# Goal Directed Dynamics

Emanuel Todorov
University of Washington and Roboti LLC

***Abstract*— We develop a general control framework where a low-level optimizer is built into the robot dynamics. This optimizer together with the robot constitute a goal directed dynamical system, controlled on a higher level. The high level command is a cost function. It can encode desired accelerations, end-effector poses, center of pressure, and other intuitive features that have been studied before. Unlike the currently popular quadratic programming framework, which comes with performance guarantees at the expense of modeling flexibility, the optimization problem we solve at each time step is non-convex and non-smooth. Nevertheless, by exploiting the unique properties of the soft-constraint physics model we have recently developed, we are able to design an efficient solver for goal directed dynamics. It is only two times slower than the forward dynamics solver, and is much faster than real time. The simulation results reveal that complex movements can be generated via greedy optimization of simple costs. This new computational infrastructure can facilitate teleoperation, feature-based control, deep learning of control policies, and trajectory optimization. It will become a standard feature in future releases of the MuJoCo simulator.**

## I. Introduction

The motivation of this work is to improve robot autonomy, namely the ability to map high-level commands to suitable low-level controls. This is generally what optimal control aims to achieve, but it involves optimization through time which can be hard to solve. In contrast, our approach here is instantaneous and greedy.

We define goal directed dynamics (GDD) as the dynamics of a robot with a layer of control intelligence built into it; see Figure 1. Instead of sending a control signal to the robot at each point in time, we send a *cost function*. This function is defined over accelerations, but later we show how it can also encode spatial goals, desired contact forces, center-of-pressure targets etc. Once this function is specified, the robot does two things at each point in time: (i) find the control signal that minimizes the specified cost; (ii) apply this control signal to the usual dynamics. This simplifies teleoperation where a human can provide goals/costs interactively, as well as facilitates the design of intuitive feature-based controllers. Optimal control also becomes easier when applied to the goal directed dynamics instead of the usual robot dynamics; this is because part of the problem is offloaded to the low-level optimizer.

Goal directed dynamics is the most general instance of an intuitive approach which has a long history. The simplest special case is computed torque control: a reference trajectory specifies the desired acceleration at the current position

This work was supported by NSF grant 1602141 to Roboti LLC. Thanks to Yuval Tassa and Igor Mordatch for comments on the manuscript.

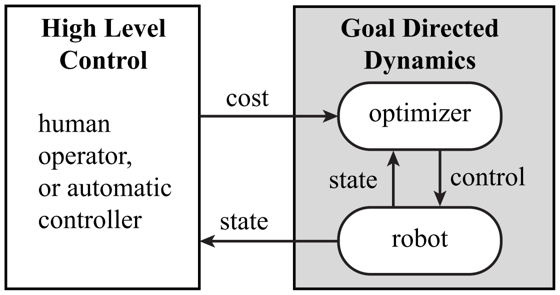

Fig. 1. Illustration of the proposed framework. A human operator or a high-level controller sends goals encoded as cost functions. An optimizer maps these instantaneous goals to control signals. The optimizer together with the robot constitute a goal directed dynamical system.

and velocity, and inverse dynamics are used to compute the necessary control force, assuming full actuation and no constraints. Operational space control [1], [2] is also related. There one defines an end-effector and its corresponding Jacobian, specifies goals in end-effector space, and uses the Jacobian to construct various mappings between end-effector and joint space. Multiple Jacobian-based methods involving differently weighted pseudo-inverses have been developed [3]. That line of work has leveraged the fact that linear algebra and matrix factorization in particular can be performed in real-time, and is still used today. For example, [4] approximate unilateral constraints with equality constraints and reduce the problem to linear algebra.

As computers become faster and numerical algorithms become more efficient, new types of computational problems become tractable in real-time. The current trend is to formulate control schemes that reduce to convex quadratic programs (QP). This has been successful in quadrotor control [5]. It is also used in locomotion, where a QP can jointly compute control torques and contact forces such that the center-of-mass accelerates according to some high-level plan while the contact forces remain in the friction cone. The cone is approximated with a pyramid whose faces become linear inequality constraints in the QP. Another application is manipulation, where a QP can compute control torques and contact forces that stabilize an object in the hand. Multiple variations on this theme have recently been explored [6], [7], [8], [9]. There is however an important limitation in such control schemes: in order to keep the QP convex, the complementarity condition in the LCP formulation of frictional contact [13], [14] is ignored, or alternatively the friction components are ignored (recall that the LCP describing frictional contact is equivalent to a non-convex QP which is

NP-hard). Because of this simplification, the solver is free to choose any contact force in the friction cone, even if it is very different from the actual contact force that results from the control torques being applied. Thus treating contact forces as independent decision variables (as in convex QP) makes little physical sense. Our framework removes this limitation, in a way that is numerically efficient. Note that if we are not concerned with numerical efficiency, it is straightforward to throw every desirable constraint into a general-purpose optimizer and hope that it finds a reasonable solution in a reasonable amount of time. However such hope is not justified in the general case of non-convex optimization.

A similar criticisms of the QP approach was recently presented in [12], and indeed that work is the most closely related to ours in the Robotics literature. These authors focused on robots in multiple contacts with the environment but under static conditions. They took into account the actual contact forces resulting from the control torques, using a spring-damper contact model, and were able to stabilize the system by solving a convex optimization problem. However it is not clear how to extend this work to dynamic conditions, given that spring-damper contact models have well-known difficulties and have been largely superseded by modern linear complementarity problem (LCP) solvers [13], [14]. Our agenda here is related to [12] but the techniques are different and apply to dynamic settings. This is possible thanks to a new formulation of the physics of soft contacts developed in [15], [16] and implemented in MuJoCo [17]. It combines the stability and long time steps of LCP-type solvers, with the softness and analytical advantages of spring-damper contacts.

As already alluded to, if we want to develop an instantaneous control scheme that is more powerful than the current QP-based schemes, we have no choice but to solve a harder optimization problem in real-time. QPs are by no means the limit of what modern computers can do. LCP-type solvers are an existence proof of this. Such solvers have been available in multiple physics engines for over a decade, and the problem they solve at each time step (albeit approximately) is an NP-hard non-convex optimization problem. While the soft contact model in [16] reduces to convex optimization in the case of forward dynamics, the goal directed dynamics developed here require solving a non-convex optimization problem.

Related non-convex optimization problems have been studied in Computer Graphics, in the context of controlling high-level features [18], [19], [20], [21], [22]. These features (corresponding to goals in our terminology) are varied online, using either hand-crafted or numerically optimized high-level controllers. The available optimization schemes however do not have robustness and performance comparable to the QP approach. In general it is difficult to think of them as giving rise to a new dynamical system that can be readily simulated in place of the usual forward dynamics. This is because the problem is formulated as generic non-convex optimization, without special structure that can enable efficient solvers. Indeed the emphasis in that literature has been not so much on efficient optimization, but rather on design of cost functions whose instantaneous optimization yields interesting behaviors. This is very much complementary to our work. Existing cost functions that are already known to work well can be plugged in our framework and work even better.

### A. Alternative view: Combining physics and control

An alternative way to understand GDD is to think of it as combining physics and control in a single computation. This is appealing because forward dynamics, i.e. computing the acceleration $\dot{v}$ resulting from control force $\tau$, already require some form of numerical optimization when unilateral constraints are present [13], [14], [16]:

$$\begin{array}{rll} \dot{v}(\tau) = & \arg\min_a & \text{physics}(a,\tau) \\ & s.t. & \text{constraint}(a,\tau) \end{array} \tag{1}$$

One way to combine physics and control is to simply add a task-related term to the objective, and obtain both the acceleration and the (one-step-optimal) control force by solving a single optimization problem:

$$\begin{array}{rll} (\dot{v},\tau) = & \arg\min_{a,t} & \text{physics}(a,t) + \text{task}(a,t) \\ & s.t. & \text{constraint}(a,t) \end{array} \tag{2}$$

This is essentially what the QP approach does. The problem however is that the result is no longer consistent with the physics. Instead, in GDD we start with nested optimization which is physically-consistent by construction:

$$\begin{array}{rll} (\dot{v},\tau) = & \arg\min_{a,t} & \text{task}(a,t) \\ & s.t. & a = \arg\min_b \text{physics}(b,t) \\ & & s.t.\ \text{constraint}(b,t) \end{array} \tag{3}$$

Our key insight then is to exploit MuJoCo's inverse dynamics – which enables us to avoid the above nested optimization, and yields an efficient yet physically-consistent GDD solver.

## II. General framework

We begin with a general definition of GDD, and later specialize it to rich yet computationally-tractable systems. Consider the second-order system

$$\begin{array}{ll} \text{configuration:} & q \in \mathcal{R}^m \\ \text{velocity:} & v \in \mathcal{R}^n \\ \text{applied force:} & \tau = (u,z) \in \mathcal{R}^n \\ \text{control force:} & u \in \mathcal{U} \subseteq \mathcal{R}^k \\ \text{root force:} & z \in \mathcal{R}^{n-k} \\ \text{forward dynamics:} & \dot{v} = f\left(q,v,u\right) \\ \text{inverse dynamics:} & \tau = g\left(q,v,\dot{v}\right) = \left(g_u, g_z\right) \\ \text{inertia matrix:} & M(q) \in \mathcal{R}^{n\times n} \end{array} \tag{4}$$

The dimensionality $n$ of the velocity and applied force vectors equals the number of degrees of freedom (DOF). We assume for simplicity that actuators are in one-to-one correspondence with scalar joints, and partition the DOF space into actuated $(u)$ and unactuated $(z)$ subspaces. This is what the bracket notation $\tau = (u,z)$ and $g = (g_u, g_z)$ means. The more general approach would be to write $\tau = B(q,v)u$ for a generic control gain matrix $B$. Our setup corresponds to $B = (I_{k\times k}\ \ 0_{k\times(n-k)})^T$.

The time-derivative $\dot{v}$ denotes the actual acceleration of the system. Below we will also use $a$ to denote hypothetical accelerations, which are optimization variables in various cost functions. The reason for using $v$ instead of $\dot{q}$ is because $q$ is a point on the configuration manifold while $v$ is a vector in the tangent space to this manifold. The orientations of floating bases and free-moving objects are best represented with quaternions, in which case $\dim(q) = 4$ while $\dim(v) = 3$ per joint, and so $m > n$. When the system is under-actuated, we also have $k < n$. Most interesting systems have these two properties. For example, a humanoid robot has $m = n + 1$ and $k = n - 6$. The same holds for a fixed-base arm manipulating one object. For a humanoid robot manipulating two objects, $m = n + 3$ and $k = n - 18$.

An important assumption implicit in (4) is that the inverse dynamics $g(q, v, \dot{v})$ exist and are uniquely defined. This is not the case for systems with hard state constraints. But as we will see later, we can use soft constraints to obtain realistic behavior with uniquely-defined inverse dynamics.

The set of admissible controls $\mathcal{U}$ is usually a box. The set of feasible accelerations $\mathcal{A}$ is more difficult to characterize. It can be defined through the forward dynamics:

$$\mathcal{A}(q, v) = \{a \in \mathcal{R}^n : \exists u \in \mathcal{U} \text{ s.t. } f(q, v, u) = a\} \tag{5}$$

or through the inverse dynamics:

$$\mathcal{A}(q, v) = \{a \in \mathcal{R}^n : g_u(q, v, a) \in \mathcal{U},\ g_z(q, v, a) = 0\} \tag{6}$$

These two definitions are mathematically equivalent. We will use the latter to obtain efficient algorithms.

Let $\ell(\cdot)$ denote the scalar cost over accelerations used to specify the instantaneous goal. The space of such functions is infinite-dimensional, so we cannot directly use them as high-level controls, but we will choose suitable finite-dimensional parameterizations later.

Now we can define the goal directed dynamics in two ways, using the forward dynamics or the inverse dynamics of the original system. Both definitions correspond to forward GDD. They are mathematically equivalent, but again the inverse definition yields more efficient algorithms. In both cases the state $(q, v)$ as well as the cost $\ell(\cdot)$ are given, and we seek to define the goal directed acceleration $\dot{v}$. The forward definition is:

$$\dot{v} = f\left(q, v, \arg\min_{u \in \mathcal{U}} \{\|(u, 0)\|_R + \ell(f(q, v, u))\}\right) \tag{7}$$

while the inverse definition is:

$$\dot{v} = \arg\min_{a \in \mathcal{A}(q,v)} \{\|g(q, v, a)\|_R + \ell(a)\} \tag{8}$$

We have added a regularizing cost on the applied forces, in addition to imposing actuation constraints. This regularizing cost is the weighted $L_2$ norm

$$\|\tau\|_R \equiv \frac{1}{2}\tau^T R \tau \tag{9}$$

The weighting matrix $R$ can in principle be any s.p.d. matrix. Here we use $R = M^{-1}$. The rationale for weighting by the inverse inertia is to match units in mixed cost functions penalizing both forces and accelerations. From Newton's second law $f = ma$ we have $f^2 m^{-1} = a^2 m$, suggesting that forces should be weighted by $M^{-1}$ while accelerations should be weighted by $M$. More formally, the inertia $M(q)$ is the metric tensor over the configuration manifold parameterized by $q$. It is used to compute dot-products for tangent vectors (velocities and accelerations), while its inverse is used to compute dot-products for co-tangent vectors (moments and forces).

The constraint $g_z = 0$ in the inverse definition (8) removes infeasible accelerations which cannot be generated by the actuators. The external forces needed to generate such infeasible accelerations are sometimes called 'root forces' or 'magic forces'. They may be beneficial for task achievement if we could somehow generate them. Thus when optimization is used in a continuation regime (i.e. starting with an easier problem and gradually transitioning towards a harder problem while tracking the solution), it is often useful to allow infeasible accelerations early on and suppress them later. The augmented Lagrangian primal-dual method we adopt later does that automatically.

*Example: end-effector control with full actuation*

We now illustrate our general framework with an example where the above optimization problem can be solved in closed form. Let $M(q)$ denote the configuration-dependent inertias as before, and $c(q, v)$ the vector of centripetal, Coriolis, gravitational and any other control-independent forces. Since the system here is fully actuated, we have $u = \tau$ and $z = \varnothing$. The inverse dynamics are

$$\tau = M(q)\,\dot{v} + c(q, v) \tag{10}$$

Since $M$ is always s.p.d. the above equation can be solved for $\dot{v}$ to obtain the corresponding forward dynamics.

Let our end-effector have Jacobian $J_e(q)$ and desired acceleration $a_e^*$ in end-effector space. The simplest cost function for specifying this acceleration goal is

$$\ell(a) = \frac{1}{2}\|J_e a - a_e^*\|^2 \tag{11}$$

Substituting in either the forward or inverse definition for the general case and assuming no actuation limits, we obtain the following acceleration for the GDD:

$$\dot{v} = \left(M + J_e^T J_e\right)^{-1}\left(J_e^T a_e^* - c\right) \tag{12}$$

This is an instance of a Jacobian pseudo-inverse. Note that we could have chosen a different $L_2$ norm in end-effector space, resulting in a differently weighted pseudo-inverse. The weighting consistent with our unit-matching approach in this case is given by the inverse end-effector inertia $J_e M^{-1} J_e^T$.

We could extend this example to under-actuated systems and still obtain an analytical solution. As long as the root force $g_z(q, v, a)$ is linear in $a$, it will add a linear equality constraint to the quadratic optimization problem, resulting in a differently weighted pseudo-inverse. Real robots however are subject to contacts, joint limits and dry friction. These phenomena make it impossible to solve our optimization problem analytically because the relationship between force and acceleration becomes nonlinear and piece-wise smooth. We turn to such systems next.

## III. Specialization to MuJoCo physics

We now apply the above framework to dynamical systems that can be modeled in the MuJoCo simulator [17]. It simulates multi-joint dynamics in joint coordinates, using a soft-constraint model [16] that can handle frictional contacts, joint and tendon limits, dry friction in joints and tendons, and a variety of equality constraints. Unlike LCP-type solvers, MuJoCo's constraint solver reduces to unconstrained convex optimization in forward dynamics. Importantly, the inverse dynamics are uniquely defined and computed analytically, making the inverse formulation of GDD (8) more appealing than the forward formulation (7).

We now provide a brief summary of MuJoCo's physics model, emphasizing those aspects that are relevant to GDD. Forward dynamics in MuJoCo are defined starting with the Gauss principle of least constraint [24], and extending it with a constraint violation term as follows:

$$\dot{v} = \arg\min_{a} \left\{ \|Ma + c - \tau\|_{M^{-1}} + s\left(Ja - r\right) \right\} \tag{13}$$

Here $s(\cdot)$ is a convex $C^1$ function which softly penalizes acceleration constraint violations [17], $J(q)$ is the constraint Jacobian, and $r(q, v)$ is a reference acceleration in constraint space which is computed from a virtual spring-damper used for constraint stabilization. This is related to Baumgarte stabilization [25], except here $r$ modifies the acceleration target for the optimizer instead of applying a force directly. The function $s(\cdot)$ is a quadratic spline when friction cones are approximated as pyramids. Thus its gradient $\nabla s$ is piece-wise affine and continuous, while its Hessian $H[s]$ is piece-wise constant and discontinuous.

Compared to the velocity-stepping schemes used in most other modern simulators, our formulation has several unique advantages: (i) the optimization problem is unconstrained; (ii) the optimization problem is convex; (iii) the forward dynamics are defined in continuous time; (iv) the inverse dynamics are uniquely-defined and computed analytically; see eq. (14). The summary provided here is not sufficient to understand how all of this is possible, and why we are able to side-step hard computational problems that have presented obstacles in prior work on contact dynamics. The MuJoCo simulation framework has taken us many years of research and product development, as described in detail elsewhere [15], [16], [17]. Our goal here is not to re-introduce the framework, but rather to leverage its unique features in solving control problems.

Since problem (13) is convex and unconstrained, it has a unique minimizer which makes the gradient vanish. This yields the identity

$$\tau = M\dot{v} + c + J^T \nabla s\left(J\dot{v} - r\right) \tag{14}$$

We can now recognize $-\nabla s$ as the constraint force. Note that for given $\dot{v}$ we have an analytical formula for $\tau$, corresponding to inverse dynamics. In forward dynamics on the other hand, we are given $\tau$ and computing $\dot{v}$ requires solving the system of non-linear equations (14), or equivalently solving optimization problem (13), both of which require a numerical method.

We can further differentiate the inverse dynamics and obtain its Jacobian with respect to acceleration, denoted $P$:

$$P \equiv \frac{\partial g}{\partial a} = M + J^T H\left[s\right] J \tag{15}$$

So the Jacobian of the inverse dynamics turns out to be an s.p.d. matrix. This is because it is also the Hessian of the convex objective in the extended Gauss principle (13) without the term $\tau$. Having an analytical Jacobian, without need for sampling or finite difference approximations, speeds up the optimization required for GDD.

With these mathematical preliminaries regarding MuJoCo physics, let us now apply the inverse formulation of GDD (8). Leaving out the constraints on $a$ for the moment, the objective function becomes

$$L(a) = \|g(a)\|_R + \ell(a) \tag{16}$$

The gradient of this function is

$$\nabla L = PRg + \nabla \ell \tag{17}$$

and the Gauss-Newton approximation to the Hessian is

$$H[L] = PRP + H[\ell] \tag{18}$$

We now have all the ingredients needed to construct an efficient optimizer. The goal is to minimize $L(a)$ subject to the constraints in the inverse formulation, namely $g_u(a) \in \mathcal{U}$ and $g_z(a) = 0$. The former is usually a box constraint specifying actuation limits. This combination of constraints is well-suited for augmented Lagrangian primal-dual methods [26].

It is notable that we did not have to impose friction cone constraints in the optimization. This is because the inverse dynamics in MuJoCo's physics model are such that for any acceleration $a$, the constraint force $-\nabla s\left(Ja - r\right)$ automatically satisfies friction cones and any other applicable constraints. Indeed if the controls $u$ are not bounded, the only constraint left in our GDD optimization problem (8) is the equality constraint $g_z(a) = 0$.

## IV. Relation to dynamic programming

We defined GDD as the solution to the equivalent optimization problems (7) and (8). Similar optimization problems also arise in dynamic programming, suggesting new algorithms for approximate dynamic programming based on GDD. To see the similarity, recall the Bellman equation for the optimal value function. To update the value at each state, we must find the control that optimizes control cost plus the value of the resulting next state. If we think of the value function as encoding the goal (thereby replacing the generic cost $\ell(a)$ we use to provide high-level commands), the Bellman update is doing what we propose to do. So we are essentially incorporating a Bellman-update-like operator in the robot dynamics. Next we develop this idea more formally.

Consider the discrete-time version of our second-order control system (4), namely:

$$\begin{aligned} q_{t+h} &= q_t + hv_t \\ v_{t+h} &= v_t + hf(q_t, v_t, u_t) \end{aligned} \tag{19}$$

Define the running cost as

$$\|(u,0)\|_R + p(q,v) \tag{20}$$

The first term is our previous regularization cost, while the second term is a new state-dependent cost that can be used to specify desirable states. The latter corresponds to the running cost in optimal control formulations.

Let $V^*(q,v)$ denote the optimal value function for the above optimal control problem (first-exit formulation). Then $V^*$ satisfies the Bellman equation

$$\begin{aligned} V^*(q,v) = \quad & p(q,v) + \\ & \min_{u \in \mathcal{U}} \|(u,0)\|_R + V^*(q + hv, v + hf(q,v,u)) \end{aligned} \tag{21}$$

The Bellman equations for finite horizon, average cost and discounted formulations have similar structure. Note that the minimization in (21) is identical to the minimization in the forward definition of GDD (7) when the goal-setting cost $\ell(\cdot)$ is defined as

$$\ell(a) = V^*(q + hv, v + ha) \tag{22}$$

This of course is not a coincidence; the optimal value function has the key property that the policy which is greedy with respect to it is the optimal policy. Conversely, if we could somehow guess $\ell(\cdot)$ that takes into account the long-term value of the state, the goal directed dynamics will match the optimally controlled dynamics.

Thus far the relation between goal-directed dynamics and dynamic programming is intuitive but does not seem to suggest new algorithms. Things get more interesting however when we turn to the inverse formulation. We will now redefine the optimal control problem as follows. Instead of using the control $u$ to drive the system, we will use the acceleration $a$. So the control system becomes

$$\begin{aligned} q_{t+h} &= q_t + hv_t \\ v_{t+h} &= v_t + ha_t \end{aligned} \tag{23}$$

When $q$ contains quaternions, the summation $q + hv$ denotes integration over the 4D unit sphere; this is the only remaining nonlinearity, and it is kinematic rather than dynamic. The running cost is

$$\|g(q,v,a)\|_R + p(q,v) \tag{24}$$

This is the same as the previous definition (20) but has been expressed as a function of acceleration.

We can now write down the Bellman equation for the inverse dynamics (or acceleration-based) formulation of optimal control:

$$\begin{aligned} V^*(q,v) = \quad & p(q,v) + \\ & \min_{a \in \mathcal{A}(q,v)} \|g(q,v,a)\|_R + V^*(q + hv, v + ha) \end{aligned} \tag{25}$$

This is mathematically equivalent to the Bellman equation (21) for the forward dynamics formulation of optimal control, meaning that the two equations characterize the same optimal value function. But as in GDD, the inverse formulation offers algorithmic advantages, in particular when it comes to trajectory optimization methods such as differential dynamic programming (DDP) [27] and iterative linear-quadratic-Gaussian control (iLQG) [28]. Such methods maintain a local quadratic approximation to $V^*$ which is propagated backwards in time through the linearized (in the case of iLQG) dynamics. But in the inverse dynamics formulation, our control system (23) became linear! Therefore propagation errors due to dynamics linearization will not accumulate. This of course is not a free lunch: what we did is to hide the complexity of the nonlinear dynamics in the cost over accelerations. So when we approximate this cost with a quadratic, the approximation error is likely to be greater compared to approximating a traditional control cost. However, control costs are often used to tame the optimizer and produce sensible behavior, and not because we care about the specific expression; for example the quadratic costs used in practice do not correspond to power or any other physically meaningful quantity. This is why we refer to them as regularization costs and control costs interchangeably. So if we could put all approximation errors in one place, the control cost is the ideal place. Another advantage in the context of MuJoCo physics is that we are working with inverse dynamics, which are computed analytically unlike forward dynamics. We have not yet developed a specific algorithm exploiting this new inverse formulation of optimal control, although we already have a name for it: acceleration-based iterative LQR (AILQR). We are looking forward to developing this algoritghm in future work.

## V. Cost function design

Coming back to the instantaneous optimization in GDD, we need a cost function $\ell(\cdot)$. We already discussed one example, namely the end-effector acceleration cost (11). This was used in an analytically-solvable special case, but it can be used in the general case as well. Another setting where we can readily obtain a desired acceleration in joint space, and construct a cost $\ell(\cdot)$ around it, are policies that are trained to output acceleration targets. For example, [23] trained such a neural network controller capable of stable 3D locomotion to spatial targets specified interactively.

Recall that the constraint forces in the MuJoCo physics model we are using are computed analytically given the acceleration: the vector of all such forces is $-\nabla s(Ja - r)$ where $J, r$ are fixed given the state $(q,v)$. This includes frictional contact forces, joint limit forces, dry friction forces and equality constraint forces. Therefore all these quantities can be used to construct cost functions for the purpose of GDD. For example, the center of pressure can be computed by identifying all contact points between the robot and the ground plane, and weighting their positions by the corresponding contact normal forces.

Spatial goals can be specified by computing a desired end-effector acceleration and using a cost similar to (11). This desired acceleration can be obtained from a virtual spring-damper, or perhaps a minimum-jerk spline which will likely result in smoother and more human-like movements [29].

Finally, as already summarized earlier, there is a rich literature in both Robotics and Graphics proposing to optimize various quantities instantaneously. These include center of pressure, ZMP and related 'capture points' [9], [10], [11], as well center of mass, linear and angular momentum, and many other intuitive features that have proven useful. Our goal here is not to design new costs, but rather to develop the computational infrastructure which can be used to optimize efficiently any cost in this broad family, without the modeling limitations imposed by the QP framework.

## VI. Numerical benchmarks

Here we describe simulation results from two MuJoCo models: a humanoid and a disembodied hand; see Figure 2 and the accompanying movie. In each case we selected a body segment and used the mouse to interactively specify a spatial goal. The cost $\ell(\cdot)$ had two terms. The first term was an end-effector acceleration term for the selected body segment, as in (11), specifying a desired acceleration towards the spatial goal. The second term was

$$\frac{\alpha}{2} \left\| a + \beta v \right\|_R \tag{26}$$

where $\alpha$ is the relative weight of the cost term and $\beta$ is a virtual damping coefficient. This tells the GDD to stop moving, resembling a joint-space damping mechanism. In this way, once the first term drives a given body segment to a desired location, it can be disabled and the second term makes GDD hold it in place – although with this particular cost it slowly falls under gravity, because we also have a cost on $\tau$.

We used the interactive simulation described above to collect a dataset of records, each containing $(q, v, a, \tau)$ for one model and one point in time. We then re-run the simulations offline, disabling rendering and interaction so that we could time the solvers more accurately. The dataset for each model had $20,000$ records. We tested performance on an Intel i7-6700K 4 GHz processor. The average CPU times per step in a single thread were as follows:

| **CPU time per step** ($\mu s$) | **humanoid** | **hand** |
|---|---|---|
| forward | 49 | 128 |
| goal-directed | 90 | 182 |
| goal-directed + forward | 99 | 194 |

Note that these times are in microseconds. So in all cases it takes well below a millisecond to execute the GDD solver to convergence (to a feasible but possibly local minimum). The GDD solver is only two times slower than the forward solver, which is impressive given that it is solving a non-convex and non-smooth problem. Even though GDD computes both the control and the acceleration, the control

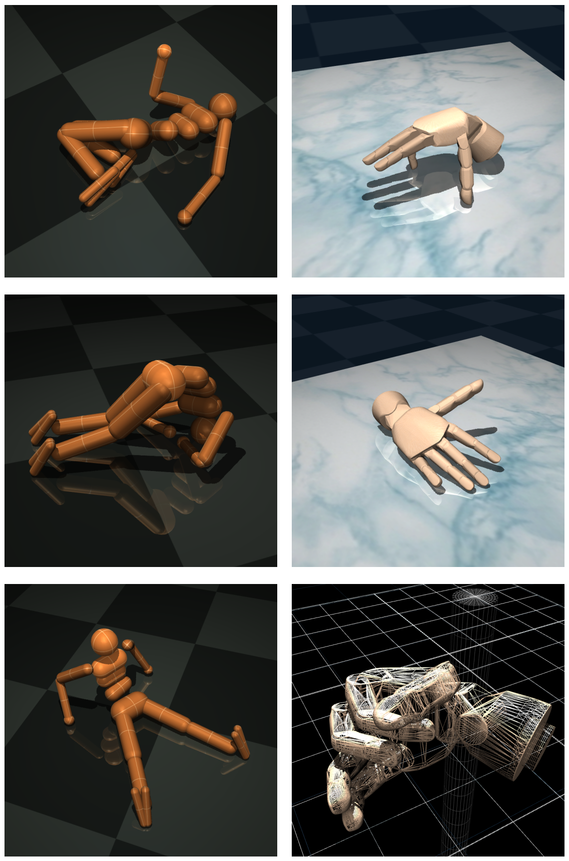
Fig. 2. Physics models used to benchmark the GDD solver.

may not be exactly feasible, since we are using a primal-dual method. Therefore we clamp the control to the feasible set and execute forward dynamics. This clamping operation introduces a very small change and the forward solver is warmstarted with the acceleration found by GDD, thus the forward solver converges almost immediately and adds only 10 microseconds on average (last line in the above table).

Next we present statistics characterizing the behavior of the GDD solver over the $20,000$ simulation step for each test model. For each statistic we show the 50th percentile (i.e. the median) and the 95th percentile:

| **satistic** (50% (95%)) | **humanoid** | **hand** |
|---|---|---|
| constraints | 23 (38) | 19 (30) |
| iterations | 4 (9) | 5 (9) |
| dual updates | 1 (1) | 1 (3) |
| physics violation | 3e-11 (6e-7) | 2e-8 (5e-7) |
| residual gradient | 1e-7 (1e-6) | 1e-9 (8e-9) |

The number of constraints reflects the number of active contacts and joint limits, and corresponds to the number of rows in the Jacobian $J$. Each iteration involves a Newton step to find the search direction, followed by exact line-search exploiting the structure of the objective function. Dual updates are needed to re-estimate the Lagrange multipliers for the constrained optimization problem. The last two rows characterize the quality of the solution. The bottom line is that both the physics violation error and the residual gradient are very small, and the number of iterations is also very small considering the problem class.

## VII. Conclusions and future work

We developed a general control framework using cost functions in place of traditional control signals. The embedded GDD solver computes the optimal controls given the instantaneous cost/goal, and applies them to the physical system. This solver is sufficiently fast and robust to be used in a low-level control loop, generalizing the QP solvers that are currently popular in robotics. While some illustrations of behaviors generated by our new solver can be found in the accompanying movie, our objective here was not to explore what cost functions are needed to generate interesting behaviors. This has been done extensively before. Instead we focused on developing a more general optimization framework that can make prior cost functions work better.

In future work we will experiment with more elaborate costs and applications to physical systems. We will also develop the new trajectory optimizer outlined earlier, using GDD in an inner loop and replacing the user-defined instantaneous cost in GDD with an approximation the optimal value function. Another possible application of GDD is in the context of model-based Reinforcement Learning, where it can be used to optimize over actions given an approximation to the optimal value function. GDD can also facilitate policy gradient methods: learn a controller that outputs desired accelerations rather than forces, and then use GDD online to compute the corresponding forces and execute the controller.

## References

[1] O. Khatib, A unified approach for motion and force control of robot manipulators: The operational space formulation. IEEE Journal on Robotics and Automation, 3: 43-53, 1987.

[2] L. Sentis, J. Park, O. Khatib, Compliant control of multicontact and center-of-mass behaviors in humanoid robots. IEEE Transactions on Robotics 26: 483-501, 2010.

[3] R. Featherstone, O. Khatib, Load-independence of the dynamically-consistent inverse of the Jacobian matrix. International Journal of Robotics Research, 16: 168-170, 1997.

[4] L. Righetti, J. Buchli, M. Mistry, M. Kalakrishnan, S. Schaal, Optimal distribution of contact forces with inverse-dynamics control. International Journal of Robotics Research, 32: 280-298, 2013.

[5] D. Mellinger, N. Michael, V. Kumar, Trajectory generation and control for precise aggressive maneuvers with quadrotors. International Journal of Robotics Research, 31: 664-674, 2012.

[6] A. Escande, N. Mansard, P. Wieber, Hierarchical quadratic programming: Fast online humanoid-robot motion generation. International Journal of Robotics Research, 33: 1006-1028, 2014.

[7] S. Feng, W. Whitman, X. Xinjilefu, C. Atkeson, Optimization-based full body control for the DARPA Robotics Challenge. Journal of Field Robotics, 32: 293-312, 2015.

[8] S. Kuindersma, R. Deits, M. Fallon, A. Valenzuela, H. Dai, F. Permenter, T. Koolen, P. Marion, R. Tedrake, Optimization-based locomotion planning, estimation, and control design for the Atlas humanoid robot. Autonomous Robots, 40: 429-455, 2016.

[9] T. Koolen, T. De oer, J. Rebula, A. Goswami, J. Pratt, Capturability-based analysis and control of legged locomotion. Part 1: Theory and application to three simple gain models. International Journal of Robotics Research, 2012.

[10] P. Saradain, G. Bessonnet, Forces acting on a biped robot. Center of pressure – zero moment point. IEEE Trans. Systems, Man, and Cybernetics, Part A, 34: 630637, 2004.

[11] M. Vukobratovic, B. Borovac, Zero-moment point – Thirty five years of its life. International Journal of Humanoid Robotics, 1: 157173, 2004.

[12] E. Farnioli, M. Gabiccini, A. Bicchi, Optimal contact force distribution for compliant humanoid robots in whole-body loco-manipulation tasks. ICRA 2015.

[13] D. Stewart, J. Trinkle, An implicit time-stepping scheme for rigid-body dynamics with inelastic collisions and coulomb friction. International Journal Numerical Methods Engineering, 39: 2673-2691, 1996.

[14] M. Anitescu, F. Potra, D. Stewart, Time-stepping for three dimensional rigid body dynamics. Computer Methods in Applied Mechanics and Engineering, 177: 183-197, 1999.

[15] E. Todorov, T. Erez, Y. Tassa, MuJoCo: A physics engine for model-based control. International Conference on Itelligent Robots and Systems (IROS) 2012.

[16] E. Todorov, Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in MuJoCo. ICRA 2014.

[17] E. Todorov, MuJoCo: Modeling, Simulation and Visualization of Multi-Joint Dynamics with Contact. Seattle WA: Roboti Publishing, 2016. `www.mujoco.org/book`

[18] Y. Abe, M. da Silva, J. Popovic, Multiobjective control with frictional contacts. ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2007.

[19] M. de Lasa, I. Mordatch, A. Hertzmann, Feature-based locomotion controllers. SIGGRAPH 2010.

[20] M. Al Borno, M. de Lasa, A. Hertzmann, Trajectory optimization for full-body movements with complex contacts. IEEE Transactions on Visualization and Computer Graphics, vol 19, 2013.

[21] Y. Lee, M. Park, T. Kwon, J. Lee, Locomotion control for many-muscle humanoids. SIGGRAPH 2016.

[22] A. Macchietto, V. Zordan, C. Shelton, Momentum control for balance. SIGGRAPH 2009.

[23] I. Mordatch, K. Lowrey, G. Andrew, Z. Popovic, E. Todorov, Interactive control of diverse complex characters with neural networks. NIPS 2015.

[24] R. Kalaba, H. Natsuyama, F. Udwadia, An extension of Gauss's principle of least constraint. International Journal of General Systems, 33: 63-69, 2004.

[25] J. Baumgarte, Stabilization of constraints and integrals of motion in dynamical systems. Computer Methods in Applied Mechanics and Engineering, 1: 1-16, 1972.

[26] A. Forsgren, P. Gill, Primal-dual interior methods for nonconvex nonlinear programming. SIAM J Optim 8: 1132-1152, 1998.

[27] D. Jacobson, D. Mayne, Differential Dynamic Programming. Elsevier Publishing Company, New York, 1970.

[28] E. Todorov, W. Li, A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. American Control Conference, 2005.

[29] T. Flash, N. Hogan, The coordination of arm movements: An experimentally confirmed mathematical model. Journal of Neuroscience, 5: 1688-1703, 1985.