Synthesis of complex movements with optimal control

Emo Todorov

Applied Mathematics, Computer Science & Engineering
University of Washington

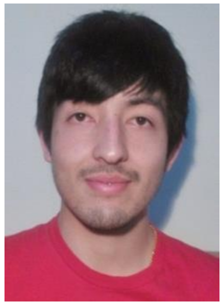
Contributions from:

Igor Mordatch

Vikash Kumar

Yuval Tassa

Tom Erez


Kendall Lowrey

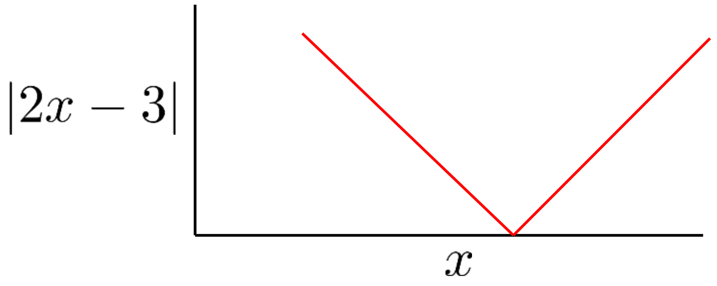
Paul Kulchenko

Funding: NSF, NIH, DARPA

Solving unsolved problems via numerical optimization

Suppose an algorithm for division was not yet invented. How can we compute $3/2$?
We are looking for a number x such that $2x = 3$.
We could check all numbers, but that will take forever.

Convert the original problem into an optimization problem: $\min_x |2x - 3|$

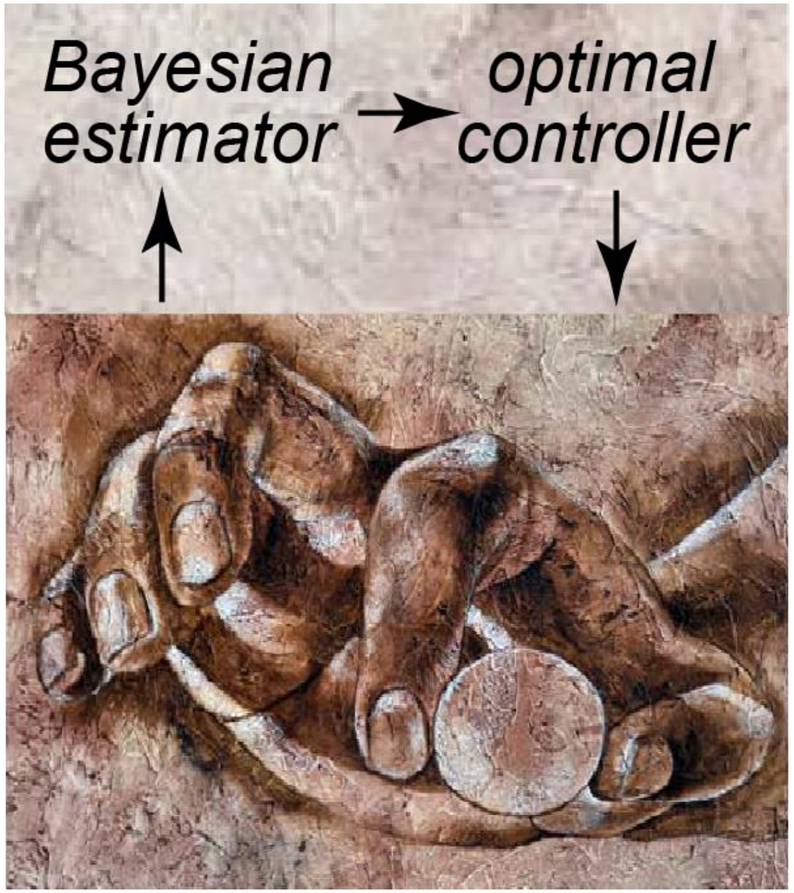


Most problems in robotics (and elsewhere) can be expressed as optimization problems.
Computers can then search the solution space efficiently and find sensible solutions,
without having an "algorithm" for the original problem.

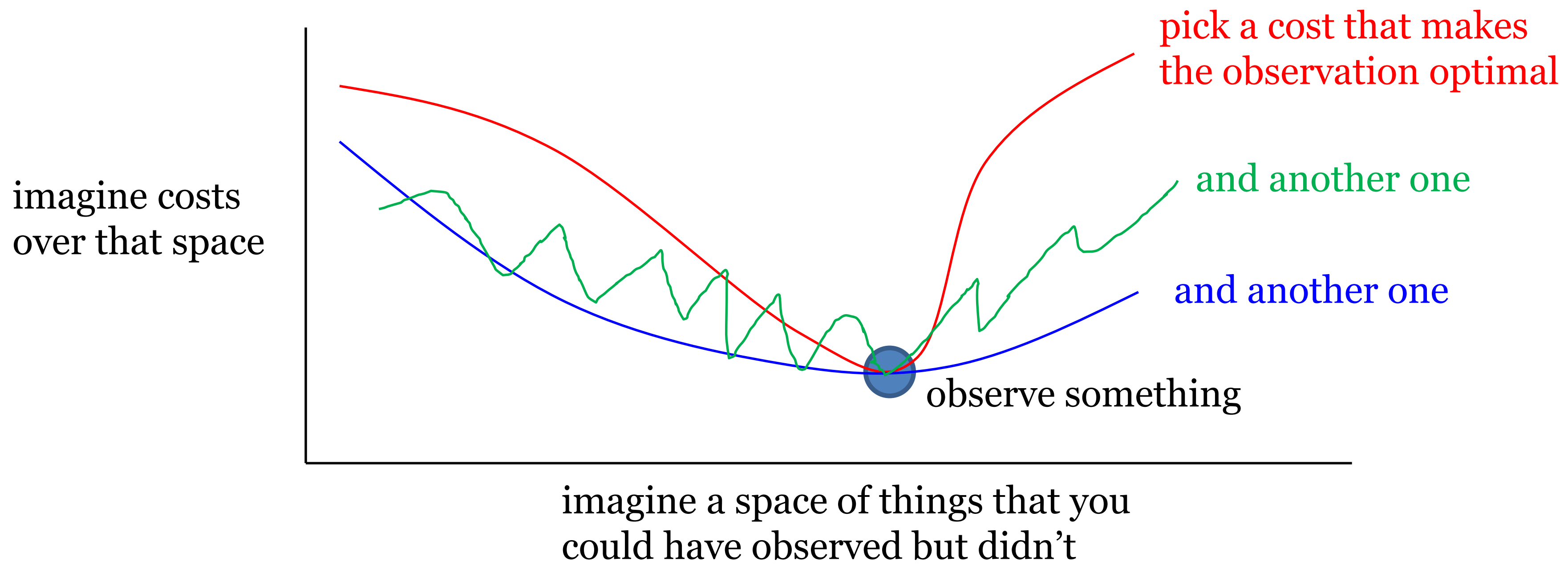
Examples:
- inverse kinematics
- state estimation
- system identification
- motion planning
- motion control

General recipe:
- define an easy-to-optimize cost whose minimum corresponds to the solution
- apply an efficient optimization method
- if necessary, wait for Moore's law to make the optimization tractable ☺

Optimal control models in biology



Optimality as a meta-theory



imagine costs over that space
pick a cost that makes the observation optimal
and another one
and another one
observe something
imagine a space of things that you could have observed but didn't

To obtain a concrete theory from the meta-theory of optimal control,
we must specify a **cost function** and/or **constraints**.
We must also specify a **level of detail** for the (bio)physics model.

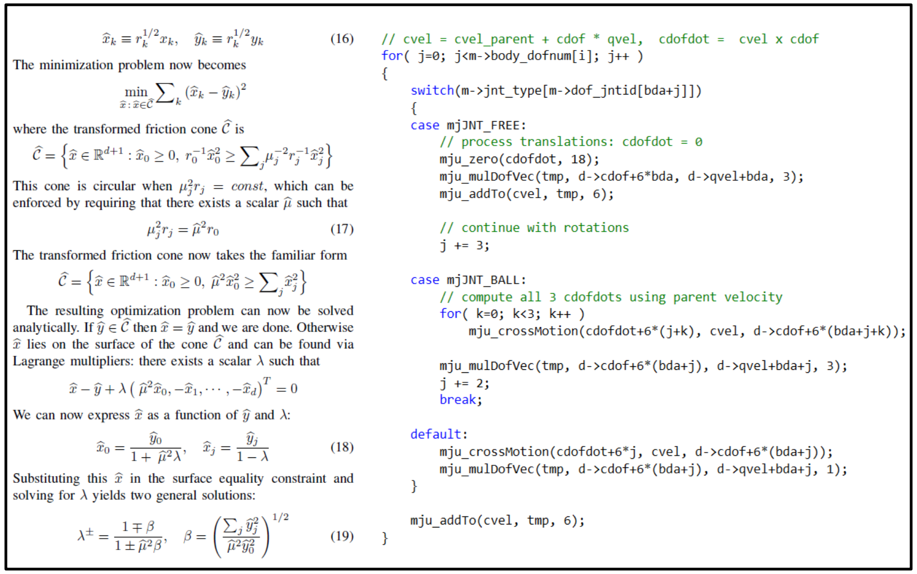
As with every scientific theory, we should keep the assumptions
as simple and *a priori* justified as possible, and seek predictions
that are as elaborate as possible (agreement with data would also be nice)

From simple assumptions to elaborate predictions

physics model

costs and constraints

optimization machine



$$\widehat{x}_k \equiv r_k^{1/2} x_k, \quad \widehat{y}_k \equiv r_k^{1/2} y_k \tag{16}$$

The minimization problem now becomes

$$\min_{\widehat{x}\,:\,\widehat{x}\in\widehat{\mathcal{C}}} \sum\nolimits_k \left(\widehat{x}_k - \widehat{y}_k\right)^2$$

where the transformed friction cone $\widehat{\mathcal{C}}$ is

$$\widehat{\mathcal{C}} = \left\{\widehat{x} \in \mathbb{R}^{d+1} : \widehat{x}_0 \geq 0,\ r_0^{-1}\widehat{x}_0^2 \geq \sum\nolimits_j \mu_j^{-2} r_j^{-1} \widehat{x}_j^2\right\}$$

This cone is circular when $\mu_j^2 r_j = const$, which can be enforced by requiring that there exists a scalar $\widehat{\mu}$ such that

$$\mu_j^2 r_j = \widehat{\mu}^2 r_0 \tag{17}$$

The transformed friction cone now takes the familiar form

$$\widehat{\mathcal{C}} = \left\{\widehat{x} \in \mathbb{R}^{d+1} : \widehat{x}_0 \geq 0,\ \widehat{\mu}^2 \widehat{x}_0^2 \geq \sum\nolimits_j \widehat{x}_j^2\right\}$$

The resulting optimization problem can now be solved analytically. If $\widehat{y} \in \widehat{\mathcal{C}}$ then $\widehat{x} = \widehat{y}$ and we are done. Otherwise $\widehat{x}$ lies on the surface of the cone $\widehat{\mathcal{C}}$ and can be found via Lagrange multipliers: there exists a scalar λ such that

$$\widehat{x} - \widehat{y} + \lambda \left(\widehat{\mu}^2 \widehat{x}_0, -\widehat{x}_1, \cdots, -\widehat{x}_d\right)^T = 0$$

We can now express $\widehat{x}$ as a function of $\widehat{y}$ and λ:

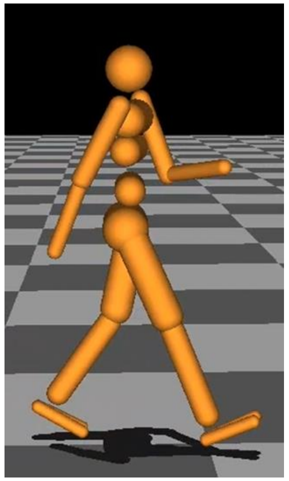
$$\widehat{x}_0 = \frac{\widehat{y}_0}{1 + \widehat{\mu}^2 \lambda}, \quad \widehat{x}_j = \frac{\widehat{y}_j}{1 - \lambda} \tag{18}$$

Substituting this $\widehat{x}$ in the surface equality constraint and solving for λ yields two general solutions:

$$\lambda^{\pm} = \frac{1 \mp \beta}{1 \pm \widehat{\mu}^2 \beta}, \quad \beta = \left(\frac{\sum_j \widehat{y}_j^2}{\widehat{\mu}^2 \widehat{y}_0^2}\right)^{1/2} \tag{19}$$

```
// cvel = cvel_parent + cdof * qvel,  cdofdot =  cvel x cdof
for( j=0; j<m->body_dofnum[i]; j++ )
{
    switch(m->jnt_type[m->dof_jntid[bda+j]])
    {
    case mjJNT_FREE:
        // process translations: cdofdot = 0
        mju_zero(cdofdot, 18);
        mju_mulDofVec(tmp, d->cdof+6*bda, d->qvel+bda, 3);
        mju_addTo(cvel, tmp, 6);

        // continue with rotations
        j += 3;

    case mjJNT_BALL:
        // compute all 3 cdofdots using parent velocity
        for( k=0; k<3; k++ )
            mju_crossMotion(cdofdot+6*(j+k), cvel, d->cdof+6*(bda+j+k));

        mju_mulDofVec(tmp, d->cdof+6*(bda+j), d->qvel+bda+j, 3);
        j += 2;
        break;

    default:
        mju_crossMotion(cdofdot+6*j, cvel, d->cdof+6*(bda+j));
        mju_mulDofVec(tmp, d->cdof+6*(bda+j), d->qvel+bda+j, 1);
    }

    mju_addTo(cvel, tmp, 6);
}
```

behavior



cost: torque²
constraint: symmetric limit cycle, fixed duration and distance

We represent the movement trajectory in a Fourier basis which guarantees that the constraints are always satisfied. Then we express the cost (integrated squared torque) as a function of the trajectory, and optimize it with a Gauss-Newton method.

Erez, Tassa and Todorov, *IROS* 2012

Contact-invariant optimization (CIO)

We introduce auxiliary decision variables $c_{i,\phi(t)}$ indicating if potential contact *i* should be active in movement phase $\phi(t)$.

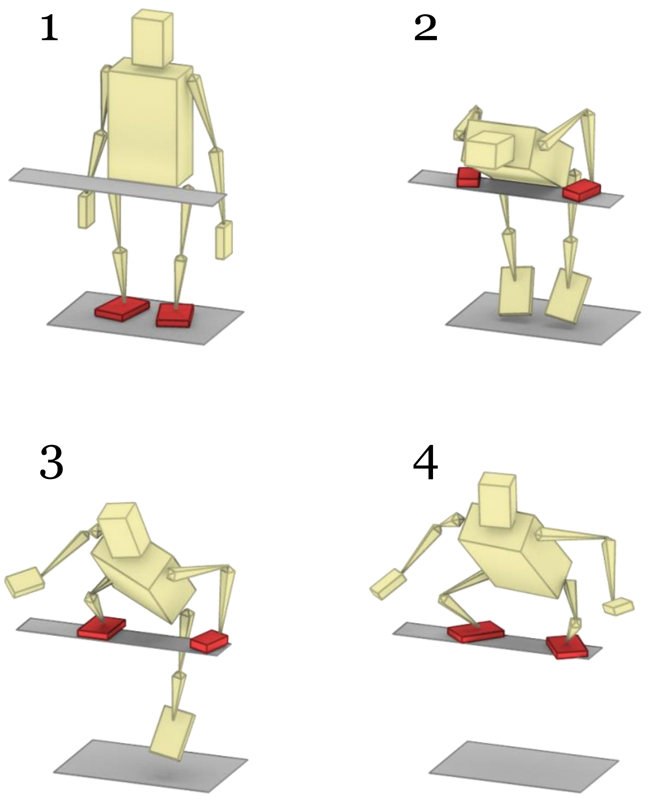
The cost has the extra term

$$\sum_t c_{i,\phi(t)}(\mathrm{s})\left(\|\mathbf{e}_{i,t}(\mathrm{s})\|^2 + \|\dot{\mathbf{e}}_{i,t}(\mathrm{s})\|^2\right)$$

where **e** is the vector distance to the nearest surface.

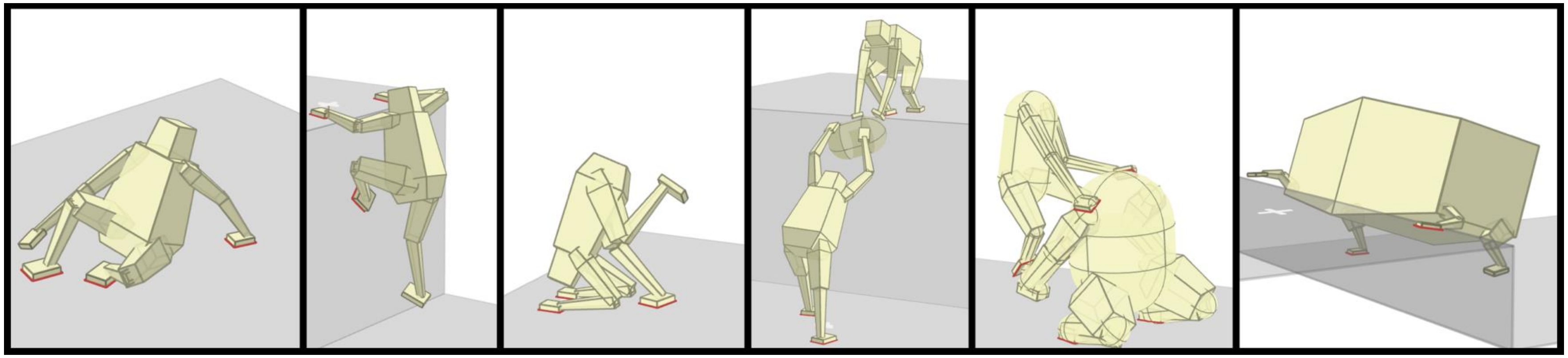
The planned contact impulse is penalized in full, but scaled by c before being applied.

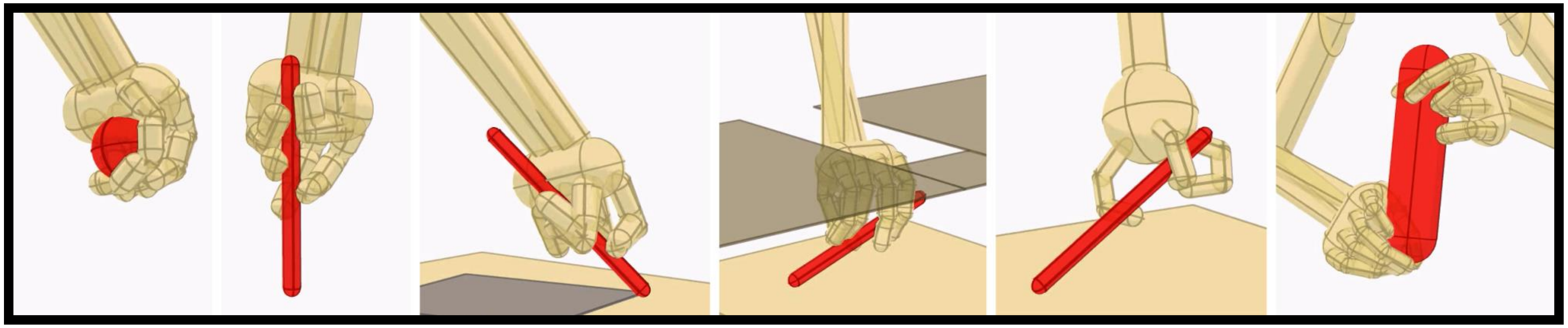
The optimizer has to "declare" the contacts it relies on, forcing it to reason about contact dynamics explicitly.



Mordatch, Todorov and Popovic, *SIGGRAPH* 2012
Mordatch, Popovic and Todorov, *SCA* 2012

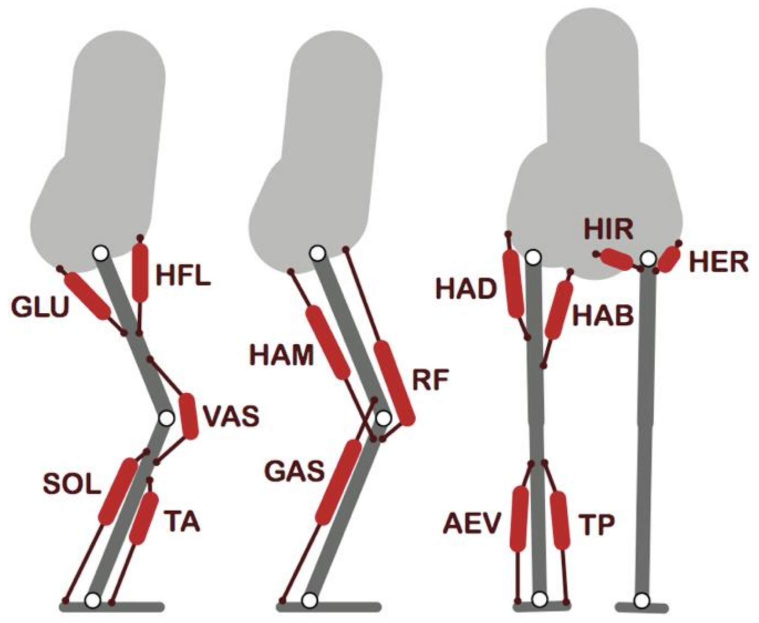
Results

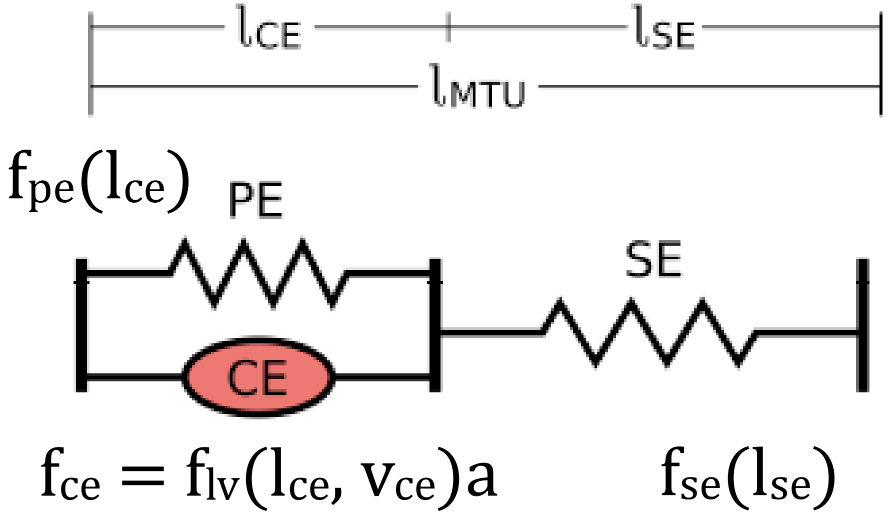




Extension to musculo-skeletal dynamics

- skeletal dynamics simulated in MuJoCo (our physics engine)
- muscle model based on Wang et al 2012
- metabolic energy model based on Anderson and Pandy 1999
- modifications to the CIO method

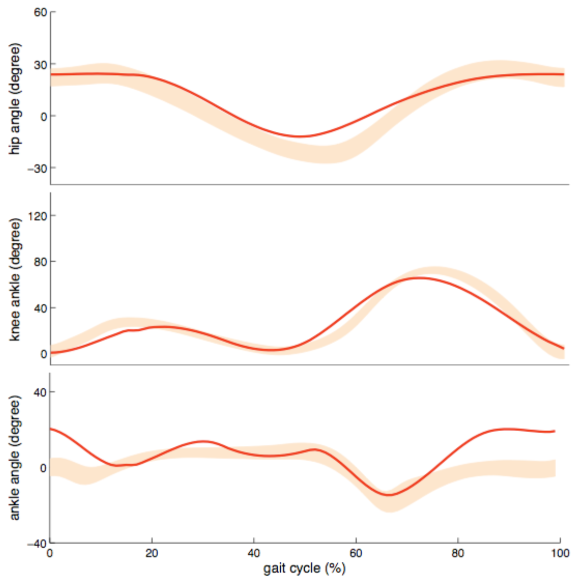




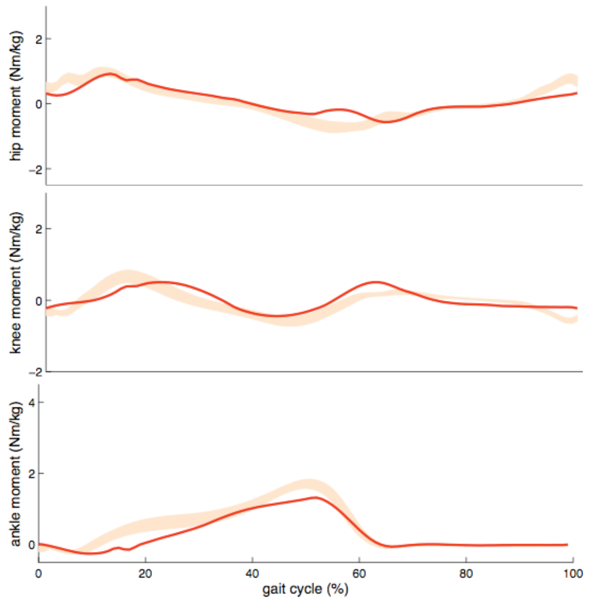
Mordatch, Wang, Todorov and Koltun, *SIGGRAPH ASIA* 2013

Walking at 1.5 m/s

Kinematics



Torques



GLU

RF

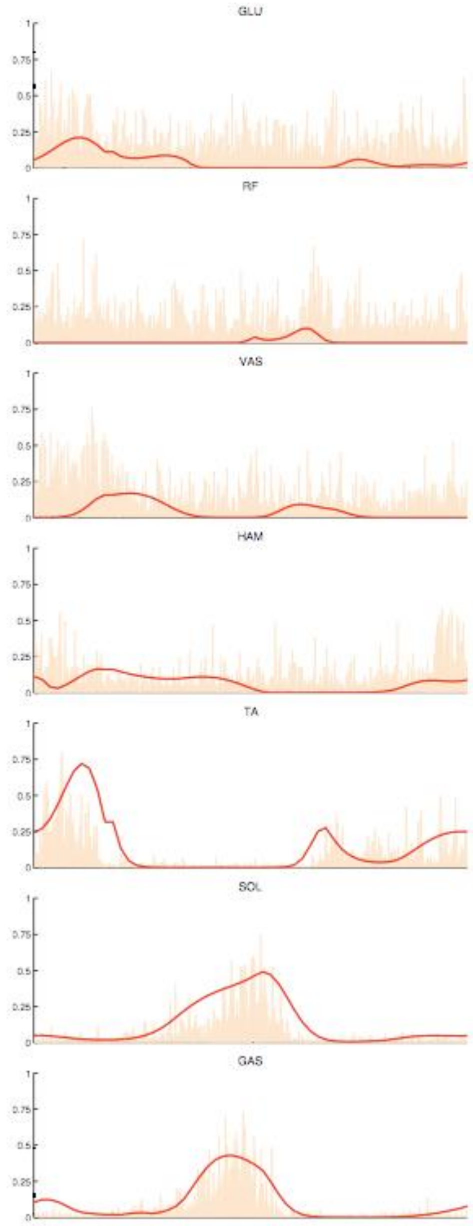
VAS

HAM

TA

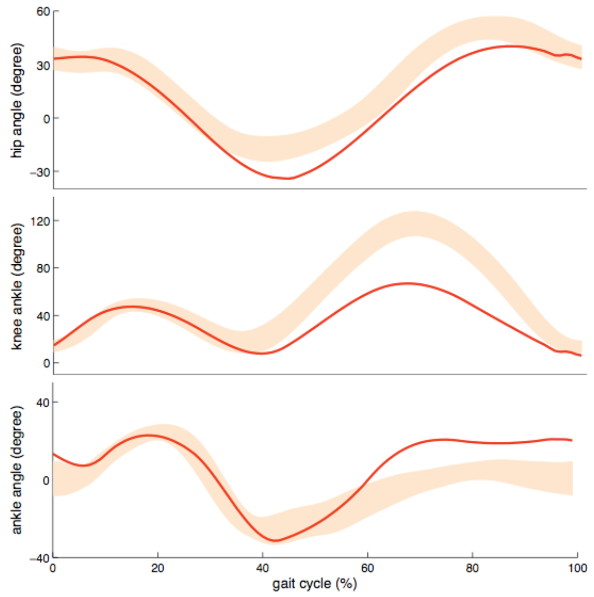
SOL

GAS

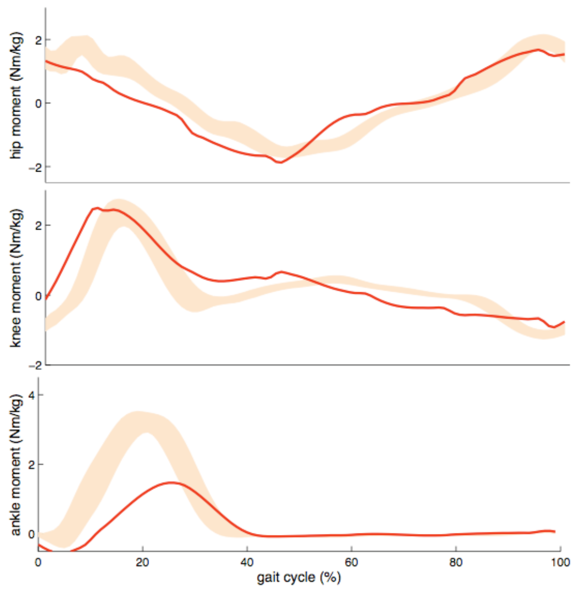


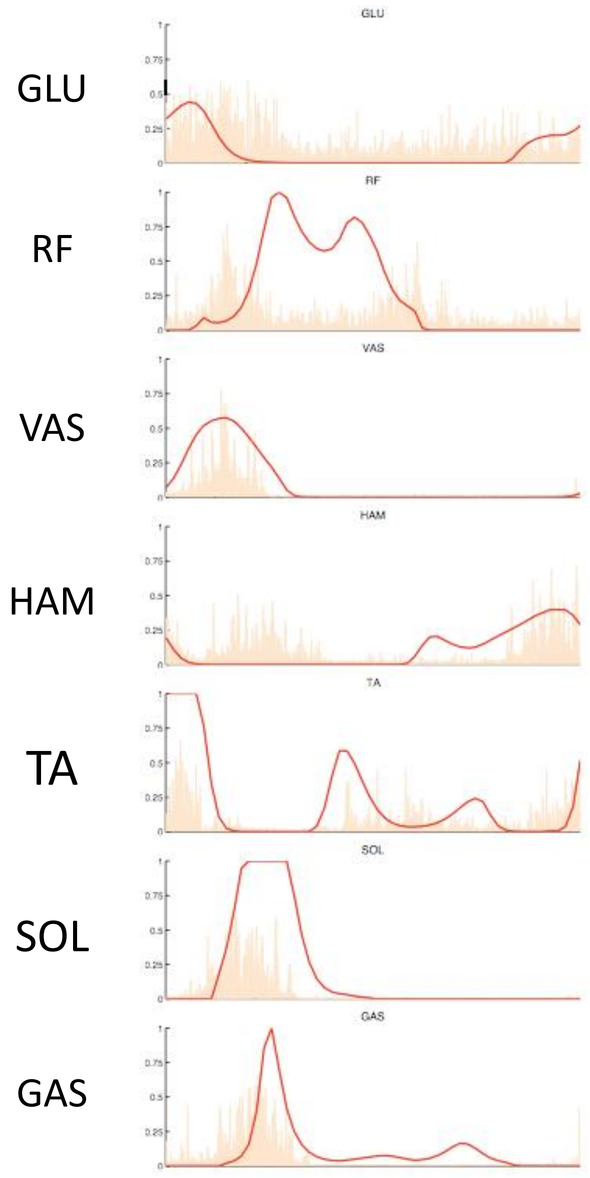
Running at 4 m/s

Kinematics



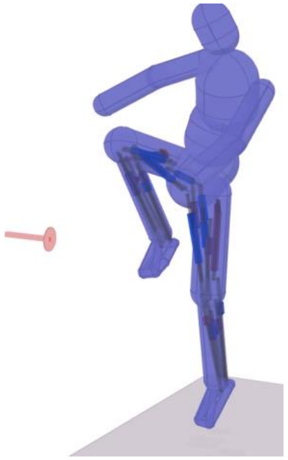
Torques

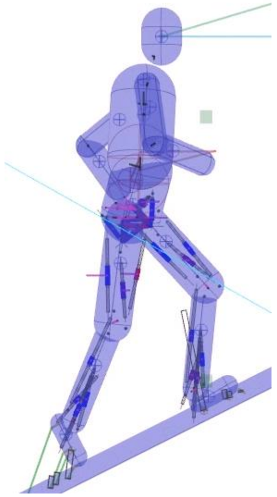


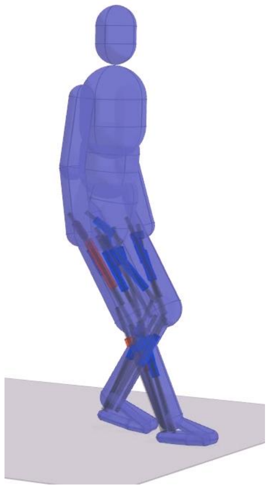


Other behaviors

jumping


kicking


slope


stopping

moon gravity

Observations

Trajectory optimization can generate surprisingly long and complex movements, without help from motion capture, manual scripting or careful initialization.

Contacts must be handled carefully, so as to provide sufficient smoothing for derivatives while avoiding the issues associated with earlier spring-dampers.

Optimizing high-level variables (as in CIO) together with the low-level trajectory produces the richest behaviors we have obtained.

We need ~1000 iterations to discover these trajectories from scratch, no matter which representation or algorithm we use.

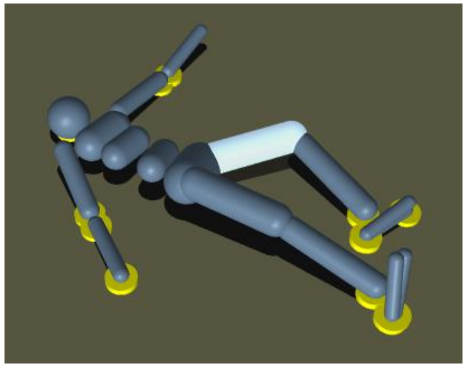
Currently this takes ~5 minutes, which is quite amazing but at the same time is too slow for control applications that require online re-planning.

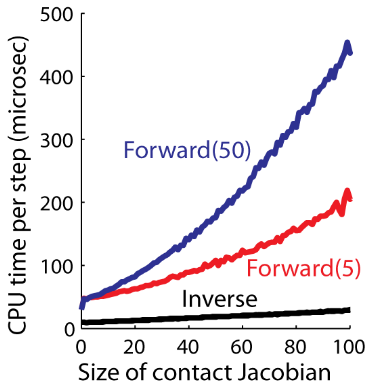
The role of **fast** simulation in optimization

The optimizer evaluates a vast number of candidate control signals in order to find a sequence that works.

This evaluation is done with a physics simulator – which needs to be much faster than real time.

We have developed the first simulator (MuJoCo) that is sufficiently fast and accurate to enable efficient optimization of complex behavior.



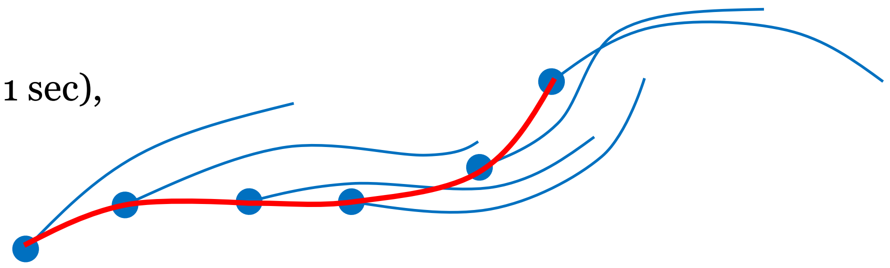


Forward dynamics: **100** times faster than real-time.
Inverse dynamics: **1000** times faster than real-time.

Todorov, *ICRA* 2011, 2014 (?)
Todorov, Tassa, Erez, *IROS* 2012

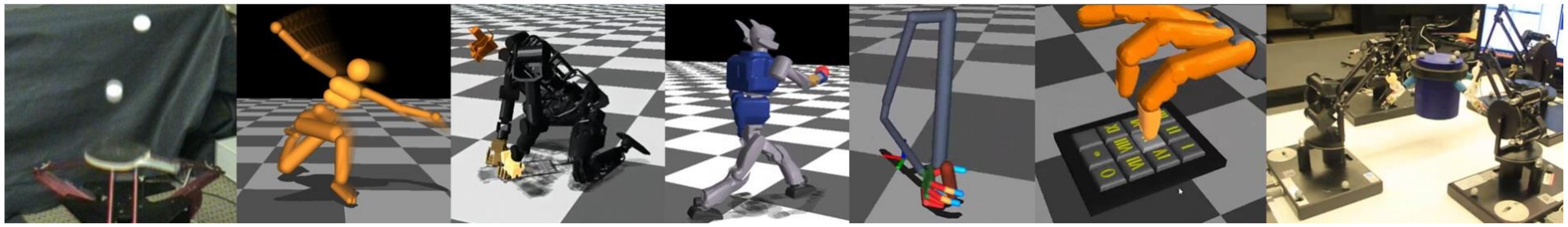
Model-predictive control (MPC)

Once every ~30 msec:
- re-optimize the plan (——) up to some horizon (~ 1 sec), starting from the current state (●)
- execute (——) the initial portion of the plan, while the next plan is being computed



MPC has been used to control chemical plants, play computer chess, drive the Google car.
However robot dynamics are too fast for existing optimizers to keep up.

We were able to apply MPC to complex robots for the first time, due to:
- improved models of contact dynamics;
- efficient physics simulator (MuJoCo);
- efficient optimization algorithm (iLQG);
- selection of cost functions that are realistic yet easier to optimize.



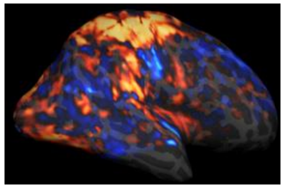
Tassa, Erez and Todorov, *IROS* 2012
Erez et al, *Humanoids* 2013
Tassa et al; Kumar et al; Erez et al; *ICRA* 2014 (?)

A case for MPC in the brain

Complex hardwired behaviors can be generated by "neural machines" with few neurons.

Reaching could also be generated this way: the frog wipe reflex is essentially a reaching movement, with coordinate transformations, online feedback etc.

We often think of primate learning as setting up a similar neural machine, which is then responsible for motor execution.



But if that were true, most of the primate brain would shut down during execution of simple familiar movements… and it doesn't.

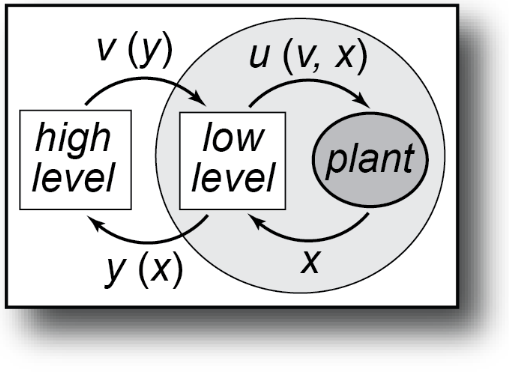
Whatever the brain is doing, it appears to be an overkill for executing movements that have already been learned. Conversely, whatever is learned must be insufficient to execute movements – so we see extensive online processing even after learning.

What is this processing doing? Unless you have a better idea, consider MPC ☺

Most neurons are in the **cerebellum** – which is presumably a big internal model. This many neurons seem an overkill given the current (limited) views on what internal models are used for.

MPC has a better use for internal models: asking lots of "what-if" questions and thereby optimizing the behavior online. This takes **70% of CPU time** in our case.

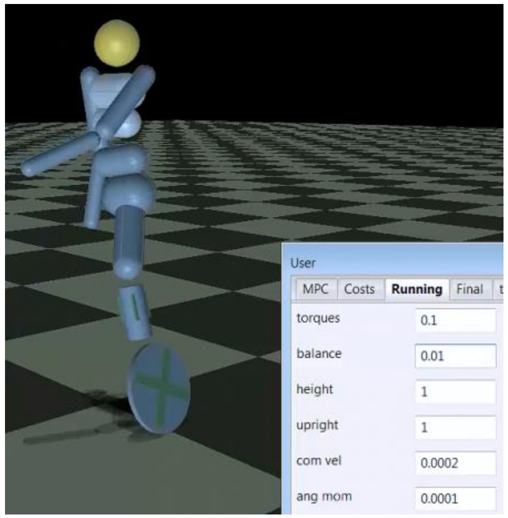
Cost parameters as high-level commands



Suppose the low level controller is capable of approximately optimizing different costs.

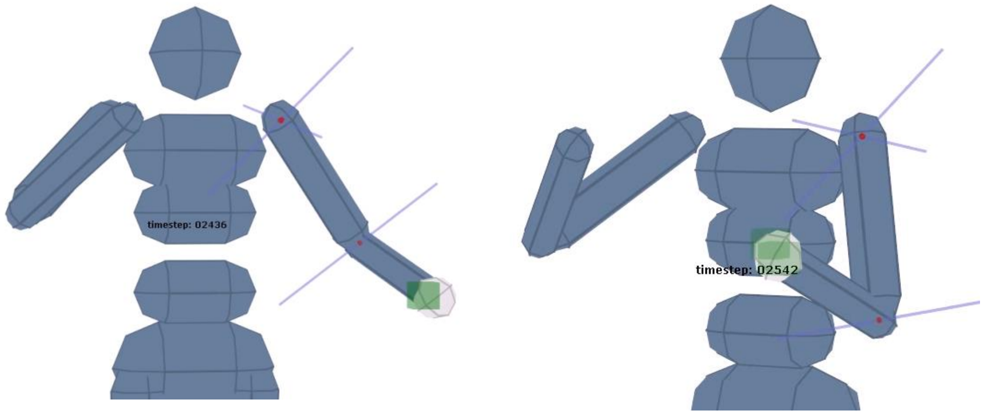
Then the high level command can be a time-varying cost, (or time-varying parameters of a cost).

Low level: MPC



torques	0.1
balance	0.01
height	1
upright	1
com vel	0.0002
ang mom	0.0001

Low level = neural net trained with MPC


timestep: 02436
timestep: 02542