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
Office hours in person
Wednesday 3pm to 4pm
370 Jay street - room 801
(except next week)

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	HW 2	
5		Lecture 5: Model-predictive control		
6		Lecture 6: 0th order methods (tentative)	HW 3	Project 1
7	Policy optimization	Lecture 7: Bellman's principle		
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			

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

Homework 1 is due next week

Karush Kuhn Tucker conditions of optimality

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

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

The vectors λ and μ 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 λ^* and μ^* such that the following conditions are satisfied

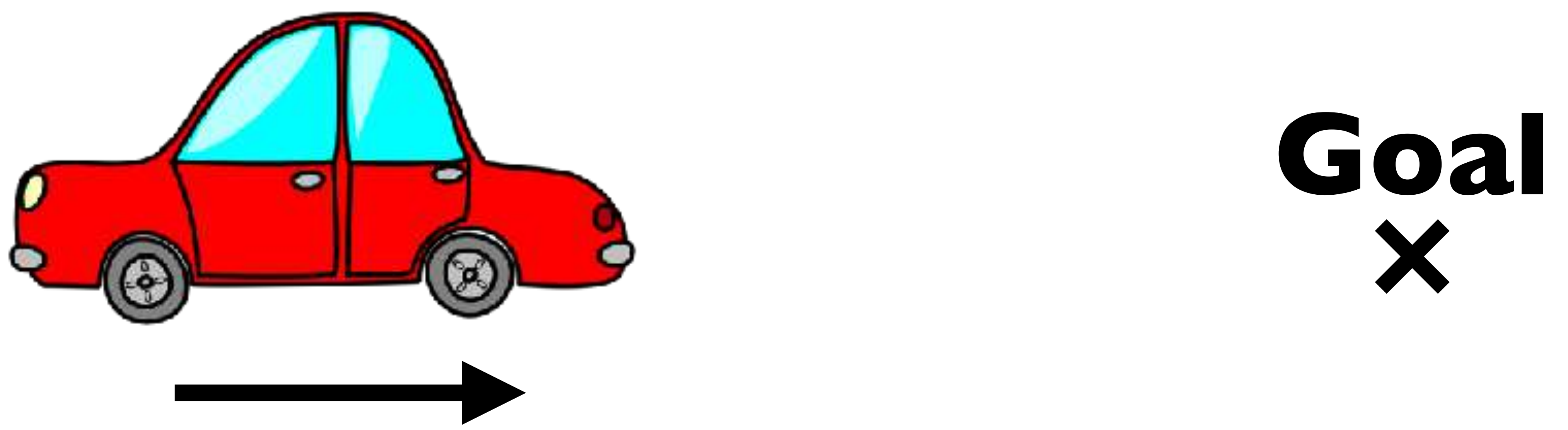
$$\begin{aligned} \nabla_x L(x^*, \lambda^*, \mu^*) &= 0 \\ g(x^*) &= 0 \\ h(x^*) &\leq 0 \\ \mu_i &\geq 0 \;\; \forall i \\ \mu_i h_i(x^*) &= 0 \;\; \forall i \end{aligned}$$

Optimal control of linear systems with quadratic costs

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

$$\text{subject to} \quad x_{n+1} = A x_n + B u_n$$

$$x_0 = x_{init}$$



$$
\begin{aligned}
\min_{x_n, u_n} \ & \frac{1}{2} \sum_{n=0}^{N-1} x_n^T Q x_n + u_n^T R u_n + x_N^T Q x_N \\
\text{subject to} \ & x_{n+1} = A x_n + B u_n \\
& x_0 = x_{init}
\end{aligned}
$$

$$\Longleftrightarrow$$

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

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

$$\Downarrow$$

$$
\begin{aligned}
\min_{y} \ & \frac{1}{2} y^T G y \\
\text{subject to} \ & My = p
\end{aligned}
$$

Optimal control of linear systems with quadratic costs

$$\min_y \frac{1}{2} y^T G y$$
subject to $My = p$

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

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

$$\nabla_x L = Gy + M^T \lambda = 0$$
$$\nabla_\lambda L = My - p = 0$$

or equivalently

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

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

If the matrix is invertible

Example

Inverting a matrix full of zeros…

$$\begin{bmatrix} G & M^T \\ M & 0 \end{bmatrix} \begin{pmatrix} y \\ \lambda \end{pmatrix} = \begin{pmatrix} 0 \\ p \end{pmatrix} \quad \not\Longrightarrow \quad \begin{pmatrix} y \\ \lambda \end{pmatrix} = \begin{bmatrix} G & M^T \\ M & 0 \end{bmatrix}^{-1} \begin{pmatrix} 0 \\ p \end{pmatrix}$$

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 $x_0 = x_{init}$

$$u_n = -K_n x_n$$
$$x_{n+1} = A x_n + B u_n$$

The number of multiplications needed to resolve 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

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

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

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

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

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

$$K_8 = \cdots$$

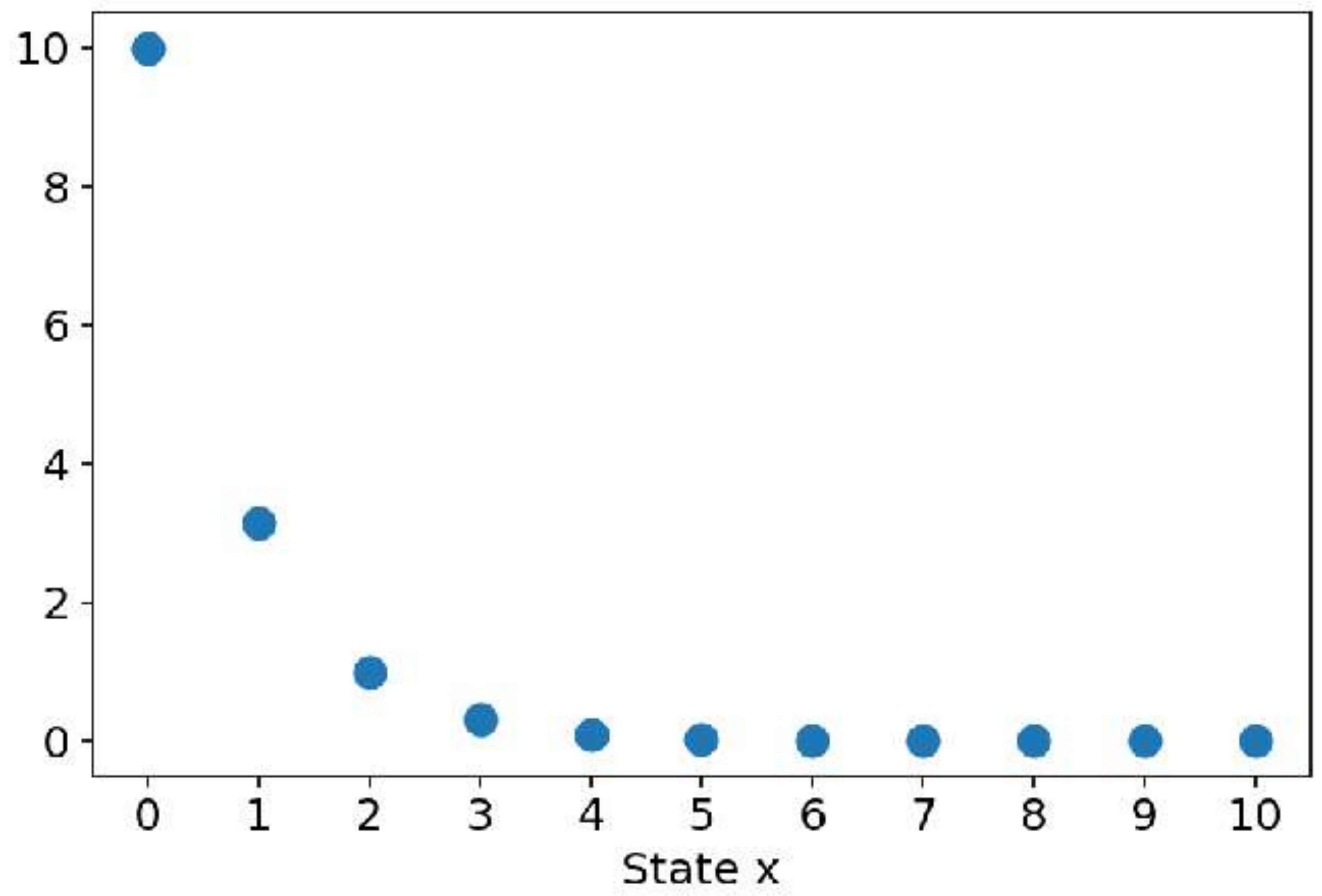
Example

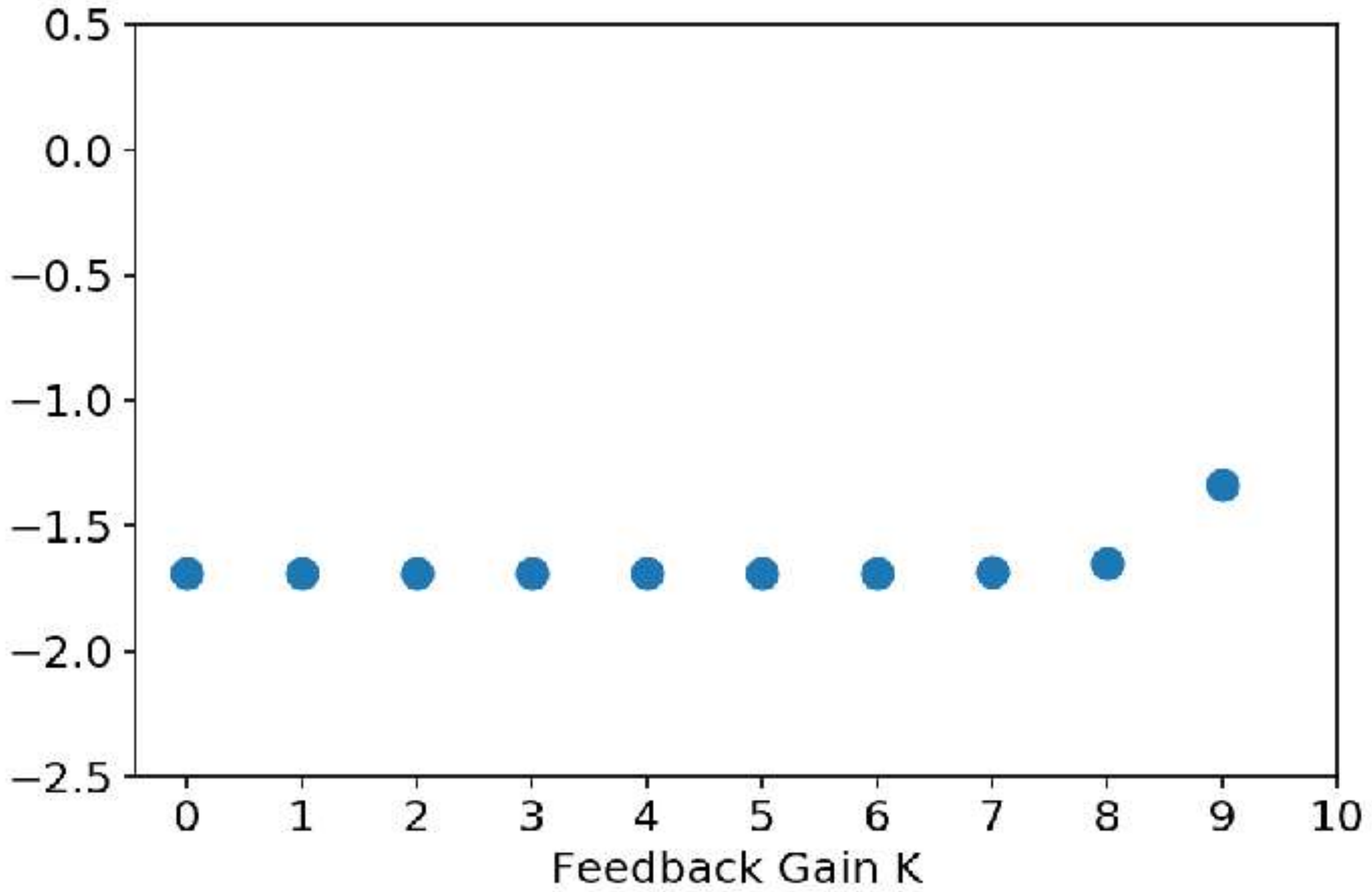
$$\min \sum_{i=0}^{N-1} \left(x_i^T Q x_i + u_i^T R u_i\right) + x_N^T Q x_N$$

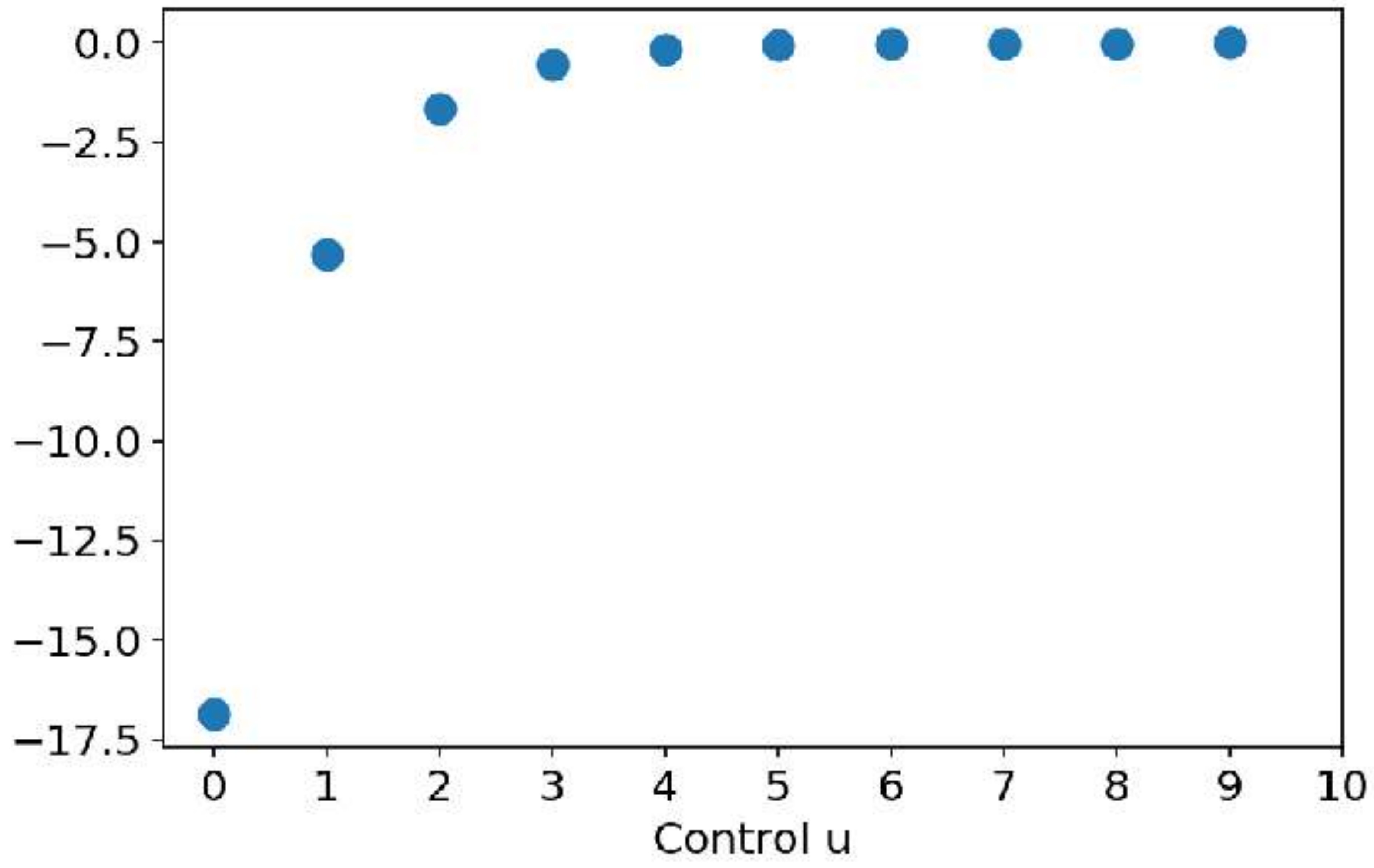
Subject to $x_{n+1} = 2x_n + u_n$

$N = 10 \qquad x_0 = 10$

$Q = 2 \; R = 1$







Example

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

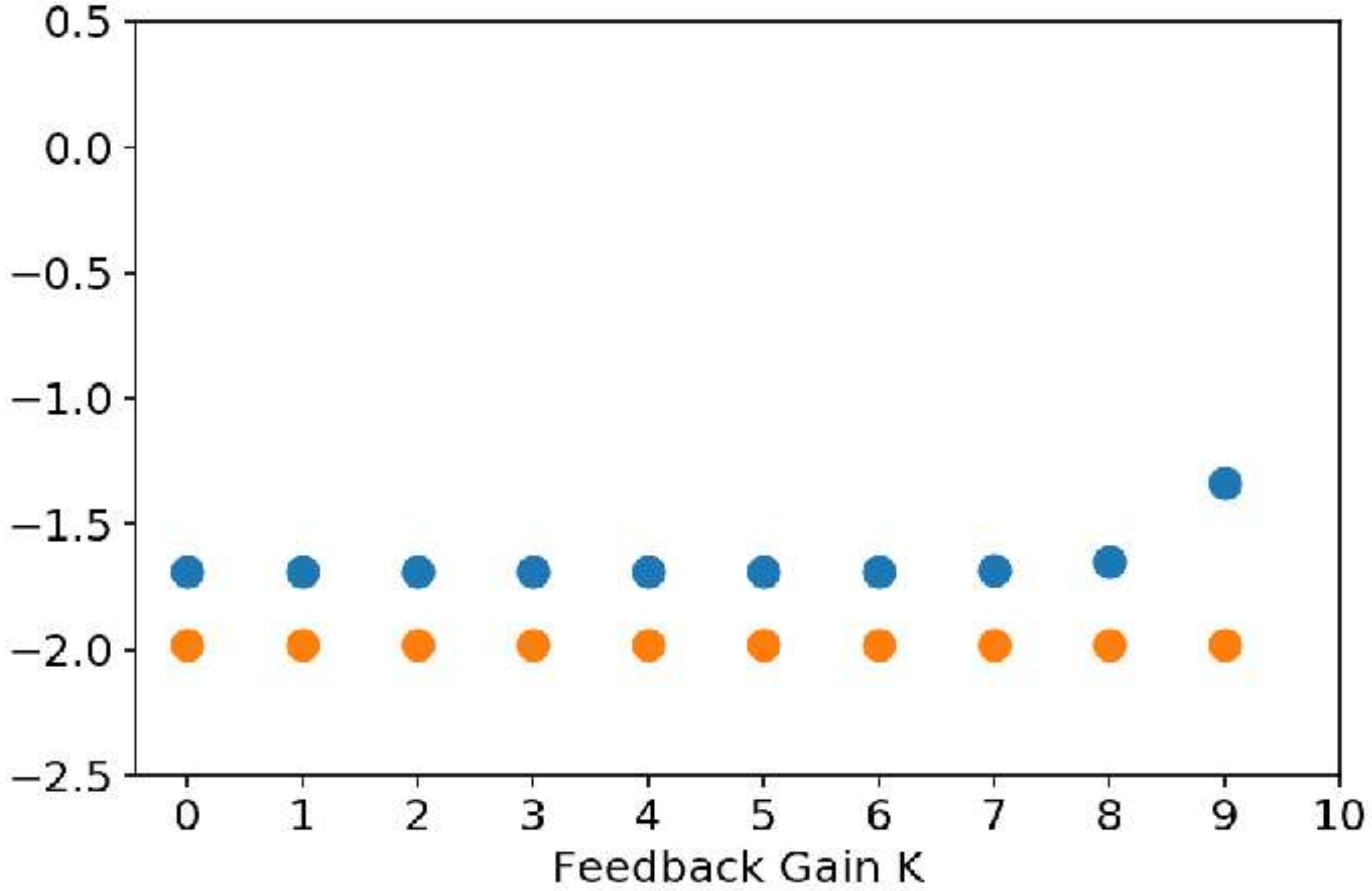
Subject to $x_{n+1} = 2x_n + u_n$

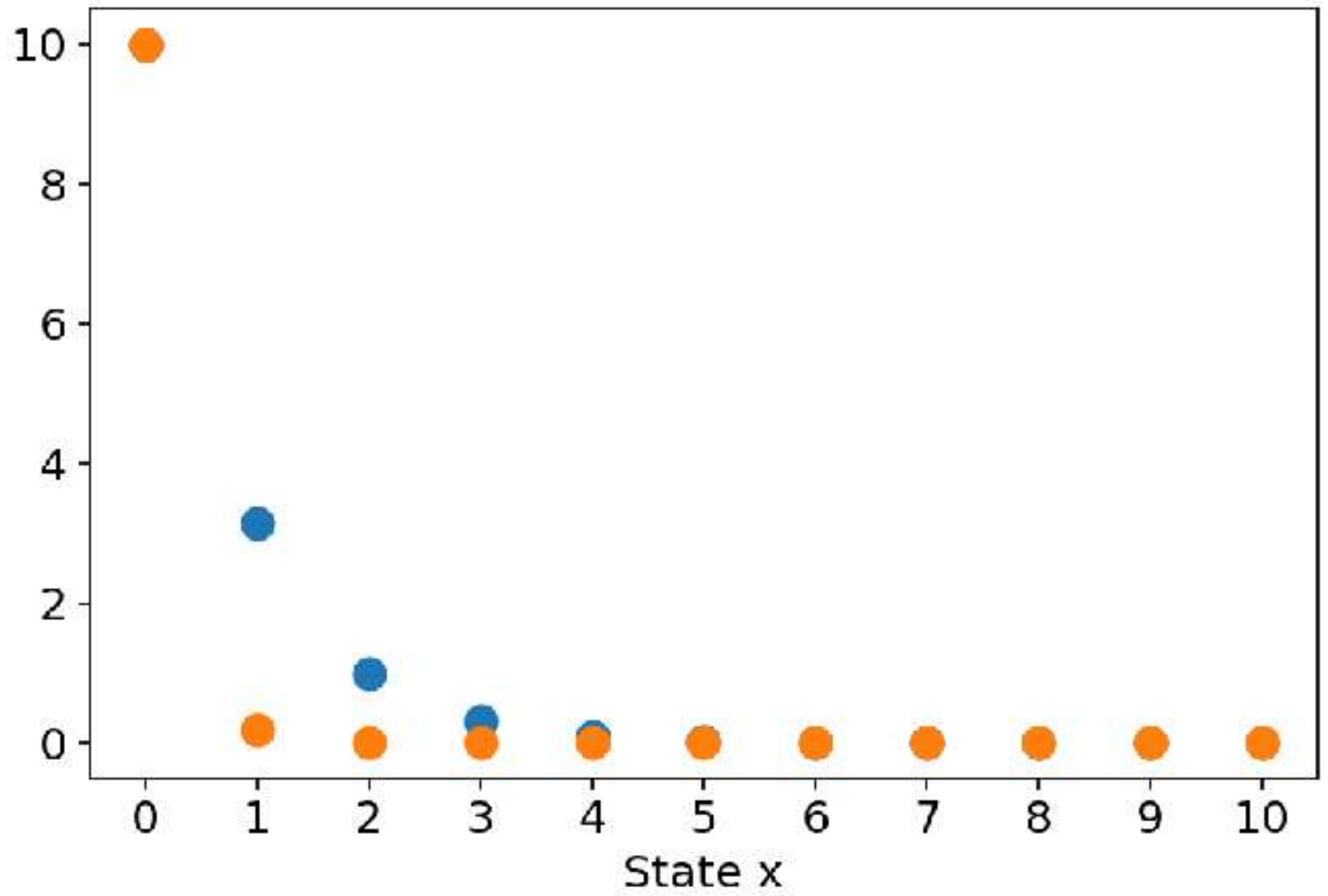
$N = 10 \quad x_0 = 10$

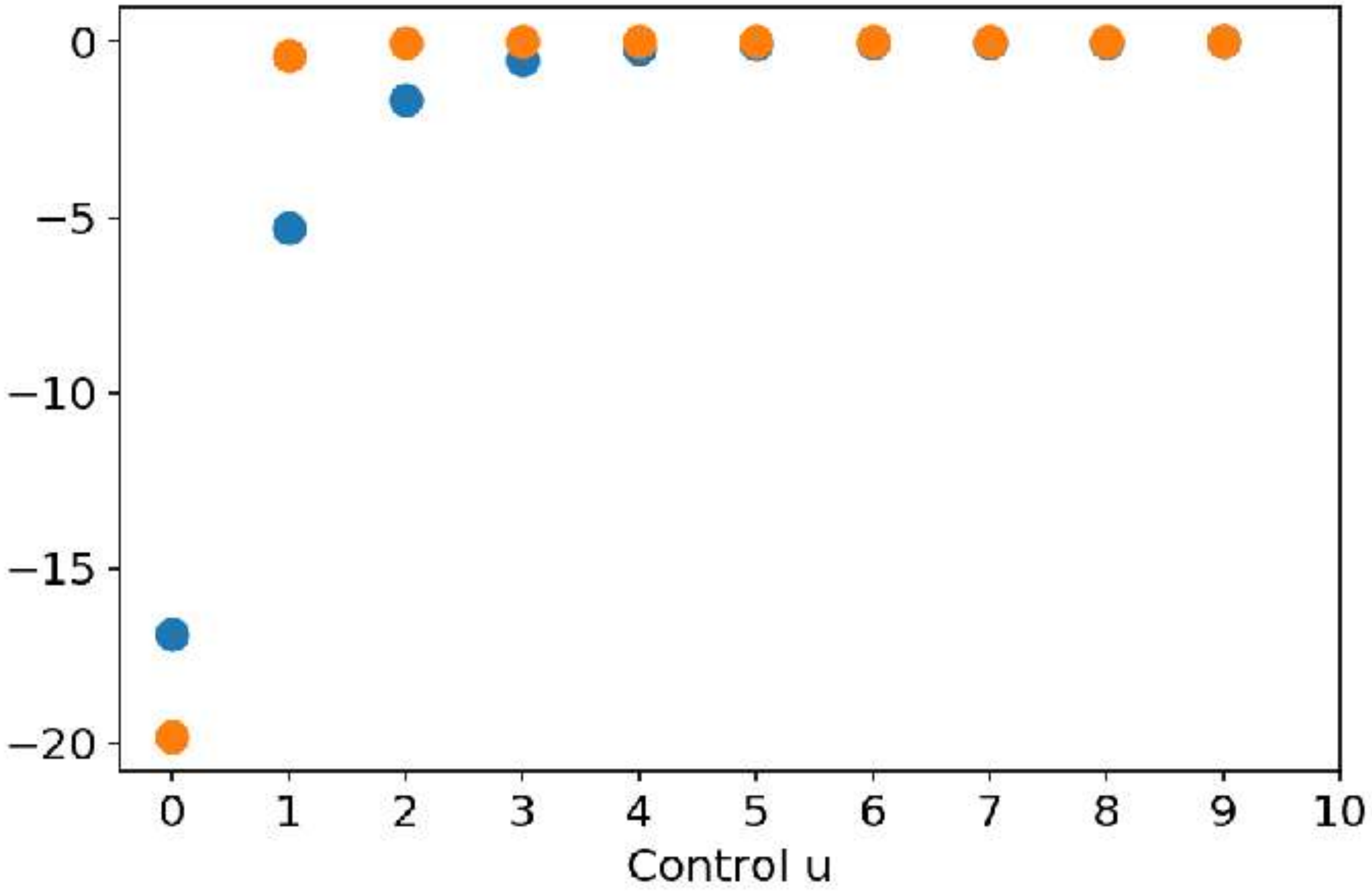
$Q = 2 \; R = 1$

$Q = 100 \; R = 1$

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





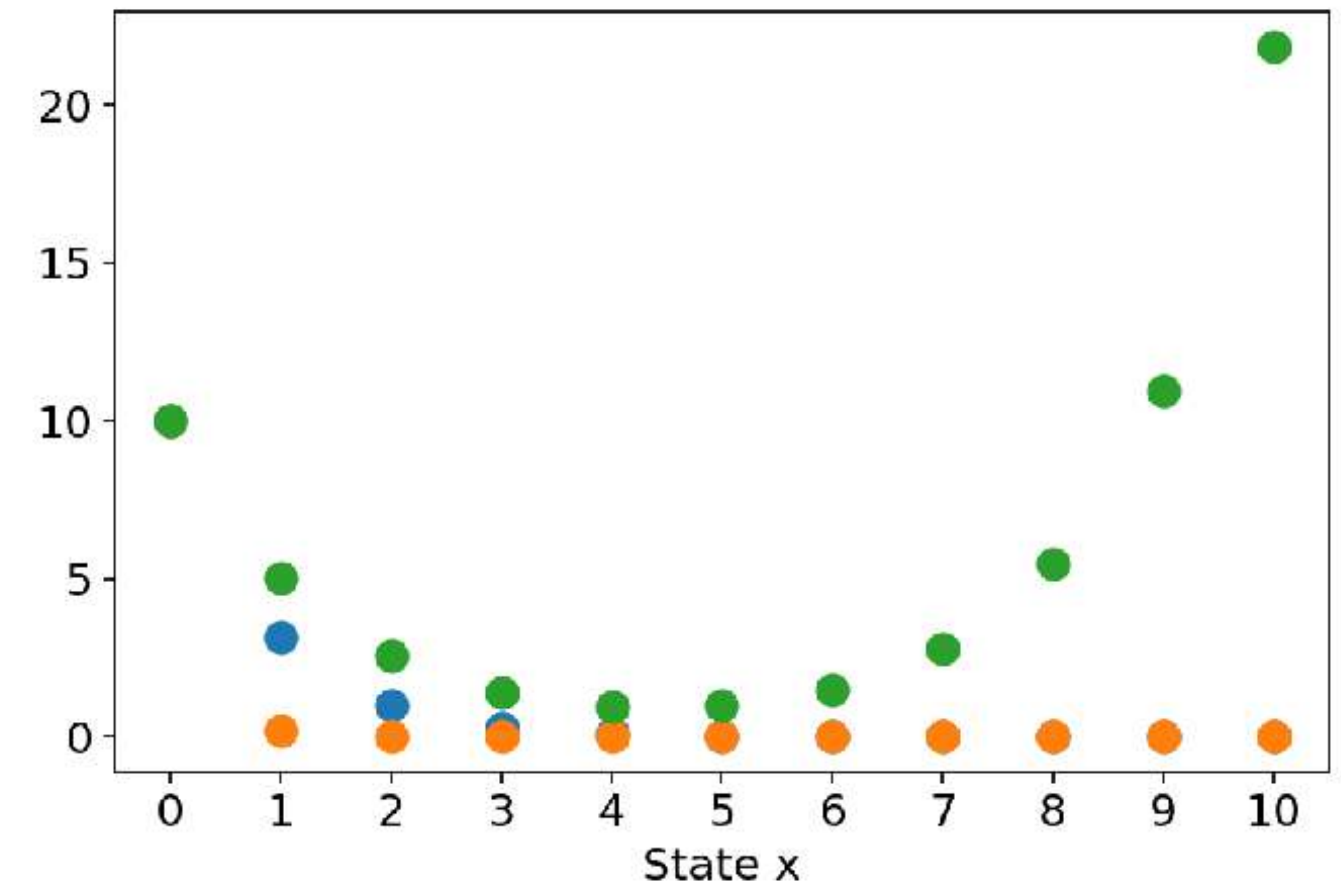


Example

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

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

$N = 10 \quad x_0 = 10$

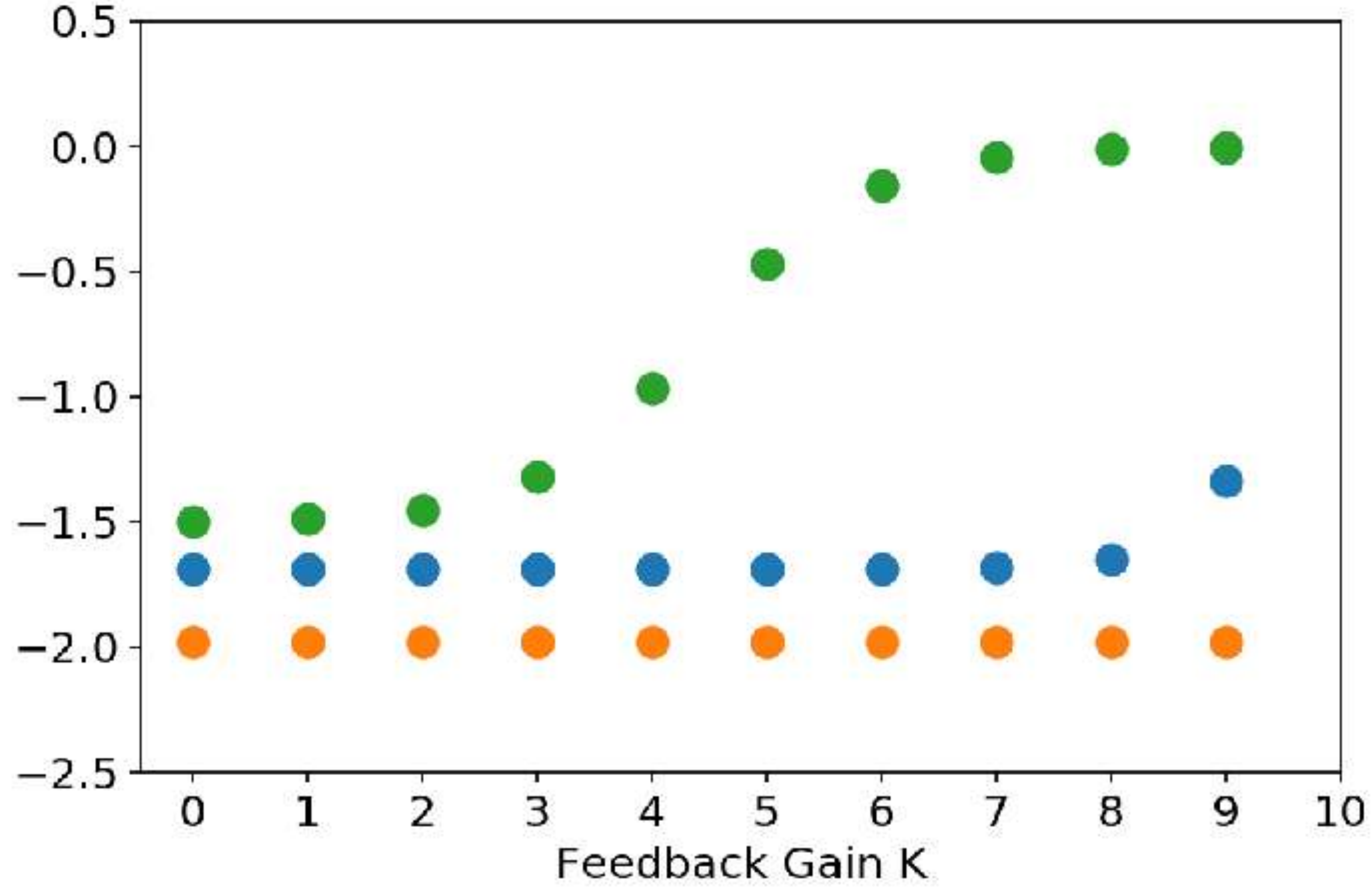


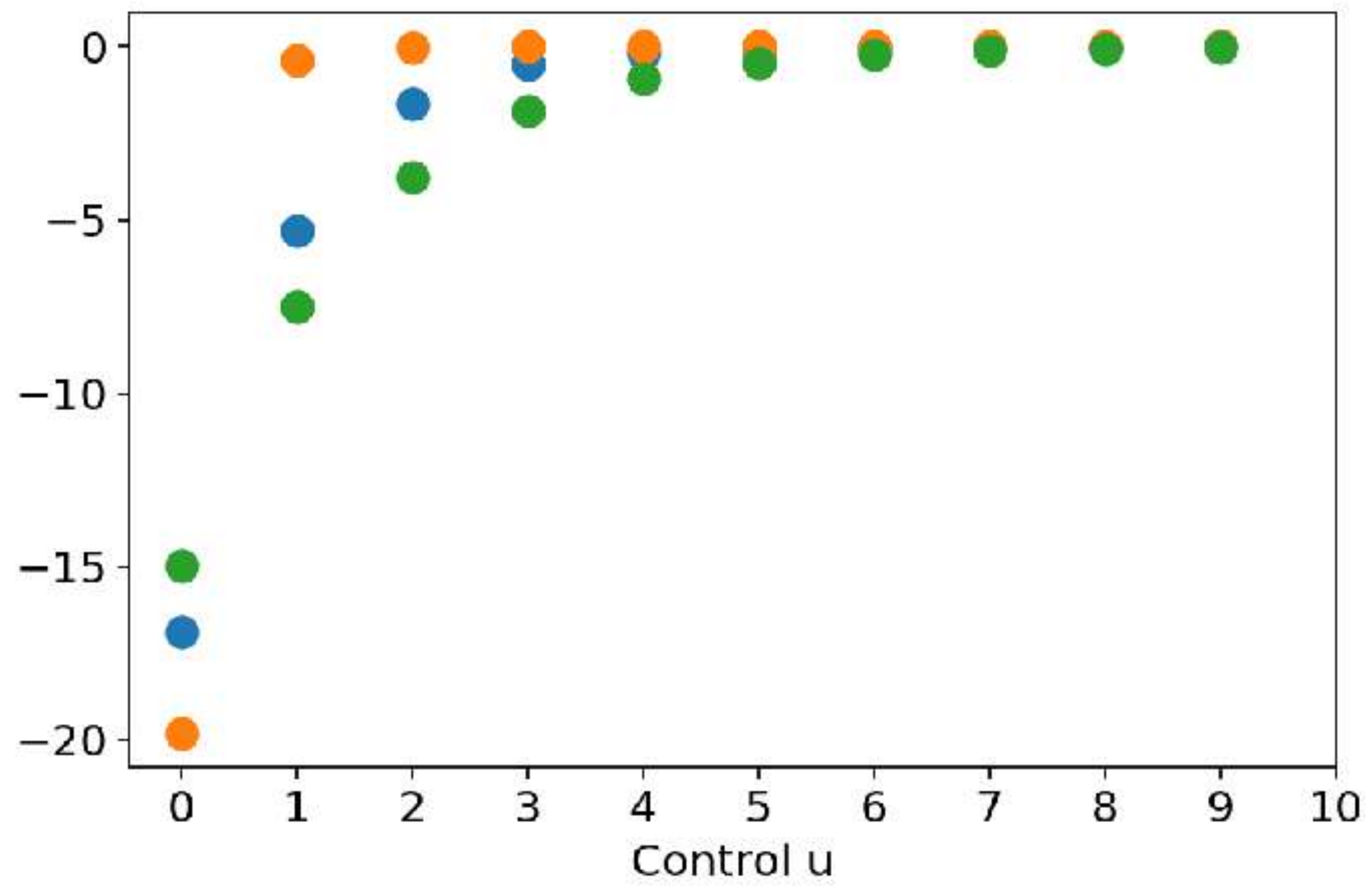
$Q = 2 \; R = 1$

$Q = 100 \; R = 1$

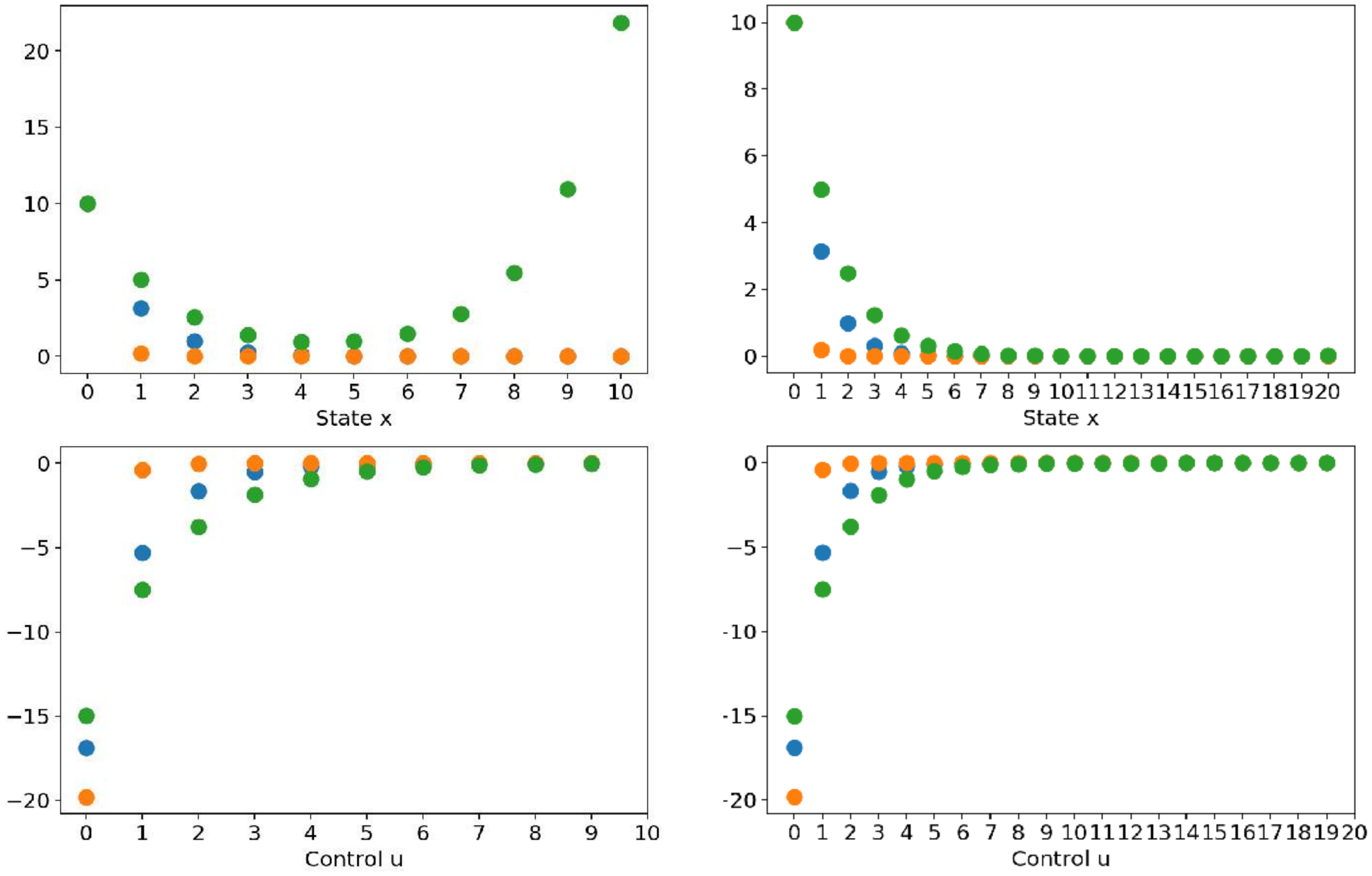
$Q = 1 \; R = 1000$

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





Effect of the horizon N



$N = 10$ $\qquad$ $N = 20$

$Q = 2 \; R = 1$

$Q = 100 \; R = 1$

$Q = 1 \; R = 1000$

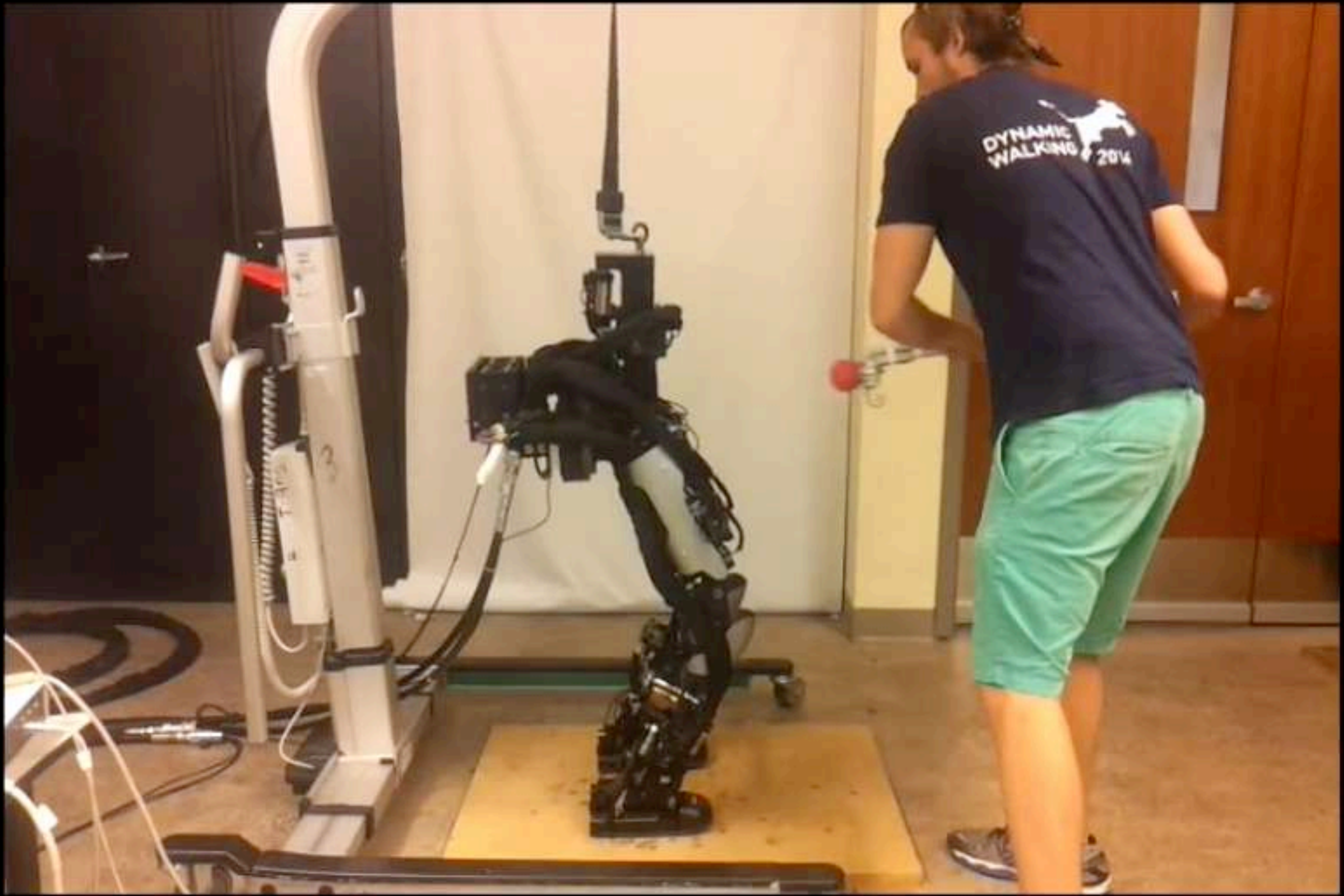
gains seem constant for early stages

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

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

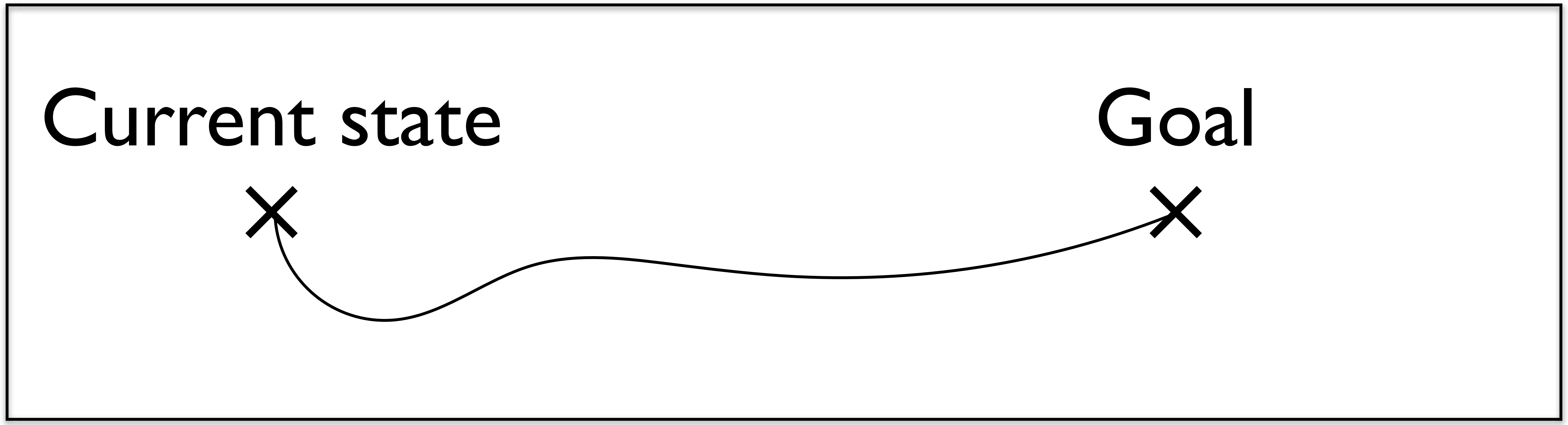
where $P \geq 0$ is called a (convex) quadratic program (QP) with equality constraints.

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

The KKT conditions are

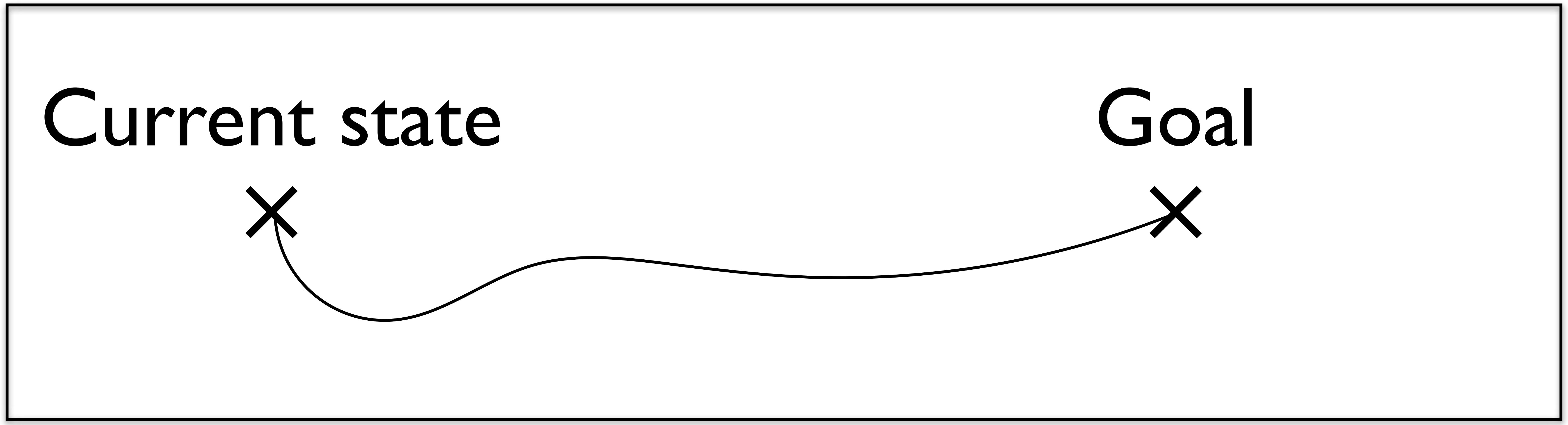
$$\begin{bmatrix} P & A^T \\ A & 0 \end{bmatrix} \begin{pmatrix} x \\ \lambda \end{pmatrix} = \begin{pmatrix} -q \\ b \end{pmatrix}$$

Solving LQ problems with constraints



=> what about inequality constraints?

Solving LQ problems with constraints



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

$$\begin{aligned} \text{subject to} \quad & x_{n+1} = A x_n + B u_n \\ & x_{min} \leq x_n \leq x_{max} \\ & u_{min} \leq u_n \leq u_{max} \\ & x_0 = x_{init} \end{aligned}$$

Solving LQ problems with constraints

Any optimization problem of the form

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

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

Using a QP solver

CVXOPT

Search docs

Copyright and license
Download
Installation instructions
Documentation
Examples
Applications and extensions

Docs » Home

C V X O P T

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

$$\begin{array}{ll} \text{minimize} & (1/2)x^T P x + q^T x \\ \text{subject to} & Gx \preceq h \\ & Ax = b \end{array}$$

Using a QP solver

qpsolvers 4.3.3

```
pip install qpsolvers
```

✓ Latest version

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:

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

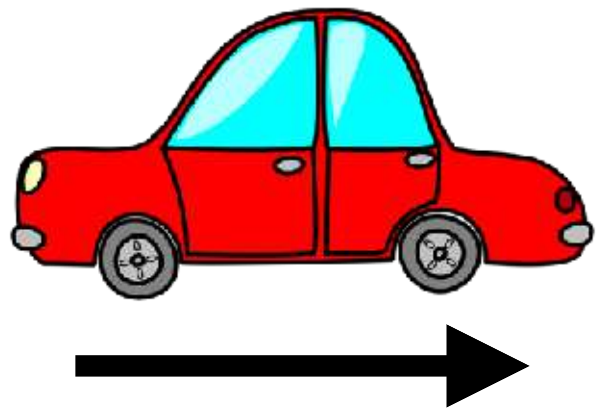
Vector inequalities apply coordinate by coordinate. The function returns the primal solution x^* found by the backend QP solver, or `None` in case of failure/unfeasible problem. All solvers require the problem to be convex, meaning the matrix P should be positive semi-definite. Some solvers further require 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
Clarabel	`clarabel`	Interior point	Sparse	Apache-2.0
CVXOPT	`cvxopt`	Interior point	Dense	GPL-3.0
DAQP	`daqp`	Active set	Dense	MIT
ECOS	`ecos`	Interior point	Sparse	GPL-3.0
Gurobi	`gurobi`	Interior point	Sparse	Commercial
HiGHS	`highs`	Active set	Sparse	MIT
HPIPM	`hpipm`	Interior point	Dense	BSD-2-Clause
MOSEK	`mosek`	Interior point	Sparse	Commercial
NPPro	`nppro`	Active set	Dense	Commercial
OSQP	`osqp`	Augmented Lagrangian	Sparse	Apache-2.0
PIQP	`piqp`	Proximal Interior Point	Dense & Sparse	BSD-2-Clause
ProxQP	`proxqp`	Augmented Lagrangian	Dense & Sparse	BSD-2-Clause
QPALM	`qpalm`	Augmented Lagrangian	Sparse	LGPL-3.0
qpOASES	`qpoases`	Active set	Dense	LGPL-2.1
qpSWIFT	`qpswift`	Interior point	Sparse	GPL-3.0
quadprog	`quadprog`	Active set	Dense	GPL-2.0
SCS	`scs`	Augmented Lagrangian	Sparse	MIT

Using a QP solver

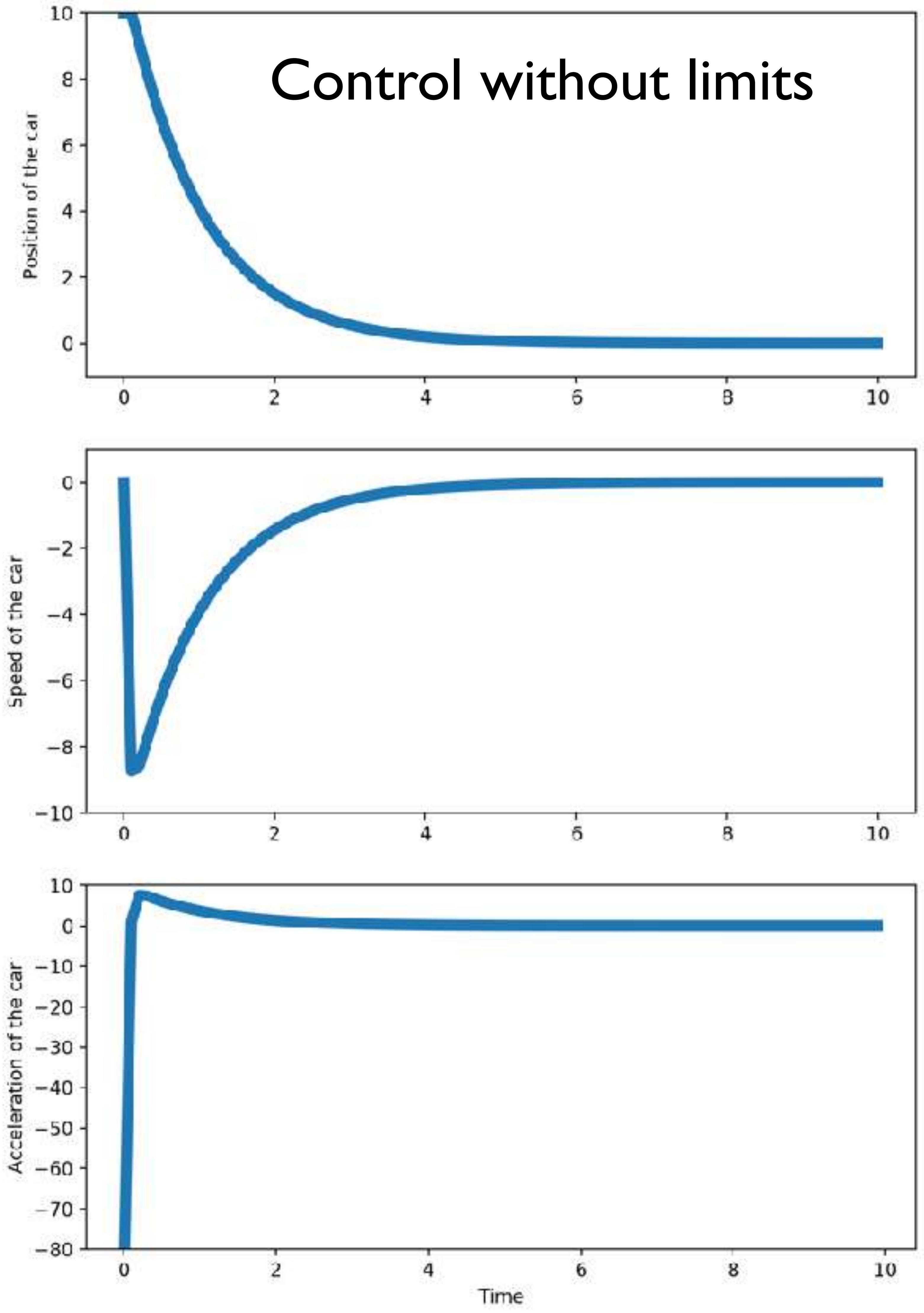


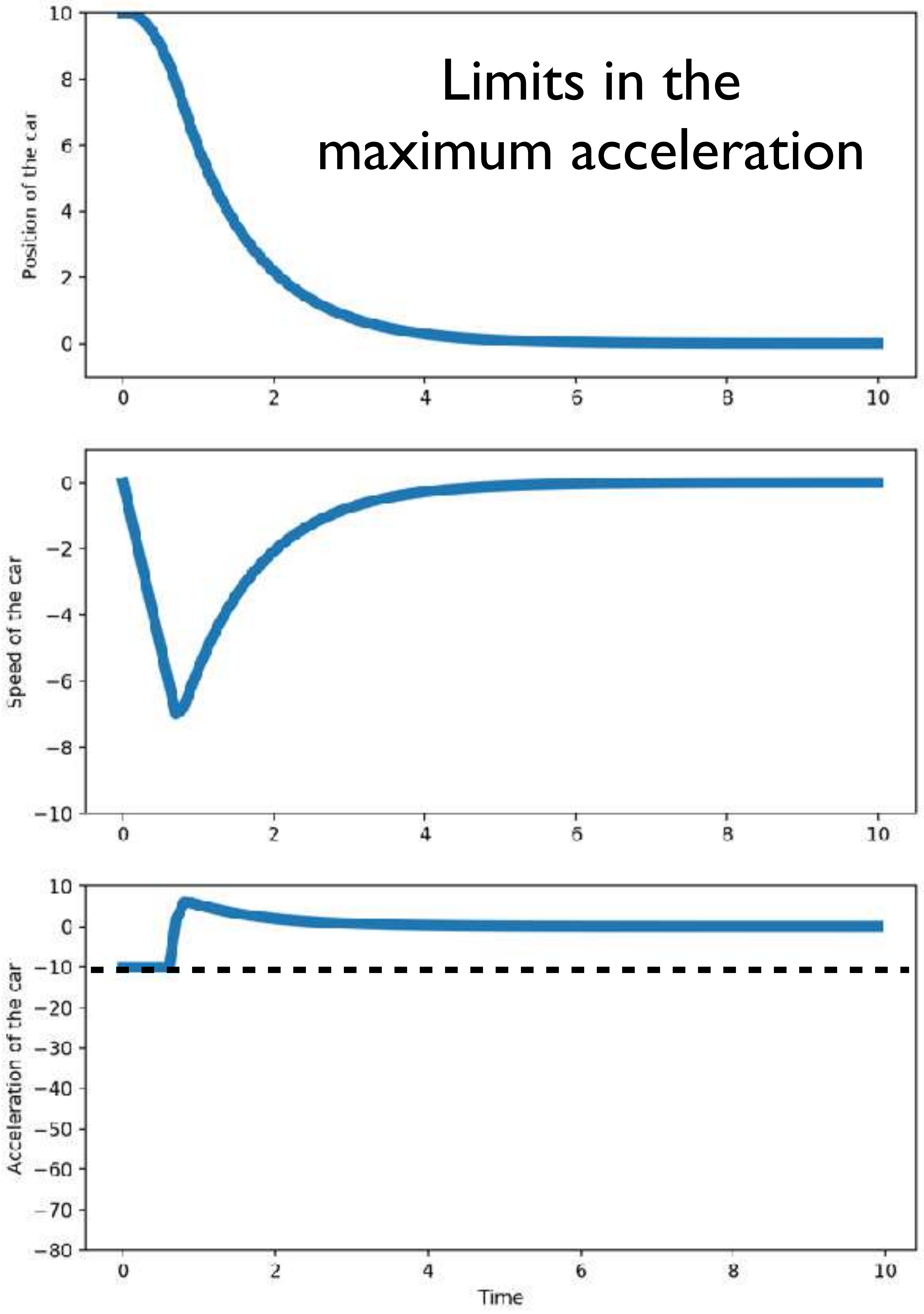
Goal

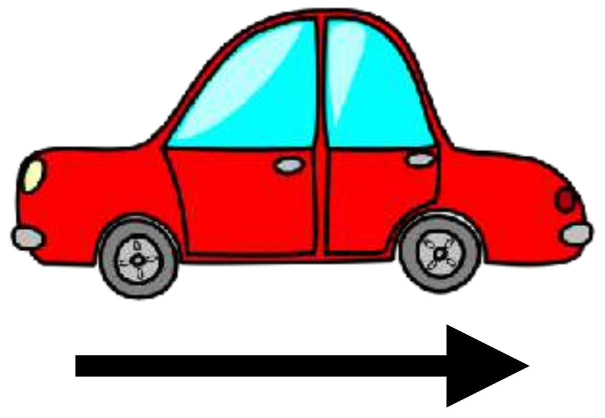
$$x_{n+1} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} x_n + \begin{bmatrix} 0 \\ \Delta t \end{bmatrix} u_n$$

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

$$R = \begin{bmatrix} 0.1 \end{bmatrix}$$



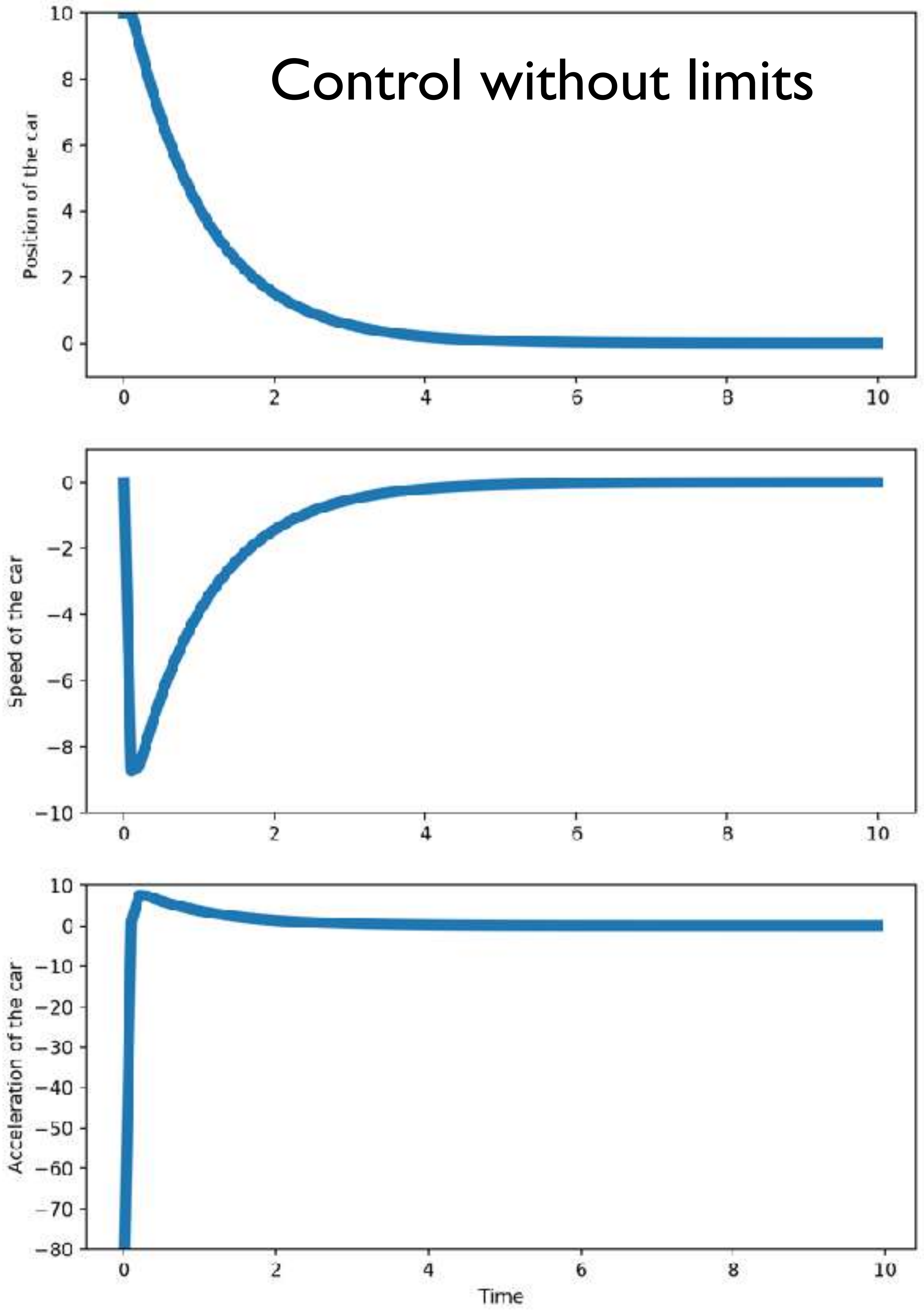


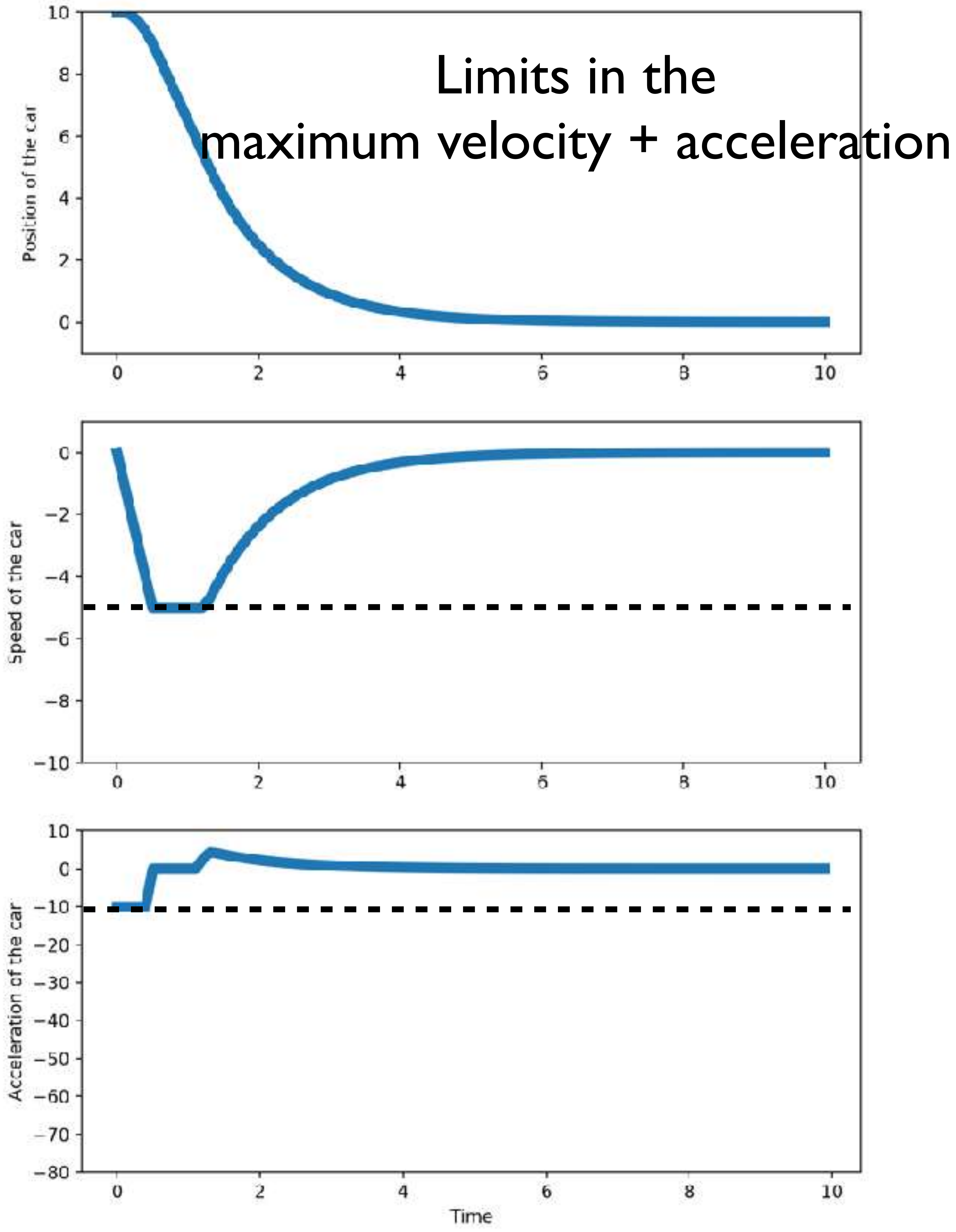


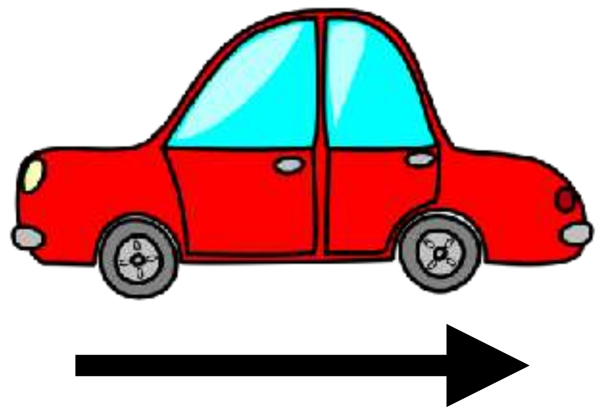
Goal

$$x_{n+1} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} x_n + \begin{bmatrix} 0 \\ \Delta t \end{bmatrix} u_n \qquad Q = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix}$$

$$R = \begin{bmatrix} 0.1 \end{bmatrix}$$

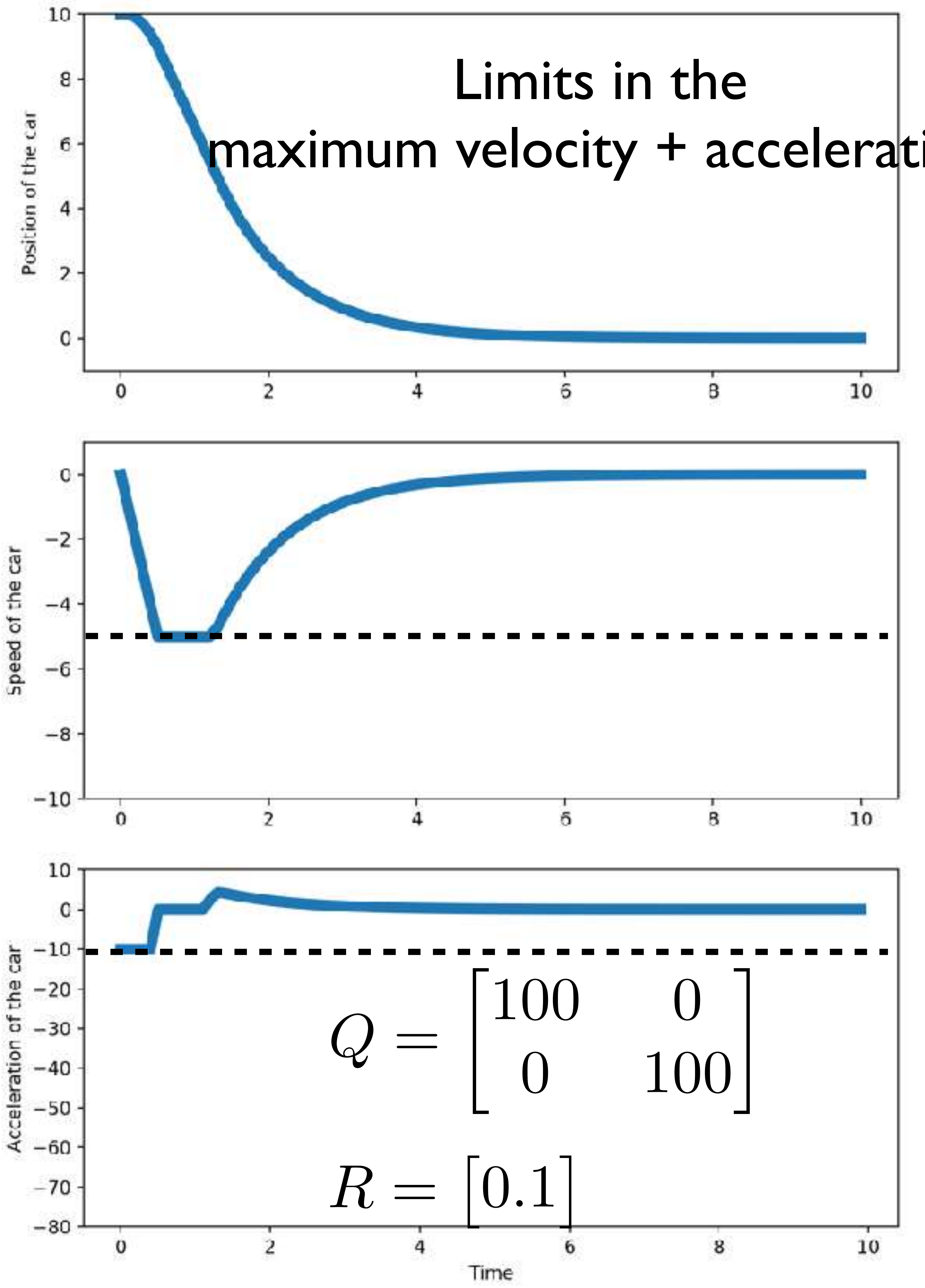






Goal ✕

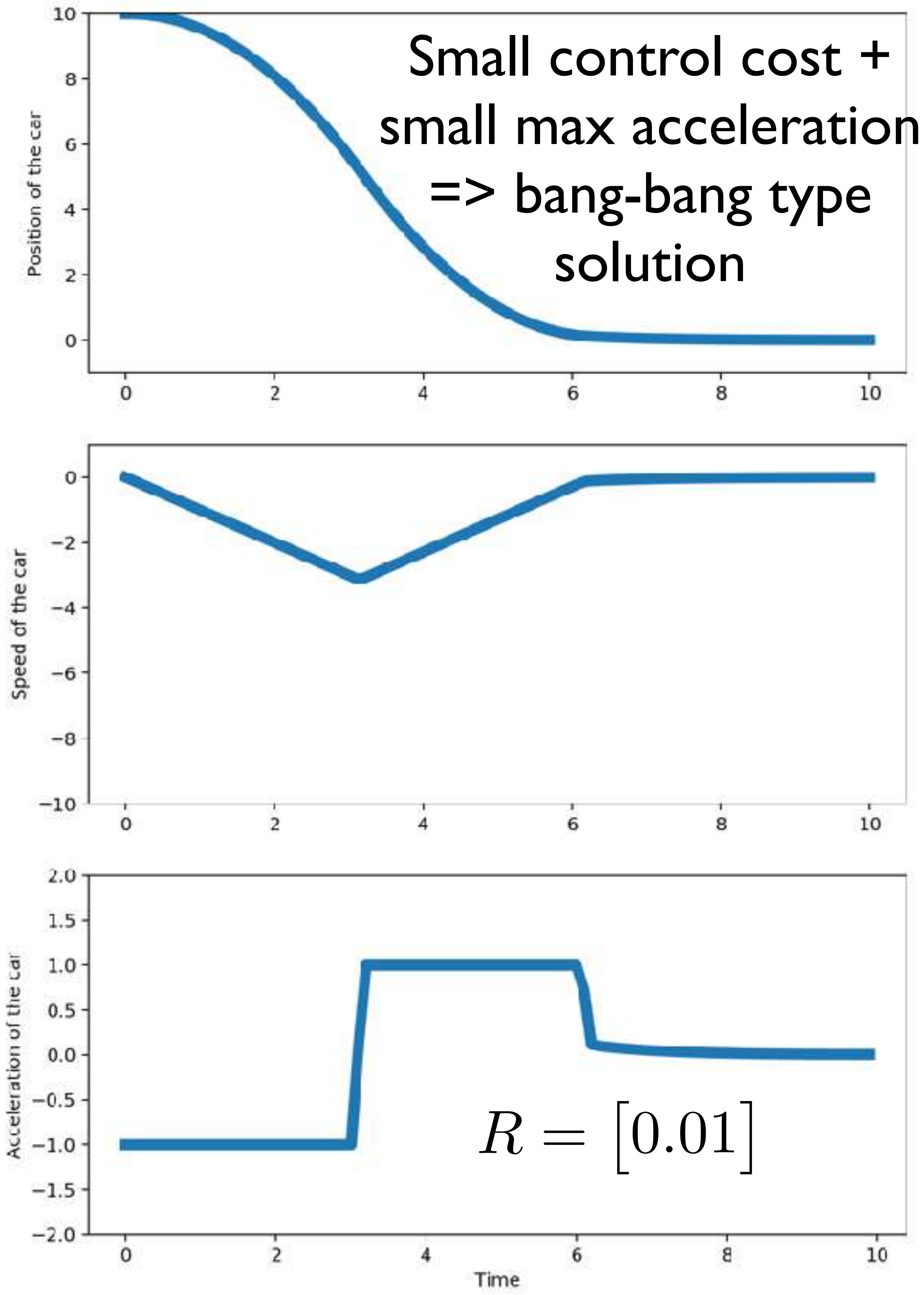
$$x_{n+1} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} x_n + \begin{bmatrix} 0 \\ \Delta t \end{bmatrix} u_n$$



Limits in the maximum velocity + acceleration

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

$$R = \begin{bmatrix} 0.1 \end{bmatrix}$$



Small control cost + small max acceleration => bang-bang type solution

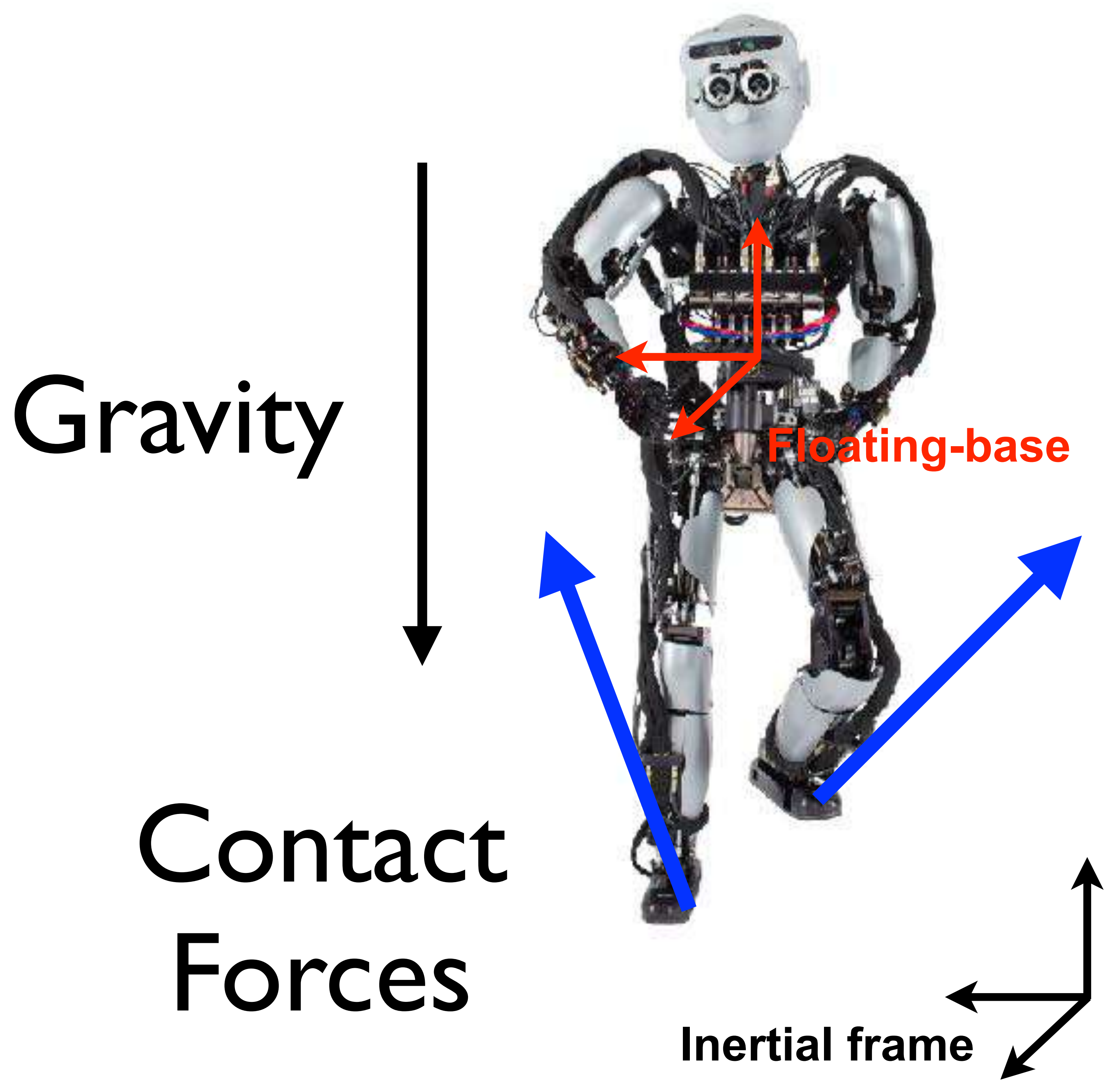
$$R = \begin{bmatrix} 0.01 \end{bmatrix}$$

Example: biped walking

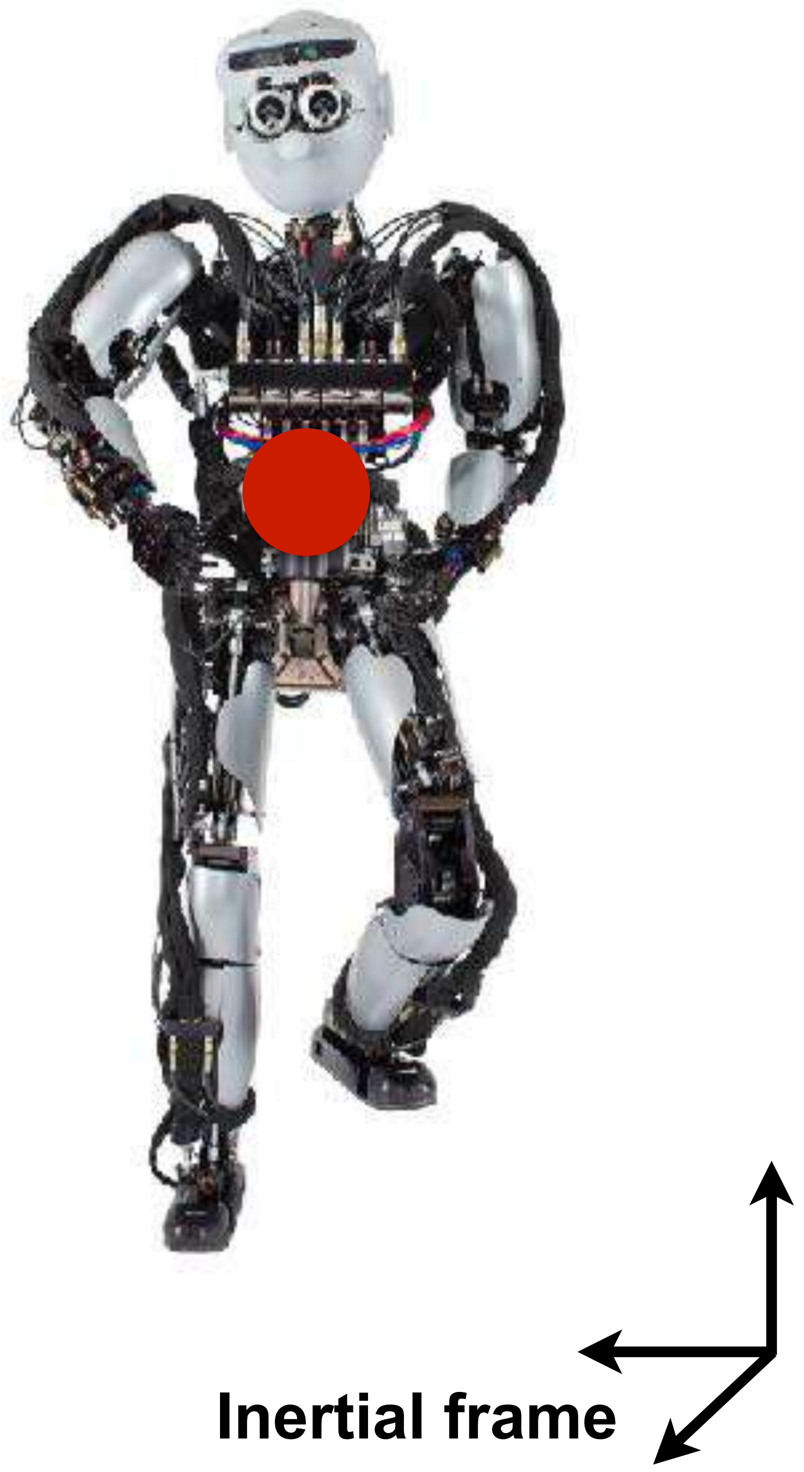
What is the problem?

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

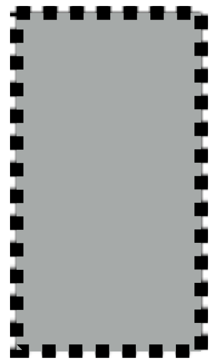


Motion of the center of mass

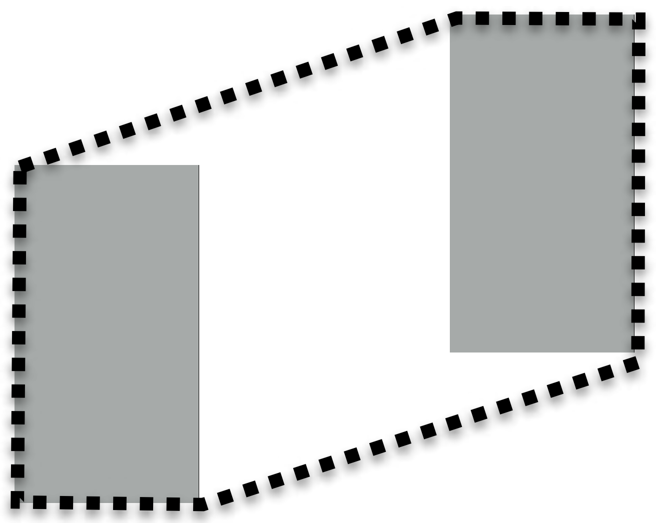


Support polygon

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



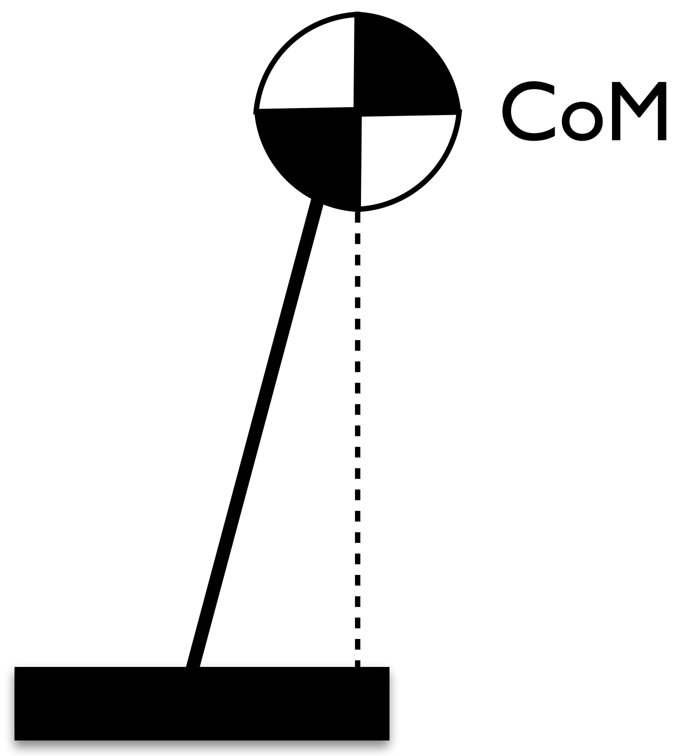
One foot on the ground and support polygon (top view)



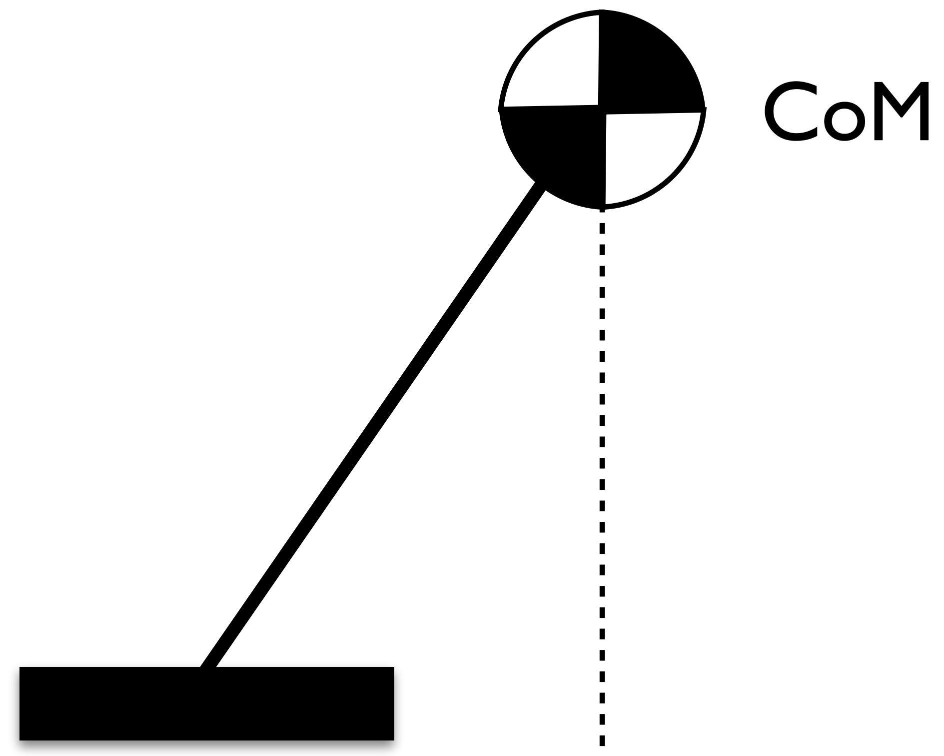
Two feet on the ground and support polygon (top view)

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

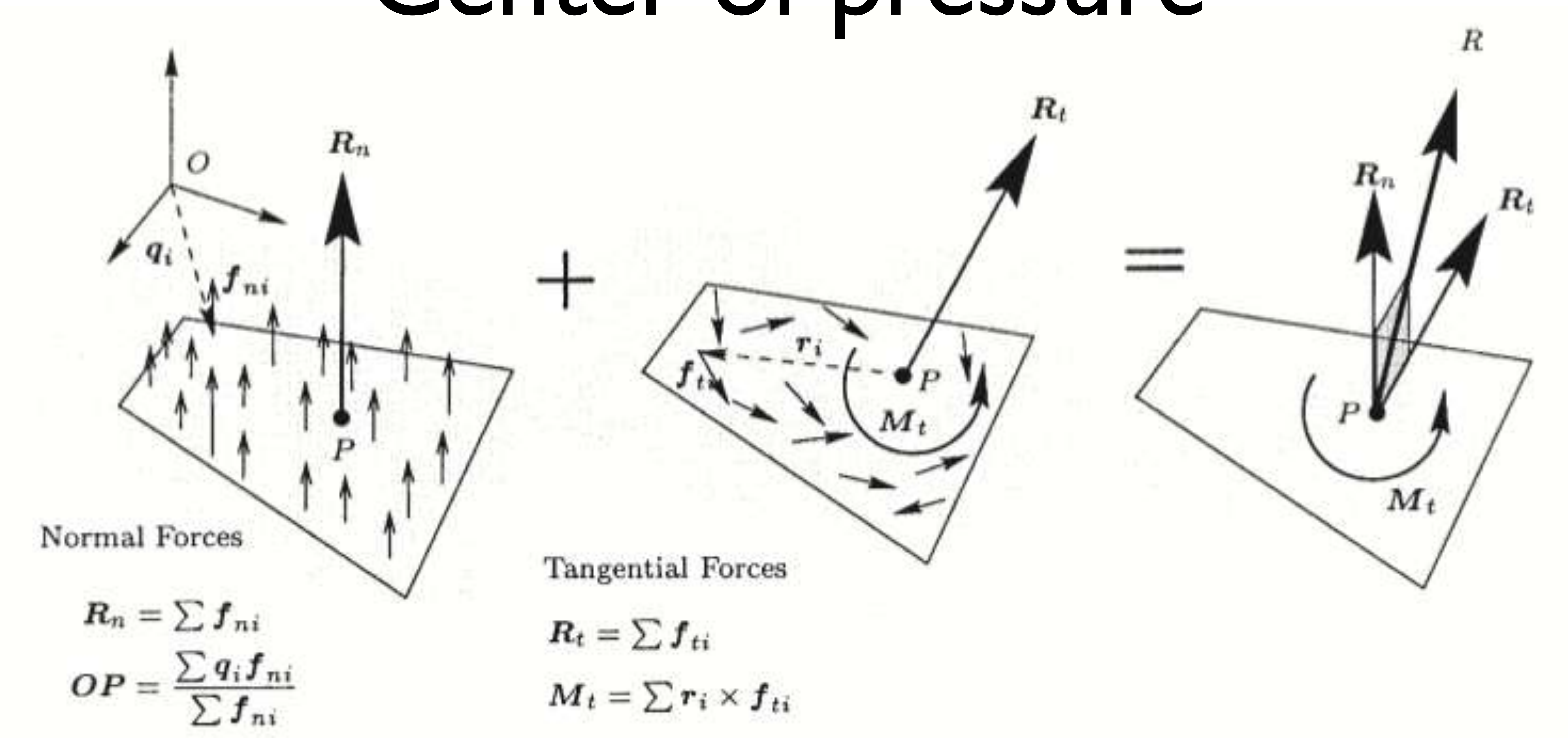


unstable (the foot will rotate around its edge)

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



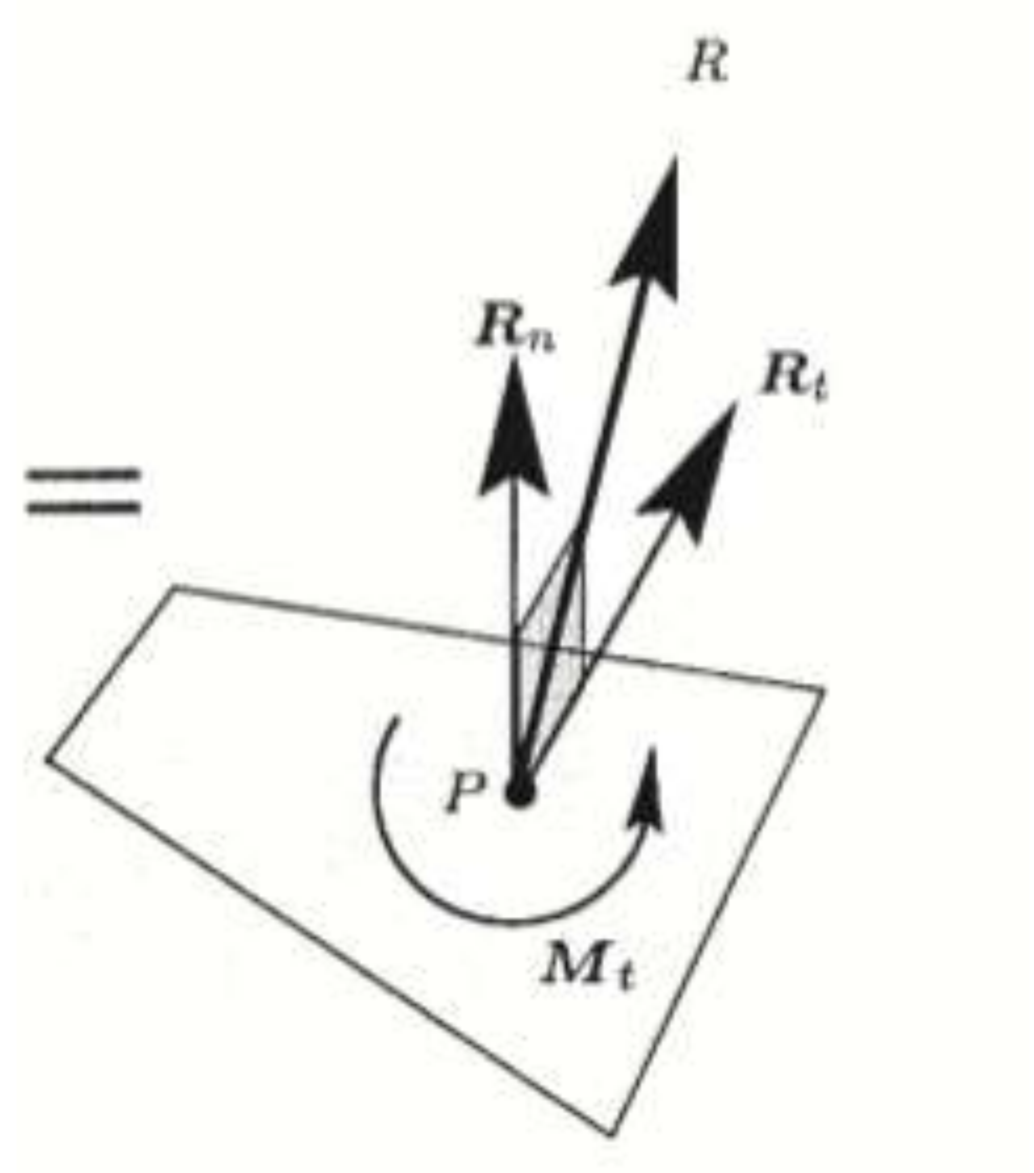
The center of pressure (CoP) is the point of the ground where the resultant force Rn acts

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

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)

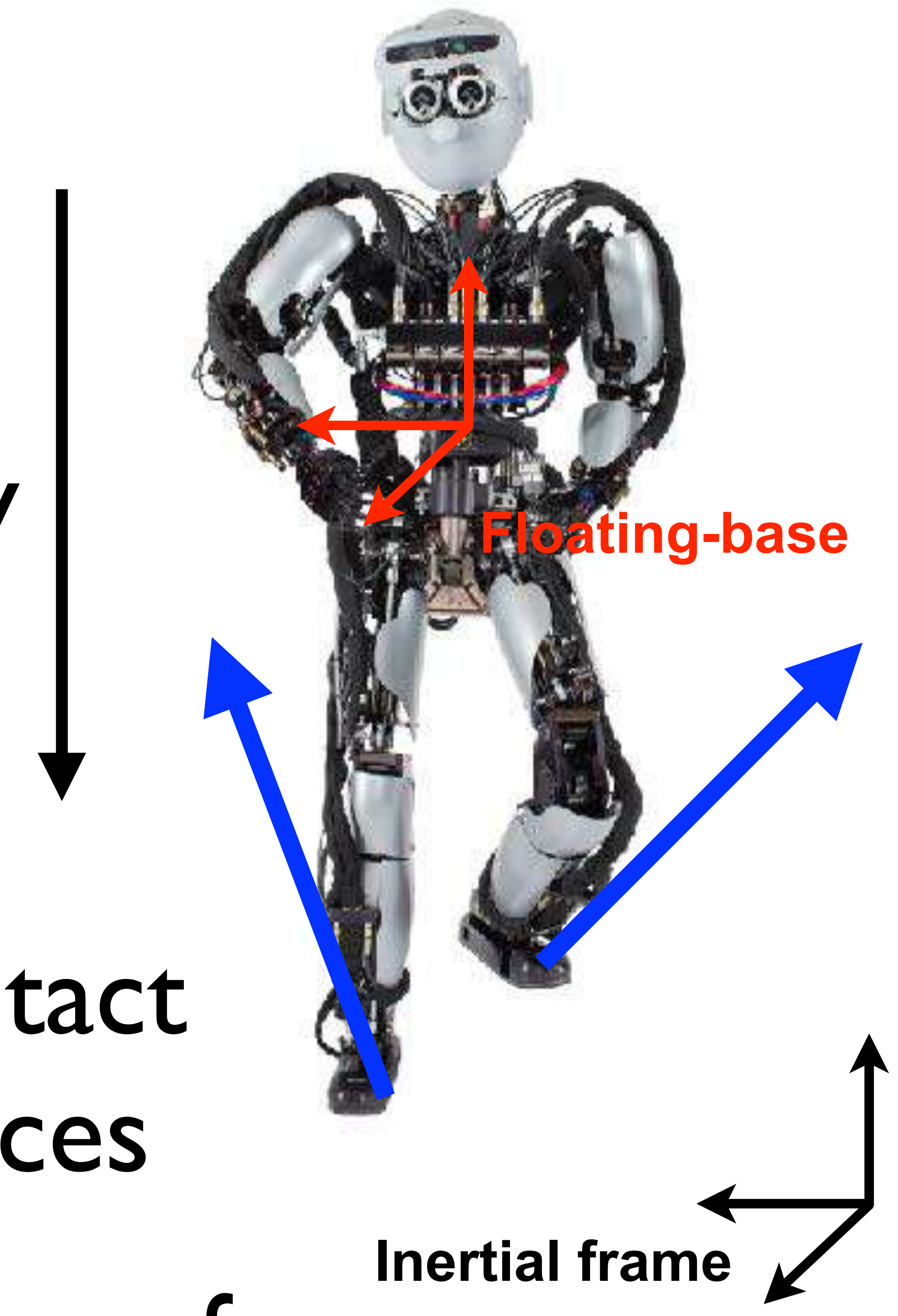
2. Compute desired motion of the center of mass compatible with the physics (OC problem)

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

Gravity

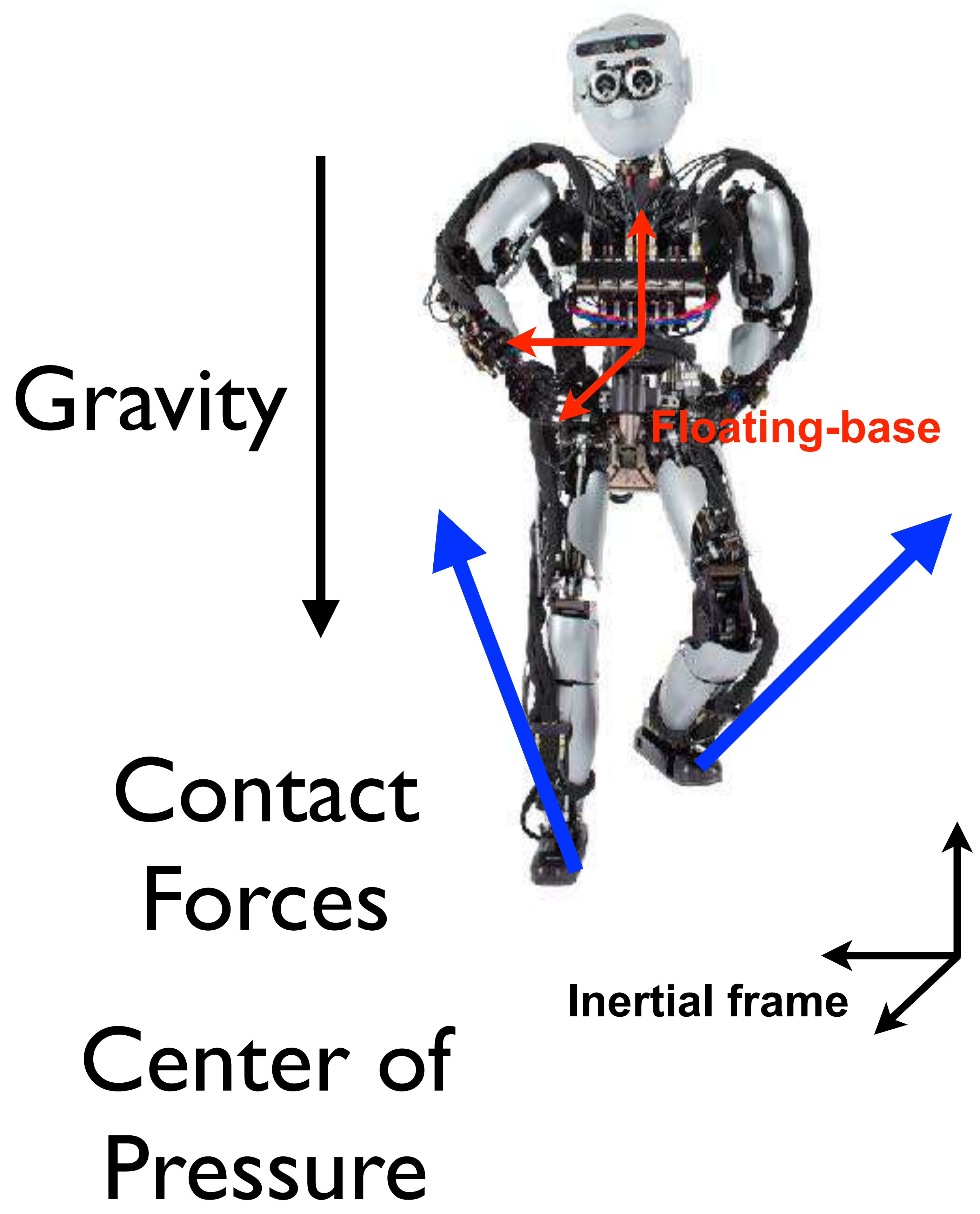


Contact Forces

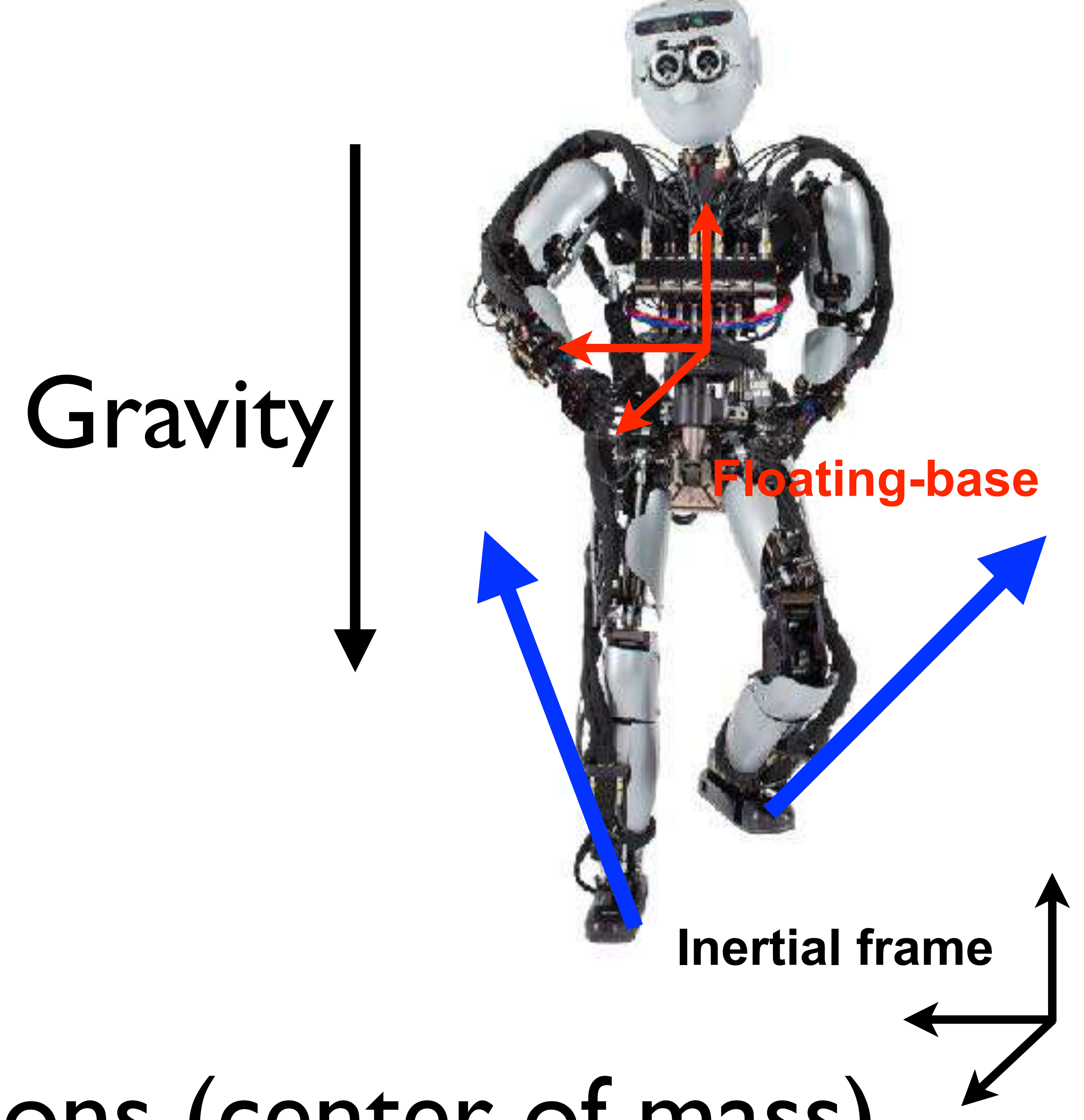
Center of Pressure

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



$$m\ddot{\mathbf{c}} = \sum_i \mathbf{f}_i - m\mathbf{g}$$ Newton equations (center of mass)

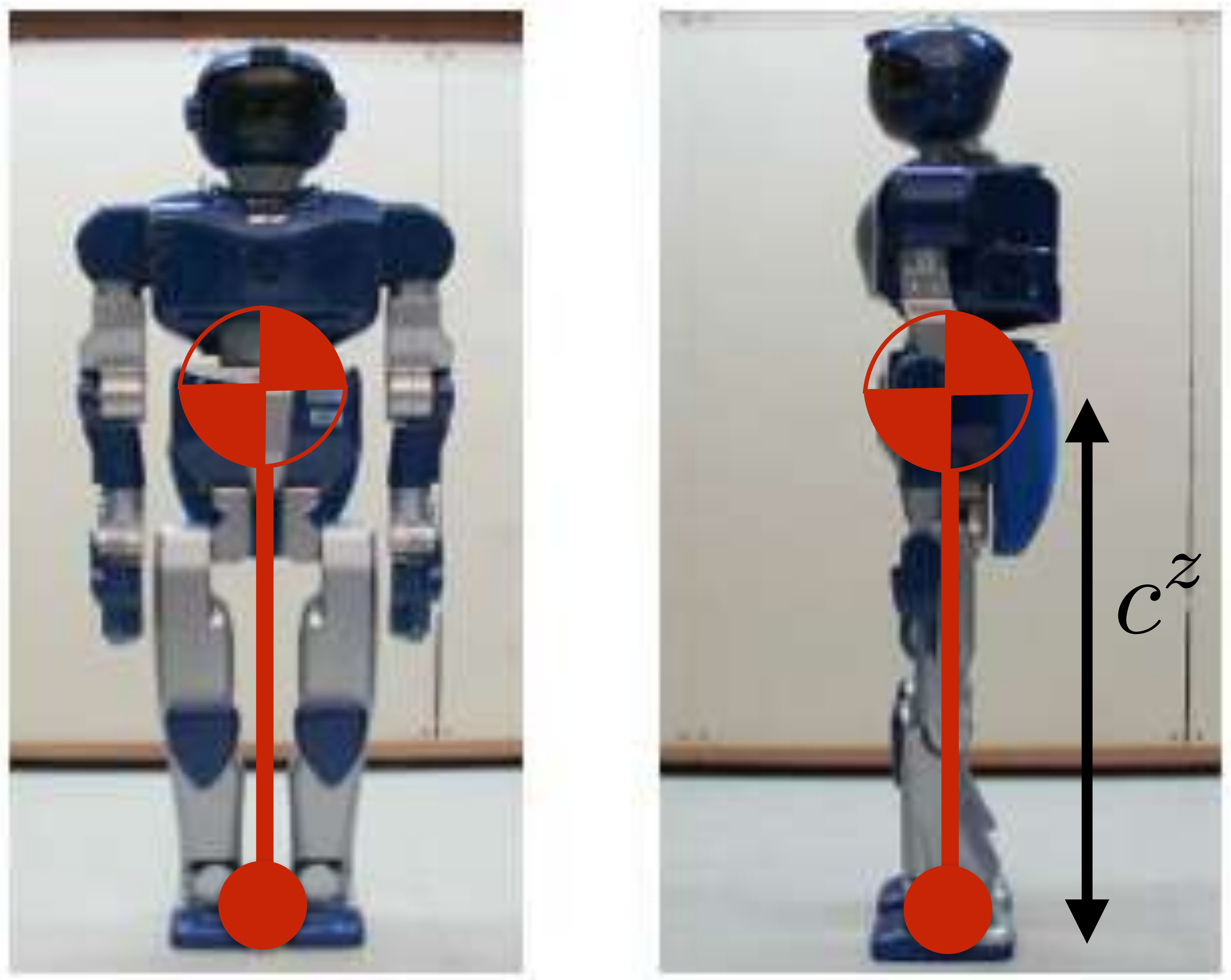
$$\dot{\mathbf{L}} = \sum_i (\mathbf{p}_i - \mathbf{c}) \times \mathbf{f}_i + \boldsymbol{\tau}_i$$ Euler equations (angular momentum)

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

$$\ddot{\mathbf{c}}^{x,y} = \frac{g}{c^z}(\mathbf{c}^{x,y} - \underbrace{\mathbf{p}^{x,y}}_{\text{CoP}})$$

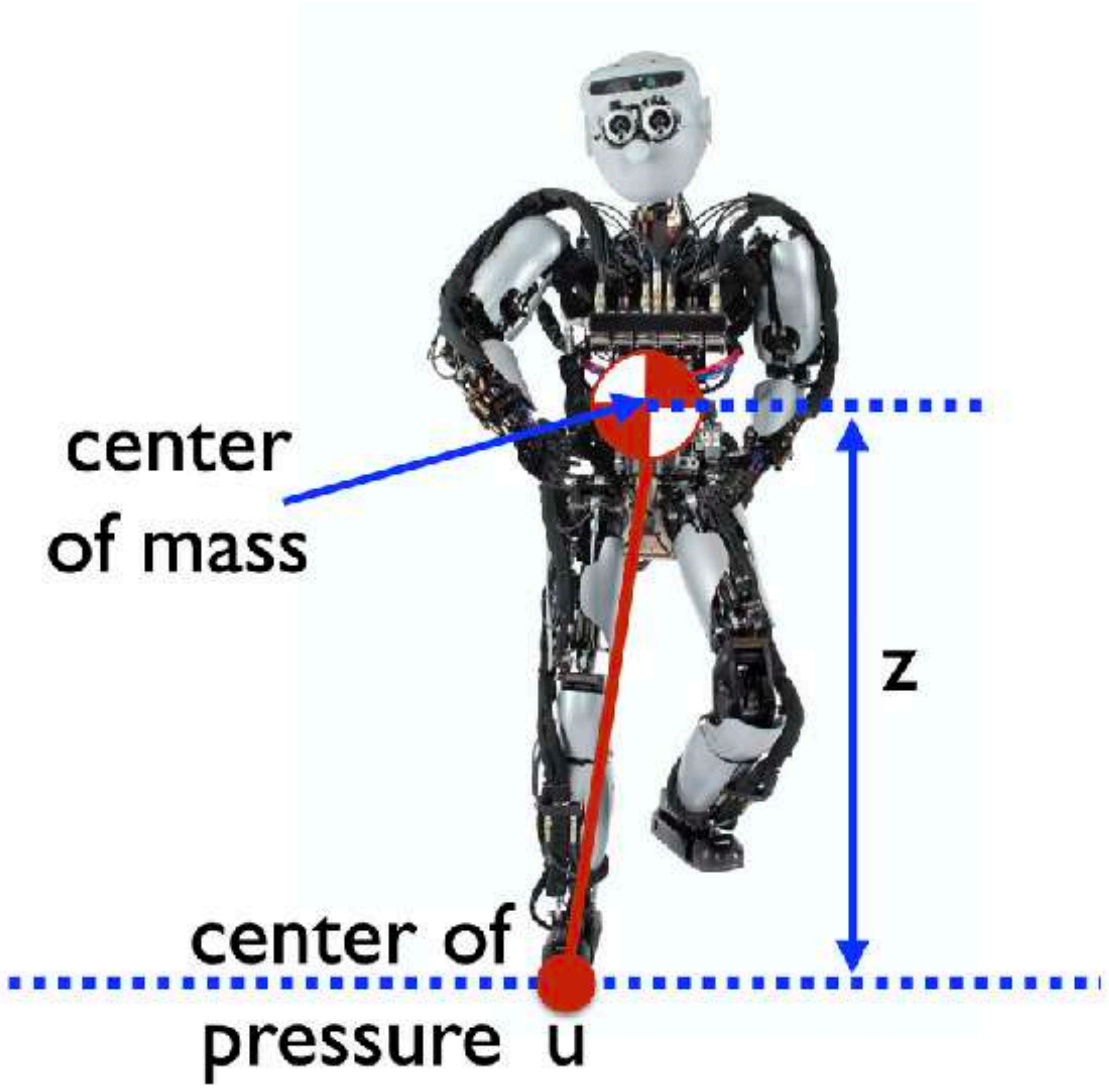


Model predictive control for walking

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)

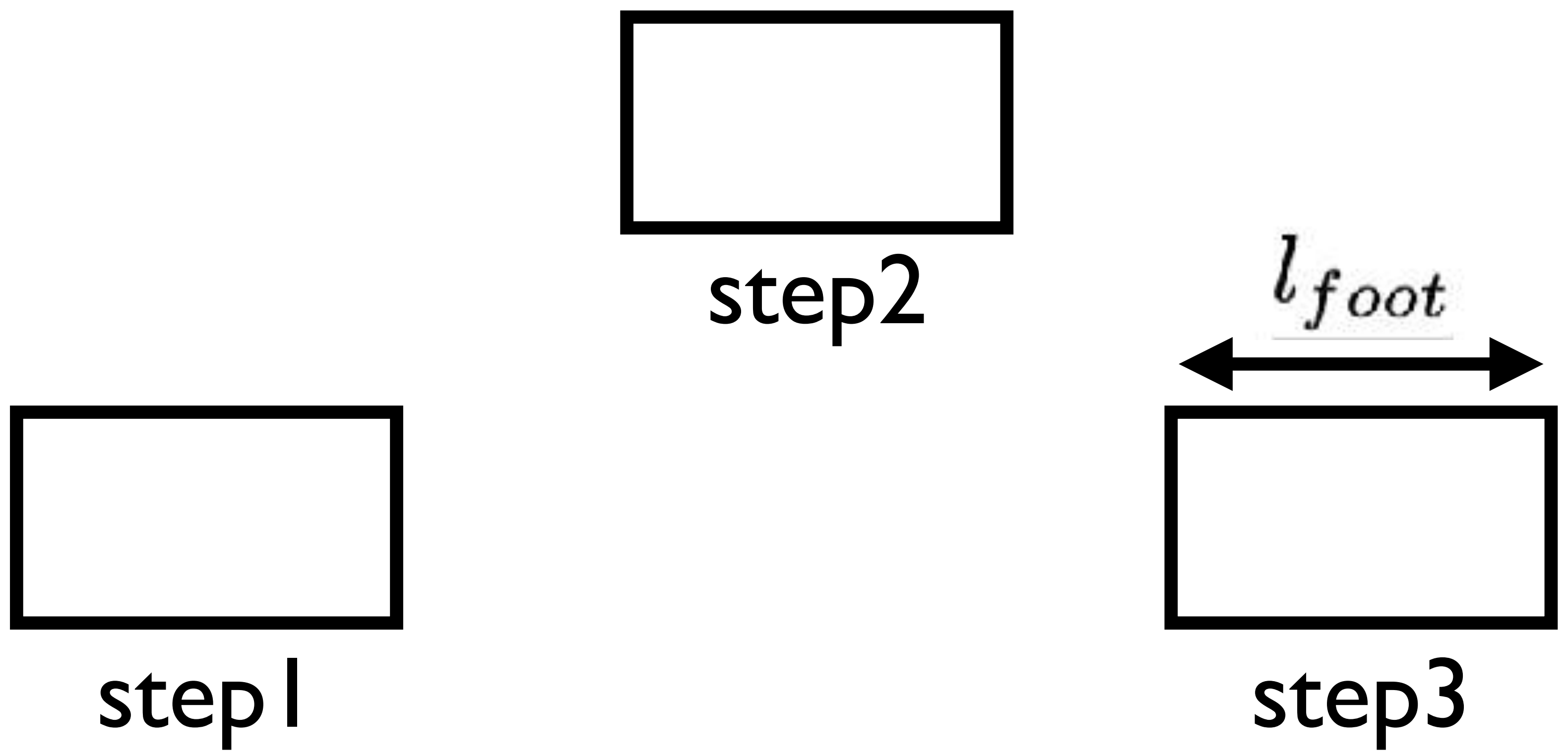
[Kajita et al. 2001]

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

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

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

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]

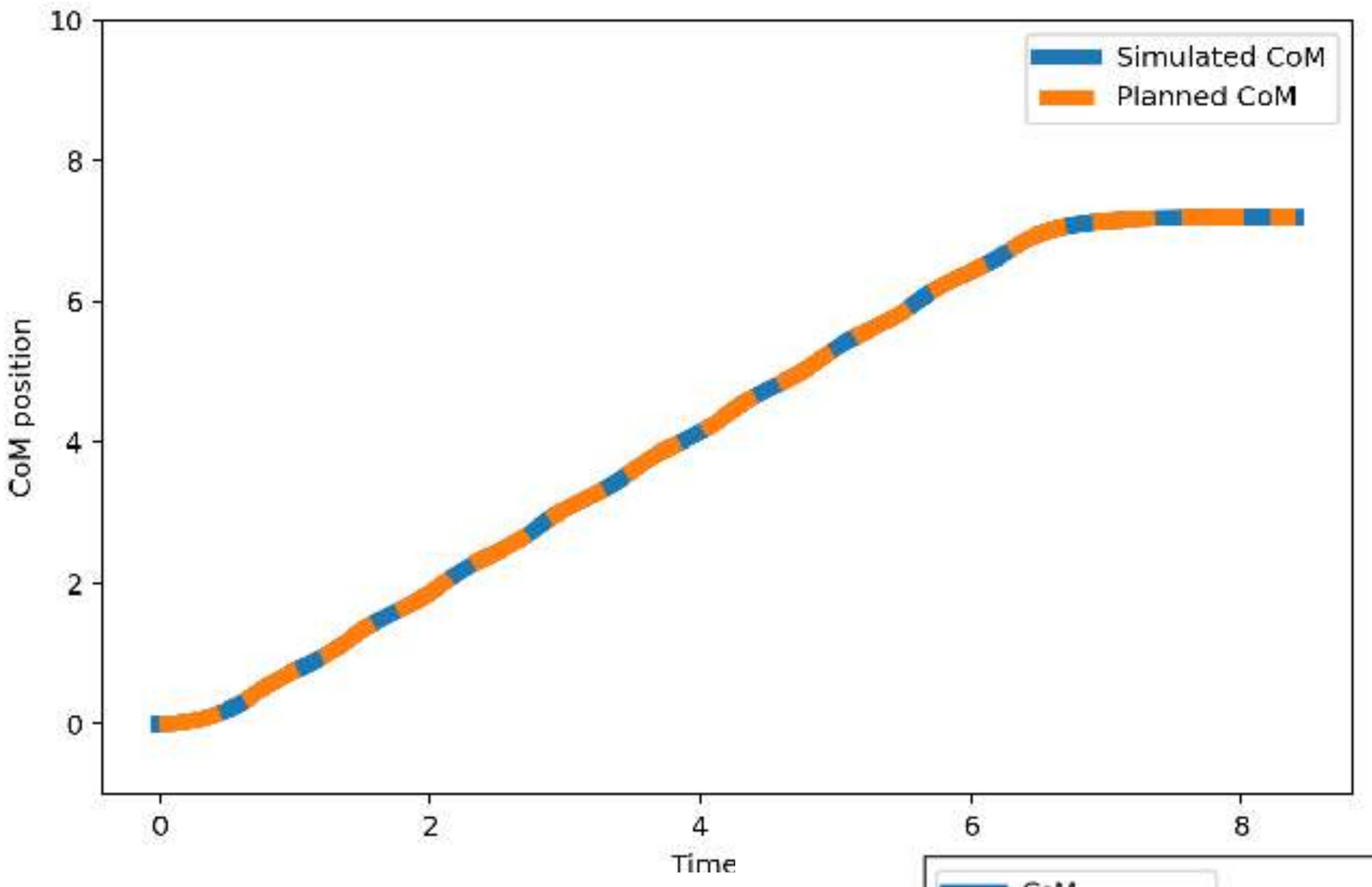
$$\min_{u_n} \sum_{n=0}^{N-1} \begin{pmatrix} c_n - f_n \\ v_n \end{pmatrix}^T Q \begin{pmatrix} c_n - f_n \\ v_n \end{pmatrix} + (p_n - f_n)^T R (p_n - f_n) + \begin{pmatrix} c_N - f_N \\ v_N \end{pmatrix}^T Q_N \begin{pmatrix} c_N - f_N \\ v_N \end{pmatrix}$$

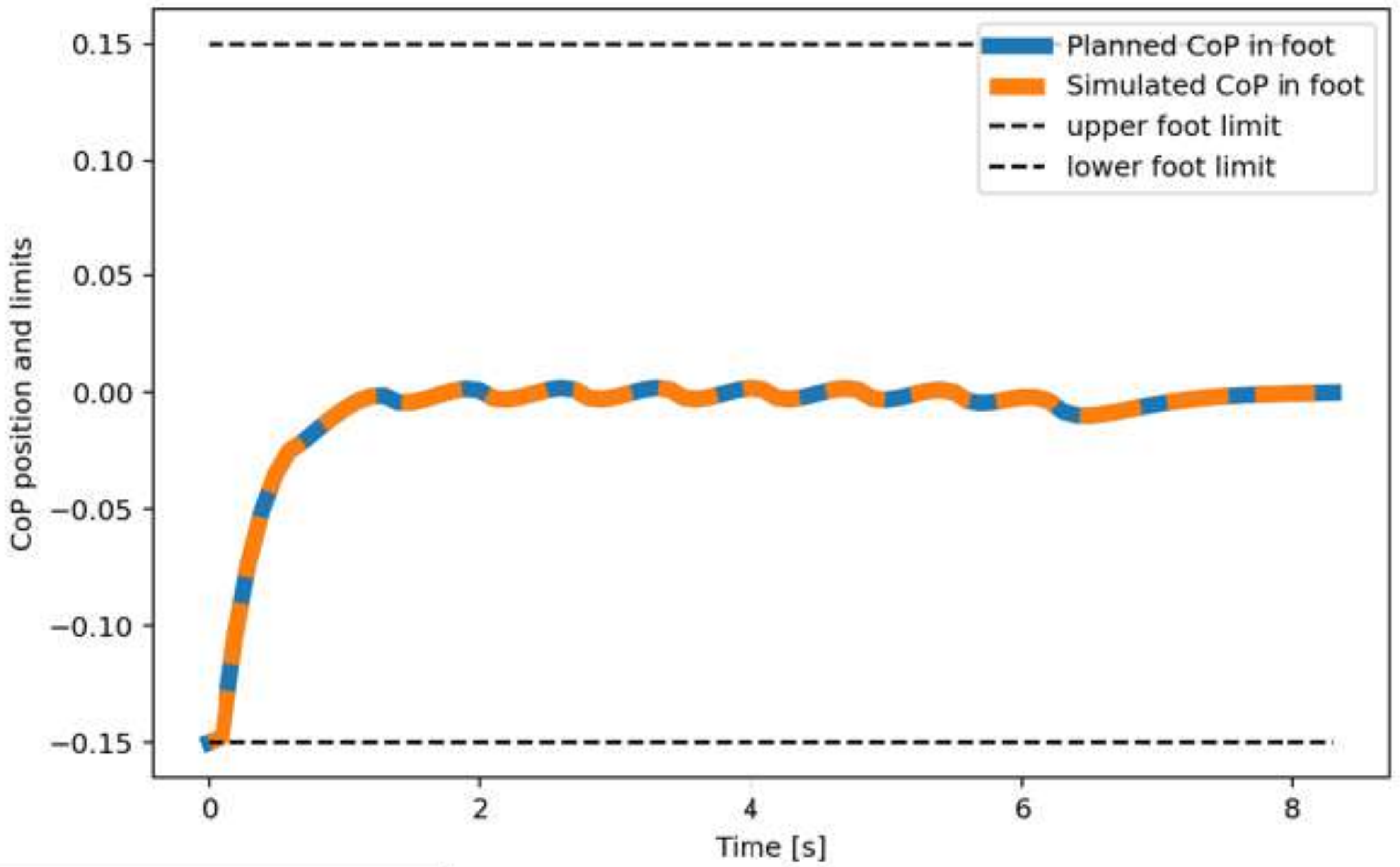
subject to

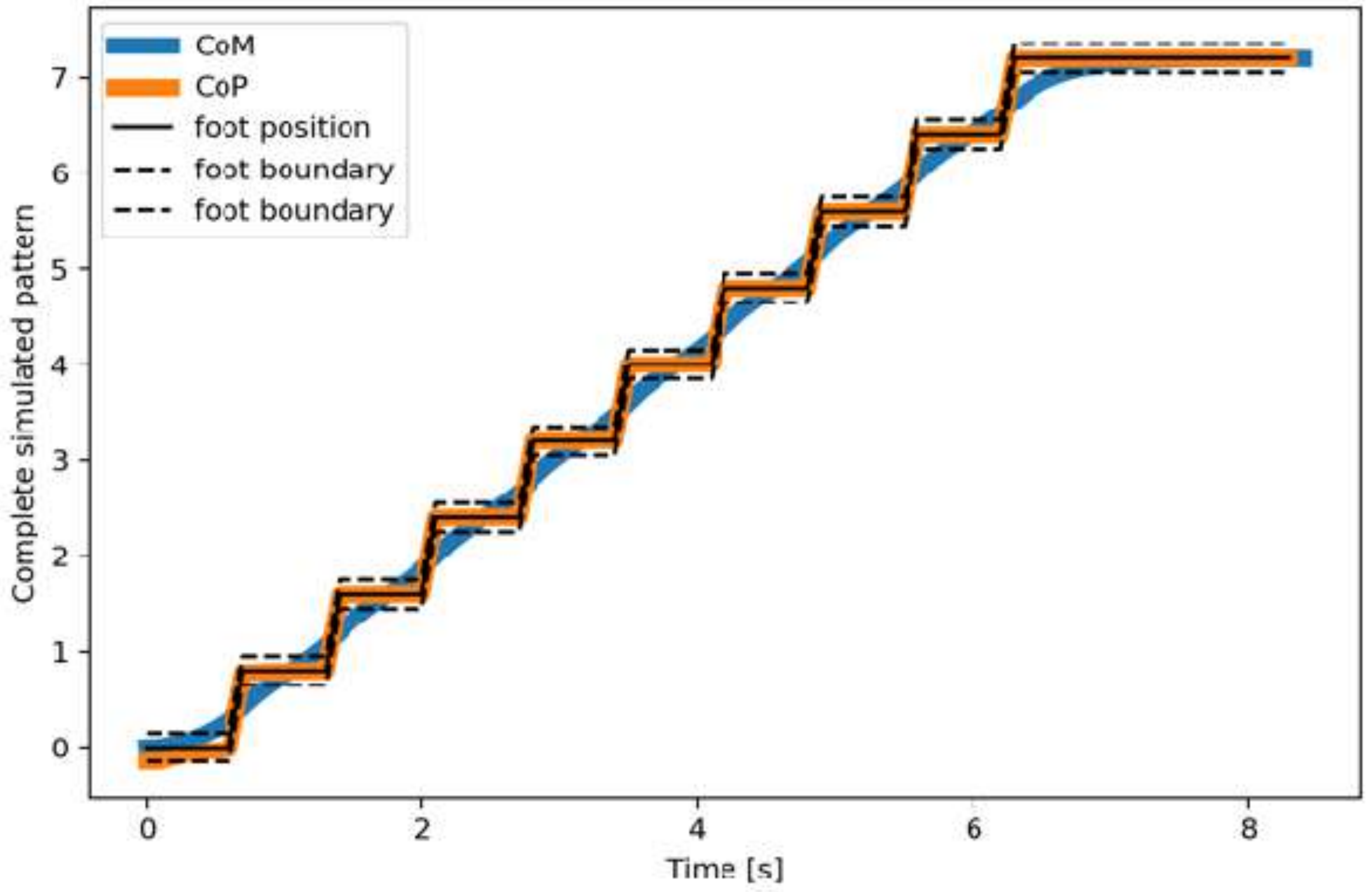
$$\begin{bmatrix} c_{n+1} \\ v_{n+1} \end{bmatrix} = \begin{bmatrix} \cosh(\omega \Delta t) & \omega^{-1} \sinh(\omega \Delta t) \\ \omega \sinh(\omega \Delta t) & \cosh(\omega \Delta t) \end{bmatrix} \begin{bmatrix} c_n \\ v_n \end{bmatrix} + \begin{bmatrix} 1 - \cosh(\omega \Delta t) \\ -\omega \sinh(\omega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

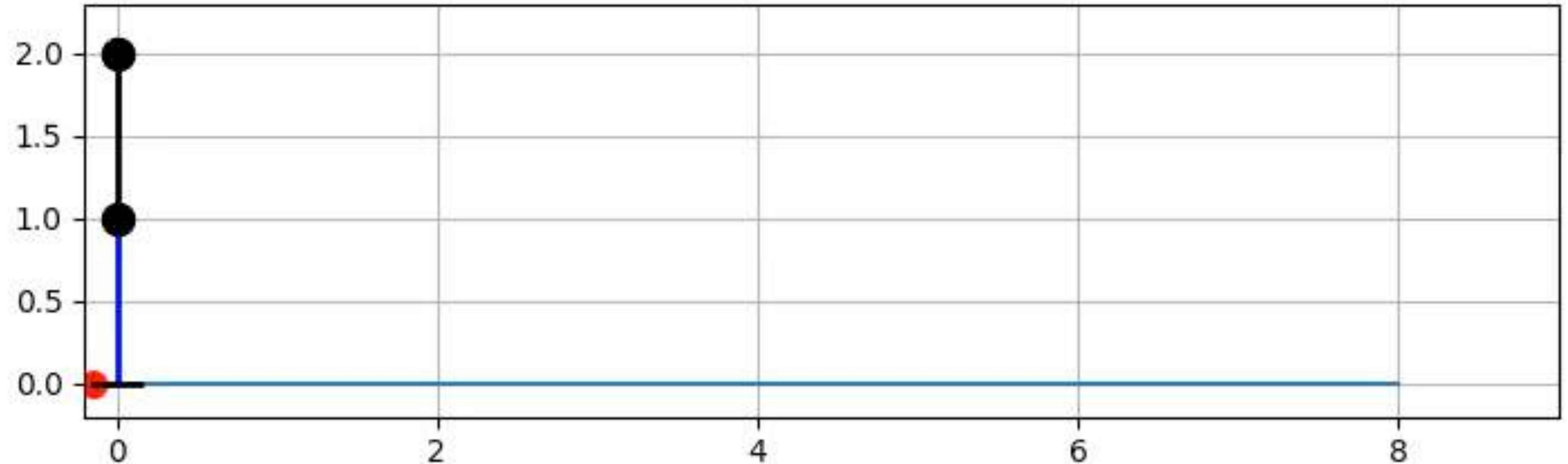
and

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


Simulated CoM
Planned CoM
CoM position
Time


Planned CoP in foot
Simulated CoP in foot
upper foot limit
lower foot limit
CoP position and limits
Time [s]


CoM
CoP
foot position
foot boundary
foot boundary
Complete simulated pattern
Time [s]

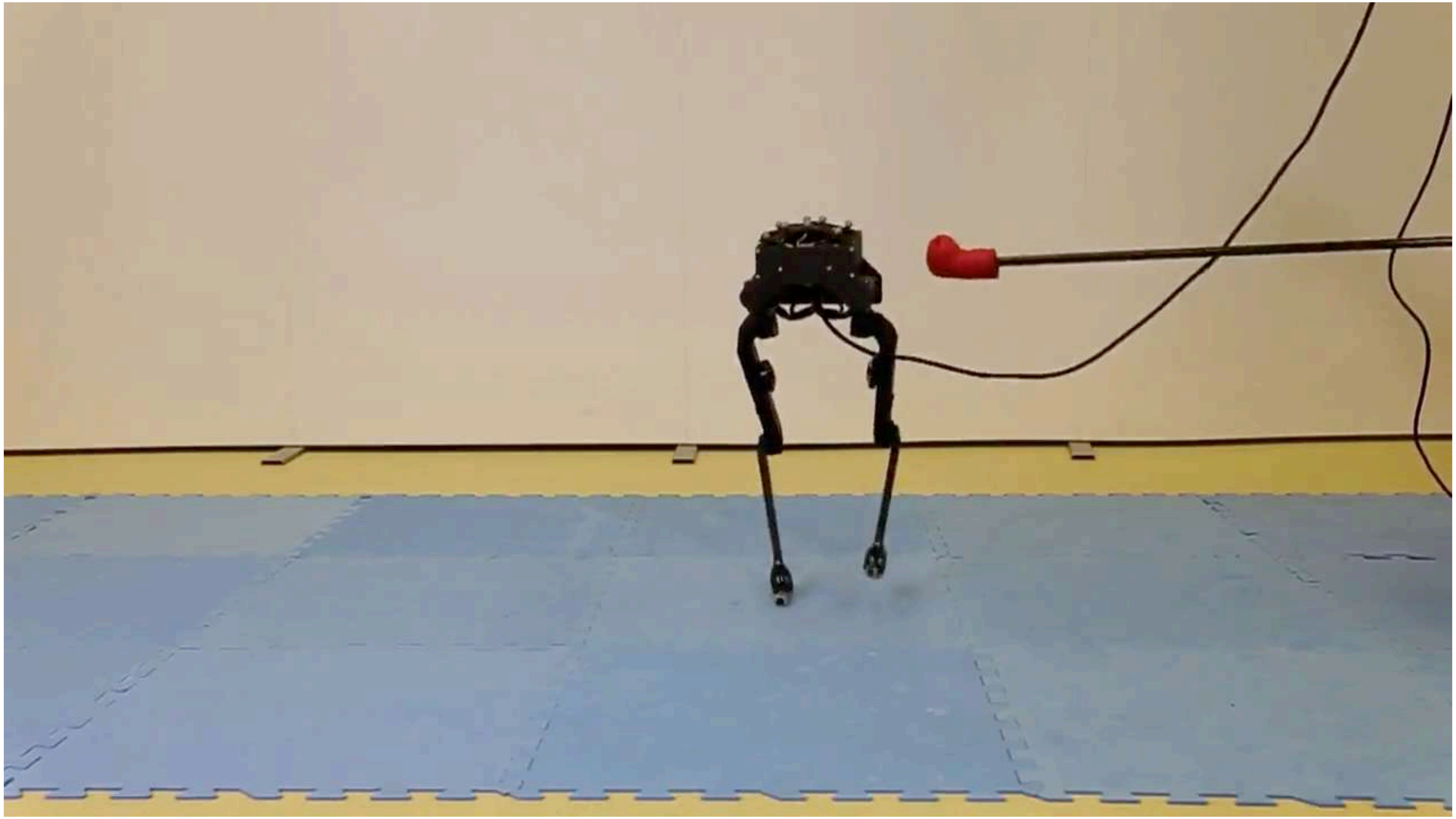


HRP2 - CNRS-AIST [Herdt, et al., 2010]

Linear inverted pendulum models are also used in quadruped robots

Kalakrishnan et al. IJRR 2011

Linear inverted pendulum models can be used with different stability criterions



Daneshmand et al. RA-L 2021

[Tesla 2022]

QPs are found everywhere!