Trajectory Optimization for Underactuated (and other) Robots

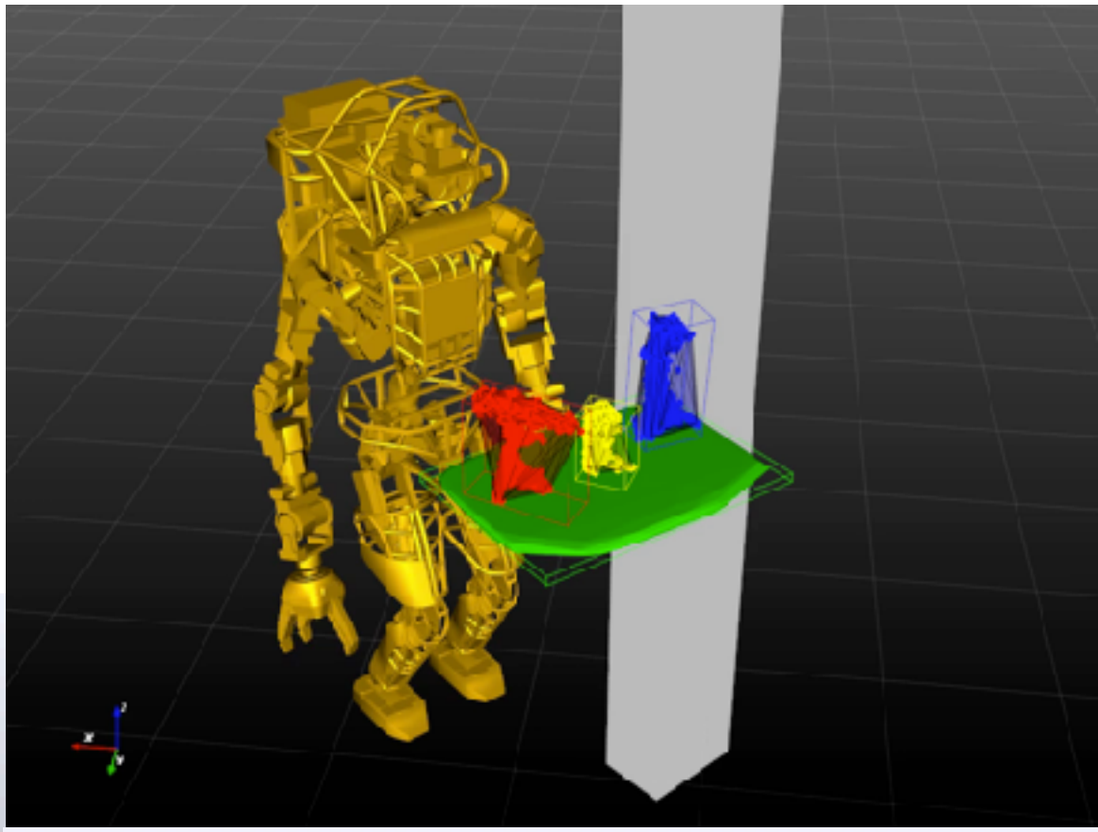
Scott Kuindersma
Harvard Agile Robotics Lab

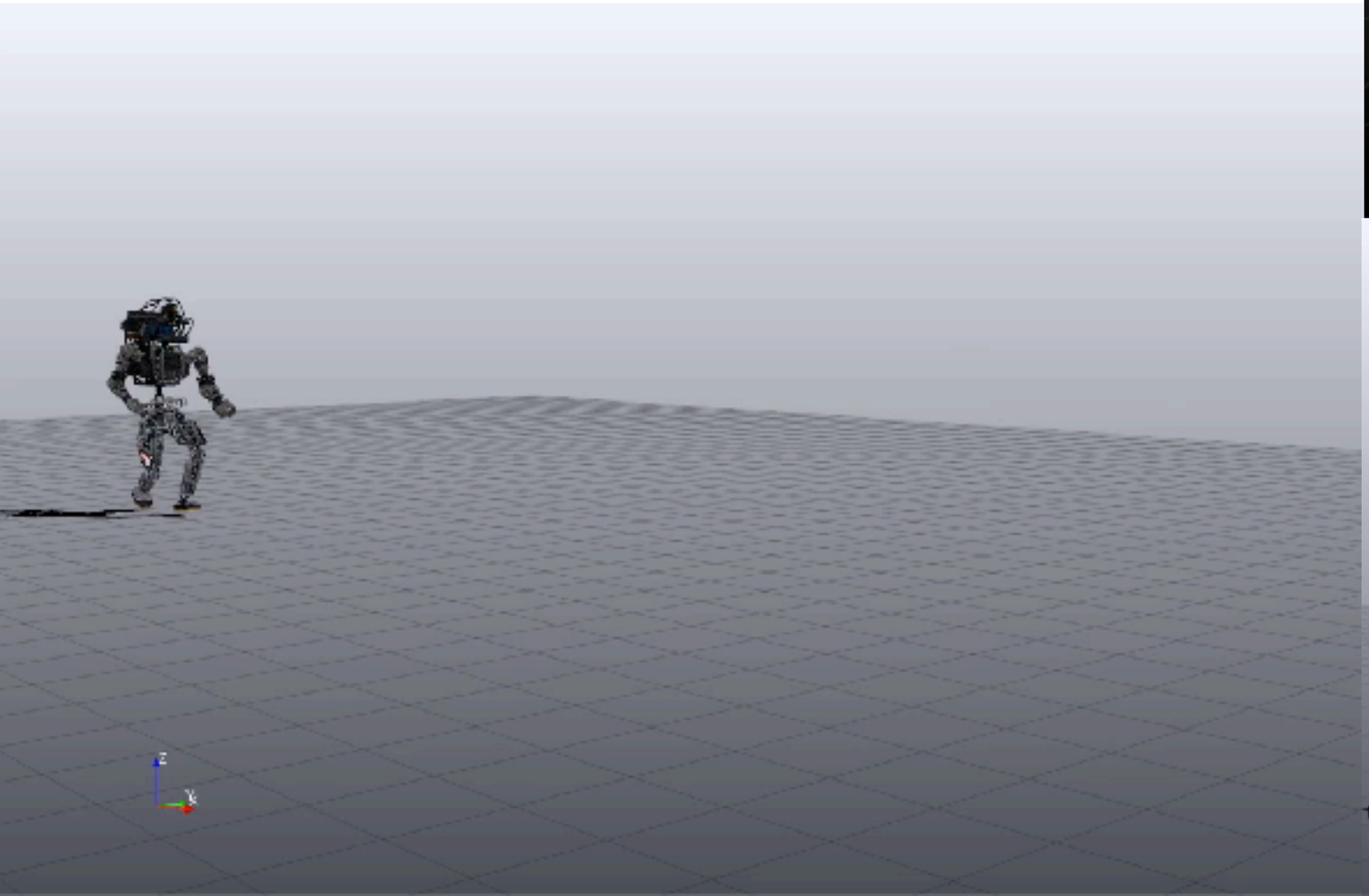
http://agile.seas.harvard.edu
scottk@seas.harvard.edu

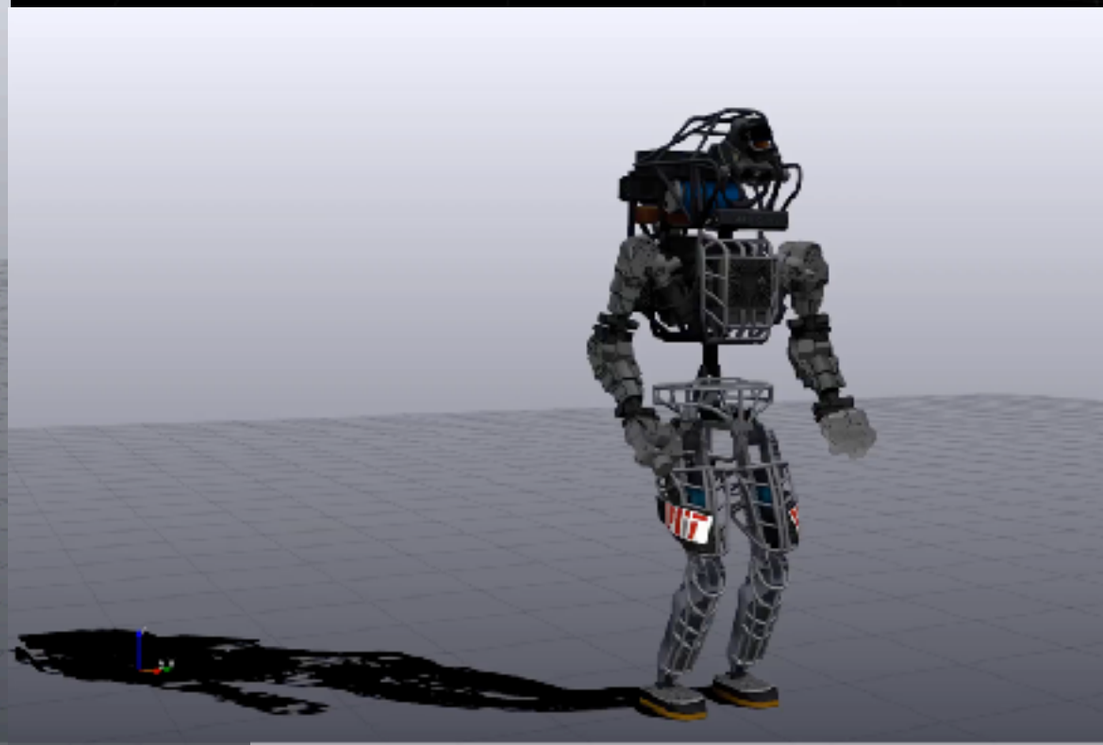
HARVARD
John A. Paulson
School of Engineering
and Applied Sciences

Why Dynamic Motion Planning?

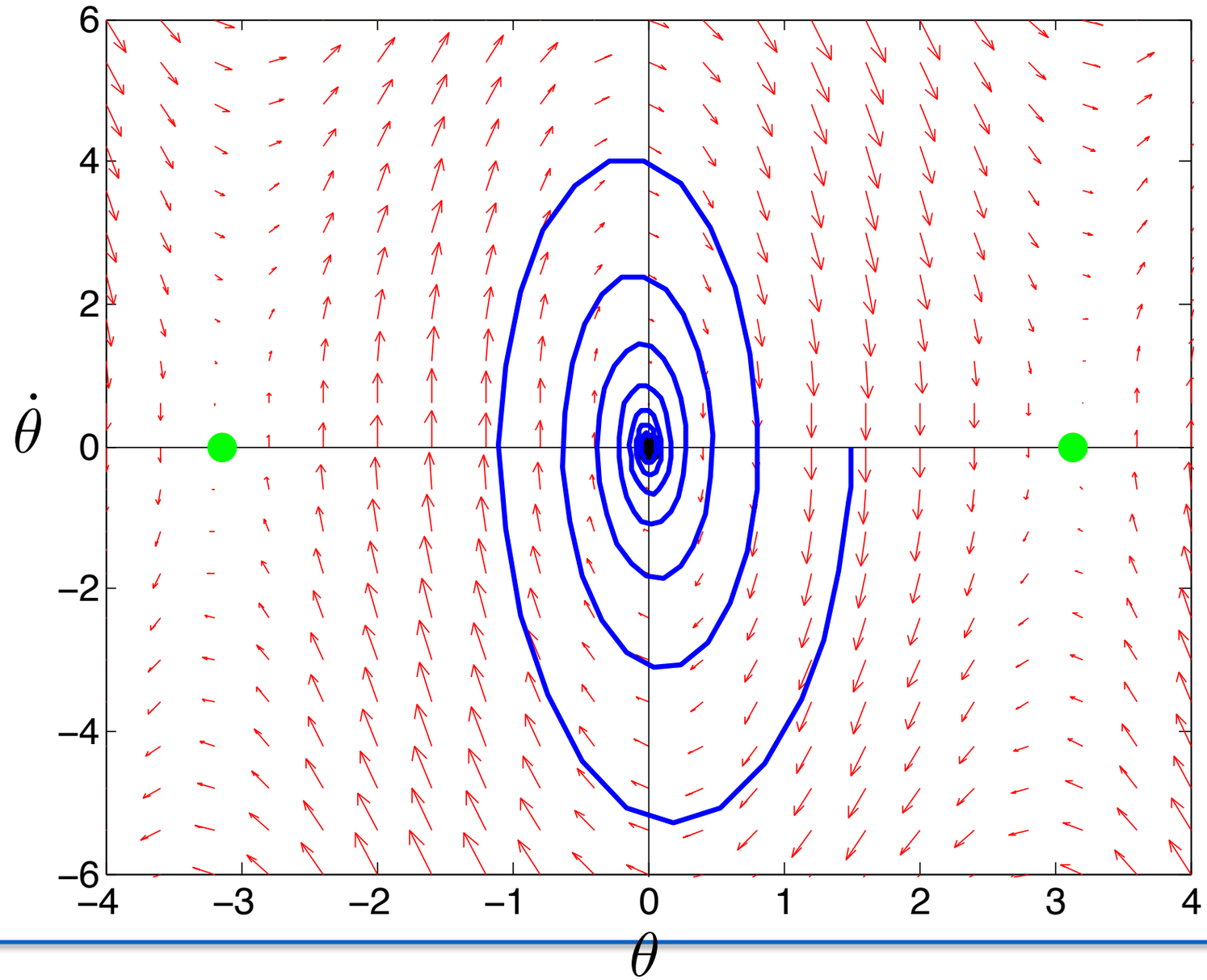
- Kinematic planning is simpler
- But many tasks require careful consideration of *dynamics*
- *Underactuated* systems highlight this point

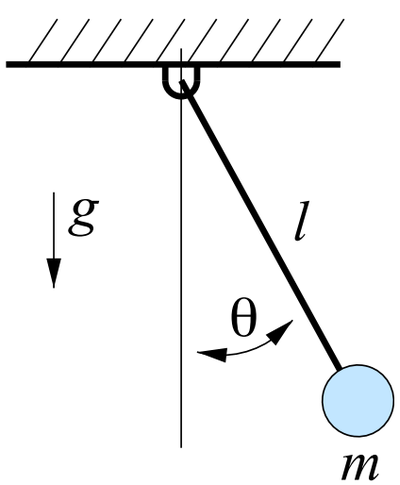






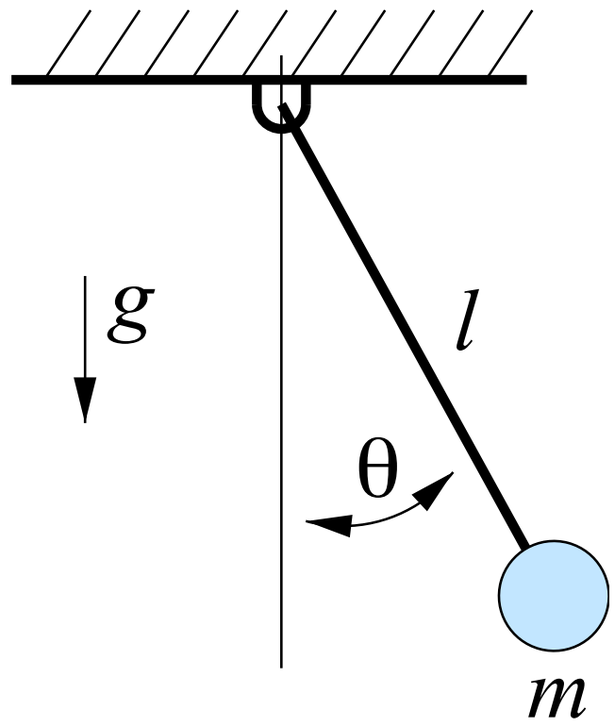
The Simplest Robot





$$x = \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix} \qquad u \equiv \text{base torque} \qquad \dot{x} = f(x, u)$$

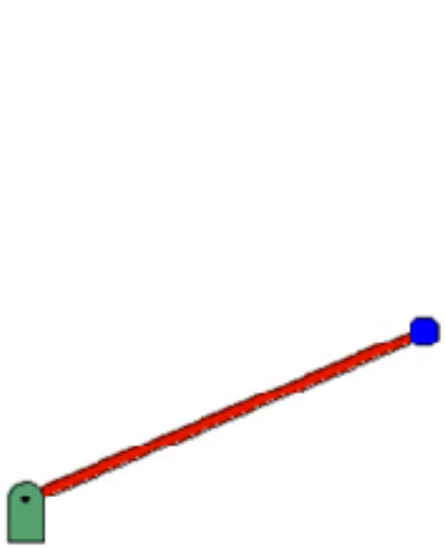
(nonlinear)

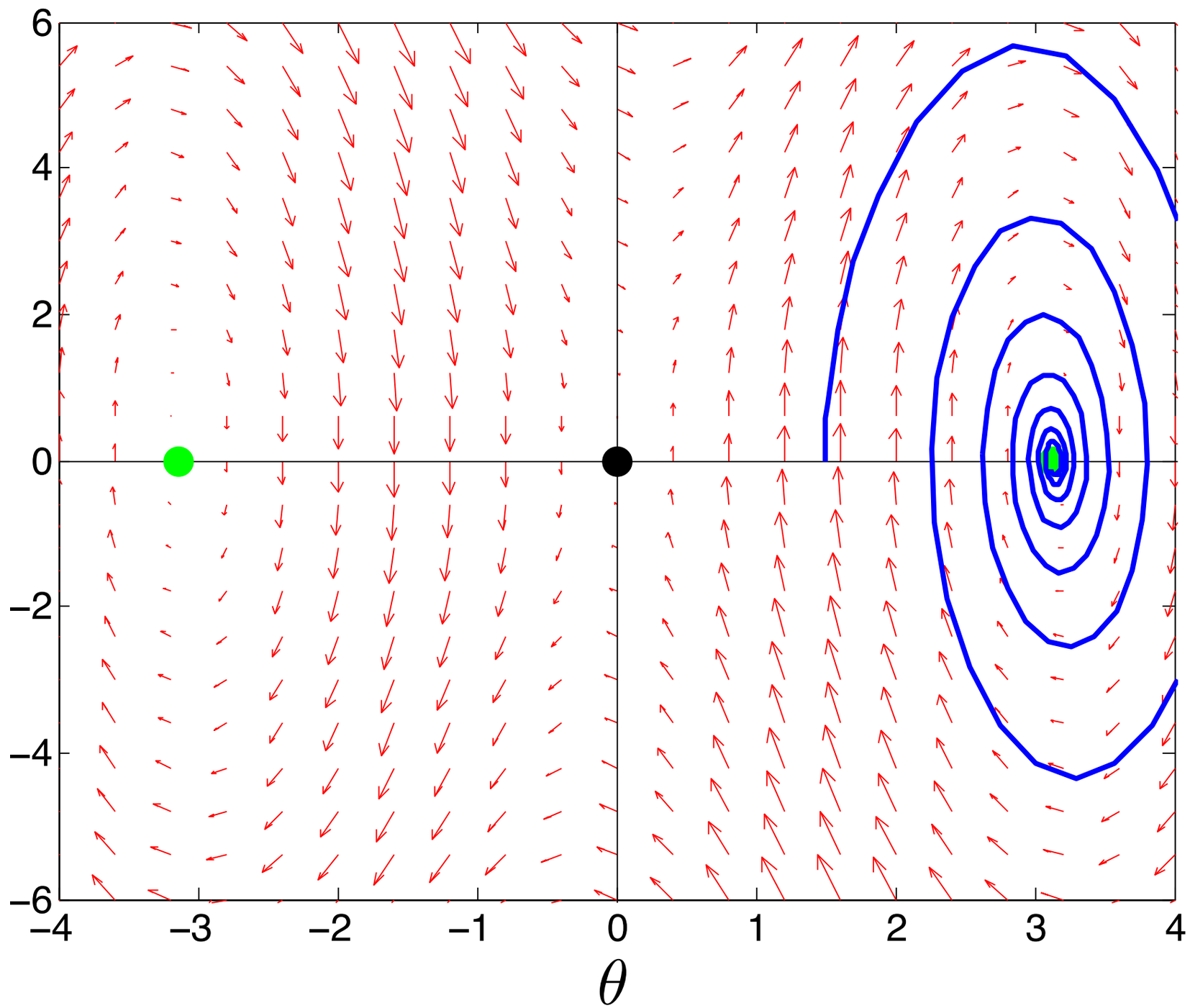


How can we choose $u=\pi(x)$ such that the pendulum stabilizes in the upright position?

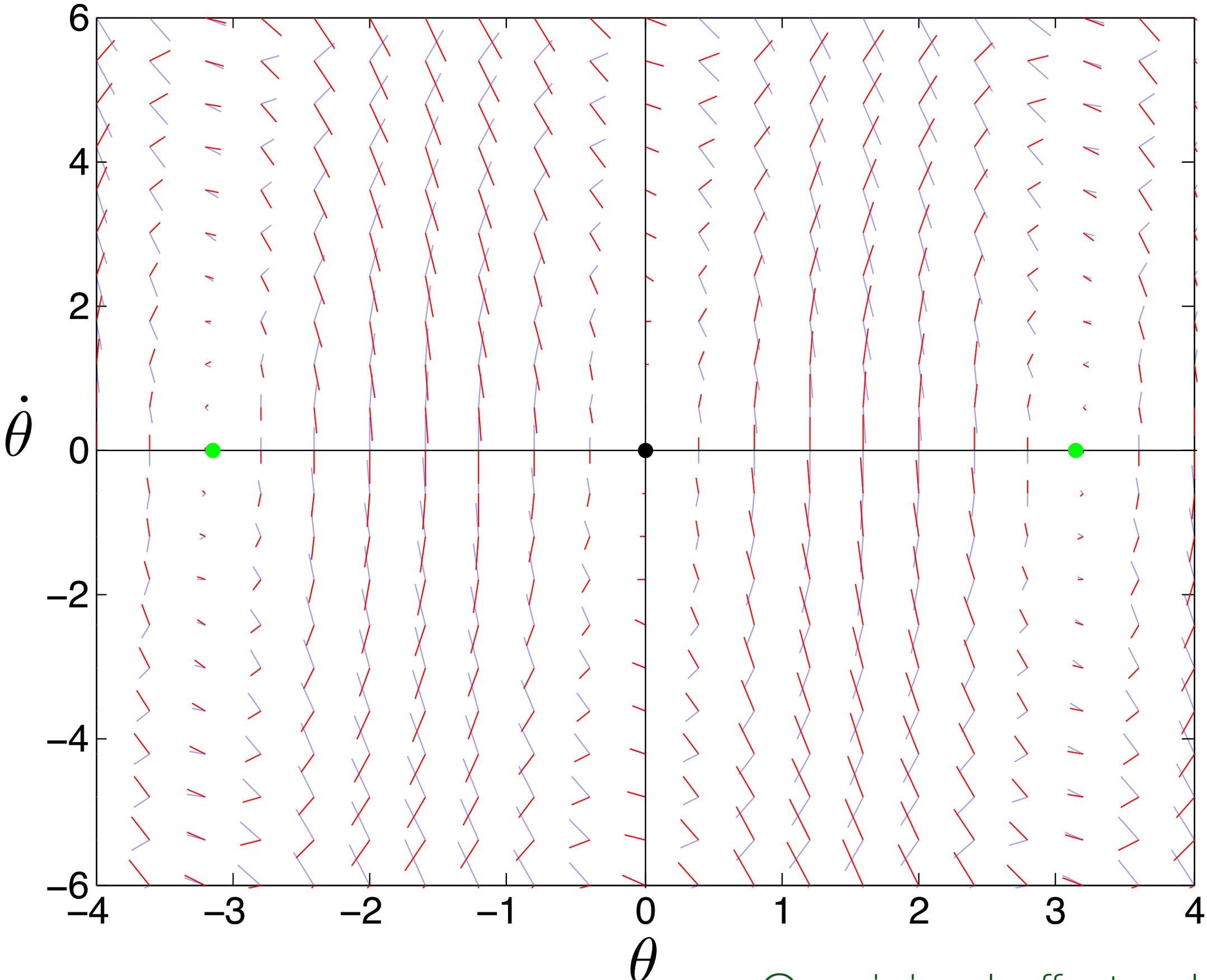
Invert gravity!

Invert Gravity





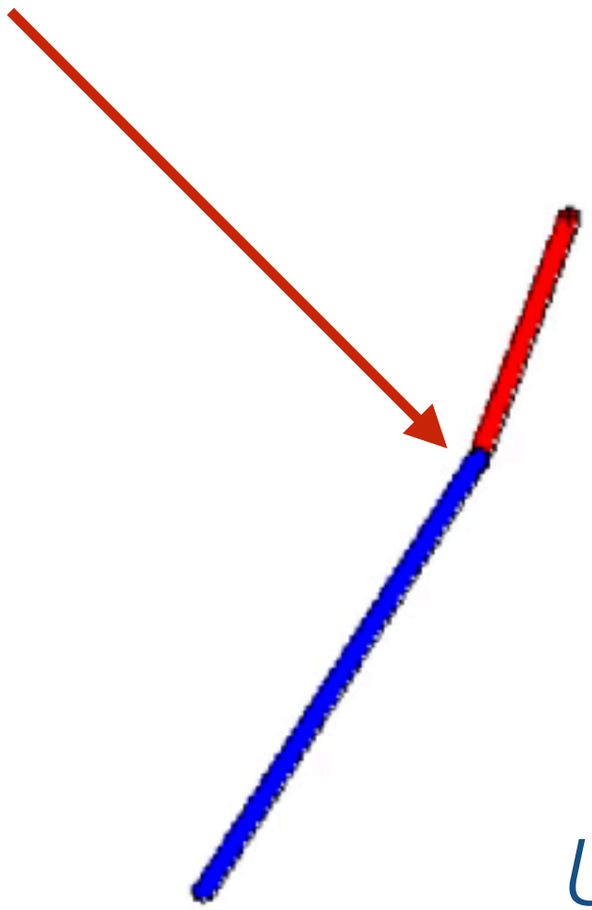
Control : reshaping the phase portrait



Q: minimal effort solution?

The Acrobot

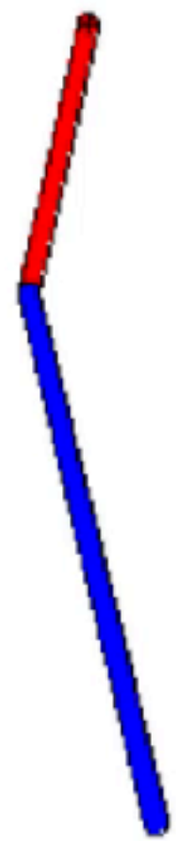
one actuator here



How do control this system to the vertical position?

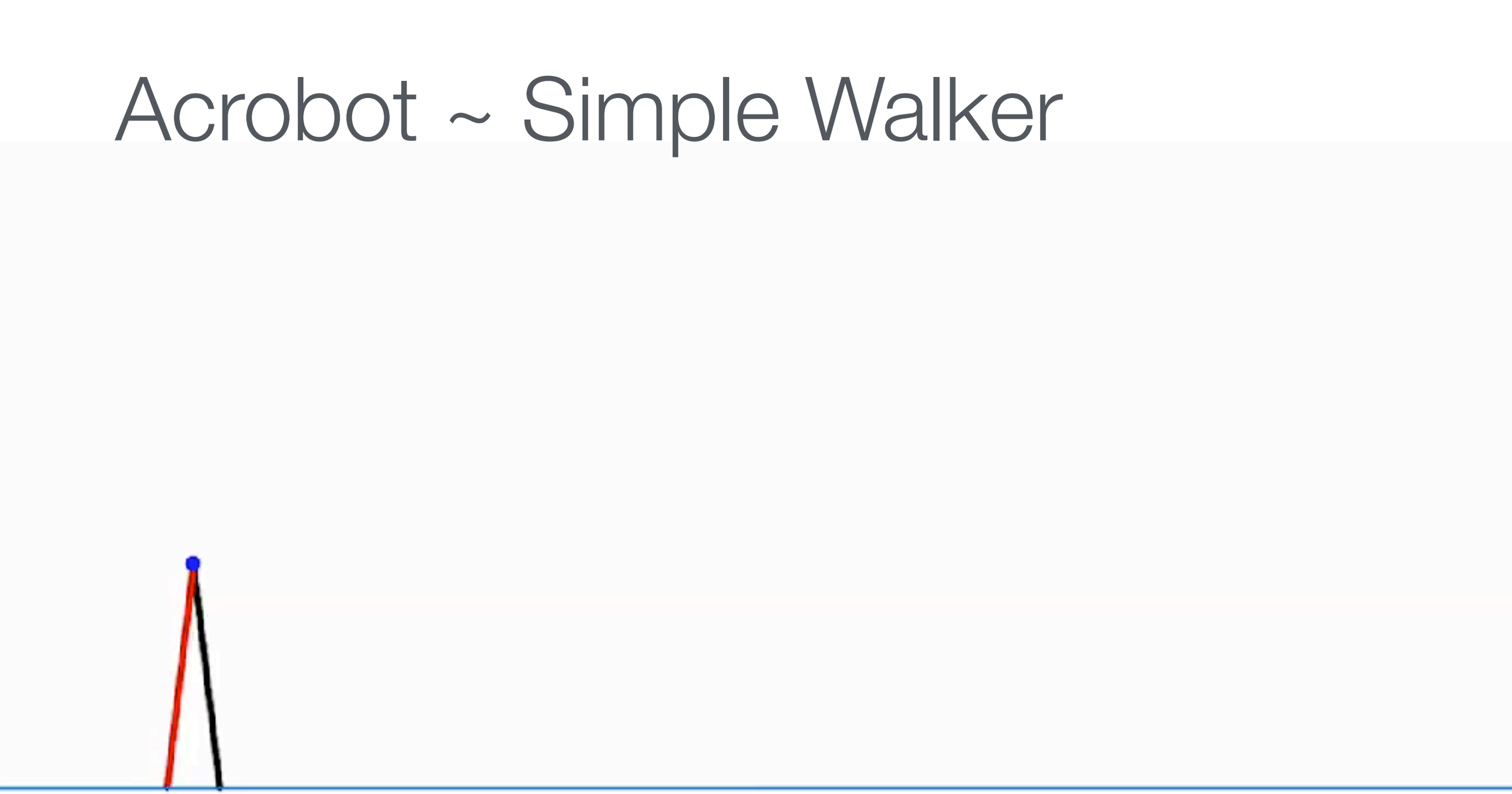
Underactuated… have exploit actuation *and* passive dynamics to achieve goals!

Acrobot Swing-up



$u=\pi(x)$ is not obvious, but we can compute it with optimization! (more later)

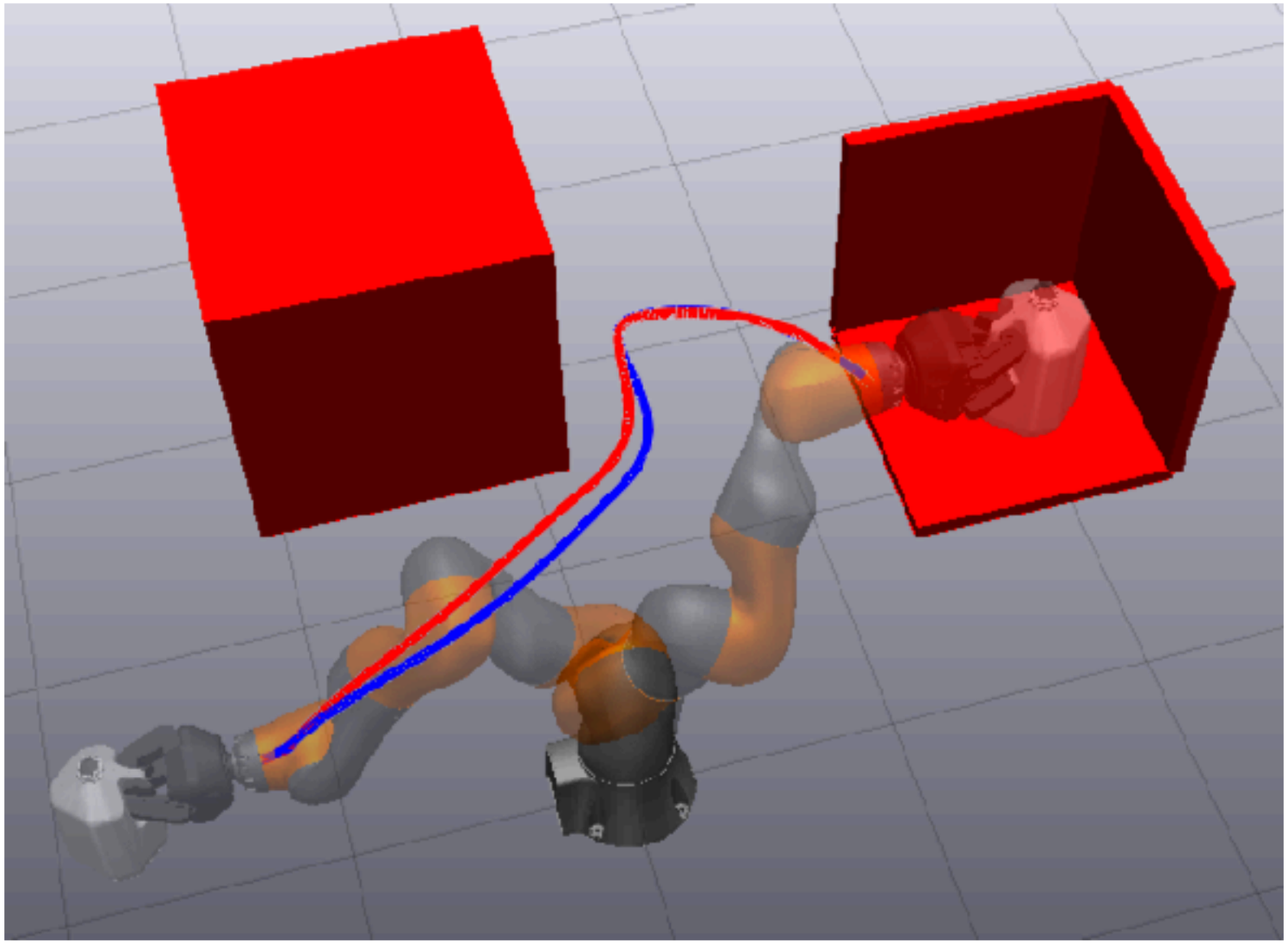
Acrobot ~ Simple Walker



But my robot has lots of motors…

Don’t worry!

The tools we will discuss today are useful for generating dynamic behaviors in all kinds of robots!



The optimization view of the world

- There are many ways to generate behaviors in robots:
    - Heuristic methods, virtual model control, programming by demonstration, etc.

- *Optimization* is a powerful and general framework for creating goal-directed behavior in robots

- Encode goals using *cost* and *constraint* functions

Optimal Control

- Find a policy, $u = \pi(x)$, that minimizes:

$$J = \ell_f(x_N) + \sum_{k=0}^{N-1} \ell(x_k, u_k)$$

subject to

- Dynamics constraints:

$$x_{k+1} = f(x_k, u_k)$$

(discrete time)

"minimize energy"
"move quickly"
"get to the goal"

A note about time discretization

$$J = \ell_f(x(t_f)) + \int_0^{t_f} \ell(x(t), u(t))dt$$

$$\dot{x} = f(x, u)$$

Continuous time

$$\Rightarrow$$

$$h = t_f/N \quad \text{(time step)}$$

$$J = \ell_f(x_N) + \sum_{k=0}^{N-1} \ell(x_k, u_k)h$$

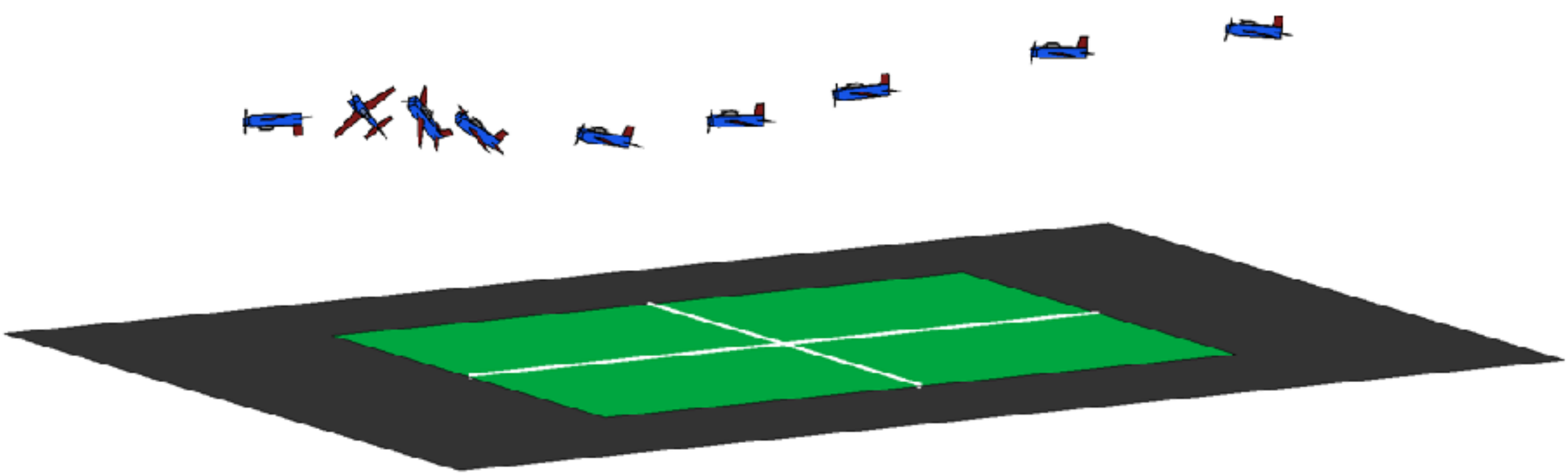
Discrete time

$$x_{k+1} = x_k + h \cdot f(x_k, u_k)$$

(you can use higher-order integrators too)

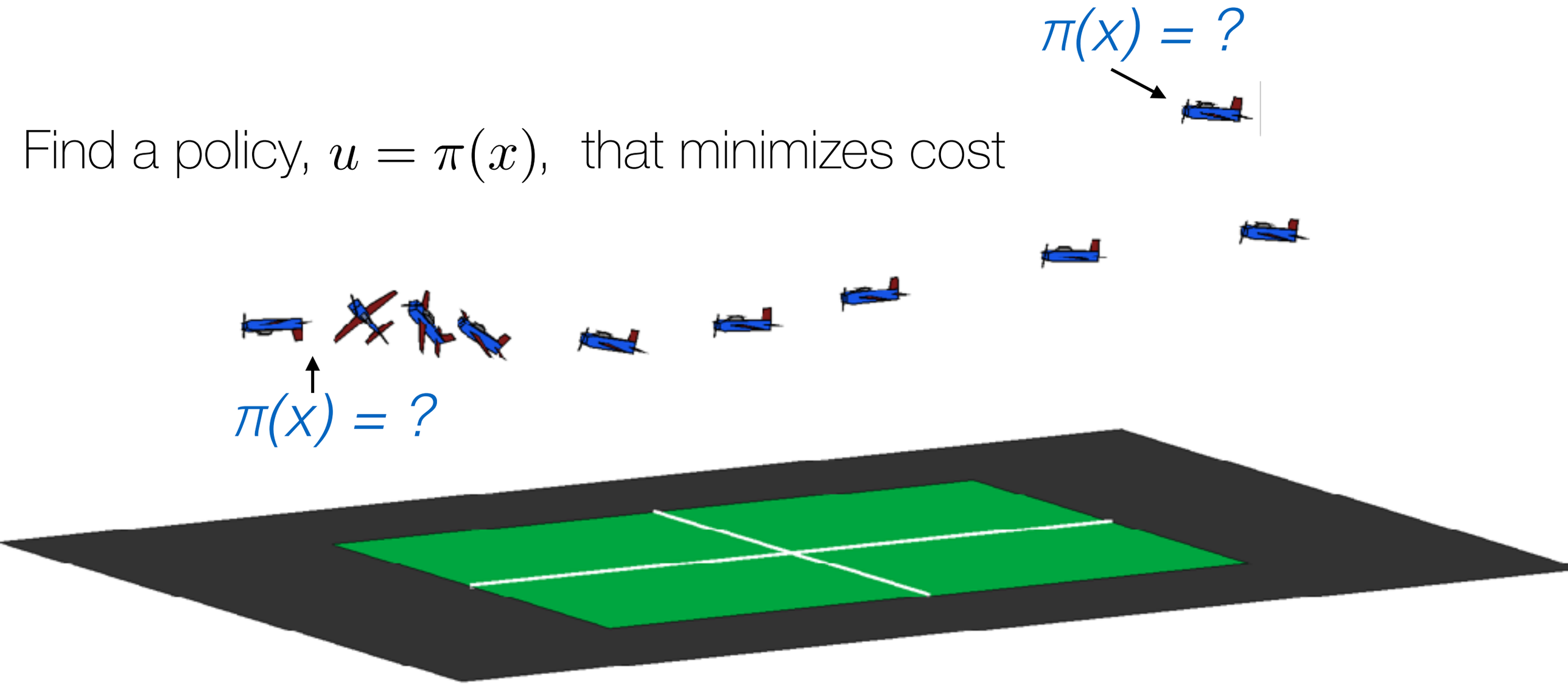
Example: Airplane Barrel Roll

- Goal: 180 degree barrel roll
- States: $x \in \mathbb{R}^{12}$
- Input: $u \in \mathbb{R}^4$
- Cost: $J = (x_N - x_{\text{ref}})^T Q (x_N - x_{\text{ref}}) + \sum_{k=0}^{N-1} u_k^T R u_k$



An Intuitive Solution

Find a policy, $u = \pi(x)$, that minimizes cost



Idea: work backwards in time!

An Algebraic View

- Cost: $J = \ell_f(x_N) + \sum_{k=0}^{N-1} \ell(x_k, u_k)$

- Define *V*(x,n)* as the lowest possible cost remaining starting in state *x* at timestep *n*

- At timestep *N*, there's nothing to do!

$$V^*(x, N) = \ell_f(x)$$

An Algebraic View

$$V^*(x, N-1) = \min_u \left[\ell(x,u) + \ell_f(x')\right] \qquad x' = f(x,u)$$

$$V^*(x, N-2) = \min_u \left[\ell(x,u) + \min_{u'} \left[\ell(x',u') + \ell_f(x'')\right]\right]$$

$$= \min_u \left[\ell(x,u) + V^*(x', N-1)\right]$$

An Algebraic View

$$V^*(x, N-1) = \min_u \left[\ell(x,u) + \ell_f(x')\right] \qquad x' = f(x,u)$$

$$V^*(x, N-2) = \min_u \left[\ell(x,u) + \min_{u'} \left[\ell(x',u') + \ell_f(x'')\right]\right]$$

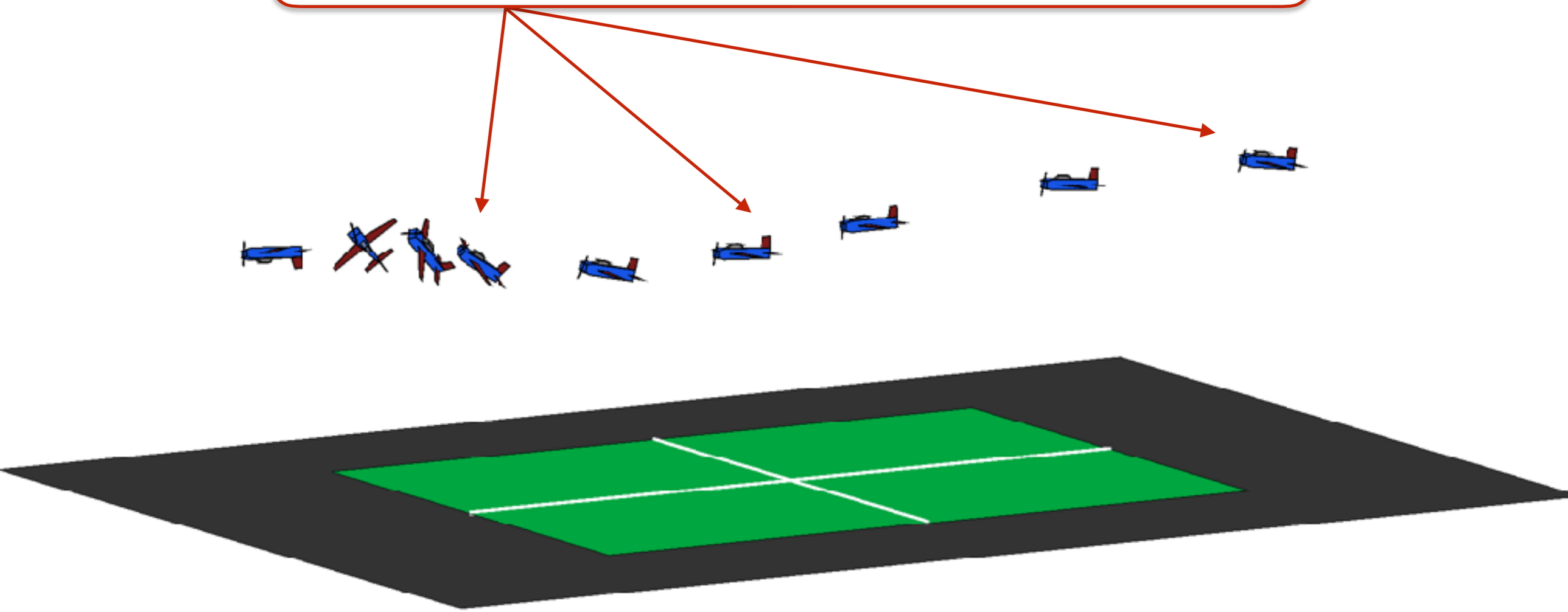
$$= \min_u \left[\ell(x,u) + V^*(x', N-1)\right]$$

- Key idea… Bellman's equation

$$V^*(x,n) = \min_u \left[\ell(x,u) + V^*(x', n+1)\right]$$

Optimal behavior with 1-step lookahead

$$\pi^*(x, n) = \arg\min_u \left[\ell(x, u) + V^*(x', n + 1)\right]$$



Quiz

$$V^*(x,n) = \min_u \left[\ell(x,u) + V^*(x',n+1)\right]$$

- Q1: Suppose my cost is quadratic, is also *V*(x,n)* quadratic in general?

A1: No (thanks to nonlinear dynamics)

- Q2: Does Bellman's equation hold for multiplicative costs?

$$J = \prod_{k=0}^{N} \ell(x_k, u_k)$$

A2: No… try it on our derivation!

Clearly *V*(x,n)* is useful, so how can we compute it?

Dynamic Programming

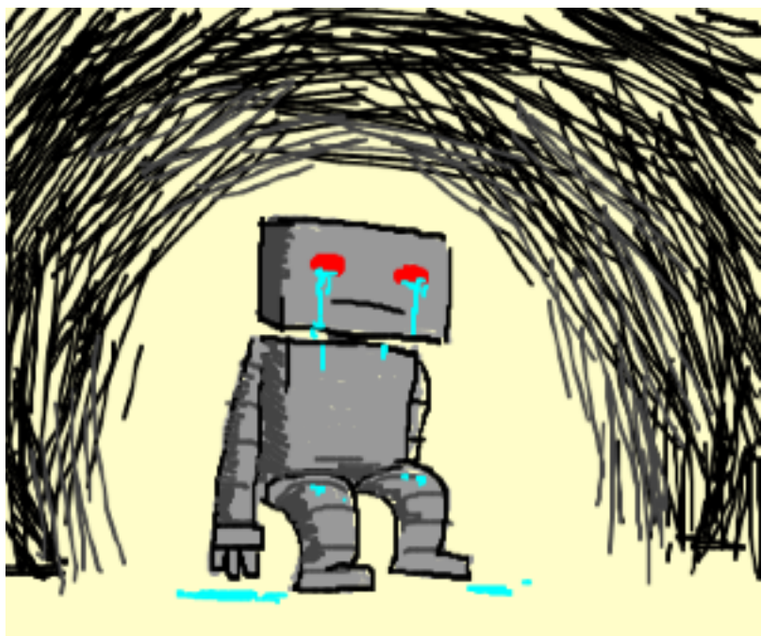
- Turn Bellman equation into an update rule

$$V^*(x,n) \leftarrow \min_u \left[\ell(x,u) + V^*(x_{n+1}, n+1)\right]$$

- Simple algorithm:
  - Discretize state space
    - Loop over all states
      - Apply Bellman update
    - Until *V** converged

➡ Polynomial in the number of states

➡ …but the number of states grows exponentially in the dimension of the robot



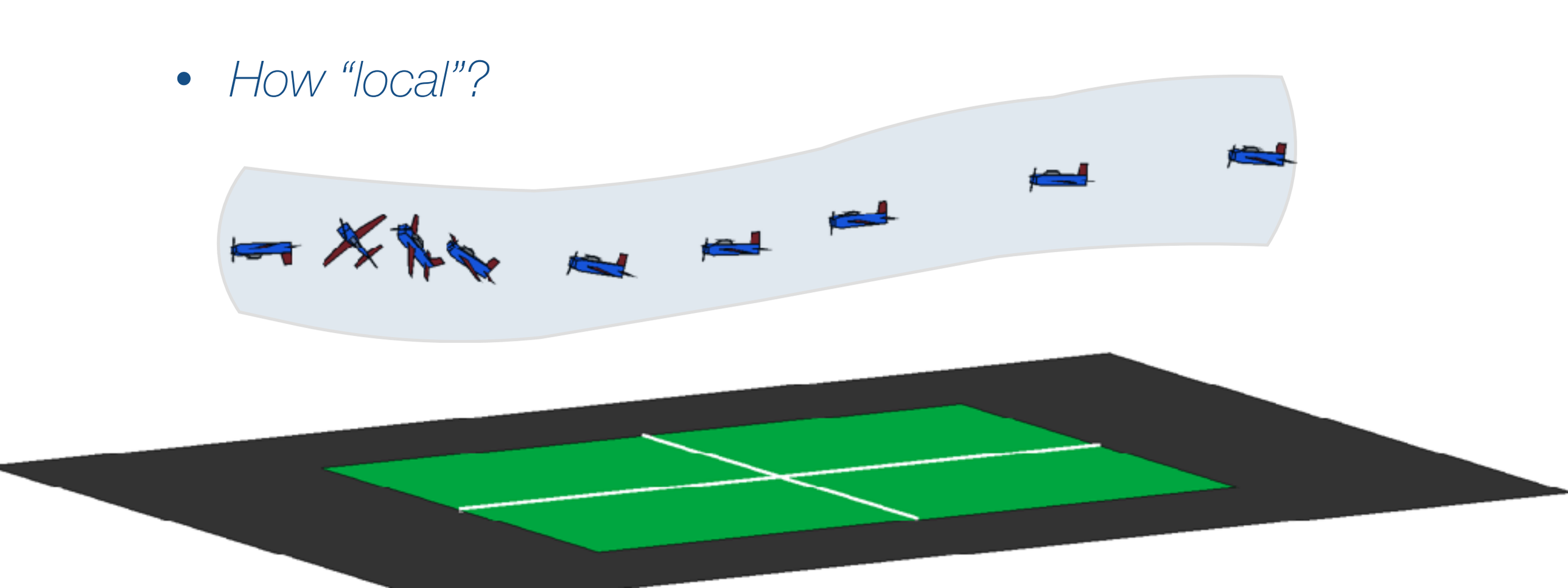
Curse of Dimensionality

- As state dimension increases, discretization results in an exponential blowup

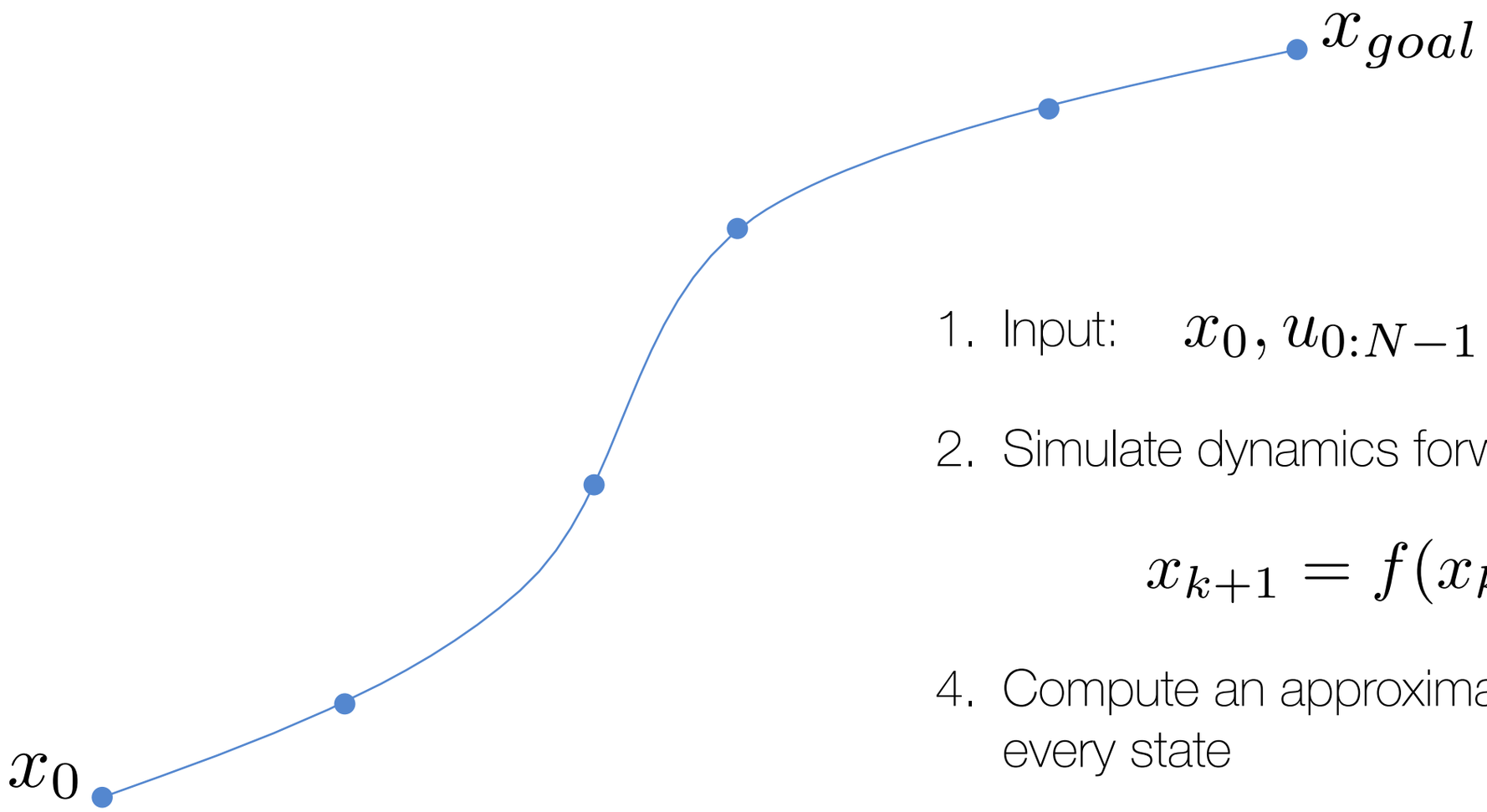
- What if we instead settle for local policies around a trajectory?

  - *Which trajectory?*

  - *How "local"?*



Differential Dynamic Programming



1. Input: $x_0, u_{0:N-1}$
2. Simulate dynamics forward using

$$x_{k+1} = f(x_k, u_k)$$

4. Compute an approximation to *V(x,k)* at every state
5. Compute control modifications using Bellman's equation
6. Go to #2 until convergence

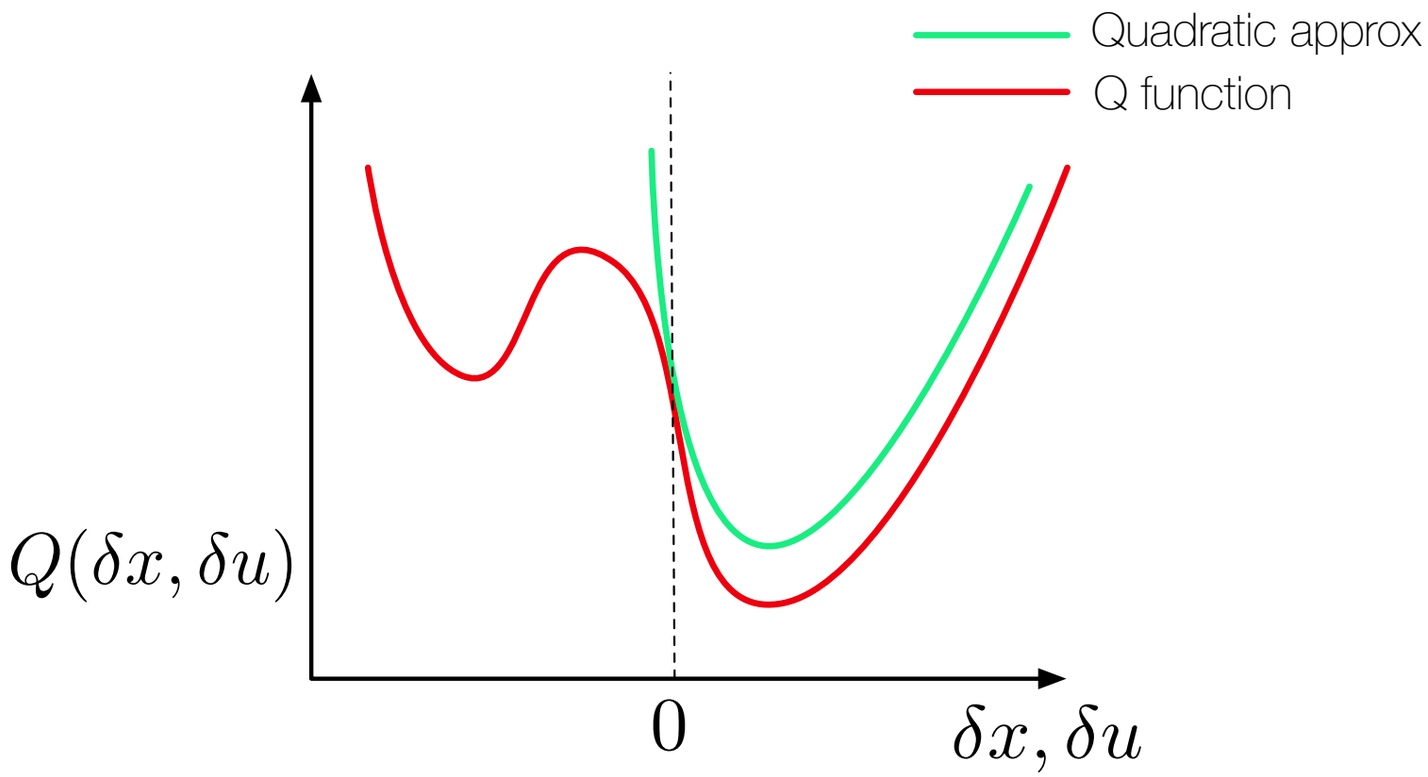
Differential Dynamic Programming

- Given initial trajectory as input, $x_{0:N}, u_{0:N-1}$

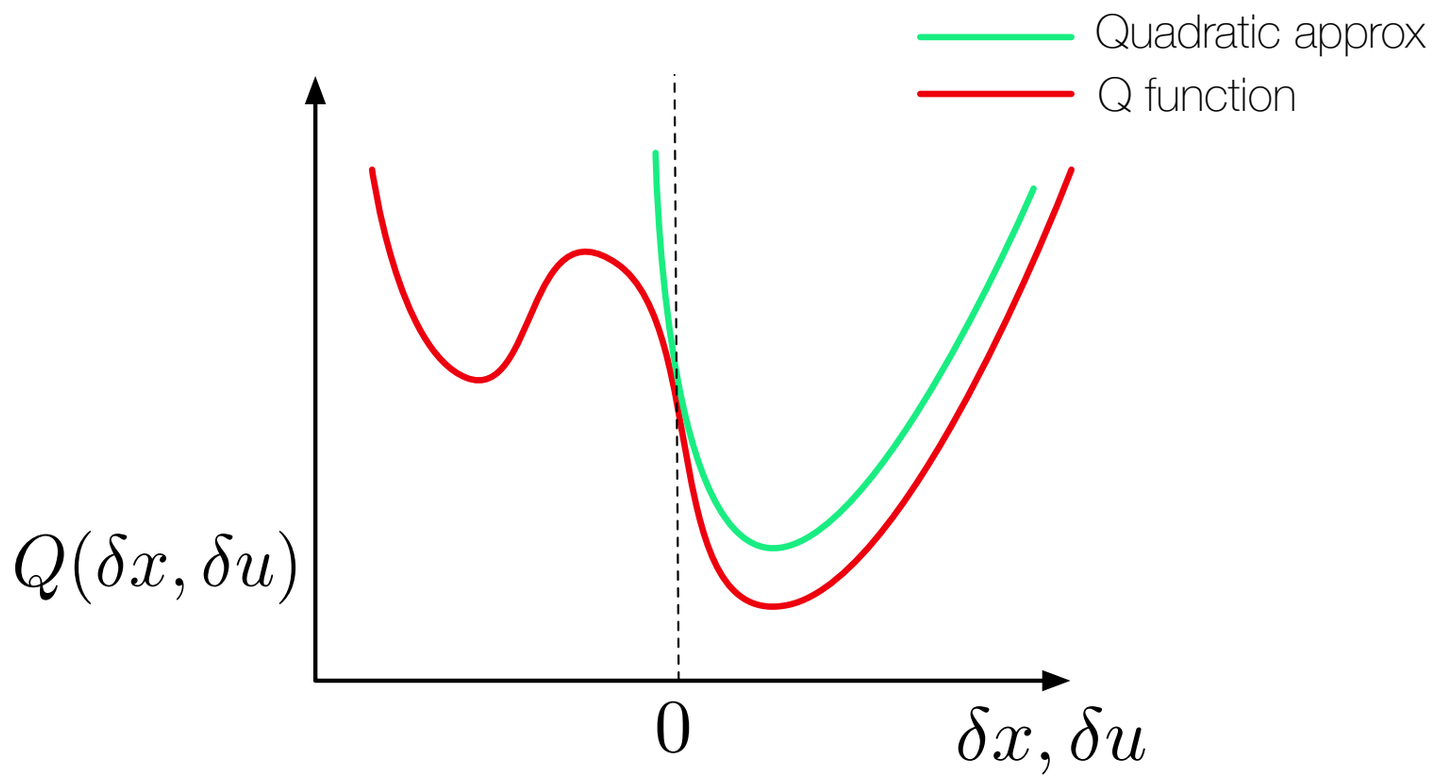
- Define:

$$Q(\delta x, \delta u) = \ell(x + \delta x, u + \delta u) + V(f(x + \delta x, u + \delta u))$$

- Approximate to second order:



Differential Dynamic Programming



$$\frac{\partial^2 Q}{\partial x \partial u}$$

etc.

$$Q(\delta x, \delta u) \approx \frac{1}{2} \begin{bmatrix} 1 \\ \delta x \\ \delta u \end{bmatrix}^T \begin{bmatrix} 0 & Q_x^T & Q_u^T \\ Q_x & Q_{xx} & Q_{xu} \\ Q_u & Q_{xu}^T & Q_{uu} \end{bmatrix} \begin{bmatrix} 1 \\ \delta x \\ \delta u \end{bmatrix},$$

[see, e.g., Tassa thesis 2011]

Backwards Pass

- Want to minimize *Q* backwards in time
- For *k=N:0*
  - Set derivative of *Q* equal to zero, solve for δu

$$\delta u = -Q_{uu}^{-1}\left(Q_{ux}\delta x + Q_u\right) \equiv K\delta x + a$$

- Substitute δu into *Q* approximation at time *k* and compute *Q* approximation at *k-1*

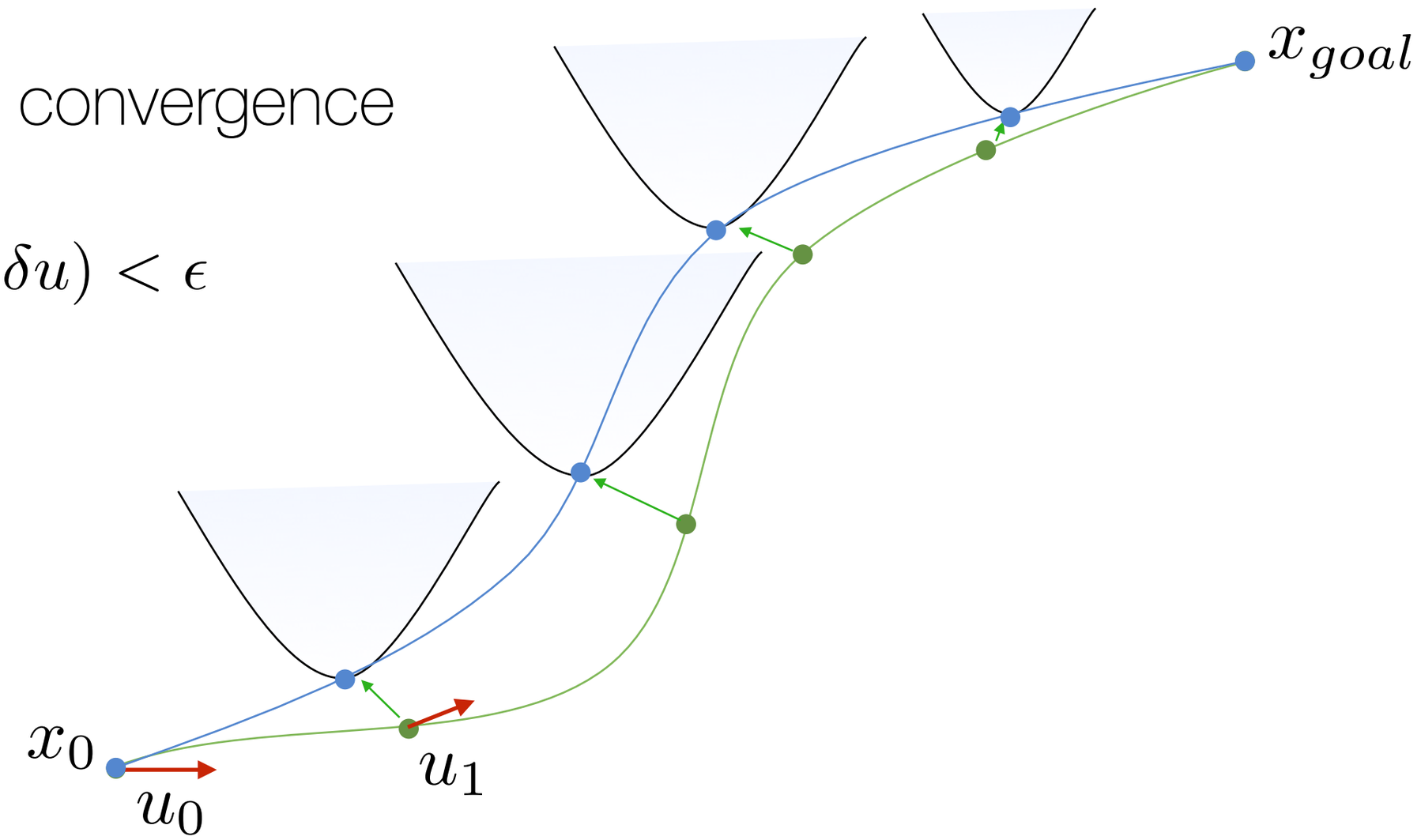
Forwards Pass

- Forward simulate system from x_0 using feedback controller computed in backwards pass

$$x_{k+1} = f(x_k, u_k + K_k \delta x_k + a_k)$$

- Repeat until convergence

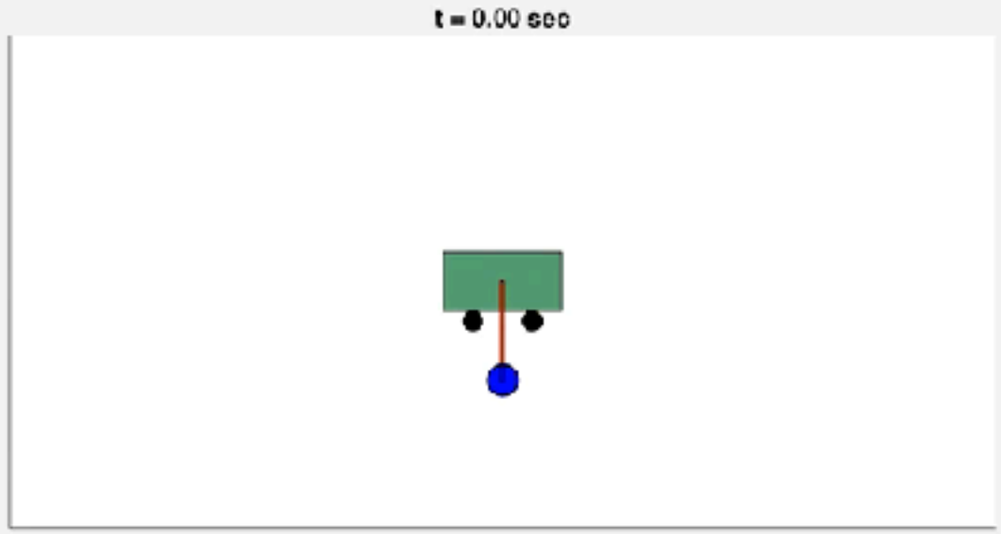
  e.g., $\max(\delta u) < \epsilon$

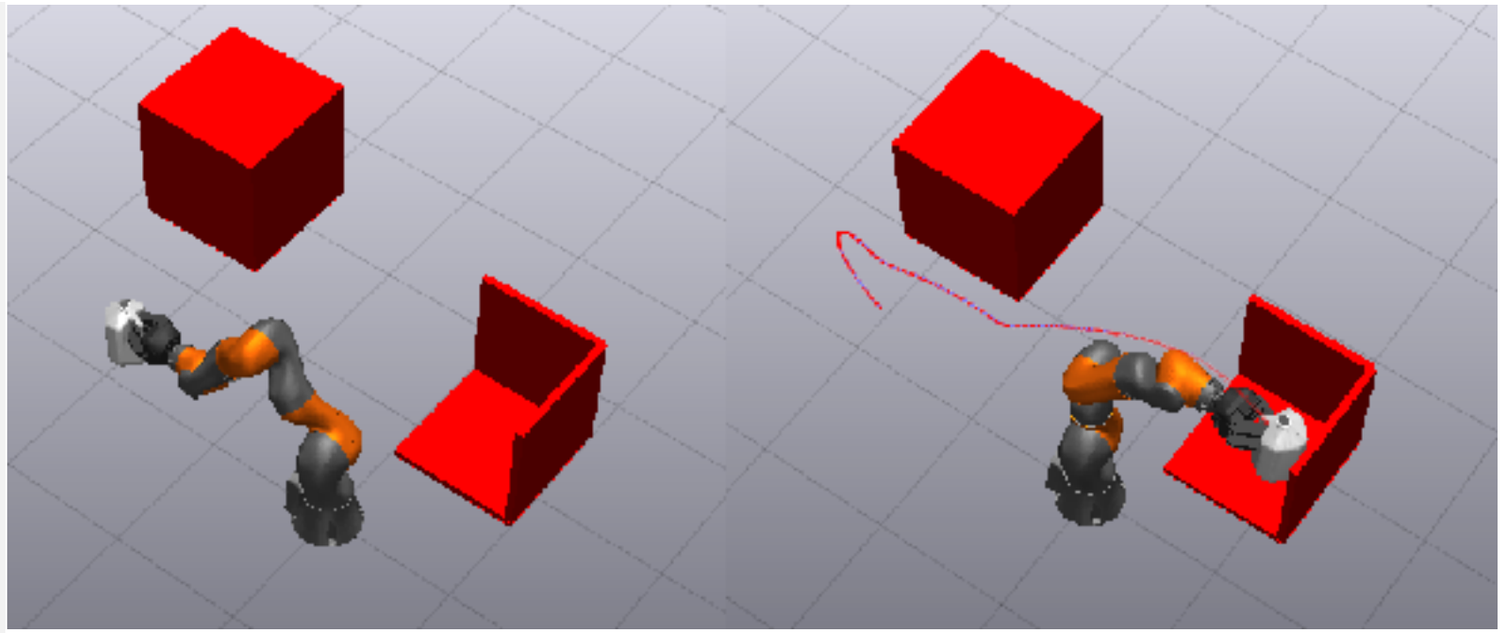


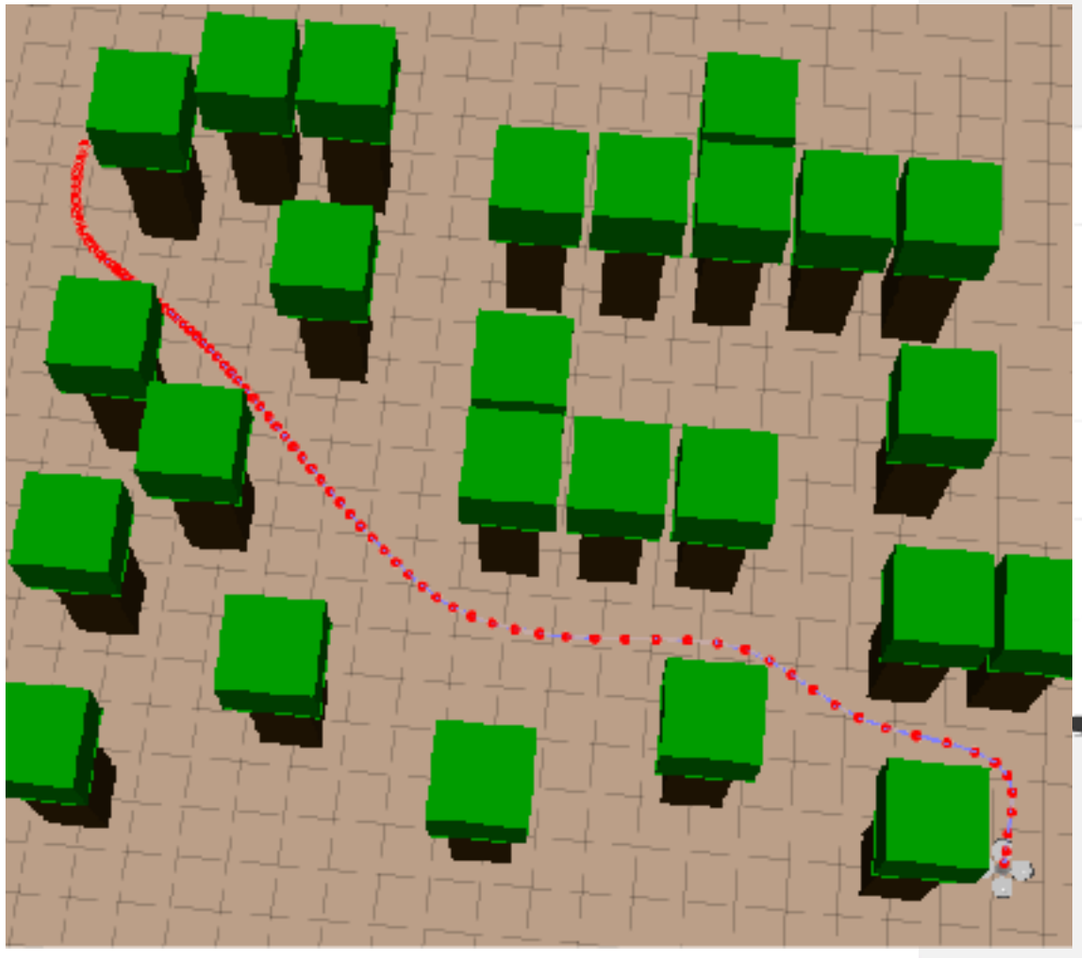
Some DDP Variants

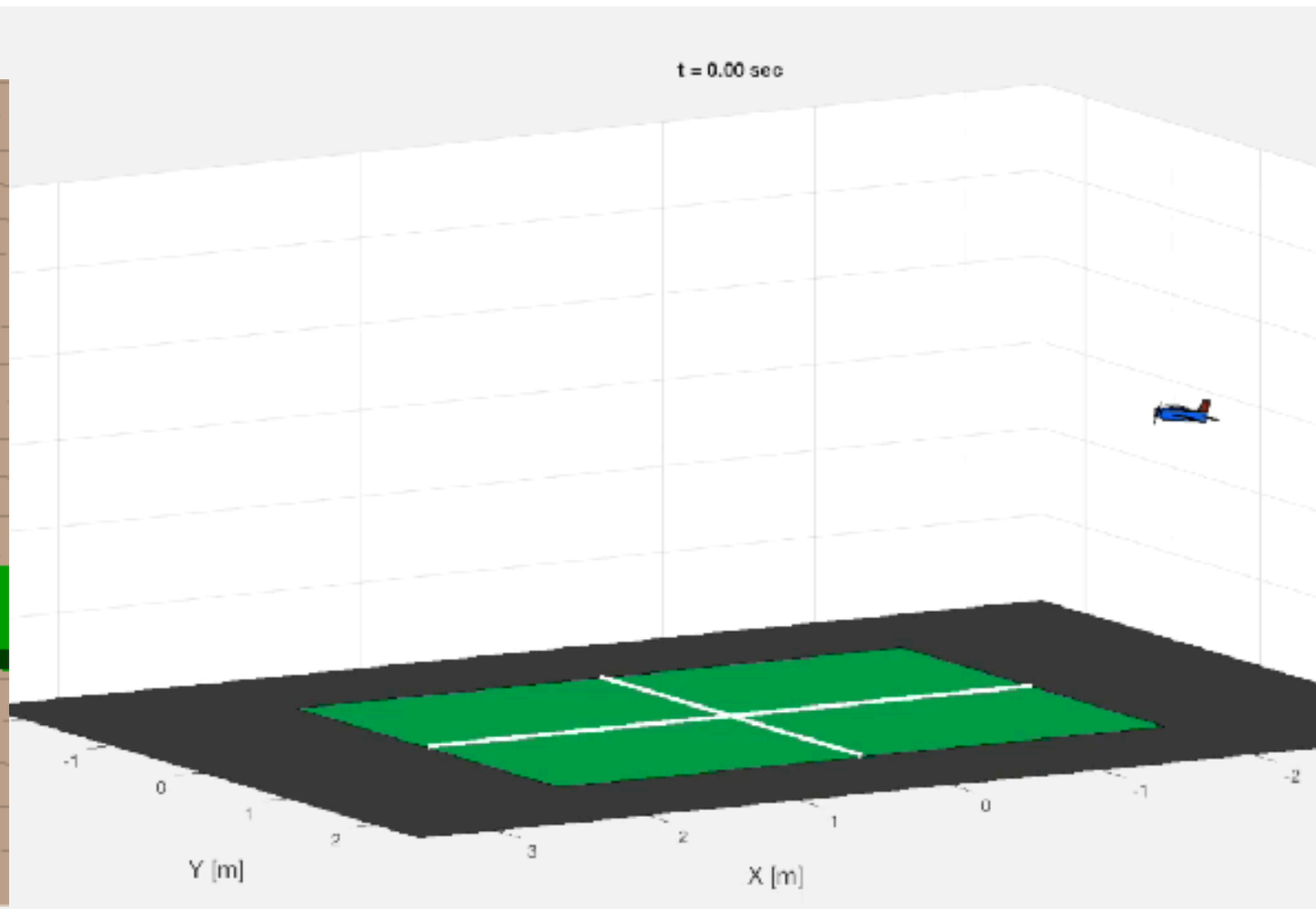
- Most commonly used variant is Iterative LQR (iLQR)
  - Leave out the 2nd derivative of dynamics in Q [Li et al., 2004]
  - Slightly slower convergence (superlinear vs quadratic), otherwise exactly the same algorithm
- Input-constrained DDP to handle torque limits [Tassa et al., ICRA'14]
- Unscented dynamic programming
  - Derivative-free, Q update based on the Unscented Transform

    [Manchester & Kuindersma, CDC'16]


t = 0.00 sec






t = 0.00 sec
-1
0
1
2
3
Y [m]
2
1
0
-1
-2
X [m]

DDP for Model-Predictive Control

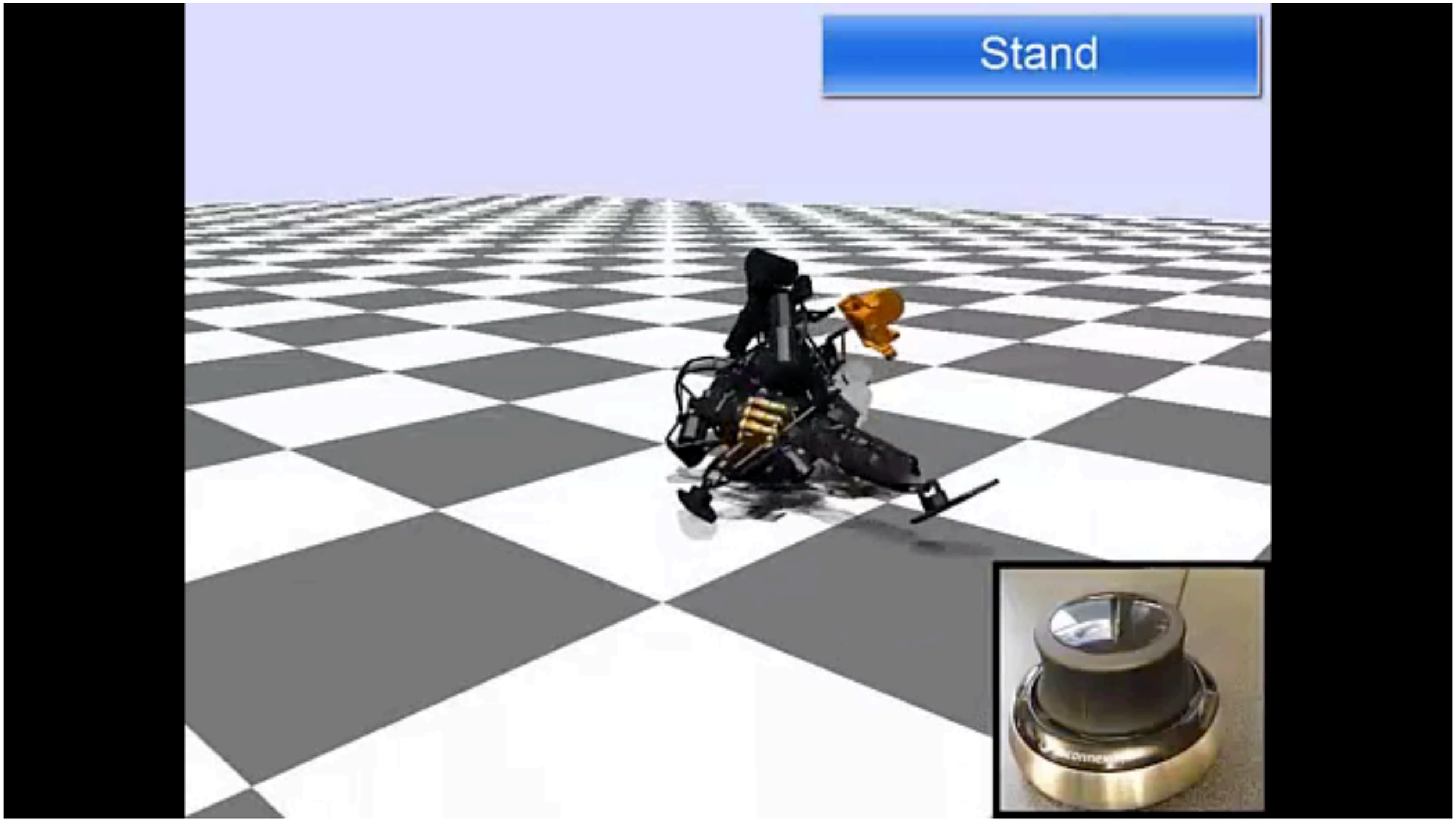
An intergated system for real-time model-predictive control of humanoid robots

Tom Erez, Kendall Lowrey, Yuval Tassa, Vikash Kumar,
Svetoslav Kolev and Emo Todorov

Movement Control Laboratory
University of Washington

Humanoids 2013

DDP for Model-Predictive Control



So far…

- Computing globally-optimal policies is infeasible for most robots
  - Instead settle for locally-optimal trajectories
- DDP is an efficient algorithm for computing trajectories and local feedback controllers, but…

➡ State constraints require care (typically approximated using costs)

➡ Attention to implementation details needed to ensure good numerical performance

➡ Assumes *f(x,u)* is a smooth function of *x,u*

Trajectory Optimization as an NLP

- We arrived at DDP using Bellman's equation + local approximation

- We can instead *transcribe* the optimal control problem as a standard nonlinear program:

$$\begin{aligned}
\underset{x_{0:N}, u_{0:N-1}}{\text{minimize}} \quad & \ell_f(x_N) + \sum_{k=0}^{N-1} \ell(x_k, u_k) \\
\text{subject to} \quad & x_{k+1} = f(x_k, u_k) \\
& g(x_k, u_k) \leq 0 \\
& h(x_k, u_k) = 0
\end{aligned}$$

explicit dynamics constraints

"don't walk through walls"

"keep your left foot on the ground"

Direct Transcription

- Why?

  ✓ Easy to add nonlinear state and input constraints

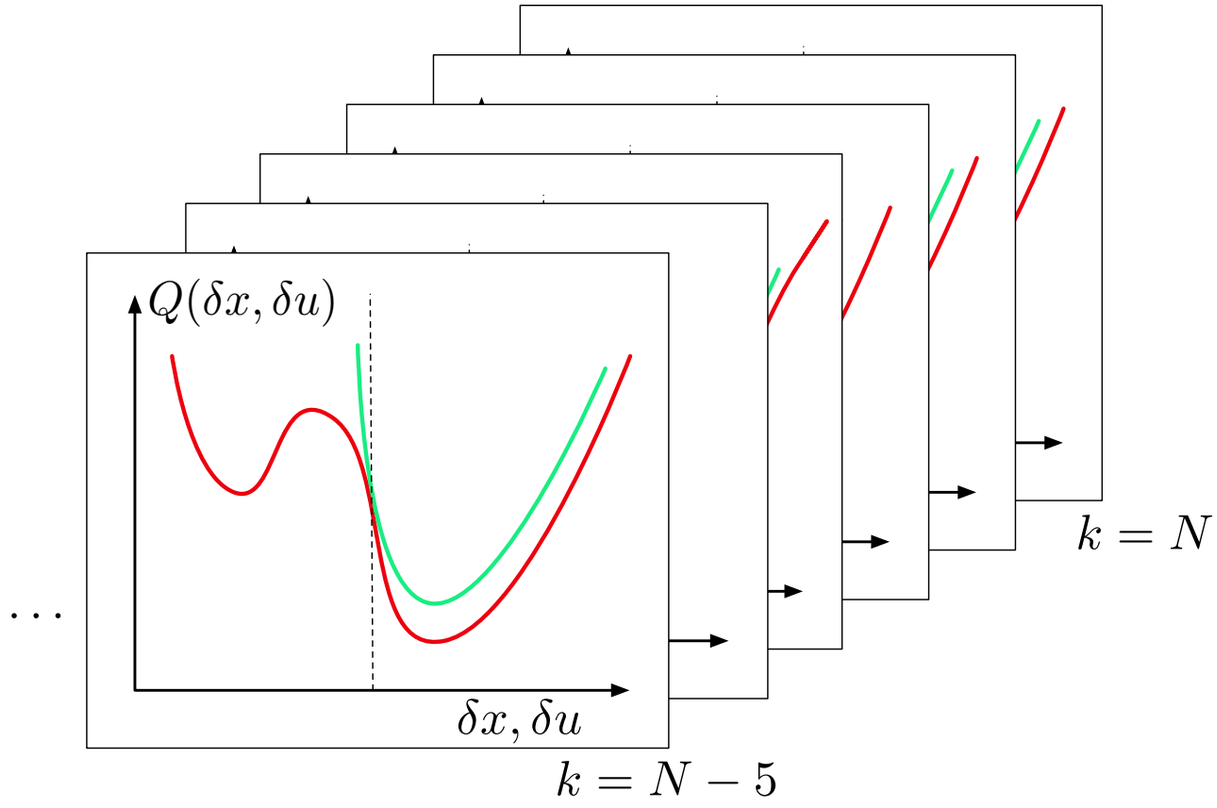
  ✓ Exploit fast and reliable off-the-shelf solvers (e.g., SNOPT [1])

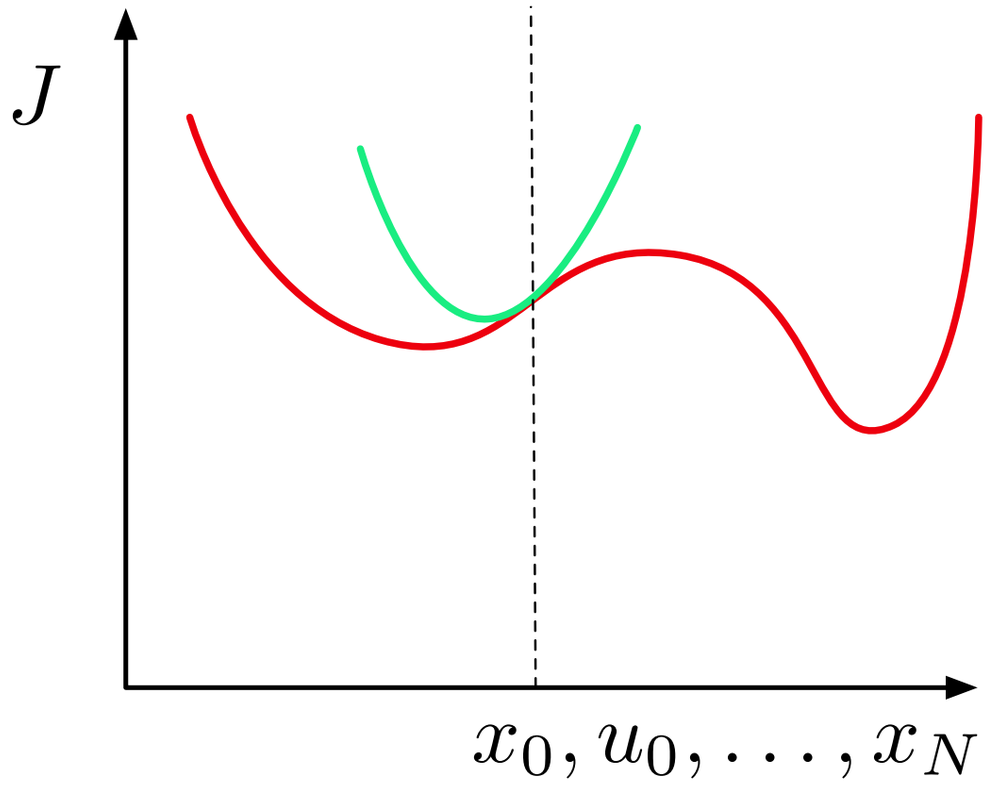
    - Spend your time designing costs/constraints, not debugging solvers

[1] http://www.sbsi-sol-optimize.com/asp/sol_product_snopt.htm

Intuition: Newton's Method

- A similar picture as before (ignoring constraints)
- Now updates are performed over entire trajectory, rather than sequentially backwards in time





Sequential Quadratic Programming

- Many powerful off-the-shelf solvers implement SQP
- High-level idea:
  - At each iteration linearize constraints at current trajectory, construct local quadratic cost
  - Solve local quadratic program (QP)—this is easy
  - Iterate until trajectory converges or max iters

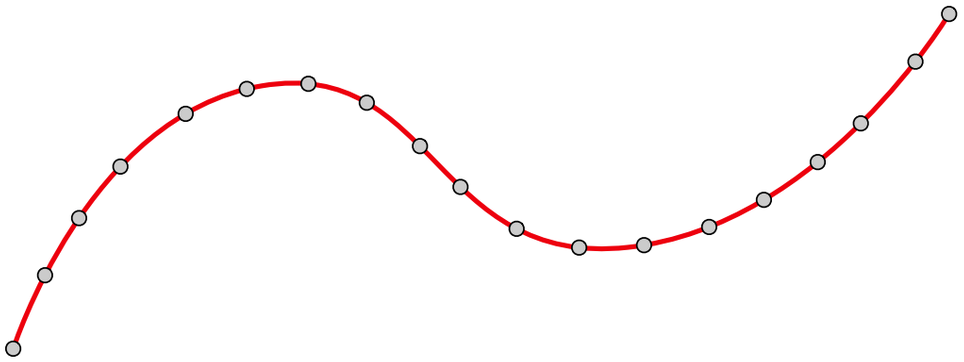
➡ Lots of important practical details you can safely ignore if using an established solver!

[see Nocedal & Wright, 2006]

DDP vs. SQP

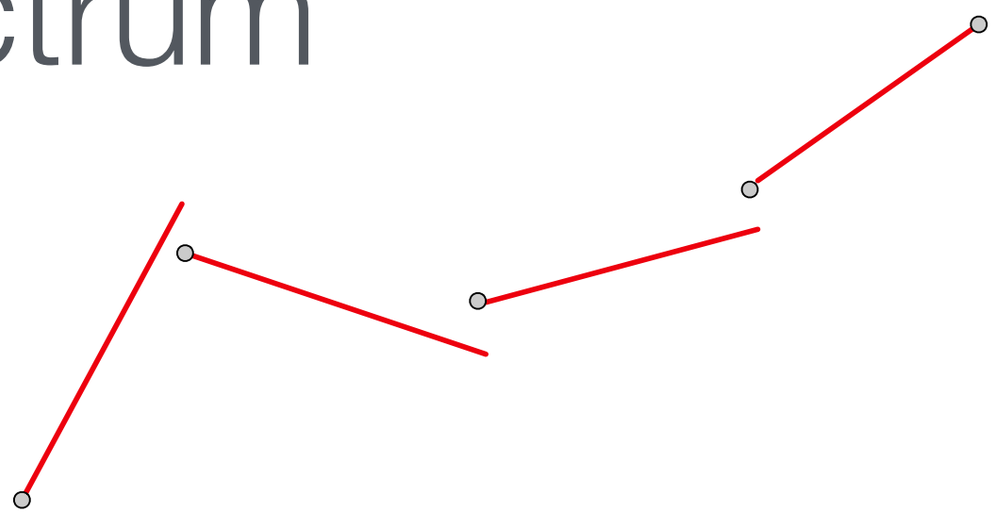
- Differential Dynamic Programming
  - Simulates dynamics to compute $x_{1:N}$
    - "Shooting method"
  - Each iteration solves a sequence of small, *1-step optimizations*
- SQP:
  - Satisfies dynamics as explicit constraints (often with smaller *N*)
    - "Direct transcription"
  - Each iteration solves a *single QP over the entire trajectory*

Two ends of a spectrum



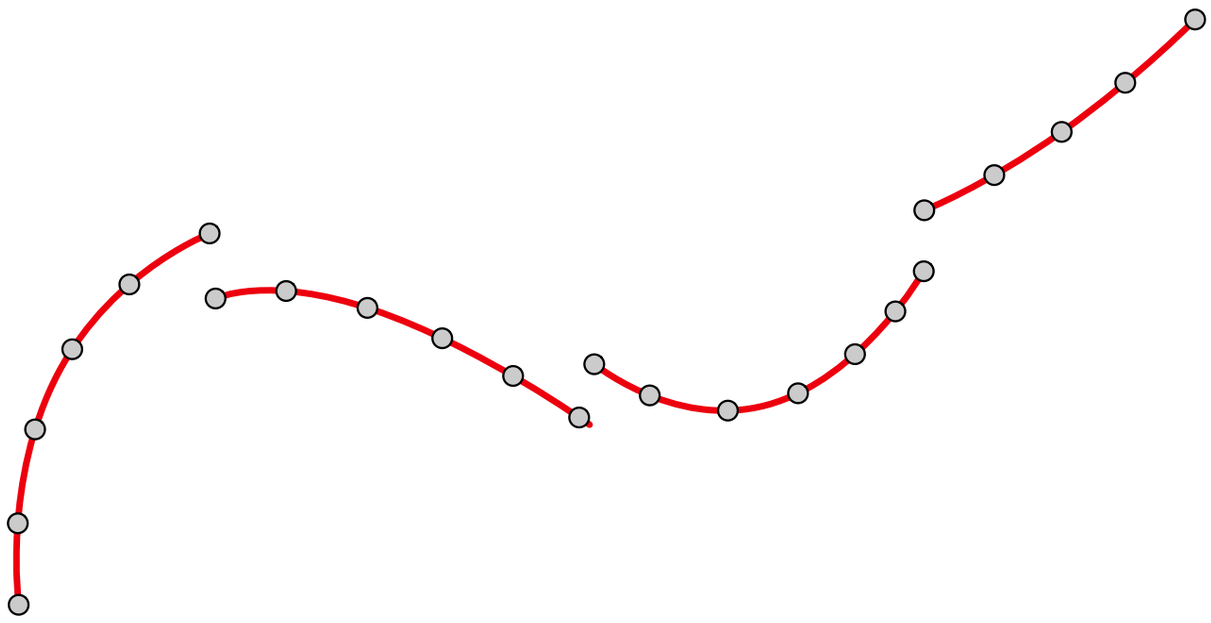
DDP (single shooting)

Simulate entire trajectory,
1-step updates



Direct transcription

Dynamics are constraints for all
timesteps, *N*-step updates

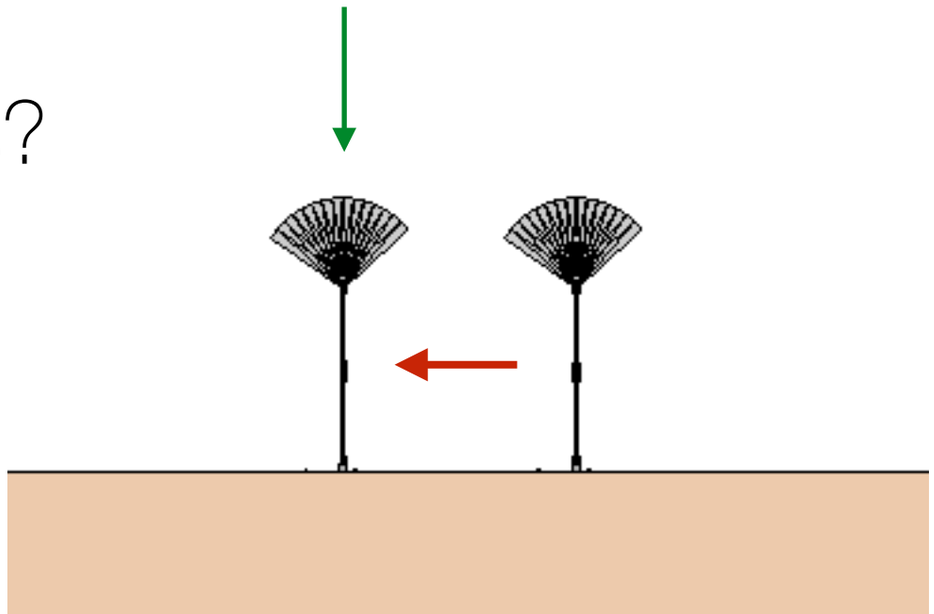


Multiple shooting

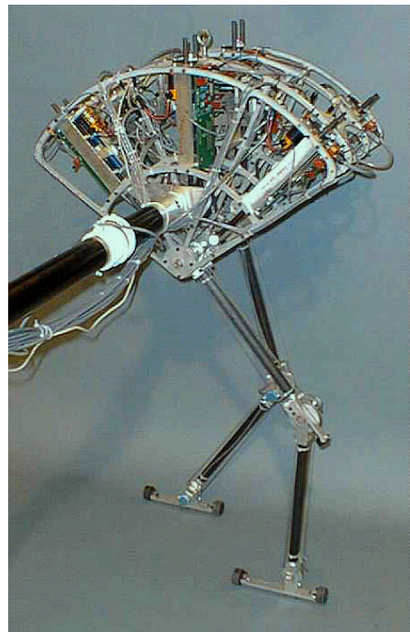
Simulate sub-trajectories,
enforce defect constraints

Example: Spring Flamingo

- States: $x = \begin{bmatrix} q \\ \dot{q} \end{bmatrix} \in \mathbb{R}^{18}$ Inputs: $u \in \mathbb{R}^4$
- Passive spring ankle
- Cost: $J = \sum_k (x_k - x_{\text{nom}})^T Q (x_k - x_{\text{nom}}) + u_k^T R u_k$
- Final state constraint: $x_N = x_{\text{goal}}$
- Dynamics constraints?



$\approx$



Handling Contact Dynamics

- Idea: add contact forces, $\lambda_{0:N-1}$, as decision variables, then constrain them to be physical

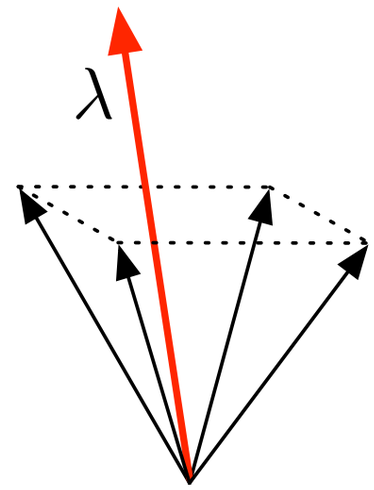
- *Approach 1*: pre-specify contact sequence ahead of time, assign force variables to states in contact

    - Works fine for two feet, but exponentially many possibilities as # of contact points grows

- *Approach 2*: Let the solver "discover" contact sequences for you!

Contact-Implicit Constraints

- Add contact forces, $\lambda_{0:N-1}$, as decision variables at each time step, then constrain them to be physical



  - Forces must obey friction limits
  - Robot cannot penetrate the ground: $\phi(q) \geq 0$
  - Forces cannot be applied unless on the ground

$$\phi(q) \cdot \lambda = 0$$

  - Tangental component maximally dissipates energy

Contact-Implicit Direct Transcription

$$\underset{x_{0:N},u_{0:N-1},\lambda_{0:N-1}}{\text{minimize}} \quad \ell_f(x_N) + \sum_{k=0}^{N-1} \ell(x_k, u_k)$$

dynamics includes forces

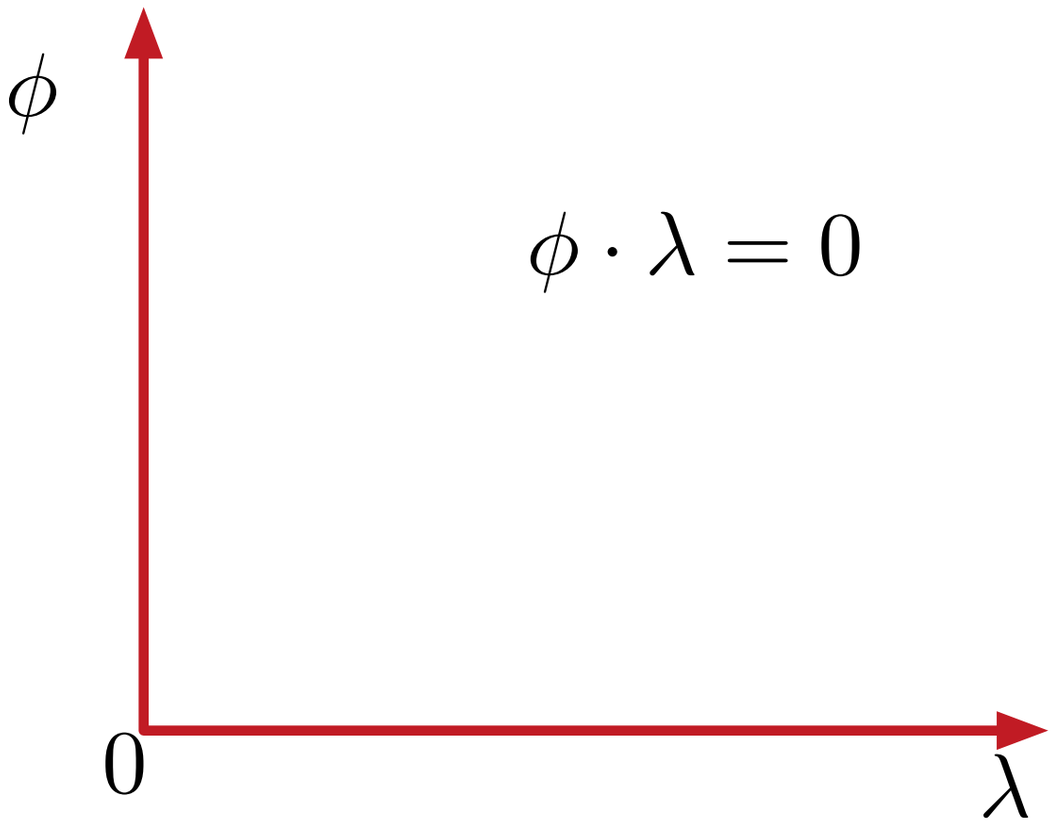
$$\text{subject to} \quad x_{k+1} = f(x_k, u_k, \lambda_k)$$

Contact-related constraints

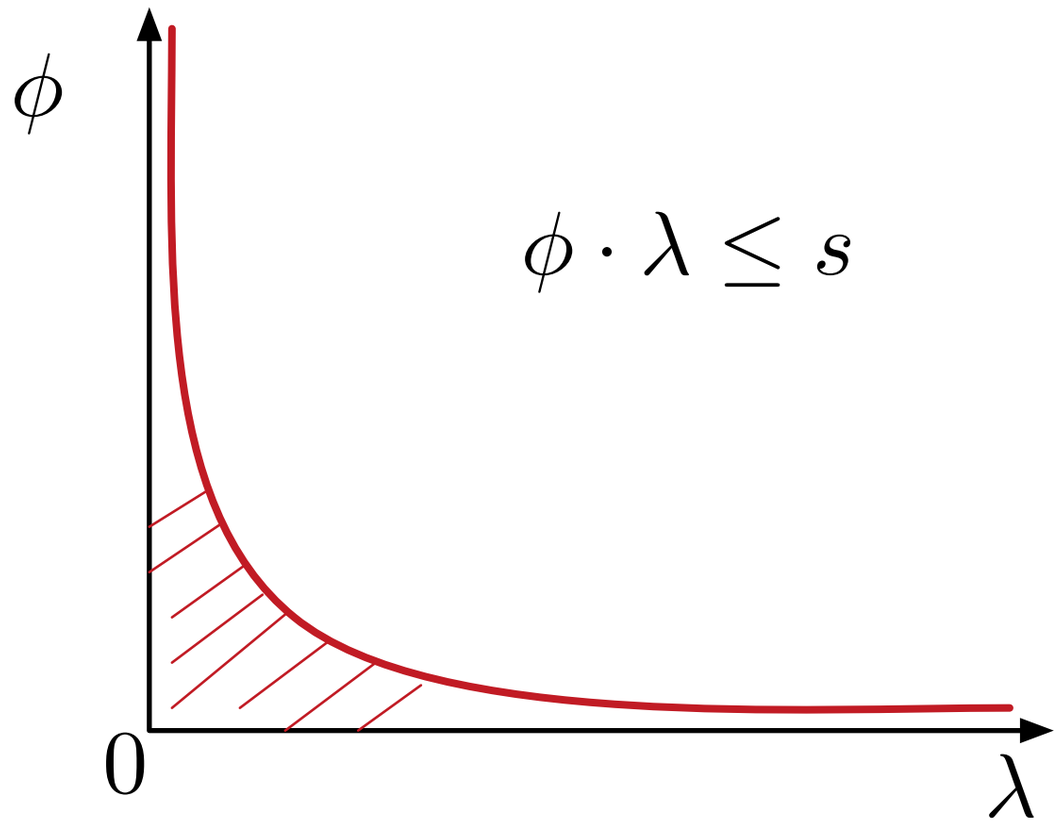
$$\begin{cases} \phi(q_k) \geq 0 \\ \phi(q_k) \cdot \lambda_k = 0 \\ \text{friction, dissipation constraints} \end{cases}$$

[see: Posa et al. IJRR'13]

Complementarity Constraints



$$\phi \cdot \lambda = 0$$

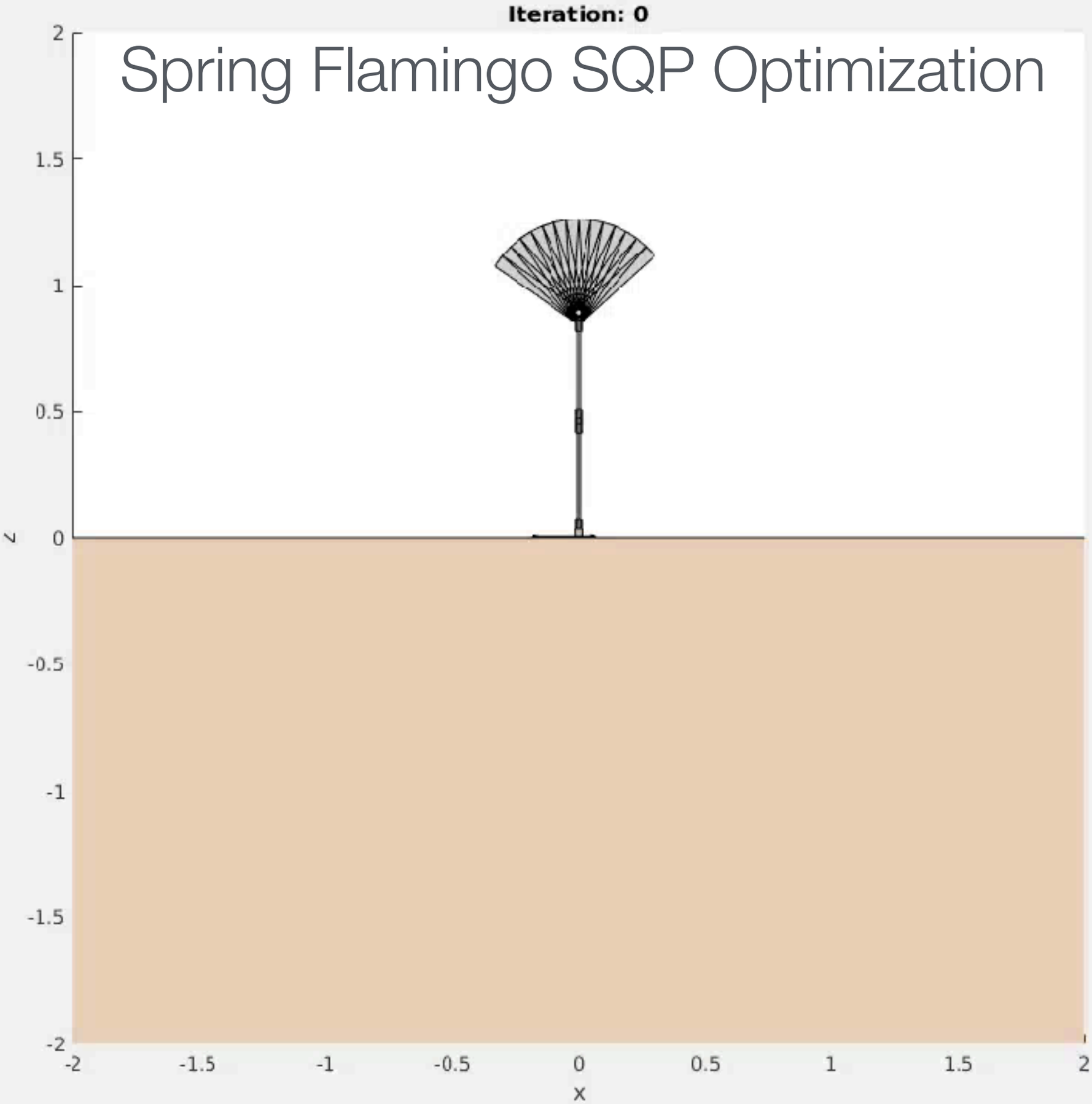


$$\phi \cdot \lambda \leq s$$

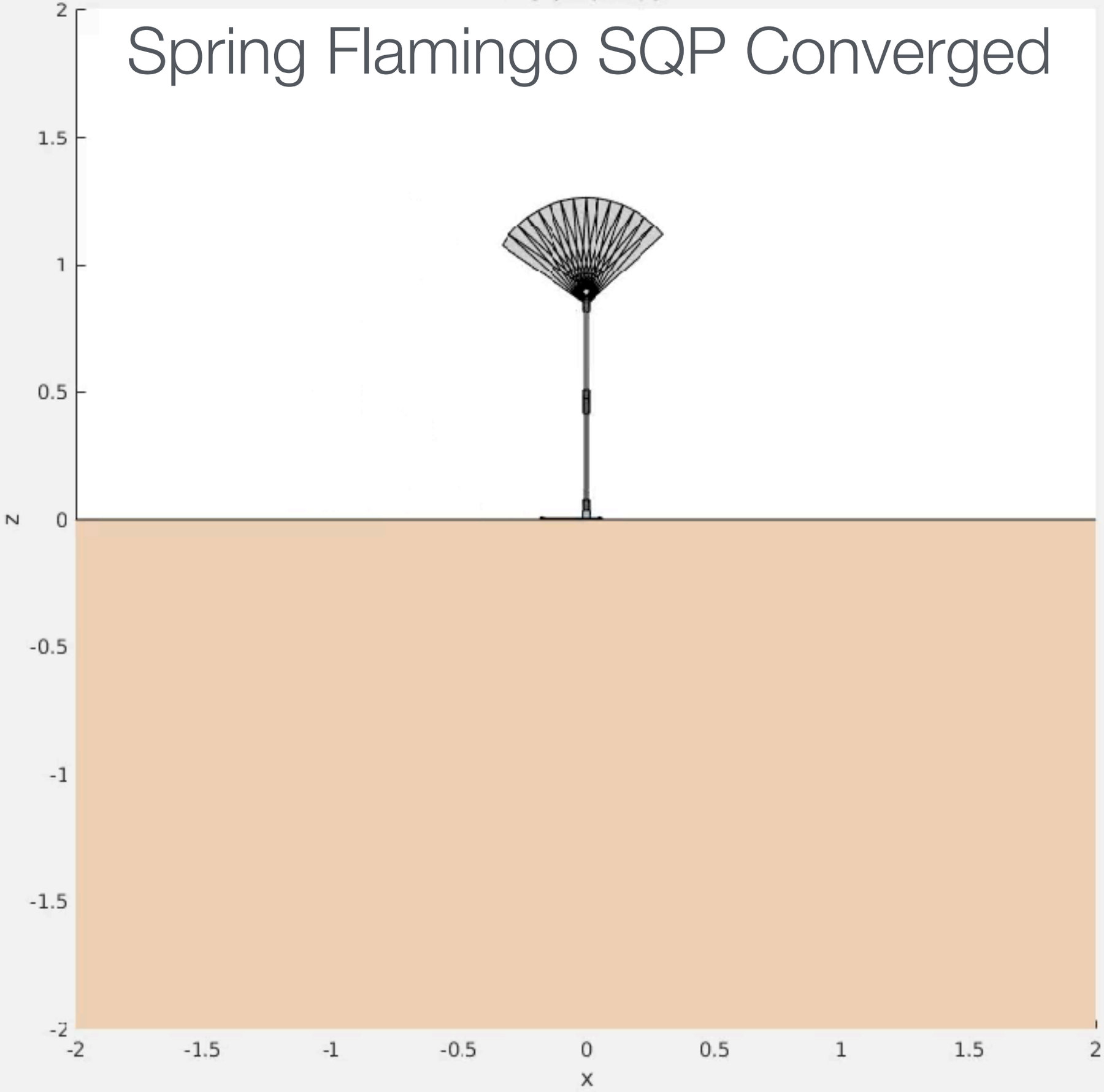
Useful trick:

add slack variable $s \geq 0$, then penalize it in the cost

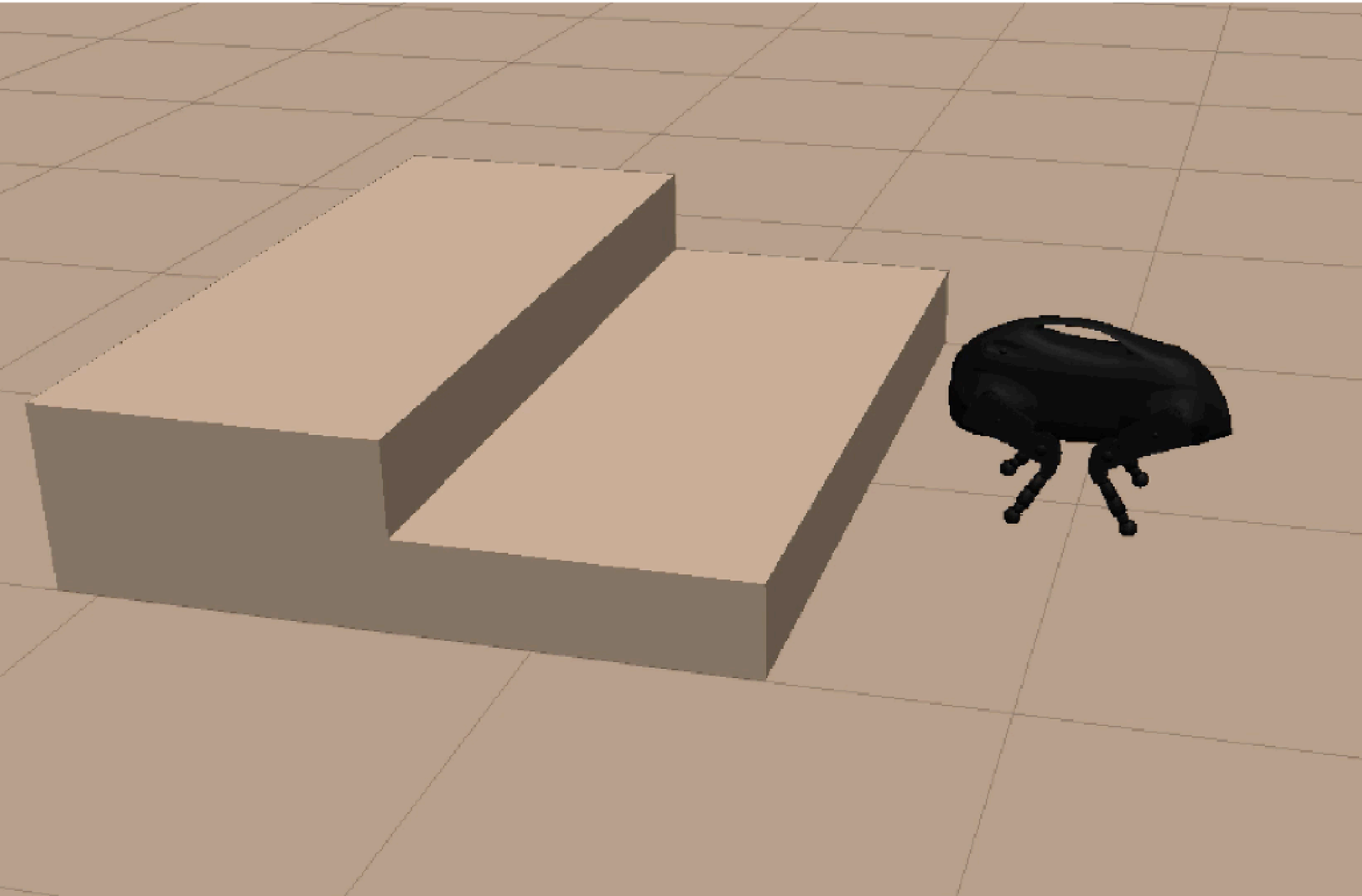
Spring Flamingo SQP Optimization



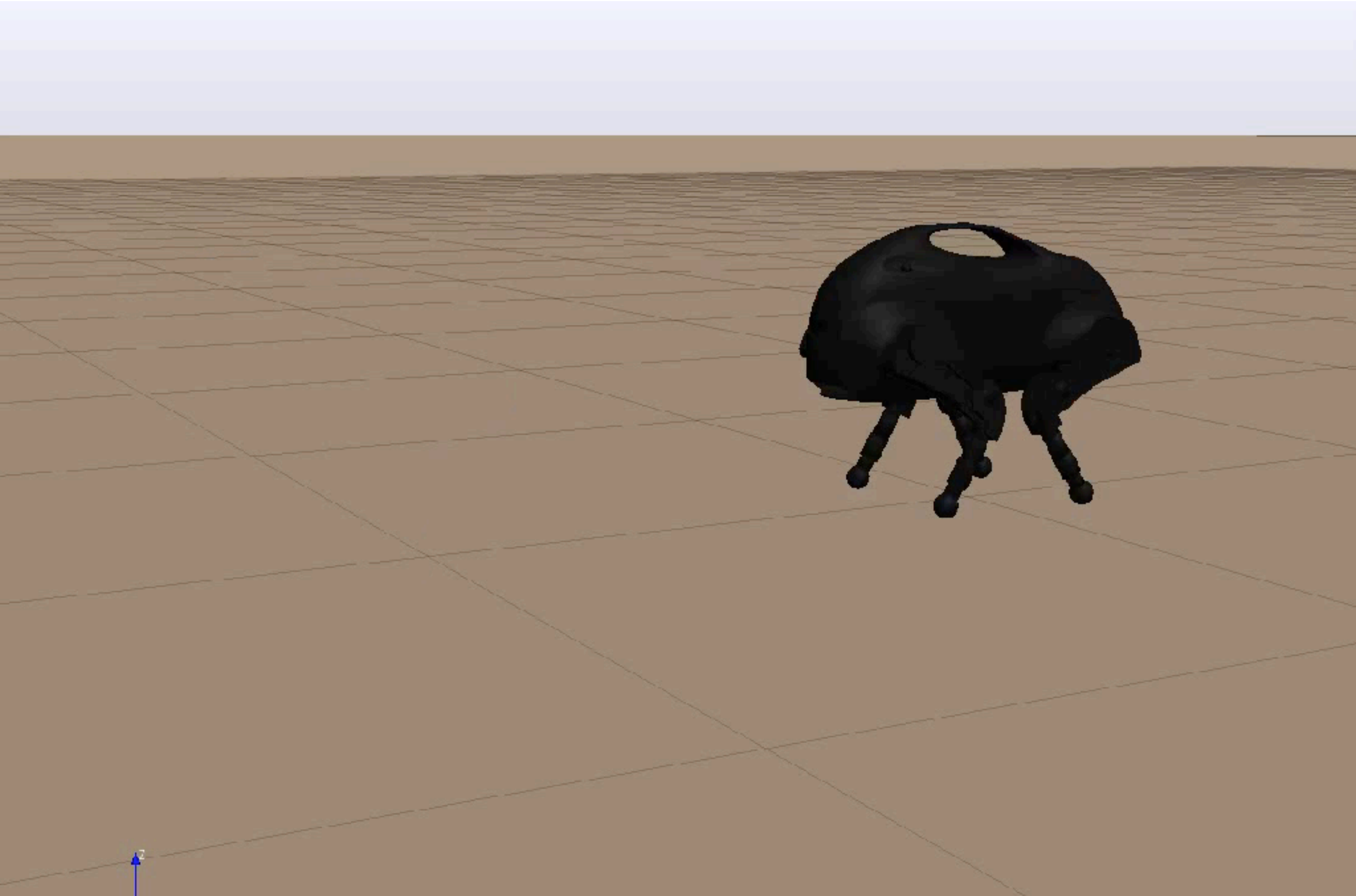
Spring Flamingo SQP Converged



LittleDog Climbing a Stair



LimpingDog



Tracking Trajectories

- Unlike DDP, direct methods output only the *nominal trajectory*

- Running this open loop will fail, even in simulation

- Hand-tuned PID can work in some cases

- Linear-quadratic regulator (LQR)

    - a slightly more elegant way to derive linear feedback controllers

LQR Trajectory Tracking

- Given trajectory: $x_{0:N}, u_{0:N-1}$
- Linearize dynamics: $\bar{x}_{k+1} = A_k \bar{x}_k + B_k \bar{u}_k$

$$\bar{x}_k = x - x_k$$
$$\bar{u}_k = u - u_k$$

- Define a tracking cost:

$$J_{\text{lqr}} = \bar{x}_N^T Q_N \bar{x}_N + \sum_{k=0}^{N-1} \bar{x}_k^T Q \bar{x}_k + \bar{u}_k^T R \bar{u}_k$$

  - *Note—in general, different from the trajectory optimization cost*

- *Desired output*: a tracking controller, $\pi(x, k)$, that works in the neighborhood of the trajectory

LQR Trajectory Tracking

- The return of Bellman!

$$V^*(\bar{x}, N) = \bar{x}^T Q_N \bar{x}$$

$$V^*(\bar{x}, k) = \bar{x}^T Q \bar{x} + \min_u \left[\bar{u}^T R \bar{u} + V^*(\bar{x}', k+1)\right]$$

- Thanks to the linearized dynamics, *V** will have a quadratic form for all *k=1:N*

$$V^*(\bar{x}, k) = \bar{x}^T S_k \bar{x}$$

$$S_{k-1} = Q + A_k^T S_k A_k - A_k^T S_k B_k (R + B_k^T S_k B_k)^{-1} (B_k^T S_k A_k)$$

(skipping ugly algebra)

Summary of LQR

- Set $S_N = Q_N$
- For *k=N:1*

$$S_{k-1} = Q + A_k^T S_k A_k - A_k^T S_k B_k (R + B_k^T S_k B_k)^{-1} (B_k^T S_k A_k)$$

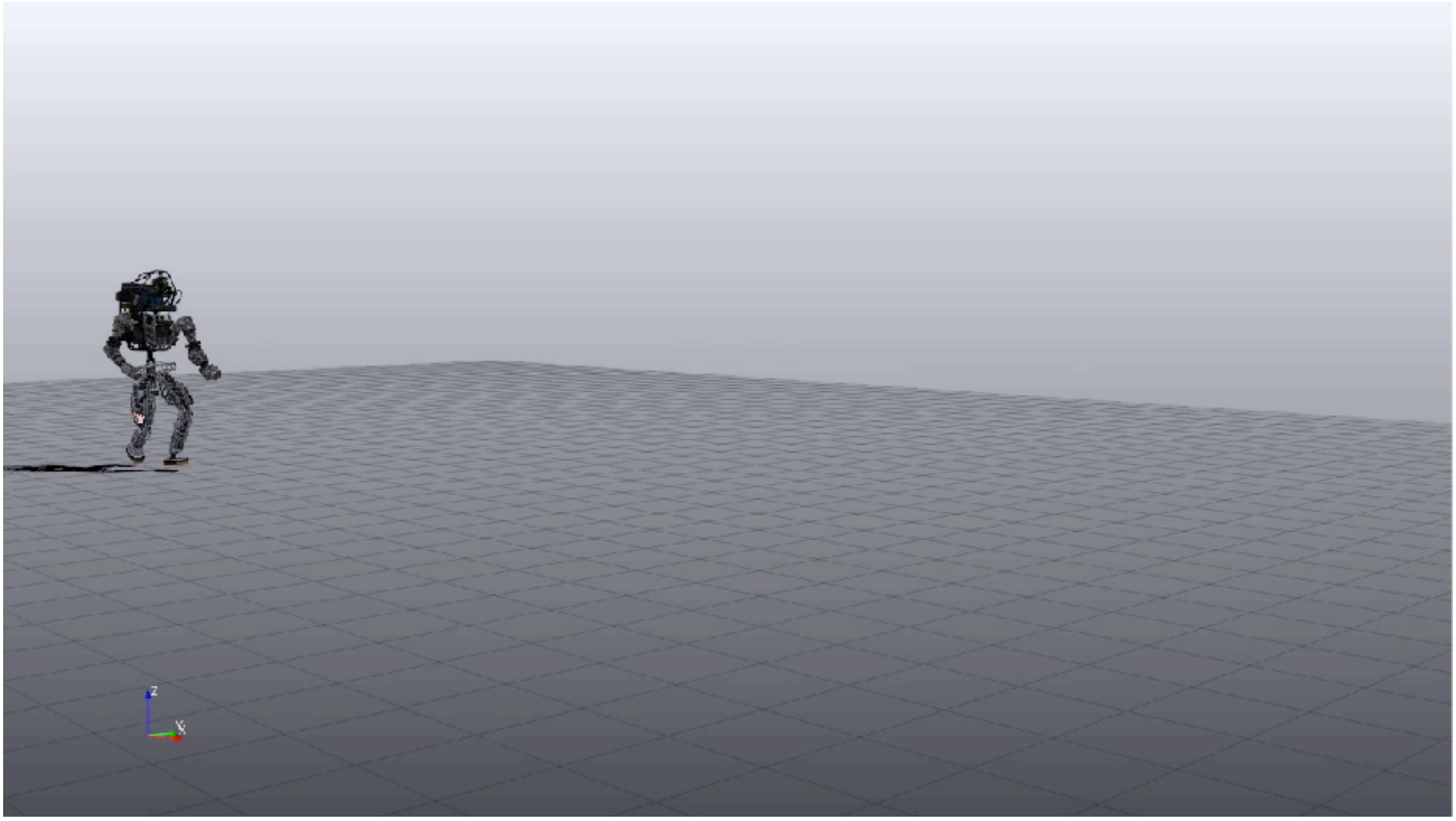
- For *k=0:N-1*

$$K_k = (R + B_k^T S_{k+1} B_k)^{-1} B_k^T S_{k+1} A_k$$

- Output linear feedback controller:

$$u_k^{\mathrm{lqr}} = -K_k(x - x_k) + u_k$$

Control Stabilized Running



LQR Variants

- Slightly different formulations for continuous time and infinite horizon

- Note MATLAB command: `lqr(A,B,Q,R)`

- Can handle equality constraints through projection into constraint nullspace (e.g. [Posa, Kuindersma, Tedrake, ICRA'15])

- "Hybrid" variants for handling impacts and reset maps

- Instantaneously handle input constraints using QPs (e.g. [Kuindersma et al., Auton. Robots 2016])

Summary

- Dynamics matters, especially (but not only) for underactuated systems
- Most of these ideas have a long history in the control literature outside of robotics
- Shooting vs direct transcription
    - Multiple shooting is a middle-ground
- Linear-quadratic feedback is a powerful tool for tracking planned motions
- Contact constraints are essential for planning manipulation and locomotion behaviors. Many competing strategies, I gave you one
- Still work to be done…

Real robots still avoid contact, but we're working on it!