Pontryagin's maximum principle

Emo Todorov

Applied Mathematics and Computer Science & Engineering

University of Washington

Winter 2012

Pontryagin's maximum principle

For deterministic dynamics $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$ we can compute *extremal* open-loop trajectories (i.e. local minima) by solving a boundary-value ODE problem with given $\mathbf{x}(0)$ and $\boldsymbol{\lambda}(T) = \frac{\partial}{\partial \mathbf{x}} q_{\mathcal{T}}(x)$, where $\boldsymbol{\lambda}(t)$ is the gradient of the optimal cost-to-go function (called *costate*).

Pontryagin's maximum principle

For deterministic dynamics $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$ we can compute *extremal* open-loop trajectories (i.e. local minima) by solving a boundary-value ODE problem with given $\mathbf{x}(0)$ and $\boldsymbol{\lambda}(T) = \frac{\partial}{\partial \mathbf{x}} q_{\mathcal{T}}(x)$, where $\boldsymbol{\lambda}(t)$ is the gradient of the optimal cost-to-go function (called *costate*).

Definition (deterministic Hamiltonian)

$$\overline{H}(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda}) \triangleq \ell(\mathbf{x}, \mathbf{u}) + \mathbf{f}(\mathbf{x}, \mathbf{u})^{\mathsf{T}} \boldsymbol{\lambda}$$

Theorem (continuous-time maximum principle)

If $\mathbf{x}(t), \mathbf{u}(t)$, $0 \leq t \leq T$ *is the optimal state-control trajectory starting at* $\mathbf{x}(0)$, *then there exists a costate trajectory* $\boldsymbol{\lambda}(t)$ *with* $\boldsymbol{\lambda}(T) = \frac{\partial}{\partial \mathbf{x}} q_{\mathcal{T}}(\mathbf{x})$ *satisfying*

$$\begin{aligned} \dot{\mathbf{x}} &= \overline{H}_{\lambda}(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda}) = \mathbf{f}(\mathbf{x}, \mathbf{u}) \\ -\dot{\boldsymbol{\lambda}} &= \overline{H}_{\mathbf{x}}(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda}) = \ell_{\mathbf{x}}(\mathbf{x}, \mathbf{u}) + \mathbf{f}_{\mathbf{x}}(\mathbf{x}, \mathbf{u})^{\mathsf{T}} \boldsymbol{\lambda} \\ \mathbf{u} &= \arg\min_{\widetilde{\mathbf{u}}} \overline{H}(\mathbf{x}, \widetilde{\mathbf{u}}, \boldsymbol{\lambda}) \end{aligned}$$

Derivation from the HJB equation (continuous time)

For deterministic dynamics $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$ the optimal cost-to-go in the finite-horizon setting satisfies the HJB equation

$$-v_t(\mathbf{x}, t) = \min_{\mathbf{u}} \left\{ \ell(\mathbf{x}, \mathbf{u}) + \mathbf{f}(\mathbf{x}, \mathbf{u})^\mathsf{T} v_{\mathbf{x}}(\mathbf{x}, t) \right\} = \min_{\mathbf{u}} \overline{H}(\mathbf{x}, \mathbf{u}, v_{\mathbf{x}}(\mathbf{x}, t))$$

If the optimal control law is $\boldsymbol{\pi}(\mathbf{x}, t)$, we can set $\mathbf{u} = \boldsymbol{\pi}$ and drop the 'min':

$$0 = v_t(\mathbf{x}, t) + \ell(\mathbf{x}, \boldsymbol{\pi}(\mathbf{x}, t)) + \mathbf{f}(\mathbf{x}, \boldsymbol{\pi}(\mathbf{x}, t))^\mathsf{T} v_{\mathbf{x}}(\mathbf{x}, t)$$

Now differentiate w.r.t. $\mathbf{x}$ and suppress the dependences for clarity:

$$0 = v_{t\mathbf{x}} + \ell_{\mathbf{x}} + \boldsymbol{\pi}_{\mathbf{x}}^\mathsf{T} \ell_{\mathbf{u}} + \left(\mathbf{f}_{\mathbf{x}}^\mathsf{T} + \boldsymbol{\pi}_{\mathbf{x}}^\mathsf{T} \mathbf{f}_{\mathbf{u}}^\mathsf{T} \right) v_{\mathbf{x}} + v_{\mathbf{xx}} \mathbf{f}$$

Using the identity $\dot{v}_{\mathbf{x}} = v_{t\mathbf{x}} + v_{\mathbf{xx}} \mathbf{f}$ and regrouping yields

$$0 = \dot{v}_{\mathbf{x}} + \ell_{\mathbf{x}} + \mathbf{f}_{\mathbf{x}}^\mathsf{T} v_{\mathbf{x}} + \boldsymbol{\pi}_{\mathbf{x}}^\mathsf{T} \left(\ell_{\mathbf{u}} + \mathbf{f}_{\mathbf{u}}^\mathsf{T} v_{\mathbf{x}} \right) = \dot{v}_{\mathbf{x}} + \overline{H}_{\mathbf{x}} + \boldsymbol{\pi}_{\mathbf{x}}^\mathsf{T} \overline{H}_{\mathbf{u}}$$

Since $\mathbf{u}$ is optimal we have $\overline{H}_{\mathbf{u}} = 0$, thus $-\dot{\boldsymbol{\lambda}} = \overline{H}_{\mathbf{x}}(\mathbf{x}, \boldsymbol{\pi}, \boldsymbol{\lambda})$ where $\lambda = v_{\mathbf{x}}$.

Derivation via Largrange multipliers (discrete time)

Optimize total cost subject to dynamics constraints $\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$.
Define the Lagrangian $L(\mathbf{x}_\cdot, \mathbf{u}_\cdot, \boldsymbol{\lambda}_\cdot)$ as

$$\begin{aligned} L &= q_{\mathcal{T}}(\mathbf{x}_N) + \textstyle\sum_{k=0}^{N-1} \ell(\mathbf{x}_k, \mathbf{u}_k) + (\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) - \mathbf{x}_{k+1})^\mathsf{T} \boldsymbol{\lambda}_{k+1} \\ &= q_{\mathcal{T}}(\mathbf{x}_N) - \mathbf{x}_N^\mathsf{T}\boldsymbol{\lambda}_N + \mathbf{x}_0^\mathsf{T}\boldsymbol{\lambda}_0 + \textstyle\sum_{k=0}^{N-1} \overline{H}(\mathbf{x}_k, \mathbf{u}_k, \boldsymbol{\lambda}_{k+1}) - \mathbf{x}_k^\mathsf{T}\boldsymbol{\lambda}_k \end{aligned}$$

Setting $L_\mathbf{x} = L_\boldsymbol{\lambda} = 0$ and explicitly minimizing w.r.t. $\mathbf{u}$ yields

Theorem (discrete-time maximum principle)

If $\mathbf{x}_k, \mathbf{u}_k$, $0 \le k \le N$ is the optimal state-control trajectory starting at $\mathbf{x}_0$, then there exists a costate trajectory $\boldsymbol{\lambda}_k$ with $\boldsymbol{\lambda}_N = \frac{\partial}{\partial \mathbf{x}} q_{\mathcal{T}}(\mathbf{x}_N)$ satisfying

$$\begin{aligned} \mathbf{x}_{k+1} &= \overline{H}_\boldsymbol{\lambda}(\mathbf{x}_k, \mathbf{u}_k, \boldsymbol{\lambda}_{k+1}) = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) \\ \boldsymbol{\lambda}_k &= \overline{H}_\mathbf{x}(\mathbf{x}_k, \mathbf{u}_k, \boldsymbol{\lambda}_{k+1}) = \ell_\mathbf{x}(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{f}_\mathbf{x}(\mathbf{x}_k, \mathbf{u}_k)^\mathsf{T} \boldsymbol{\lambda}_{k+1} \\ \mathbf{u}_k &= \arg\min_{\widetilde{\mathbf{u}}} \overline{H}(\mathbf{x}_k, \widetilde{\mathbf{u}}, \boldsymbol{\lambda}_{k+1}) \end{aligned}$$

Gradient of the total cost

The maximum principle provides an efficient way to evaluate the gradient of the total cost w.r.t. $\mathbf{u}$, and thereby optimize the controls numerically.

Theorem (gradient)

For given control trajectory $\mathbf{u}_k$*, let* $\mathbf{x}_k, \boldsymbol{\lambda}_k$ *be such that*

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right) \\ \boldsymbol{\lambda}_k &= \ell_{\mathbf{x}}\left(\mathbf{x}_k, \mathbf{u}_k\right) + \mathbf{f}_{\mathbf{x}}\left(\mathbf{x}_k, \mathbf{u}_k\right)^{\mathsf{T}} \boldsymbol{\lambda}_{k+1} \end{aligned}$$

with $\mathbf{x}_0$ *given and* $\boldsymbol{\lambda}_N = \frac{\partial}{\partial \mathbf{x}} q_{\mathcal{T}}\left(\mathbf{x}_N\right)$*. Let* $J\left(\mathbf{x}_\cdot, \mathbf{u}_\cdot\right)$ *be the total cost. Then*

$$\frac{\partial}{\partial \mathbf{u}_k} J\left(\mathbf{x}_\cdot, \mathbf{u}_\cdot\right) = \overline{H}_{\mathbf{u}}\left(\mathbf{x}_k, \mathbf{u}_k, \boldsymbol{\lambda}_{k+1}\right) = \ell_{\mathbf{u}}\left(\mathbf{x}_k, \mathbf{u}_k\right) + \mathbf{f}_{\mathbf{u}}\left(\mathbf{x}_k, \mathbf{u}_k\right)^{\mathsf{T}} \boldsymbol{\lambda}_{k+1}$$

Note that $\mathbf{x}_k$ can be found in a forward pass (since it does not depend on $\boldsymbol{\lambda}$), and then $\boldsymbol{\lambda}_k$ can be found in a backward pass.

Proof by induction

The cost accumulated from time k until the end can be written recursively as

$$J_k\left(\mathbf{x}_{k\cdots N}, \mathbf{u}_{k\cdots N-1}\right) = \ell\left(\mathbf{x}_k, \mathbf{u}_k\right) + J_{k+1}\left(\mathbf{x}_{k+1\cdots N}, \mathbf{u}_{k+1\cdots N-1}\right)$$

Noting that $\mathbf{u}_k$ affects future costs only through $\mathbf{x}_{k+1} = \mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right)$, we have

$$\frac{\partial}{\partial \mathbf{u}_k} J_k = \ell_{\mathbf{u}}\left(\mathbf{x}_k, \mathbf{u}_k\right) + \mathbf{f}_{\mathbf{u}}\left(\mathbf{x}_k, \mathbf{u}_k\right)^{\mathsf{T}} \frac{\partial}{\partial \mathbf{x}_{k+1}} J_{k+1}$$

We need to show that $\boldsymbol{\lambda}_k = \frac{\partial}{\partial \mathbf{x}_k} J_k$. For $k = N$ this holds because $J_N = q_{\mathcal{T}}$.
For $k < N$ we have

$$\frac{\partial}{\partial \mathbf{x}_k} J_k = \ell_{\mathbf{x}}\left(\mathbf{x}_k, \mathbf{u}_k\right) + \mathbf{f}_{\mathbf{x}}\left(\mathbf{x}_k, \mathbf{u}_k\right)^{\mathsf{T}} \frac{\partial}{\partial \mathbf{x}_{k+1}} J_{k+1}$$

which is identical to $\boldsymbol{\lambda}_k = \ell_{\mathbf{x}}\left(\mathbf{x}_k, \mathbf{u}_k\right) + \mathbf{f}_{\mathbf{x}}\left(\mathbf{x}_k, \mathbf{u}_k\right)^{\mathsf{T}} \boldsymbol{\lambda}_{k+1}$.

Enforcing terminal states

- The final state $\mathbf{x}(T)$ is usually different from the minimum of the final cost $q_{\mathcal{T}}$, because it reflects a trade-off between final and running cost.
- We can enforce $\mathbf{x}(T) = \overline{\mathbf{x}}$ as a boundary condition and remove the boundary condition on $\boldsymbol{\lambda}(T)$.
- Once the solution is found, we can construct a function $q_{\mathcal{T}}$ such that $\boldsymbol{\lambda}(T) = \frac{\partial}{\partial \mathbf{x}} q_{\mathcal{T}}(\mathbf{x}(T))$. However if $\boldsymbol{\lambda}(T) \neq 0$ then $\mathbf{x}(T)$ is not the minimum of this $q_{\mathcal{T}}$.
- We can also define the problem as infinite horizon average cost, in which case it is usually suboptimal to have an asymptotic state different from the minimum of the state cost function. The maximum principle does not apply to infinite horizon problems, so one has to use the HJB equations.

More tractable problems

When the dynamics and cost are in the restricted form

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{a}(\mathbf{x}) + B\mathbf{u} \\ \ell(\mathbf{x}, \mathbf{u}) &= q(\mathbf{x}) + \frac{1}{2}\mathbf{u}^{\mathsf{T}} R\mathbf{u} \end{aligned}$$

the Hamiltonian can be minimized analytically, which yields the ODE

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{a}(\mathbf{x}) - BR^{-1}B^{\mathsf{T}}\boldsymbol{\lambda} \\ -\dot{\boldsymbol{\lambda}} &= q_{\mathbf{x}}(\mathbf{x}) + \mathbf{a}_{\mathbf{x}}(\mathbf{x})^{\mathsf{T}}\boldsymbol{\lambda} \end{aligned}$$

with boundary conditions $\mathbf{x}(0)$ and $\boldsymbol{\lambda}(T) = \frac{\partial}{\partial \mathbf{x}} q_{\mathcal{T}}(\mathbf{x})$. If B, R depend on $\mathbf{x}$, the second equation has additional terms involving the derivatives of B, R.

More tractable problems

When the dynamics and cost are in the restricted form

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{a}(\mathbf{x}) + B\mathbf{u} \\ \ell(\mathbf{x}, \mathbf{u}) &= q(\mathbf{x}) + \frac{1}{2}\mathbf{u}^\mathsf{T} R\mathbf{u} \end{aligned}$$

the Hamiltonian can be minimized analytically, which yields the ODE

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{a}(\mathbf{x}) - BR^{-1}B^\mathsf{T}\boldsymbol{\lambda} \\ -\dot{\boldsymbol{\lambda}} &= q_{\mathbf{x}}(\mathbf{x}) + \mathbf{a}_{\mathbf{x}}(\mathbf{x})^\mathsf{T}\boldsymbol{\lambda} \end{aligned}$$

with boundary conditions $\mathbf{x}(0)$ and $\boldsymbol{\lambda}(T) = \frac{\partial}{\partial \mathbf{x}} q_{\mathcal{T}}(\mathbf{x})$. If B, R depend on $\mathbf{x}$, the second equation has additional terms involving the derivatives of B, R.

We have $\overline{H}_{\mathbf{u}} = R(\mathbf{x})\mathbf{u} + B(\mathbf{x})^\mathsf{T}\boldsymbol{\lambda}$ and $\overline{H}_{\mathbf{uu}} = R(\mathbf{x}) \succ 0$. Thus the maximum principle here is both a necessary and a sufficient condition for a local minimum.

Pendulum example

Passive dynamics:

$$\begin{aligned}\mathbf{a}(\mathbf{x}) &= \begin{bmatrix} x_2 \\ k\sin(x_1) \end{bmatrix} \\ \mathbf{a}_{\mathbf{x}}(\mathbf{x}) &= \begin{bmatrix} 0 & 1 \\ k\cos(x_1) & 0 \end{bmatrix}\end{aligned}$$

Optimal control:

$$u = -r^{-1}\lambda_2$$

ODE (with $q = 0$):

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= k\sin(x_1) - r^{-1}\lambda_2 \\ -\dot{\lambda}_1 &= k\cos(x_1)\lambda_2 \\ -\dot{\lambda}_2 &= \lambda_1\end{aligned}$$

Pendulum example

Passive dynamics:

$$\mathbf{a}(\mathbf{x}) = \begin{bmatrix} x_2 \\ k \sin(x_1) \end{bmatrix}$$

$$\mathbf{a_x}(\mathbf{x}) = \begin{bmatrix} 0 & 1 \\ k \cos(x_1) & 0 \end{bmatrix}$$

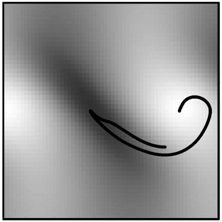
Optimal control:

$$u = -r^{-1}\lambda_2$$

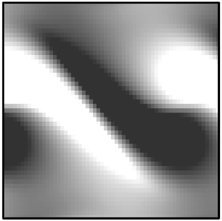
ODE (with $q = 0$):

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= k \sin(x_1) - r^{-1}\lambda_2 \\ -\dot{\lambda}_1 &= k \cos(x_1)\lambda_2 \\ -\dot{\lambda}_2 &= \lambda_1 \end{aligned}$$

Cost-to-go and trajectories:



Control law (from HJB):



Trajectory-based optimization

Emo Todorov

Applied Mathematics and Computer Science & Engineering

University of Washington

Winter 2012

Using the maximum principle

Recall that for deterministic dynamics $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$ and cost rate $\ell(\mathbf{x}, \mathbf{u})$ the optimal state-control-costate trajectory $(\mathbf{x}(\cdot), \mathbf{u}(\cdot), \boldsymbol{\lambda}(\cdot))$ satisfies

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}, \mathbf{u}) \\ -\dot{\boldsymbol{\lambda}} &= \ell_{\mathbf{x}}(\mathbf{x}, \mathbf{u}) + \mathbf{f}_{\mathbf{x}}(\mathbf{x}, \mathbf{u})^{\mathsf{T}} \boldsymbol{\lambda} \\ \mathbf{u} &= \arg\min_{\widetilde{\mathbf{u}}} \left\{ \ell(\mathbf{x}, \widetilde{\mathbf{u}}) + \mathbf{f}(\mathbf{x}, \widetilde{\mathbf{u}})^{\mathsf{T}} \boldsymbol{\lambda} \right\} \end{aligned}$$

with $\mathbf{x}(0)$ given and $\boldsymbol{\lambda}(T) = \frac{\partial}{\partial \mathbf{x}} q_{\mathcal{T}}(\mathbf{x}(T))$. Solving this boundary-value ODE problem numerically is a trajectory-based method.

Using the maximum principle

Recall that for deterministic dynamics $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$ and cost rate $\ell(\mathbf{x}, \mathbf{u})$ the optimal state-control-costate trajectory $(\mathbf{x}(\cdot), \mathbf{u}(\cdot), \boldsymbol{\lambda}(\cdot))$ satisfies

$$
\begin{aligned}
\dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}, \mathbf{u}) \\
-\dot{\boldsymbol{\lambda}} &= \ell_{\mathbf{x}}(\mathbf{x}, \mathbf{u}) + \mathbf{f}_{\mathbf{x}}(\mathbf{x}, \mathbf{u})^{\mathsf{T}} \boldsymbol{\lambda} \\
\mathbf{u} &= \arg\min_{\widetilde{\mathbf{u}}} \left\{ \ell(\mathbf{x}, \widetilde{\mathbf{u}}) + \mathbf{f}(\mathbf{x}, \widetilde{\mathbf{u}})^{\mathsf{T}} \boldsymbol{\lambda} \right\}
\end{aligned}
$$

with $\mathbf{x}(0)$ given and $\boldsymbol{\lambda}(T) = \frac{\partial}{\partial \mathbf{x}} q_{\mathcal{T}}(\mathbf{x}(T))$. Solving this boundary-value ODE problem numerically is a trajectory-based method.

We can also use the fact that, if $(\mathbf{x}(\cdot), \boldsymbol{\lambda}(\cdot))$ satisfies the ODE for some $\mathbf{u}(\cdot)$ which is not a minimizer of the Hamiltonian $H(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda}) = \ell(\mathbf{x}, \mathbf{u}) + \mathbf{f}(\mathbf{x}, \mathbf{u})^{\mathsf{T}} \boldsymbol{\lambda}$, then the gradient of the total cost J is given by

$$
\begin{aligned}
J(\mathbf{x}(\cdot), \mathbf{u}(\cdot)) &= q_{\mathcal{T}}(\mathbf{x}(T)) + \int_0^T \ell(\mathbf{x}(t), \mathbf{u}(t))\, dt \\
\frac{\partial J}{\partial \mathbf{u}(t)} &= H_{\mathbf{u}}(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda}) = \ell_{\mathbf{u}}(\mathbf{x}, \mathbf{u}) + \mathbf{f}_{\mathbf{u}}(\mathbf{x}, \mathbf{u})^{\mathsf{T}} \boldsymbol{\lambda}
\end{aligned}
$$

Thus we can perform gradient descent on J with respect to $\mathbf{u}(\cdot)$

Compact representations

Given the current $\mathbf{u}(\cdot)$, each step of the algorithm involves computing $\mathbf{x}(\cdot)$ by integrating forward in time starting with the given $\mathbf{x}(0)$, then computing $\boldsymbol{\lambda}(\cdot)$ by integrating backward in time starting with $\boldsymbol{\lambda}(T) = \frac{\partial}{\partial \mathbf{x}} q_{\mathcal{T}}(\mathbf{x}(T))$.

Compact representations

Given the current $\mathbf{u}(\cdot)$, each step of the algorithm involves computing $\mathbf{x}(\cdot)$ by integrating forward in time starting with the given $\mathbf{x}(0)$, then computing $\boldsymbol{\lambda}(\cdot)$ by integrating backward in time starting with $\boldsymbol{\lambda}(T) = \frac{\partial}{\partial \mathbf{x}} q_{\mathcal{T}}(\mathbf{x}(T))$.

One way to implement the above methods is to discretize the time axis and represent $(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda})$ independently at each time step. This may be inefficient because the values at nearby time steps are usually very similar, thus it is a waste to represent/optimize them independently.

Compact representations

Given the current $\mathbf{u}(\cdot)$, each step of the algorithm involves computing $\mathbf{x}(\cdot)$ by integrating forward in time starting with the given $\mathbf{x}(0)$, then computing $\boldsymbol{\lambda}(\cdot)$ by integrating backward in time starting with $\boldsymbol{\lambda}(T) = \frac{\partial}{\partial \mathbf{x}} q_{\mathcal{T}}(\mathbf{x}(T))$.

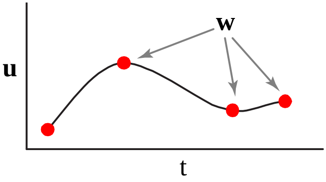
One way to implement the above methods is to discretize the time axis and represent $(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda})$ independently at each time step. This may be inefficient because the values at nearby time steps are usually very similar, thus it is a waste to represent/optimize them independently.

Instead we can splines, Legendre or Chebyshev polynomials, etc.

$$\mathbf{u}(t) = \mathbf{g}(t, \mathbf{w})$$

Gradient:

$$\frac{\partial J}{\partial \mathbf{w}} = \int_0^T \mathbf{g}_{\mathbf{w}}(t, \mathbf{w})^{\mathsf{T}} \frac{\partial J}{\partial \mathbf{u}(t)} dt$$



Space-time constraints

We can also minimize the total cost J as an explicit function of the (parameterized) state-control trajectory:

$$\begin{aligned} \mathbf{x}(t) &= \mathbf{h}(t, \mathbf{v}) \\ \mathbf{u}(t) &= \mathbf{g}(t, \mathbf{w}) \end{aligned}$$

We have to make sure that the state-control trajectory is consistent with the dynamics $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$. This yields a constrained optimization problem:

$$\min_{\mathbf{v},\mathbf{w}} \left\{ q_{\mathcal{T}}(\mathbf{h}(T, \mathbf{v})) + \int_0^T \ell(\mathbf{h}(t, \mathbf{v}), \mathbf{g}(t, \mathbf{w}))\, dt \right\}$$

$$s.t. \quad \frac{\partial \mathbf{h}(t, \mathbf{v})}{\partial t} = \mathbf{f}(\mathbf{h}(t, \mathbf{v}), \mathbf{g}(t, \mathbf{v})), \quad \forall t \in [0, T]$$

Space-time constraints

We can also minimize the total cost J as an explicit function of the (parameterized) state-control trajectory:

$$\begin{aligned}\mathbf{x}(t) &= \mathbf{h}(t,\mathbf{v}) \\ \mathbf{u}(t) &= \mathbf{g}(t,\mathbf{w})\end{aligned}$$

We have to make sure that the state-control trajectory is consistent with the dynamics $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x},\mathbf{u})$. This yields a constrained optimization problem:

$$\min_{\mathbf{v},\mathbf{w}} \left\{ q_{\mathcal{T}}(\mathbf{h}(T,\mathbf{v})) + \int_0^T \ell(\mathbf{h}(t,\mathbf{v}), \mathbf{g}(t,\mathbf{w}))\, dt \right\}$$

$$s.t. \quad \frac{\partial \mathbf{h}(t,\mathbf{v})}{\partial t} = \mathbf{f}(\mathbf{h}(t,\mathbf{v}), \mathbf{g}(t,\mathbf{v})), \quad \forall t \in [0,T]$$

In practice we cannot impose the contraint for all t, so instead we choose a finite set of points $\{t_k\}$ where the constraint is enforced. The same points can also be used to approximate $\int \ell$. There may be no feasible solution (depending on $\mathbf{h}, \mathbf{g}$) in which case we have to live with constraint violations.

This requires no knowledge of optimal control (which may be why it is popular:)

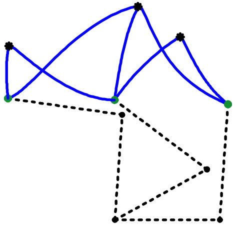
Second-order methods

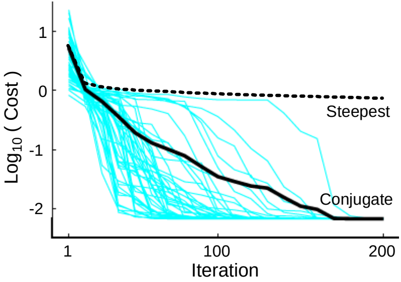
More efficient methods (DDP, iLQG) can be constructed by using the Bellman equations locally. Initialize with some open-loop control $\mathbf{u}^{(0)}(\cdot)$, and repeat:

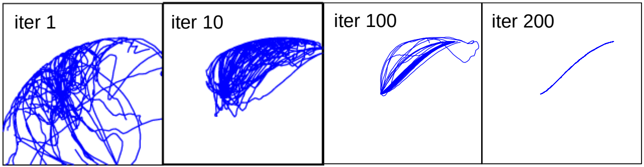
1. Compute the state trajectory $\mathbf{x}^{(n)}(\cdot)$ corresponding to $\mathbf{u}^{(n)}(\cdot)$.
2. Construct a time-varying linear (iLQG) or quadratic (DDP) approximation to the function $\mathbf{f}$ around $\mathbf{x}^{(n)}(\cdot), \mathbf{u}^{(n)}(\cdot)$, which gives the local dynamics in terms of the state and control deviations $\delta\mathbf{x}(\cdot), \delta\mathbf{u}(\cdot)$. Also construct quadratic approximations to the costs ℓ and $q_{\mathcal{T}}$.
3. Compute the locally-optimal cost-to-go $v^{(n)}(\delta\mathbf{x}, t)$ as a quadratic in $\delta\mathbf{x}$. In iLQG this is exact (because the local dynamics are linear and the cost is quadratic) while in DDP this is approximate.
4. Compute the locally-optimal linear feedback control law in the form $\boldsymbol{\pi}^{(n)}(\delta\mathbf{x}, t) = \mathbf{c}(t) - L(t)\,\delta\mathbf{x}$.
5. Apply $\boldsymbol{\pi}^{(n)}$ to the local dynamics (i.e. integrate forward in time) to compute the state-control modification $\delta\mathbf{x}^{(n)}(\cdot), \delta\mathbf{u}^{(n)}(\cdot)$, and set $\mathbf{u}^{(n+1)}(\cdot) = \mathbf{u}^{(n)}(\cdot) + \delta\mathbf{u}^{(n)}(\cdot)$. This requires linesearch to avoid jumping outside the region where the local approximation is valid.

Numerical comparison

	Conj.	ODE	DDP
time(s)	**5.9**	**1.25**	**0.61**







Finding Locally-Optimal, Collision-Free Trajectories with Sequential Convex Optimization

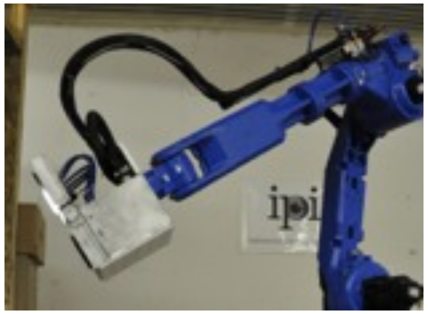
John Schulman, Jonathan Ho, Alex Lee,
Ibrahim Awwal, Henry Bradlow, Pieter Abbeel

Motion Planning

- Sampling-based methods like RRT
- Graph search methods like A*

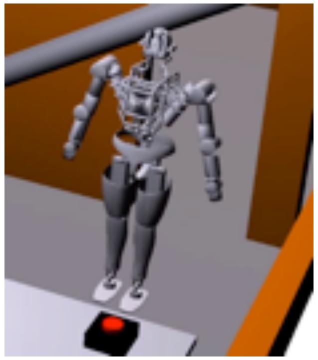
} slow down as dimensionality increases

- Optimization based methods
  - Reactive control
    - Potential-based methods for high-DOF problems (Khatib, ’86)
  - Optimize over the entire trajectory
    - Elastic bands (Quinlan & Khatib, ’93)
    - CHOMP (Ratliff, et al. ‘09) & variants (STOMP, ITOMP)



Industrial robot arm (6 DOF)

Mobile manipulator (18 DOF)



Humanoid (34 DOF)

Trajectory Optimization

$$\min_{\boldsymbol{\theta}_{1:T}} \sum_t \|\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t\|^2 + \text{ other costs}$$

subject to

no collisions ⟵ non-convex

joint limits

other constraints

Trajectory Optimization

$$\min_{\boldsymbol{\theta}_{1:T}} \sum_t \|\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t\|^2 + \text{ other costs}$$

subject to

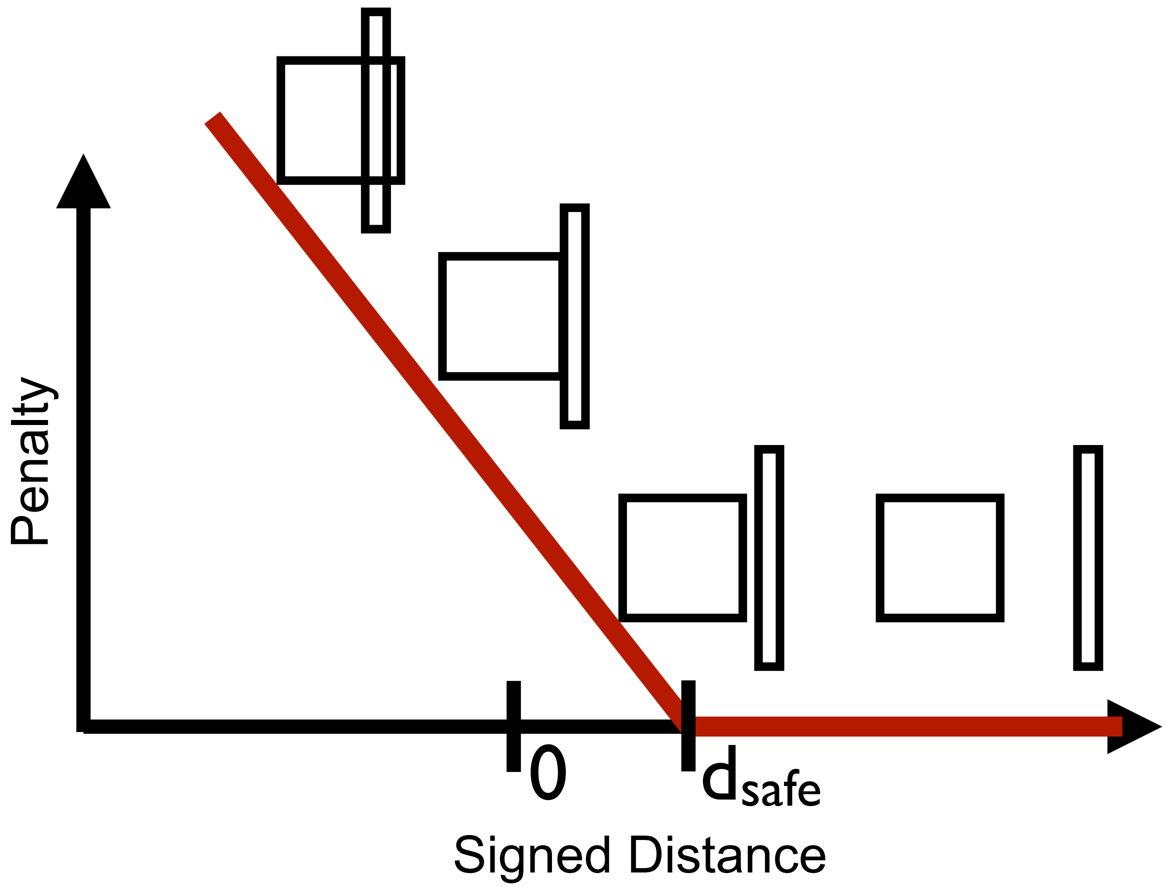
no collisions ⟵ non-convex

joint limits

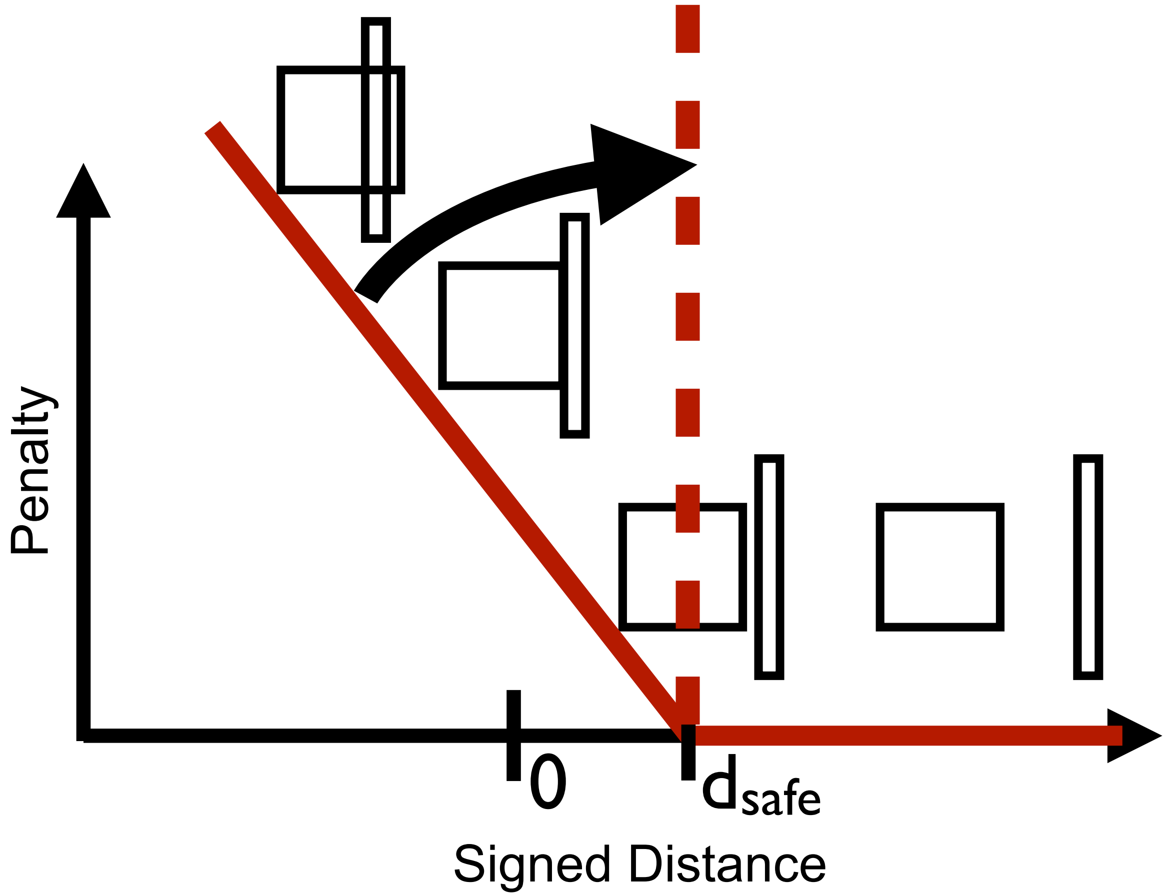
other constraints

- Sequential convex optimization
  - Repeatedly solve local convex approximation
- Challenge
  - Approximating collision constraint

Collision Constraint as L1 Penalty



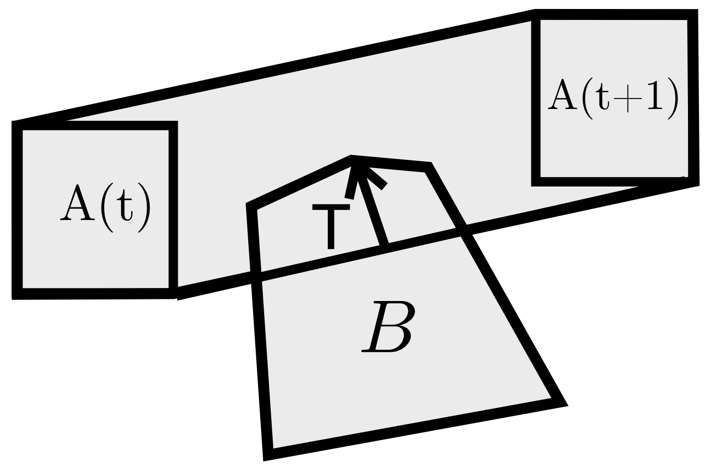
Collision Constraint as L1 Penalty



Linearize w.r.t. degrees of freedom

$$\mathrm{sd}_{AB}(\boldsymbol{\theta}) \approx \mathrm{sd}_{AB}(\boldsymbol{\theta}_0) + \hat{\mathbf{n}}^T J_{\mathbf{p}_A}(\boldsymbol{\theta}_0)(\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

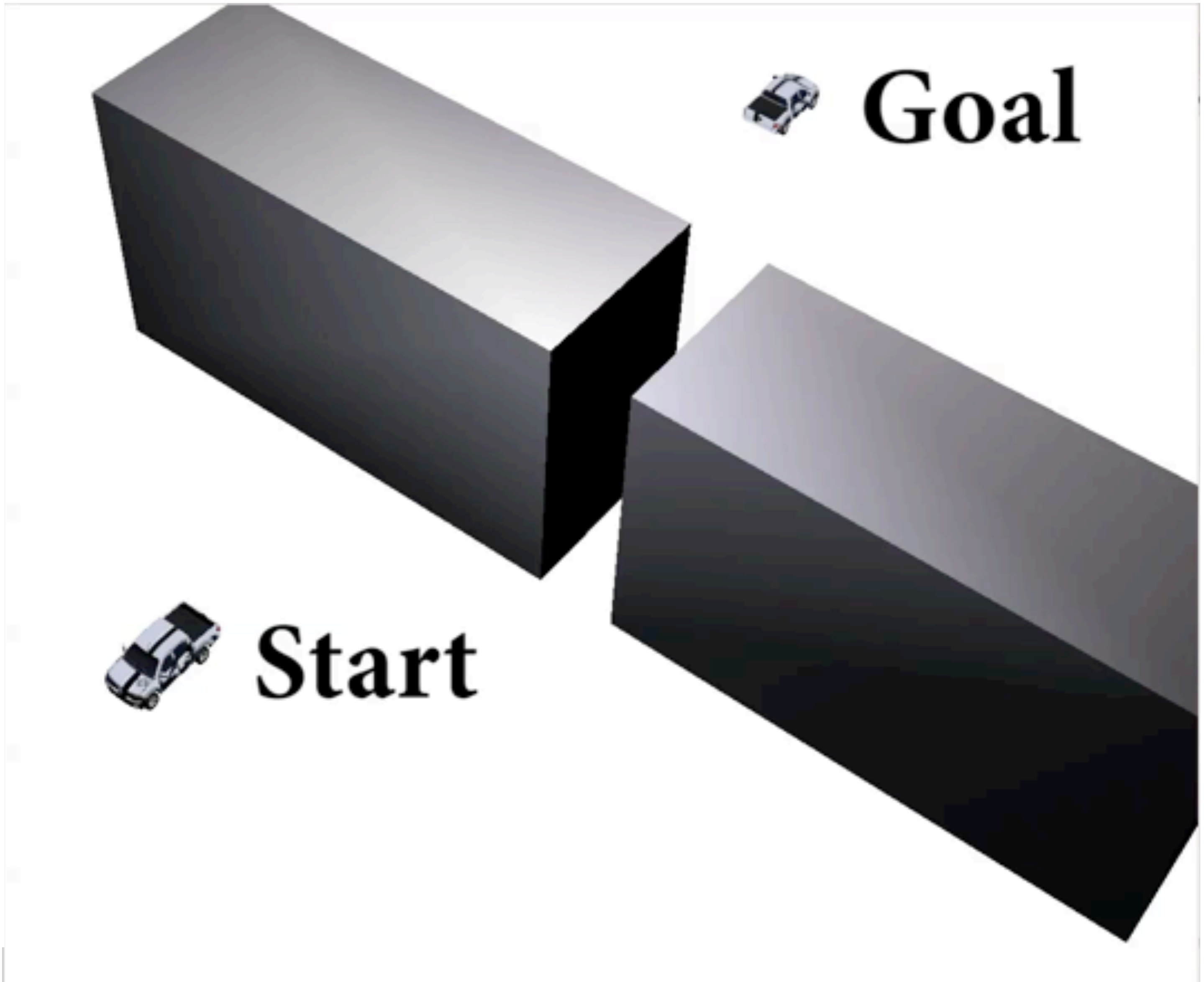
Continuous-Time Safety



Collision check against swept-out volume

- Continuous-time collision avoidance
- Allows coarsely sampling trajectory
  - overall faster
- Finds better local optima

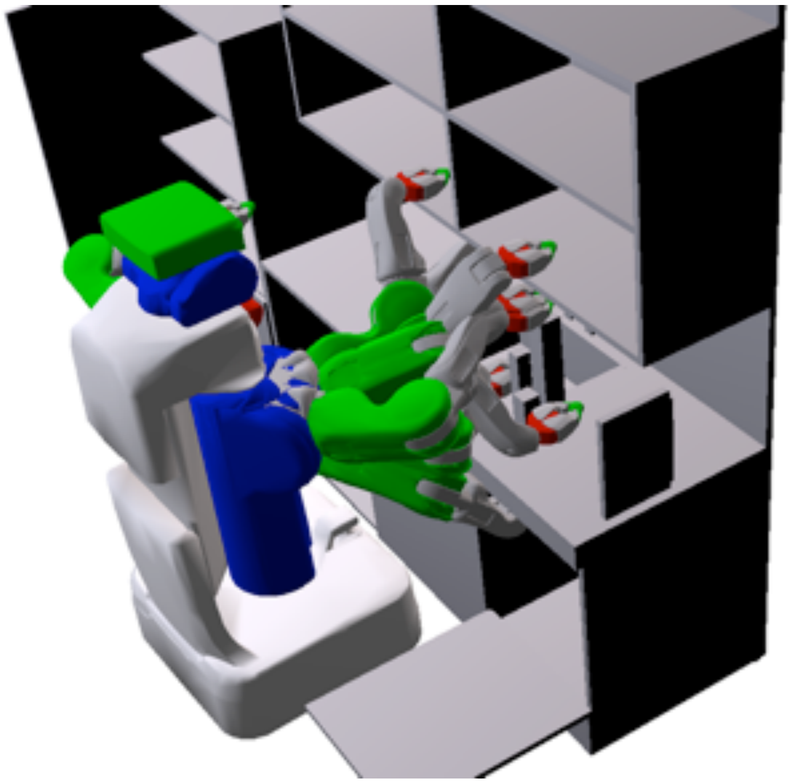
Optimization: Toy Example



Benchmark: Example Scenes

7 DOF (one arm)

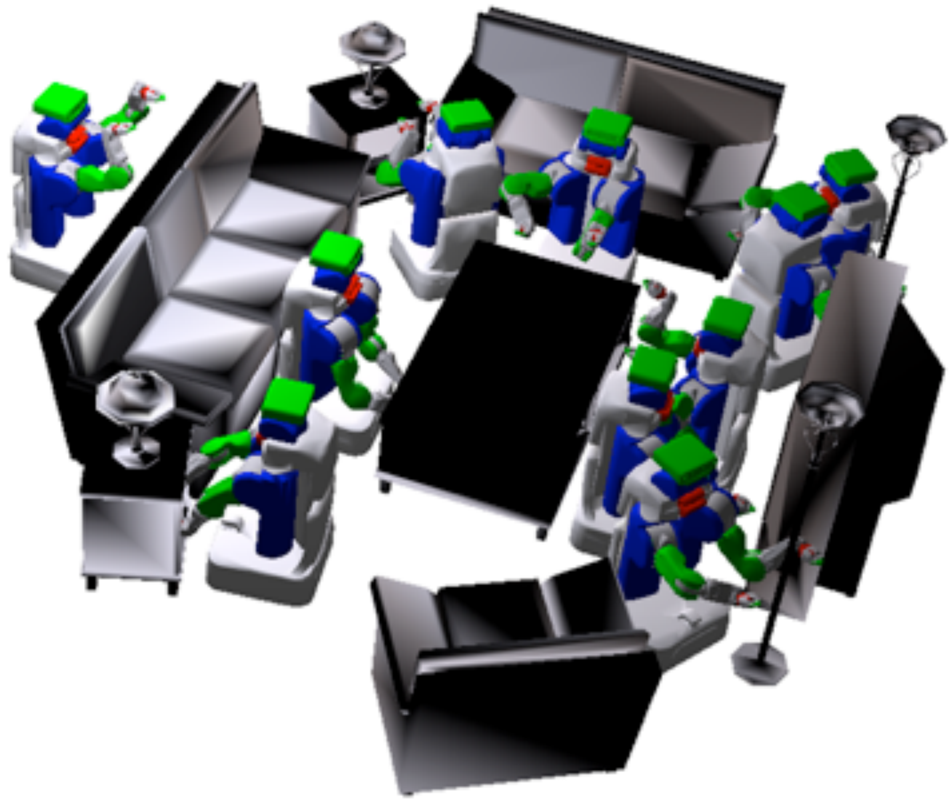
198 problems



example scene (taken from MoveIt collection)

18 DOF (two arms + base + torso)

96 problems



example scene (imported from Trimble 3d Warehouse / Google Sketchup)

Benchmark Results

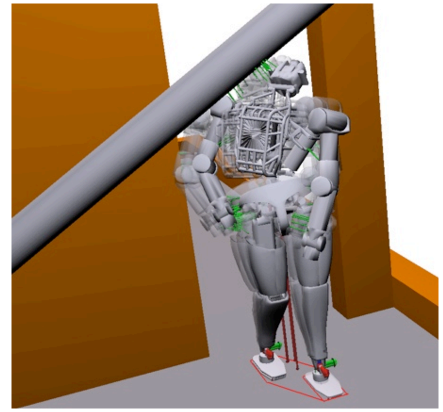
Arm planning (7 DOF) 10s limit			
	Trajopt	BiRRT (*)	CHOMP
success	99%	97%	85%
time (s)	0.32	1.2	6.0
path length	1.2	1.6	2.6

Full body (18 DOF) 30s limit			
	Trajopt	BiRRT (*)	CHOMP (**)
success	84%	53%	N/A
time (s)	7.6	18	N/A
path length	1.1	1.6	N/A

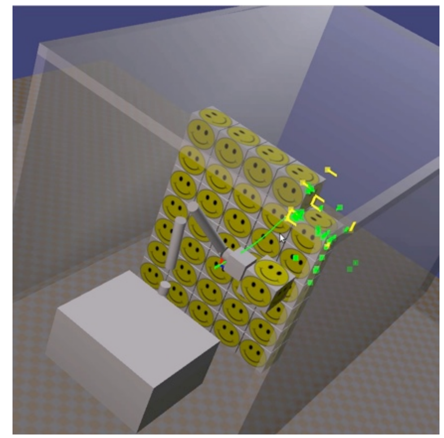
(*) Top-performing algorithm from MoveIt/OMPL
(**) Not supported in available implementation

Other Experiments -- Videos at Interactive Session

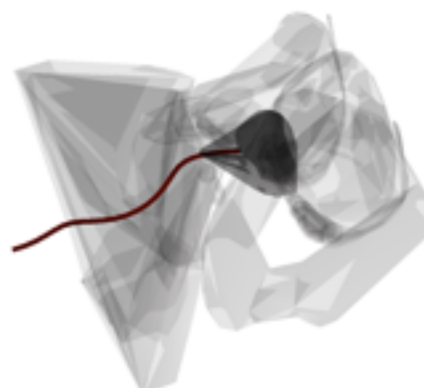
- Planning for 34-DOF humanoid (stability constraints)



- Box picking with industrial robot (orientation constraints)



- Constant-curvature 3D needle steering (non-holonomic constraint)



Try it out yourself!

- Code and docs: rll.berkeley.edu/trajopt
- Run our benchmark: github.com/joschu/planning_benchmark