

ROB-GY 6323  
reinforcement learning and optimal  
control for robotics

Lecture 3  
Linear Quadratic optimal control problems  
Quadratic programs

# **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](mailto:ludovic.righetti@nyu.edu)

Office hours in person

Wednesday 3pm to 4pm

370 Jay street - room 801

(except next week)

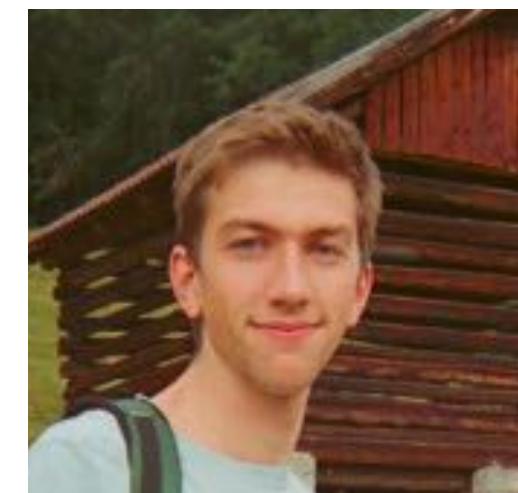
### **Course Assistant**

Armand Jordana

[aj2988@nyu.edu](mailto:aj2988@nyu.edu)

Office hours Monday 1pm to 2pm

Rogers Hall 515



any other time by appointment only

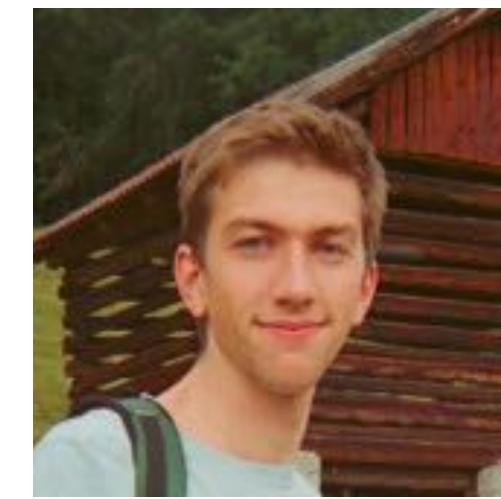
# Tentative schedule (subject to change)

Week	Lecture	Homework	Project
1	<u>Intro</u>	Lecture 1: introduction	
2		Lecture 2: Basics of optimization	HW 1
3		Lecture 3: QPs	
4	<u>Trajectory optimization</u>	Lecture 4: Nonlinear optimal control	HW 2
5		Lecture 5: Model-predictive control	
6		Lecture 6: 0th order methods (tentative)	HW 3
7		Lecture 7: Bellman's principle	
8		Lecture 8: Value iteration / policy iteration	Project 1
9		Lecture 9: TD learning - Q-learning	
10	<u>Policy optimization</u>	Lecture 10: Deep Q learning	HW 4
11		Lecture 11: Actor-critic algorithms	HW 5
12		Lecture 12: Learning by demonstration	
13		Lecture 13: Monte-Carlo Tree Search	HW 6
14		Lecture 14: Beyond the class	Project 2
15	Finals week		

# Next week

I am attending the ICRA@40 conference  
My office hour is canceled (use Slack!)

The class will be given by Armand  
He will also hold his office hours



ICRA™  
IEEE INTERNATIONAL CONFERENCE  
ON ROBOTICS AND AUTOMATION

40th Anniversary  
of the  
IEEE International Conference  
on Robotics & Automation  
(icra@40)

Home About Registration & Travel Social Events / Networking Program

ICRA@40

40th Anniversary of the IEEE International Conference on  
Robotics and Automation

23-26 September, 2024  
Rotterdam, Netherlands

**Homework I is due next week**

# Karush Kuhn Tucker conditions of optimality

$$\begin{array}{ll} \min_x f(x) & \text{subject to} \\ & g(x) = 0 \\ & h(x) \leq 0 \end{array}$$

We define the Lagrangian as  $L(x, \lambda, \mu) = f(x) + \lambda^T g(x) + \mu^T h(x)$

The vectors  $\lambda$  and  $\mu$  are called the Lagrange multipliers

## First order necessary conditions (KKT conditions)

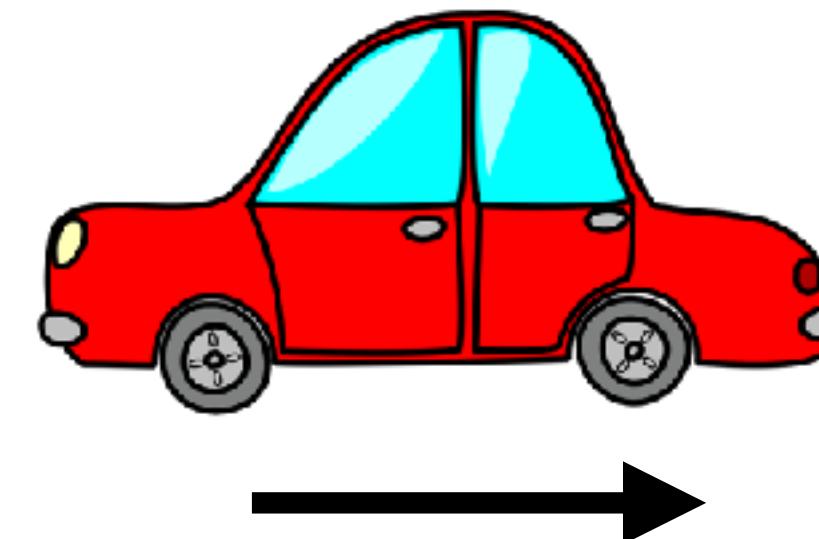
Suppose that  $x^*$  is a local solution and that the LICQ holds at  $x^*$  (and that  $f$ ,  $g_i$  and  $h_i$  are continuously differentiable). Then there are Lagrange multiplier vectors  $\lambda^*$  and  $\mu^*$  such that the following conditions are satisfied

$$\begin{aligned} \nabla_x L(x^*, \lambda^*, \mu^*) &= 0 \\ g(x^*) &= 0 \\ h(x^*) &\leq 0 \\ \mu_i &\geq 0 \quad \forall i \\ \mu_i h_i(x^*) &= 0 \quad \forall i \end{aligned}$$

# Optimal control of linear systems with quadratic costs

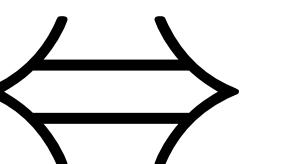
$$\min_{x_n, u_n} \frac{1}{2} \sum_{n=0}^{N-1} x_n^T Q x_n + u_n^T R u_n + x_N^T Q x_N$$

subject to  $x_{n+1} = Ax_n + Bu_n$   
 $x_0 = x_{init}$



**Goal**  
X

$$\min_{x_n, u_n} \frac{1}{2} \sum_{n=0}^{N-1} x_n^T Q x_n + u_n^T R u_n + x_N^T Q x_N$$



$$\min_{x_n, u_n} \frac{1}{2} \begin{pmatrix} x_0 \\ u_0 \\ x_1 \\ u_1 \\ \vdots \end{pmatrix}^T \begin{bmatrix} Q & 0 & 0 & 0 & \cdots \\ 0 & R & 0 & 0 & \cdots \\ 0 & 0 & Q & 0 & \cdots \\ 0 & 0 & 0 & R & \cdots \\ 0 & 0 & 0 & 0 & \ddots \end{bmatrix} \begin{pmatrix} x_0 \\ u_0 \\ x_1 \\ u_1 \\ \vdots \end{pmatrix}$$

$G$

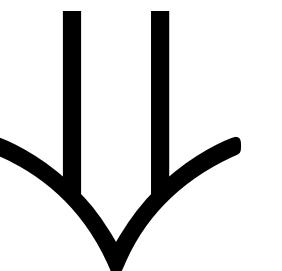
$y$

subject to  $x_{n+1} = Ax_n + Bu_n$   
 $x_0 = x_{init}$

subject to

$$\begin{bmatrix} I & 0 & 0 & 0 & 0 & 0 & \cdots \\ A & B & -I & 0 & 0 & 0 & \cdots \\ 0 & 0 & A & B & -I & 0 & \cdots \\ 0 & 0 & 0 & 0 & A & B & \cdots \end{bmatrix} \begin{pmatrix} x_0 \\ u_0 \\ x_1 \\ u_1 \\ \vdots \end{pmatrix} = \begin{pmatrix} x_{init} \\ 0 \\ 0 \\ 0 \\ \vdots \\ p \end{pmatrix}$$

$M$



$$\min_y \frac{1}{2} y^T G y$$

subject to  $My = p$

# Optimal control of linear systems with quadratic costs

$$\min_y \frac{1}{2} y^T G y$$

$$L(y, \lambda) = \frac{1}{2} y^T G y + \lambda^T (M y - p)$$

subject to  $M y = p$

As long as  $G \geq 0$ , the problem is convex (convex domain and convex function to minimize) and therefore the solution to the KKT system is guaranteed to be a minimum

The KKT conditions are then

$$\nabla_x L = G y + M^T \lambda = 0$$

$$\nabla_\lambda L = M y - p = 0$$

or equivalently

$$\begin{bmatrix} G & M^T \\ M & 0 \end{bmatrix} \begin{pmatrix} y \\ \lambda \end{pmatrix} = \begin{pmatrix} 0 \\ p \end{pmatrix}$$

$$\Rightarrow \begin{pmatrix} y \\ \lambda \end{pmatrix} = \begin{bmatrix} G & M^T \\ M & 0 \end{bmatrix}^{-1} \begin{pmatrix} 0 \\ p \end{pmatrix}$$

If the matrix is invertible

# Example

# Inverting a matrix full of zeros...

$$\begin{bmatrix} G & M^T \\ M & 0 \end{bmatrix} \begin{pmatrix} y \\ \lambda \end{pmatrix} = \begin{pmatrix} 0 \\ p \end{pmatrix} \quad \text{✖} \quad \begin{pmatrix} y \\ \lambda \end{pmatrix} = \begin{bmatrix} G & M^T \\ M & 0 \end{bmatrix}^{-1} \begin{pmatrix} 0 \\ p \end{pmatrix}$$

Efficient algorithm to solve the KKT system:

- 1 Compute backward (from N to 0):

$$P_N = Q$$

$$K_n = (R + B^T P_n B)^{-1} B^T P_n A$$

$$P_{n-1} = Q + A^T P_n A - A^T P_n B K_n$$

$K_n$  are called "feedback gains"

- 2 Compute forward (from 0 to N) starting with  $x_0 = x_{init}$

$$u_n = -K_n x_n$$

$$x_{n+1} = Ax_n + Bu_n$$

The number of multiplications needed to re-solve the KKT system without taking into account 0s will grow like  $N^3$  while the efficient algorithm as a number of operations that grow like  $N$ . This is much better!

# Example

$$\min \sum_{i=0}^{N-1} (2x_i^2 + u_i^2) + 2x_N^2 \quad Q = 2 \quad R = 1$$

Subject to  $x_{n+1} = 2x_n + u_n \quad A = 2 \quad B = 1$

$$N = 10 \quad P_{10} = Q = 2$$

$$\begin{aligned} K_9 &= (B^T P_{10} B + R)^{-1} B^T P_{10} A \\ &= (2+1)^{-1} 4 = \frac{4}{3} \end{aligned}$$

$$\begin{aligned} P_9 &= Q + A^T P_{10} A - A^T P_{10} B K_9 \\ &= 2 + 8 - \frac{16}{3} = \frac{14}{3} \end{aligned}$$

$$K_8 = \dots$$

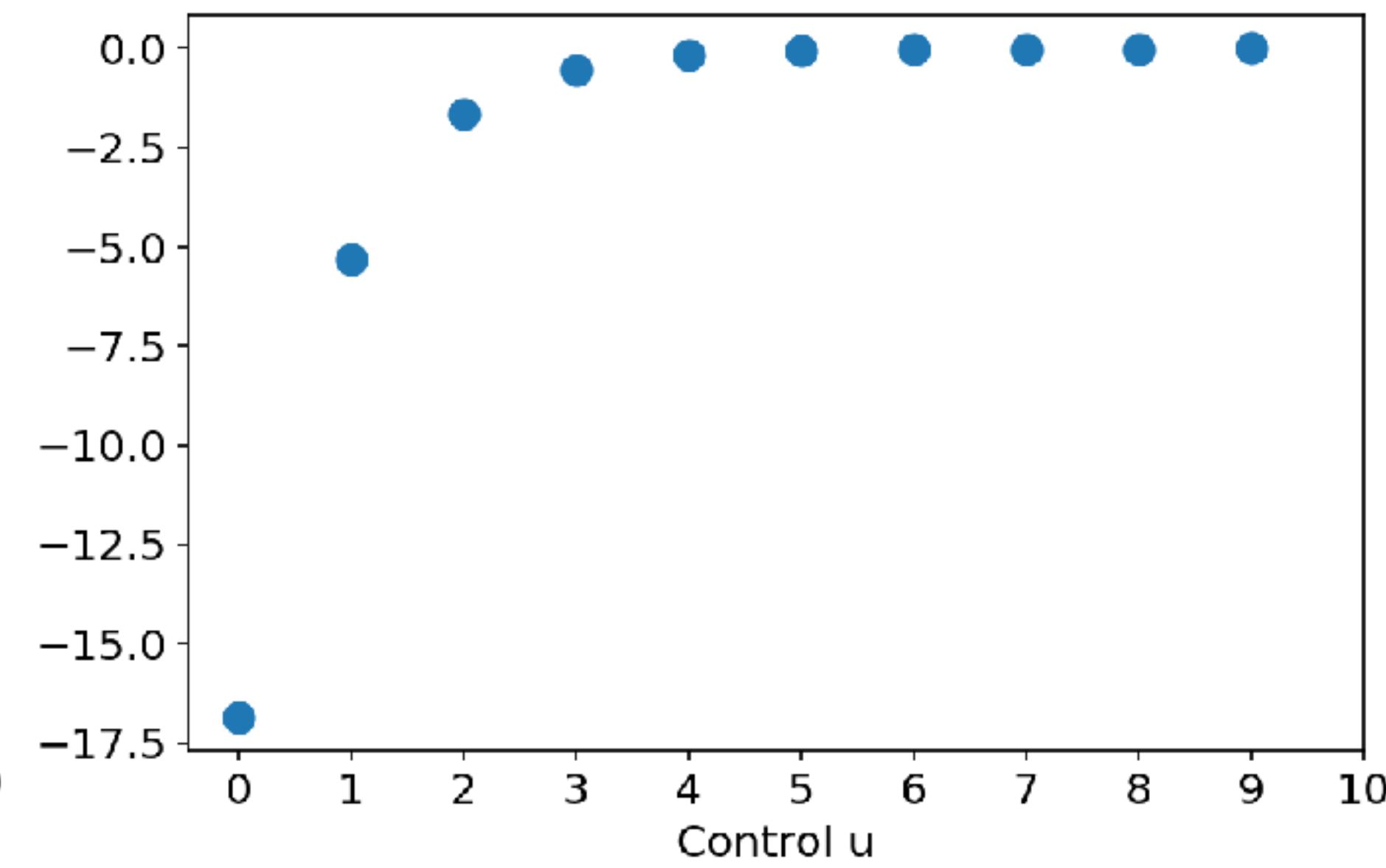
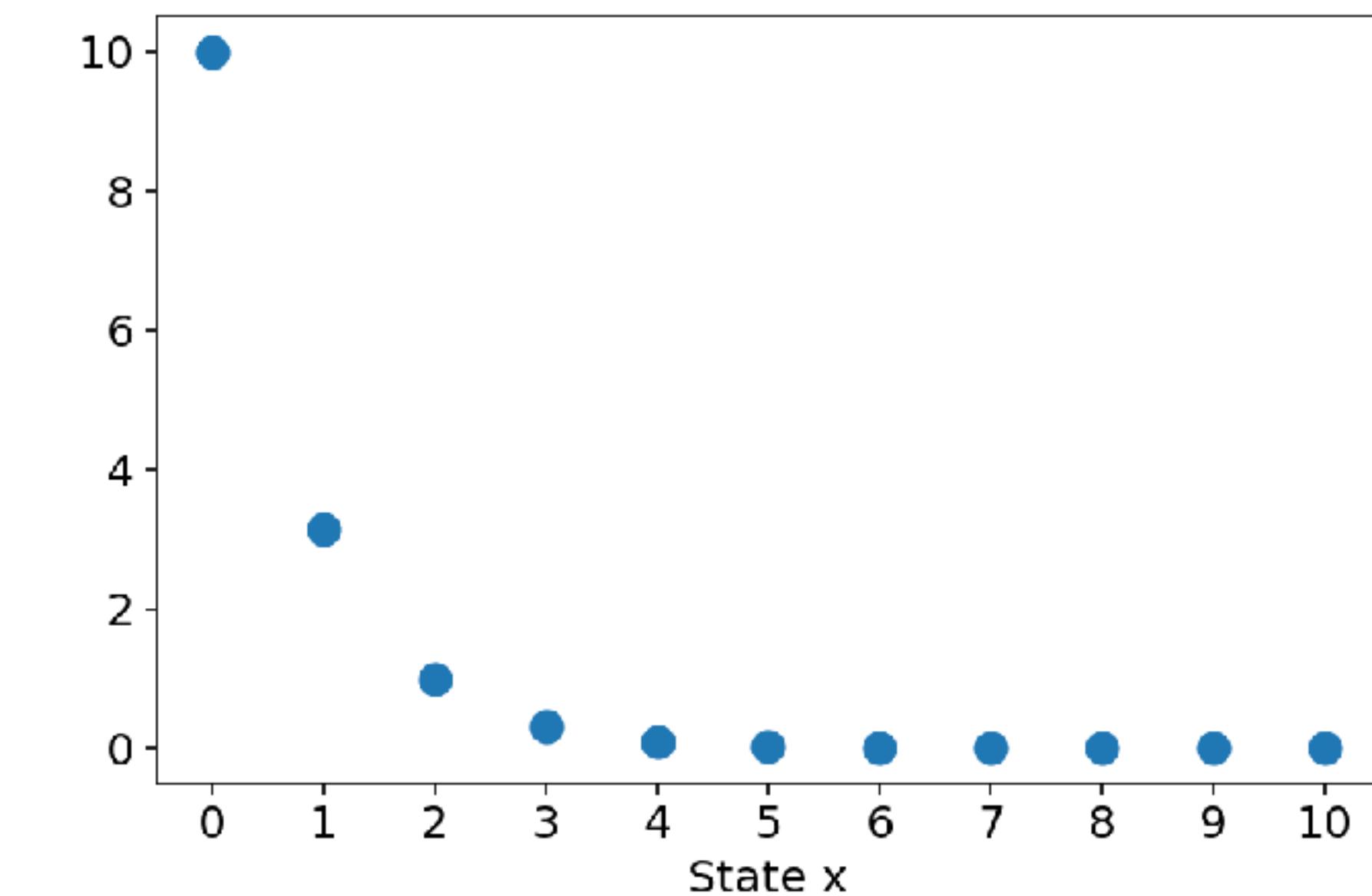
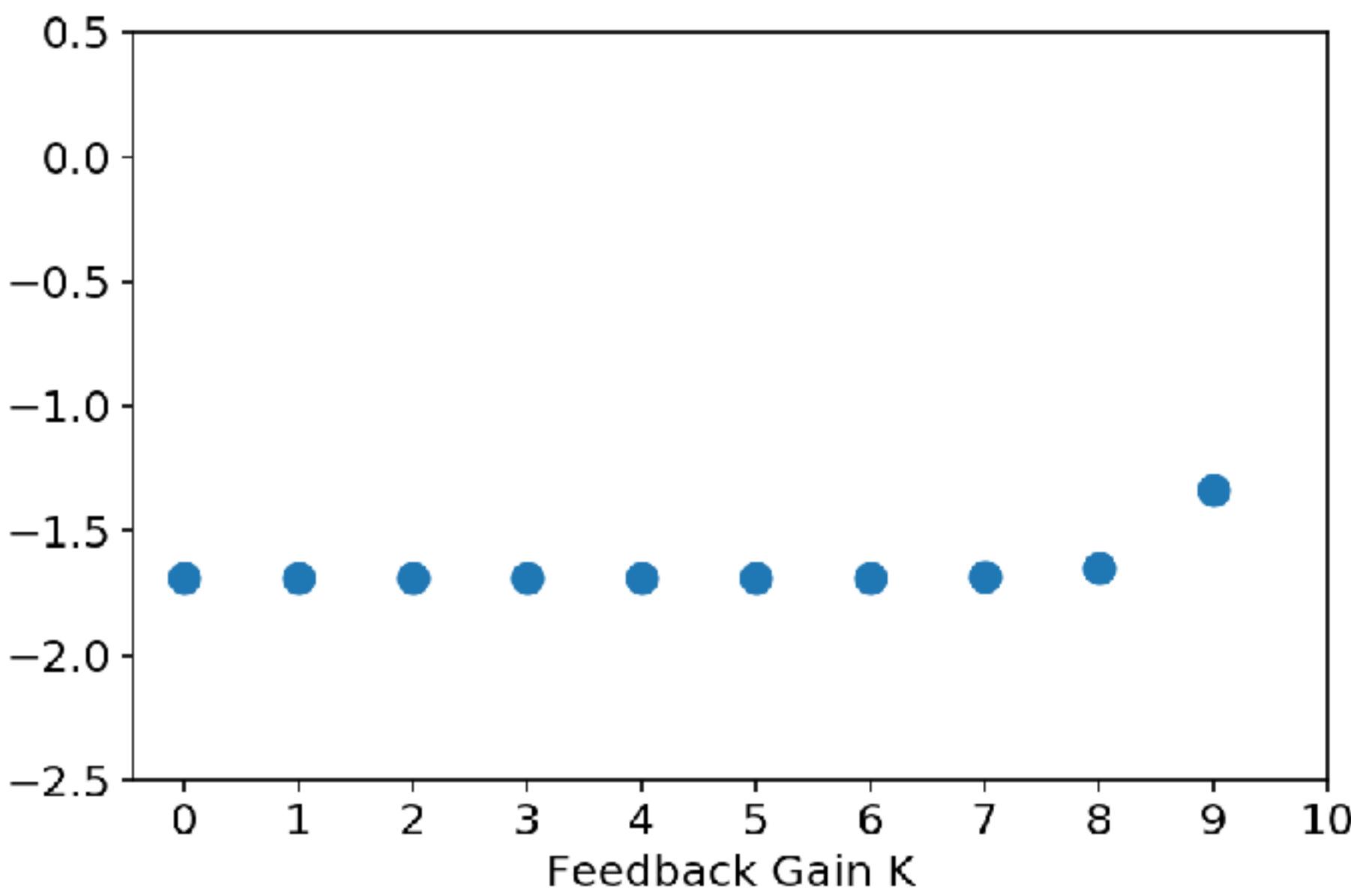
# Example

$$\min \sum_{i=0}^{N-1} (x_i^T Q x_i + u_i^T R u_i) + x_N^T Q x_N$$

**Subject to**  $x_{n+1} = 2x_n + u_n$

$$N = 10 \quad x_0 = 10$$

$$Q = 2 \quad R = 1$$



# Example

$$\min \sum_{i=0}^{N-1} (x_i^T Q x_i + u_i^T R u_i) + x_N^T Q x_N$$

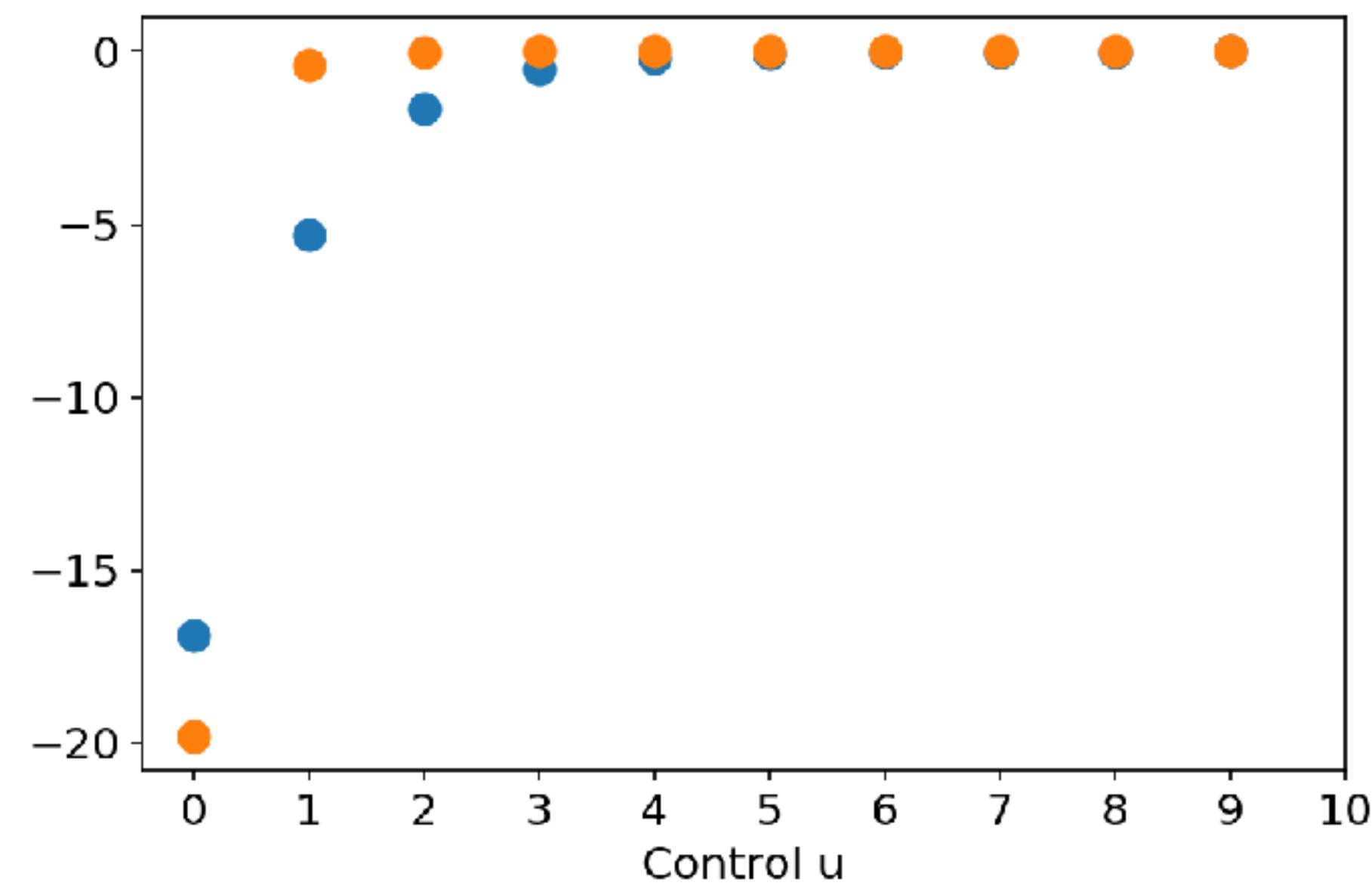
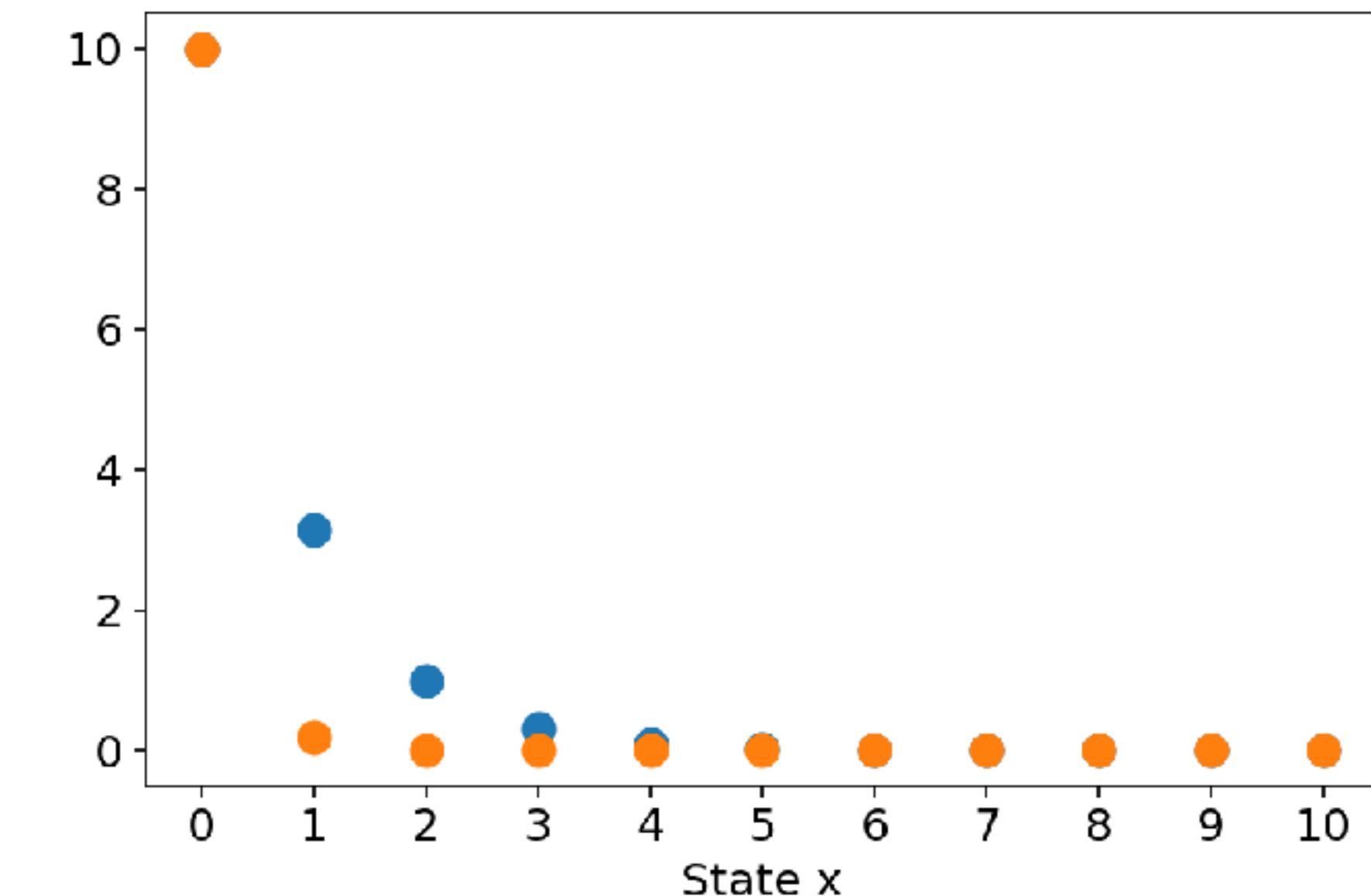
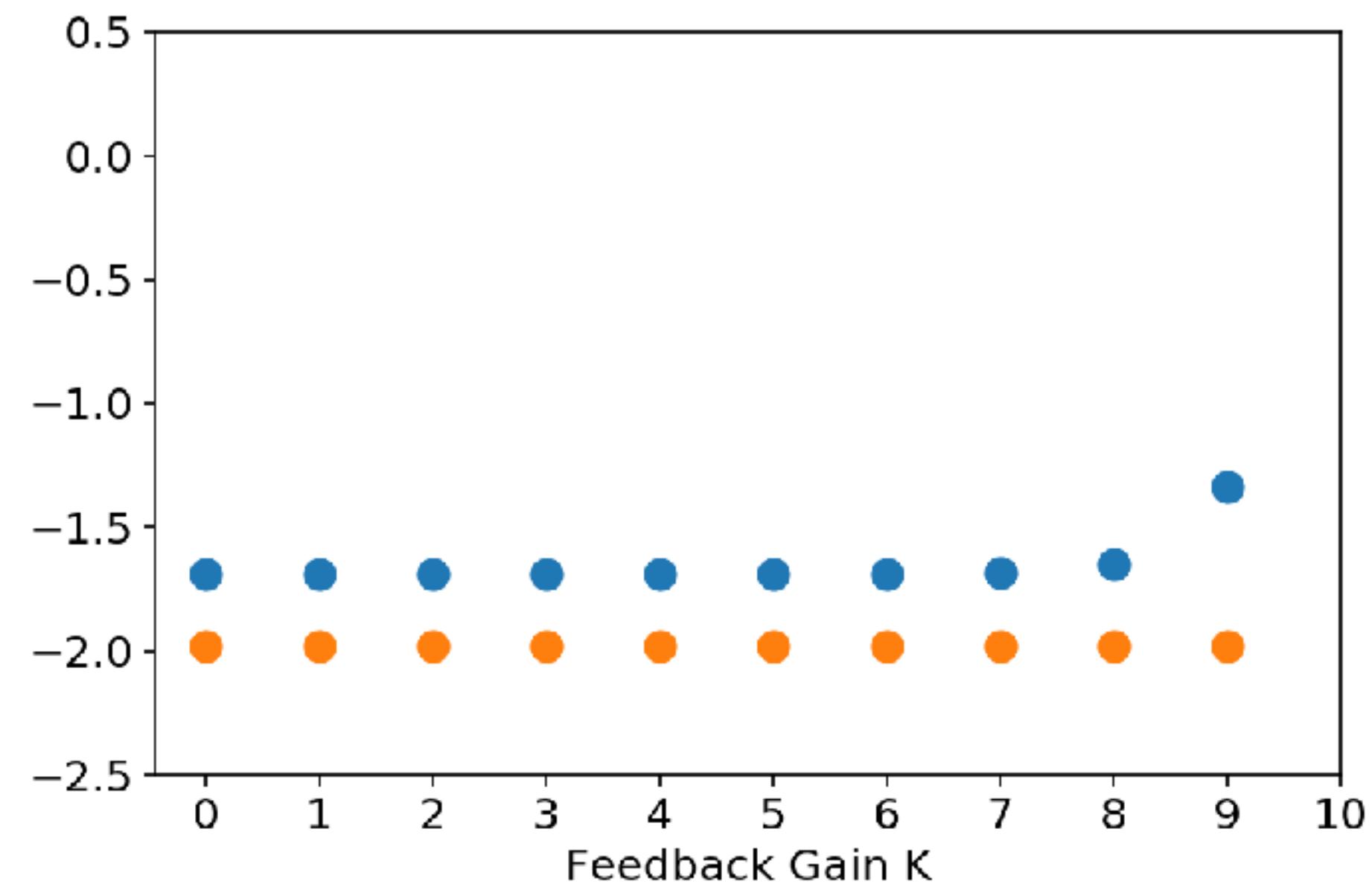
Subject to  $x_{n+1} = 2x_n + u_n$

$$N = 10 \quad x_0 = 10$$

$$Q = 2 \quad R = 1$$

$$Q = 100 \quad R = 1$$

increasing cost of state leads to higher gains, larger controls but faster stabilization



# Example

$$\min \sum_{i=0}^{N-1} (x_i^T Q x_i + u_i^T R u_i) + x_N^T Q x_N$$

Subject to  $x_{n+1} = 2x_n + u_n$

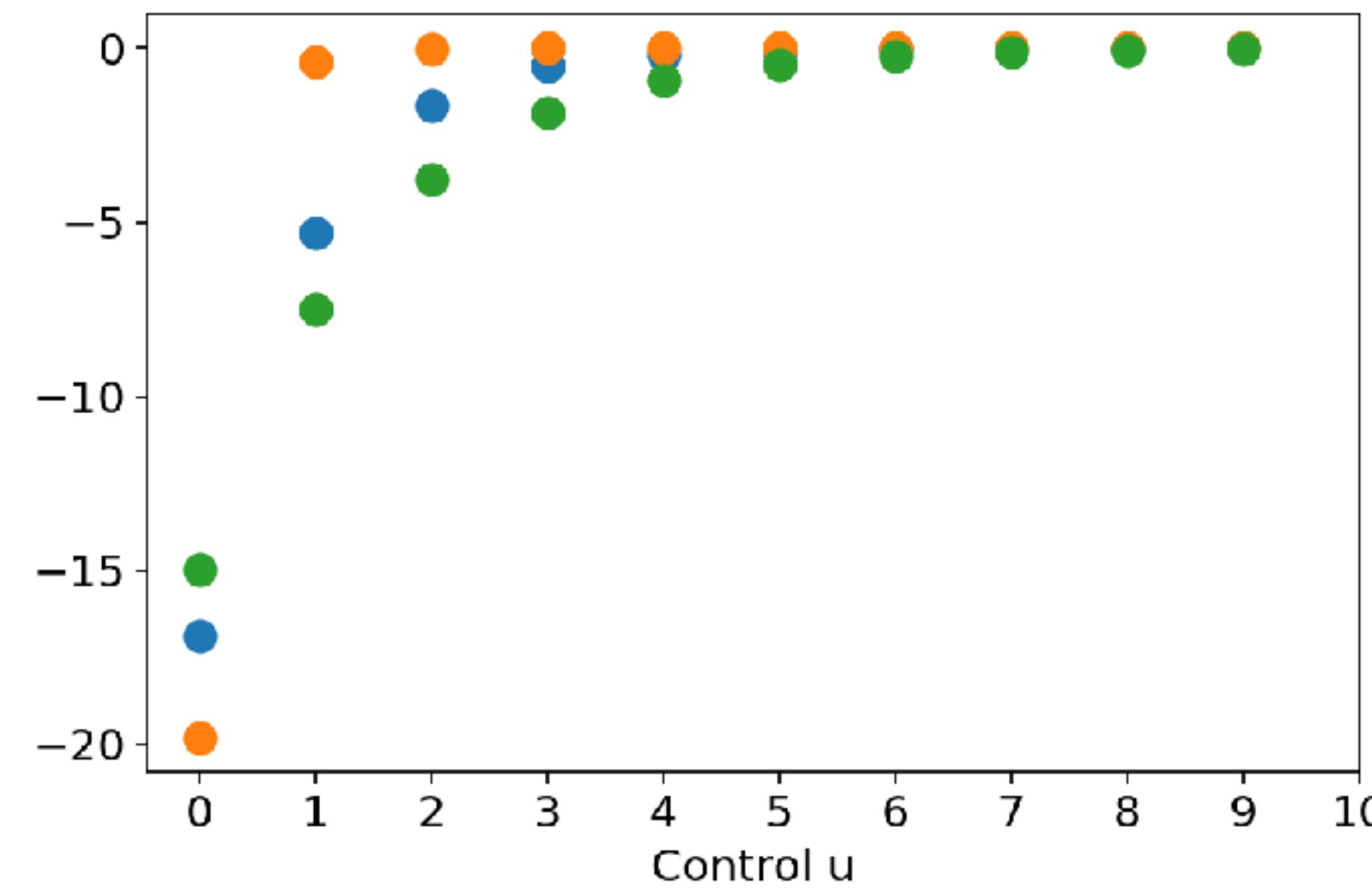
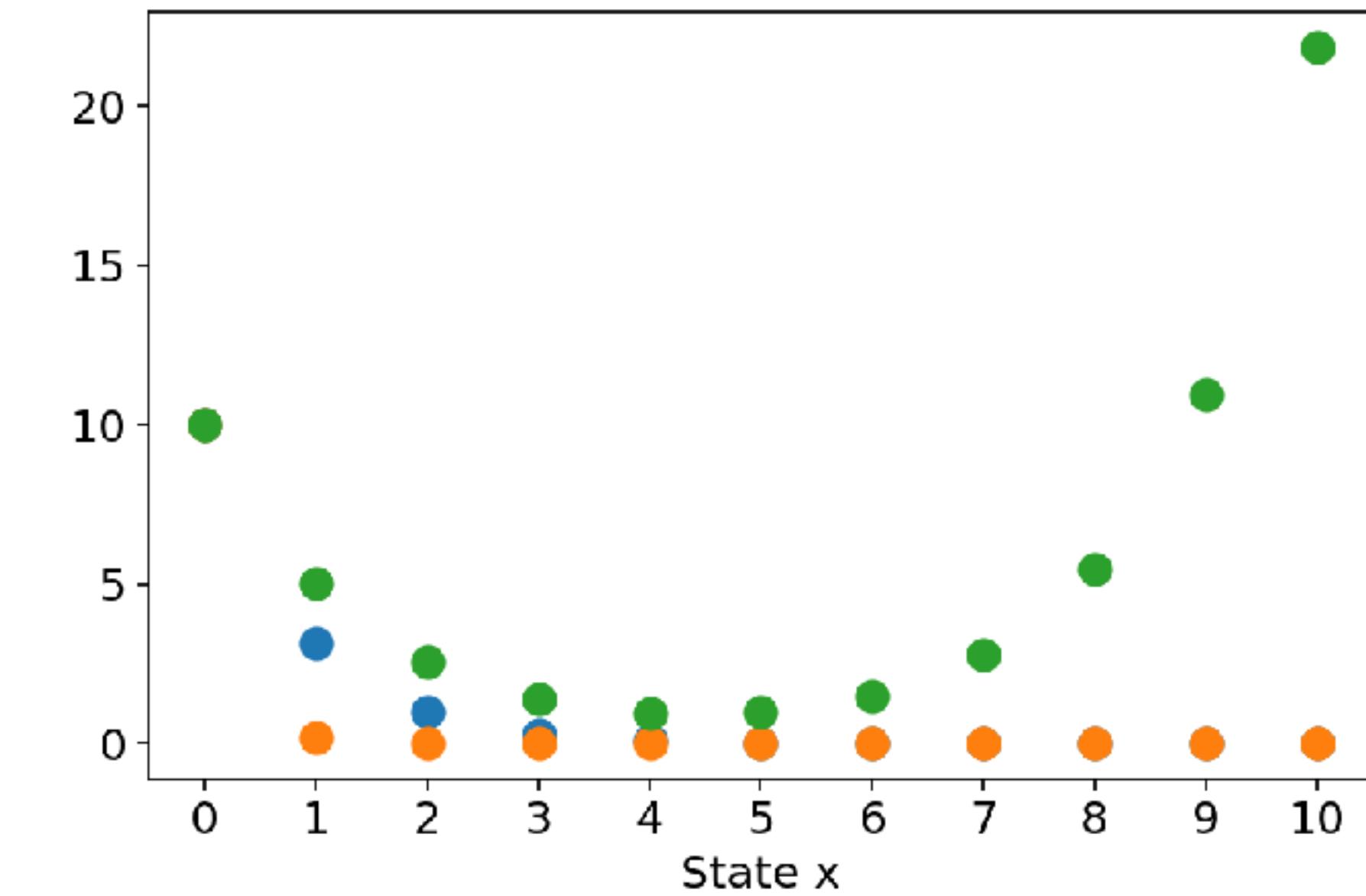
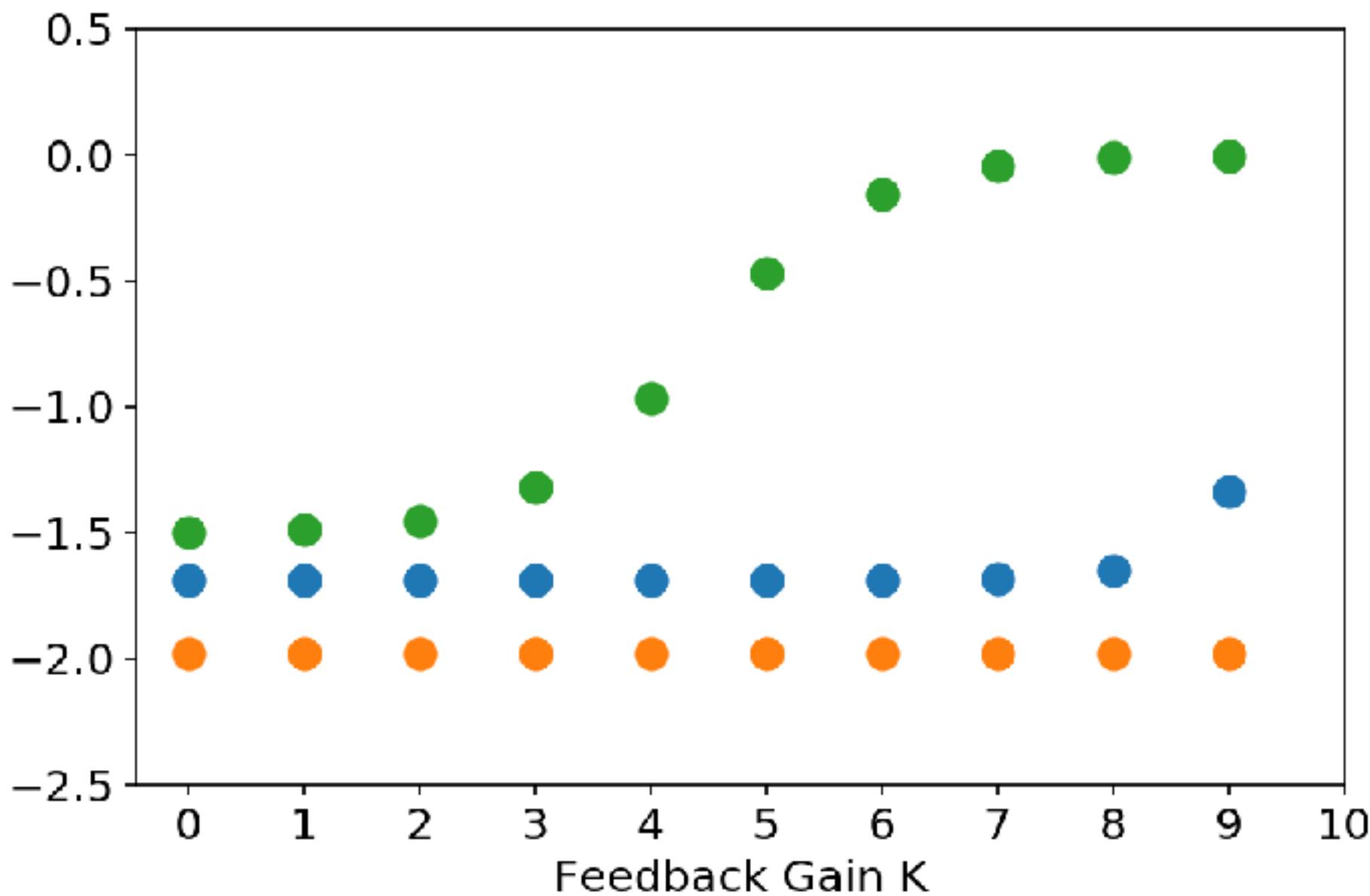
$$N = 10 \quad x_0 = 10$$

$$Q = 2 \quad R = 1$$

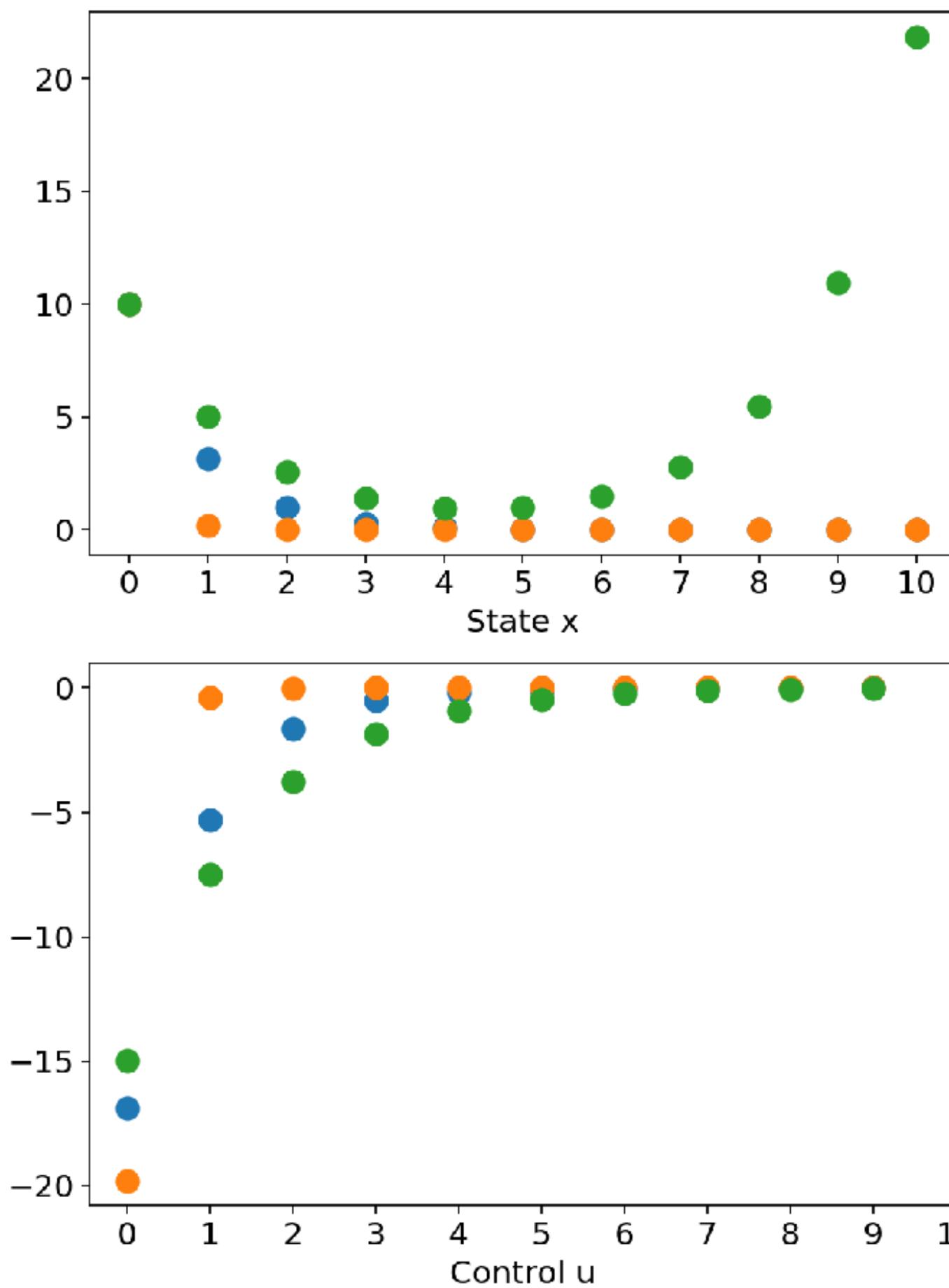
$$Q = 100 \quad R = 1$$

$$Q = 1 \quad R = 1000$$

increase control cost  
leads to smaller gains  
and control but  
stabilization does not  
occur for  $N=10$



# Effect of the horizon N



$N = 10$

$N = 20$

$$Q = 2 \ R = 1$$

$$Q = 100 \ R = 1$$

$$Q = 1 \ R = 1000$$

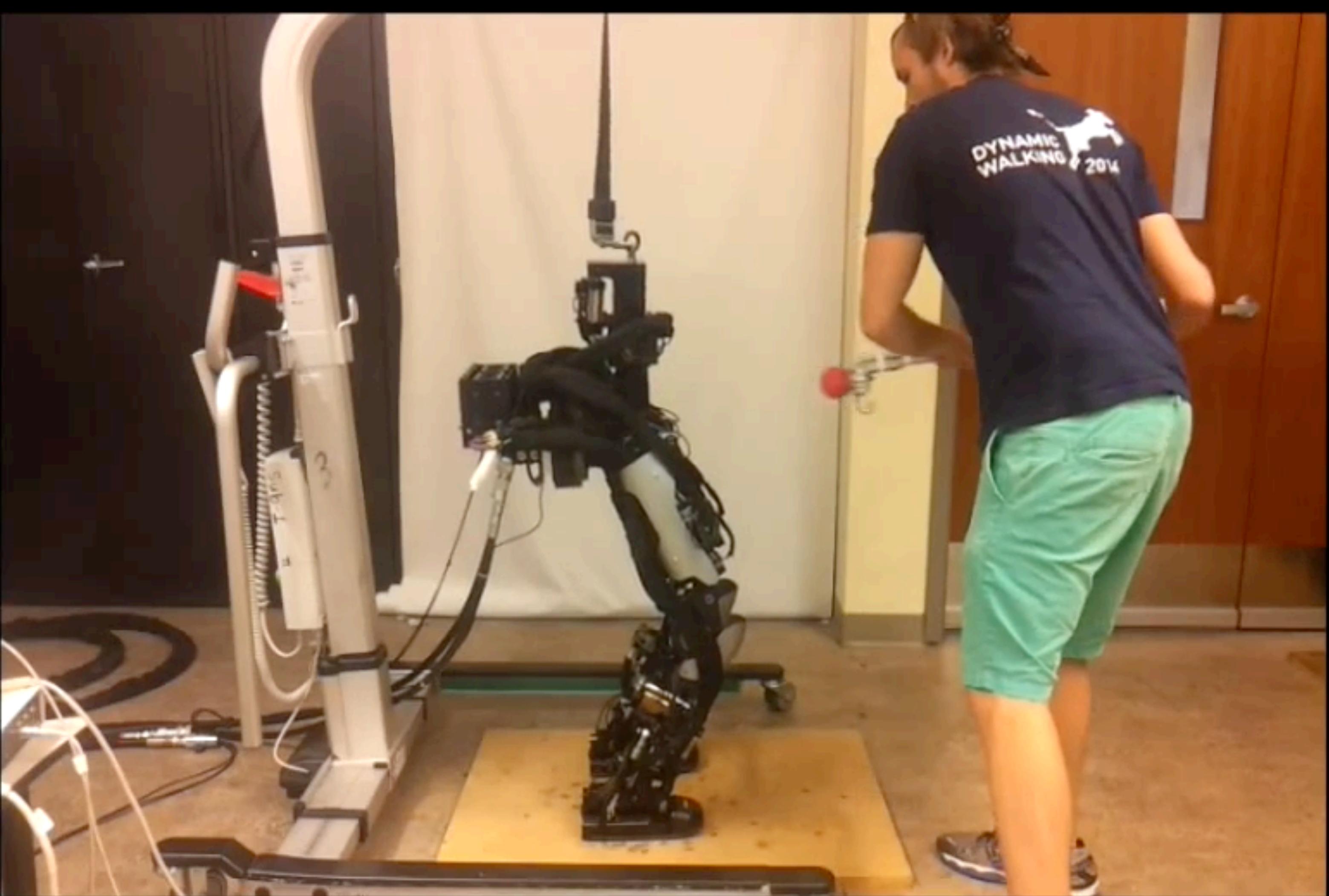
gains seem constant for  
early stages

increasing N seems to lead  
to more stable behavior  
even with low control

is it always true?

Impulse  
8.1 Ns

Peak Force  
287 N



**LQR Controller**

[Mason, Righetti and Schaal, 2014]

# More generic LQ problems

# Optimization of quadratic costs and linear constraints

Any optimization problem of the form

$$\begin{aligned} & \min_x \frac{1}{2} x^T P x + q^T x \\ & \text{subject to } Ax = b \end{aligned}$$

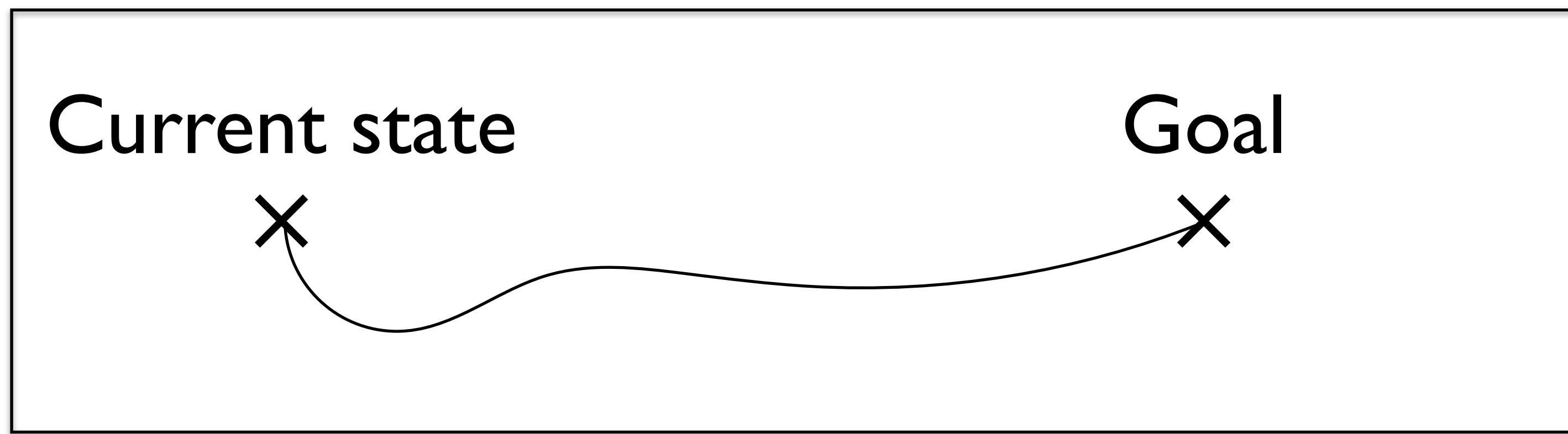
where  $P \geq 0$  is called a (convex) quadratic program (QP) with equality constraints.

The Lagrangian is  $L(x, \lambda) = \frac{1}{2} x^T P x + q^T x + \lambda^T (Ax + b)$

The KKT conditions are

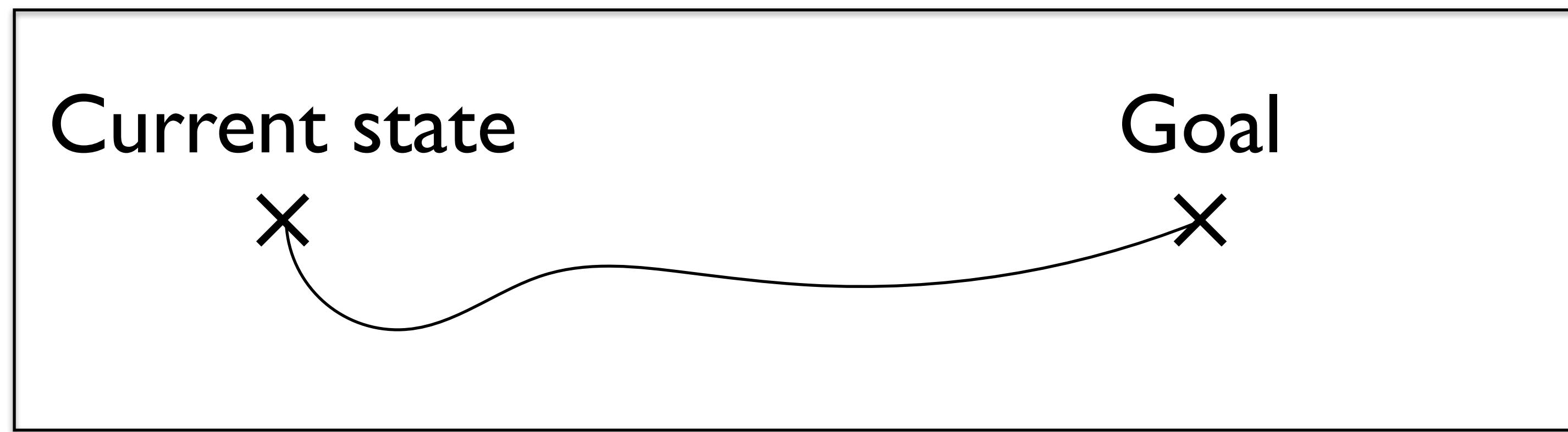
$$\begin{bmatrix} P & A^T \\ A & 0 \end{bmatrix} \begin{pmatrix} x \\ \lambda \end{pmatrix} = \begin{pmatrix} -q \\ b \end{pmatrix}$$

# Solving LQ problems with constraints



=> what about inequality constraints?

# Solving LQ problems with constraints



$$\min_{x_n, u_n} \frac{1}{2} \sum_{n=0}^{N-1} x_n^T Q x_n + u_n^T R u_n + x_N^T Q x_N$$

subject to  $x_{n+1} = Ax_n + Bu_n$

$$x_{min} \leq x_n \leq x_{max}$$

$$u_{min} \leq u_n \leq u_{max}$$

$$x_0 = x_{init}$$

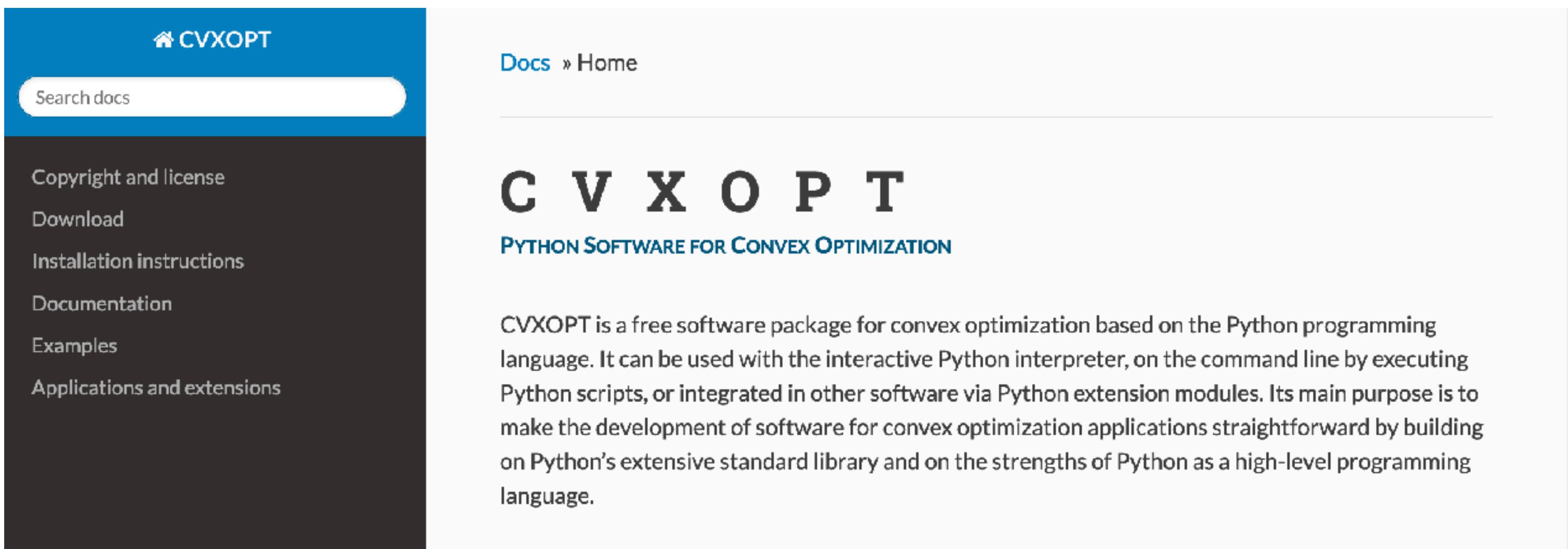
# Solving LQ problems with constraints

Any optimization problem of the form

$$\begin{aligned} & \min_x \frac{1}{2} x^T P x + q^T x \\ & \text{subject to } Ax = b \\ & \quad Gx \leq h \end{aligned}$$

where  $P \geq 0$  is called a (convex) quadratic program (QP).

# Using a QP solver



The screenshot shows the CVXOPT documentation homepage. The header includes a logo, a search bar, and navigation links for 'Docs' and 'Home'. The main content features the CVXOPT logo and a brief description of the software as a free package for convex optimization based on Python. A sidebar on the left contains links to 'Copyright and license', 'Download', 'Installation instructions', 'Documentation', 'Examples', and 'Applications and extensions'.

CVXOPT  
PYTHON SOFTWARE FOR CONVEX OPTIMIZATION

CVXOPT is a free software package for convex optimization based on the Python programming language. It can be used with the interactive Python interpreter, on the command line by executing Python scripts, or integrated in other software via Python extension modules. Its main purpose is to make the development of software for convex optimization applications straightforward by building on Python's extensive standard library and on the strengths of Python as a high-level programming language.

## Quadratic Programming

The function `qp` is an interface to `coneqp` for quadratic programs. It also provides the option of using the quadratic programming solver from MOSEK.

```
cvxopt.solvers.qp(P,q[,G,h[,A,b[,solver[,initvals]]]]) %
```

Solves the pair of primal and dual convex quadratic programs

$$\begin{aligned} & \text{minimize} && (1/2)x^T Px + q^T x \\ & \text{subject to} && Gx \leq h \\ & && Ax = b \end{aligned}$$

# Using a QP solver

## qpsolvers 4.3.3

`pip install qpsolvers`

Released: Aug 6, 2024

Quadratic programming solvers in Python with a unified API.

**Navigation**

- Project description**
- Release history
- Download files

---

**Verified details** ([What is this?](#))  
These details have been verified by PyPI

**Maintainers**

 [stephane-caron](#)

---

**Unverified details**

## Project description

### Quadratic Programming Solvers in Python

[build](#) [passing](#) [docs](#) [passing](#) [coverage](#) [83%](#) [conda-forge](#) [v4.3.3](#) [pypi](#) [v4.3.3](#) [downloads](#) [264k/month](#)

This library provides a one-stop shop `solve_qp` function to solve convex quadratic programs:

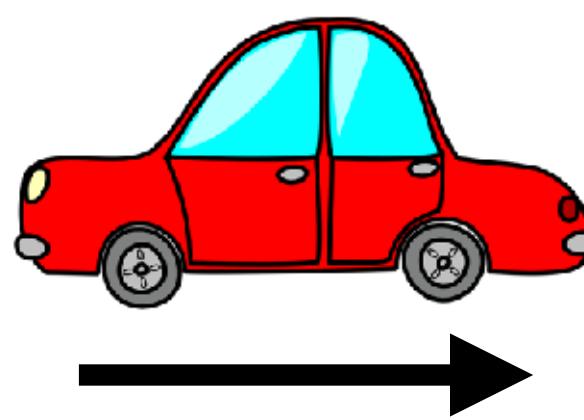
$$\underset{x}{\text{minimize}} \quad \frac{1}{2} x^T P x + q^T x \quad \text{subject to} \quad Gx \leq h \quad Ax = b \quad lb \leq x \leq ub$$

Vector inequalities apply coordinate by coordinate. The function returns the primal solution  $x^*$  found by the backend QP solver, or `None` in case of failure/unfeasible problem. All solvers require the problem to be convex, meaning the matrix  $P$  should be [positive semi-definite](#). Some solvers further require the problem to be strictly convex, meaning  $P$  should be positive definite.

**Dual multipliers:** there is also a `solve_problem` function that returns not only the primal solution, but also its dual multipliers and all other relevant quantities computed by the backend solver.

Solvers				
Solver	Keyword	Algorithm	API	License
<a href="#">Clarabel</a>	<code>clarabel</code>	Interior point	Sparse	Apache-2.0
<a href="#">CVXOPT</a>	<code>cvxopt</code>	Interior point	Dense	GPL-3.0
<a href="#">DAQP</a>	<code>daqp</code>	Active set	Dense	MIT
<a href="#">ECOS</a>	<code>ecos</code>	Interior point	Sparse	GPL-3.0
<a href="#">Gurobi</a>	<code>gurobi</code>	Interior point	Sparse	Commercial
<a href="#">HiGHS</a>	<code>highs</code>	Active set	Sparse	MIT
<a href="#">HPIPM</a>	<code>hpipm</code>	Interior point	Dense	BSD-2-Clause
<a href="#">MOSEK</a>	<code>mosek</code>	Interior point	Sparse	Commercial
<a href="#">NPPro</a>	<code>nppro</code>	Active set	Dense	Commercial
<a href="#">OSQP</a>	<code>osqp</code>	Augmented Lagrangian	Sparse	Apache-2.0
<a href="#">PIQP</a>	<code>piqp</code>	Proximal Interior Point	Dense & Sparse	BSD-2-Clause
<a href="#">ProxQP</a>	<code>proxqp</code>	Augmented Lagrangian	Dense & Sparse	BSD-2-Clause
<a href="#">QPALM</a>	<code>qpalm</code>	Augmented Lagrangian	Sparse	LGPL-3.0
<a href="#">qpOASES</a>	<code>qpoases</code>	Active set	Dense	LGPL-2.1
<a href="#">qpSWIFT</a>	<code>qpswift</code>	Interior point	Sparse	GPL-3.0
<a href="#">quadprog</a>	<code>quadprog</code>	Active set	Dense	GPL-2.0
<a href="#">SCS</a>	<code>scs</code>	Augmented Lagrangian	Sparse	MIT

# Using a QP solver

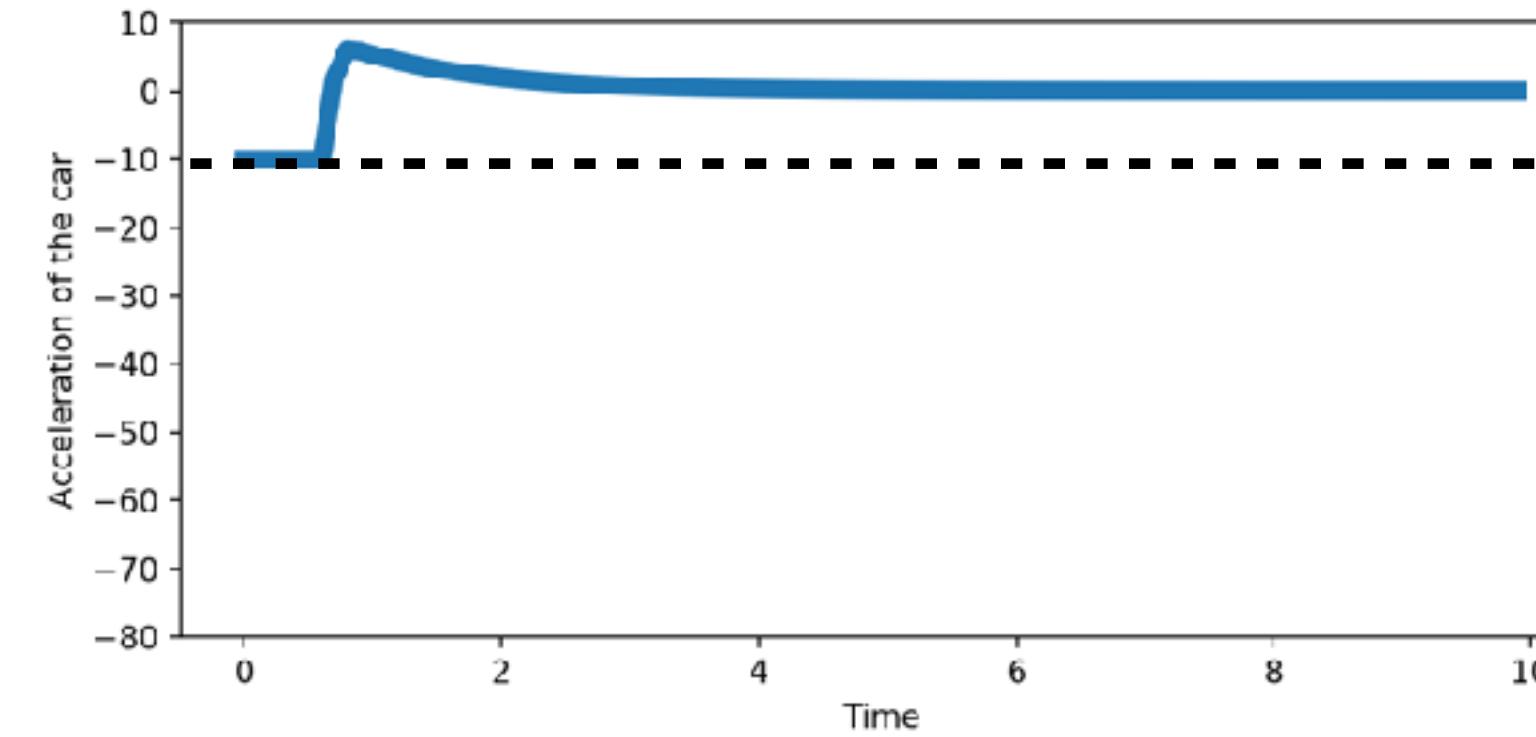
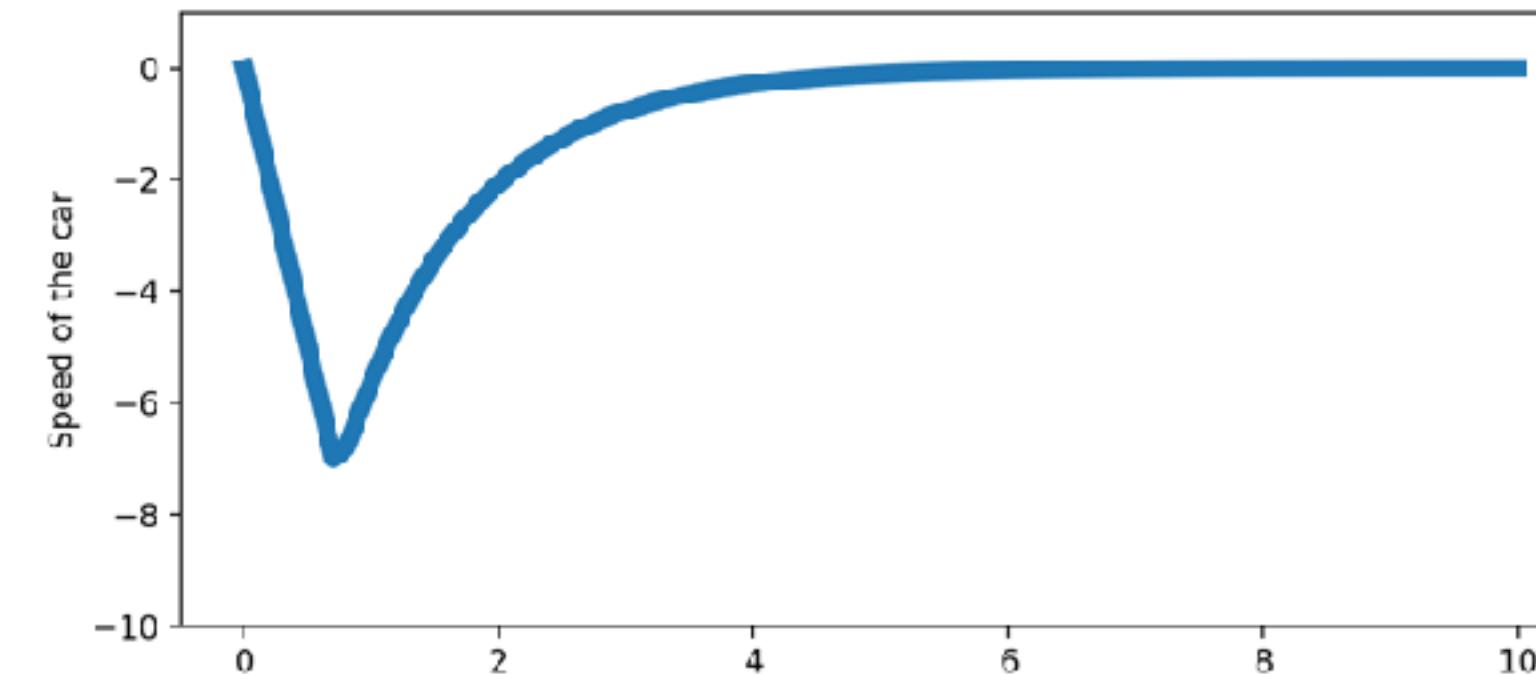
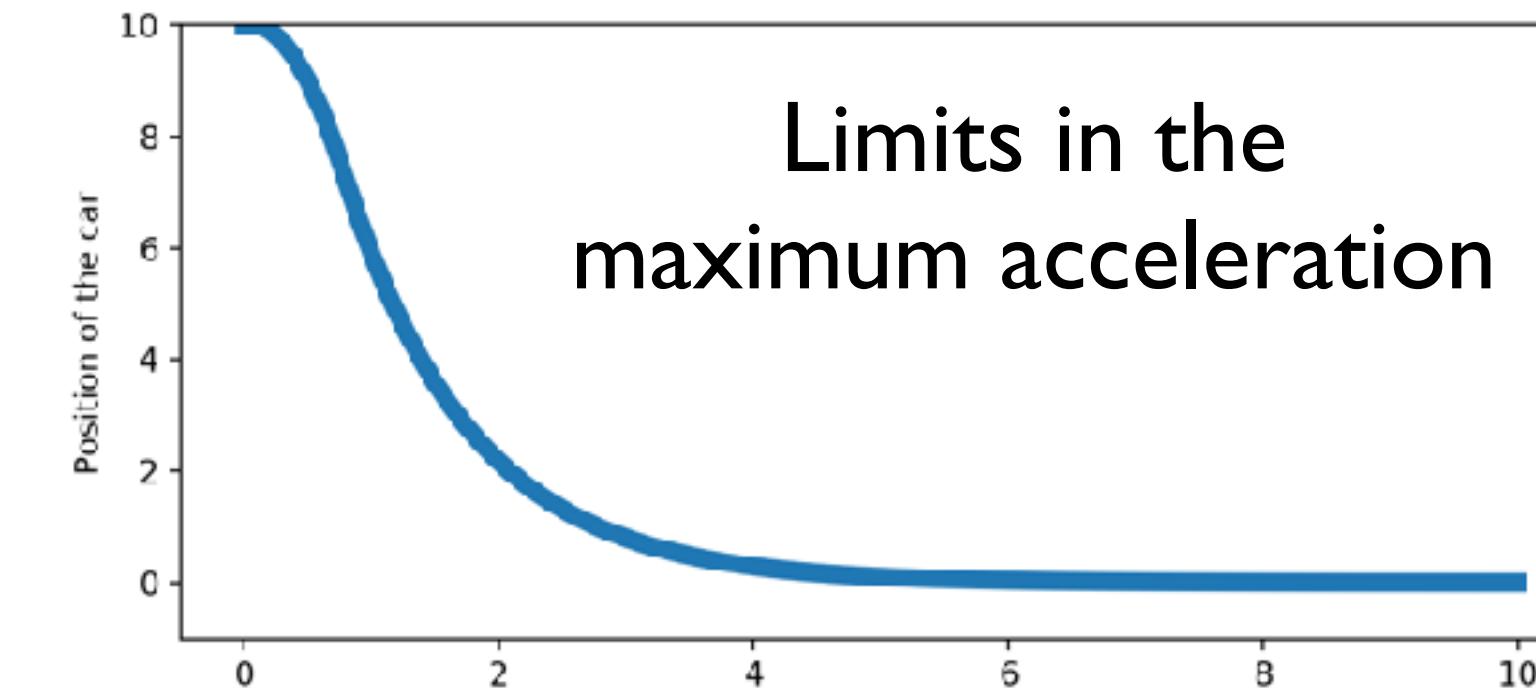
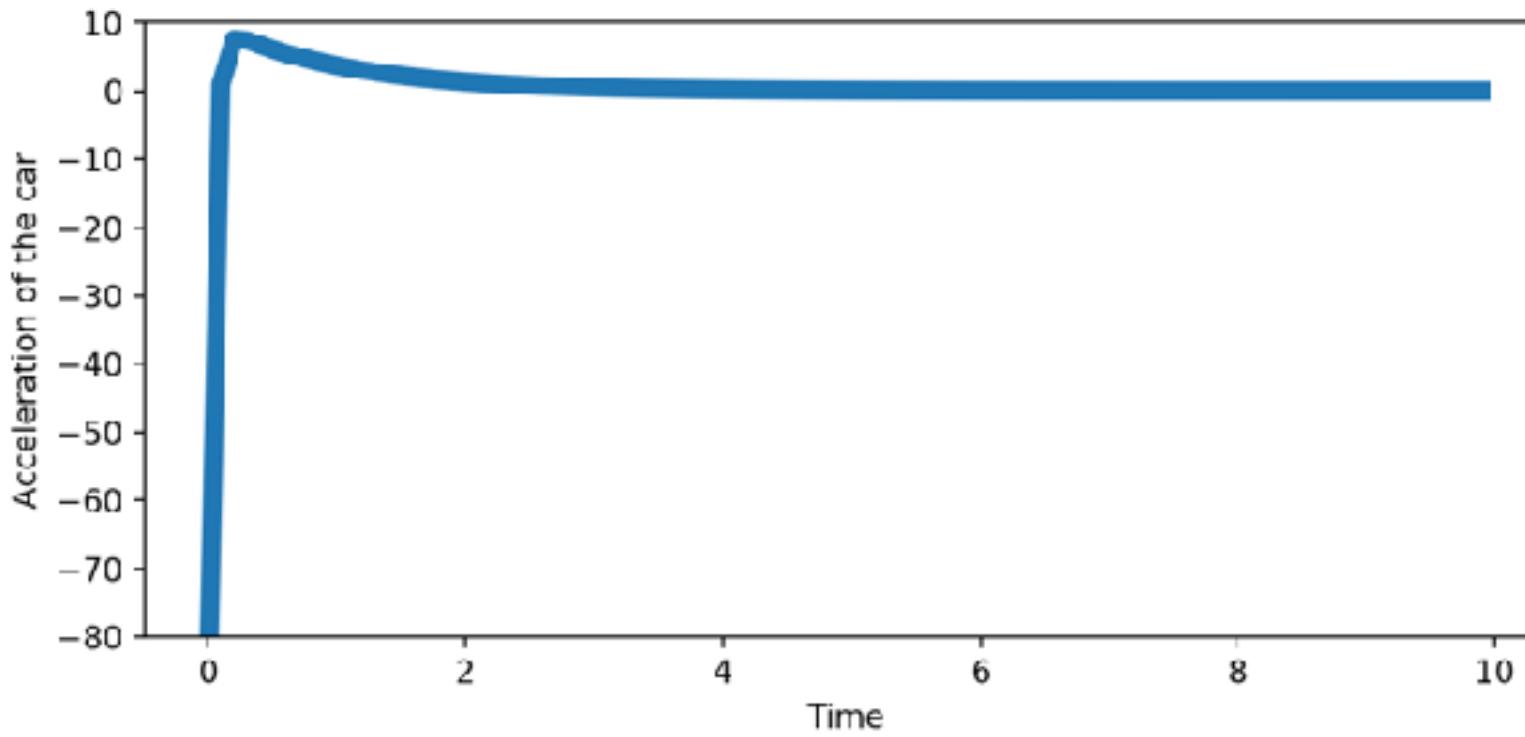
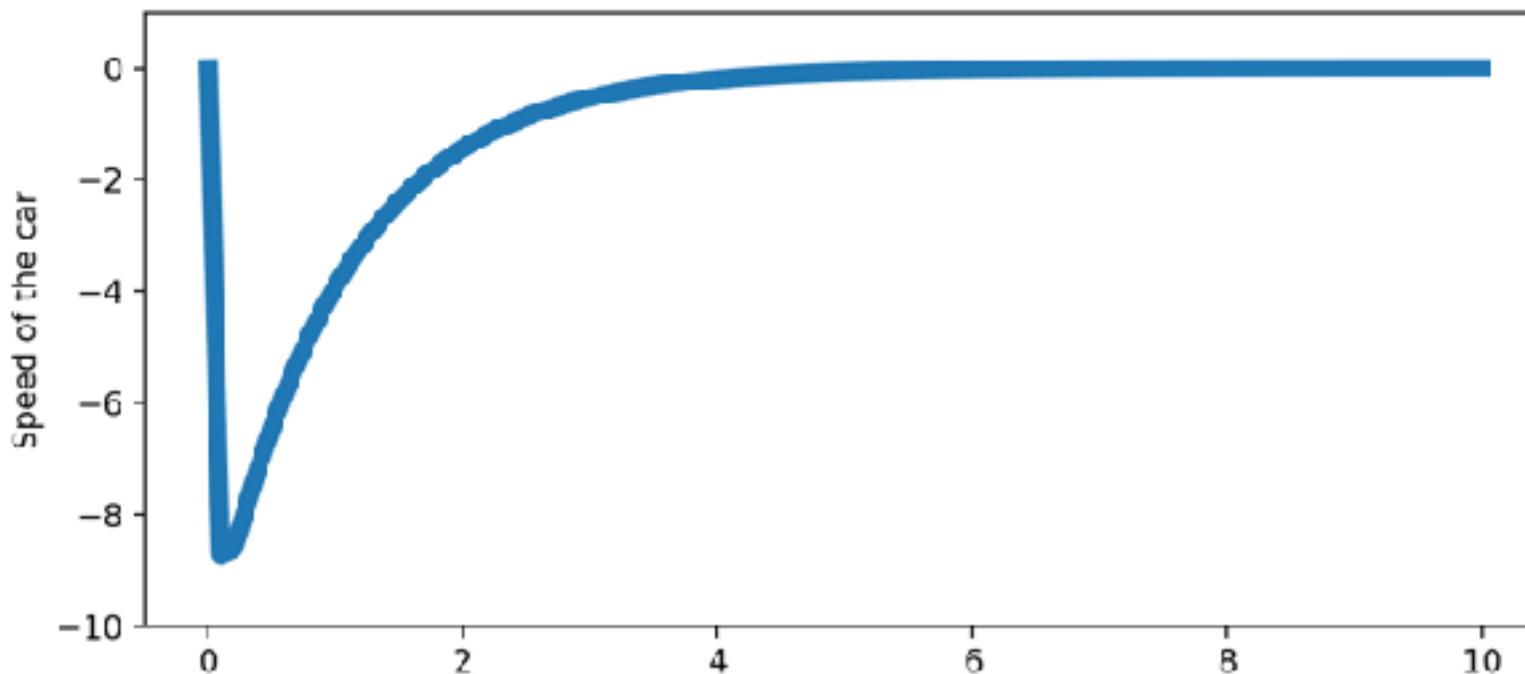
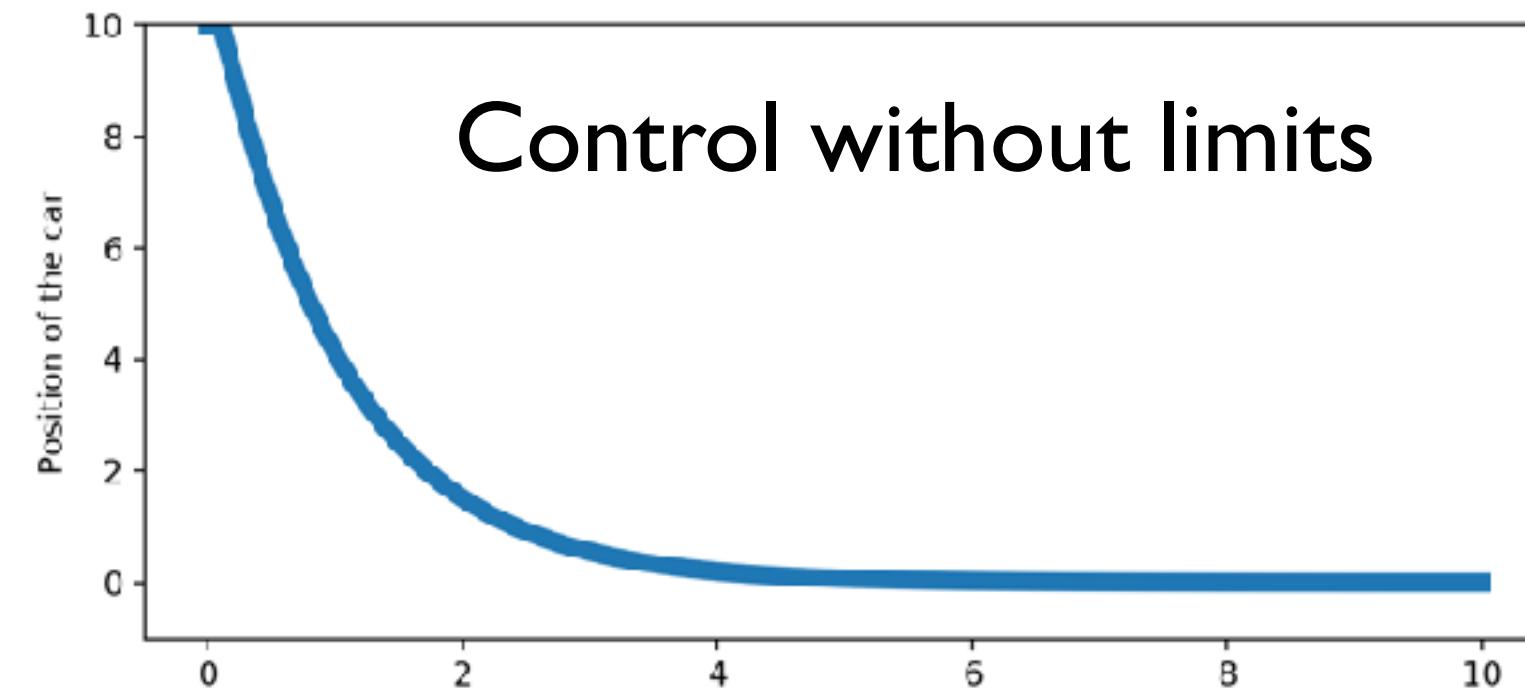


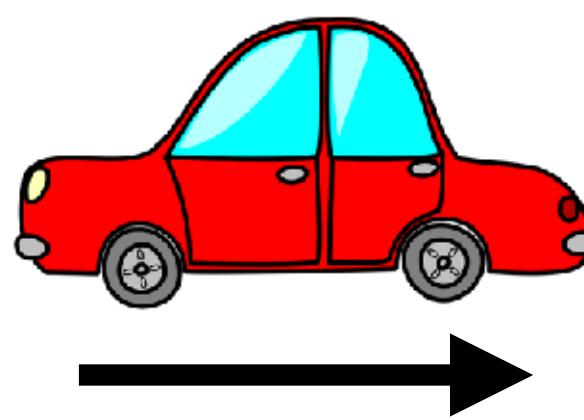
**Goal**  
**X**

$$x_{n+1} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} x_n + \begin{bmatrix} 0 \\ \Delta t \end{bmatrix} u_n$$

$$Q = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix}$$

$$R = [0.1]$$



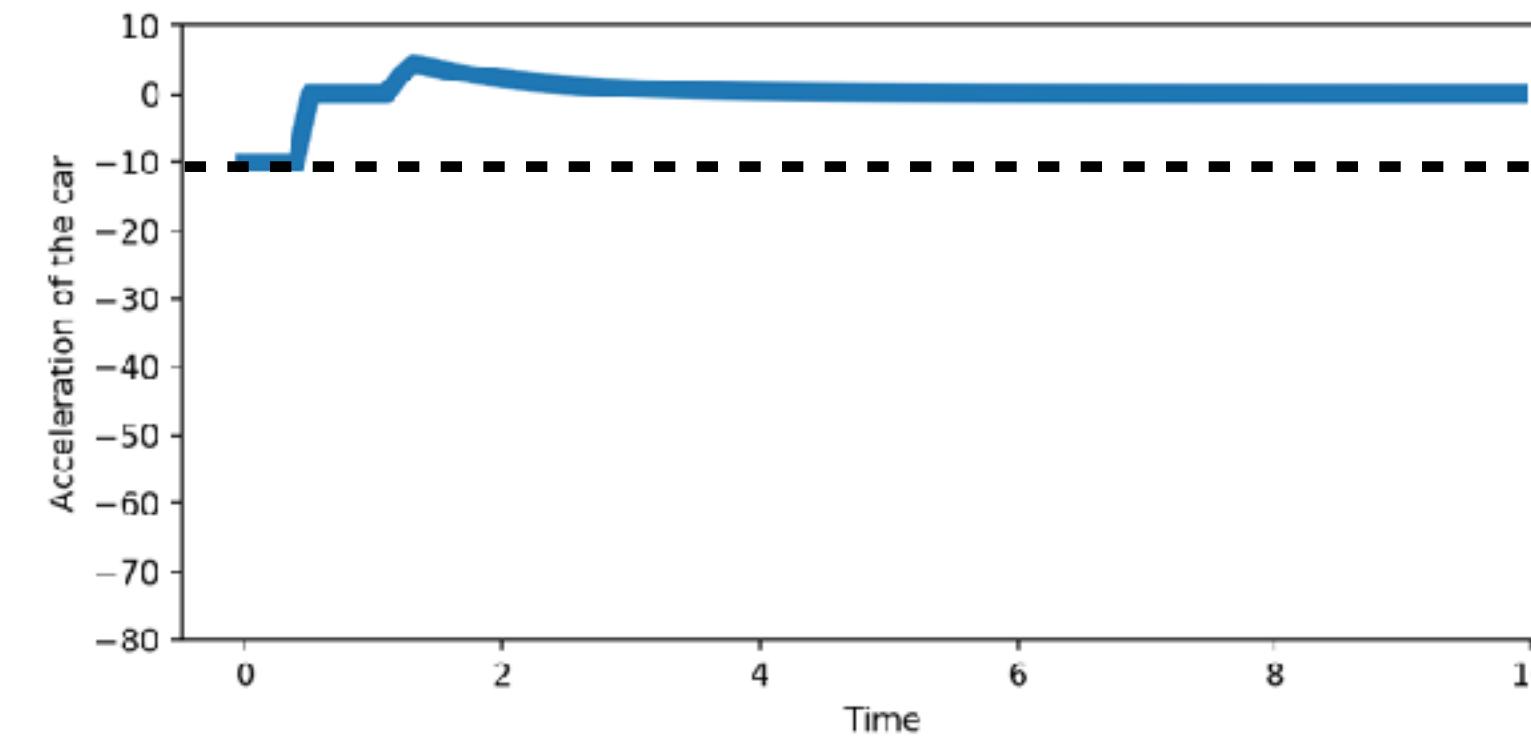
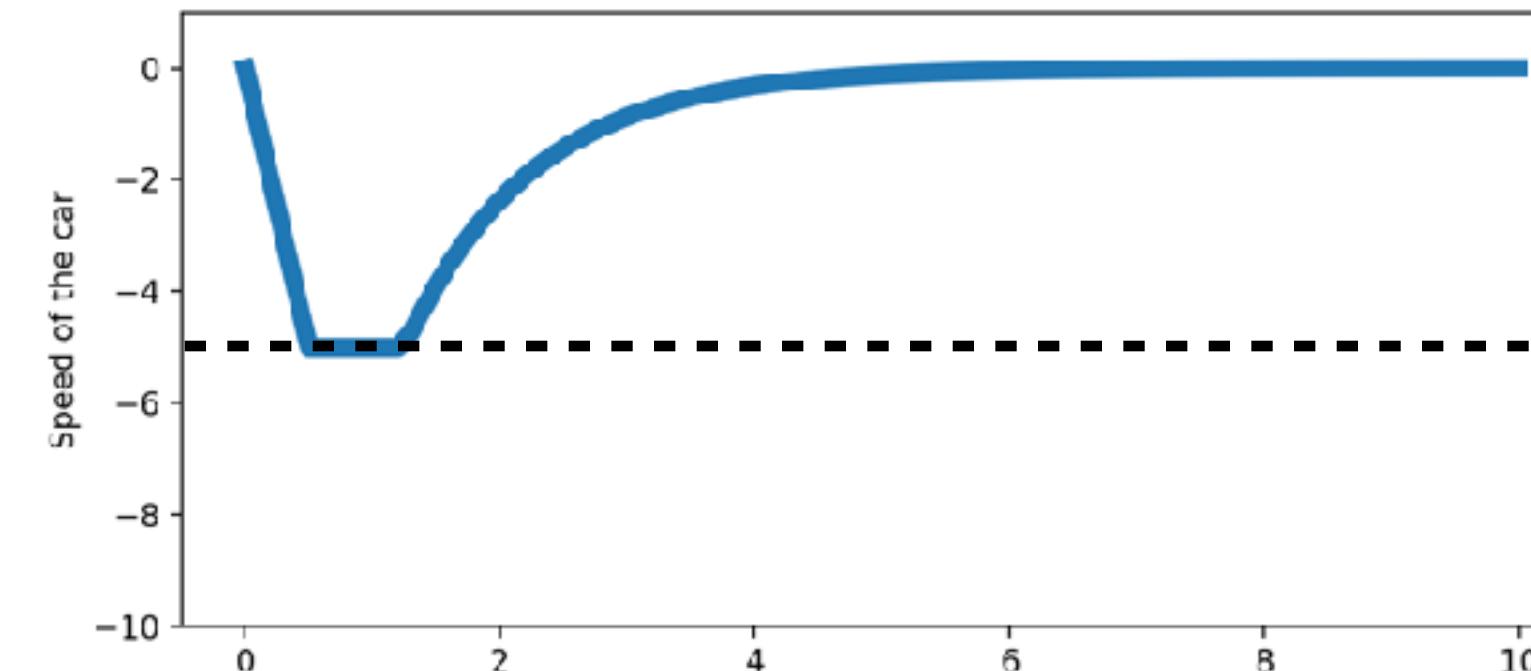
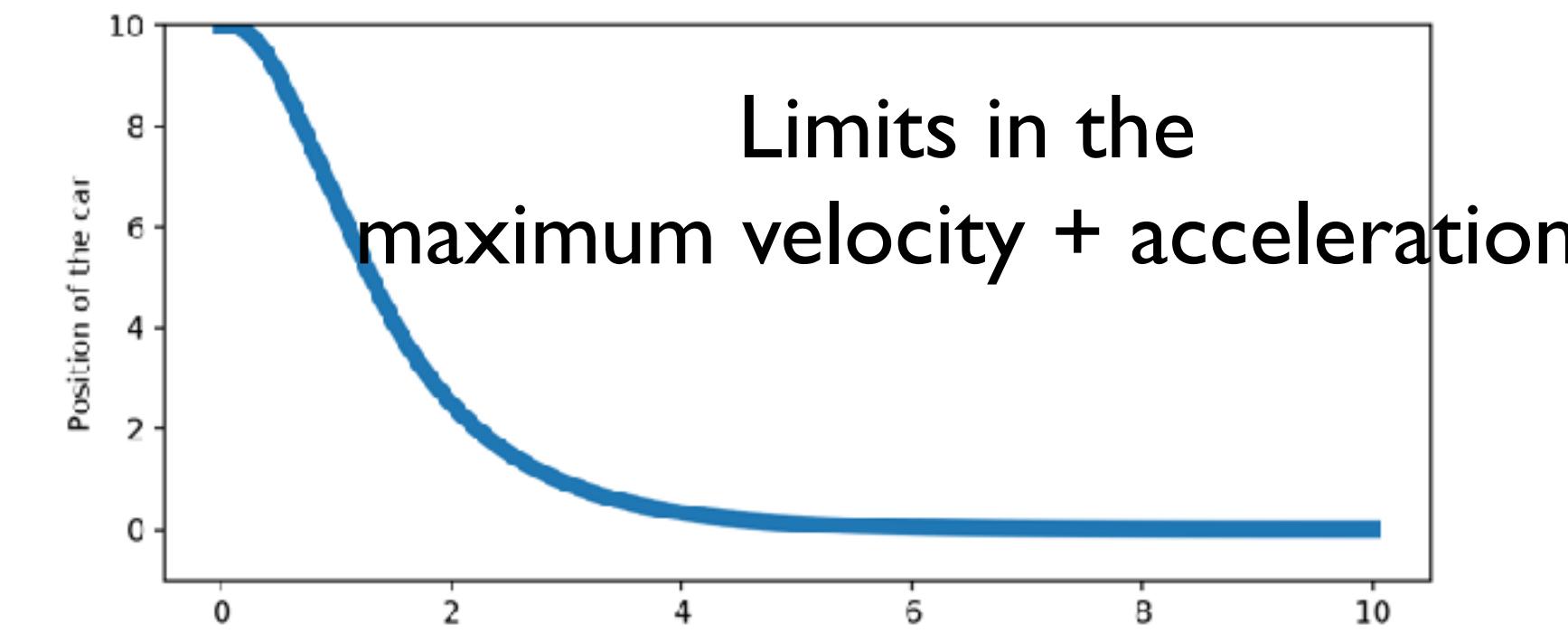
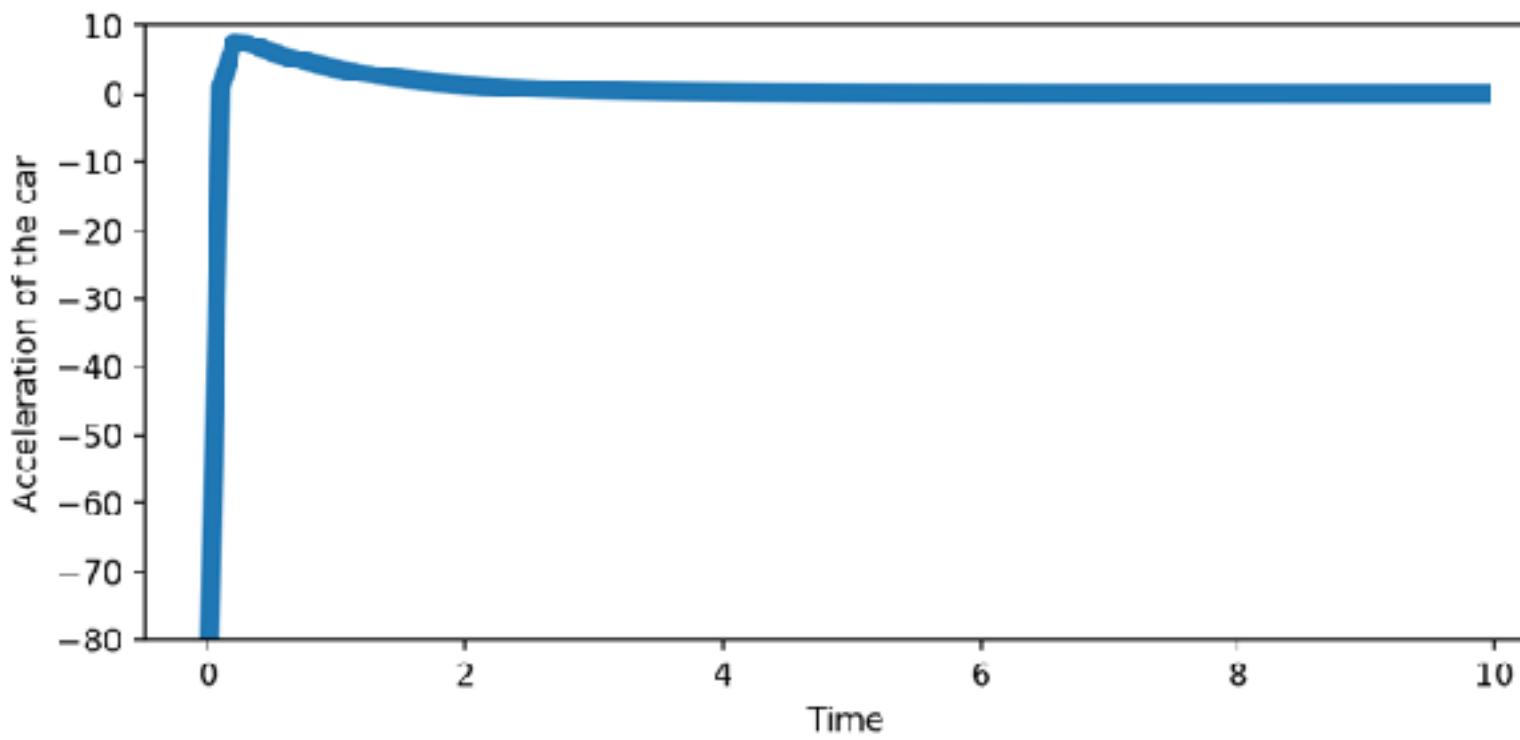
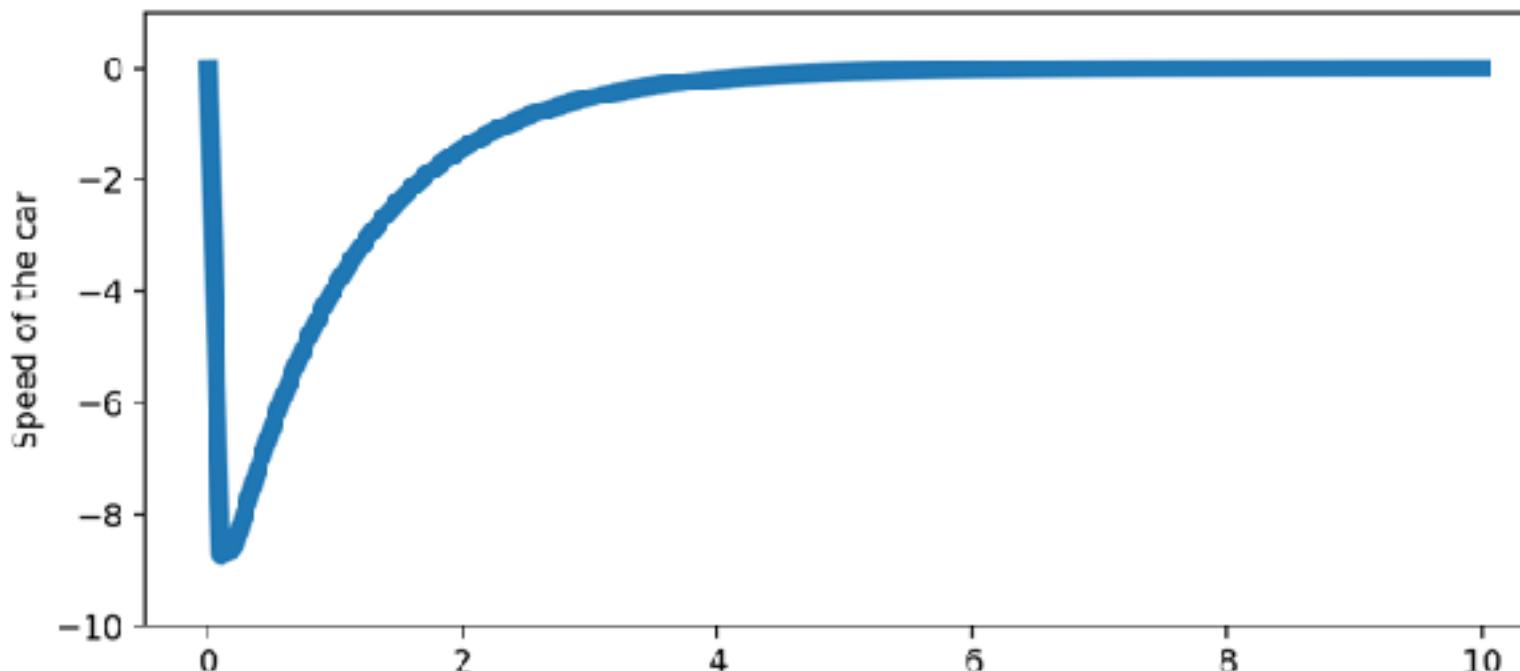
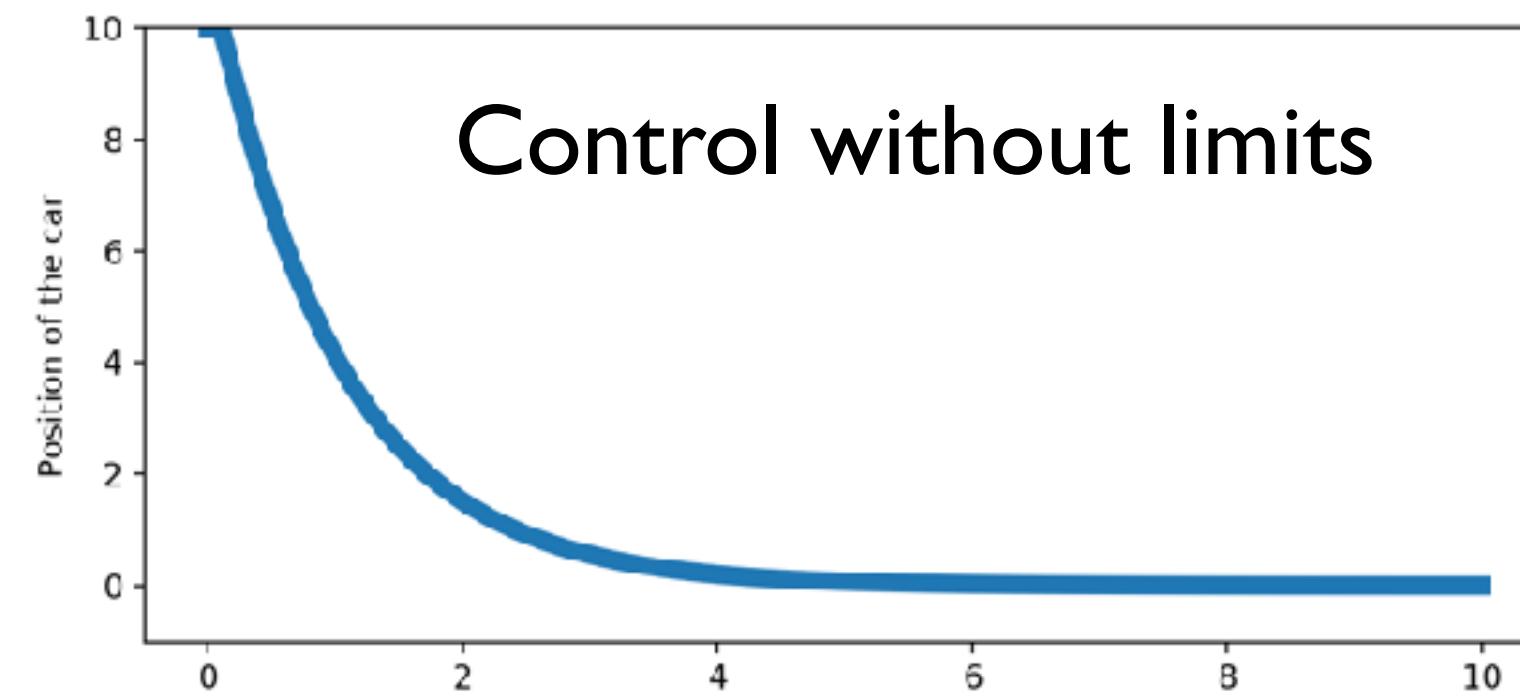


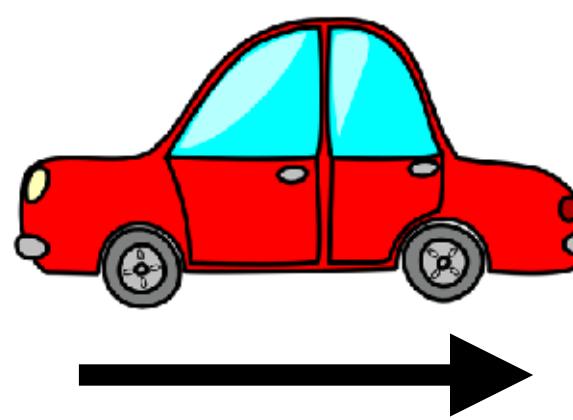
**Goal**  
**X**

$$x_{n+1} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} x_n + \begin{bmatrix} 0 \\ \Delta t \end{bmatrix} u_n$$

$$Q = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix}$$

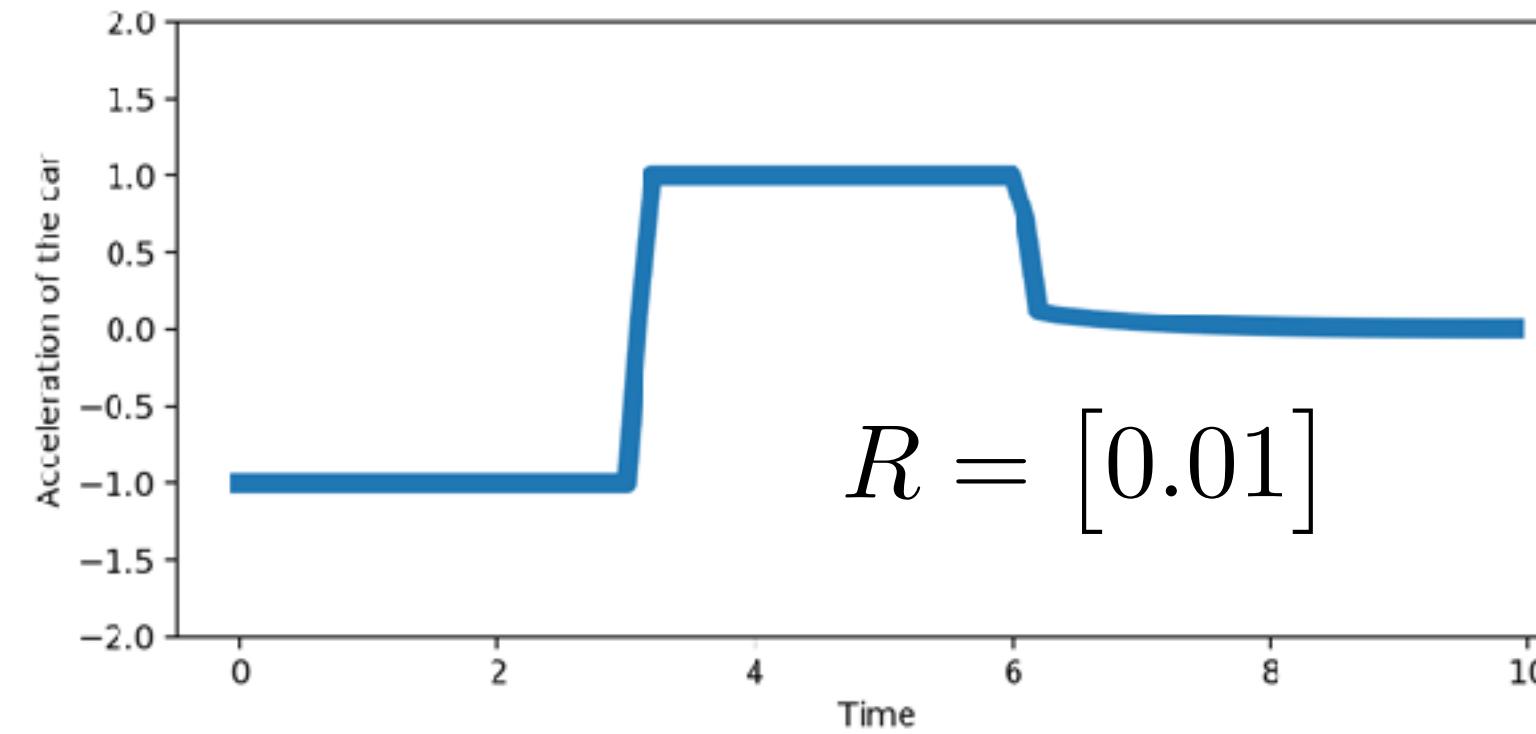
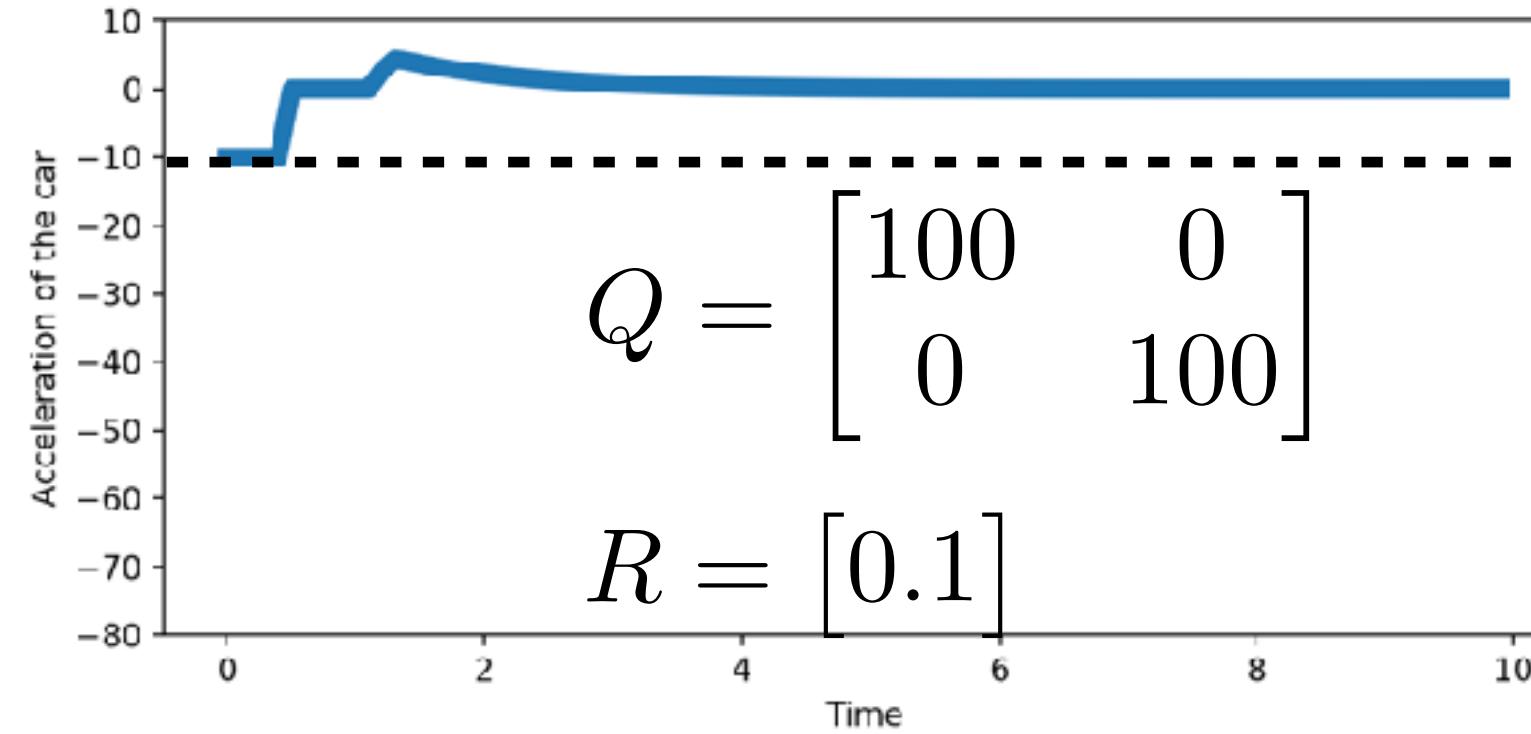
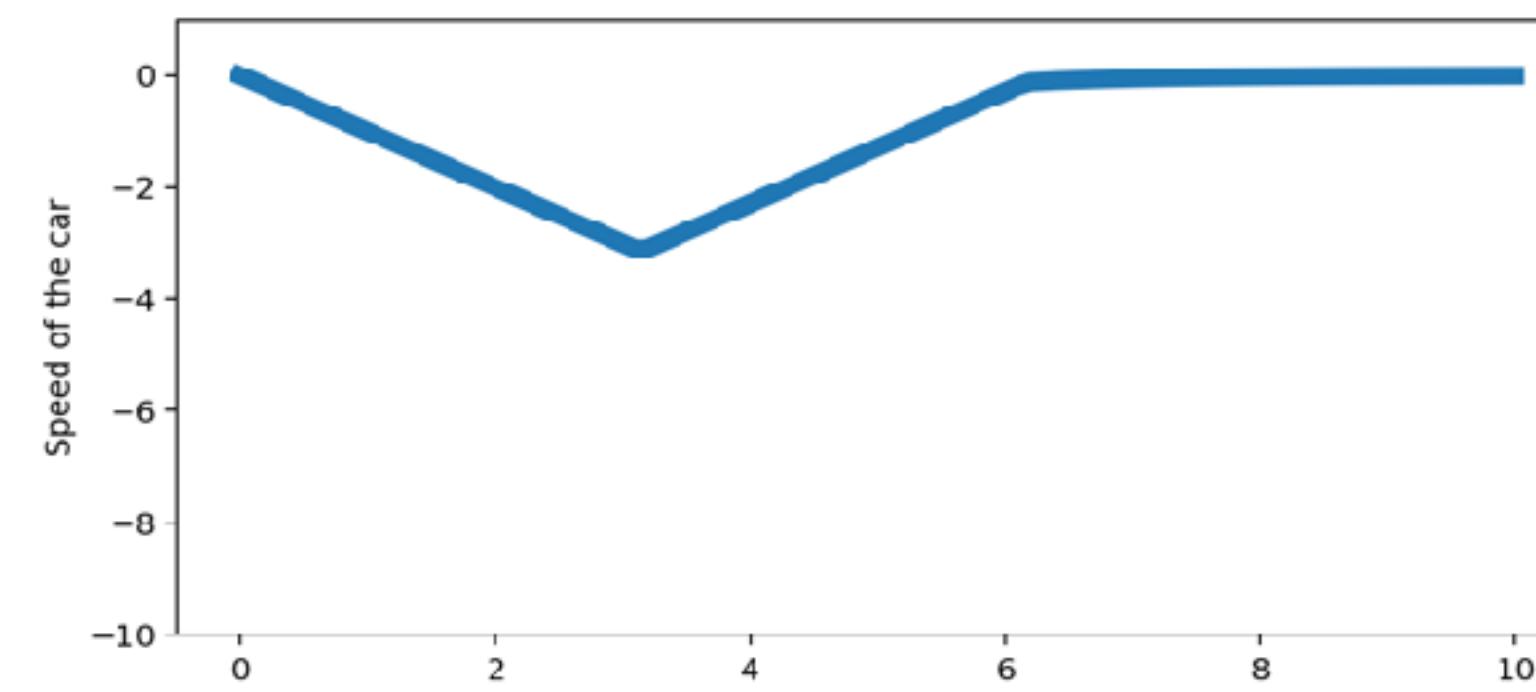
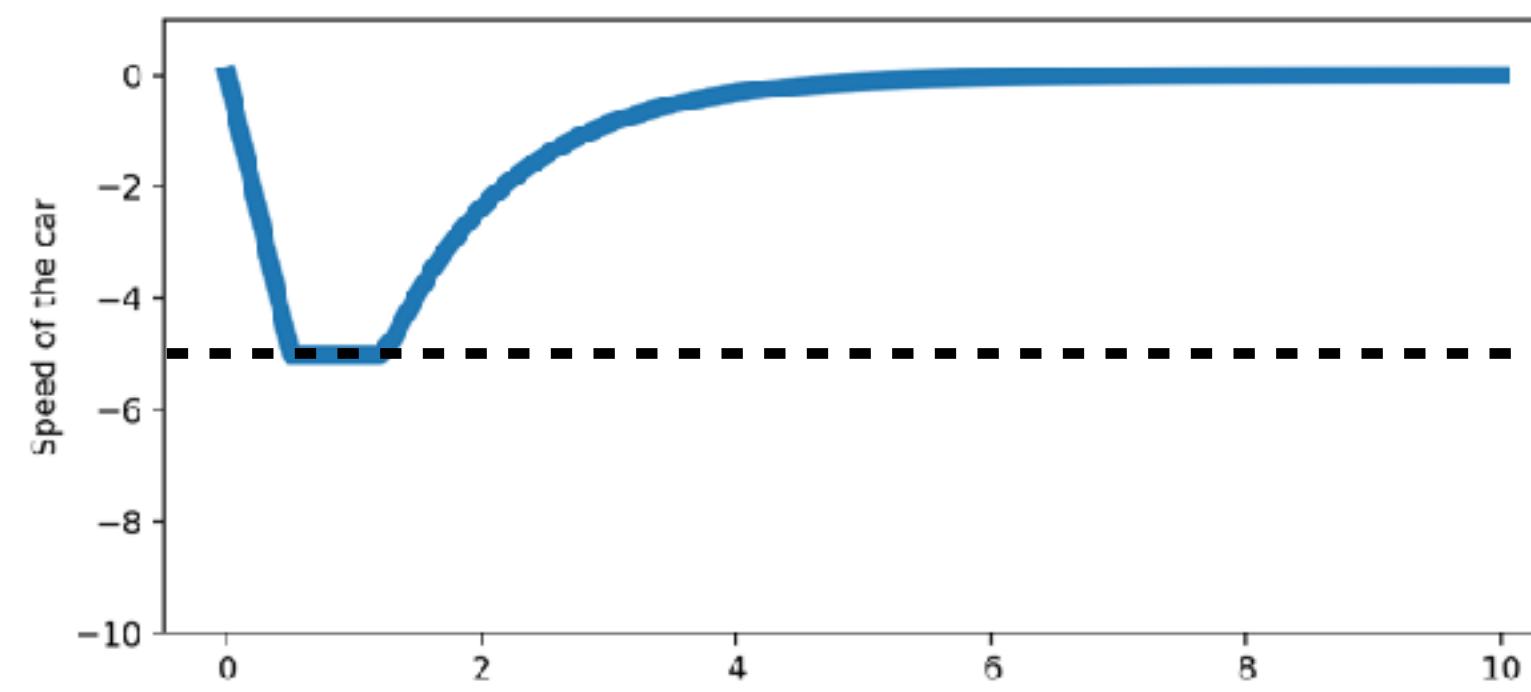
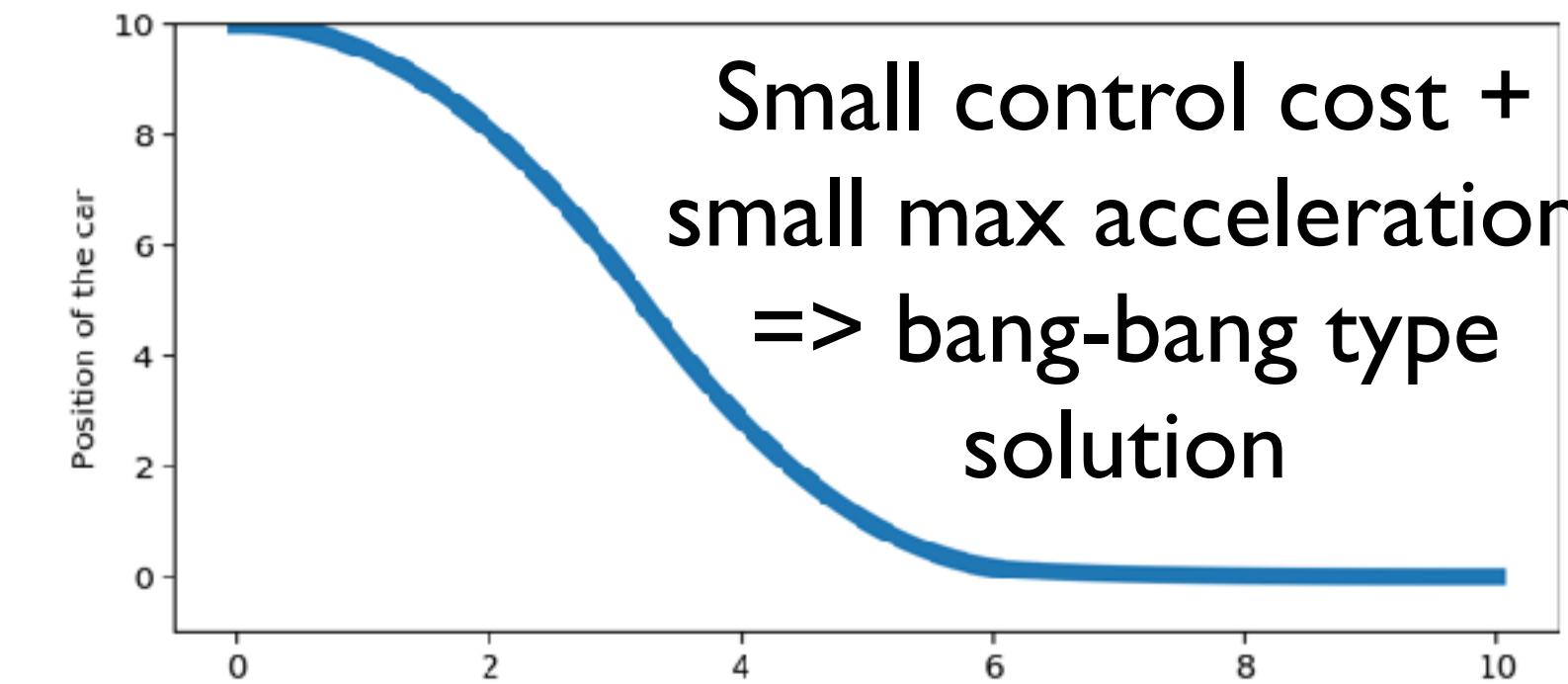
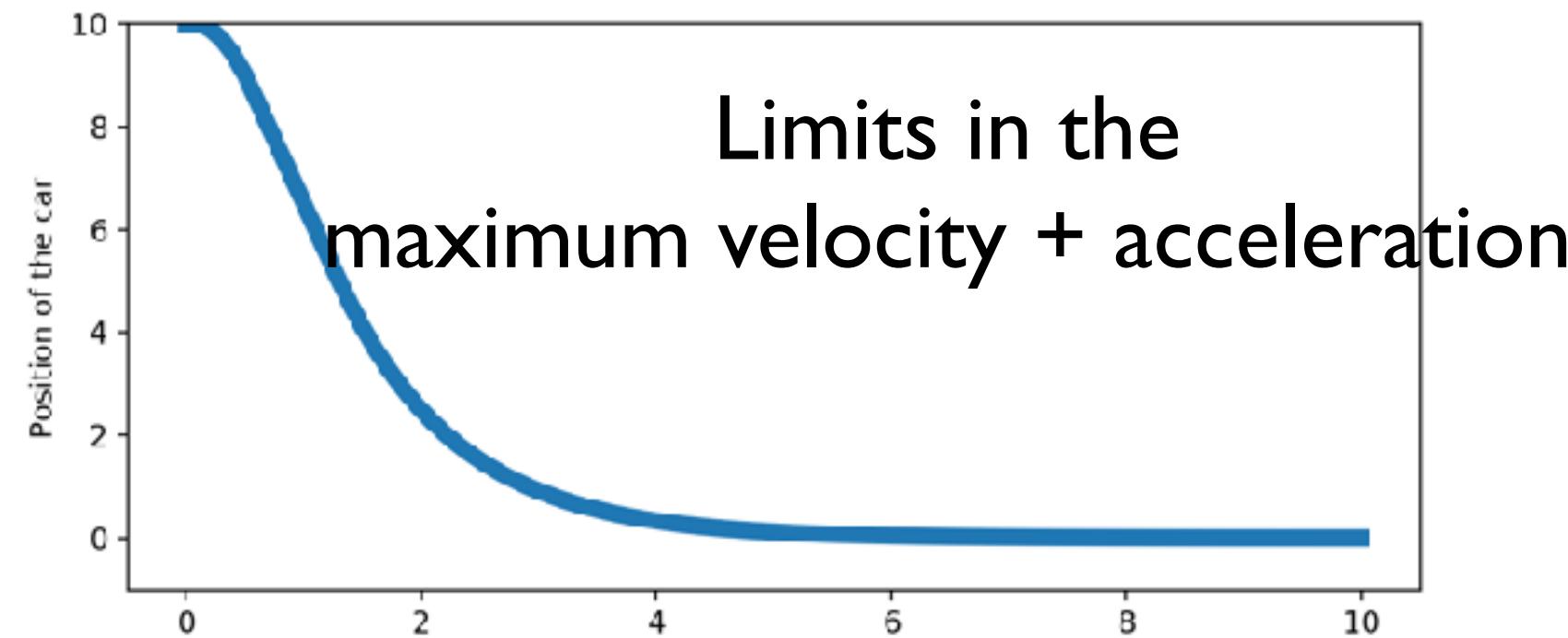
$$R = [0.1]$$





**Goal**  
**X**

$$x_{n+1} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} x_n + \begin{bmatrix} 0 \\ \Delta t \end{bmatrix} u_n$$



# Example: biped walking

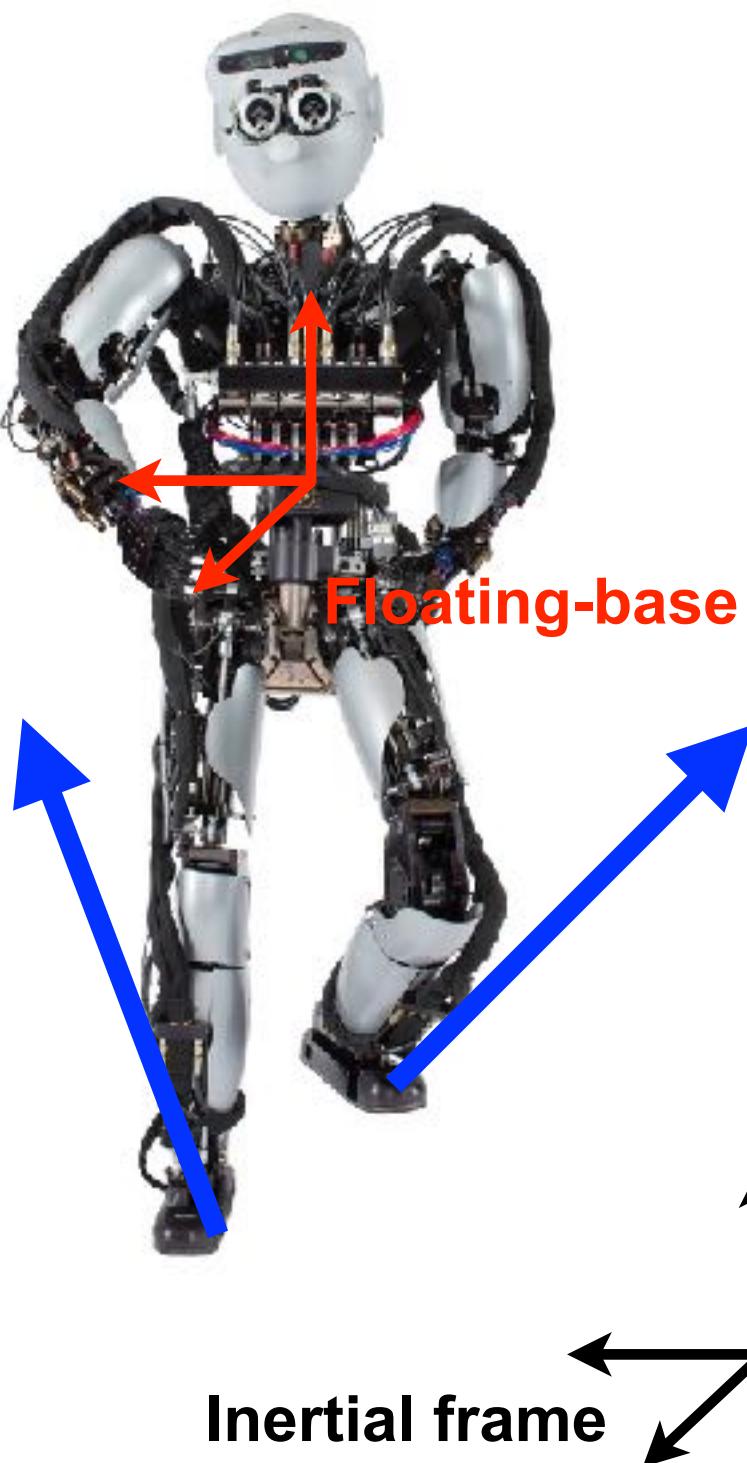


# What is the problem?

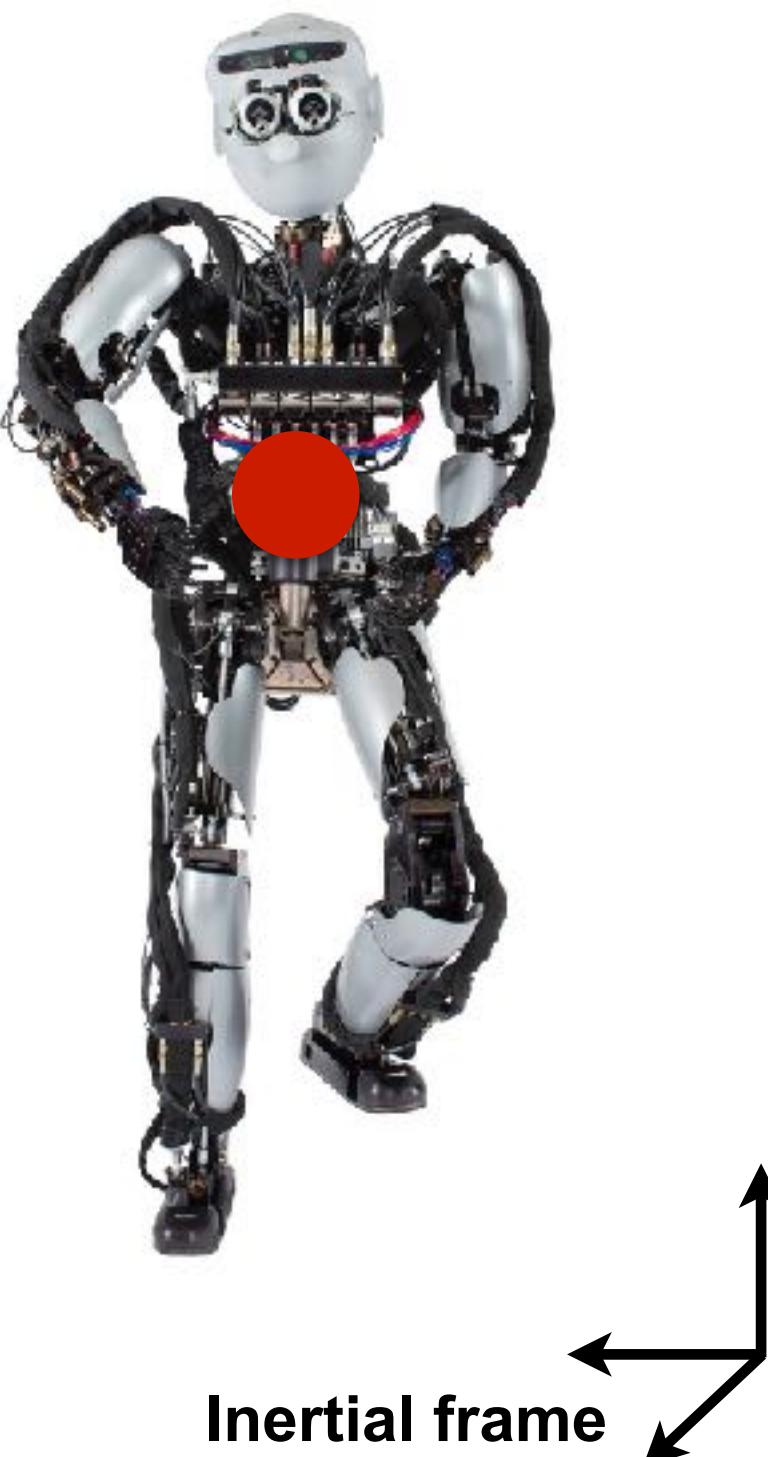
How can the robot move forward then?

Legged locomotion is about creating the right contact forces on the ground to keep balance and move forward

Gravity  
↓  
Contact Forces

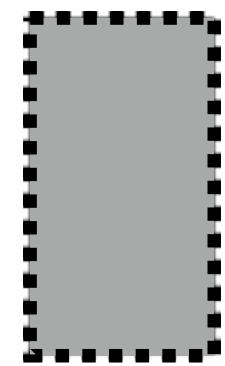


Motion of the  
center of mass

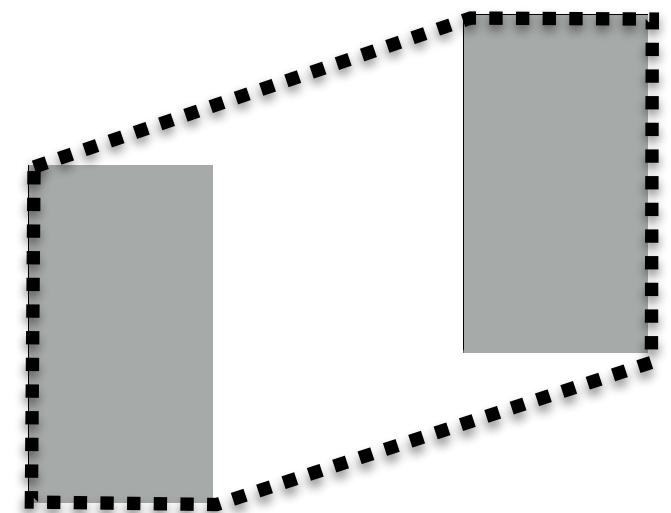


# Support polygon

For a robot with its feet on flat ground, the support polygon is the convex hull of the feet



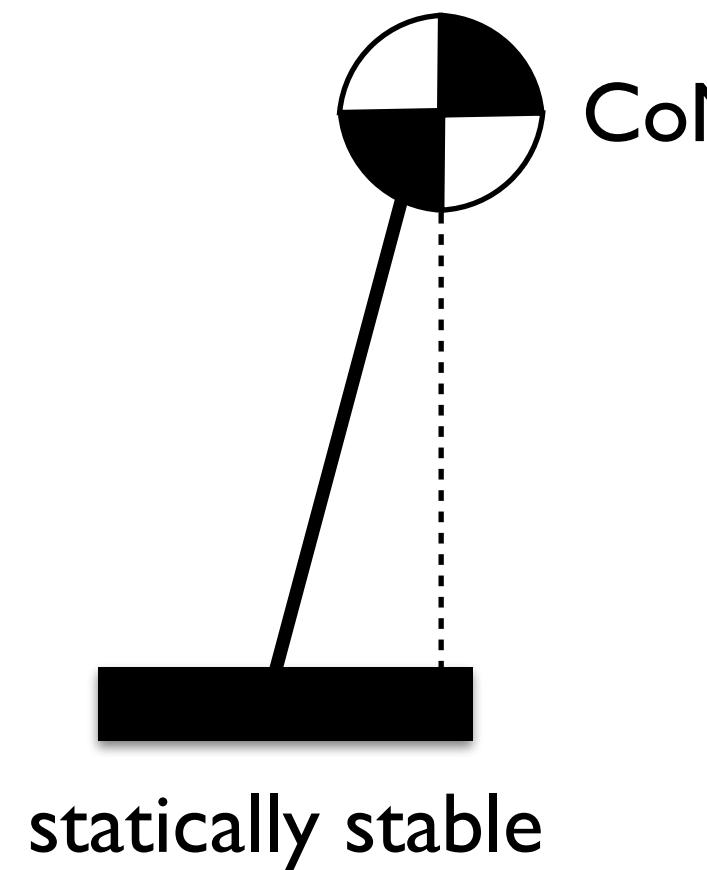
One foot on the ground and  
support polygon (top view)



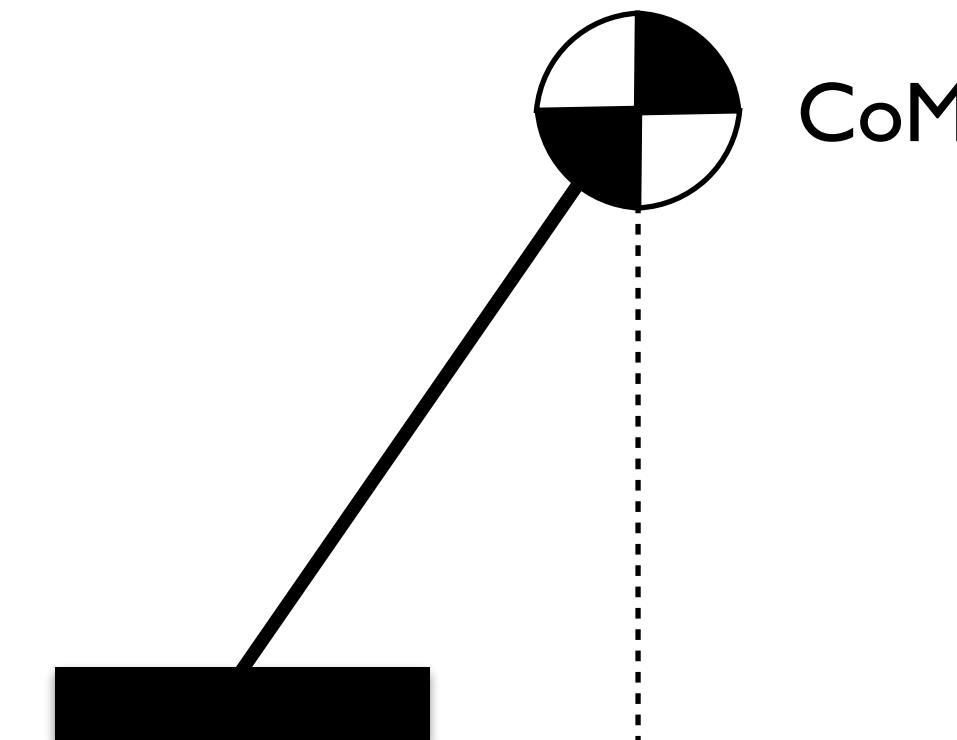
Two feet on the ground and  
support polygon (top view)

# Static stability

If a body starts in a configuration with zero velocity at  $t=0$  and stays in this configuration for  $t>0$  we say that the body is statically stable



statically stable

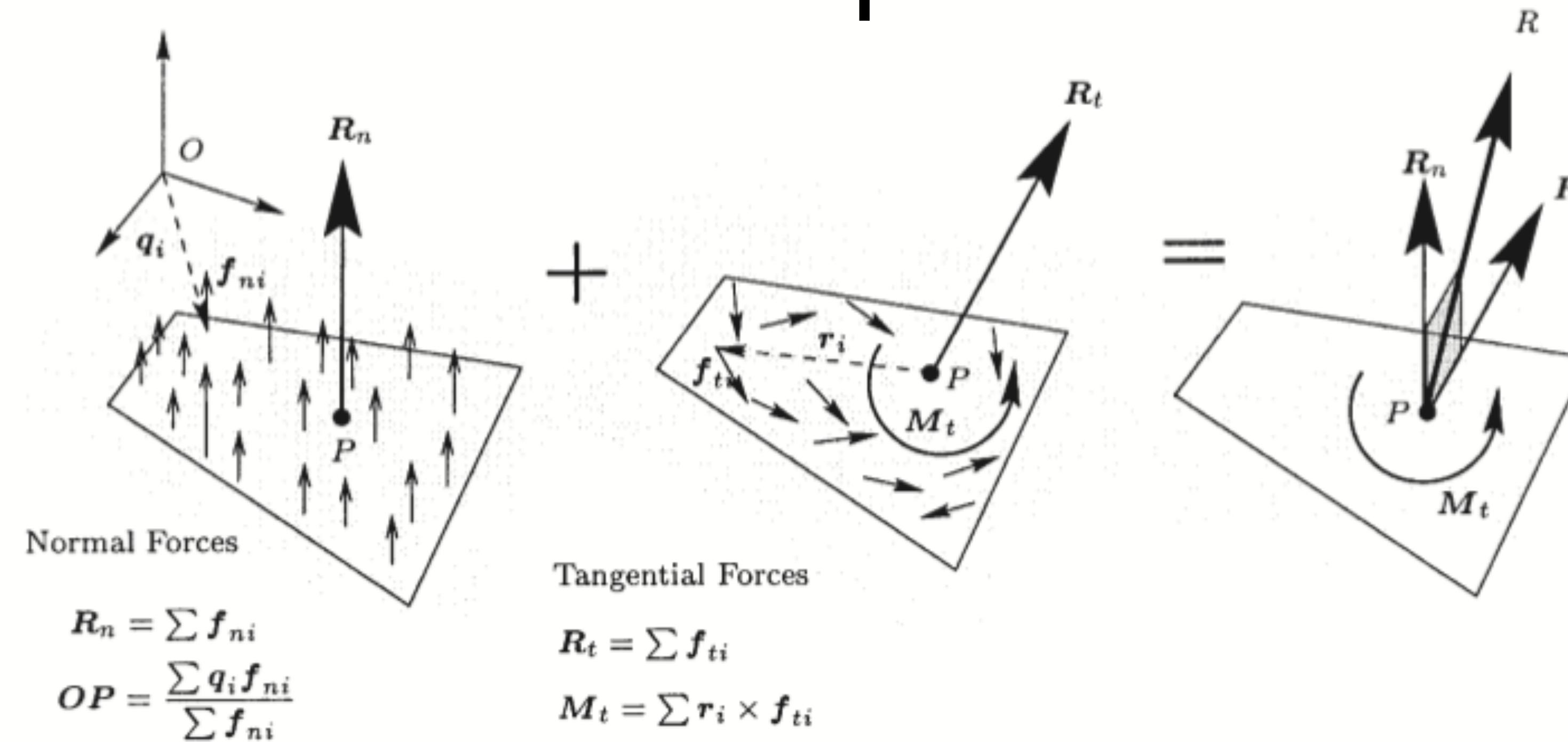


unstable (the foot will rotate around its edge)

A robot is statically stable if the projection of its Center of Mass (CoM) lies inside the support polygon

Static walking strategy: move the robot slowly (velocity close to 0), such that its CoM is always above the support polygon

# Center of pressure



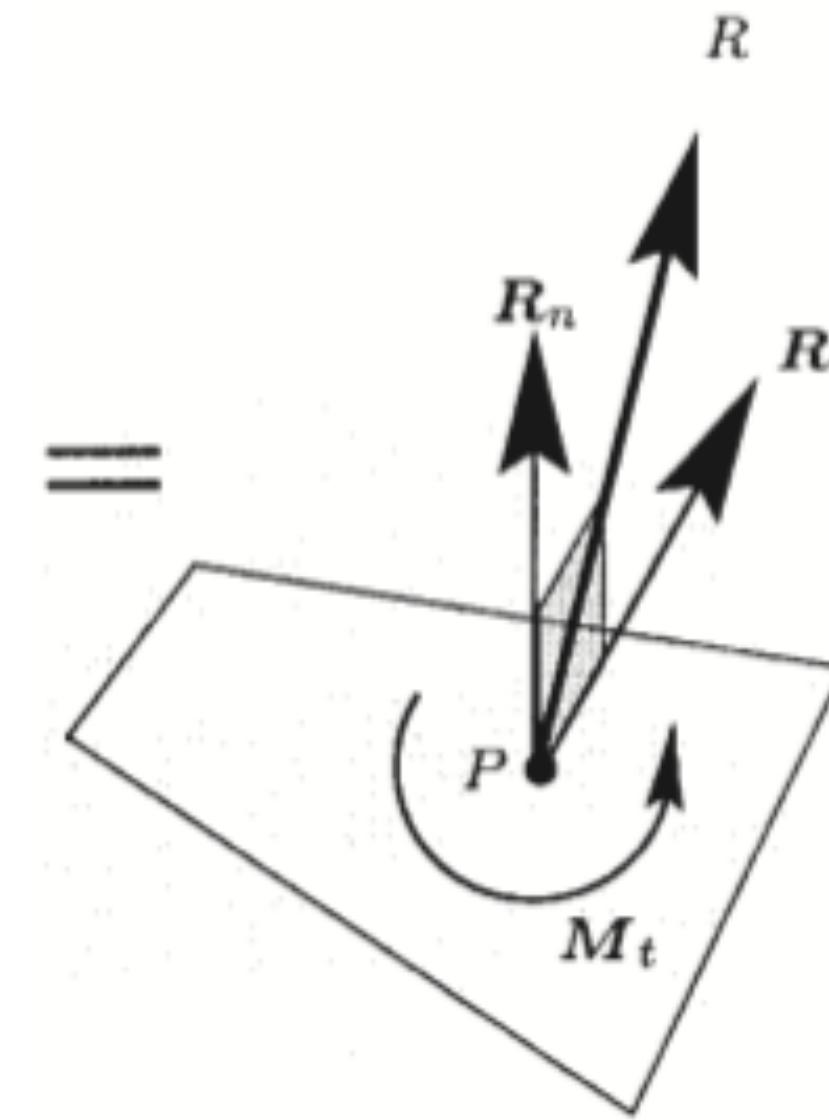
The center of pressure (CoP) is the point of the ground where the resultant force  $R_n$  acts

$$OP = \frac{\sum q_i f_{ni}}{\sum f_{ni}}$$

The CoP is the point of application of the ground reaction force vector

# Center of pressure

The CoP is the point of application of the ground reaction force vector

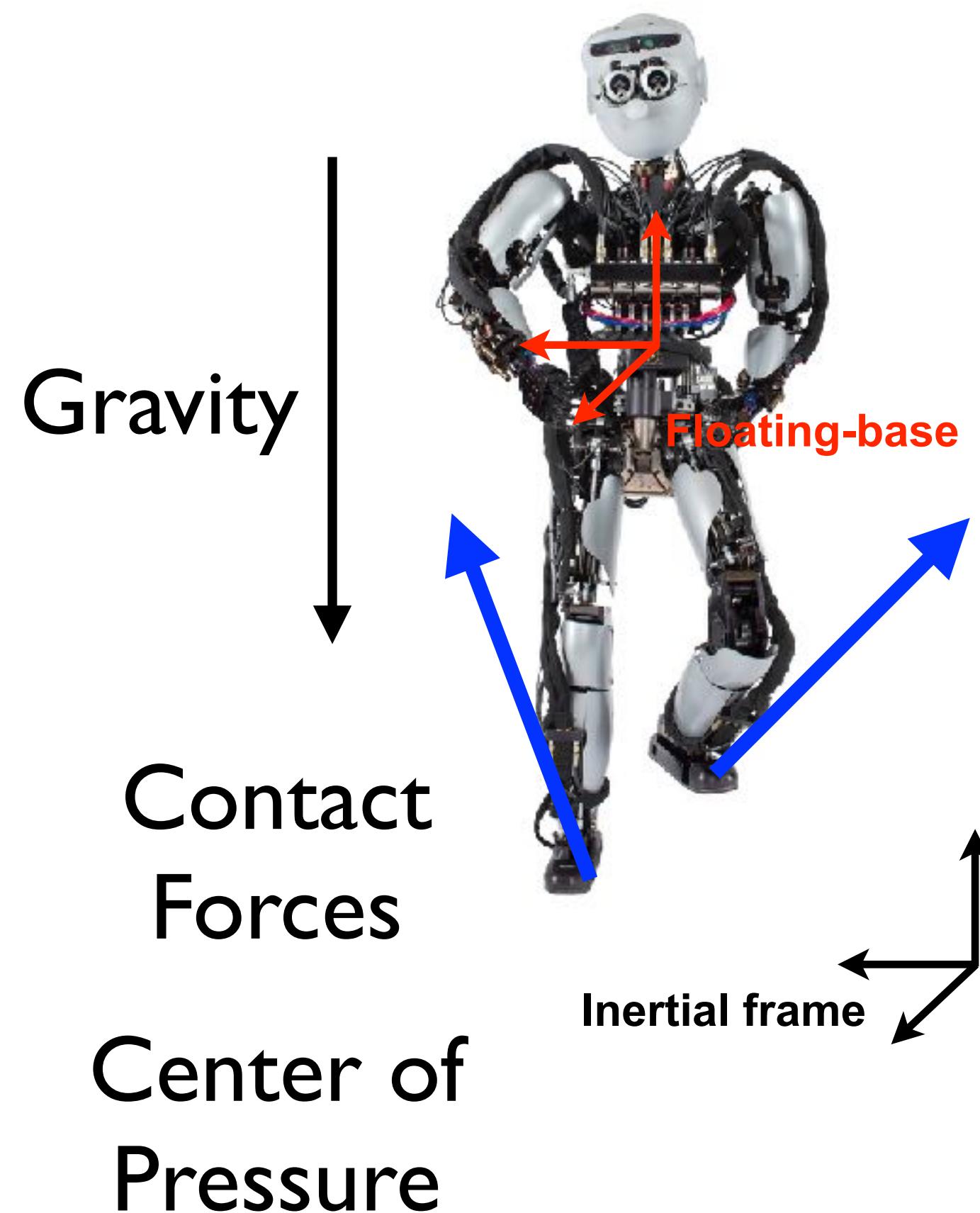


On flat ground, the CoP lies inside the polygon of support

If the CoP is on the edge of the support polygon, the foot will rotate and leave the ground

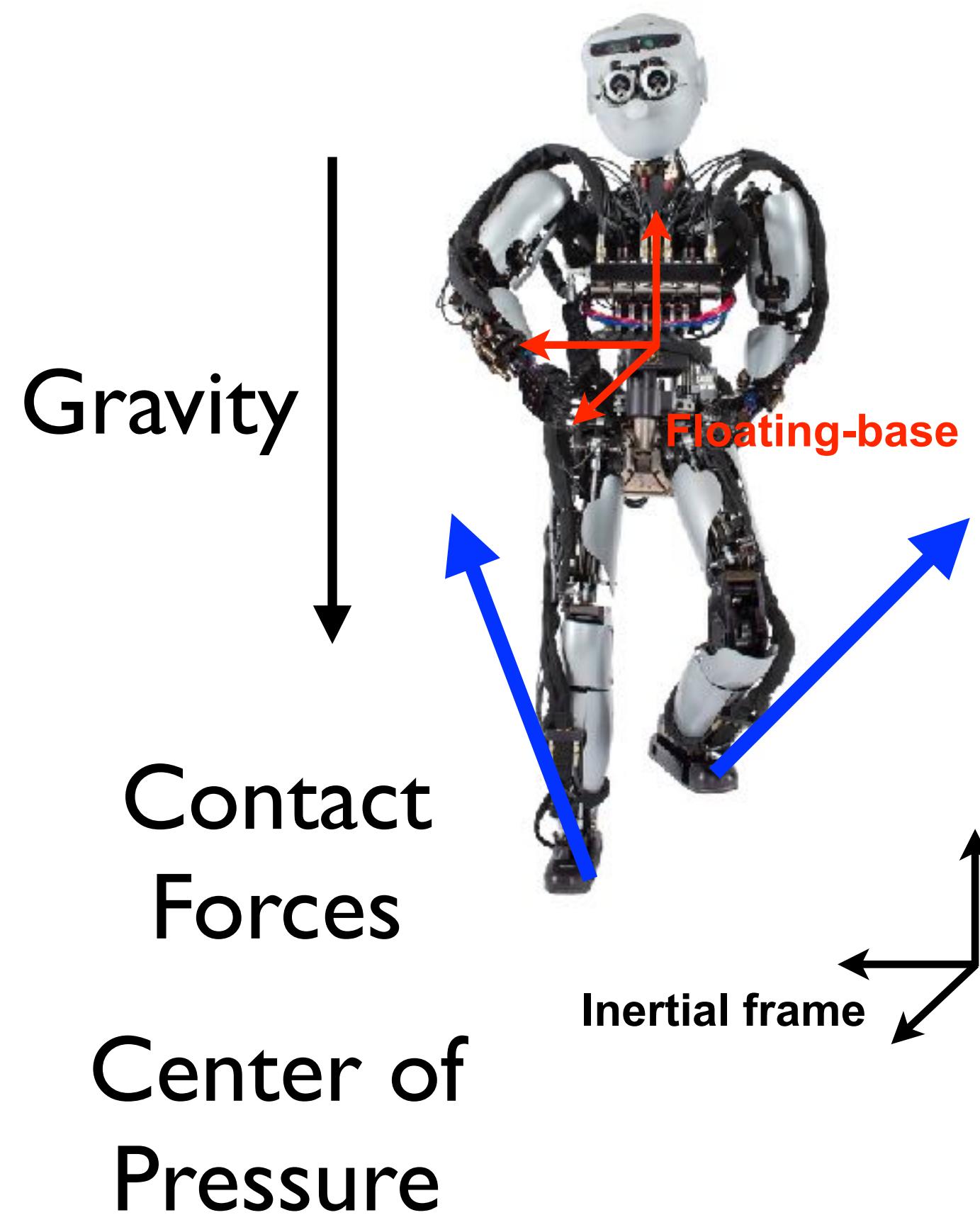
# Control of walking

1. Decide where to step  
(footstep planning)
2. Compute desired motion of the center of mass compatible with the physics (OC problem)
3. Compute a motion of the full robot to follow this desired CoM motion (in particular compute swing foot motions)
4. Control the robot to execute this movement
5. Adapt and Repeat



# Control of walking

1. Decide where to step  
(footstep planning)
2. Compute desired motion of the center of mass compatible with the physics (OC problem)
3. Compute a motion of the full robot to follow this desired CoM motion (in particular compute swing foot motions)
4. Control the robot to execute this movement
5. Adapt and Repeat



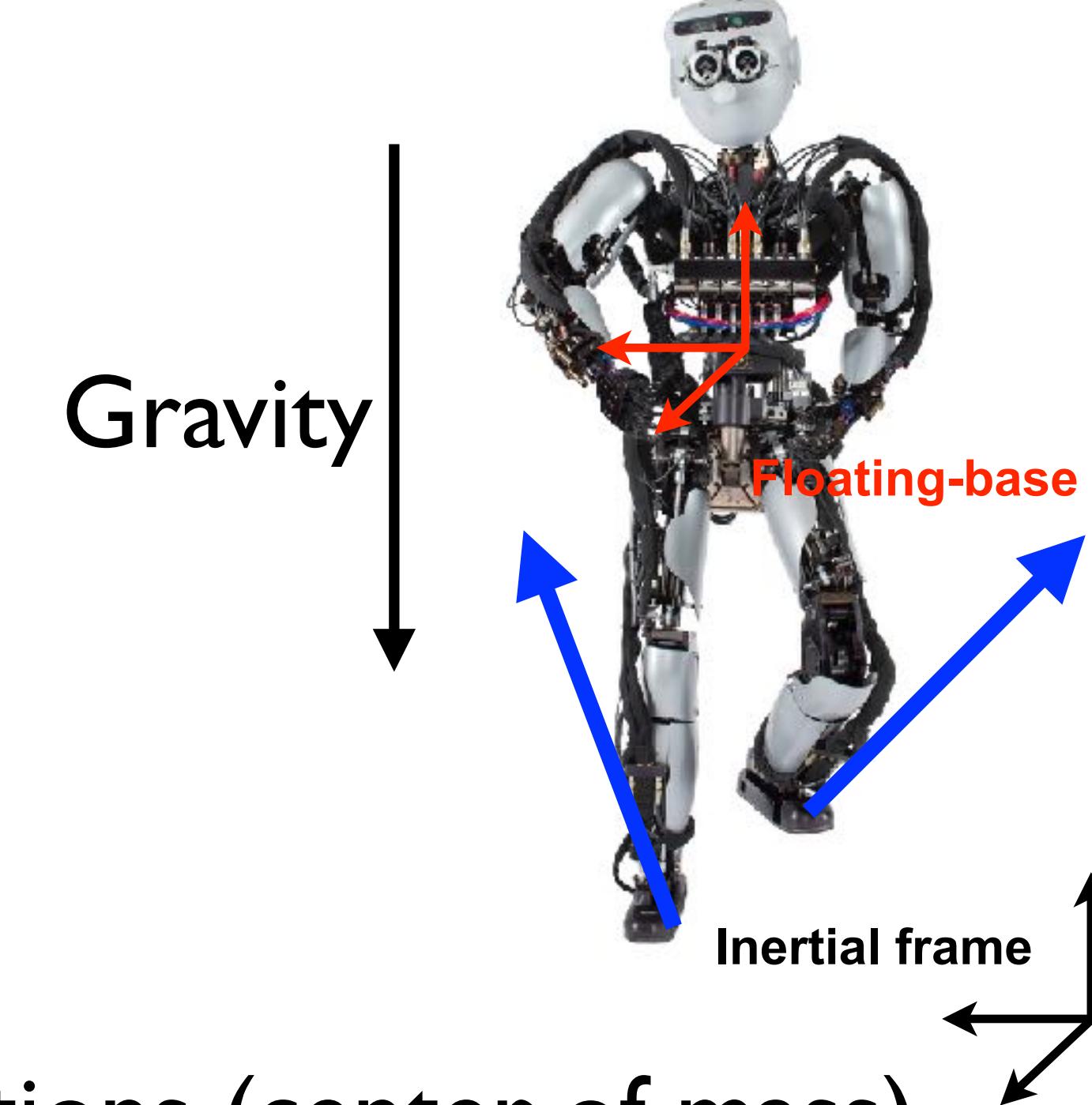
We need to relate the motion of the CoM to the forces exerted on the ground through the CoP

$$m\ddot{\mathbf{c}} = \sum_i \mathbf{f}_i - mg$$

$$\dot{\mathbf{L}} = \sum_i (\mathbf{p}_i - \mathbf{c}) \times \mathbf{f}_i + \boldsymbol{\tau}_i$$

Newton equations (center of mass)

Euler equations (angular momentum)



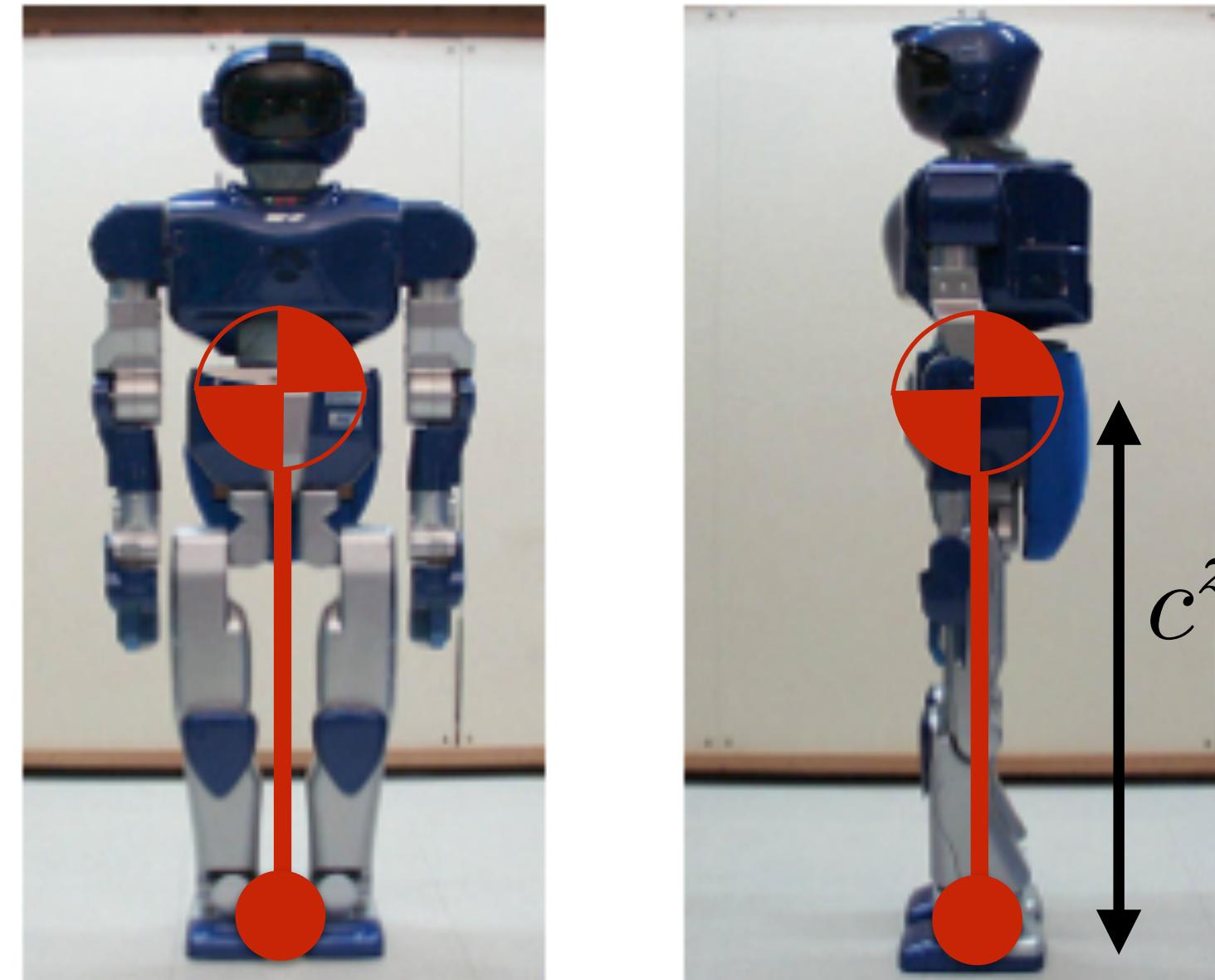
# Linear Inverted pendulum model (LIPM)

[Kajita et al. 2001]

If we assume  $\ddot{c}^z \simeq 0$  (i.e. the height of the CoM does not change) and also that  $\dot{\mathbf{L}}^{x,y} \simeq 0$  (i.e. that the angular momentum around the CoM does not vary either) we get the linear inverted pendulum model

$$\ddot{\mathbf{c}}^{x,y} = \frac{g}{c^z} (\mathbf{c}^{x,y} - \underline{\mathbf{p}}^{x,y})$$

**CoP**

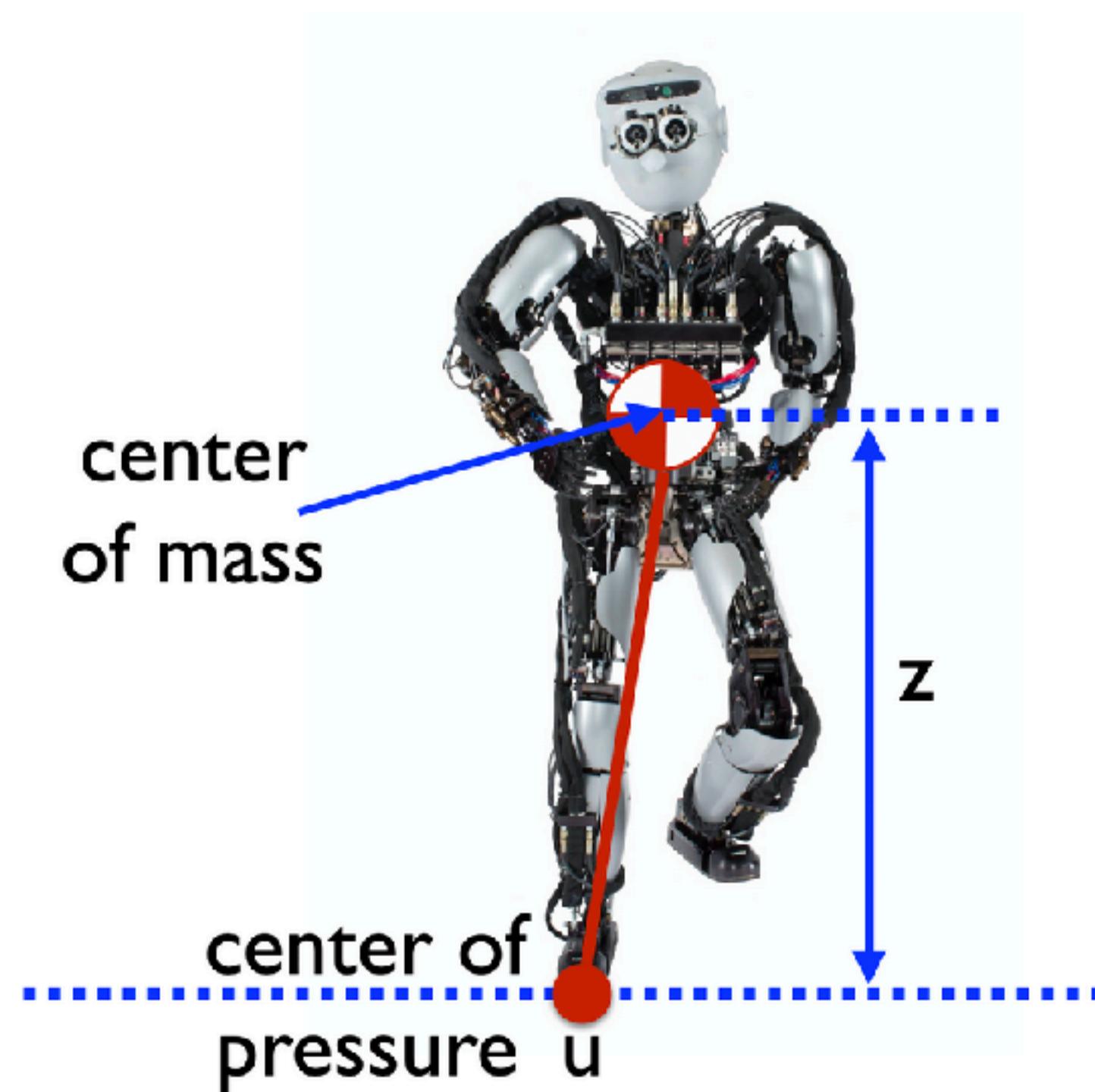


# Model predictive control for walking

The equations of motion of the LIPM can be written as a function of the CoP

$$\ddot{\mathbf{c}}^x = \frac{g}{c^z}(\mathbf{c}^x - \mathbf{p}^x)$$

$$\ddot{\mathbf{c}}^y = \frac{g}{c^z}(\mathbf{c}^y - \mathbf{p}^y)$$



# Linear Inverted pendulum model (LIPM)

[Kajita et al. 2001]

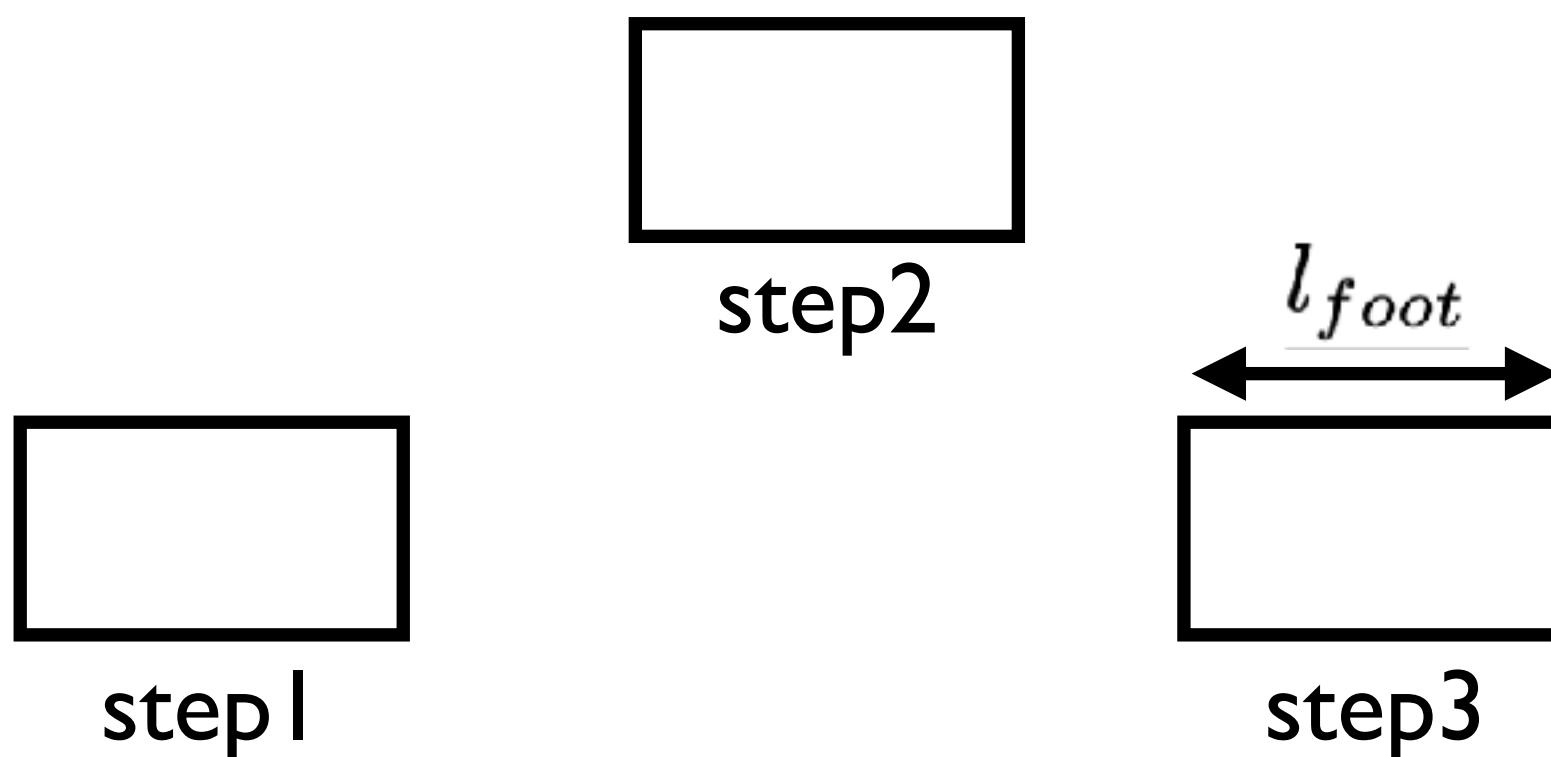
$$\dot{c} = v$$

$$\dot{c} = \omega(c - p)$$

Assuming  $p$  is constant during  $\Delta t$  then we get the exact discrete equation (where  $t = n\Delta t$ )

$$\Rightarrow \begin{bmatrix} c_{n+1} \\ v_{n+1} \end{bmatrix} = \begin{bmatrix} \cosh(\omega\Delta t) & \omega^{-1} \sinh(\omega\Delta t) \\ \omega \sinh(\omega\Delta t) & \cosh(\omega\Delta t) \end{bmatrix} \begin{bmatrix} c_n \\ v_n \end{bmatrix} + \begin{bmatrix} 1 - \cosh(\omega\Delta t) \\ -\omega \sinh(\omega\Delta t) \end{bmatrix} p_n$$

Assuming a predefined sequence of steps (right-left), with predefined stepping time and foot positions, we can define at each time step  $n$  support polygon constraints



$$f_n - \frac{l_{foot}}{2} < p_n < f_n + \frac{l_{foot}}{2}$$

where  $f_n$  is foot position at time  $n$   
 $l_{foot}$  is the foot length

# Linear Inverted pendulum model (LIPM)

[Kajita et al. 2001]

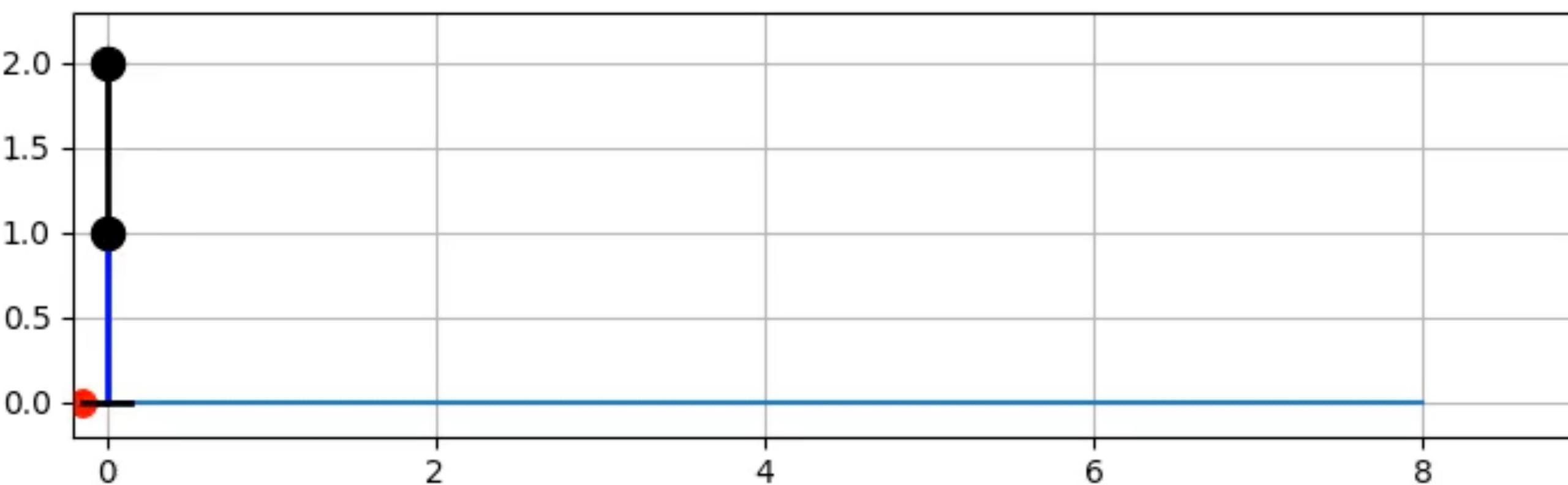
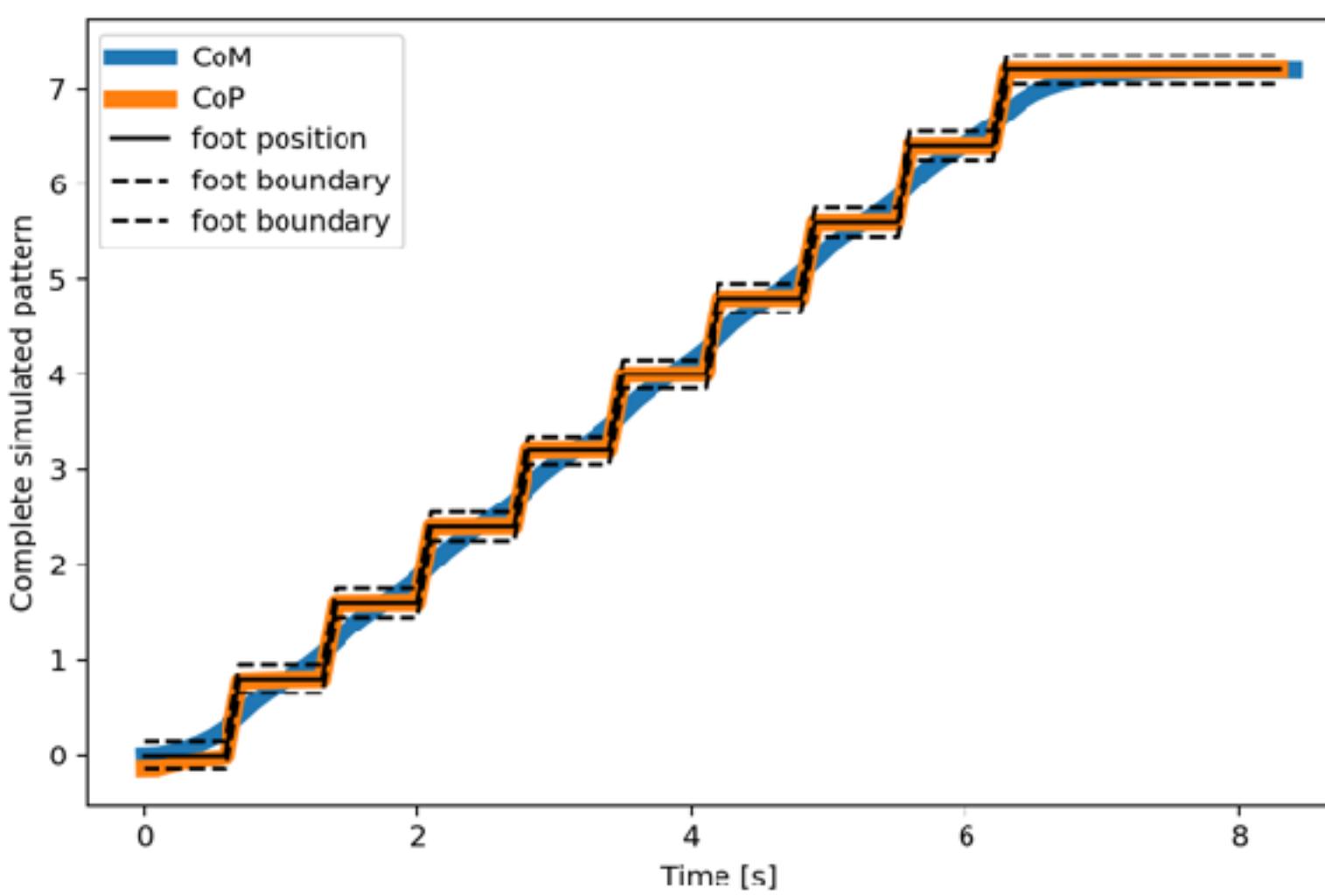
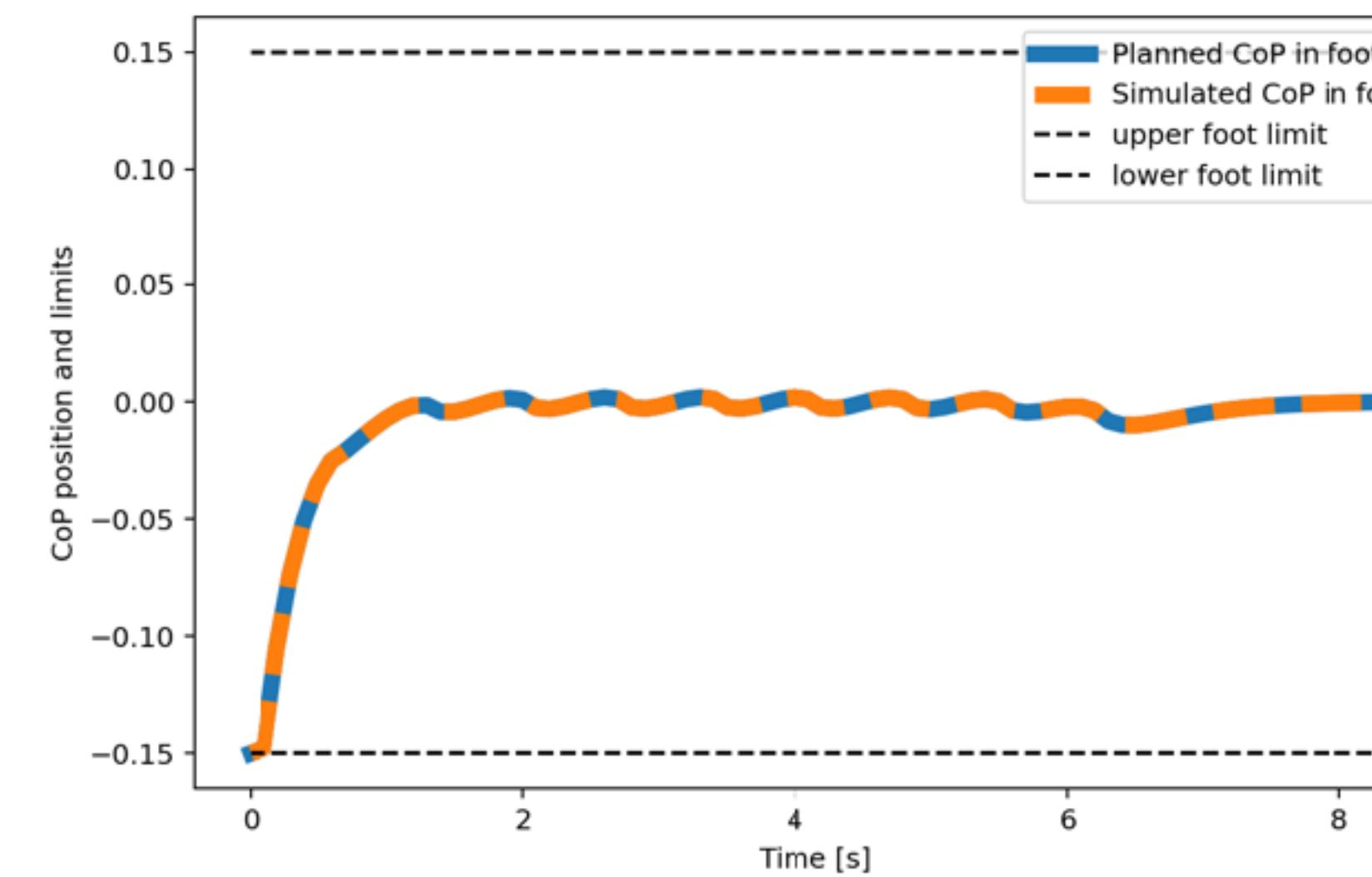
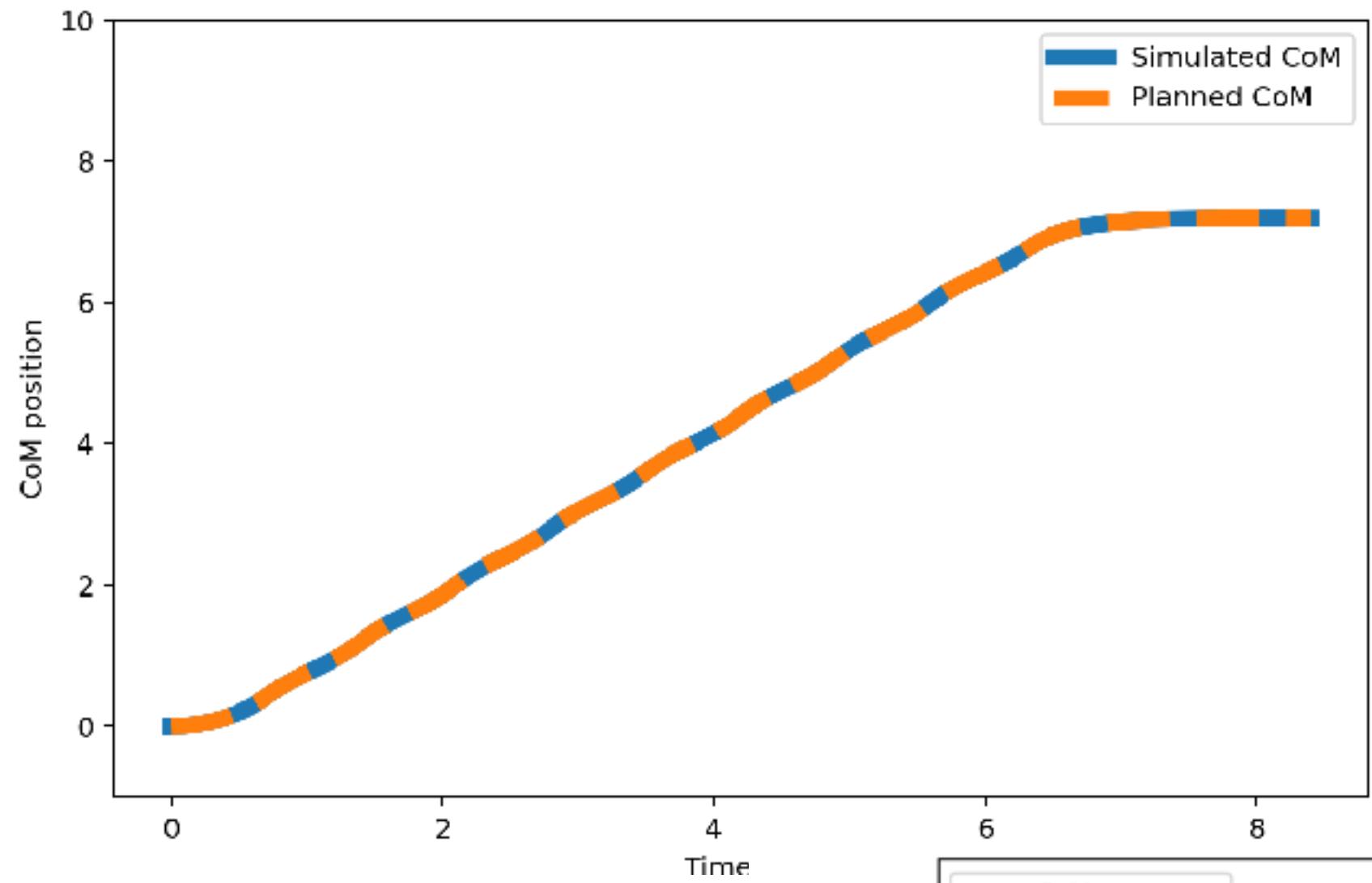
$$\min_{u_n} \sum_{n=0}^{N-1} \begin{pmatrix} c_n - f_n \\ v_n \end{pmatrix}^T Q \begin{pmatrix} c_n - f_n \\ v_n \end{pmatrix} + (p_n - f_n)^T R (p_n - f_n) + \begin{pmatrix} c_N - f_N \\ v_N \end{pmatrix}^T Q_N \begin{pmatrix} c_N - f_N \\ v_N \end{pmatrix}$$

subject to

$$\begin{bmatrix} c_{n+1} \\ v_{n+1} \end{bmatrix} = \begin{bmatrix} \cosh(\omega\Delta t) & \omega^{-1} \sinh(\omega\Delta t) \\ \omega \sinh(\omega\Delta t) & \cosh(\omega\Delta t) \end{bmatrix} \begin{bmatrix} c_n \\ v_n \end{bmatrix} + \begin{bmatrix} 1 - \cosh(\omega\Delta t) \\ -\omega \sinh(\omega\Delta t) \end{bmatrix} p_n$$

and

$$f_n - \frac{l_{foot}}{2} \leq p_n \leq f_n + \frac{l_{foot}}{2}$$



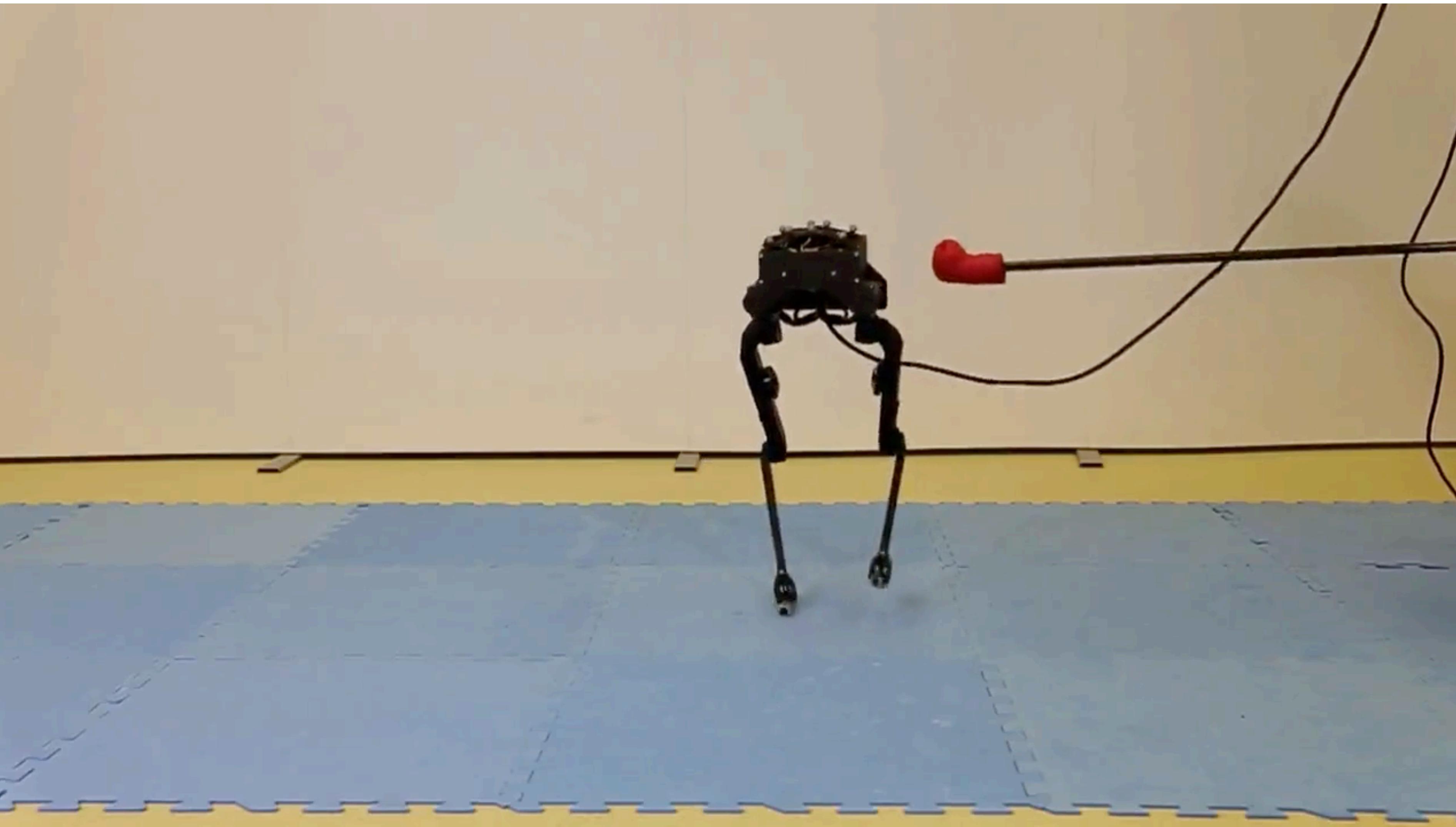


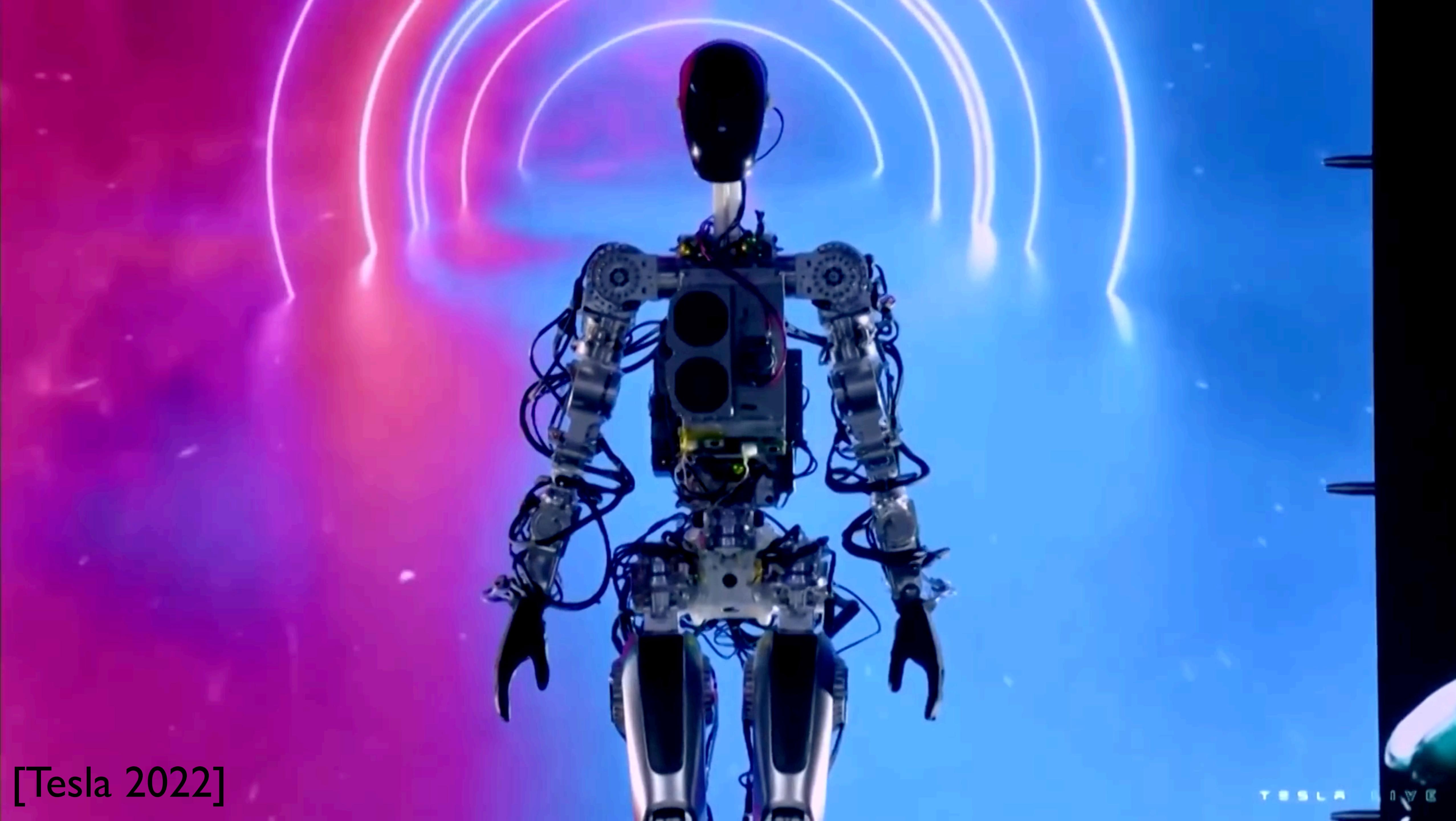
HRP2 - CNRS-AIST [Herdt, et al., 2010]

Linear inverted pendulum models  
are also used in quadruped robots



Linear inverted pendulum models can be used with different stability criterions





[Tesla 2022]

TESLA LIVE

**QPs are found everywhere!**