ROB-GY 6323
reinforcement learning and optimal control for robotics

Lecture 5
Model predictive 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
(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		
5		Lecture 5: Model-predictive control	HW 2	
6		Lecture 6: 0th order methods (tentative)		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			

Homework 2 will be posted tomorrow

Homework 1…

What do you expect to learn in this class by using ChatGPT to do your homework?

Homework 1…

Do not use ChatGPT:
=> the answers are wrong
=> you learn nothing
=> you will receive 0 to the HW. Next time I will report you for academic misconduct and you will get an F to the class

Instead… use your brain and come ask me or Armand questions when you do not understand

Quick recap on optimization

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

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

Case 1: quadratic cost and linear equalities

Quadratic costs and linear equality constraints

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

$$\text{subject to} \quad My = p$$

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

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

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

Case 1: quadratic cost and linear equalities

Specificities for optimal control problem
(the KKT matrix has a lot of zeros!)

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

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

$$x_0 = x_{init}$$

$$\Leftrightarrow$$

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

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

Efficient algorithm to solve the KKT system:

1 Compute backward (from N to 0):

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

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 backward recursion is often called the Riccati recursion

K_n are called "feedback gains"

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!

Case II: quadratic cost and linear equalities and inequalities

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).

Use a QP solver - there are many

QP solvers for optimal control problems also use the zeros in the KKT for speed (e.g. HPIPM)

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

Case III: finding the zeroes of a function
(Newton’s method for finding zeros)

Finding the zeros of a function

Newton's method to find the zeros of a function $f(x) = 0$

Start with a guess x_0 and then iterate:

1 Find p_k such that $f(x_k) + \nabla f(x_k)^T p_k = 0$

2 Set $x_{k+1} = x_k + p_k$

Case IV: minimizing an arbitrary function

Minimizing an arbitrary function

Algorithm to minimize $f(x)$:

Start with a initial guess x_0 and iterate until convergence

1. Find a descent direction p_k (i.e. a direction that will decrease the function)
2. Find the length of the step $\alpha \in (0, 1]$
3. Set $x_{k+1} = x_k + \alpha p_k$

Minimizing an arbitrary function

Algorithm to minimize $f(x)$:

Start with a initial guess x_0 and iterate until convergence

1 Find a descent direction p_k (i.e. a direction that will decrease the function)

2 Find the length of the step $\alpha \in (0, 1]$

3 Set $x_{k+1} = x_k + \alpha p_k$

Options for Step 1

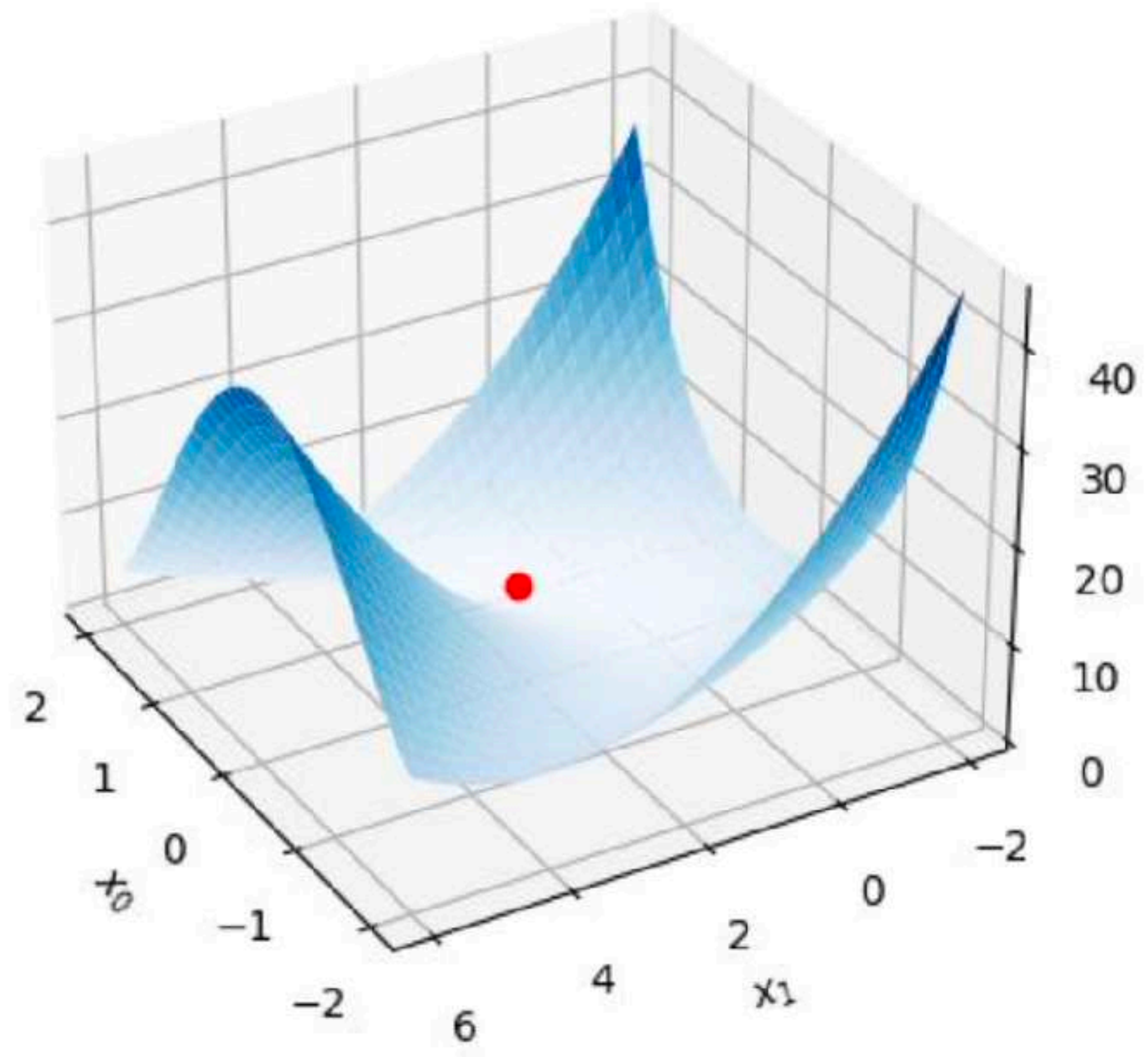
If we choose $p_k = -\nabla f(x_k)$ this is gradient descent

If we choose p_k such that $\nabla^2 f(x_k) p_k = \nabla f(x_k)$ this is Newton's method (only when $\nabla^2 f(x_k) > 0$)

Other methods exist (e.g. Gauss-Newton for least square problems, quasi-Newton methods, etc)

How to fix Newton's method when the Hessian is not positive definite?

If we choose p_k such that $\nabla^2 f(x_k) p_k = \nabla f(x_k)$ this is Newton's method (only when $\nabla^2 f(x_k) > 0$)



Minimizing an arbitrary function

Algorithm to minimize $f(x)$:

Start with a initial guess x_0 and iterate until convergence

1 Find a descent direction p_k (i.e. a direction that will decrease the function)

2 Find the length of the step $\alpha \in (0, 1]$

3 Set $x_{k+1} = x_k + \alpha p_k$

Options for Step 2
Backtracking line search (the easiest to implement)
Note that other line search methods exist

Algorithm 3.1 (Backtracking Line Search).

Choose $\bar{\alpha} > 0$, $\rho \in (0, 1)$, $c \in (0, 1)$; Set $\alpha \leftarrow \bar{\alpha}$;
repeat until $f(x_k + \alpha p_k) \leq f(x_k) + c\alpha \nabla f_k^T p_k$
    $\alpha \leftarrow \rho\alpha$;
end (repeat)
Terminate with $\alpha_k = \alpha$.

Case IV: minimizing an arbitrary function with constraints

Minimizing an arbitrary function with constraints

$$\min_x f(x) = 0$$
$$\text{subject to } g(x) = 0$$

KKT conditions:

$$\nabla f(x) + \lambda \nabla g(x) = 0$$
$$g(x) = 0$$

The Sequential Quadratic Programming (SQP) idea is to apply a Newton step on the KKT conditions:

$$\begin{bmatrix} \nabla^2_{xx}\mathcal{L}(x_k) & \nabla g(x_k) \\ \nabla g(x_k)^T & 0 \end{bmatrix} \begin{pmatrix} p_k \\ p_\lambda \end{pmatrix} = \begin{pmatrix} -\nabla f(x_k) - \nabla g(x_k)^T \lambda_k \\ -g(x_k) \end{pmatrix}$$

Solve for (p_k, p_λ). The Newton step iterate is then $\begin{pmatrix} x_{k+1} \\ \lambda_{k+1} \end{pmatrix} = \begin{pmatrix} x_k \\ \lambda_k \end{pmatrix} + \begin{pmatrix} p_k \\ p_\lambda \end{pmatrix}$

$$\begin{bmatrix} \nabla^2_{xx}\mathcal{L}(x_k) & \nabla g(x_k) \\ \nabla g(x_k)^T & 0 \end{bmatrix} \begin{pmatrix} p_k \\ p_\lambda \end{pmatrix} = \begin{pmatrix} -\nabla f(x_k) - \nabla g(x_k)^T \lambda_k \\ -g(x_k) \end{pmatrix} \quad \textbf{with} \quad \begin{pmatrix} x_{k+1} \\ \lambda_{k+1} \end{pmatrix} = \begin{pmatrix} x_k \\ \lambda_k \end{pmatrix} + \begin{pmatrix} p_k \\ p_\lambda \end{pmatrix}$$

Is the same as solving $\begin{bmatrix} \nabla^2_{xx}\mathcal{L}(x_k) & \nabla g(x_k) \\ \nabla g(x_k)^T & 0 \end{bmatrix} \begin{pmatrix} p_k \\ \lambda_{k+1} \end{pmatrix} = \begin{pmatrix} -\nabla f(x_k) \\ -g(x_k) \end{pmatrix}$

which is the same as solving the following QP

$$\min_p \frac{1}{2} p^T \nabla^2_{xx}\mathcal{L}(x_k) p + p^T \nabla f(x_k)$$
$$\text{subject to } \nabla g(x_k)^T p + g(x_k) = 0$$

Sequential Quadratic Programming (SQP)

$$\min_x f(x)$$
$$\text{subject to } g(x) = 0$$

Step 1: Find a direction

$$\min_p \; p^T \nabla^2_{xx} \mathcal{L}(x_k) p + p^T \nabla f(x_k)$$
$$\text{subject to } \nabla g(x_k)^T p + g(x_k) = 0$$

Step 2: Find a step length α_k with a line search

$$x_{k+1} = x_k + \alpha p$$

Use a merit function for line search (to balance cost reduction and constraint satisfaction)

Merit function

$$\phi(x) = f(x) + \mu ||g(x)||_1$$

Sufficient decrease condition

$$\phi(x_k + \alpha_k p_k) \leq \phi(x_k) + \eta \alpha_k (\nabla f_k^T p_k - \mu ||g(x_k)||_1)$$

SQP extension to inequalities

$$\min_x f(x)$$
$$\text{subject to } g(x) = 0$$
$$h(x) \leq 0$$

Step 1: Find a direction

$$\min_p \; p^T \nabla^2_{xx} \mathcal{L}(x_k) p + p^T \nabla f(x_k)$$
$$\text{subject to } \nabla g(x_k)^T p + g(x_k) = 0$$
$$\nabla h(x_k)^T p + h(x_k) \leq 0$$

Use any QP solver!

Step 2: Find a step length α_k with a merit function and line search

$$x_{k+1} = x_k + \alpha p$$

Case V: application to optimal control problems
(again exploring the zeros of the KKT matrix for speed)

Nonlinear optimal control problems

$$\min_{x_1,\ldots,x_T,u_0,\ldots,u_{T-1}} \sum_{t=0}^{T-1} \ell_t(x_t,u_t) + \ell_T(x_T)$$

$$\text{subject to } x_{t+1} = f(x_t, x_t)$$

can be solved using a SQP algorithm

In the QP, we need the Hessian of the Lagrangian $\nabla^2_{xx}\mathcal{L}$. Most algorithms ignore the Hessian of the dynamics $f(x,u)$ in this case

The resulting KKT matrix will have the same sparsity structure we saw with optimal control problems with quadratic costs and linear constraints. So similar backward-forward recursions can be derived (with more terms!) to solve the KKT system and the QP.

The QP inside the SQP…

$$\min_{\Delta \boldsymbol{x}, \Delta \boldsymbol{u}} \sum_{k=0}^{T-1} \frac{1}{2} \begin{bmatrix} \Delta x_k \\ \Delta u_k \end{bmatrix}^\top \begin{bmatrix} Q_k & S_k \\ S_k^\top & R_k \end{bmatrix} \begin{bmatrix} \Delta x_k \\ \Delta u_k \end{bmatrix} + \begin{bmatrix} q_k \\ r_k \end{bmatrix}^\top \begin{bmatrix} \Delta x_k \\ \Delta u_k \end{bmatrix} + \frac{1}{2} \Delta x_T^\top Q_T \Delta x_T + \Delta x_T^\top q_T$$

$$\text{s.t.} \Delta x_{k+1} = A_k \Delta x_k + B_k \Delta u_k + \gamma_{k+1}$$

where

$$\begin{aligned} Q_T &= \left(\nabla_{xx}^2 \ell_T - \mu_T^\top \nabla_{xx}^2 c_T\right) (x_T^{[n]}) \\ Q_k &= \left(\nabla_{xx}^2 \ell_k + \lambda_{k+1}^\top \nabla_{xx}^2 f_k - \mu_k^\top \nabla_{xx}^2 c_k\right) (x_k^{[n]}, u_k^{[n]}) \\ S_k &= \left(\nabla_{xu}^2 \ell_k + \lambda_{k+1}^\top \nabla_{xu}^2 f_k - \mu_k^\top \nabla_{xu}^2 c_k\right) (x_k^{[n]}, u_k^{[n]}) \\ R_k &= \left(\nabla_{uu}^2 \ell_k + \lambda_{k+1}^\top \nabla_{uu}^2 f_k - \mu_k^\top \nabla_{uu}^2 c_k\right) (x_k^{[n]}, u_k^{[n]}) \end{aligned}$$

$$\begin{aligned} A_k &= \nabla_x f_k(x_k^{[n]}, u_k^{[n]}), \quad B_k = \nabla_u f_k(x_k^{[n]}, u_k^{[n]}) \\ q_k &= \nabla_x \ell_k(x_k^{[n]}, u_k^{[n]}), \quad r_k = \nabla_u \ell_k(x_k^{[n]}, u_k^{[n]}) \\ q_T &= \nabla_x \ell_T(x_T^{[n]}) \end{aligned}$$

… can be simply solved by the following recursions!

1. Backward recursion (from T to 0)

$$V_T = Q_T \qquad v_T = q_T$$

$$h_k = r_k + B_k^\top (v_{k+1} + V_{k+1}\gamma_{k+1})$$

$$G_k = S_k^\top + B_n^\top V_{k+1} A_k \qquad K_k = -H_k^{-1} G_k$$

$$H_k = R_k + B_k^\top V_{k+1} B_k \qquad k_k = -H_k^{-1} h_k$$

$$V_k = Q_k + A_k^\top V_{k+1} A_k - K_k^\top H_k K_k$$

$$v_k = q_k + K_k^\top r_k + (A_k + K_k B_k)^\top (v_{k+1} + V_{k+1}\gamma_{k+1})$$

Similar recursion seen in Lecture 3!

We call it a Riccati recursion

2. Forward recursion (from 0 to T)

$$\Delta x_0 = 0 \qquad \Delta x_{k+1} = (A_k + B_k K_k)\Delta x_k + B_k k_k + \gamma_{k+1}$$

$$\Delta u_k = K_k \Delta x_k + k_k$$

$$\lambda_k = V_k \Delta x_k + v_k$$

Fast solvers for nonlinear optimal control (they all use the "Riccati recursion trick" and handle inequality constraints)

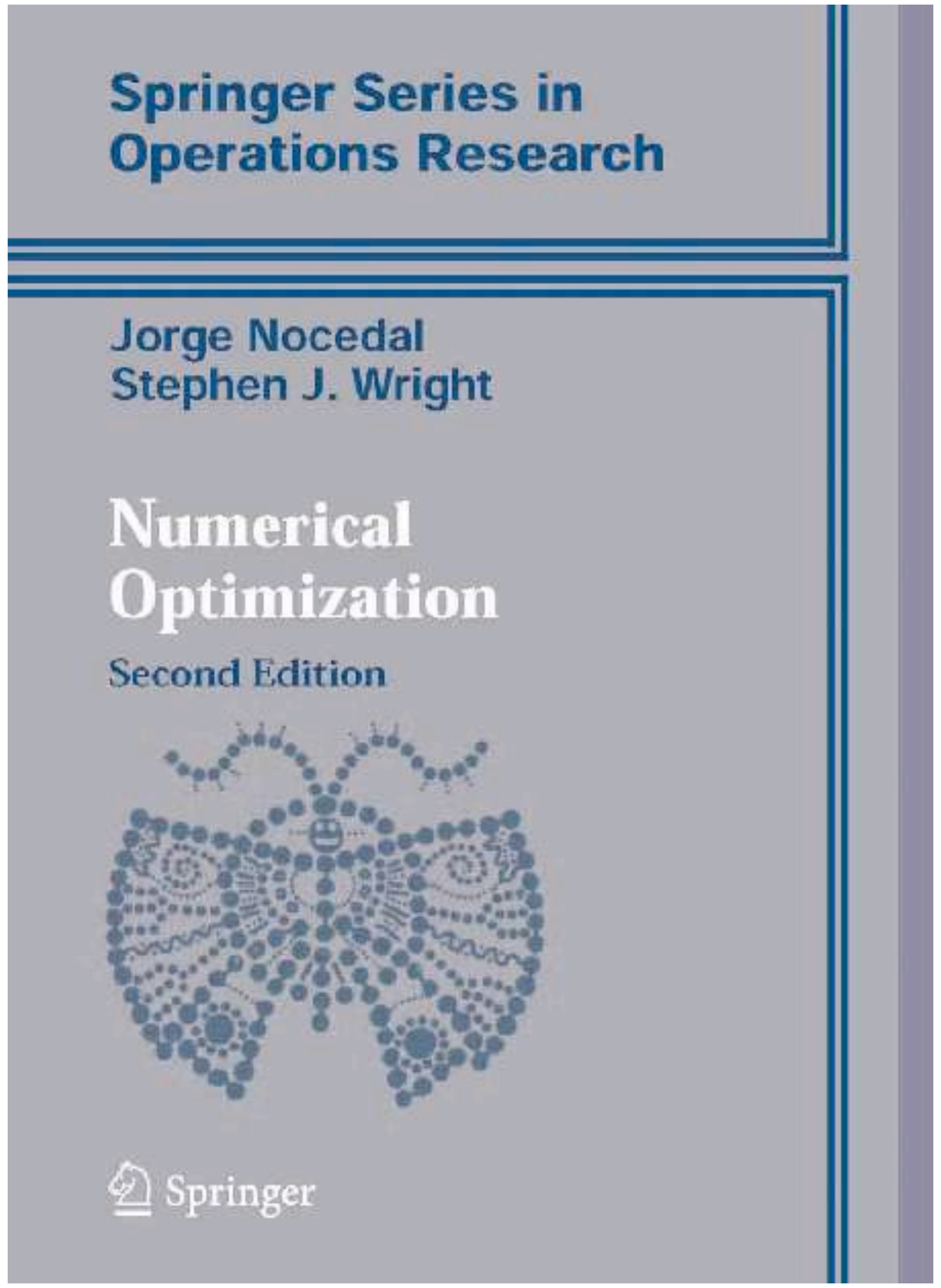
crocoddyl
Contact Robot Optimal Control by Differential Dynamic Library

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

acados

https://github.com/acados/acados

Other algorithms

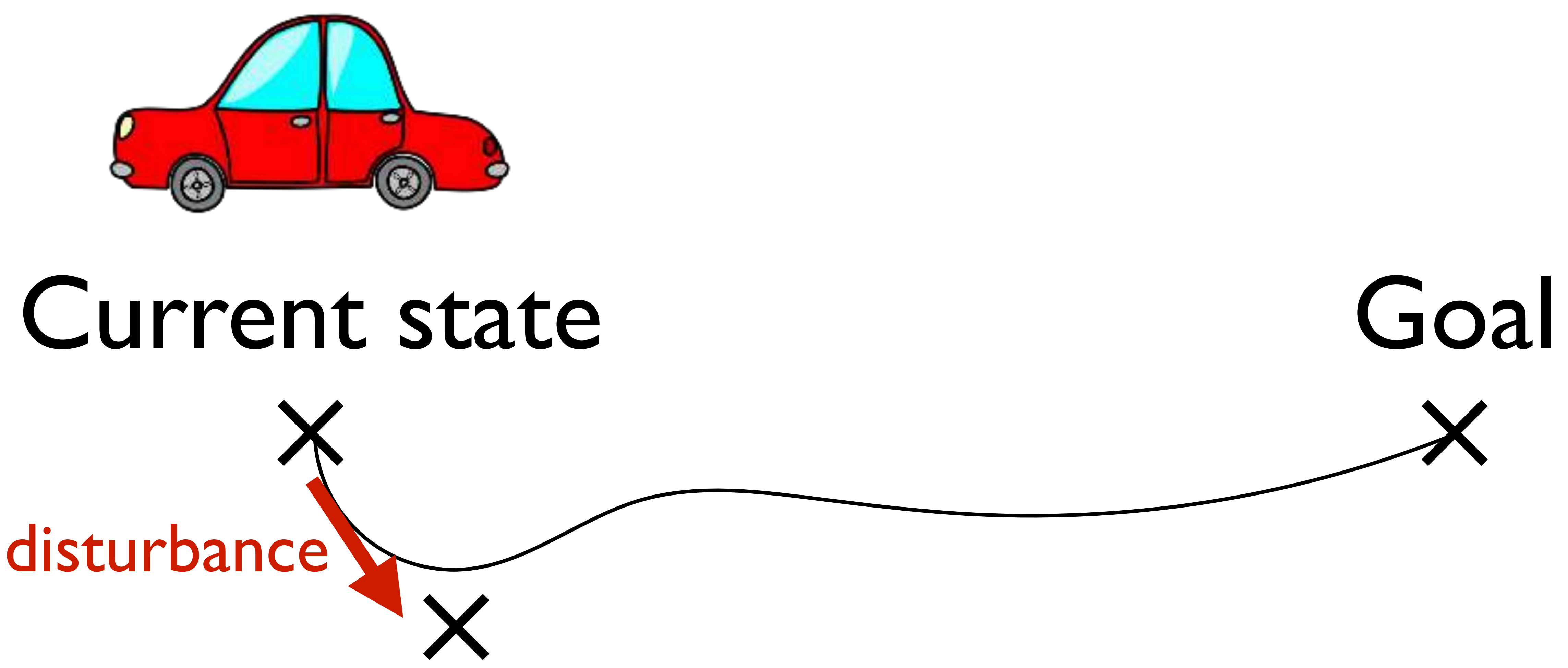


(many) other optimization algorithms exist!
They all use similar ideas (KKT conditions, sparsity, etc)

You have all the tools to implement your own SQP!

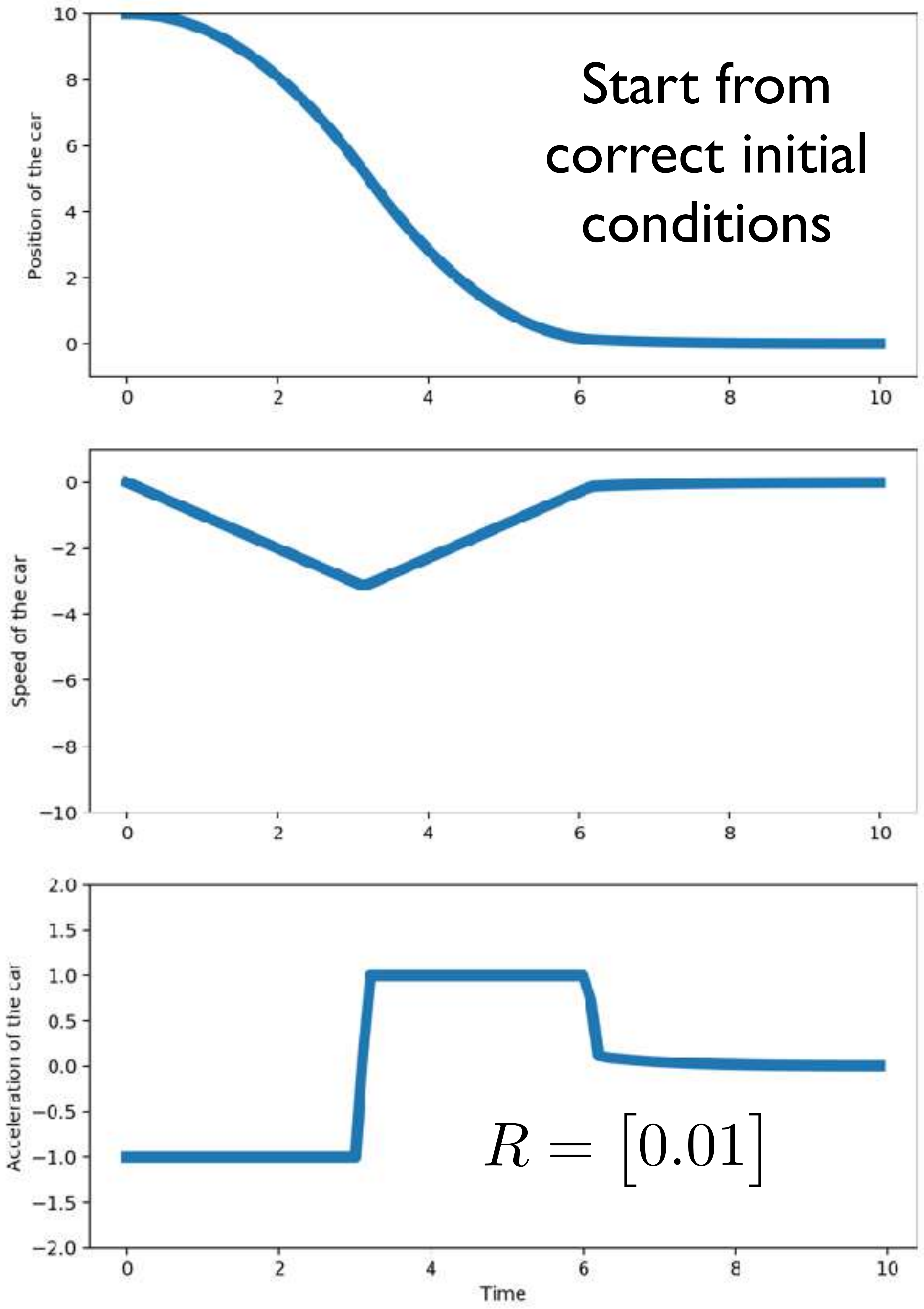
In robotics, algorithms names "differential dynamic programming" (DDP) and "iterative linear quadratic regulators"(iLQR) are popular
=> They are variations of the SQP algorithm described in class (and usually cannot handle inequality constraints and they lack convergence properties)

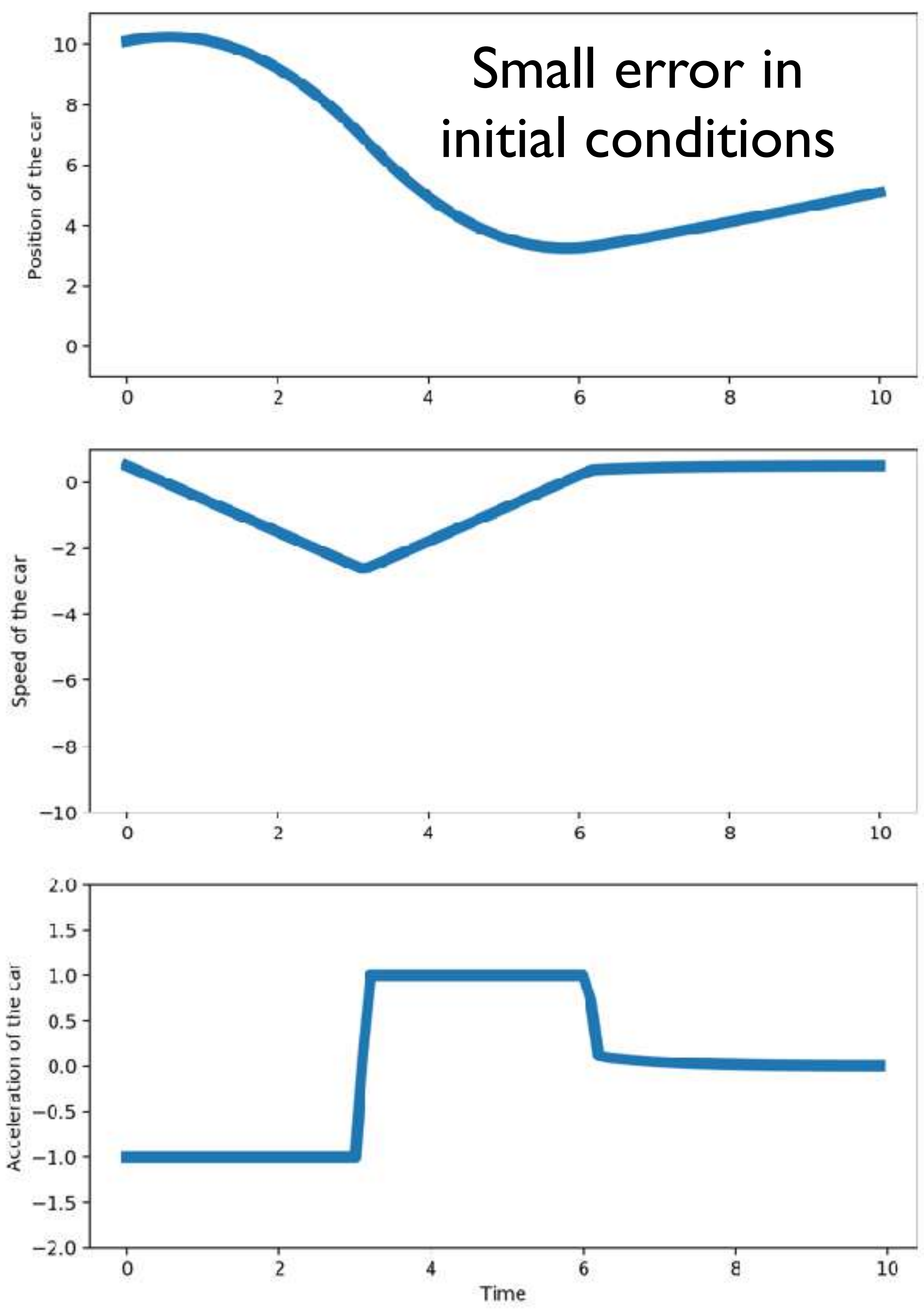
Model predictive control



Current state
Goal
disturbance

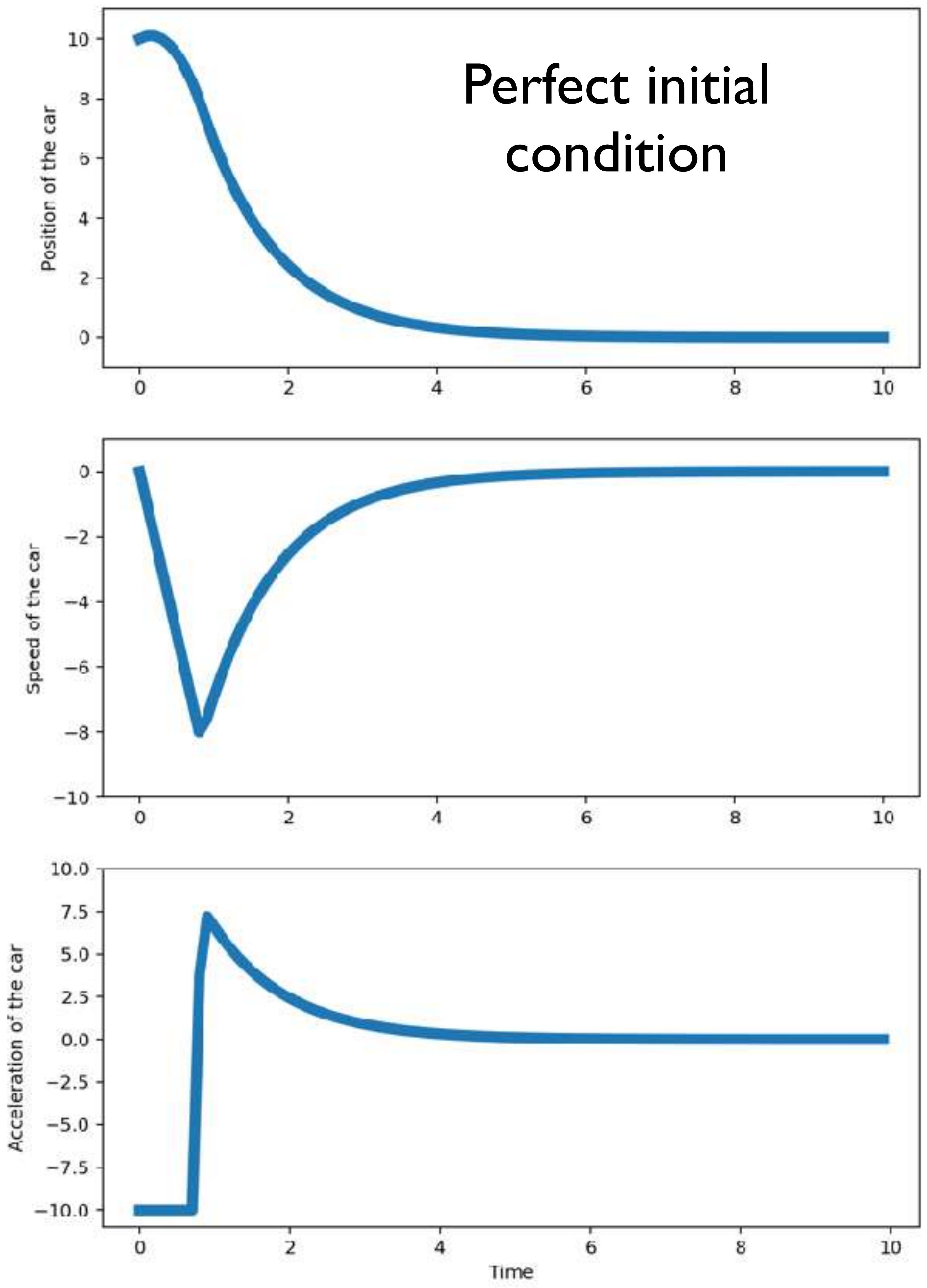
$$x_{n+1} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} x_n + \begin{bmatrix} 0 \\ \Delta t \end{bmatrix} u_n$$

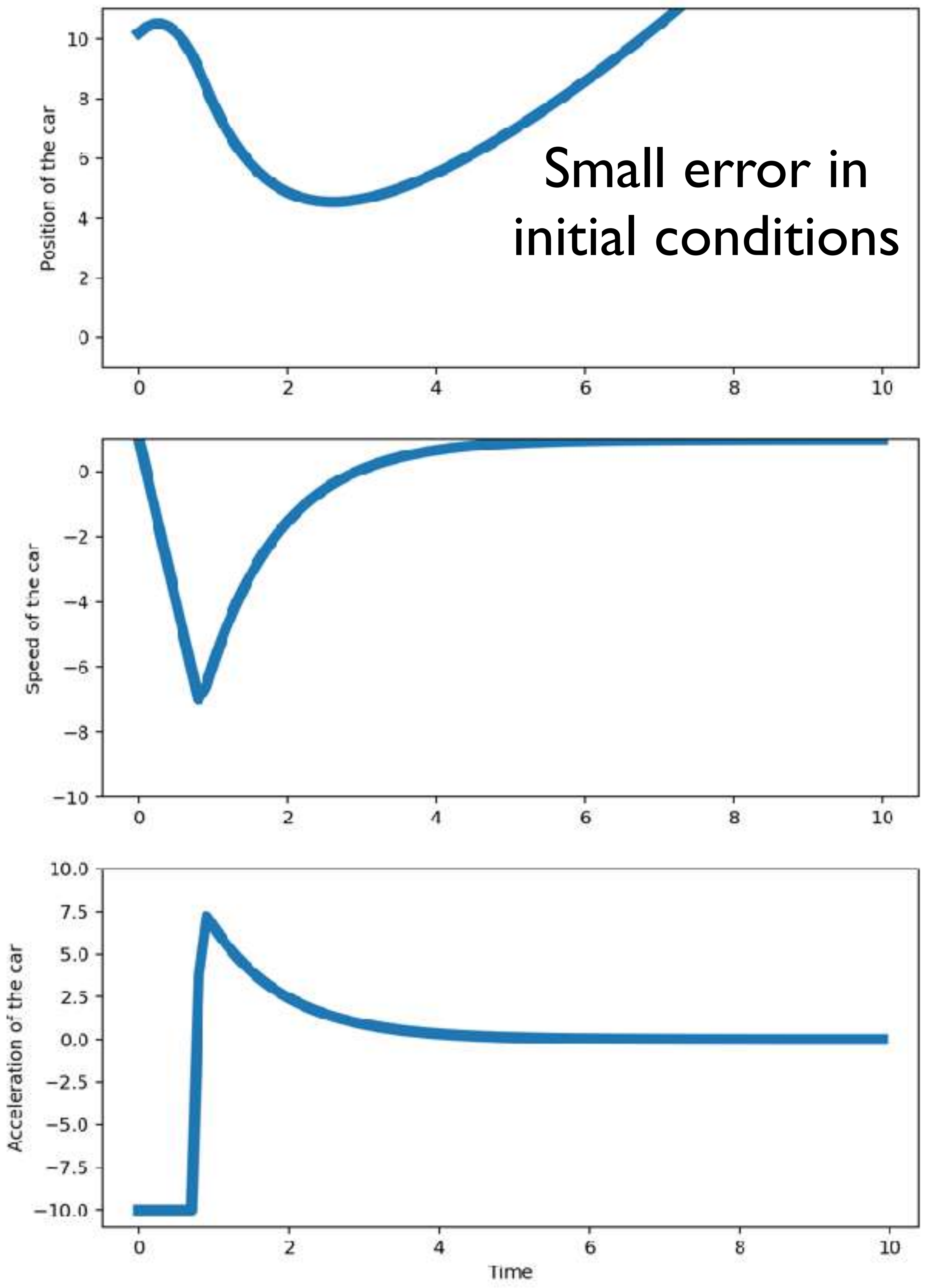


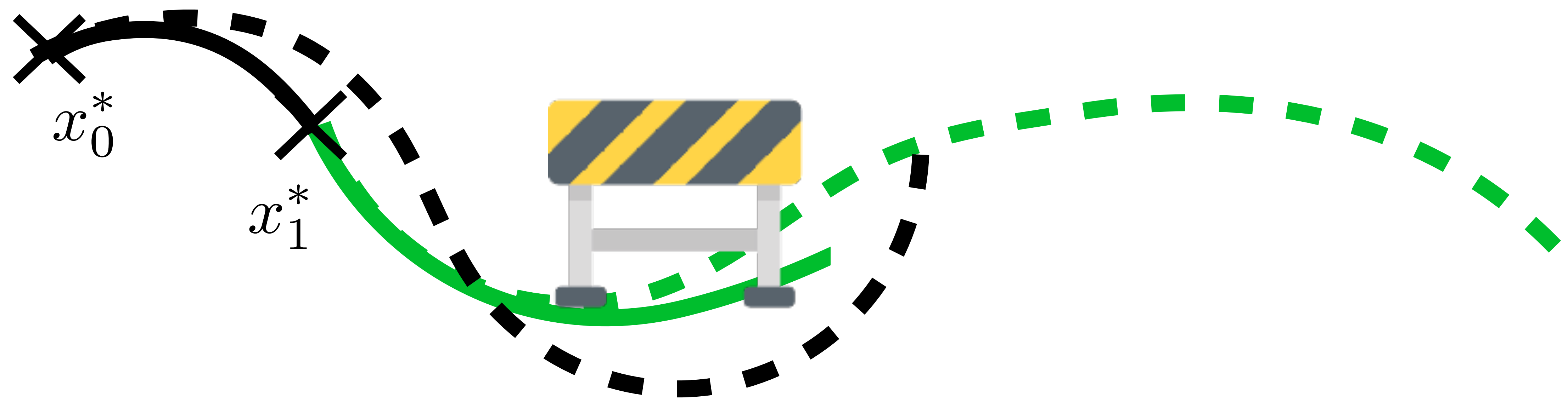


$$x_{n+1} = \begin{bmatrix} 1.01 & \Delta t \\ 0 & 1 \end{bmatrix} x_n + \begin{bmatrix} 0 \\ \Delta t \end{bmatrix} u_n$$

Unstable system

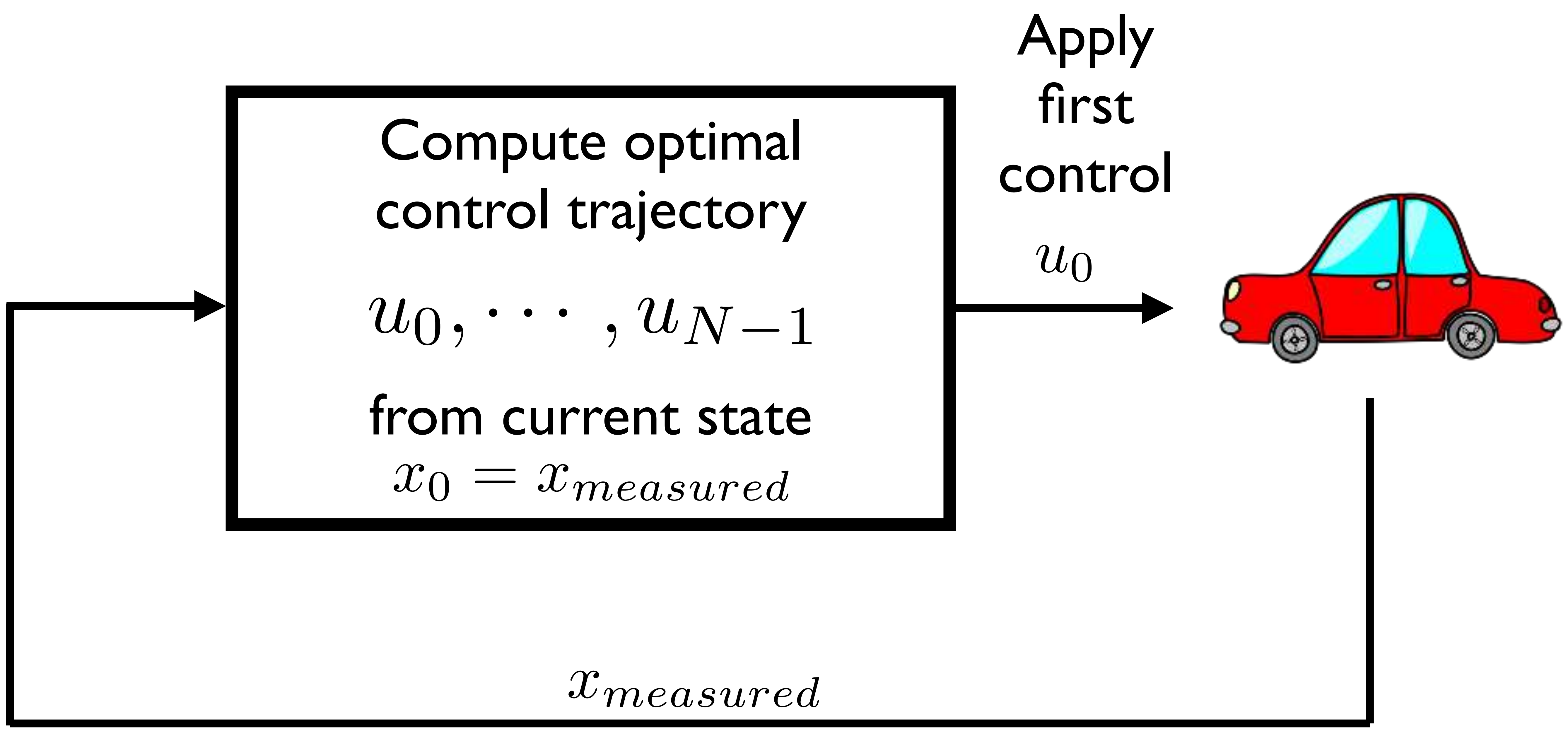






x_0^*
x_1^*

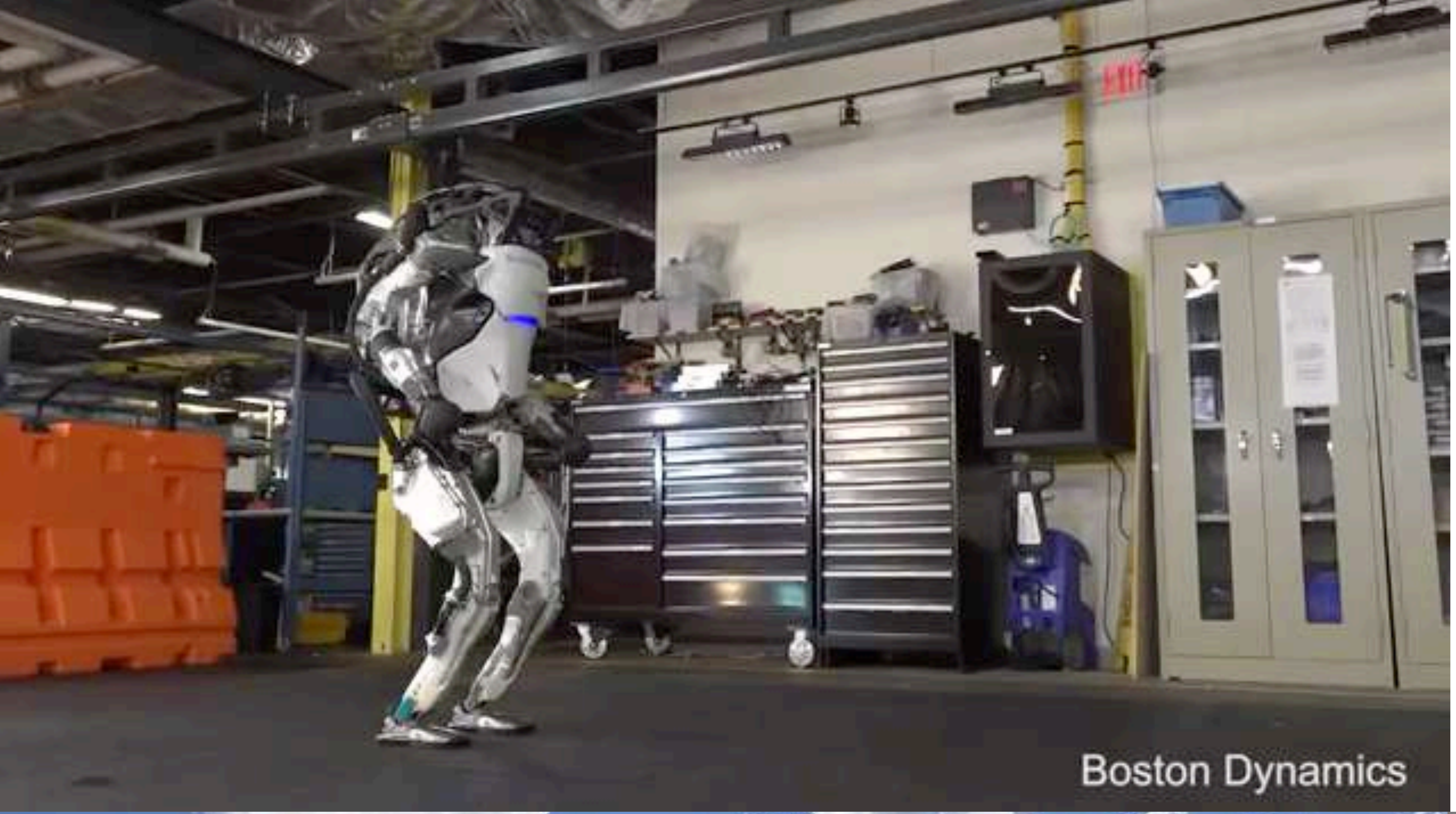
Model predictive control
(receding horizon control)



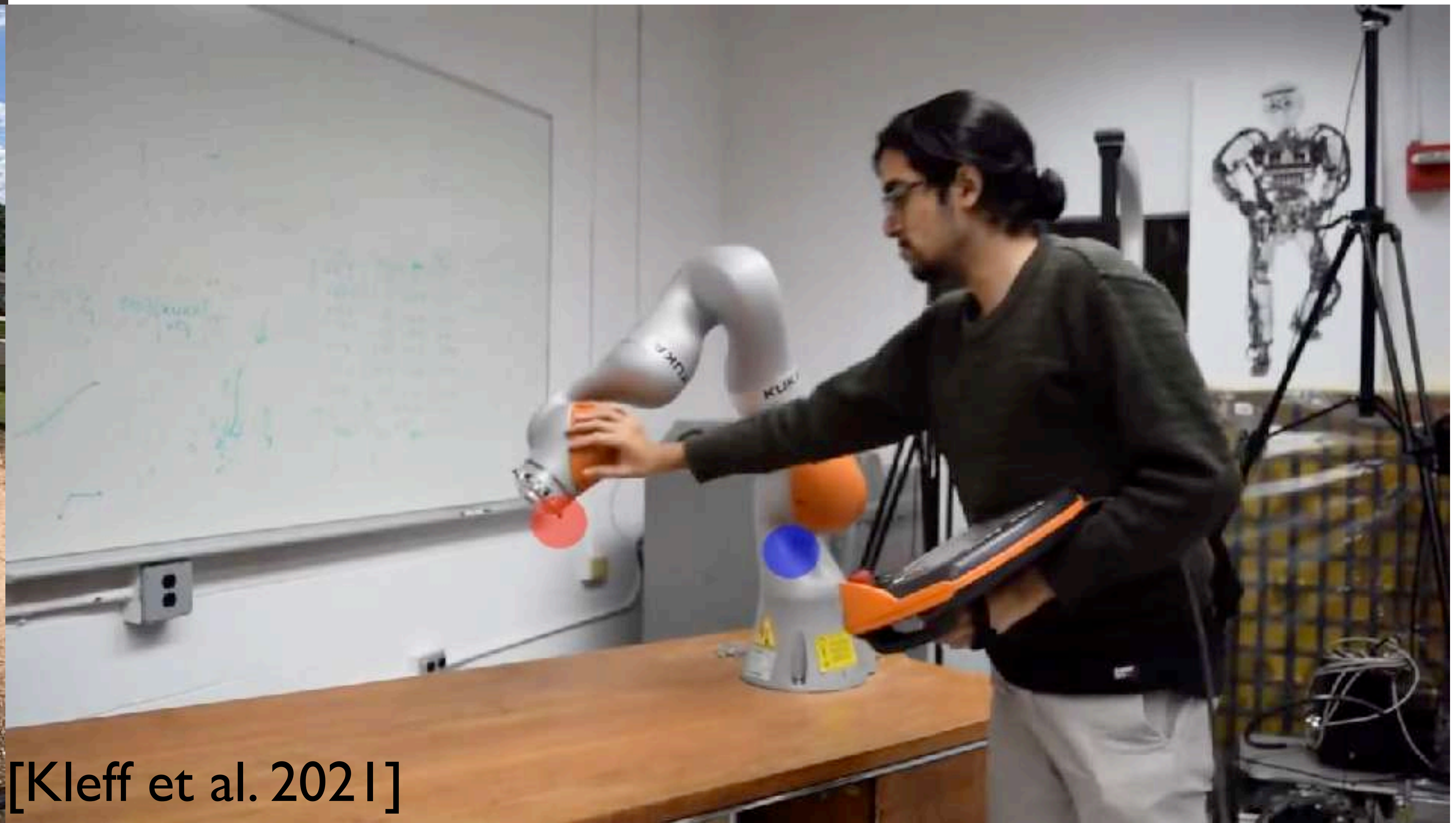
The control law solves an optimization problem at each control cycle

Model predictive control
Some issues

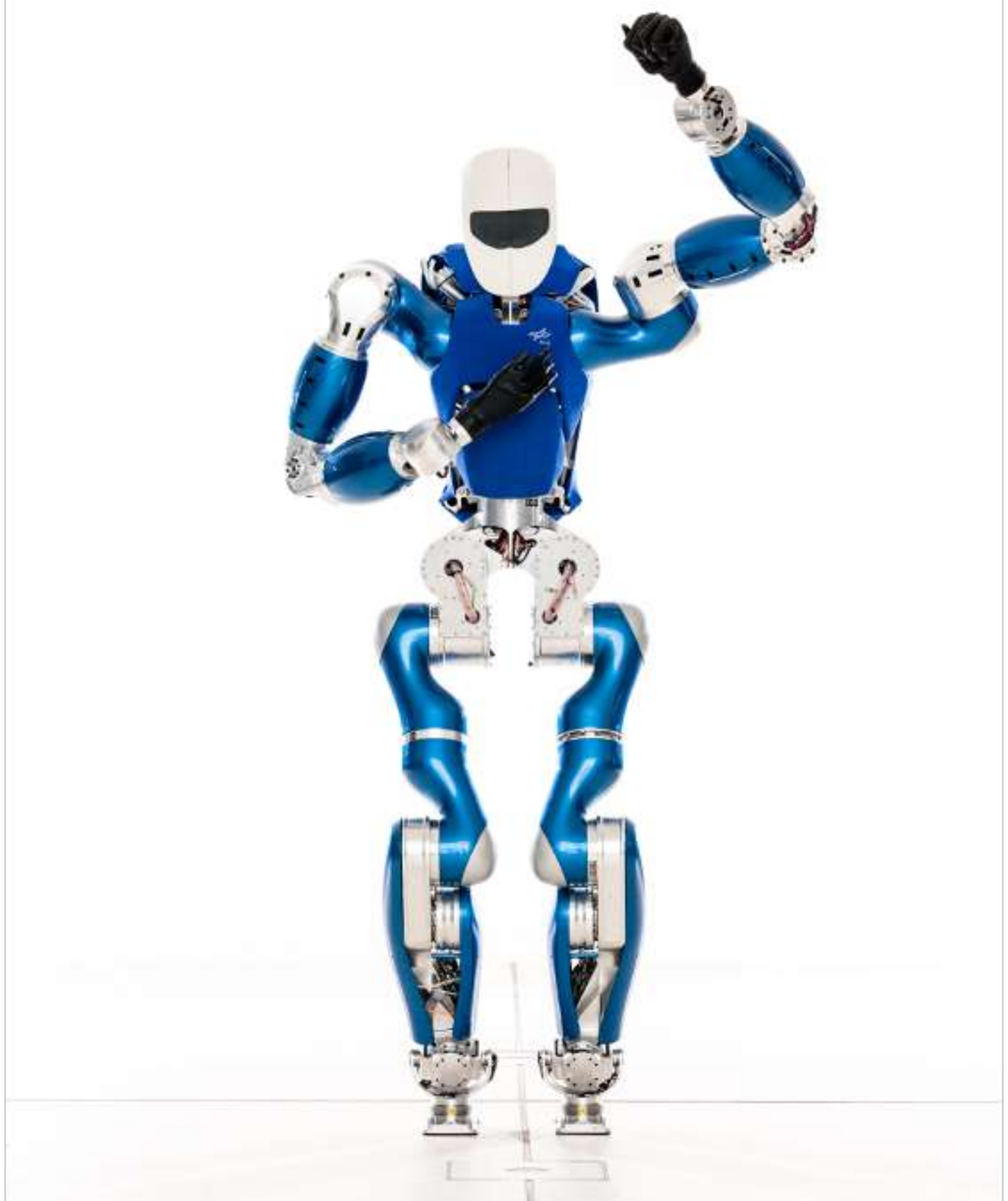
Model predictive control: a core ingredient in robotics




[Gandhi et al. 2020]


[Kleff et al. 2021]

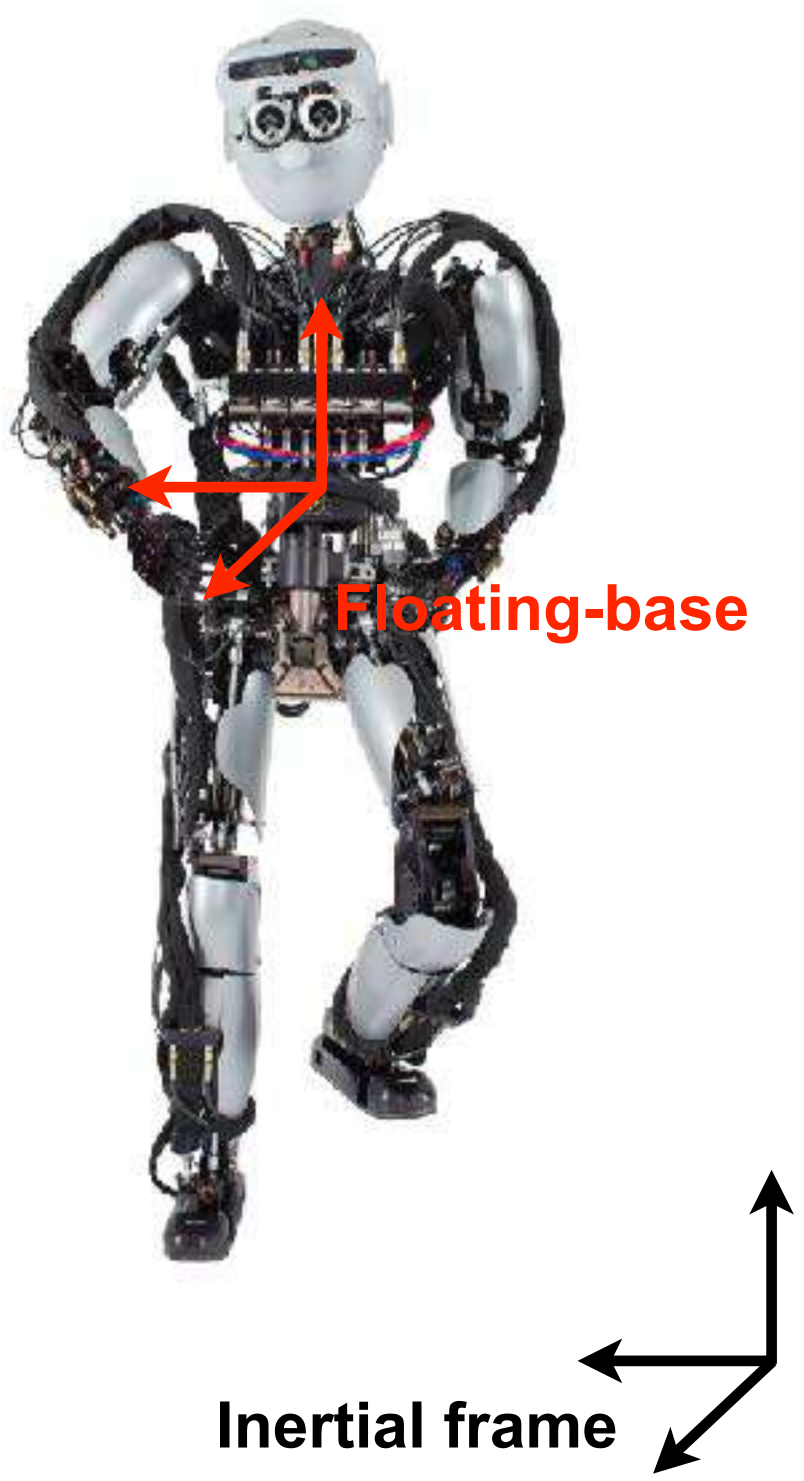
Example: biped walking



What is the problem?

Keeping balance while moving forward

Configuration Space		Actuation Space
Joint angles $\theta \in \mathbb{R}^n$	$\Leftrightarrow$	Motor Torques $\boldsymbol{\tau} \in \mathbb{R}^n$
Pose of the robot in space $(\mathbf{p}, \mathbf{R}) \in SE(3)$	$\cancel{\Leftrightarrow}$	Pose of robot is not actuated!

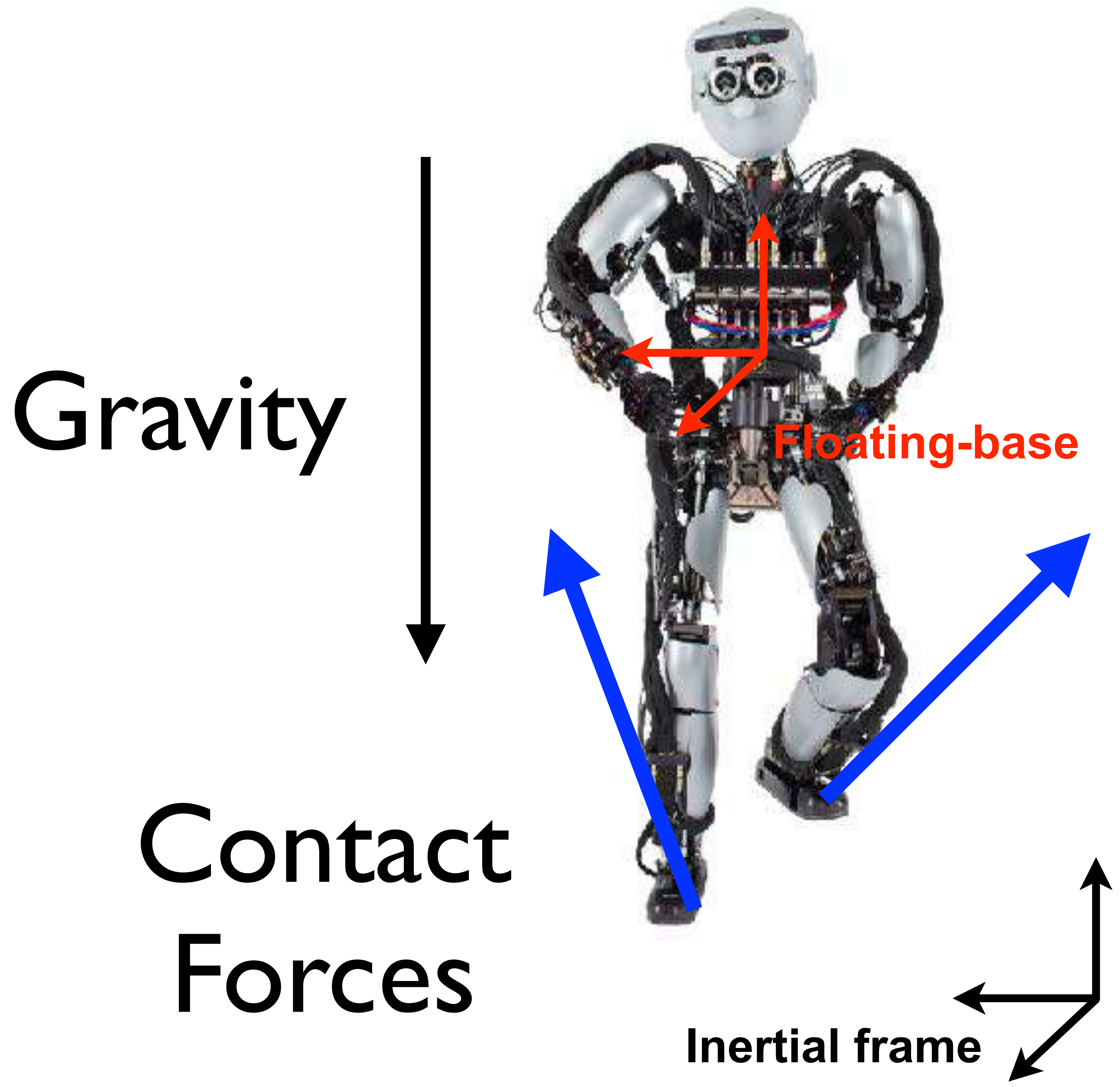


Underactuation!

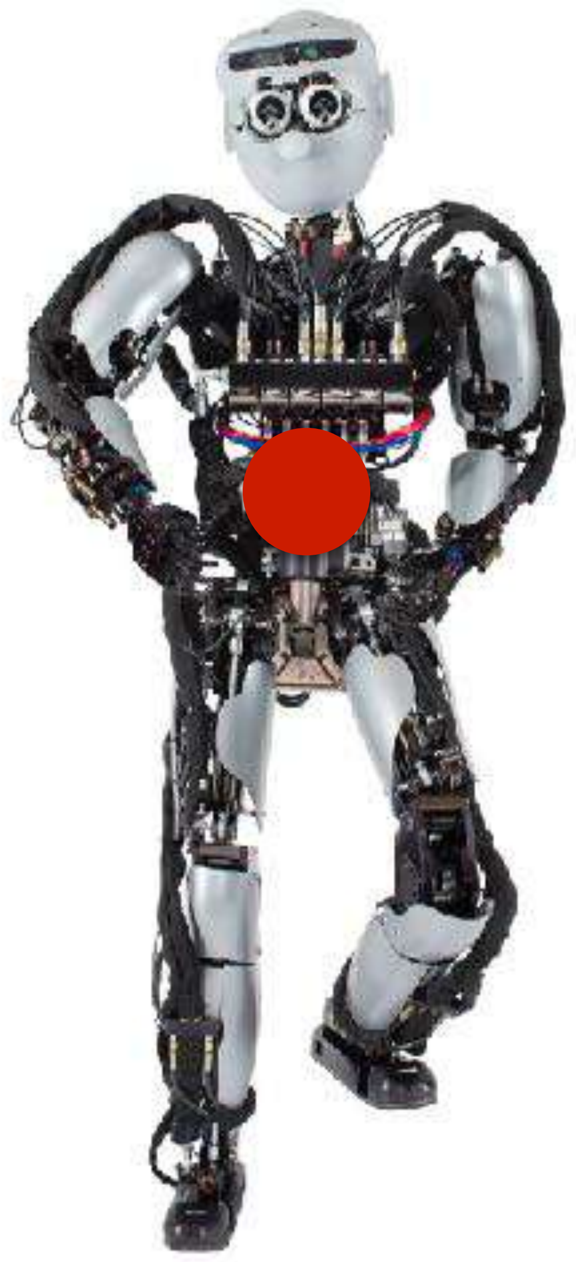
What is the problem?

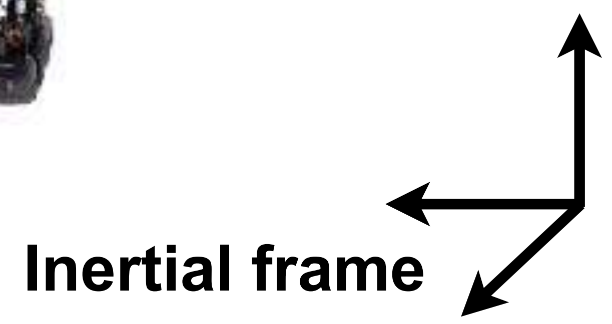
How can the robot move forward then?

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



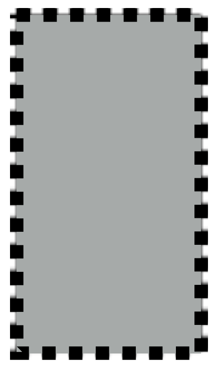
Motion of the center of mass



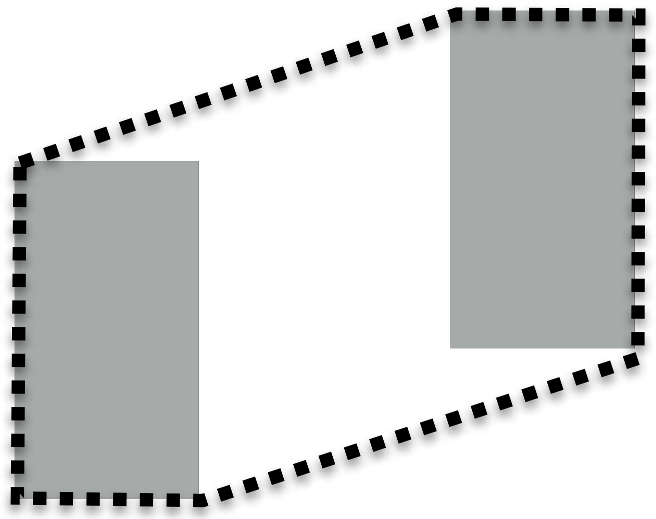


Support polygon

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



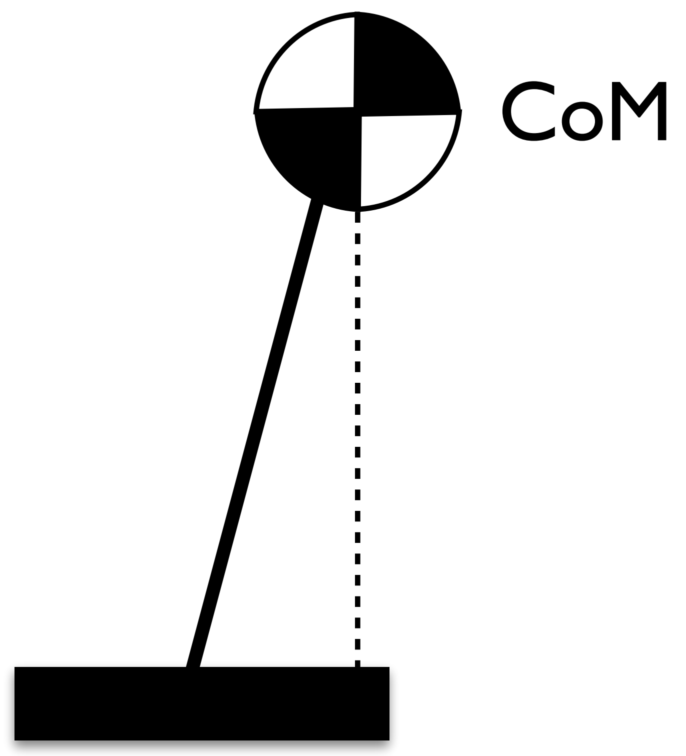
One foot on the ground and support polygon (top view)



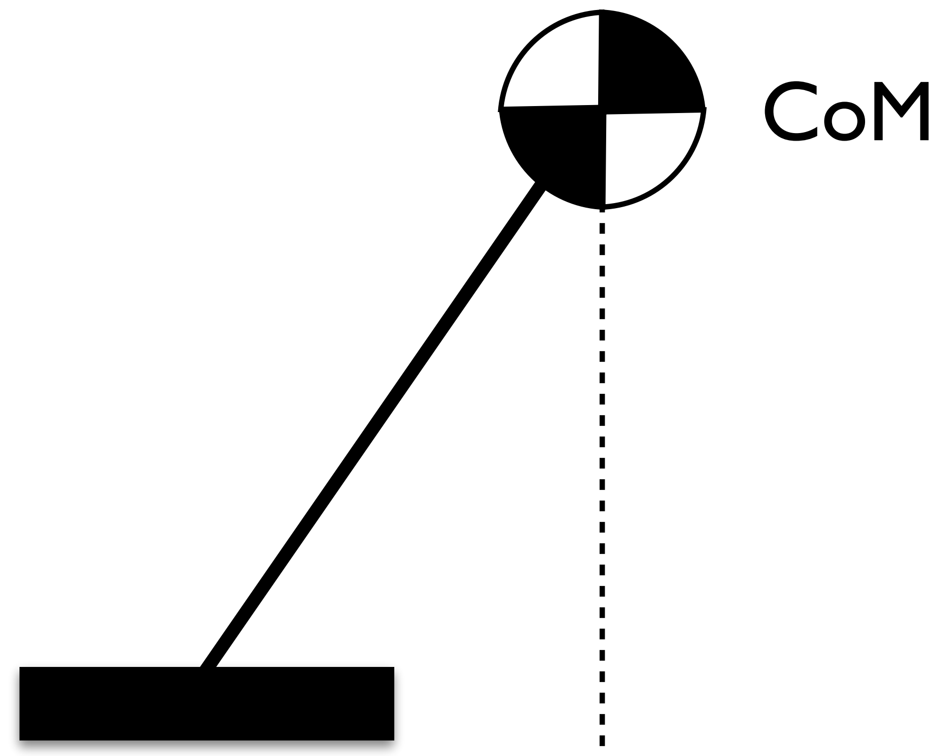
Two feet on the ground and support polygon (top view)

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

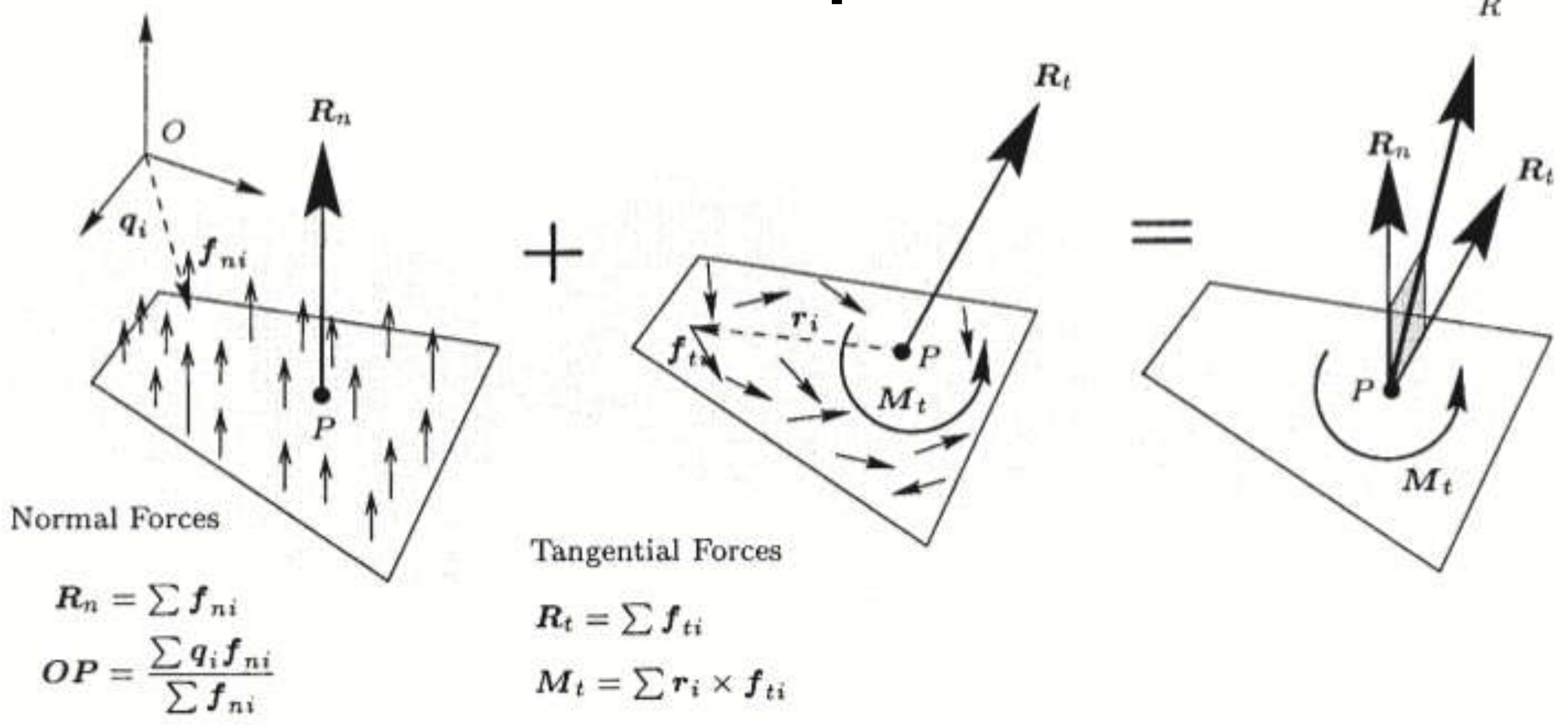


unstable (the foot will rotate around its edge)

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



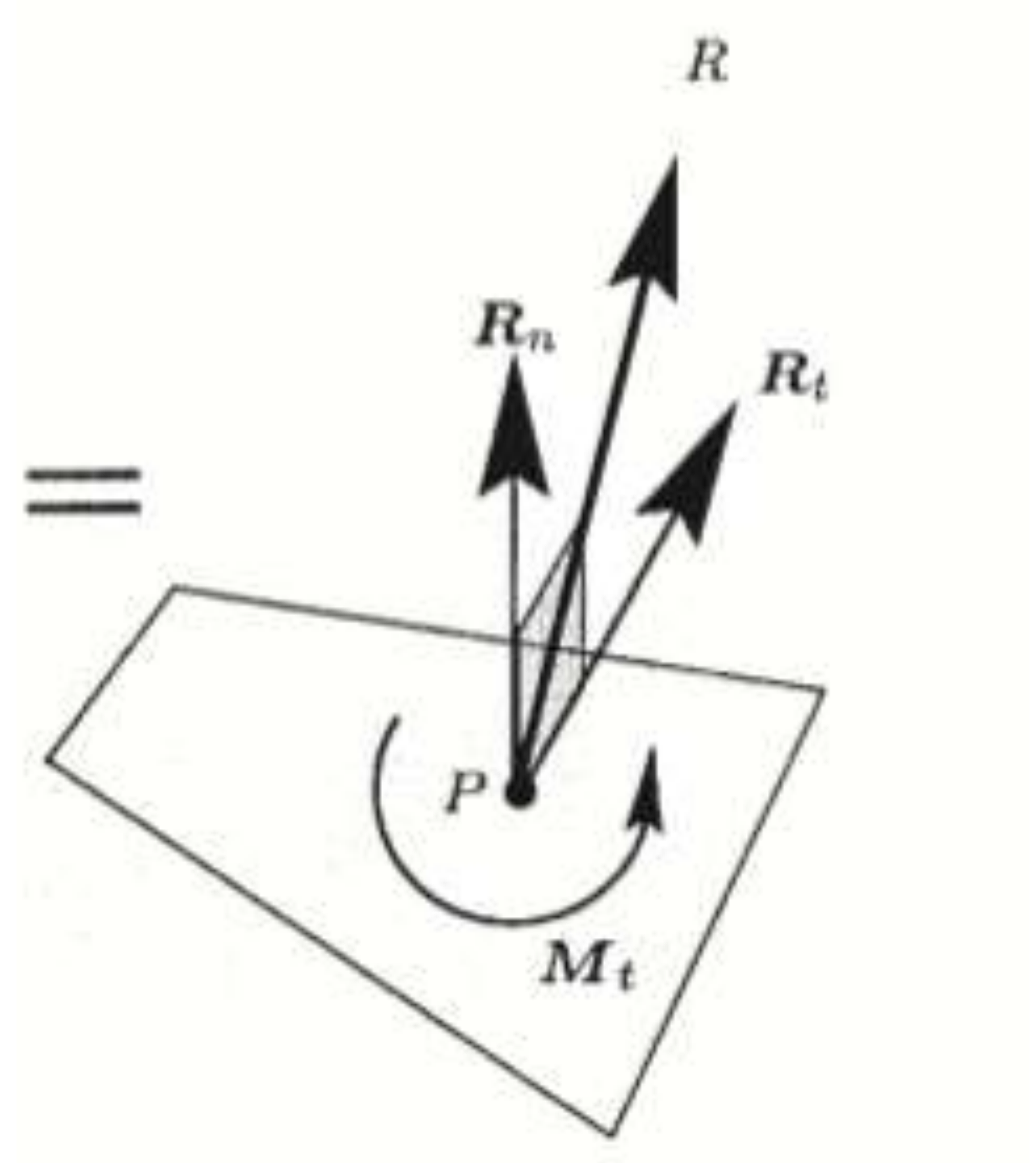
The center of pressure (CoP) is the point of the ground where the resultant force Rn acts

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

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

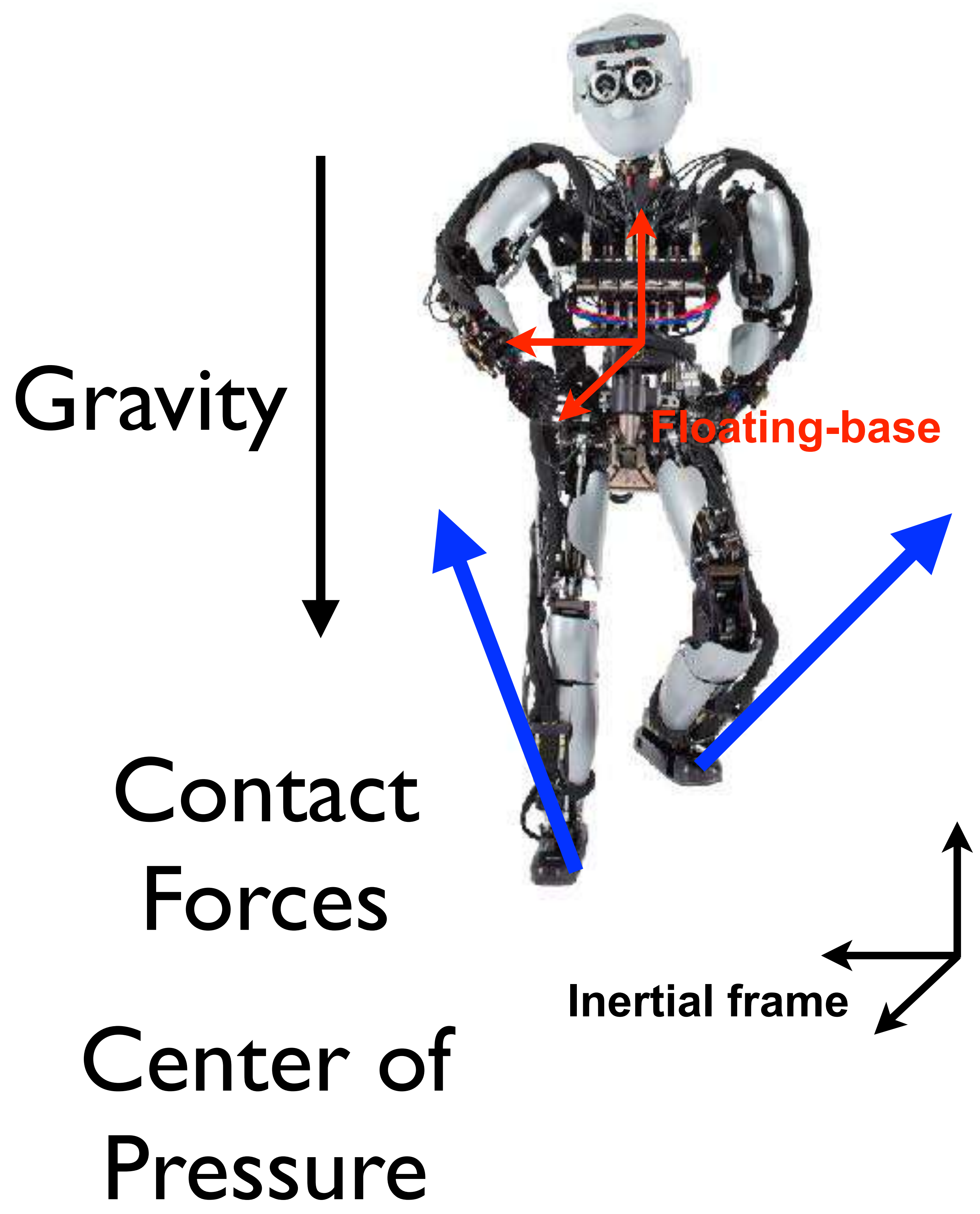
1. Decide where to step (footstep planning)

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



Control of walking

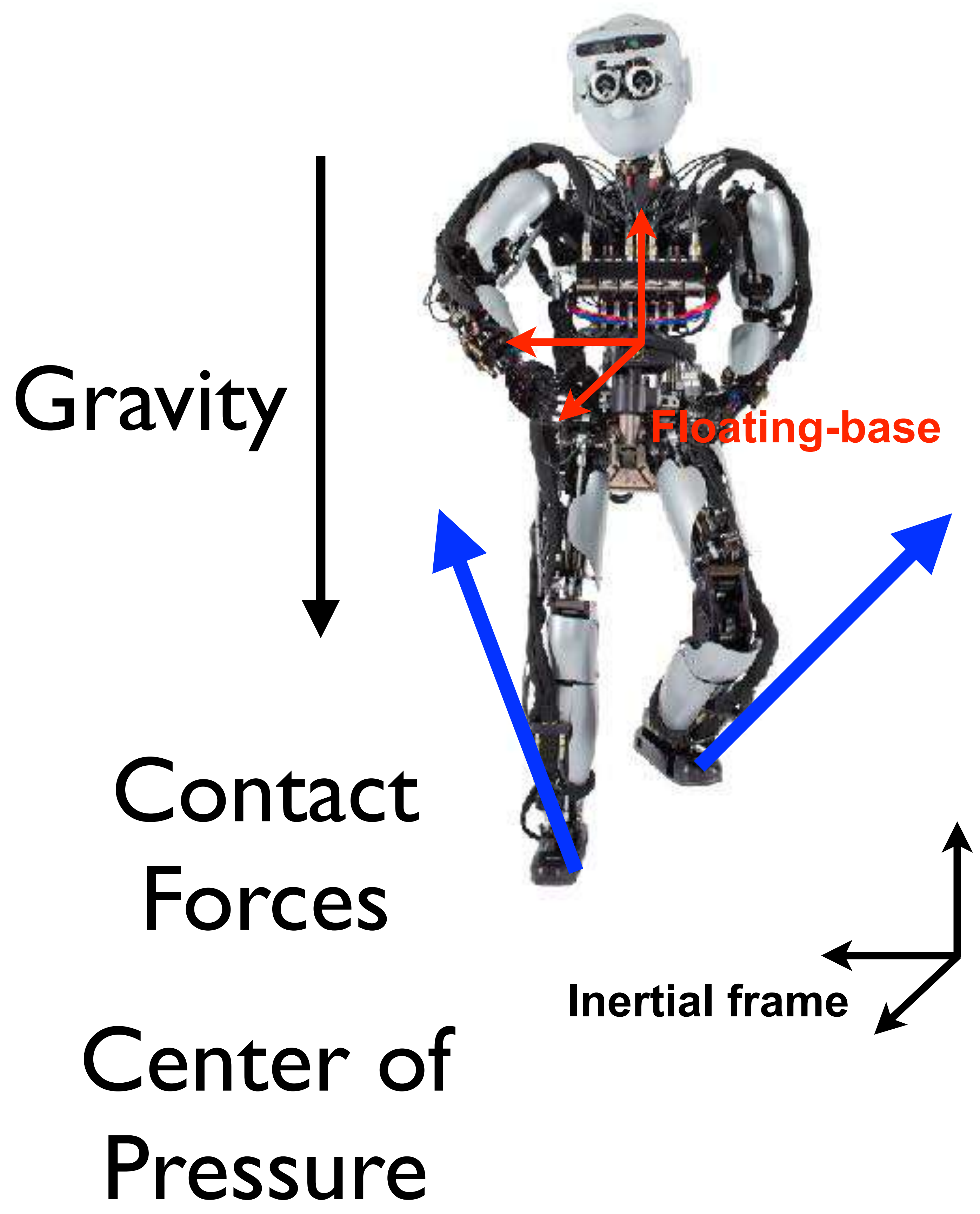
1. Decide where to step (footstep planning)

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

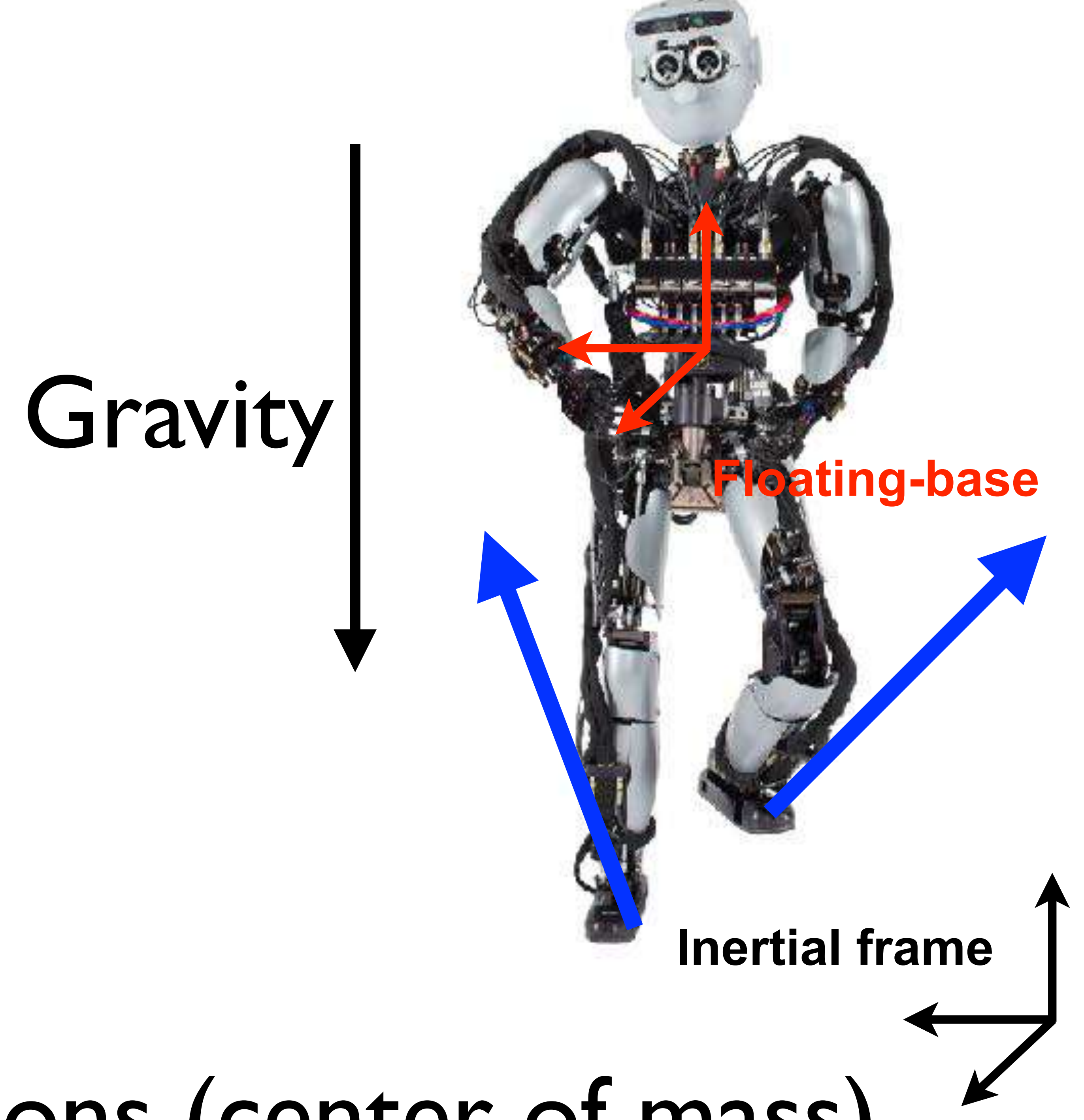
3. Compute a motion of the full robot to follow this desired CoM motion (in particular compute swing foot motions)

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)

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

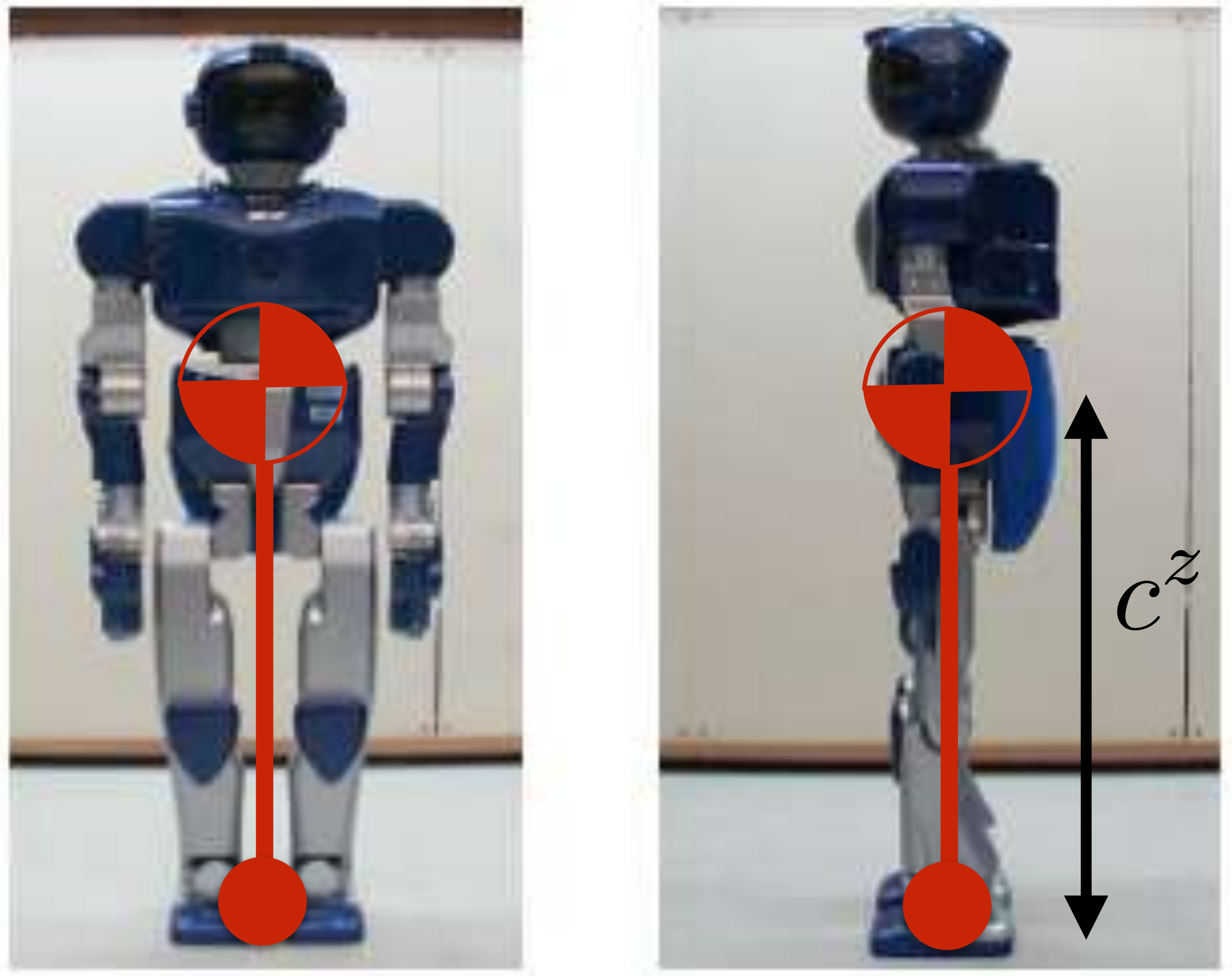
Euler equations (angular momentum)

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

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

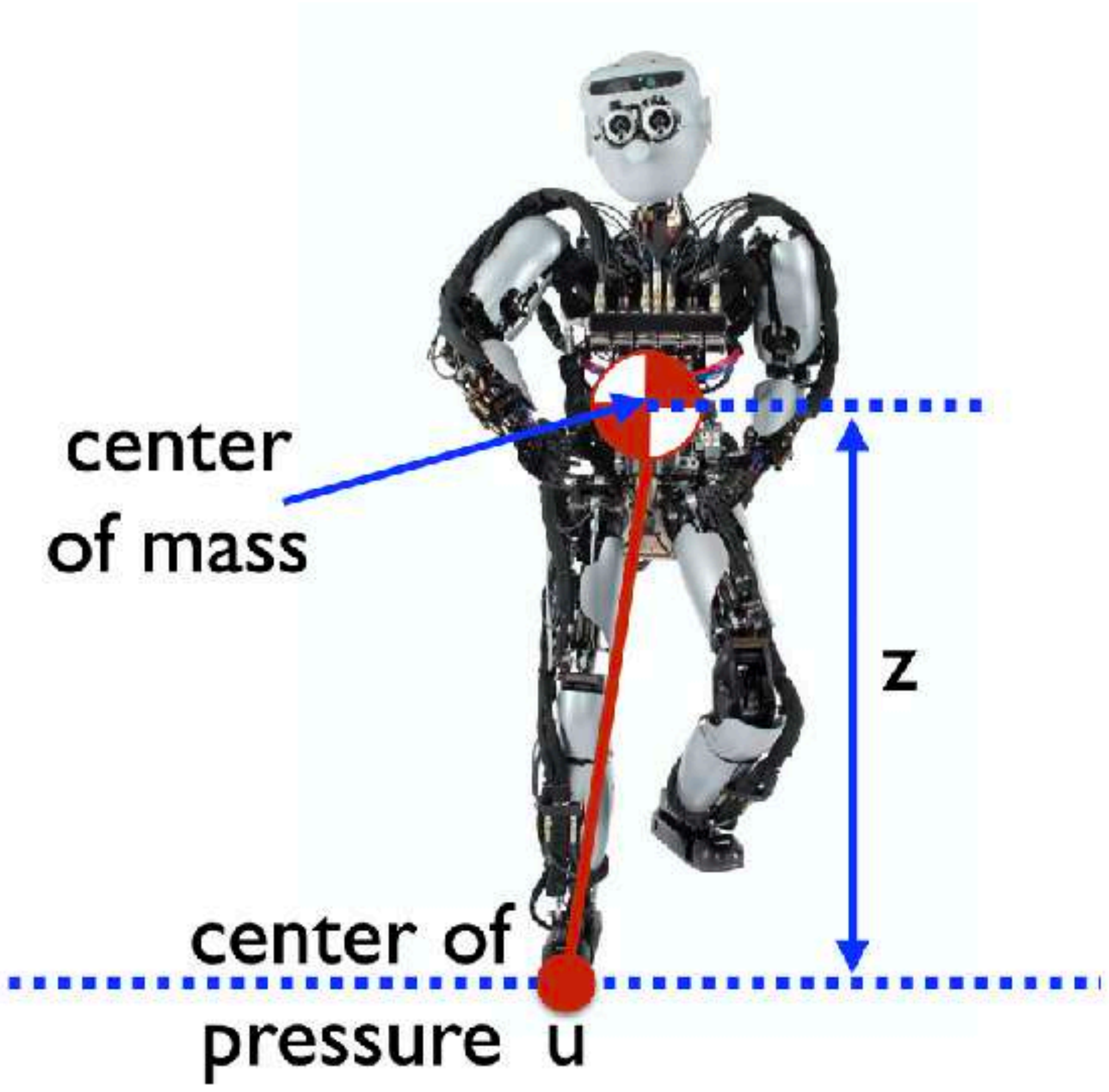


Model predictive control for walking

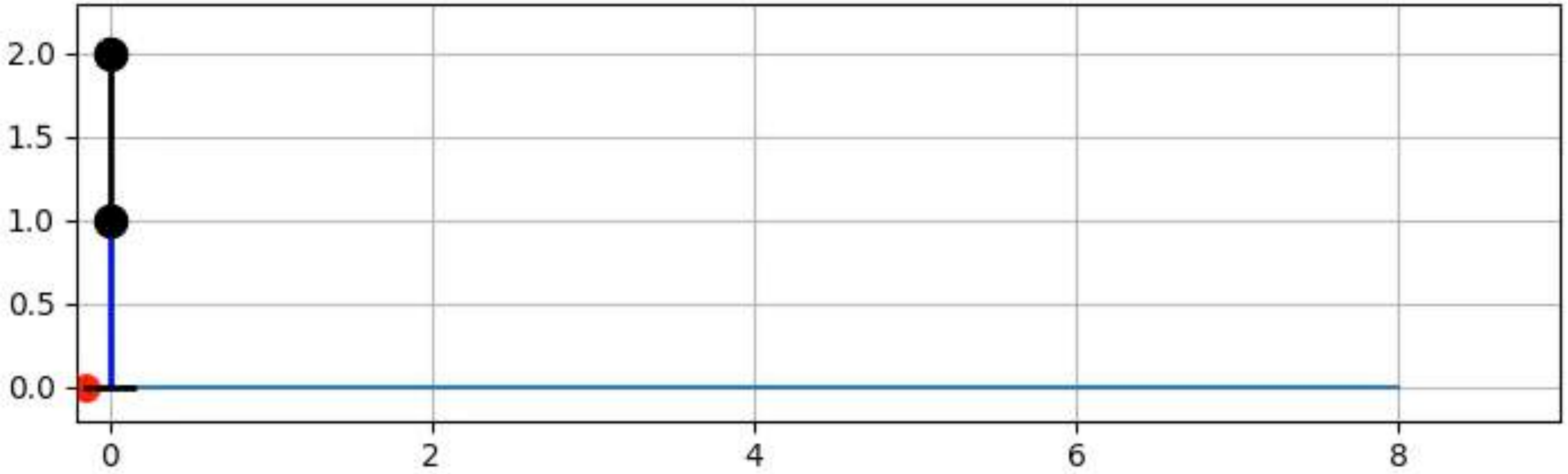
The equations of motion of the LIPM can be written as a function of the CoP

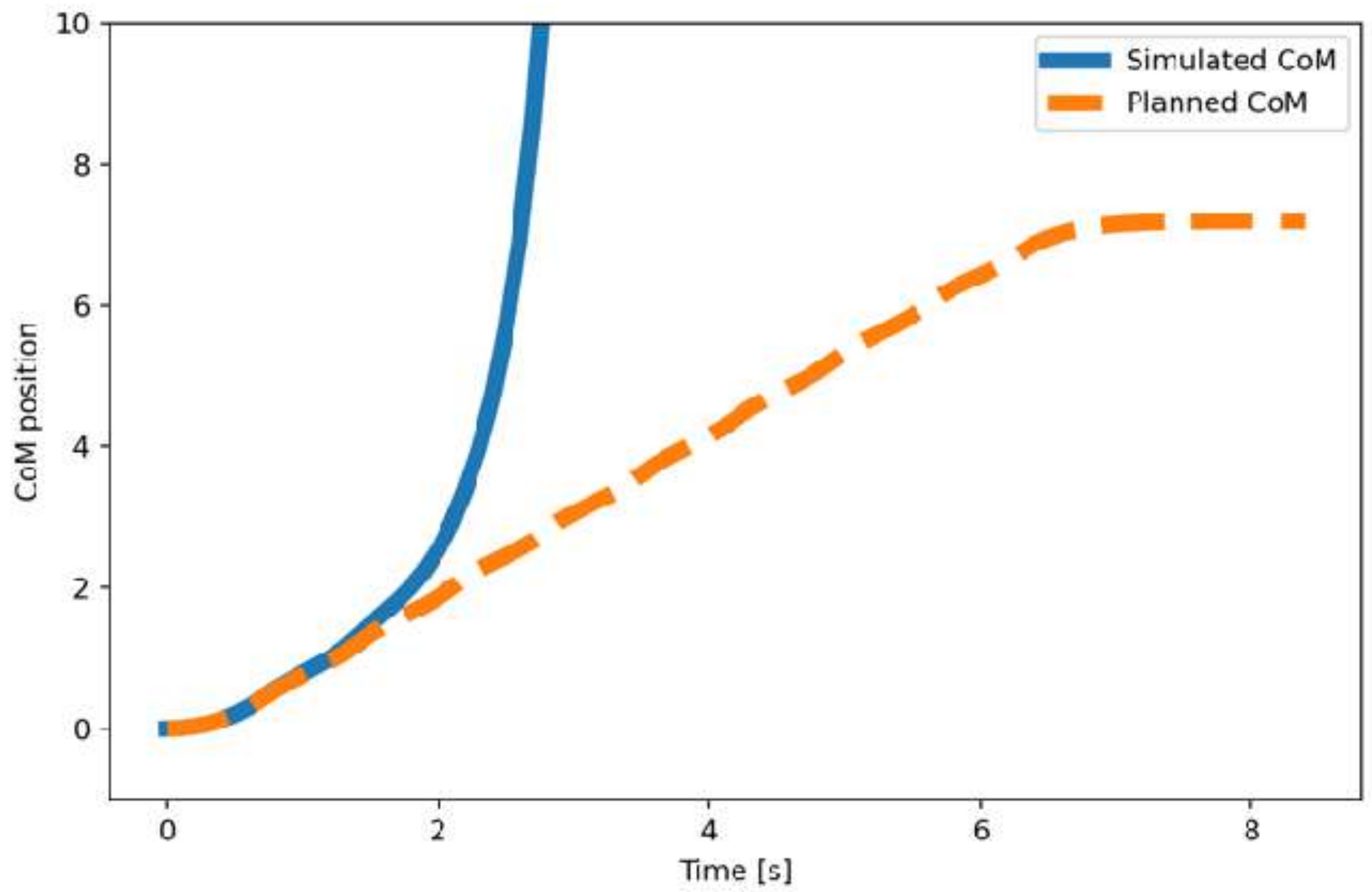
$$\ddot{\mathbf{c}}^x = \frac{g}{c^z}(\mathbf{c}^x - \mathbf{p}^x)$$

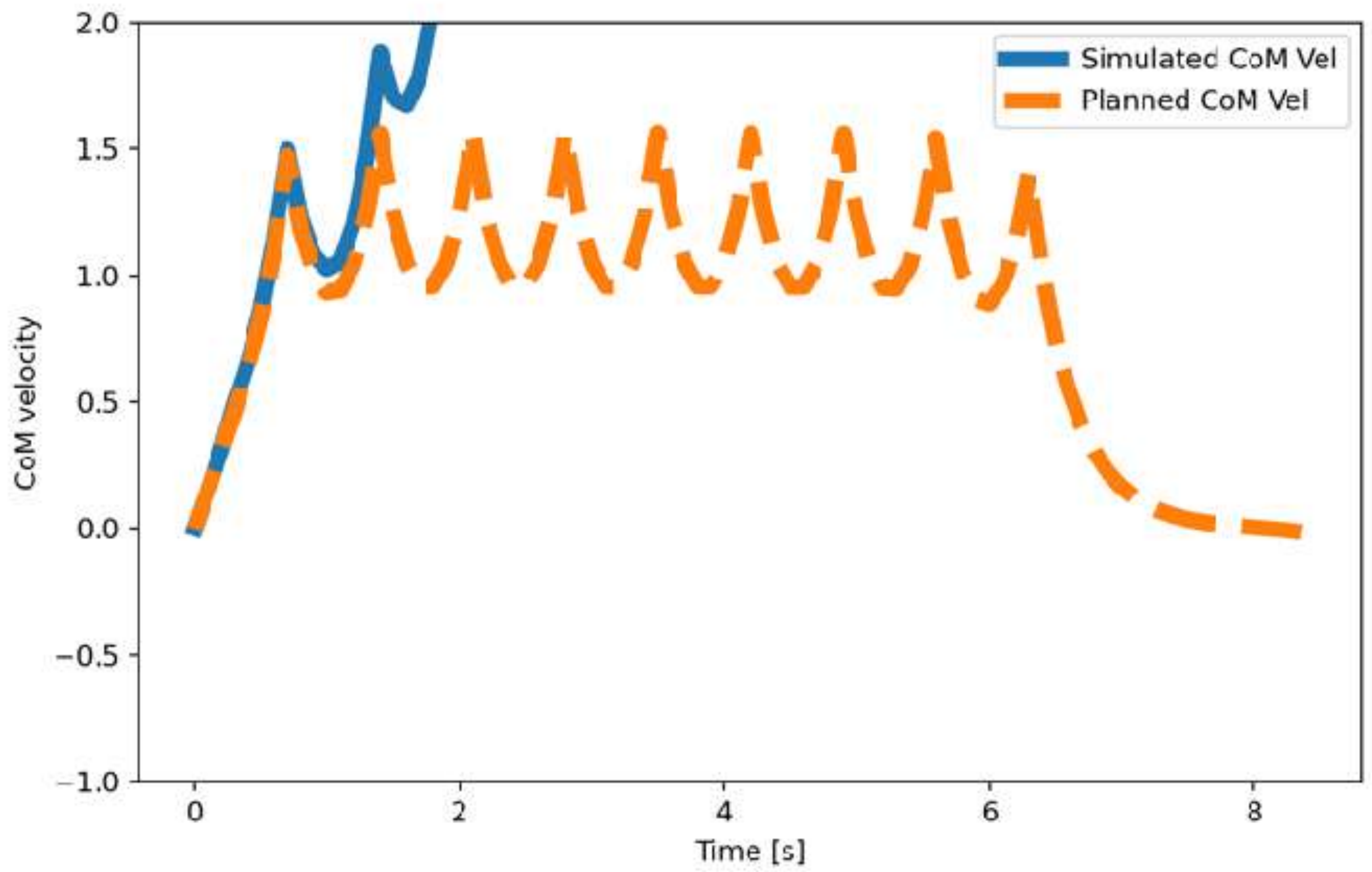
$$\ddot{\mathbf{c}}^y = \frac{g}{c^z}(\mathbf{c}^y - \mathbf{p}^y)$$

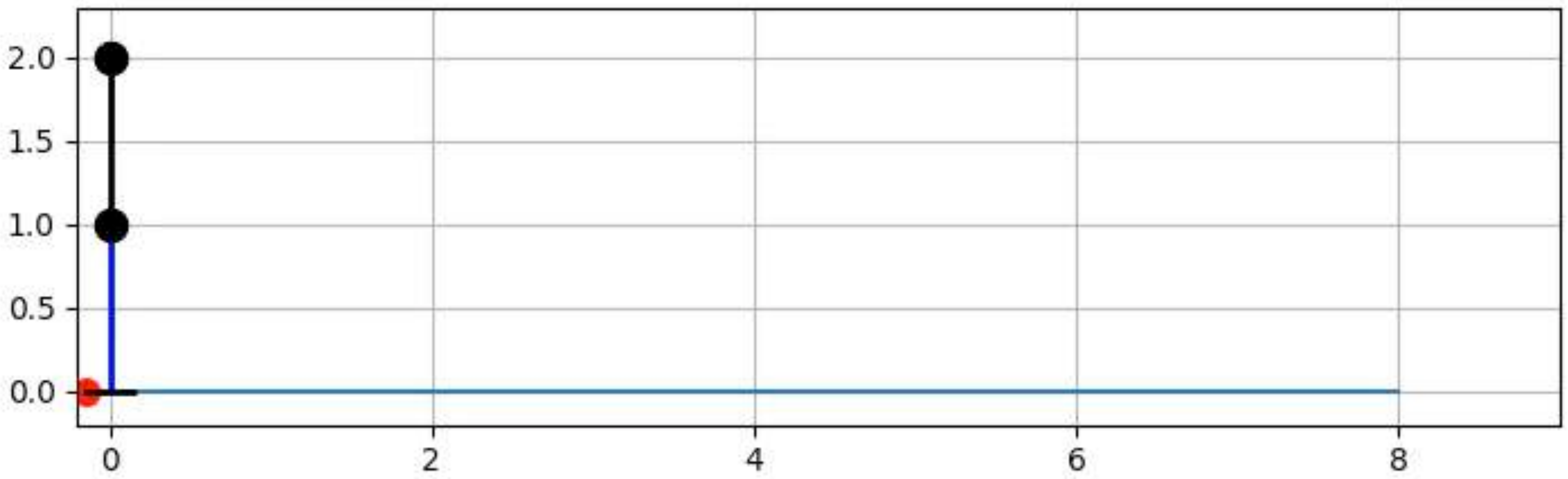


Planned CoM
Actual CoM
CoM position
Time [s]
Actual CoM Vel
Planned CoM Vel
CoM velocity
Time [s]
Actual CoP in foot
Planned CoP in foot
upper foot limit
lower foot limit
CoP position and limits
Time [s]

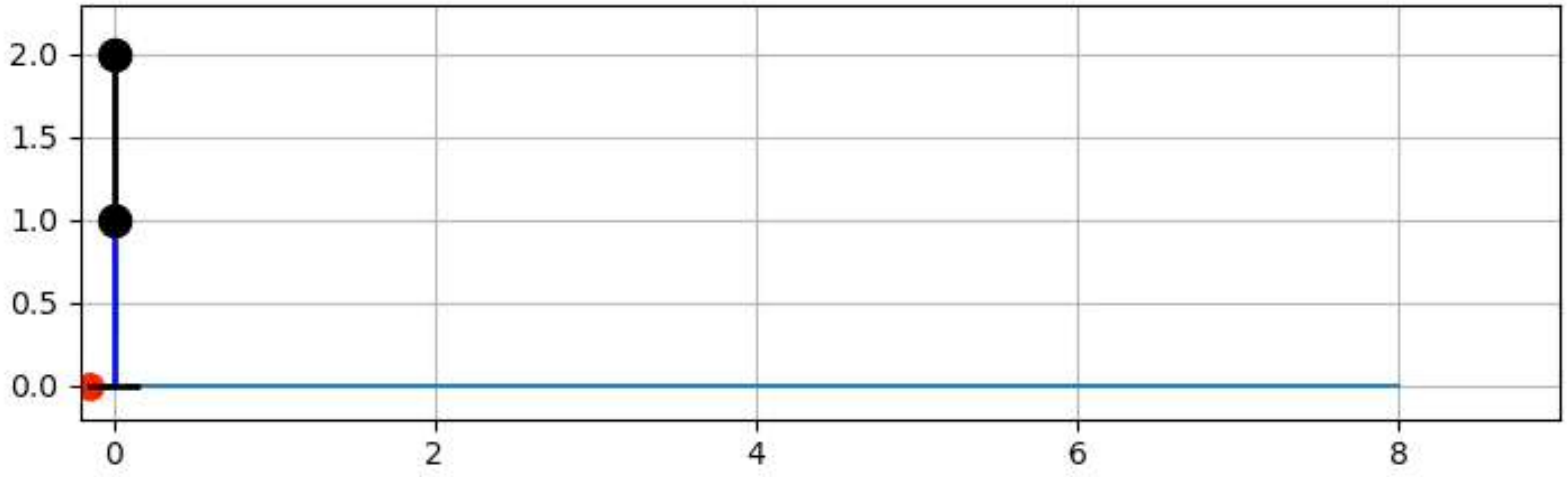



Simulated CoM
Planned CoM
CoM position
Time [s]


Simulated CoM Vel
Planned CoM Vel
CoM velocity
Time [s]



Planned CoM
Actual CoM
CoM position
Time [s]
Actual CoM Vel
Planned CoM Vel
CoM velocity
Time [s]
Actual CoP in foot
Planned CoP in foot
upper foot limit
lower foot limit
CoP position and limits
Time [s]



Control of walking

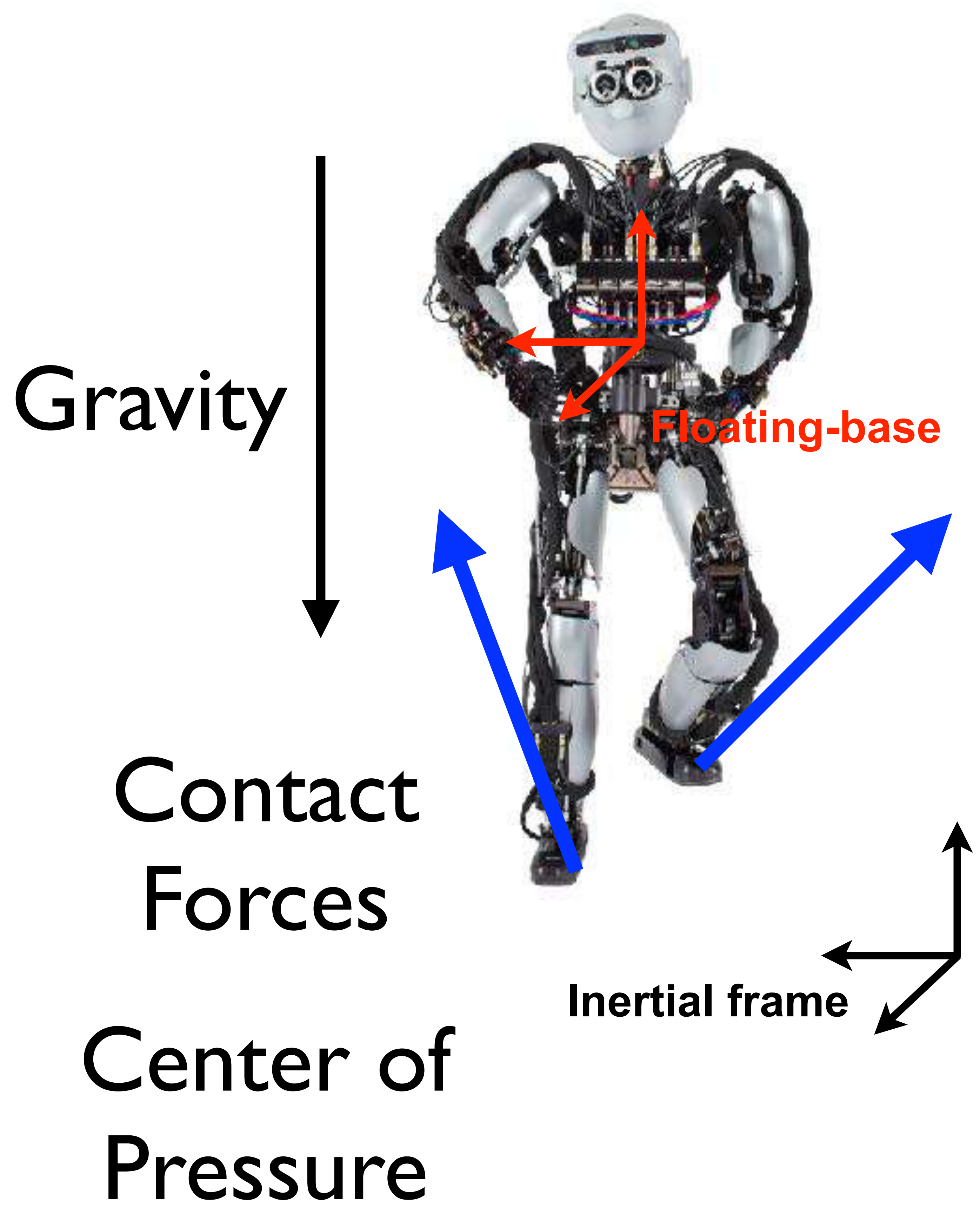
1. Decide where to step (footstep planning)

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



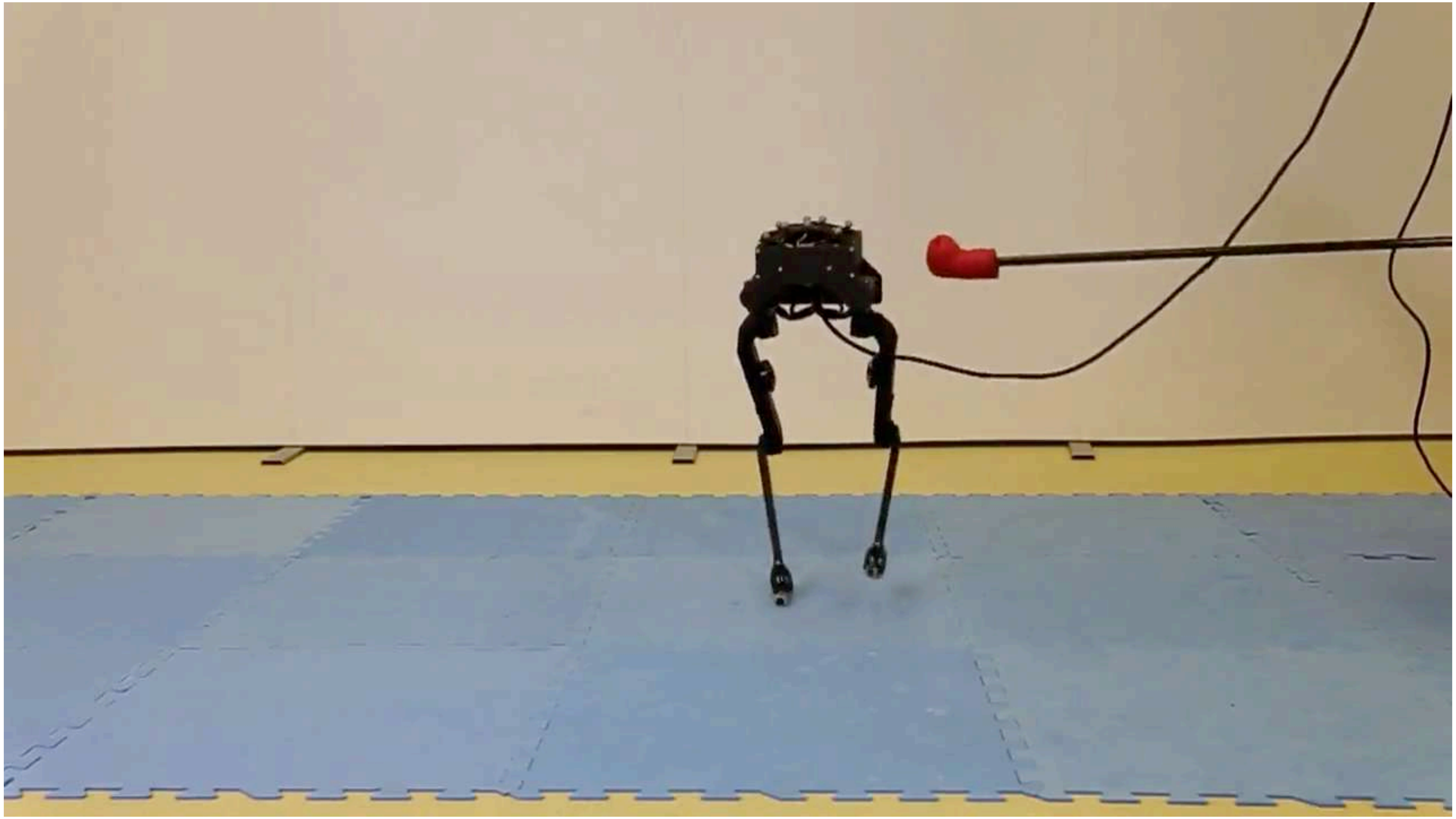
HRP3L - Tokyo Univ. [Urata et al. 2012]

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

[Tesla 2022]

Linear inverted pendulum models are also used in quadruped robots

Linear inverted pendulum models can be used with different stability criterions



Daneshmand et al. RA-L 2021

Nonlinear MPC


NYC RECYCLE