Trajectory optimization

Emo Todorov

University of Washington

Part 1:

Trajectory optimization in real time

Model-predictive control (MPC)

At every time step t, solve the finite-horizon trajectory optimization problem

$$\min_{u} \; h\left(x_{t+N}\right) + \sum_{k=t}^{t+N-1} \ell\left(x_k, u_k\right)$$

Apply the first control u_t, observe/estimate the next state x_{t+1}, and repeat.

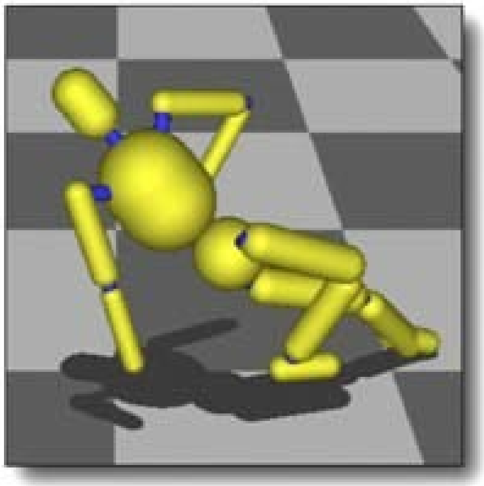
Larger horizon N results in better performance but requires more computation.

The final cost $h(x)$ should approximate all future costs incurred after time $t+N$, i.e. the optimal cost-to-go function (or value function).

The running cost $\ell\left(x, u\right)$ usually penalizes task errors and control energy.

There is always a plan, the plan changes all the time,
and only the initial portion is ever executed.

MuJoCo: A physics engine for control



Recursive algorithms for smooth dynamics

New algorithms for contact dynamics
(Todorov, *ICRA* 2010, 2011)

Parallel evaluation of trajectory costs,
gradients and Hessians

Efficient C implementation

400,000 dynamics evaluations per second on a 12-core 3GHz PC,
18-dof humanoid model with 6 active contacts

A full Newton step of trajectory optimization takes 100 msec

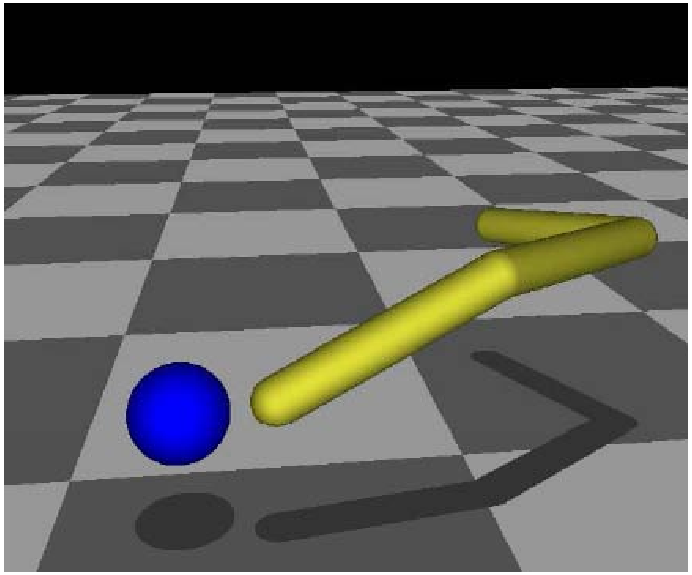
Almost all the CPU time is spent in finite-differencing the dynamics

Application to swimming

Tassa, Erez and Todorov, *work in progress*

method 1: optimize only the running cost, ignore the final cost / cost-to-go

this works well when a lot of progress can be made within the planning horizon

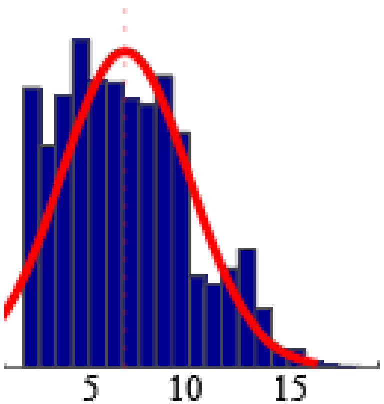


Application to ball bouncing

Kulchenko and Todorov, *ICRA* 2011

method 2: use some heuristic approximation to the optimal cost-to-go

similar to evaluation function in chess; rough approximation is usually sufficient



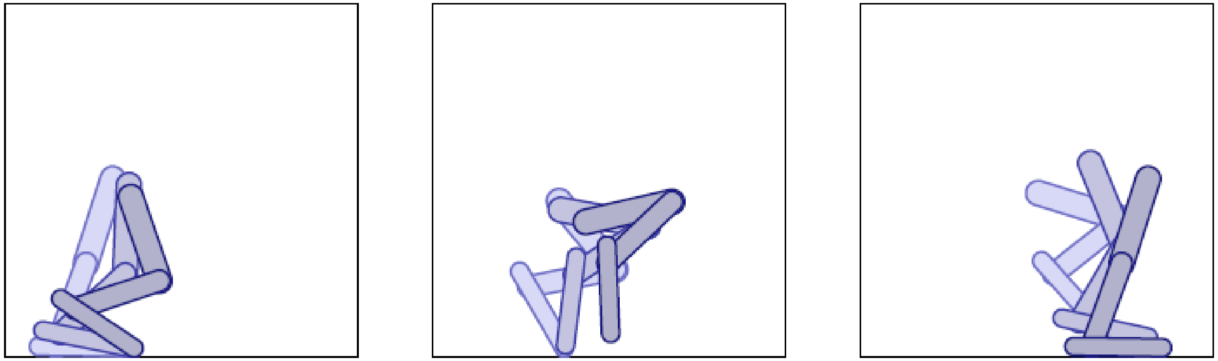
397,000 views on YouTube !?

Application to hopping

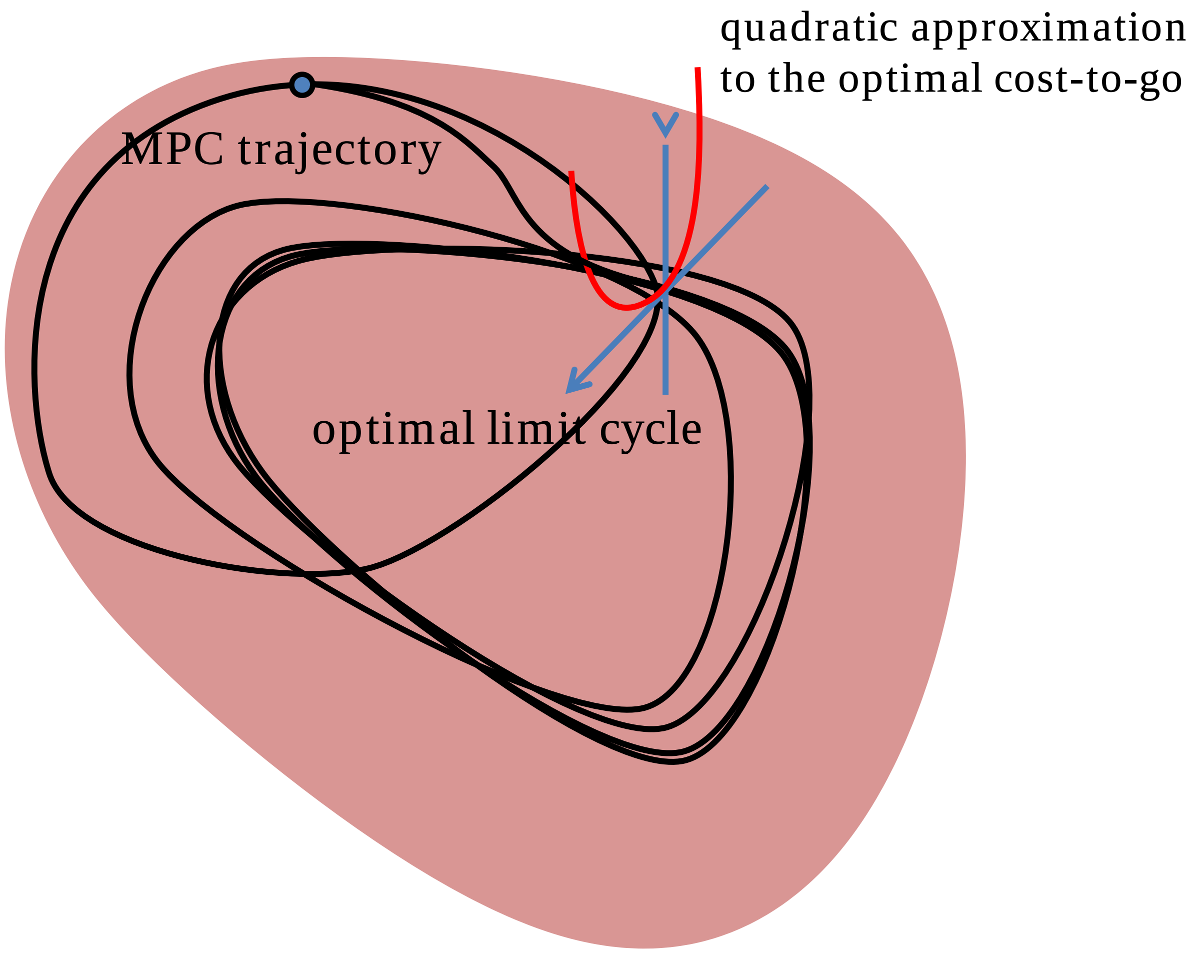
Erez, Tassa and Todorov, *RSS* 2011

method 3: use offline optimization to model the optimal cost-to-go

The offline model can be obtained via trajectory optimization or ADP/RL



Outline of the algorithm

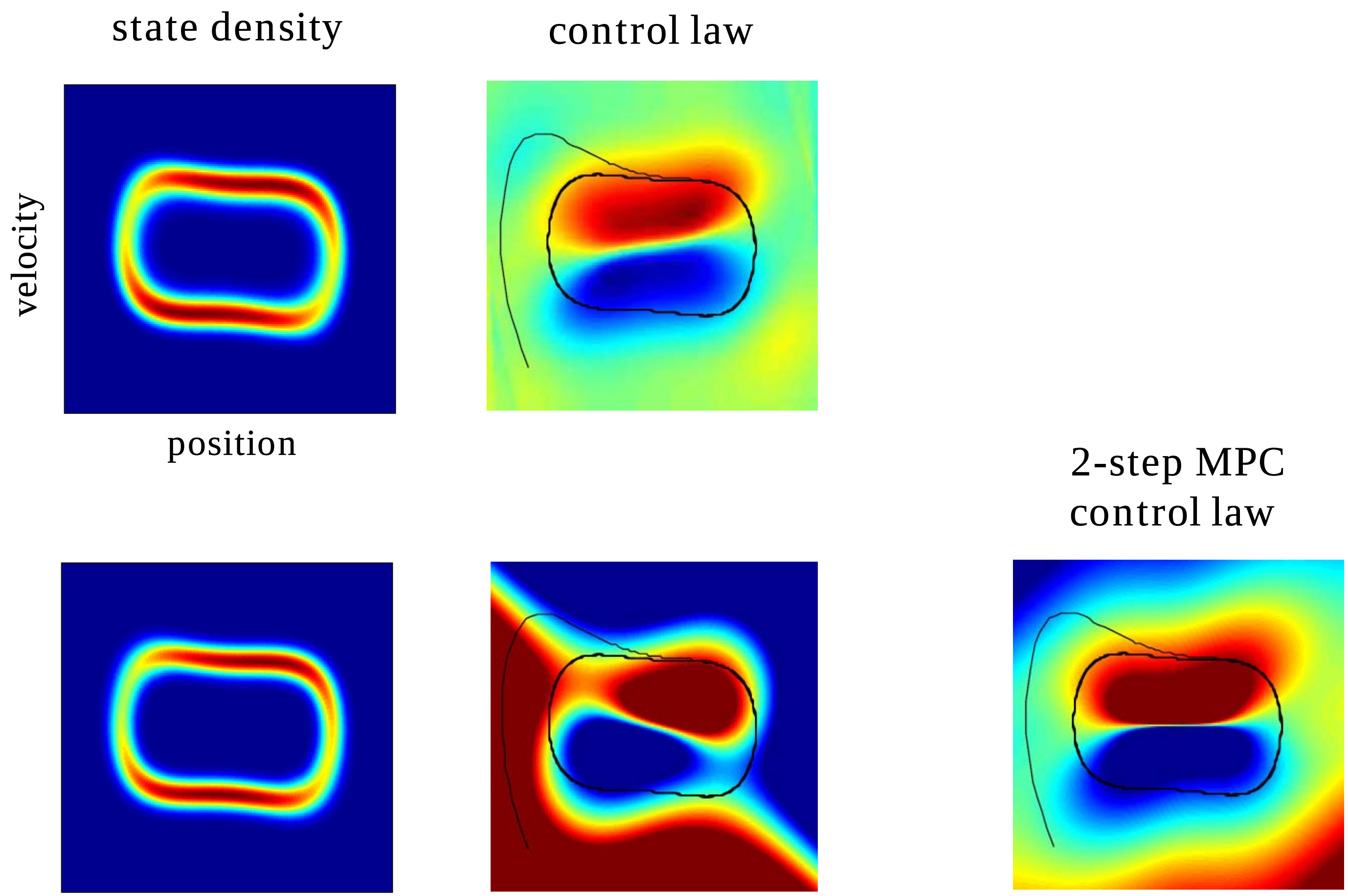


Extending the local region of validity via MPC

Stochastic nonlinear spring; the task is to move at constant speed in either direction.

global minimum
(dense discretization)

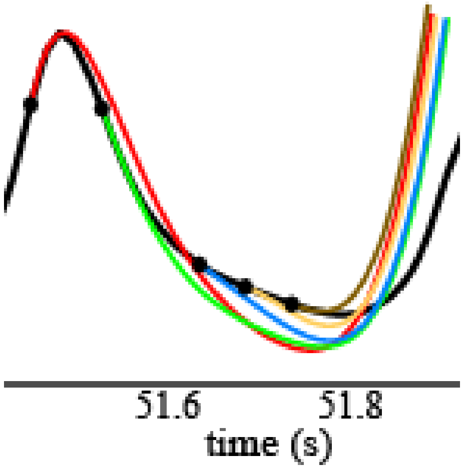
locally-optimal
limit cycle

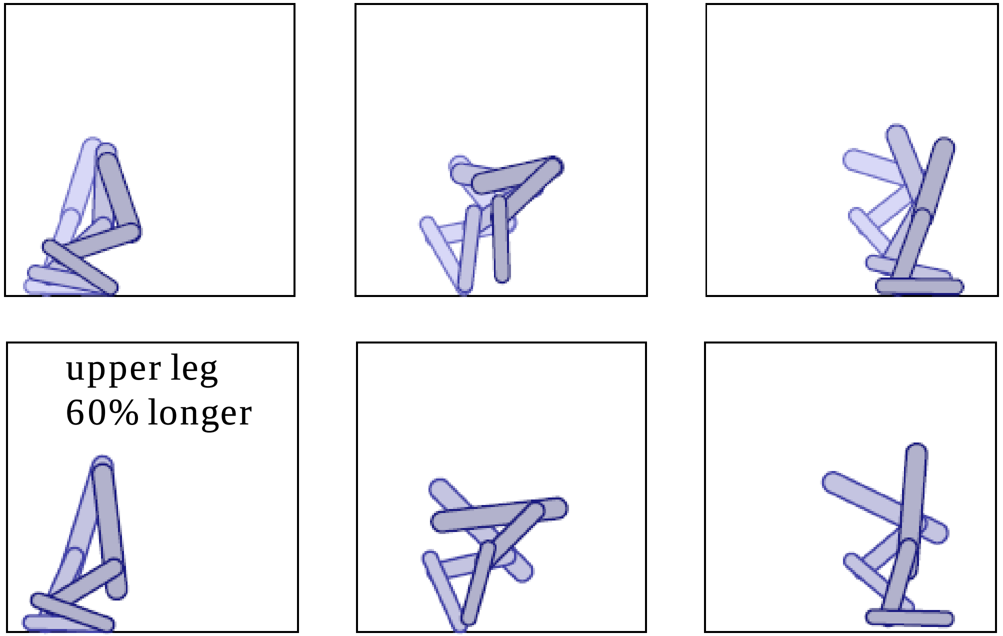


Empirical robustness to model errors

It is hard to obtain theoretical guarantees, because MPC control laws are defined implicitly as the outcome of optimization.

un-modeled friction





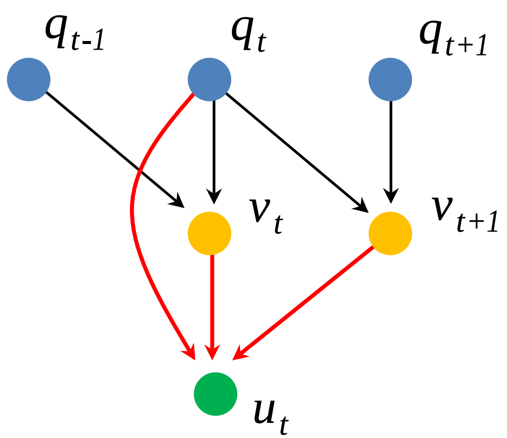
Part 2:

Making trajectory optimization easier

Optimization through inverse dynamics

Standard trajectory optimization methods rely on forward dynamics
(Maximum Principle, DDP, iLQG, iLDP)

However trajectory optimization may be faster using inverse dynamics:
- no need for forward integration, no instability, large time steps
- minimal representation, yet the Hessian of the cost is sparse



Represent trajectory as sequence of positions q

Compute velocities v using finite differencing

Compute controls u using inverse dynamics

The trajectory cost is

$$\sum_{k=1}^{n} \ell\left(q_k, v(q_{k+1}, q_k), u(q_{k+1}, q_k, q_{k-1})\right) + \alpha \left\| B^{\perp} u(q_{k+1}, q_k, q_{k-1}) \right\|^2$$

How can we compute inverse dynamics in the presence of contacts?
See Todorov, *ICRA* 2011.

Helper forces and contact smoothing

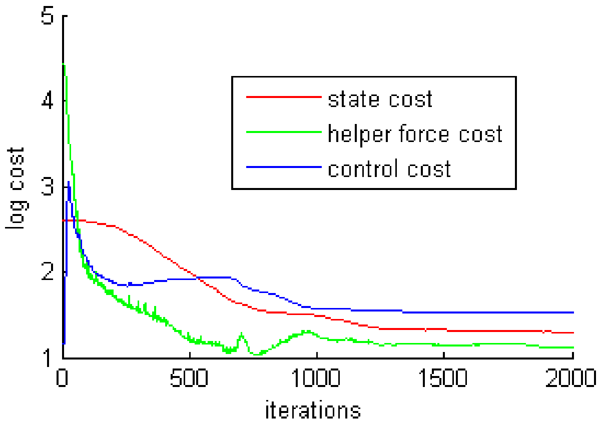
Tassa and Todorov, *RSS* 2010; Todorov, *ICRA* 2011
Erez and Todorov, *work in progress*

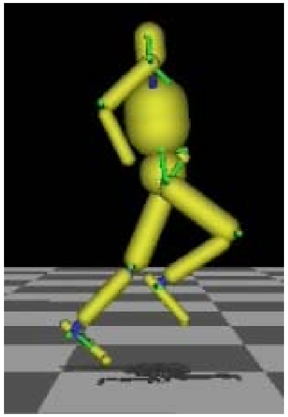
Under-actuation and contact discontinuities make trajectory optimization hard.

We allow some "helper forces" in the un-actuated space, and penalize them.

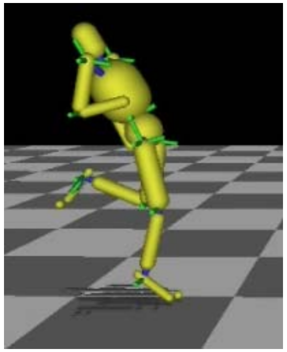
We smooth contacts, so that some contact forces can be applied from a distance while the ground is still hard.

learning progress





inappropriate weight
on helper force cost



Allowing rigid-body deformations

Mordatch and Todorov, *work in progress*

Independent point-mass dynamics

Costs used to (softly) enforce the dynamics:
- constant segment lengths
- no penetration
- no slip

These are mixed with regular costs:
- CoM should follow reference trajectory
- acceleration and jerk should be small

Pushing towards the goal in end-effector space

Todorov et al, *IEEE BioRob* 2010

joint space configuration	q
end effector position	$y(q)$
end effector Jacobian	$J(q) = \dfrac{\partial y(q)}{\partial q}$
Jacobian null space	$N(q)$

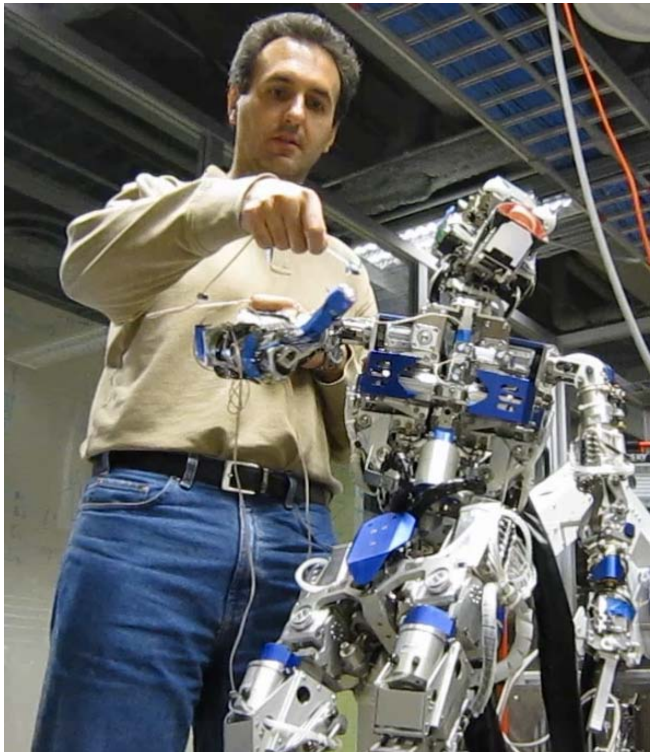
Push hand towards target:

$$u = k\, J(q)^T \left(y^* - y(q)\right)$$

Push hand towards target,
while staying close to default configuration:

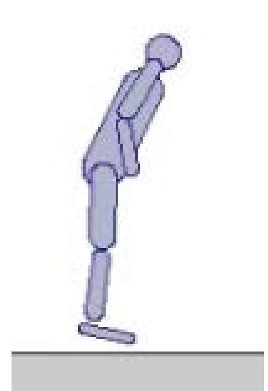
$$u = k_1\, J(q)^T \left(y^* - y(q)\right) + k_2\, N(q)\left(q^* - q\right)$$

Pneumatic robot (Diego-san)
actuator time constants ~ 80 msec



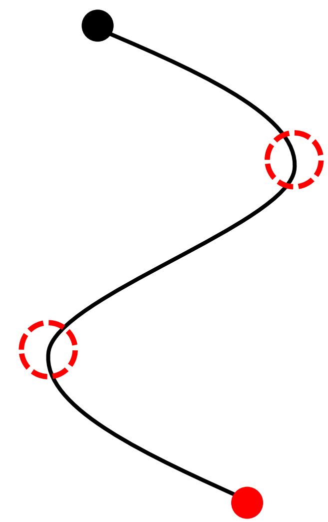
Sequential sub-goals in end-effector / feature space

Mordatch, Popovic and Todorov, *work in progress*



Getting up

Simple versions can be solved directly
via trajectory optimization (Erez, in progress)



The general case seems too hard for trajectory optimization,
and can benefit from suitable sub-goals