Physics-based control

Emo Todorov

Roboti LLC
University of Washington

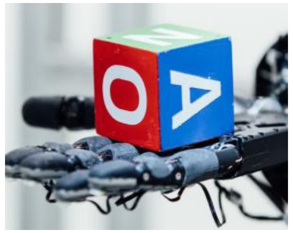
Physics-based control works on real systems

Abbeel, Coates and Ng, *IJRR* 2010

Williams et al, *ICRA* 2016

nominal model

Mordatch, Lowrey and Todorov, *IROS* 2015



OpenAI, 2018

randomization

Computing physics derivatives

Analytical derivatives:

ideal, yet difficult to implement
only need to be implemented once (like writing an OS)

Automatic differentiation:

good idea for smooth dynamics
not a good idea with collision and constraint solvers (iterative)

Finite differences:

this is what people should do in the absence of analytical derivatives
alternative: sampling + linear regression (less accurate, maybe more robust)

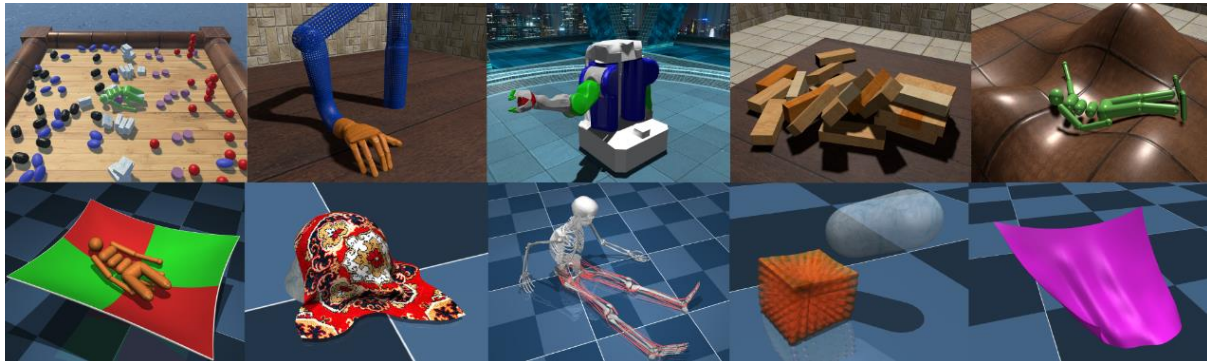
Learning generic differentiable models (NNs etc):

speed and generalization will not be comparable to a physics-based model
if we ignore generalization, we might as well fit time-varying linear models

Kumar, Todorov and Levine, *ICRA* 2016

MuJoCo physics

q	configuration
v	velocity
τ	applied force
$c\left(q,v\right)$	internal force
$M\left(q\right)$	inertia matrix
$J\left(q\right)$	constraint Jacobian
$\lambda\left(q,v,\tau/\dot{v}\right)$	constraint force



model	evals / s 20 threads
humanoid	300,800
humanoid100	17,100
hammock	40,400
particle	7,150
grid2	19,350
ellipsoid	31,600

10-core processor

Forward dynamics: convex optimization

$$\dot{v} = \arg\min_{a} \left\| a + M^{-1}\left(c-\tau\right)\right\|_M^2 + s\left(Ja - r\right)$$

Inverse dynamics: analytical solution

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

Analytical derivatives of inverse dynamics

$$\tau\left(q, v, a\right) = M\left(q\right) a + c\left(q, v\right) + J\left(q\right)^T \nabla s\left(J\left(q\right) a - r\left(q, v\right); D\left(q\right)\right)$$

exact derivative

treated as constant for now

CPU time, humanoid, 50 timesteps per core

```
cost only:                1.5 ms
cost + analytical:        7.5 ms
cost + one-sided FD:     48.0 ms
```

Derivatives of forward dynamics

inverse $\tau\left(q, v, a\right)$

forward $a\left(q, v, \tau\right)$

$$\frac{\partial a}{\partial \tau} = \left(\frac{\partial \tau}{\partial a}\right)^{-1}$$

$$\frac{\partial a}{\partial v} = -\left(\frac{\partial \tau}{\partial a}\right)^{-1} \frac{\partial \tau}{\partial v}$$

$$\frac{\partial a}{\partial q} = -\left(\frac{\partial \tau}{\partial a}\right)^{-1} \frac{\partial \tau}{\partial q}$$

GDD: Goal Directed Dynamics

Todorov, *ICRA* 2018

Force partitioning: $\tau = (u, z)$

Forward dynamics: $\dot{v} = f(q, v, \tau)$

Inverse dynamics: $\tau = g(q, v, \dot{v}) = (g_u, g_z)$

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

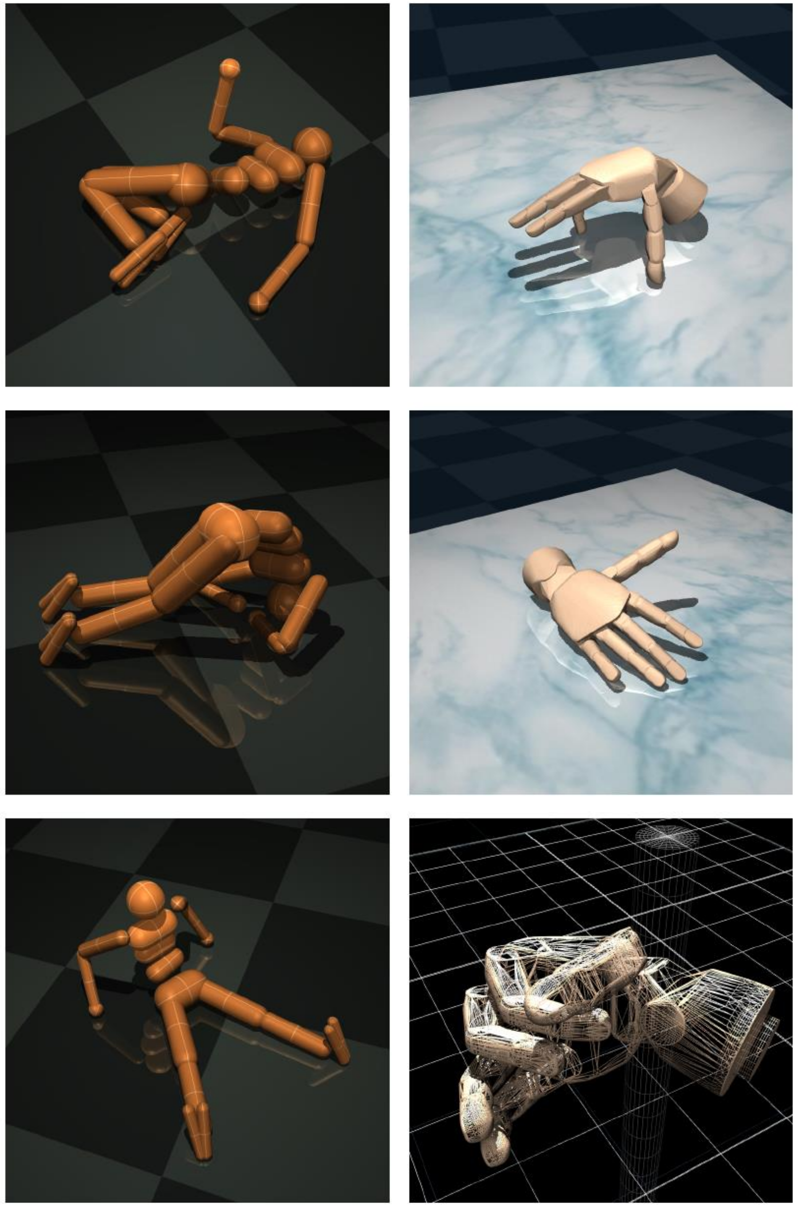
GDD: find a feasible acceleration that minimizes a given cost

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

Non-convex non-smooth constrained optimization problem.
Primal-dual method with exact line search specific to MuJoCo.
Use analytical derivative with respect to acceleration.

With N contacts and pyramidal friction cones, we have 4^N smooth pieces.
GDD optimizes over all pieces, instead of assuming a fixed piece (like QP methods).

GDD simulation results



Cost with three terms:

- virtual damping on all joints

- virtual spring-damper between selected body and user-controlled spatial target

- optional desired pose for grasping

CPU time per step (μs)	**humanoid**	**hand**
forward	49	128
goal-directed	90	182
goal-directed + forward	99	194

AILQR: Acceleration-based Iterative LQR

Discrete-time integration

$$q_{t+h} = q_t + hv_t$$
$$v_{t+h} = v_t + ha_t$$

Running cost

$$\|g(q, v, a)\|_R + p(q, v)$$

Bellman equation

$$V^*(q, v) = p(q, v) + \min_{a \in \mathcal{A}(q,v)} \|g(q, v, a)\|_R + V^*(q + hv, v + ha)$$

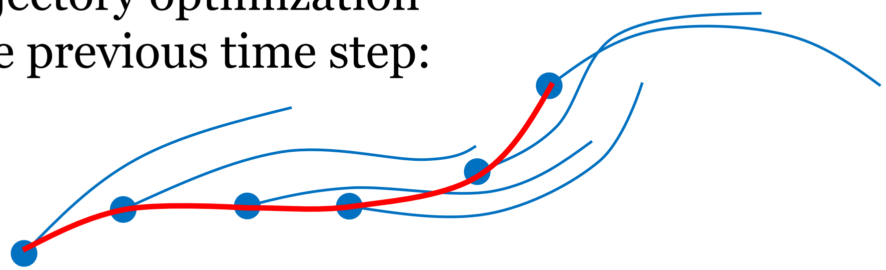
The minimization step in the Bellman equation is equivalent to GDD with

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

Apply iLQR to this problem formulation, with GDD at each step.
Use position and velocity derivatives to propagate V* through time.

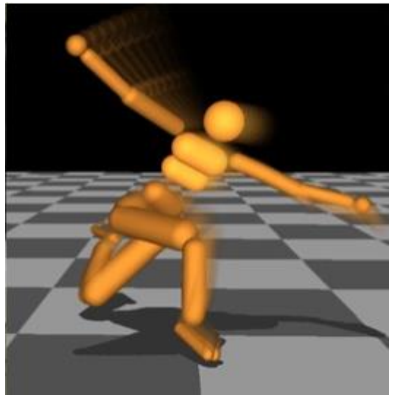
Online trajectory optimization (MPC)

At each time step, do trajectory optimization
with warm-start from the previous time step:



We have been able to apply MPC to the full-body dynamics thanks to:

- efficient physics simulation (MuJoCo);
- efficient optimization algorithm (iLQG);
- carefully designed cost functions.



Tassa, Erez and Todorov, *IROS* 2012
Erez et al, *Humanoids* 2013
Tassa et al; Kumar et al; Erez et al; *ICRA* 2014

FDPG: Fully Deterministic Policy Gradient

Performance: $L(\theta) = \sum_t \ell(x_t, u_t) \quad \text{where } x_{t+1} = f(x_t, u_t), u_t = \pi(x_t, \theta)$

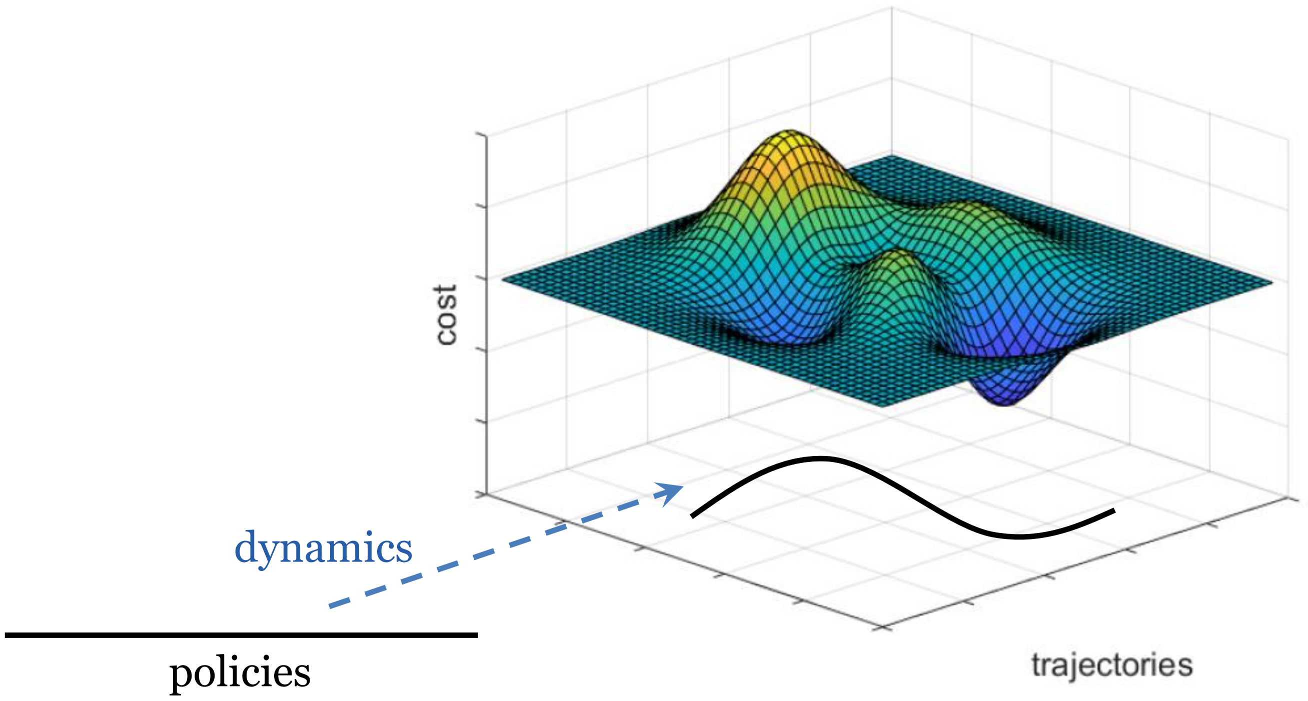
Policy gradient: $$\frac{\partial L}{\partial \theta} = \sum_t \left(\frac{\partial \ell}{\partial x_t} + \frac{\partial \ell}{\partial u_t} \frac{\partial \pi}{\partial x_t} \right) \frac{\partial x_t}{\partial \theta} + \frac{\partial \ell}{\partial u_t} \frac{\partial \pi}{\partial \theta}$$

Forward propagation: $$\frac{\partial x_{t+1}}{\partial \theta} = \left(\frac{\partial f}{\partial x_t} + \frac{\partial f}{\partial u_t} \frac{\partial \pi}{\partial x_t} \right) \frac{\partial x_t}{\partial \theta} + \frac{\partial f}{\partial u_t} \frac{\partial \pi}{\partial \theta}$$

Initialization: $$\frac{\partial x_0}{\partial \theta} = 0$$

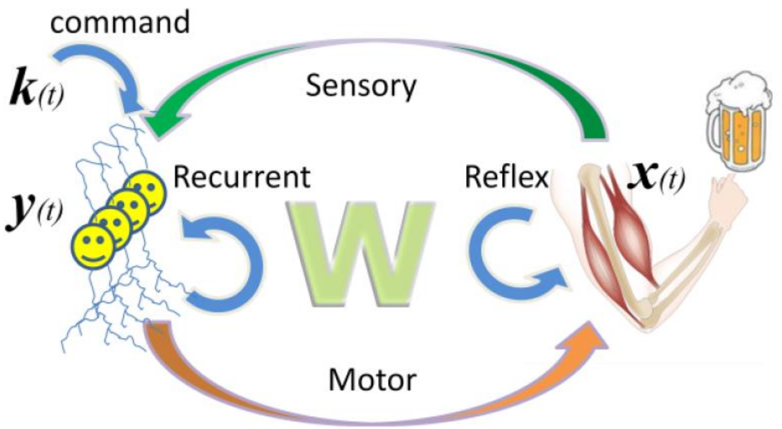
There is also a backward propagation algorithm for the gradient of the cost-to-go function.

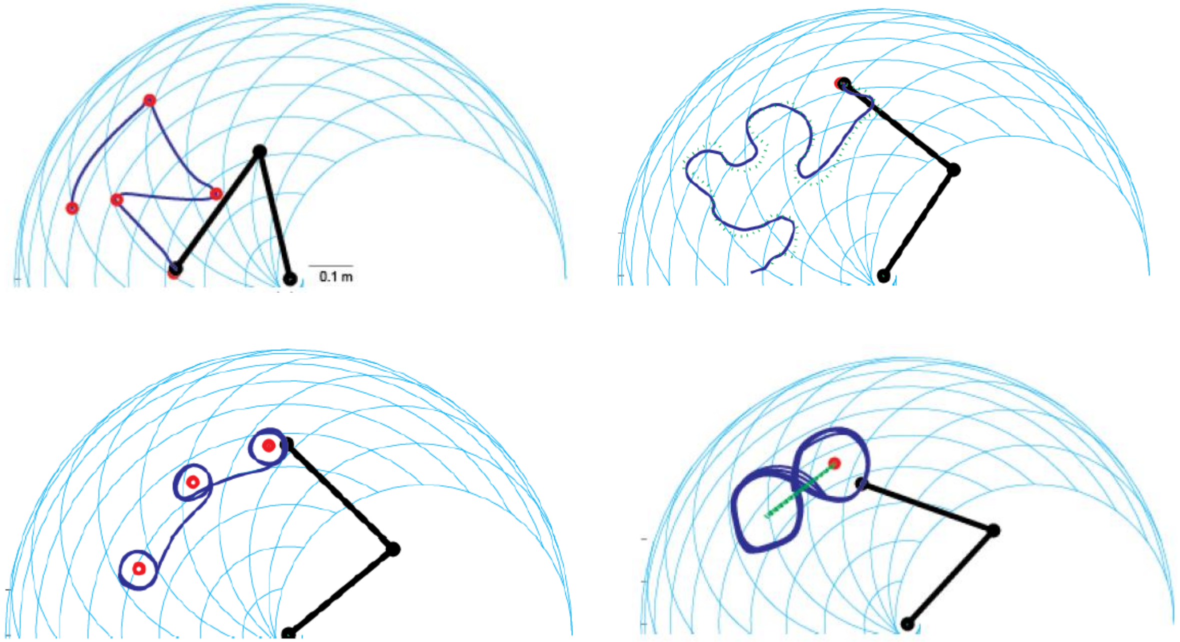
It is called Pontryagin's Maximum Principle, derived 30 years before backpropagation for NNs.



RNN control of human arm model

Huh and Todorov, *ADPRL* 2009





Unfold network and arm dynamics in time

Compute analytical deterministic policy gradient (backpropagation-through-time)

Optimize weights with nonlinear conjugate gradient

the method was so fast and obvious that we called it optimization instead of learning

OBSERVATION:
the same network could learn multiple tasks, but at some point it seemed to hit a wall and there was no way to add more tasks

POLO: Plan Online, Learn Offline

Lowrey, Rajeswaran, Kakade, Todorov and Mordatch, arXiv 2018

10-100x less samples than policy gradient

Infinite-horizon average-cost Bellman equation:

$$c + v(x) = \min_u \{\ell(x,u) + v(f(x,u))\}$$

Equivalent constrained optimization problem:

$$\min_{c, v(\cdot)} \{c\} \quad \text{s.t.} \quad \text{(Bellman)}$$

$$\min_{c, v(\cdot)} \left\{ \min_u \{\ell(x,u) + v(f(x,u))\} - v(x) \right\} \quad \text{s.t.} \quad \text{(Bellman)}$$

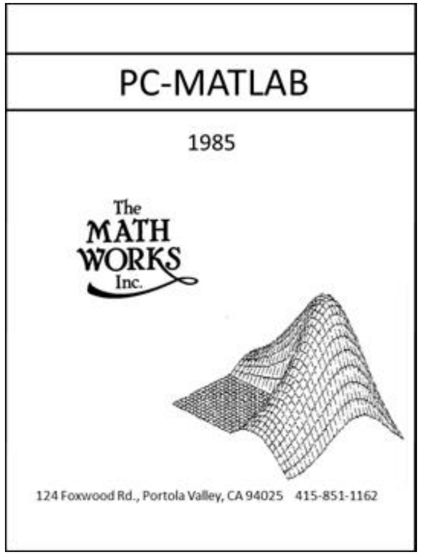
Replace the value function $v(x)$ with an approximation $v(x, \theta)$.
Unfold the dynamics for T steps: $x_{t+1} = f(x_t, u_t)$.
Replace the hard constraint with a soft penalty:

$$\min_{c, \theta, \{u_t\}} \left(\sum\nolimits_t \ell(x_t, u_t) + v(x_T, \theta) - v(x_0, \theta)\right) + \rho \sum\nolimits_t \left(\ell(x_t, u_t) + v(x_{t+1}, \theta) - v(x_t, \theta) - c\right)^2$$

MPC: $\min_{\{u_t\}} \sum_t \ell(x_t, u_t) + v(x_T, \theta)$ Learning: $\min_{c, \theta} \sum_t \left(\ell(x_t, u_t) + v(x_{t+1}, \theta) - v(x_t, \theta) - c\right)^2$

Making computational tools accessible

BLAS, NAG, LAPACK



MuJoCo

OpenAI Gym
DM Control Suite
(more coming)

Optico SDK: costs, derivatives, optimizers

Optico User: dynamic workspace, scripting, GUI