterative LQG) which is the control analog of the Gauss-Newton method for nonlinear least-squares optimization. Below we provide some background on finite-horizon optimal control and trajectory optimization, and then focus on the specific improvements we have made here.

A. *Finite-Horizon Optimal Control*

The discrete-time dynamics

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

describe the evolution of the state $\mathbf{x}$ given the control $\mathbf{u}$. The *total cost* J_0 is the sum of running costs ℓ and final cost ℓ_f, incurred when starting from state $\mathbf{x}_0$ and applying the control sequence $\mathbf{U} \equiv \{\mathbf{u}_0, \mathbf{u}_1 \dots, \mathbf{u}_{N-1}\}$ until the horizon is reached:

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

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

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

By *trajectory optimization*, we mean finding $\mathbf{U}^*(\mathbf{x})$ for a particular $\mathbf{x}$, rather than for all possible initial states[1].

B. *Trajectory Optimizer*

In the experiments described below we used the *iterative Linear Quadratic Gaussian* (iLQG) trajectory optimizer [4]. iLQG is a variant of the classic *Differential Dynamic Programming* (DDP) algorithm [5], the main difference being that only *first* rather than *second* derivatives of the dynamics are used. This means that iLQG no longer exhibits the quadratic convergence properties of DDP, however in the MPC context the minimum is a moving target and convergence never actually happens, so the benefits of having faster dynamics evaluation greatly outweigh the decrease in performance. In our version of iLQG, we implemented several improvements to the regularization and line-search aspects of the algorithm.

Let $\mathbf{U}_i \equiv \{\mathbf{u}_i, \mathbf{u}_{i+1} \dots, \mathbf{u}_{N-1}\}$ and define the *cost-to-go* J_i as the partial sum of costs from i to N:

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

The *Value* at time i is the cost-to-go given the minimizing control sequence

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

Setting $V(\mathbf{x}, N) \equiv \ell_f(\mathbf{x}_N)$, the Dynamic Programming Principle reduces the minimization over an entire sequence of controls to a sequence of minimizations over a single control, proceeding backwards in time:

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

[1]Other trajectory optimization schemes interpret Eq. (1) as a constraint and optimize over both states and controls.

Define the argument of the minimum in (2) as a function of perturbations around the i-th $(\mathbf{x}, \mathbf{u})$ pair:

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

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 expansion coefficients are[2]

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

The last terms in (5c, 5d, 5e), which denote contraction with a tensor, are ignored in iLQG (but not in DDP). Since the step computed by DDP is nearly identical to the full Newton step (see e.g. [6]), iterative LQG can be seen to correspond to the Gauss-Newton Hessian approximation.

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

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

giving us an open-loop term $\mathbf{k} = -Q_{\mathbf{uu}}^{-1} Q_{\mathbf{u}}$ and a feedback gain term $\mathbf{K} = -Q_{\mathbf{uu}}^{-1} Q_{\mathbf{ux}}$. Plugging the policy into (4), we now have a quadratic model of the Value at time i:

$$\Delta V(i) = -\tfrac{1}{2} Q_{\mathbf{u}} Q_{\mathbf{uu}}^{-1} Q_{\mathbf{u}} \tag{7a}$$
$$V_{\mathbf{x}}(i) = Q_{\mathbf{x}} - Q_{\mathbf{u}} Q_{\mathbf{uu}}^{-1} Q_{\mathbf{ux}} \tag{7b}$$
$$V_{\mathbf{xx}}(i) = Q_{\mathbf{xx}} - Q_{\mathbf{xu}} Q_{\mathbf{uu}}^{-1} Q_{\mathbf{ux}}. \tag{7c}$$

Recursively computing the local quadratic models of $V(i)$ and the control modifications $\{\mathbf{k}(i), \mathbf{K}(i)\}$, constitutes the backward pass. Once it is completed, a forward pass computes a new trajectory:

$$\hat{\mathbf{x}}(1) = \mathbf{x}(1) \tag{8a}$$
$$\hat{\mathbf{u}}(i) = \mathbf{u}(i) + \mathbf{k}(i) + \mathbf{K}(i)(\hat{\mathbf{x}}(i) - \mathbf{x}(i)) \tag{8b}$$
$$\hat{\mathbf{x}}(i{+}1) = \mathbf{f}(\hat{\mathbf{x}}(i), \hat{\mathbf{u}}(i)) \tag{8c}$$

C. *Improved Regularization*

It has been shown [6] that the steps taken by DDP are comparable to or better than a full Newton step for the entire control sequence. And as in Newton's method, care must be taken when the Hessian is not positive-definite or when the minimum is not close and the quadratic model inaccurate. The standard regularization, proposed in [5] and further explored in [7], is to add a diagonal term to the local control-cost Hessian

$$\widetilde{Q}_{\mathbf{uu}} = Q_{\mathbf{uu}} + \mu \mathbf{I}_m, \tag{9}$$

[2]Dropping the index i, primes denoting the next time-step: $V' \equiv V(i{+}1)$.

where μ plays the role of a Levenberg-Marquardt parameter. This modification amounts to adding a quadratic cost around the current control-sequence, making the steps more conservative. The drawback to this regularization scheme is that the same control perturbation can have different effects at different times, depending on the control-transition matrix $\mathbf{f_u}$. We therefore introduce a scheme that penalizes deviations from the states rather than controls:

$$\widetilde{Q}_{\mathbf{uu}} = \ell_{\mathbf{uu}} + \mathbf{f}_{\mathbf{u}}^{\mathsf{T}}(V'_{\mathbf{xx}} + \mu \mathbf{I}_n)\mathbf{f_u} + V'_{\mathbf{x}} \cdot \mathbf{f_{uu}} \tag{10a}$$

$$\widetilde{Q}_{\mathbf{ux}} = \ell_{\mathbf{ux}} + \mathbf{f}_{\mathbf{u}}^{\mathsf{T}}(V'_{\mathbf{xx}} + \mu \mathbf{I}_n)\mathbf{f_x} + V'_{\mathbf{x}} \cdot \mathbf{f_{ux}} \tag{10b}$$

$$\mathbf{k} = -\widetilde{Q}_{\mathbf{uu}}^{-1}\widetilde{Q}_{\mathbf{u}} \tag{10c}$$

$$\mathbf{K} = -\widetilde{Q}_{\mathbf{uu}}^{-1}\widetilde{Q}_{\mathbf{ux}} \tag{10d}$$

This regularization amounts to placing a quadratic state-cost around the previous sequence. Unlike the standard control-based regularization, the feedback gains $\mathbf{K}$ do not vanish as $\mu \to \infty$, but rather force the new trajectory closer to the old one, significantly improving robustness.

Finally, we make use of the improved Value update proposed in [4]. Examining (4, 6, 7), we see that several cancelations of $Q_{\mathbf{uu}}$ and its inverse have taken place, but since we are modifying this matrix in (9) or (10a), making those cancelations induces an error. The improved Value update is therefore

$$\Delta V(i) = \quad + \tfrac{1}{2}\mathbf{k}^{\mathsf{T}} Q_{\mathbf{uu}}\mathbf{k} + \mathbf{k}^{\mathsf{T}} Q_{\mathbf{u}} \tag{11a}$$

$$V_{\mathbf{x}}(i) = Q_{\mathbf{x}} + \mathbf{K}^{\mathsf{T}} Q_{\mathbf{uu}}\mathbf{k} + \mathbf{K}^{\mathsf{T}} Q_{\mathbf{u}} + Q_{\mathbf{ux}}^{\mathsf{T}}\mathbf{k} \tag{11b}$$

$$V_{\mathbf{xx}}(i) = Q_{\mathbf{xx}} + \mathbf{K}^{\mathsf{T}} Q_{\mathbf{uu}}\mathbf{K} + \mathbf{K}^{\mathsf{T}} Q_{\mathbf{ux}} + Q_{\mathbf{ux}}^{\mathsf{T}}\mathbf{K}. \tag{11c}$$

D. *Improved Line Search*

The forward pass of iLQG/DDP, given by Eqs. (8) is the key to the algorithm's fast convergence. This is because the feedback gains in (8b) generate a new control sequence that takes into account the new states as they are being integrated. For example when applying the algorithm to a linear-quadratic system, even a time-varying one, an exact solution is obtained after a single iteration. The caveat is that for a general non-linear system, when the new trajectory strays too far from the model's region of validity, the cost may not decrease, and divergence may occur. The solution is to introduce a backtracking line-search parameter $0 < \alpha \leq 1$ and integrate using

$$\hat{\mathbf{u}}(i) = \mathbf{u}(i) + \alpha\mathbf{k}(i) + \mathbf{K}(i)(\hat{\mathbf{x}}(i) - \mathbf{x}(i)) \tag{12}$$

For $\alpha = 0$ the trajectory would be unchanged, but for intermediate values the resulting control step is not a simple scaled version of the full step, due to the presence of feedback. As advocated in [5], we use the expected total-cost reduction in the line-search procedure, but using the improved formula (11a), we can derive a better estimate:

$$\Delta J(\alpha) = \alpha \sum_{i=1}^{N-1} \mathbf{k}(i)^{\mathsf{T}} Q_{\mathbf{u}}(i) + \frac{\alpha^2}{2} \sum_{i=1}^{N-1} \mathbf{k}(i)^{\mathsf{T}} Q_{\mathbf{uu}}(i)\mathbf{k}(i).$$

When comparing the actual and expected reductions

$$z = [J(\mathbf{u}_{1..N-1}) - J(\hat{\mathbf{u}}_{1..N-1})]/\Delta J(\alpha),$$

we accept the iteration only if

$$0 < c_1 < z. \tag{13}$$

E. *Trajectory Optimizer Summary*

A single iteration of iLQG is composed of 3 steps:

1. Derivatives: Given a nominal $(\mathbf{x}, \mathbf{u}, i)$ sequence, compute the derivatives of ℓ and $\mathbf{f}$ in the RHS of Eq. (5). This step is parallelized for all i across all available CPU cores.

2. Backward pass: Iterate Eqs. (5, 10, 11) for decreasing $i = N-1, \ldots 1$. If a non-PD $\widetilde{Q}_{\mathbf{uu}}$ is encountered, increase μ and restart the backward pass. If successful, decrease μ.

3. Forward pass: Set $\alpha = 1$. Iterate (12) and (8c) to compute a new nominal sequence. If the integration diverged or condition (13) was not met, decrease α and restart the forward pass.

F. *Regularization Schedule*

The fast and accurate modification of the regularization parameter μ in step **2** turns out to be quite important due to three conflicting requirements. If we are near the minimum we would like μ to quickly go to zero to enjoy fast convergence. If the back-pass fails (a non-PD $\widetilde{Q}_{\mathbf{uu}}$), we would like it to increase very rapidly, since the minimum value of μ which prevents divergence is often very large. Finally, if we are in a regime where some $\mu > 0$ is required, we would like to accurately tweak it to be as close as possible to the minimum value, but not smaller. Our solution is to use a quadratic modification schedule. Defining some minimal value μ_{min} (we use $\mu_{\text{min}} = 10^{-6}$) and a minimal modification factor Δ_0 (we use $\Delta_0 = 2$), we adjust μ as follows:

increase μ:

$$\Delta \leftarrow \max(\Delta_0, \Delta \cdot \Delta_0)$$

$$\mu \leftarrow \max(\mu_{\text{min}}, \mu \cdot \Delta)$$

decrease μ:

$$\Delta \leftarrow \min(\tfrac{1}{\Delta_0}, \tfrac{\Delta}{\Delta_0})$$

$$\mu \leftarrow \begin{cases} \mu \cdot \Delta & \text{if } \mu \cdot \Delta > \mu_{\text{min}}, \\ 0 & \text{if } \mu \cdot \Delta < \mu_{\text{min}}. \end{cases}$$

G. *Policy-Lag and Asynchronous Control*

The experiments described below are performed in simulation using two instantiations of the dynamics. One is used by the controller for MPC while a different one is used to simulate the robot. A clear advantage of this is that we can easily introduce "modeling errors" to quantify robustness. However, the most important quantity this allows us to examine is the *policy-lag*, i.e. the time required by the controller to complete one MPC iteration, and its effect on performance. During this time the previous policy is used, and at some point it stops being a good one. Note that this is a very problem-dependent quantity, since it is closely related to the time-variation of the policy – a smooth policy would be less sensitive to lag than a rapidly changing one. We quantify the effects of policy lag by running the controller and simulation asynchronously on different execution threads; if performance is unacceptable, we artificially slow down the plant simulation. As it slows, new policies effectively arrive earlier and performance improves. Once the performance of the controller is acceptable, the slowdown coefficient answers the question "How much faster should our computer be so that this controller would work on a real robot?"

A side benefit of this architecture is that because the separate execution threads communicate over TCP/IP sockets, it is trivial to run them on two different machines. In this scenario a small, cheap CPU running on the robot performs estimation, while an MPC "policy server" runs on a more powerful machine.

III. DYNAMICS MODELING AND SIMULATION

Online trajectory optimization is only possible when the dynamics and its derivatives can be evaluated very quickly. While some work has been done on analytical differentiation [8], it is limited to smooth dynamics and does not apply to general physics engines that must deal with collision detection, computation of contact interaction forces, enforcing nonlinear equality constraints etc. Therefore the derivatives have to be approximated using finite-differencing. Indeed this is where almost all the CPU time is spent. How many dynamics evaluations does MPC require? Suppose we have a system with 20 dofs (and so the state space is 40 dimensional because it includes positions and velocities) and the horizon is 50 time steps of 10ms each. Thus approximating the first derivative at each point along the trajectory (the quantity $\mathbf{f}_\mathbf{x}$ above) using centered finite differencing requires 4,000 dynamics evaluations per time step of the control loop. A sequential real-time simulation of these time-steps would take 4 seconds, but a typical maximal policy-lag (e.g. for our humanoid problem) was on the order of 10ms, so the physics engine must run at least 4,000 faster than real-time! Existing engines are not designed for such speed, and so we had to implement a new full-featured physics engine from scratch (see below). Apart from careful implementation and choice of contact simulation methods, we use parallel processing, which is well-suited for finite differencing because the dynamics at many states can be evaluated in parallel, without need for synchronization or exchange of data.

A. The MuJoCo physics engine

The simulations described in this paper were carried out using MuJoCo [9], which stands for Multi Joint dynamics with Contact. This engine will soon be made publicly available and will be free for academic research. MuJoCo is a platform-independent physics simulator tailored to control applications. Multi-joint dynamics are represented in joint coordinates and computed via recursive algorithms. The computation is $O(n^3)$ because the inverse inertia matrix is needed (to compute contact responses), however due to tree-induced sparsity, performance is comparable to $O(n)$ algorithms in typical usage scenarios (e.g. simulating a humanoid). Geometry is modeled using a small library of smooth shapes allowing fast and accurate collision detection. Contact responses are computed by efficient new algorithms [10]–[12] that appear to be faster and more accurate than LCP-based methods, and are suitable for numerical optimization. Models are specified using either a high-level C++ API or an XML file. A built-in compiler transforms the user model into an optimized data structure used for runtime computation. This data structure contains a scratchpad where all routines write their output. In this way all intermediate results are accessible to the user, making it easy to add functionality. The user can modify all real-valued model parameters in runtime without recompiling. To facilitate optimal control applications, MuJoCo provides routines for parallel computation of the cost of a given trajectory as well as the gradient and a Gauss-Newton approximation to the Hessian. The engine can be used either as a library linked to a user program, or via a MATLAB interface. A utility for interactive 3D rendering is also provided.

B. Contact Modeling

Frictional contact is perhaps the most difficult aspect of dynamics modeling. In the physical world contact phenomena are very stiff, i.e. they happen on very short time-scales. When bodies are modeled as infinitely stiff, the simulation becomes discontinuous. It is possible to model compliant bodies, but this adds many degrees-of-freedom to the model. Because trajectory optimizers require derivatives of the dynamics, discontinuous models cannot be used. Differentiable contact models for rigid bodies fall into two categories. The first type attempt to model the physics as closely as possible, e.g. with Hertz-Hunt-Crossley spring-dampers. These models are indeed accurate, but their stiffness demands extremely small time-steps, on the order of microseconds, and are therefore prohibitively expensive. The second type of models are based on *time stepping* integrators [13], which attempt to model the effects of contact and friction impulses over fixed, relatively large time-steps. Because many contacts can occur in each time-step (typically on the order of $10ms$), these integrators need to consider all contacts at once, usually by solving a Linear Complementarity Problem. Some smooth variants of the time stepping approach such as [10] and [12] solve an optimization problem rather than an LCP, but the result is the same – small but cheap time-steps are exchanged for larger but more expensive ones. Here we introduce a new contact model, based on the time stepping formulation, that attempts to find a balance between these two types. It is as cheap to compute as spring-damper models, yet not as stiff, producing realistic behavior for time-steps in the range of 1-$10ms$.

C. Time-Stepping

The state $\mathbf{x}$ of a mechanical system is a set of generalized positions $\mathbf{q}$ and velocities $\mathbf{v}$. The dynamics are given by the equations of motion

$$\mathbf{M}\dot{\mathbf{v}} = \mathbf{r} + \mathbf{u}$$
$$\dot{\mathbf{q}} = \mathbf{v},$$

where $\mathbf{M} = \mathbf{M}(\mathbf{q})$ is the mass matrix, $\mathbf{r} = \mathbf{r}(\mathbf{q}, \mathbf{v})$ the vector of total external forces (gravity, drag, centripetal, coriolis etc.) and $\mathbf{u}$ is the applied control (e.g. motor torques). For a timestep h, a semi-implicit Euler integration step (primes denoting the next time step) is:

$$\mathbf{M}\mathbf{v}' = h(\mathbf{r} + \mathbf{u}) + \mathbf{M}\mathbf{v}$$
$$\mathbf{q}' = \mathbf{q} + h\mathbf{v}'.$$

The unilateral constraint vector function $\phi(\mathbf{q})$ is a signed distance between objects – positive for separation, zero for contact and negative for penetration. We therefore search for impulses λ such that

$$\phi(\mathbf{q}') \approx \phi(\mathbf{q}) + h\mathbf{J}\mathbf{v}' \geq 0,$$

where $\mathbf{J} = \nabla\phi(\mathbf{q})$. This leads to a mixed complementarity problem for $\mathbf{v}'$ and λ:

$$\mathbf{M}\mathbf{v}' = h(\mathbf{r} + \mathbf{u}) + \mathbf{M}\mathbf{v} + \mathbf{J}^{\mathsf{T}}\lambda \tag{14a}$$
$$\lambda \geq 0, \tag{14b}$$
$$\phi(\mathbf{q}) + h\mathbf{J}\mathbf{v}' \geq 0, \tag{14c}$$
$$\lambda^{\mathsf{T}}(\phi(\mathbf{q}) + h\mathbf{J}\mathbf{v}') = 0. \tag{14d}$$

Conditions (14b) and (14c) are read element-wise, and respectively constrain the contact impulse to be non-adhesive, and the distance to be non-penetrating. Condition (14d) asserts that $\phi(\mathbf{q}') > 0$ (broken contact) and $\lambda > 0$ (collision impact), are mutually exclusive. Since the mass matrix is always invertible, we can solve (14a) for $\mathbf{v}'$, and plug into (14c). Defining

$$\mathbf{A} = \mathbf{J}\mathbf{M}^{-1}\mathbf{J}^{\mathsf{T}} \tag{15a}$$
$$\mathbf{b} = \phi(\mathbf{q})/h + \mathbf{J}(\mathbf{v} + h\mathbf{M}^{-1}(\mathbf{r} + \mathbf{u})), \tag{15b}$$

we can now write (14) in standard LCP form:

$$\text{Find } \lambda \quad s.t. \quad 0 \leq \lambda \perp \mathbf{A}\lambda + \mathbf{b} \geq 0. \tag{16}$$

D. Diagonal Approximation

Instead of solving (16) simultaneously for all the impulses λ, we first take the diagonal approximation to $\mathbf{A}$, and solve independently for each component[3]:

$$\bar{\lambda}_i = -\mathbf{b}_i/\mathbf{A}_{ii}.$$

The $\bar{\lambda}_i$ are the impulses that would make the ϕ_i vanish, regardless of sign. We now scale these by some factor $0 < \eta < 1$ and make them positive with a smooth approximation to $\max(\cdot, 0)$:

$$\lambda_i = \operatorname{smax}(\eta\bar{\lambda}_i, \beta) \tag{17}$$

The function $\operatorname{smax}(x, \beta)$ is shown in Figure 1.

In order to incorporate the frictional impulses ν, we need to use the Coulomb friction law $\|\nu\| \leq \mu\lambda$, where μ is the friction coefficient. Similarly to the procedure for the normal impulses, we first find those frictional impulses which would make the sliding velocity vanish

$$\bar{\nu}_{i,1} = -\mathbf{b}_{i,1}/\mathbf{A}_{ii,1}$$
$$\bar{\nu}_{i,2} = -\mathbf{b}_{i,2}/\mathbf{A}_{ii,2}$$

Here the subscripts 1,2 indicate the two components tangential to $\mathbf{J} = \nabla\phi(\mathbf{q})$ at the contact, and the vector $\mathbf{b}$ and matrix $\mathbf{A}$ are computed using the tangent Jacobians. We then smooth-clamp the frictional impulses to lie inside the friction cone:

$$\nu_i = \frac{\bar{\nu}_i}{\|\bar{\nu}_i\|}\operatorname{smin}(\|\bar{\nu}_i\|, \mu\lambda_i).$$

Where the function $\operatorname{smin}(x, \beta) = \beta - \operatorname{smax}(-x, \beta)$ is a smooth approximation to $\min(x, \beta)$.

[3]Here i indexes over the contacts.

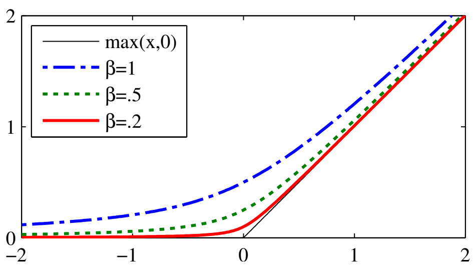


Fig. 1. The smooth-max function $\operatorname{smax}(x, \beta) = (\sqrt{x^2 + \beta^2} + x)/2$.

IV. COST FUNCTION DESIGN

As with any control framework, synthesizing a good controller involves iterations of control design. In the Optimal Control context the designer must specify the cost-function, the horizon length, and dynamical parameters like the controller's time-step, contact-model parameters etc. A tangible benefit of the MPC framework is the fast control-design loop. Any change to the design parameters is immediately reflected in the performance of the controller. This is in contrast to offline optimization-based approaches such as policy-gradient or global dynamic programming, where data must be collected and/or expensive computations performed before the effects of design choices can be appreciated. In order to maximize these benefits, our graphical user interface allows us to change the most important parameters on-the-fly, while the controller is running (Fig. 4).

A. Cost design

Two important parameters which have a direct impact on performance are the simulation time-step dt and the horizon length T. Since speed is of the essence, the goal is to choose those values which minimize the number of steps in the trajectory, i.e. the largest possible time-step and the shortest possible horizon. The size of dt is limited by our use of Euler integration; beyond some value the simulation becomes unstable. The minimum length of the horizon T is a problem-dependent quantity which must be found by trial-and-error. Quadratic functions are most likely the first choice when selecting state-cost and control-cost terms, due to their familiarity from the LQR framework. We started by using these, but have subsequently come to prefer different functions.

For the state-cost, we use the "smooth-abs" function

$$\ell(x) = \sqrt{x^2 + \alpha^2} - \alpha.$$

This function, shown in Figure 2, is smooth in an α-sized neighborhood of the origin, and then becomes linear further away. The linear regime offers two advantages. The first is that in the finite-horizon formulation, a change in x integrated over the trajectory leads to a fixed change in total cost, and therefore encourages periodic behaviour. For example if x is the distance from some target, the same behaviour would emerge at different distances. The second benefit has to do with the relative weighting of state-cost terms. Because the linear regime is unit-preserving, the relative weight of different cost terms maintains the original relationship of the underlying units. For example if one cost term encodes reaching a target in the xy plane while another encodes keeping the torso at certain height z, it is easier to find the appropriate weights for these terms because their contribution, in the linear regime, is proportional to distance.

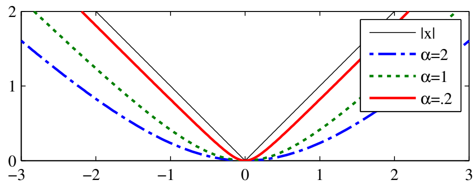


Fig. 2. The smooth-abs cost function $\ell(x) = \sqrt{x^2 + \alpha^2} - \alpha$.

For the control-cost term the quadratic works well in the sense of finding a solution, however that solution is not always desirable. It is often the case that controls are inherently limited and ideally one would want a control cost that grows to infinity at these limits. The problem with such functions is that the trajectory optimizer will often try to specify a control that is outside the limits, leading to infinite or undefined total cost and wasting an MPC iteration. Instead, we use the function

$$\ell(u) = \alpha^2(\cosh(u/\alpha) - 1).$$

This function, shown in Figure 3, has a second-derivative of 1 at the origin and then grows exponentially beyond an α-sized neighborhood. By varying α, we can easily limit the controls to a particular volume of u-space, without risking undefined or infinite values.

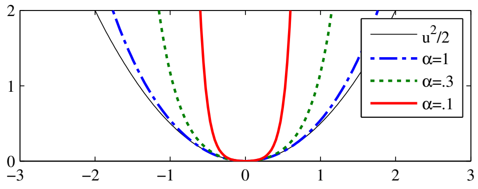


Fig. 3. The control-limiting cost function $\ell(u) = \alpha^2(\cosh(u/\alpha) - 1)$.

B. Interactive GUI

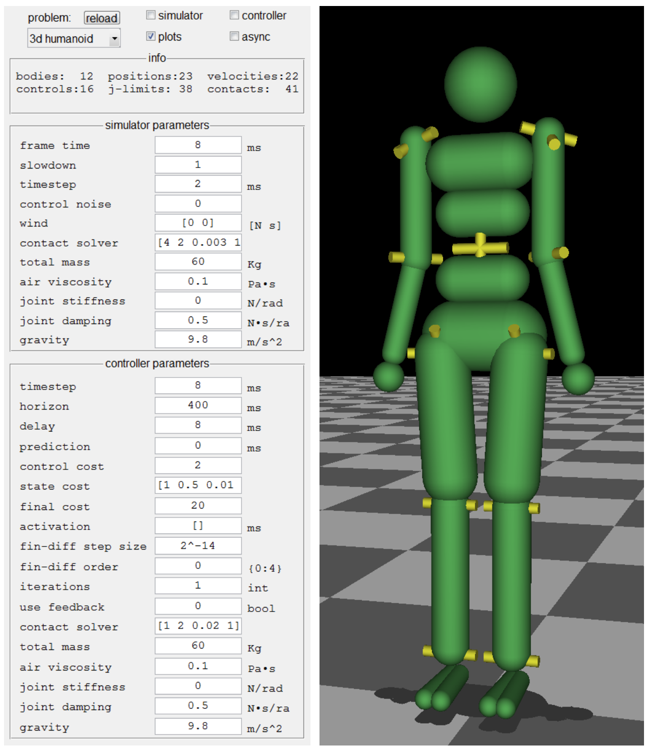
We implemented a MATLAB-based GUI shown in Figure 4. It enabled us to explore the effects of dynamics and algorithm parameters interactively. The ability to modify the parameters in real time and observe their effect on the controller proved to be invaluable for proper tuning. Note that we have two sets of dynamics parameters, one used by the optimizer and the other by the simulator. By setting them to different values, we can simulate the effects of model errors.

V. RESULTS

The power of our resulting controller can only be fully appreciated by watching the video, also available here:

www.cs.washington.edu/homes/tassa/media/IROS12.mp4

As seen in the video, we experimented with several control problems, but due to space constraints we will focus here only on the most challenging one – a 22-DoF humanoid model. It is 1.6m tall and weighs 55Kg. The hip, shoulder and abdomen joints are 2-DoF while elbows, knees and ankles are 1-DoF, see Fig 4.



```
bodies:  12  positions:23  velocities:22
controls:16  j-limits: 38  contacts:  41
```

frame time	8	ms
slowdown	1	
timestep	2	ms
control noise	0	
wind	[0 0]	[N s]
contact solver	[4 2 0.003 1	
total mass	60	Kg
air viscosity	0.1	Pa•s
joint stiffness	0	N/rad
joint damping	0.5	N•s/ra
gravity	9.8	m/s^2

timestep	8	ms
horizon	400	ms
delay	8	ms
prediction	0	ms
control cost	2	
state cost	[1 0.5 0.01	
final cost	20	
activation	[]	ms
fin-diff step size	2^-14	
fin-diff order	0	{0:4}
iterations	1	int
use feedback	0	bool
contact solver	[1 2 0.02 1]	
total mass	60	Kg
air viscosity	0.1	Pa•s
joint stiffness	0	N/rad
joint damping	0.5	N•s/ra
gravity	9.8	m/s^2

Fig. 4. **Left**: Our MATLAB-based GUI for real-time exploration of problem and algorithm parameters. **Right**: the 22-DoF humanoid model at its initial configuration, with visualization of the 16 joints.

The state-cost is composed of 4 terms. The first term penalizes the horizontal distance (in the xy-plane) between the center-of-mass (CoM) and the mean of the feet positions. The second term penalizes the horizontal distance between the torso and the CoM. The third penalizes the vertical distance between the torso and a point 1.3m over the mean of the feet. All three terms use the smooth-abs norm (Figure 2).

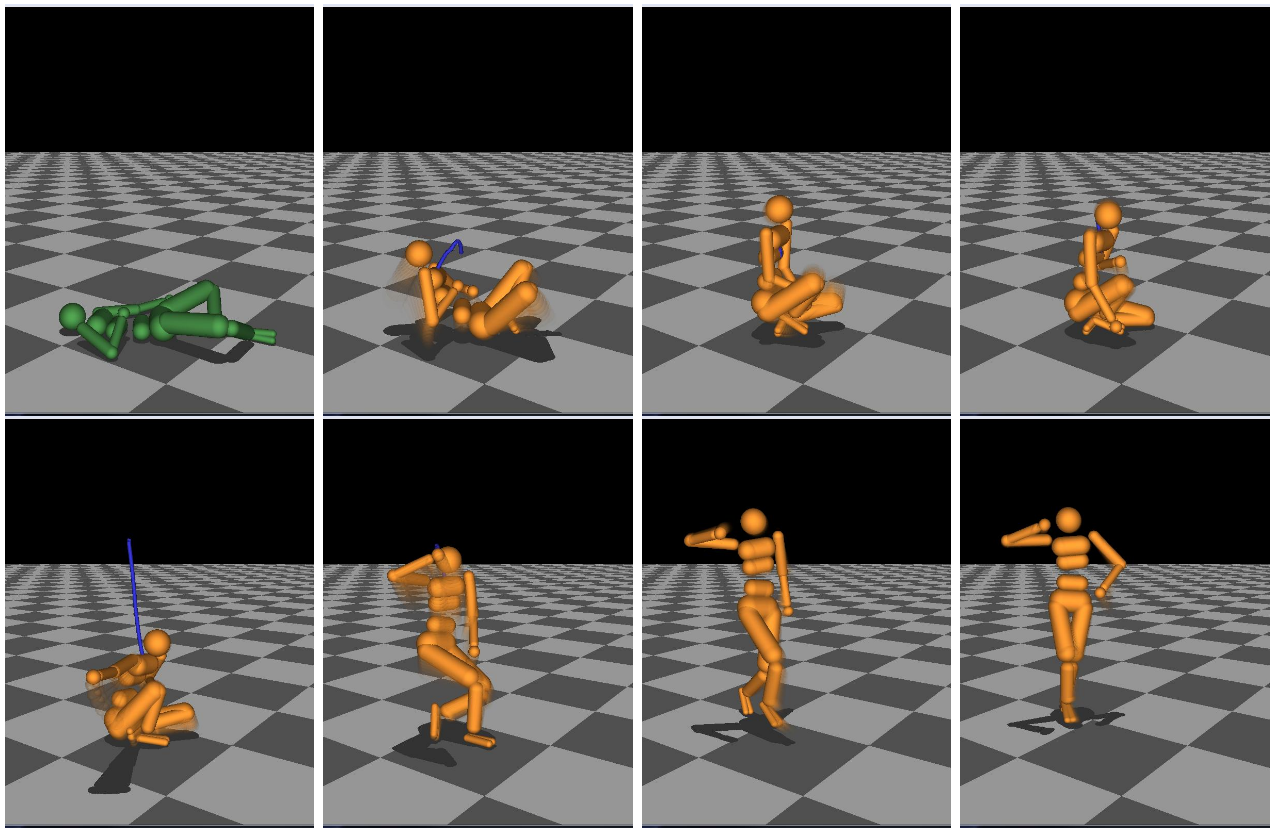


Fig. 5. Getting up. In the first frame the controller is off. The blue line shows the planned position of the torso. This sequence is at 3m50s in the video.

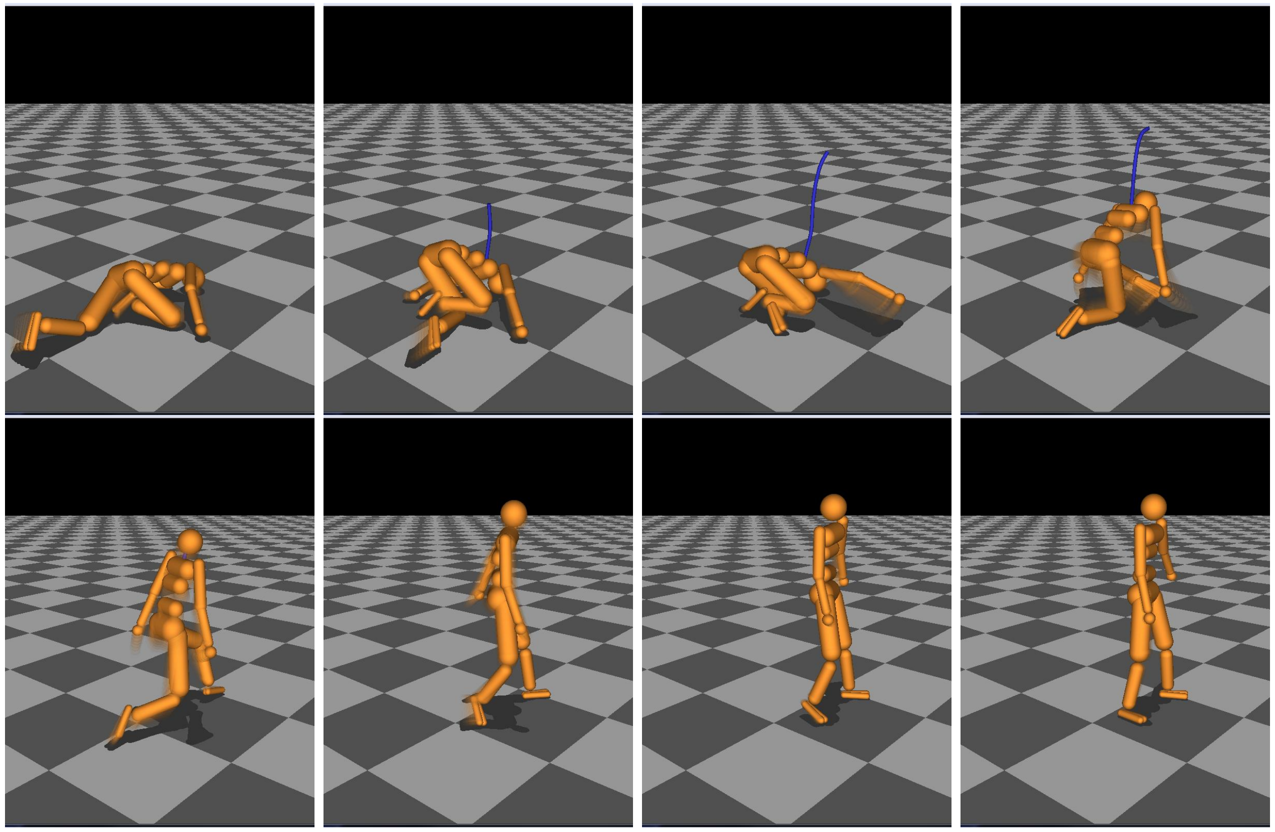


Fig. 6. Getting up from a different initial pose. This sequence is shown at 3m19s in the video.

The fourth state-cost term is a quadratic penalty on the horizontal CoM velocity. We tried several combinations of state-cost terms, all designed so that cost is small when the humanoid is standing still. Though we finally picked the one described above, it was not difficult to find a cost which would promote getting-up, and other variations worked well.

The dynamics of the controller and the simulator were different in two important respects. First, the time step of the simulator was 2ms while the controller's was 8ms. Second, we used $\eta = 0.7$ for the simulator and $\eta = 0.4$ for the controller (see Eq. (17), we used $\beta = 1$ for both models). Both of these choices had the effect of stiffer contact in the plant than in the model. Stiff contact and small time steps in the plant led to realistic contact and friction, while larger time steps and smoother contact for the led to an easier optimization problem for the controller.

The planning horizon was 500 milliseconds. Performance degraded for shorter horizons, but did not qualitatively improve for longer ones.

The most significant fact about our parameter choices was that they were fairly arbitrary. Performance was qualitatively similar for wide range of parameters for both the cost and dynamics. Additionally, the sequences shown in the video and in Figures 5 and 6, were not exceptional or carefully selected. They show the typical performance of the controller.

The maximal policy-lag which admitted a qualitatively reasonable policy was about 20ms. Since the actual iteration time was 140ms, the required slowdown was x 7.

VI. FUTURE DIRECTIONS

One way to speed up MPC is to provide an approximation to the optimal value function, and use it as final cost applied at the horizon. Specifically, this makes it possible to use shorter horizons while avoiding myopic behavior. If the exact optimal value function is available, MPC will return the optimal controls, but in that case we do not it, because one-step greedy optimization recovers the optimal controls given the optimal value function. Better approximations will generally yield better performance. An example of such approximation is computer chess, where simple heuristics for evaluating board configurations are very effective when combined with sufficiently-deep search. A robotics example is [14], where ball-bouncing is achieved via MPC, using a heuristic that specifies what is a good way to hit a ball. Such approximations can clearly help, however they are orthogonal to the issue of efficient trajectory optimization in the MPC context. Therefore in this paper we avoided using informative final costs; instead we simply set the final cost equal to the running cost. Thus the results reported here are in some sense worst-case results, and the performance of our method can be improved by using domain-specific approximations to the optimal value function. Nevertheless we found it useful to focus on this worst case, because it enables us to isolate the trajectory optimization machinery and refine it, and also because we prefer methods that are fully automated and do not rely on a fortuitous guess of the optimal value function.

VII. CONCLUSION

We presented an MPC method applicable to humanoid robots performing complex tasks such as getting up from the ground and rejecting large perturbations. We were able to achieve near-real-time performance on a standard desktop machine, without using any approximations to the optimal value function – which can presumably speed up our method even further. This was possible due to multiple improvements throughout the MPC pipeline, including the trajectory optimization algorithm, the physics engine, and cost function design. With some additional refinements, we believe that our MPC methodology will be applicable to complex humanoid robots. This of course requires a sufficiently accurate dynamics model, which is beyond the scope of the present paper. We show however that our approach is reasonably robust to model errors and state perturbations. How well it will work on physical robots performing different tasks remains to be seen, and the answer may depend on the hardware and task. Nevertheless, having the tools to apply MPC to complex robots is likely to enable many robotic control tasks that are beyond the reach of existing methods for real-time feedback control.

REFERENCES

[1] M. Diehl, H. Ferreau, and N. Haverbeke, "Efficient numerical methods for nonlinear mpc and moving horizon estimation," *Nonlinear Model Predictive Control*, p. 391, 2009.

[2] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, "An application of reinforcement learning to aerobatic helicopter flight," in *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference*, 2007, p. 1.

[3] T. Erez, Y. Tassa, and E. Todorov, "Infinite horizon model predictive control for nonlinear periodic tasks," *Manuscript under review*, 2011.

[4] 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.

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

[6] 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.

[7] ——, "Convergence in unconstrained discrete-time differential dynamic programming," *IEEE Transactions on Automatic Control*, vol. 36, no. 6, p. 692, 1991.

[8] G. Sohl and J. Bobrow, "A recursive multibody dynamics and sensitivity algorithm for branched kinematic chains," *Journal of Dynamic Systems, Measurement, and Control*, vol. 123, p. 391, 2001.

[9] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: a physics engine for model-based control," in *Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[10] Y. Tassa and E. Todorov, "Stochastic complementarity for local control of discontinuous dynamics," in *Proceedings of Robotics: Science and Systems (RSS)*, 2010.

[11] E. Todorov, "Implicit nonlinear complementarity: a new approach to contact dynamics," in *International Conference on Robotics and Automation*, 2010.

[12] ——, "A convex, smooth and invertible contact model for trajectory optimization," in *2011 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2011, pp. 1071–1076.

[13] D. E. Stewart, "Rigid-body dynamics with friction and impact," *SIAM Review*, vol. 42, no. 1, pp. 3–39, Jan. 2000.

[14] P. Kulchenko and E. Todorov, "First-exit model predictive control of fast discontinuous dynamics: Application to ball bouncing," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 2011, p. 21442151.