ROB-GY 6323
reinforcement learning and optimal control for robotics

Lecture 6
Sampling-based optimal control

Course material

All necessary material will be posted on Brightspace

Code will be posted on the Github site of the class

https://github.com/righetti/optlearningcontrol

Discussions/Forum with Slack

Contact

ludovic.righetti@nyu.edu

Office hours in person

Wednesday 3pm to 4pm

370 Jay street - room 801

Course Assistant

Armand Jordana

aj2988@nyu.edu

Office hours Monday 1pm to 2pm

Rogers Hall 515

any other time by appointment only

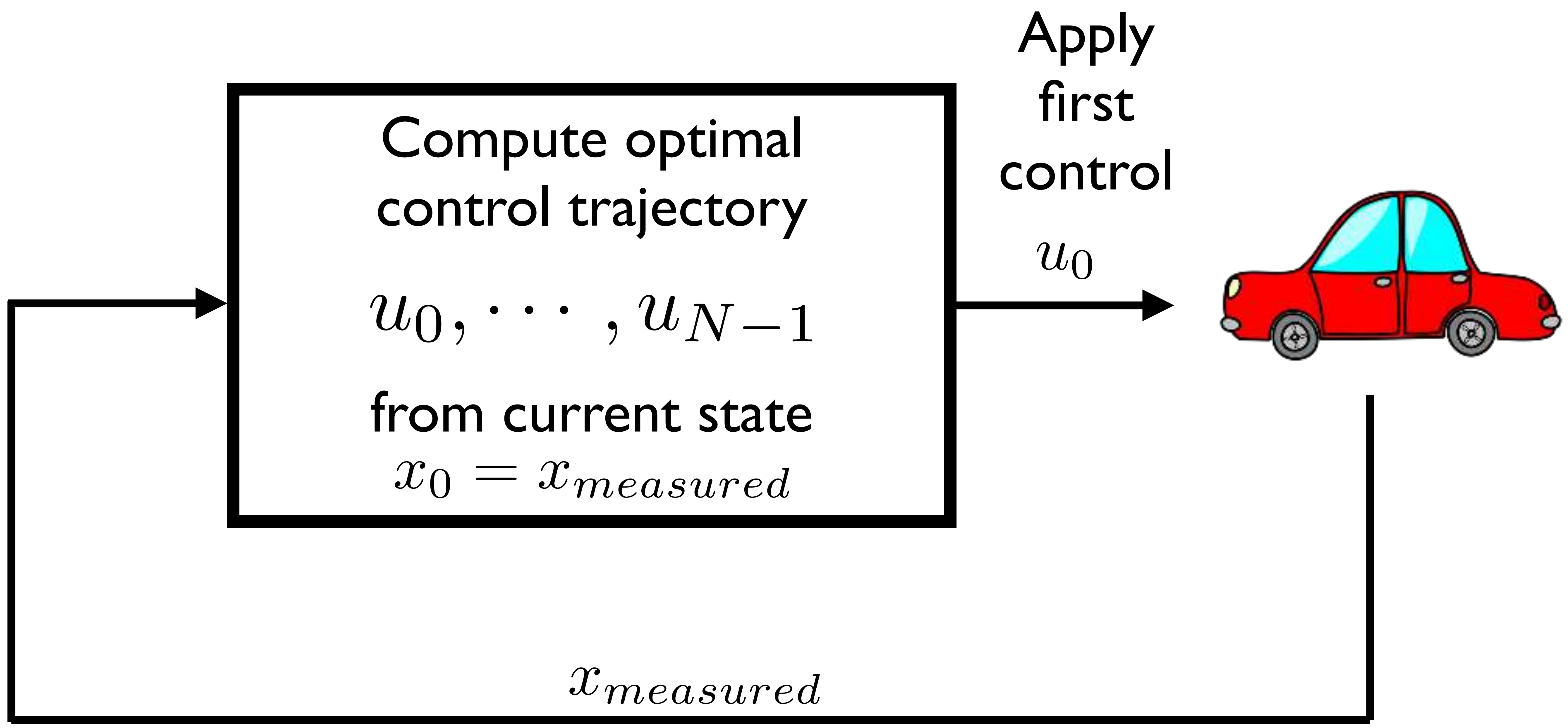
Tentative schedule (subject to change)

Week	Lecture		Homework	Project
1	Intro	Lecture 1: introduction		
2	Trajectory optimization	Lecture 2: Basics of optimization	HW 1	
3		Lecture 3: QPs		
4		Lecture 4: Nonlinear optimal control		
5		Lecture 5: Model-predictive control	HW 2	
6		Lecture 6: Sampling-based optimal control		
7	Policy optimization	Lecture 7: Bellman's principle		Project 1
8		Lecture 8: Value iteration / policy iteration	HW 4	
9		Lecture 9: TD learning - Q-learning		
10		Lecture 10: Deep Q learning	HW 5	
11		Lecture 11: Actor-critic algorithms		Project 2
12		Lecture 12: Learning by demonstration	HW 6	
13		Lecture 13: Monte-Carlo Tree Search		
14		Lecture 14: Beyond the class		
15	Finals week			

Homework 2 is available - do not wait to start!

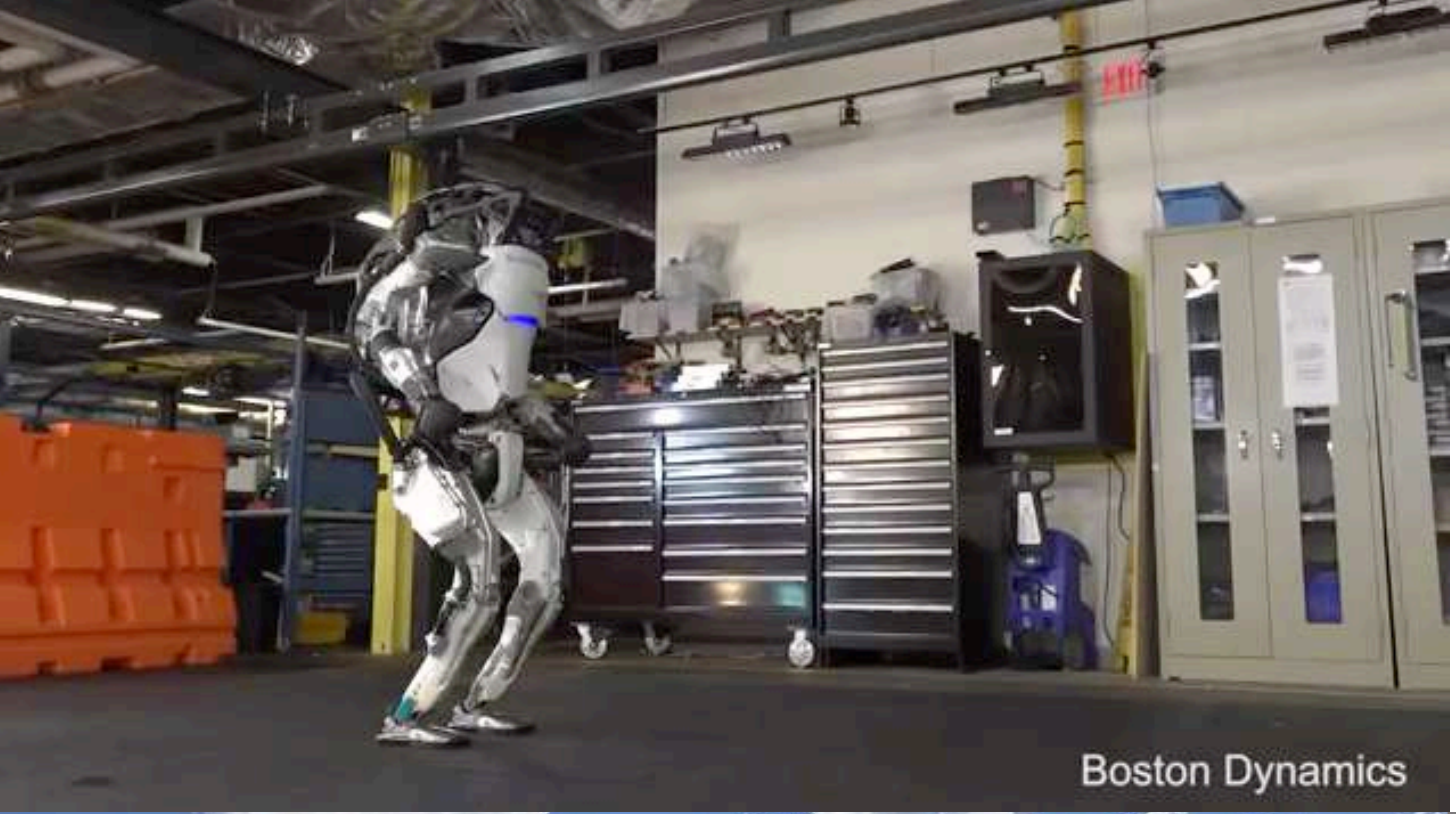
Model predictive control

Model predictive control
(receding horizon control)

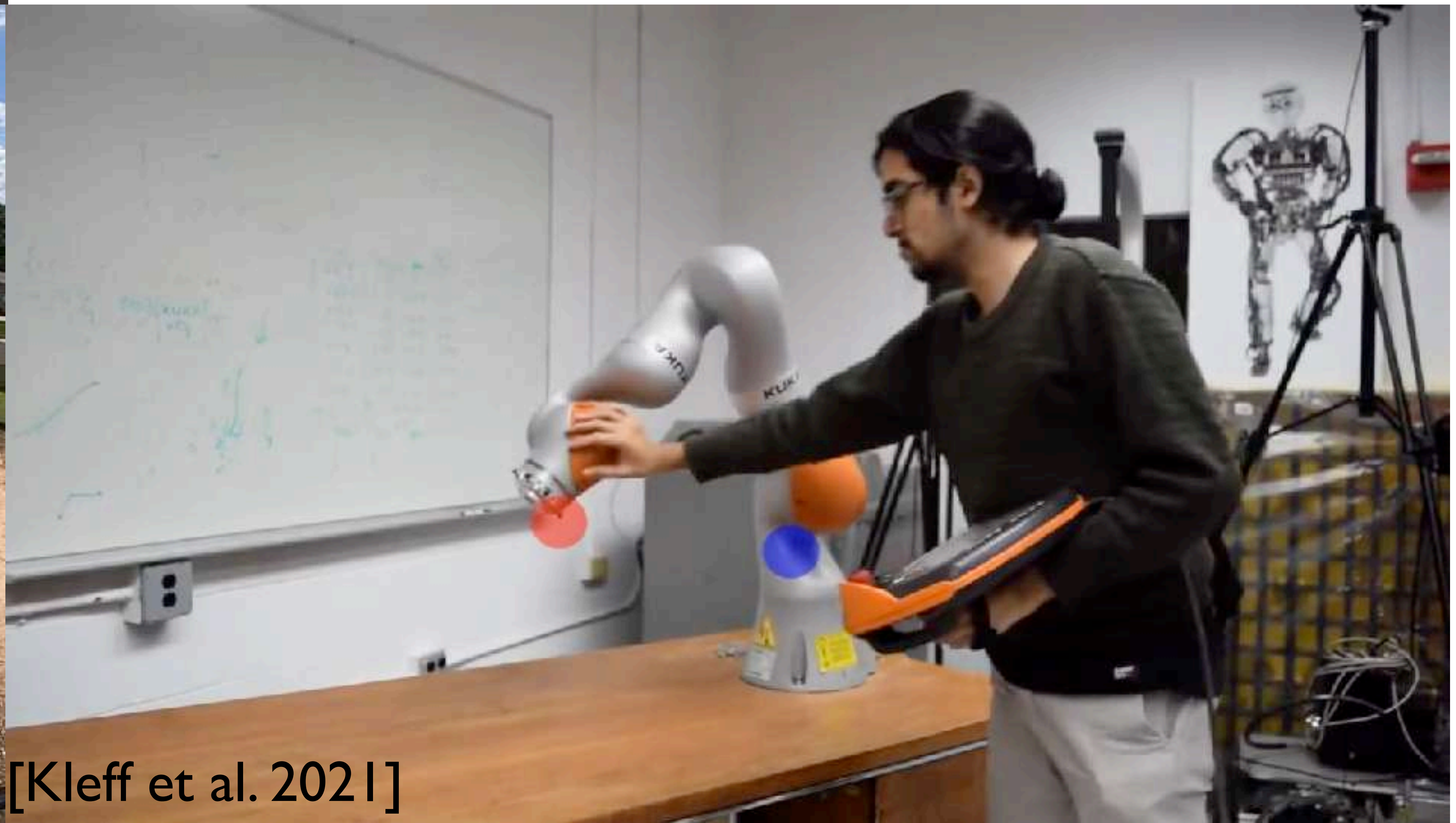


The control law solves an optimization problem at each control cycle

Model predictive control: a core ingredient in robotics

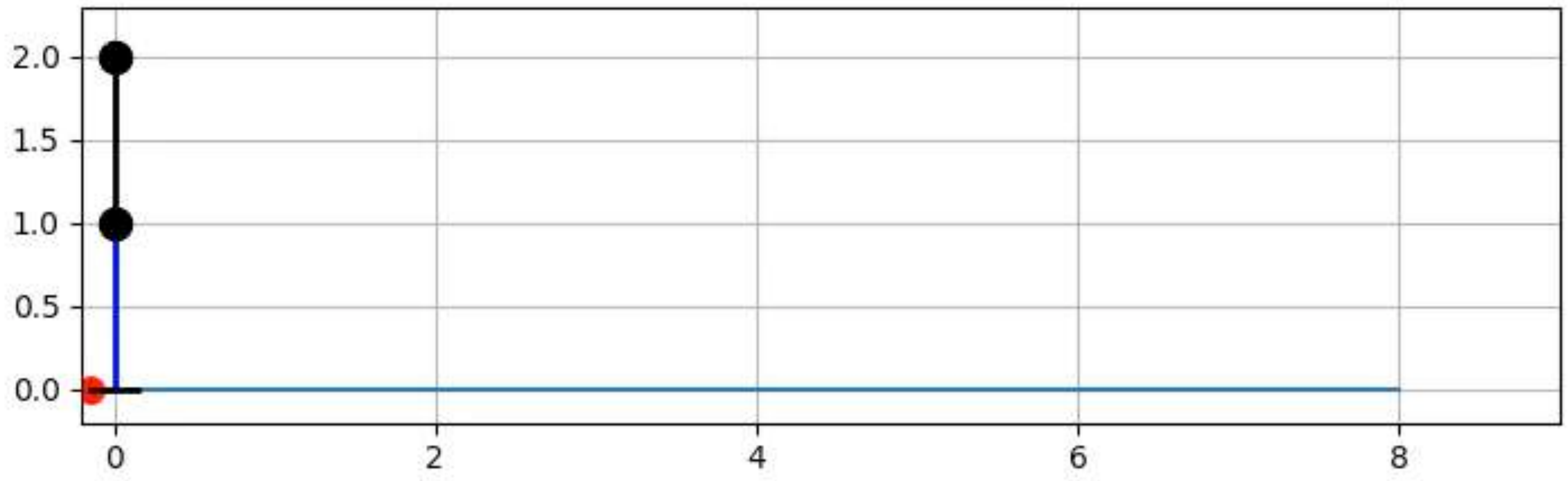


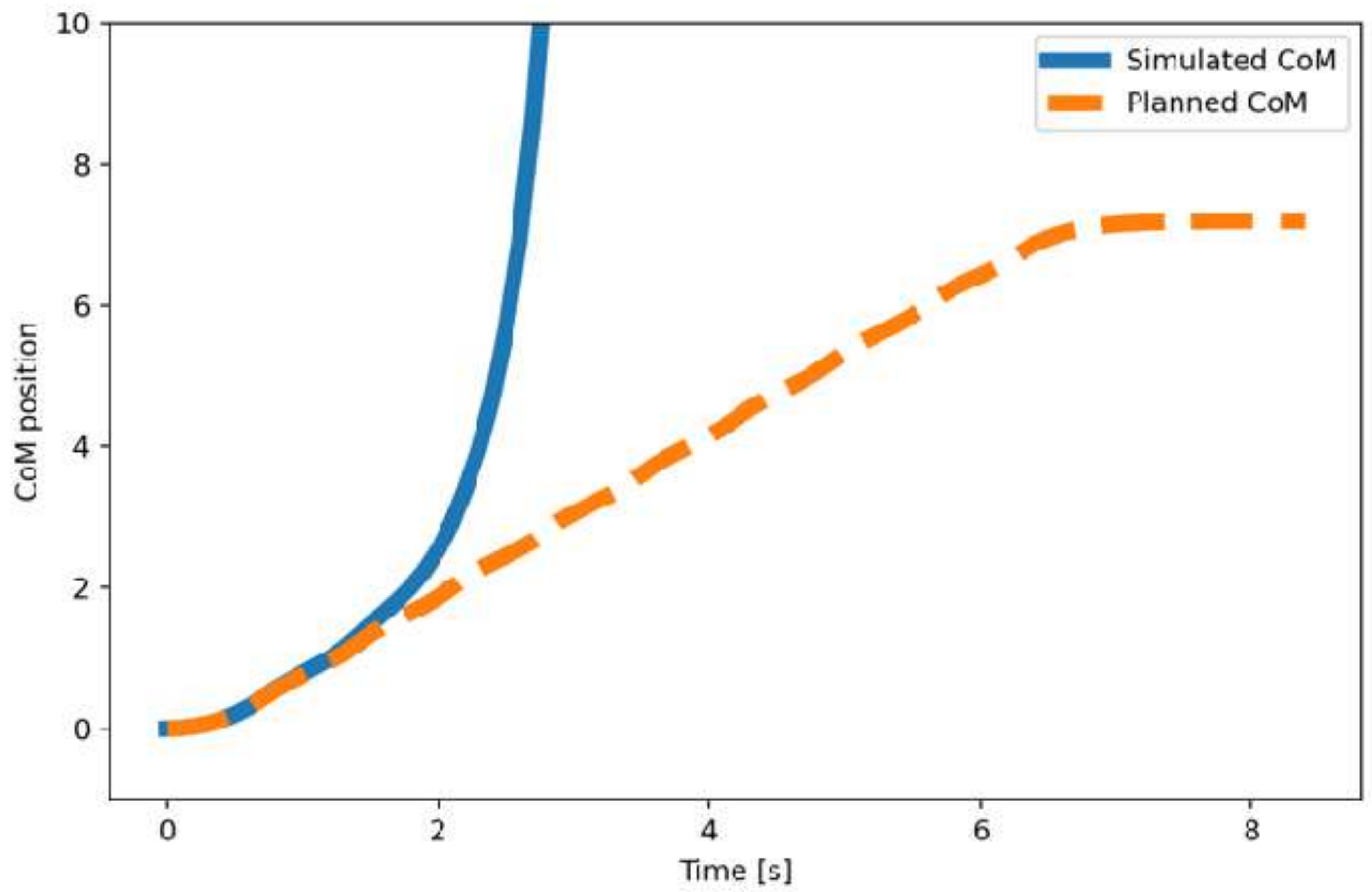
[Gandhi et al. 2020]

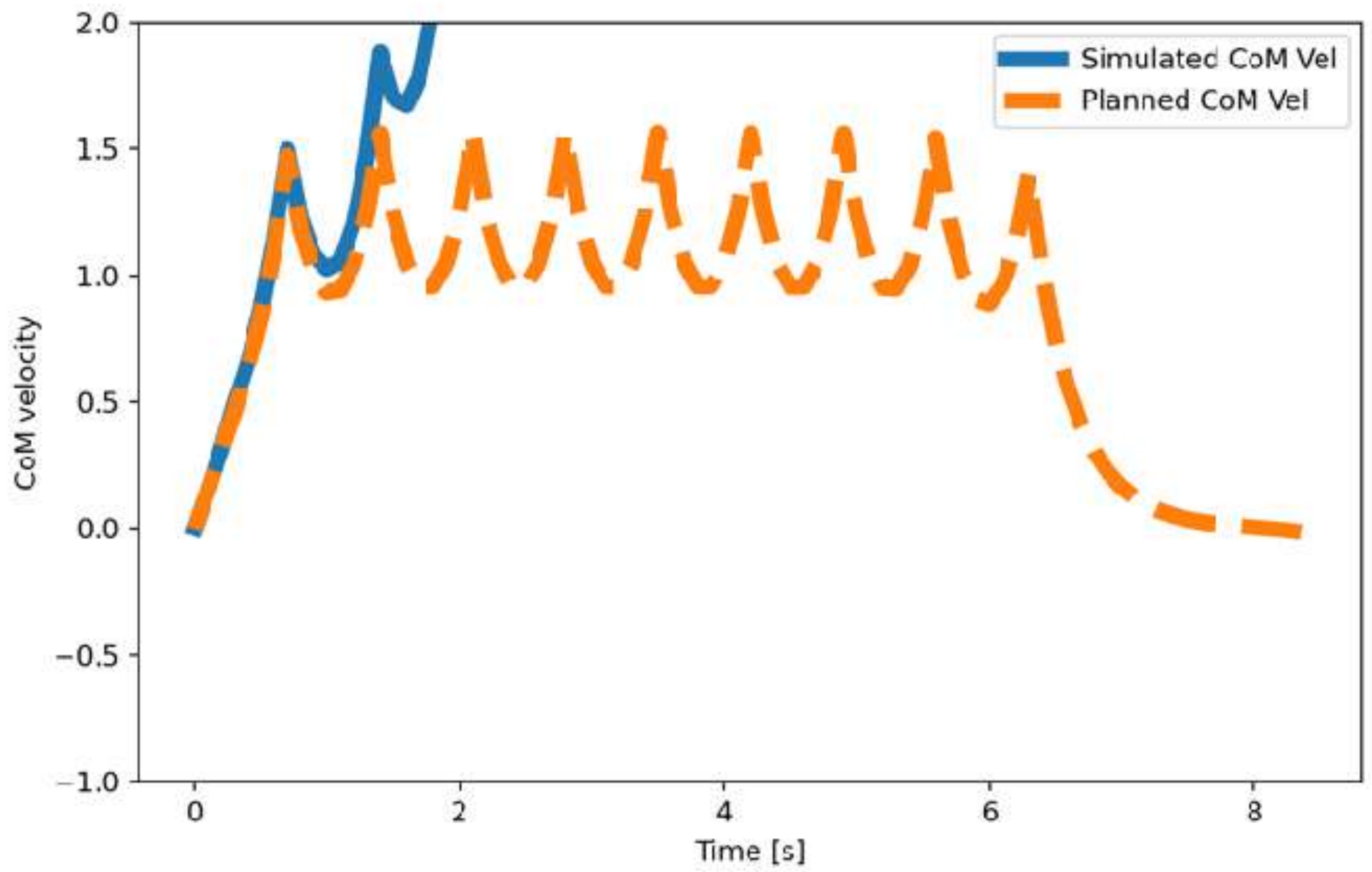


[Kleff et al. 2021]

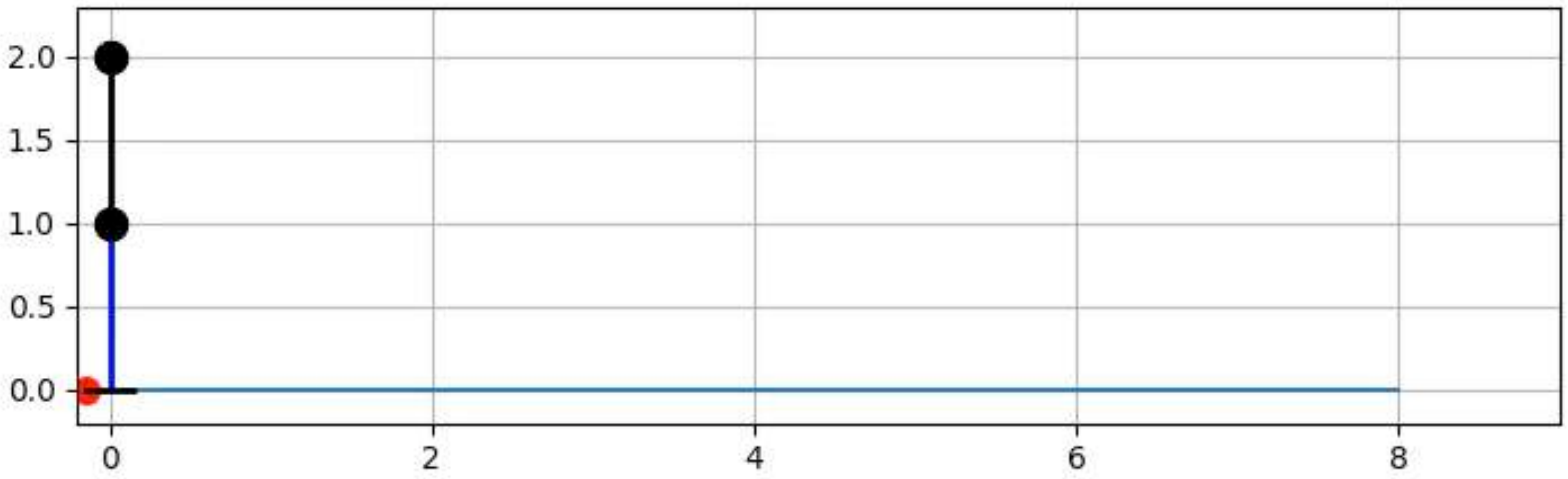
(Open-loop) optimal control executed in a “perfect environment"

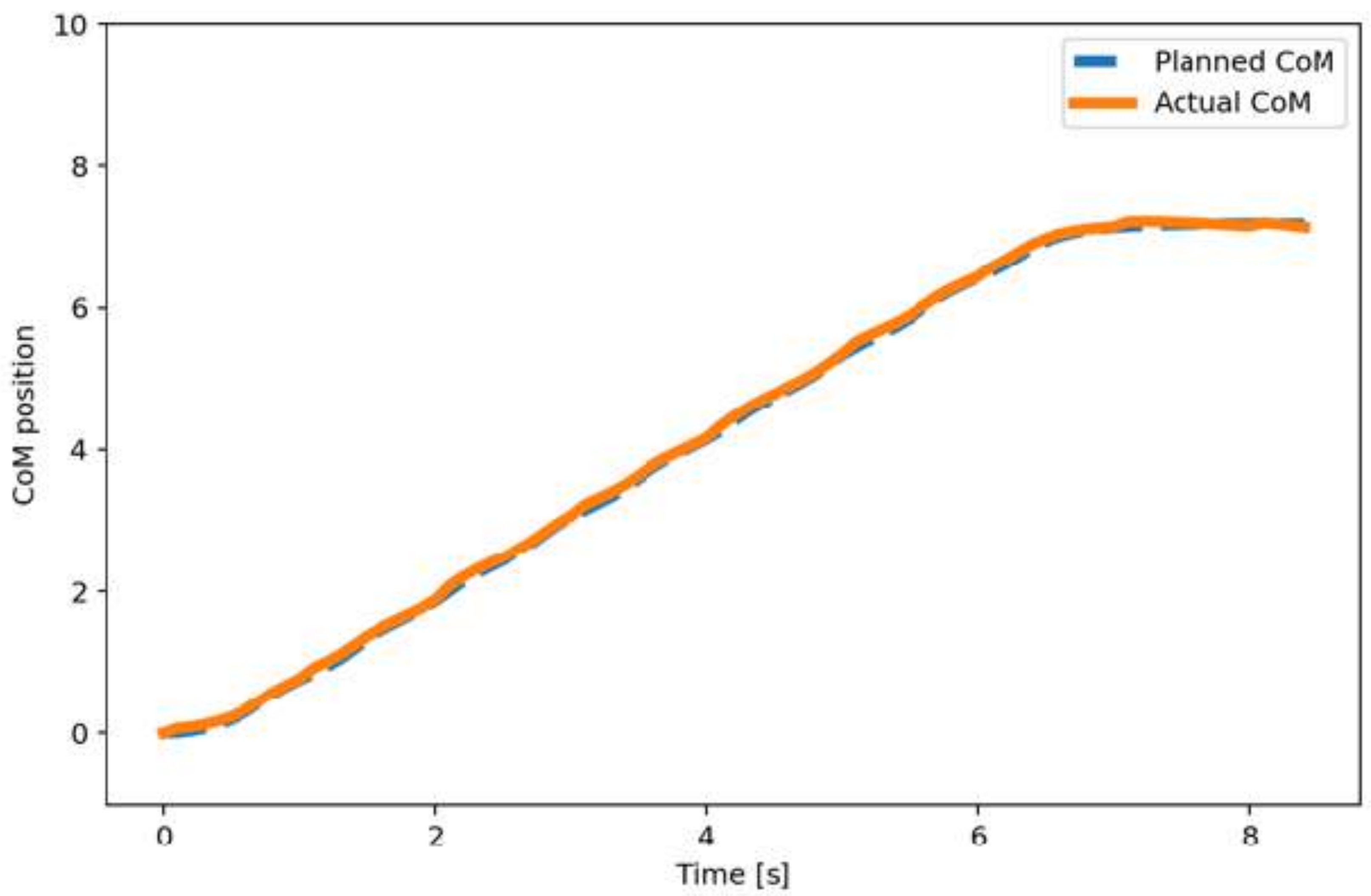


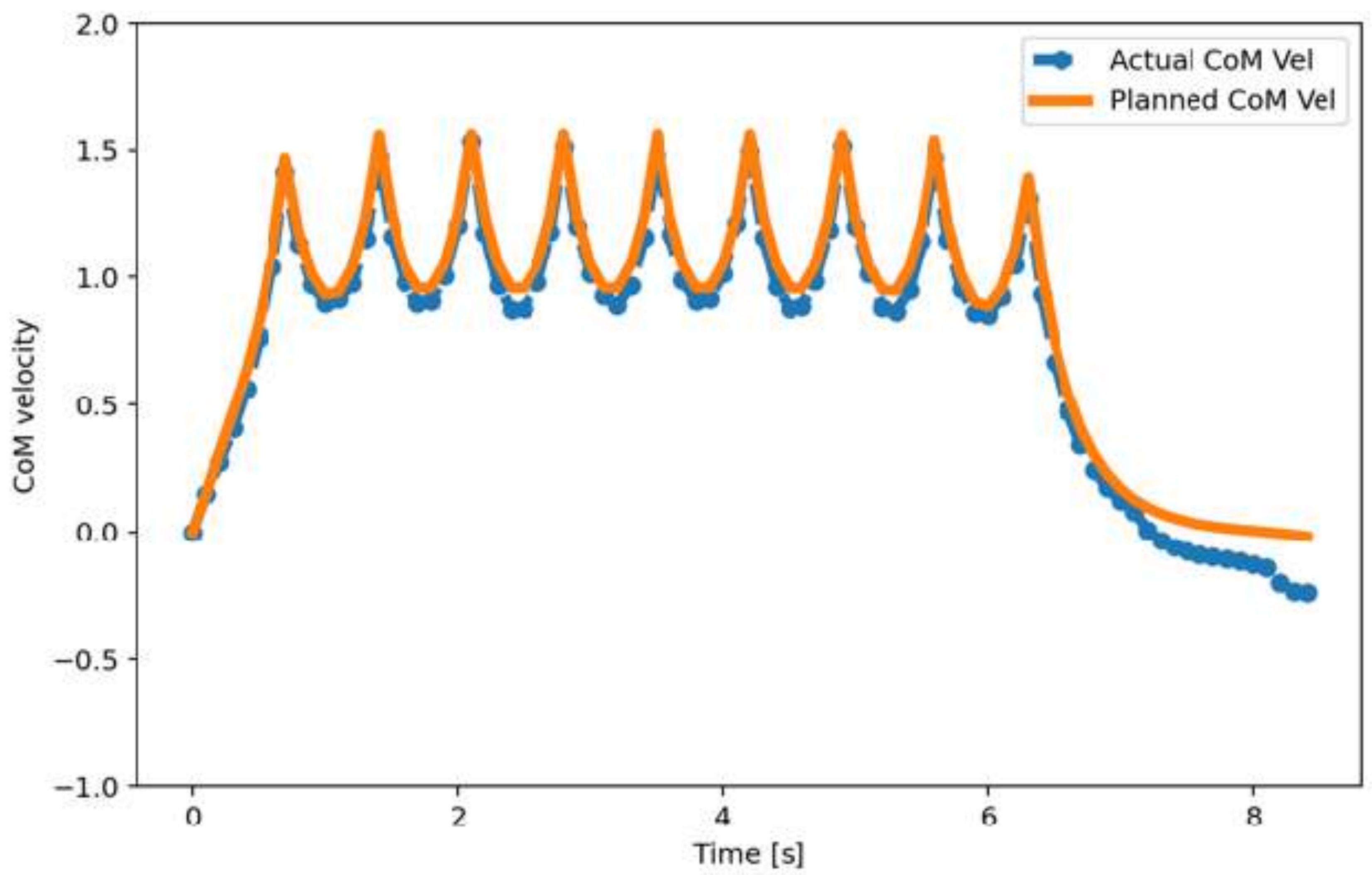


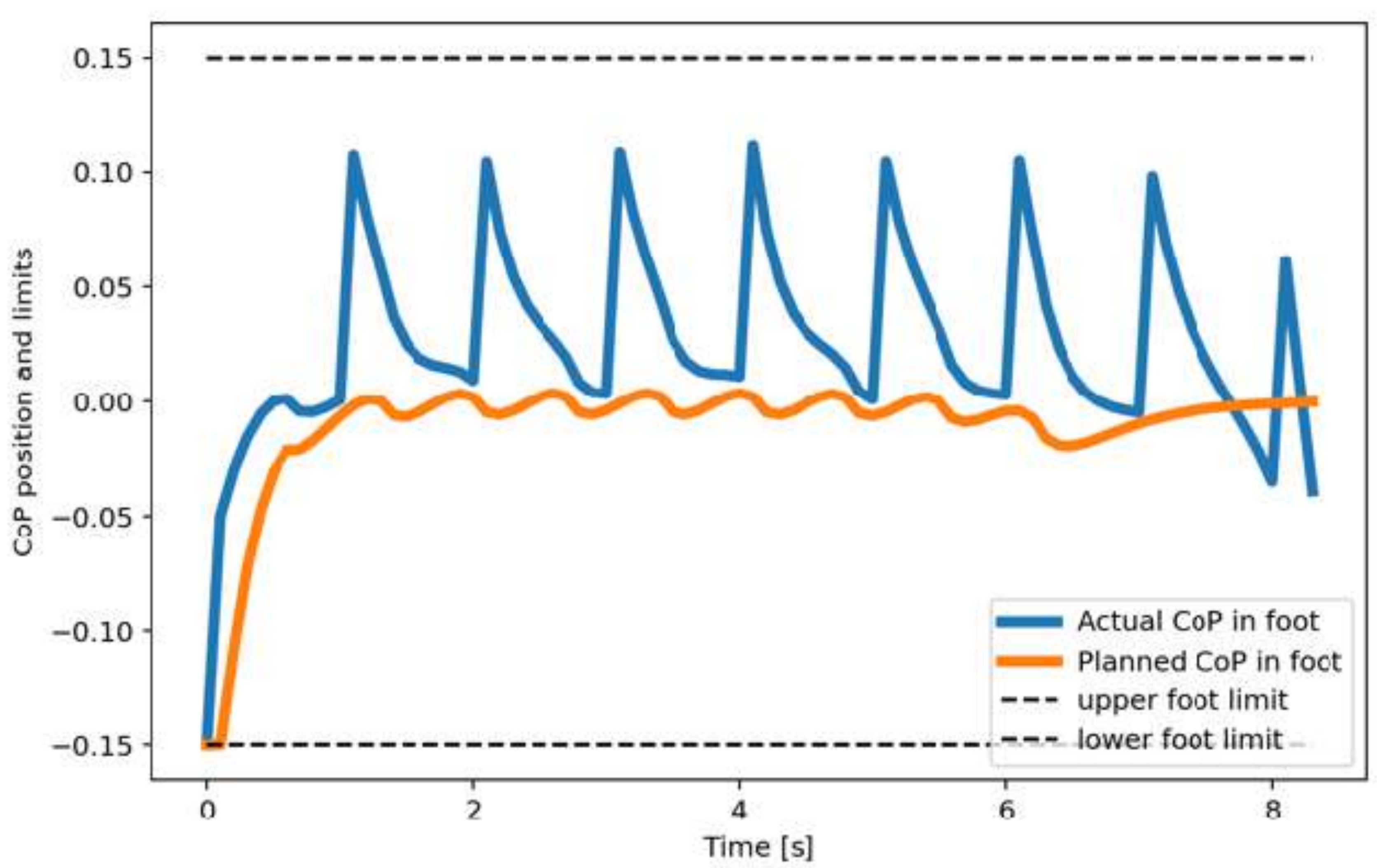


(Open-loop) optimal control executed in a “perturbed environment"

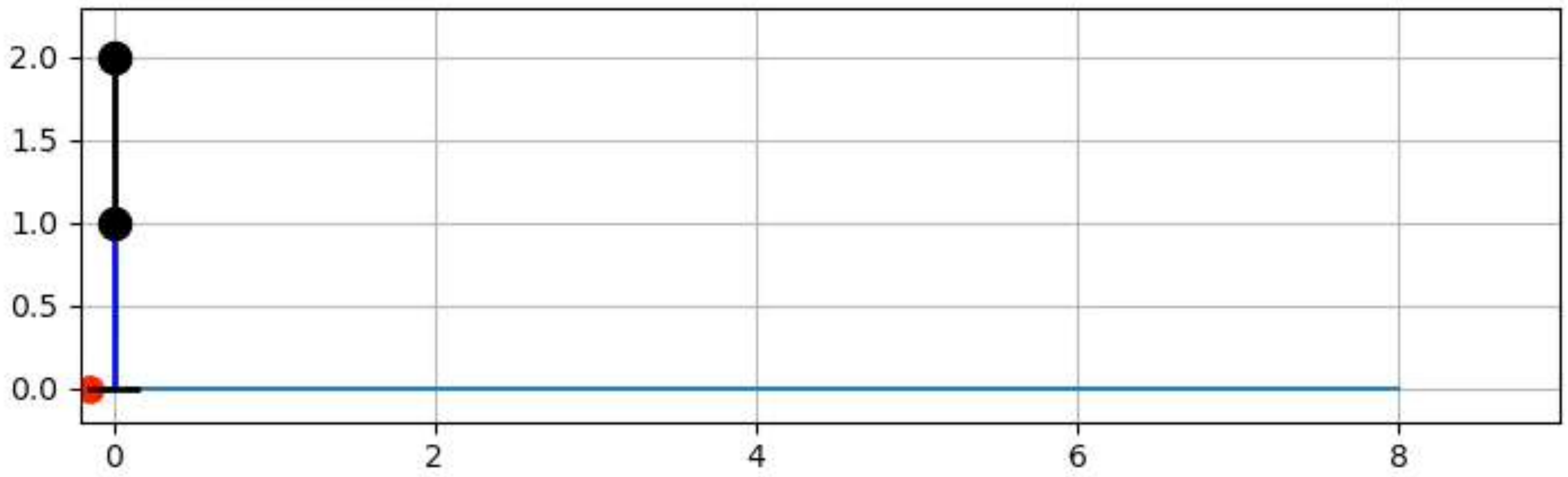








(closed-loop) model-predictive control in a “perturbed environment"



MPC: some issues

To make MPC stable and performant:

- Ideally we should optimize over an infinite horizon (not possible)
- Optimize over a horizon that is “long enough”
- Importance of having a <u>terminal cost</u> or <u>terminal constraint</u> to “stabilize” the system

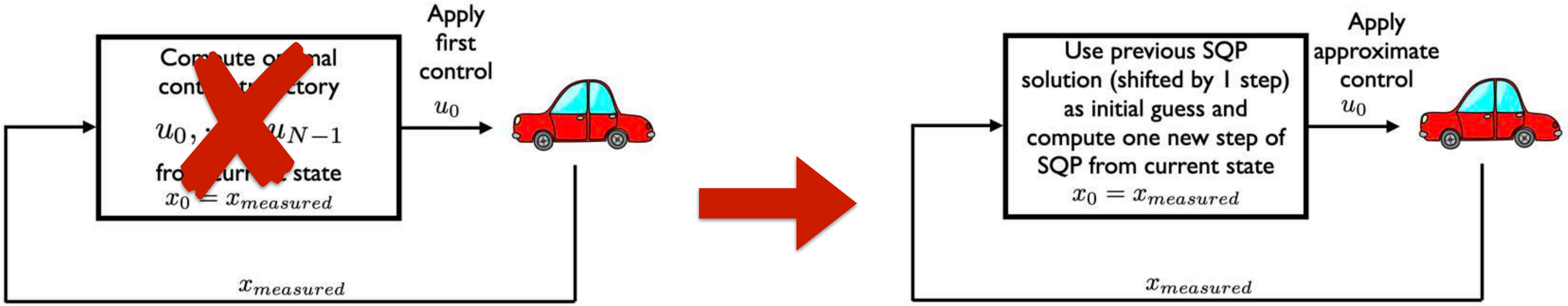
Trade-off between computational cost and horizon length

=> major importance of <u>efficient solvers</u> (e.g. use sparsity of KKT matrix, etc)

Nonlinear MPC: real-time iteration scheme

Nonlinear solvers such as SQP are iterative methods
=> they need to do several Newton-step (or equivalent) to find a solution
=> this can be very costly

Real-time iteration scheme consists in doing one step per control cycle



Computing gradients and Hessians “by hand” is generally not possible

Various gradient/Hessian estimation methods

1) finite differences

Various gradient/Hessian estimation methods

2) analytic differences

Difficult to compute for most problems (e.g. using Sympy for symbolic diff)
Very efficient algorithms for robotics!

https://github.com/stack-of-tasks/pinocchio

https://github.com/machines-in-motion/mim_solvers/

Various gradient/Hessian estimation methods

3) automatic differentiation

Various gradient/Hessian estimation methods

3) automatic differentiation

https://web.casadi.org/

https://github.com/acados/acados

https://jax.readthedocs.io/en/latest/index.html

All neural network libraries implement (some) automatic differentiation because Back Propagation is just one form of automatic differentiation!

https://pytorch.org/

Gradient estimation via Gaussian smoothing

A sampling-based gradient-descent algorithm

A detour: single shooting methods

$$\min_{\substack{x_0, \cdots, x_N \\ u_0, \cdots, u_{N-1}}} \sum_{n=0}^{N-1} l_n(x_n, u_n) + l_N(x_N)$$

$$\text{subject to} \quad x_{n+1} = f(x_n, u_n)$$

$$x_0 \text{ given}$$

Once x_0 and $u_0, \cdots, u_{N-1}$ are defined all the x_n are also defined via the dynamic constraint. They are redundant - why not remove them?

Single shooting with sampling based gradient descent

$$\min_{u_0,\cdots,u_{N-1}} \sum_{n=0}^{N-1} l_n(f^n(x_0, u_0, \cdots, u_{n-1}), u_n) + l_N(f^N(x_0, u_0, \cdots, u_{N-1}))$$

Start with a control guess $\bar{u} = [u_0, \cdots, u_{N-1}]$, then repeat until convergence

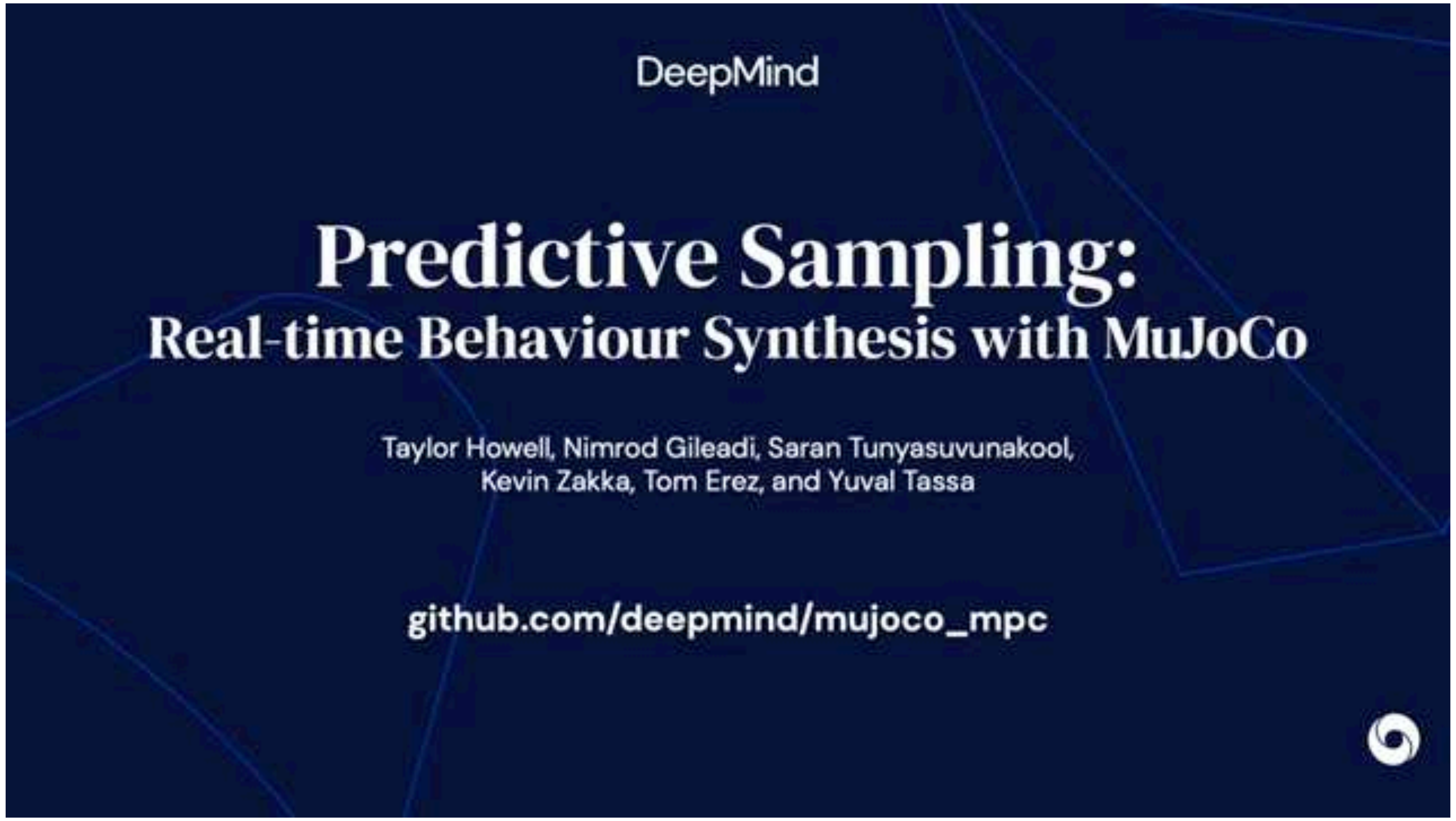
- Sample N trajectories $\bar{u} + \epsilon_n$ where ϵ $\mathcal{N}(0, I)$
- Estimate the gradient of the cost $\nabla l(\bar{u}) \simeq \frac{1}{N\sigma} \sum \left(f(\bar{u} + \epsilon_n) - f(\bar{u})\right) \epsilon_n$
- Update $\bar{u} \leftarrow \bar{u} + \alpha \nabla l(\bar{u})$

Very convenient as we can replace the dynamics f() by a simulator!

MPC variant: just take the best control from N samples

Start with a control guess $\bar{u} = [u_0, \cdots, u_{N-1}]$, then repeat forever

- Sample N trajectories $\bar{u} + \epsilon_n$ where ϵ $\mathcal{N}(0, I)$
- Compute the cost of each sample $f(\bar{u} + \epsilon_n)$
- Apply the first control of the best sample to the simulator



MPPI (Model-predictive path integral control)

MPPI (Model-predictive path integral control)

$$\bar{u} \leftarrow \bar{u} + \sum_{n=0}^{N} \omega_n \epsilon_n$$

with

$$\begin{aligned}
\omega_n &= \frac{1}{\eta} \mathrm{e}^{-\frac{1}{\lambda}(S_n - \rho)} \\
S_n &= l(\bar{u} + \epsilon_n) + \gamma \bar{u} \Sigma^{-1} \epsilon_n \\
\rho &= \min S_n \\
\eta &= \sum_n \mathrm{e}^{-\frac{1}{\lambda}(S_n - \rho)}
\end{aligned}$$

where ϵ_n is drawn from a multi-dimensional Gaussian $\mathcal{N}(0, \Sigma)$

From trajectories to policies…