Automatic synthesis of complex behaviors with optimal control

Emo Todorov

Applied Mathematics
Computer Science and Engineering

University of Washington
What makes control hard?

Some of the usual suspects are:

- non-linearity
- high dimensionality
- redundancy
- noise and uncertainty

These properties can make an already hard problem harder, however none of them is a root cause of difficulty.

A control problem can have all these properties and still be easy, in the sense that there exists a simple strategy that always works:

*Push towards the goal!*

The problem is hard when this strategy is infeasible, due to *constraints.*
Easy example: Reaching with a redundant arm

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

Pneumatic robot (Diego-san)
air pressure similar to muscle activation, but with longer time constant (~ 80 ms)

Push hand towards target:

$$u = k \cdot J(q)^T (y^* - y(q))$$

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

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

The controller does not need to worry about the path, or the speed profile, or stability, or anything else - it all emerges from the nicely damped dynamics.
Easy example: Trajectory tracking with PD control

1 minute of tracking

--- reference trajectory

--- actual trajectory
Solving hard problems via optimal control

Erez, Tassa and Todorov
Recent results

Kulchenko and Todorov  
*ICRA 2011*

Tassa, Erez and Todorov  
*IROS 2012*

Erez, Tassa and Todorov  
*IROS 2012*

Mordatch, Todorov and Popovic  
*SIGGRAPH 2012*

Mordatch, Popovic and Todorov  
*SCA 2012*
Methods for optimal control
Online optimization

At time step $t$, solve a trajectory optimization problem of the form

$$\min h(x_{t+N}) + \sum_{k=t}^{t+N-1} \ell(x_k, u_k)$$

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).

At each time step:
- optimize the trajectory (initializing from the previous time step)
- apply the first control $u_t$
- observe/estimate the next state $x_{t+1}$

There is always a plan;
plan changes all the time;
the initial portion is executed.
MuJoCo: A physics engine for control

100 times faster than real-time on single core
parallel evaluation of costs and derivatives

15,000 lines of C/C++ code
recursive algorithms for smooth dynamics
new algorithms for contact dynamics
modeling of tendons, muscles, pneumatics
The case for online optimization in the brain

complex “hardwired” behaviors can be generated with few neurons

we usually think of learning/optimization as setting up a similar neural machine, which is then responsible for online generation of behavior

yet most neurons in your brain are doing something when you move... perhaps online re-optimization of the movement?

small brain:
box of chocolates / behaviors

large brain:
chocolate factory / behavior optimizer

If you have expensive machinery, you should use it all the time.
Contact-aware optimization

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

These variables affect the dynamics and cost function, and are optimized along with the movement trajectory.

The cost has the extra term

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

where $\mathbf{e}$ is the vector distance to the nearest surface.

The planned contact impulse $f$ is penalized in full, but scaled by $c$ before being applied.
Trajectory optimization methods

General procedure:
- divide the relevant variables (positions, velocities, torques, contact forces) into independent and dependent sets; compute the latter given the former
- if the independent variables are physically coupled, impose constraints
- use contact smoothing to ensure that the cost function is differentiable
- apply a suitable 2\textsuperscript{nd}-order optimization method, usually with continuation

<table>
<thead>
<tr>
<th>Independent</th>
<th>Dependent (computation)</th>
<th>Constraints</th>
<th>Smoothing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Torques</td>
<td>Positions (integration)</td>
<td>n/a</td>
<td>Iterative contact approximation</td>
</tr>
<tr>
<td></td>
<td>Velocities (forward dynamics)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Contact forces (contact solver)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Torques Contact forces</td>
<td>Positions (integration)</td>
<td>Contact &amp; friction</td>
<td>Soft constraints</td>
</tr>
<tr>
<td></td>
<td>Velocities (forward dynamics)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Positions</td>
<td>Velocities (differentiation)</td>
<td>Under-actuation</td>
<td>Finite differences, Convex contact model</td>
</tr>
<tr>
<td></td>
<td>Torques (inverse dynamics)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Contact forces (inverse contact solver)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Positions Contact forces</td>
<td>Velocities (differentiation)</td>
<td>Under-actuation</td>
<td>Soft constraints</td>
</tr>
<tr>
<td></td>
<td>Torques (inverse dynamics)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Positions Velocities Torques Contact forces</td>
<td>n/a</td>
<td>Differentiation Dynamics Contact &amp; friction</td>
<td>Soft constraints</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Optimizing through contacts
Finite difference approximations to the derivatives always exist, even for non-differentiable cost functions

\[ f(x) = \text{abs}(x) \]

\[ f'(x) = \frac{f(x + \varepsilon) - f(x - \varepsilon)}{2\varepsilon} \]

\[ f''(x) = \frac{f(x + \varepsilon) + f(x - \varepsilon) - 2f(x)}{\varepsilon^2} \]
Contact smoothing via soft constraints

If the contact forces are treated as independent variables and physical realism is enforced via penalty functions, then the cost function is differentiable.

Coulomb friction model

\[ v = Af + b \]
\[ v_N \perp f_N \]
\[ \|v_T\| \perp \mu f_N - \|f_T\| \]
\[ v_T = \alpha f_T, \quad \alpha \leq 0 \]

<table>
<thead>
<tr>
<th>( x \perp y )</th>
<th>complementarity: ( x \geq 0, y \geq 0, xy = 0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A )</td>
<td>contact inverse inertia: ( A = JM^{-1}J^T )</td>
</tr>
<tr>
<td>( b )</td>
<td>contact velocity before impulse</td>
</tr>
<tr>
<td>( f )</td>
<td>contact impulse</td>
</tr>
<tr>
<td>( \nu )</td>
<td>contact velocity after impulse</td>
</tr>
</tbody>
</table>

Penalty functions instead of hard constraints:

replace \( x \perp y \) with one of \( \min(x, y)^2 \)
\[
\left( x + y - \sqrt{x^2 + y^2} \right)^2
\]

...
The contact impulse $f$ is the solution to a linear equation:

$$v = Af + b$$

subject to complicated but easy-to-enforce constraints:

\begin{align*}
    v_N & \perp f_N \\
    \|v_T\| & \perp \mu f_N - \|f_T\| \\
    v_T & = \alpha f_T, \quad \alpha \leq 0
\end{align*}

Iterative solvers tend to work well and produce smooth results:
- solve the linear equation
- enforce the constraints softly
- repeat a couple of times
Convex, smooth and invertible contact model

Define the contact impulse by minimizing contact-space kinetic energy

\[ \frac{1}{2} v^T A^{-1} v \]

subject to \( f_N \geq 0, \quad v_N \geq 0, \quad \mu f_N \geq \|f_T\| \) for each contact.

Replace the constraints with penalty functions \( d_F(f), d_V(v) \)

Forward contact dynamics: \((A, b) \rightarrow (f, v = Af + b)\)

\[ f^* = \arg \min_f \frac{1}{2} f^T A f + f^T b + d_F(f) + d_V(Af + b) \]

Inverse contact dynamics: \((A, v) \rightarrow (f, b = v - Af)\)

\[ f^* = \arg \min_f f^T (v + A \nabla d_V(v)) + d_F(f) \]

Todorov

ICRA 2011
Continuation methods

Solve an easier problem first, then gradually make the problem harder and more realistic while tracking the solution: use the last solution as initial condition for the next problem.

How to define easier problems:

- allow “root” forces in the un-actuated space
- allow contact forces to act from a distance
- allow contact penetrations, joint limit violations, limb stretching
- restrict the trajectory to a smooth function (polynomial, Fourier)
MPC: plan with iLQG until next contact (adaptive horizon). Contact dynamics handled implicitly: heuristic value function used as final cost (encodes what is a good way to hit the ball).

MPC: plan with iLQG 0.5 sec into the future (fixed horizon). Smoothing due to soft approximation to contact.

Space-time optimization: Fourier representation of trajectory. Contact smoothing due to inverse contact solver. Continuation: root forces, contact forces from a distance.

Space-time optimization: spline representation of trajectory. Contact smoothing due to finite differencing. Auxiliary decision variables for all potential contacts. Continuation: contact forces from a distance.

Similar to above, but contact forces and their origins are now independent variables, and contact smoothing is due to soft constraints.
Future applications
Robot hands we are building

ModBot
(Simpkins, Kelley)

ShadowHand
(Kumar, Xu, Simpkins)

3D-printed hand
(Xu, Kumar)
DARPA Robotics Challenge

**Subtasks:**

- Drive utility vehicle to site
- Walk across rubble
- Remove debris blocking door
- Open door, enter building
- Climb ladder and stairs
- Use tool to break concrete panel
- Close leaking valve
- Replace component
Brain-machine interfaces

Donoghue et al.  
*Nature* 2012
Robotic control works better

Existing methods for automatic control are faster and more accurate than BMIs, and more advanced and flexible methods are being developed.

Incorporating human intelligence in the control loop should make things better, not worse.
Task-level brain-machine interfaces

Human user -> **Task specification**
Computer vision -> **Environment model**

Automatic controller

Kumar and Todorov

Use **eye-tracking** to point to objects and locations of interest
Use **speech recognition** for high-level commands and go signals
Leave the rest to **optimal control**