

Underactuated Robots
Optimization methods
for planning and control:
Part 1
Giulio Turrisi

DIPARTIMENTO DI INGEGNERIA INFORMATICA
AUTOMATICA E GESTIONALE ANTONIO RUBERTI



SAPIENZA
UNIVERSITÀ DI ROMA

introduction

- different control techniques exist for fully actuated and underactuated systems, such as:

1. feedback linearization (FL):

- it can transform a fully actuated robot in a simple linear system, easier to control; in the case of underactuation, it linearizes only a part of the dynamics
- it requires a perfect knowledge of the dynamics, and it may need a high amount of control effort to cancel nonlinearities

2. energy based control:

- often used in conjunction with FL
- nice theoretical properties, but they require some “work” to be found

optimization in robotics

- another type of controllers which arise from the solution of an **optimization** problem
- they are general, i.e., they work both for fully actuated or underactuated systems
- they can generate complex behaviors that can be elicited through a user-defined **cost function**, e.g.
 - distance to a goal
 - minimum time
 - control effort

general formulation

- **goal**: we want to find a feedback $\mathbf{u}(\mathbf{x})$ that minimizes the cost function, while satisfying constraints on the input and/or state

$$\begin{aligned} \min_{\mathbf{u}(\mathbf{x})} \quad & J(\mathbf{x}, \mathbf{u}) \\ \text{s.t.} \quad & \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) \\ & \mathbf{x}_{t_0} = \mathbf{x}_0 \\ & \mathbf{h}(\mathbf{x}, \mathbf{u}) \leq 0 \end{aligned}$$

general formulation

- **goal:** we want to find a feedback $u(x)$ that minimizes the cost function, while satisfying constraints on the input and/or state

\min
 $u(x)$



input

$$J(x, u)$$

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

$$x_{t_0} = x_0$$


$$h(x, u) \leq 0$$

general formulation

- **goal:** we want to find a feedback $u(x)$ that minimizes the cost function, while satisfying constraints on the input and/or state

$$\begin{array}{ll} \min & J(x, u) \\ u(x) & \\ \text{s.t.} & \dot{x} = f(x, u) \\ & x_{t_0} = x_0 \\ & h(x, u) \leq 0 \end{array}$$

cost function



general formulation

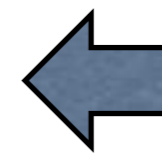
- **goal**: we want to find a feedback $u(x)$ that minimizes the cost function, while satisfying constraints on the input and/or state

$$\min_{u(x)}$$

$$J(x, u)$$

s.t.

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



dynamics (in general nonlinear)

$$x_{t_0} = x_0$$

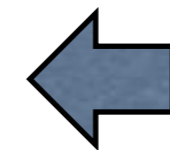
$$h(x, u) \leq 0$$

initial state

general formulation

- **goal**: we want to find a feedback $u(x)$ that minimizes the cost function, while satisfying constraints on the input and/or state

$$\begin{aligned} \min_{u(x)} \quad & J(x, u) \\ \text{s.t.} \quad & \dot{x} = f(x, u) \\ & x_{t_0} = x_0 \\ & h(x, u) \leq 0 \end{aligned}$$



input and/or state constraints, such as:

- limit on the inputs
- maximum velocity

outline of part 1

- we will look to some methods that can be used to solve our optimization problem
 - dynamic programming
 - Hamilton-Jacobi-Bellman (HJB) equation
 - Linear Quadratic Regulator

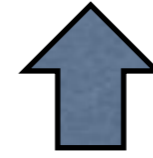
dynamic programming

- a first approach is based on a mathematical optimization method known as **dynamic programming** (DP)
- for systems with a finite, discrete set of states and actions, DP represents a numerical algorithm which can compute an optimal feedback controller $\mathbf{u}(\mathbf{x})$
- for continuous systems, DP provides the foundations for the HJB equation
- in general, it requires an additive cost formulation

$$J(\mathbf{x}, \mathbf{u}) = \int_0^T g(\mathbf{x}, \mathbf{u}) dt + g(\mathbf{x}(T))$$



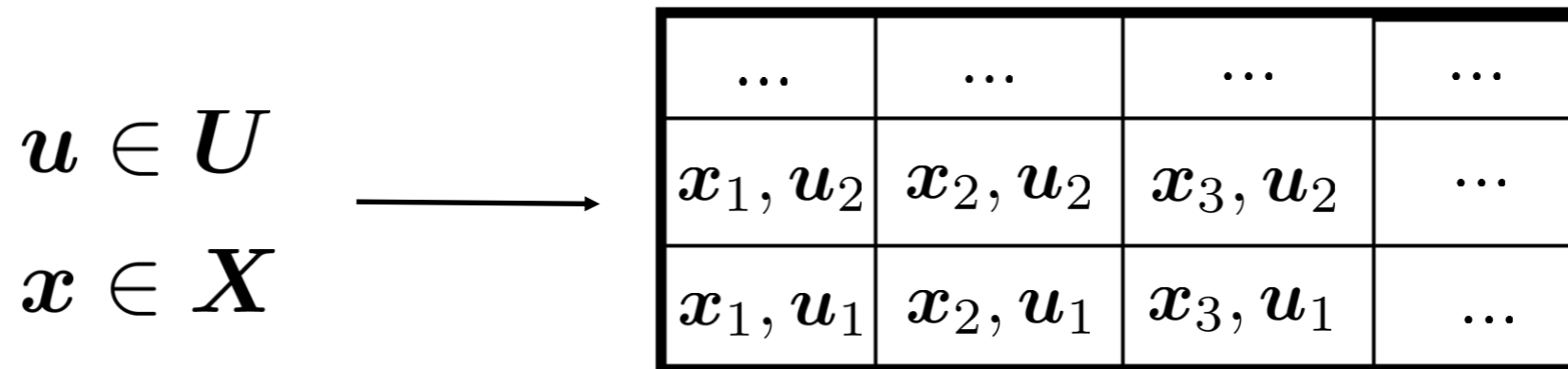
running cost



cost-to-go

dynamic programming

- we will start to look into DP for **discrete system and finite problem**
- DP seeks a solution $\mathbf{u}(\mathbf{x})$ from **any** initial state \mathbf{x}_0 and **requires** a discretization of the input and of the state over a finite dimensional grid



- our discrete problem (finite horizon) can be formulated as

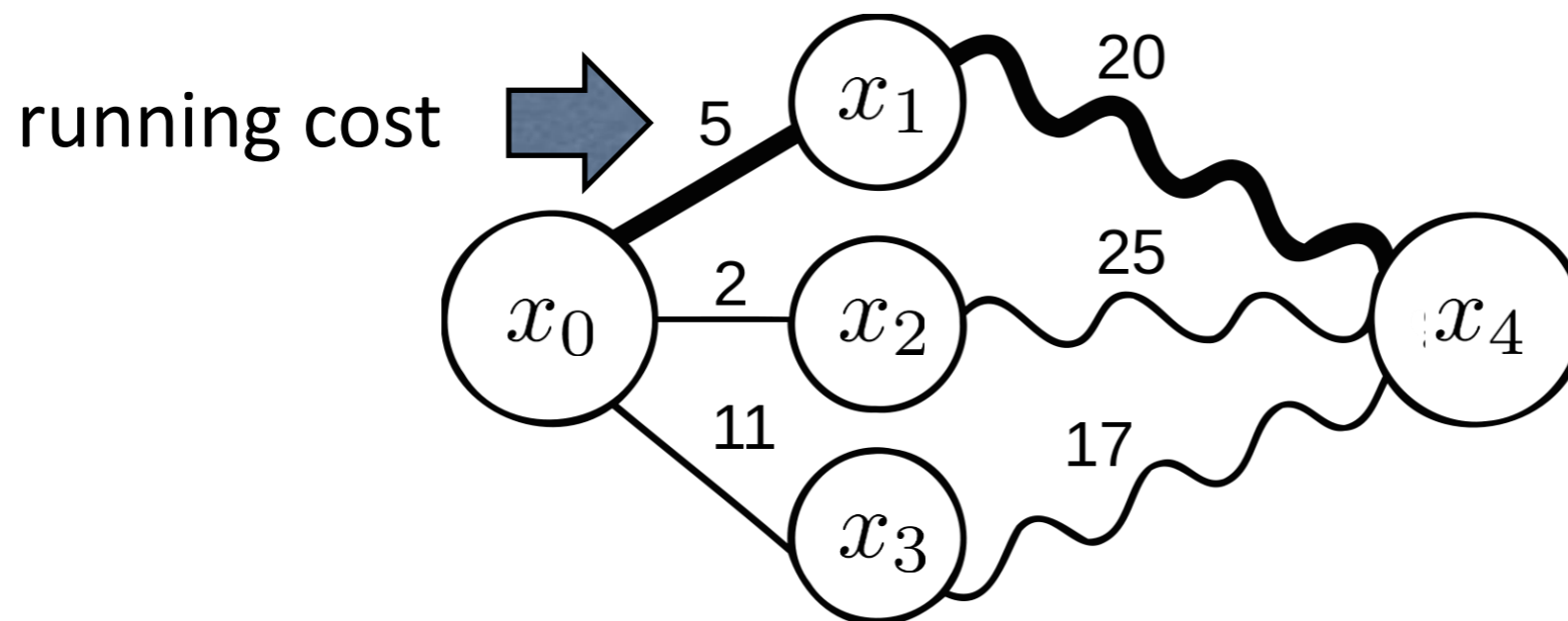
$$\min_{\mathbf{u}_0, \dots, \mathbf{u}_{N-1} \in U} \sum_{k=0}^{N-1} g(\mathbf{x}_k, \mathbf{u}_k) + g_N(\mathbf{x}_N)$$

s.t.

$$\mathbf{x}_{k+1} = \mathbf{f}_d(\mathbf{x}_k, \mathbf{u}_k)$$
$$\mathbf{x}_{k+1} \in X$$

dynamic programming - assumption

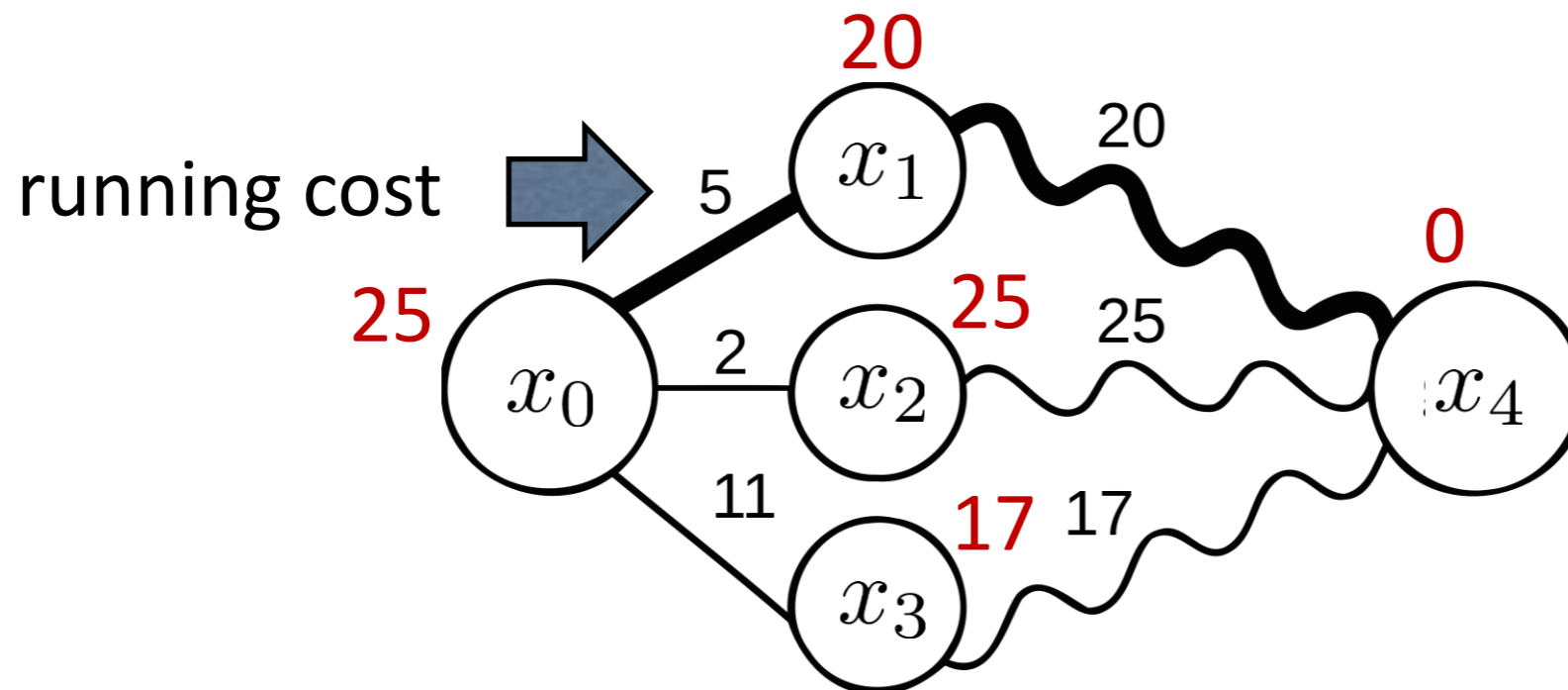
- DP exploits the **Bellman principle of optimality**, which states that in an optimal sequence of decisions, each subsequence must also be optimal
- consider a simple graph problem (shortest path)



- to find the optimal controller $u(x)$ leading from x_0 to the goal, we need to find first the optimal **cost-to-go** for each state

dynamic programming - assumption

- DP exploits the **Bellman principle of optimality**, which states that in an optimal sequence of decisions, each subsequence must also be optimal
- consider a simple graph problem (shortest path)



- to find the optimal controller $u(x)$ leading from x_0 to the goal, we need to find first the optimal **cost-to-go** for each state

dynamic programming - algorithm

- DP searches for the optimal cost-to-go J^* **backward** (starting from the goal) and in an **iterative** fashion. We optimize for

$$J^*(\mathbf{x}_k) = \min_{\mathbf{u}_0, \dots, \mathbf{u}_{N-1} \in U} \sum_{k=0}^{N-1} g(\mathbf{x}_k, \mathbf{u}_k)$$

with the boundary condition (on the final state)

$$J(\mathbf{x}_N) = J^*(\mathbf{x}_N) = g_N(\mathbf{x}_N)$$

- key idea: additive cost and minimization over a single step! **For each discrete state** \mathbf{x}_k we can search just for a single action

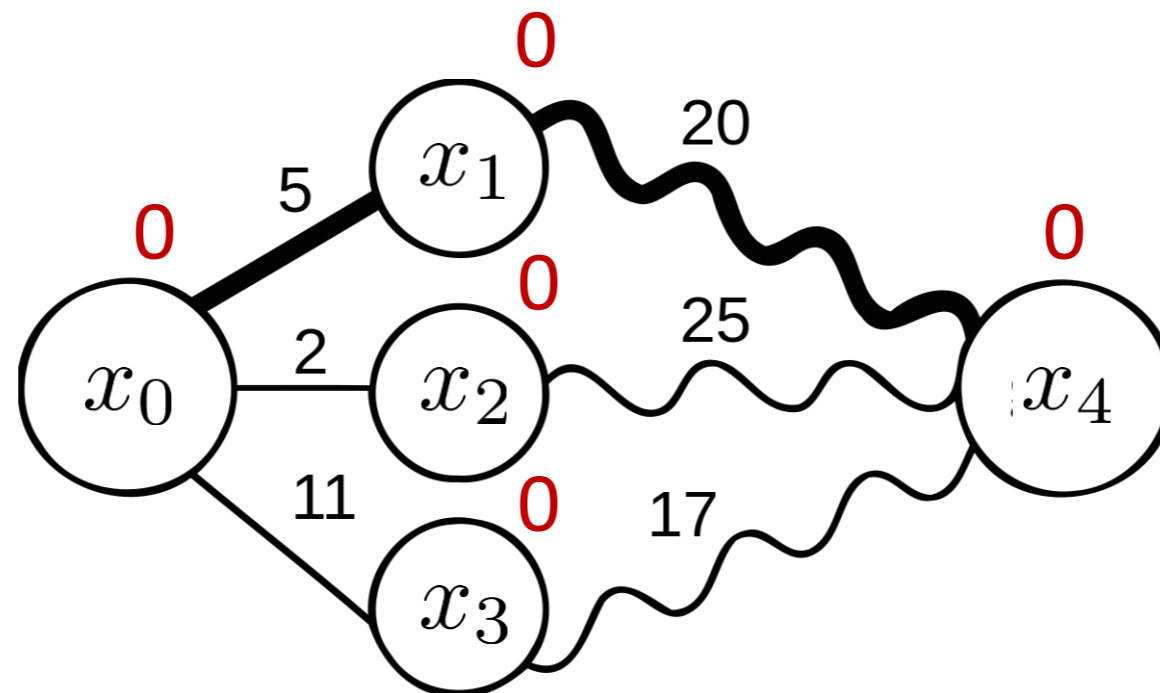
$$J^*(\mathbf{x}_k) = \min_{\mathbf{u}_k \in U} [g(\mathbf{x}_k, \mathbf{u}_k) + J^*(\mathbf{f}_d(\mathbf{x}_k, \mathbf{u}_k))]$$

dynamic programming - algorithm

- J^* is the true unknown optimal cost-to-go that we want to find
- we can start with an initial guess \hat{J}^* (for example zero) and iterate until convergence

$$\hat{J}^*(\mathbf{x}_k) = \min_{\mathbf{u}_k \in U} [g(\mathbf{x}_k, \mathbf{u}_k) + \hat{J}^*(\mathbf{f}_d(\mathbf{x}_k, \mathbf{u}_k))]$$

starting from the goal where we know the cost-to-go!

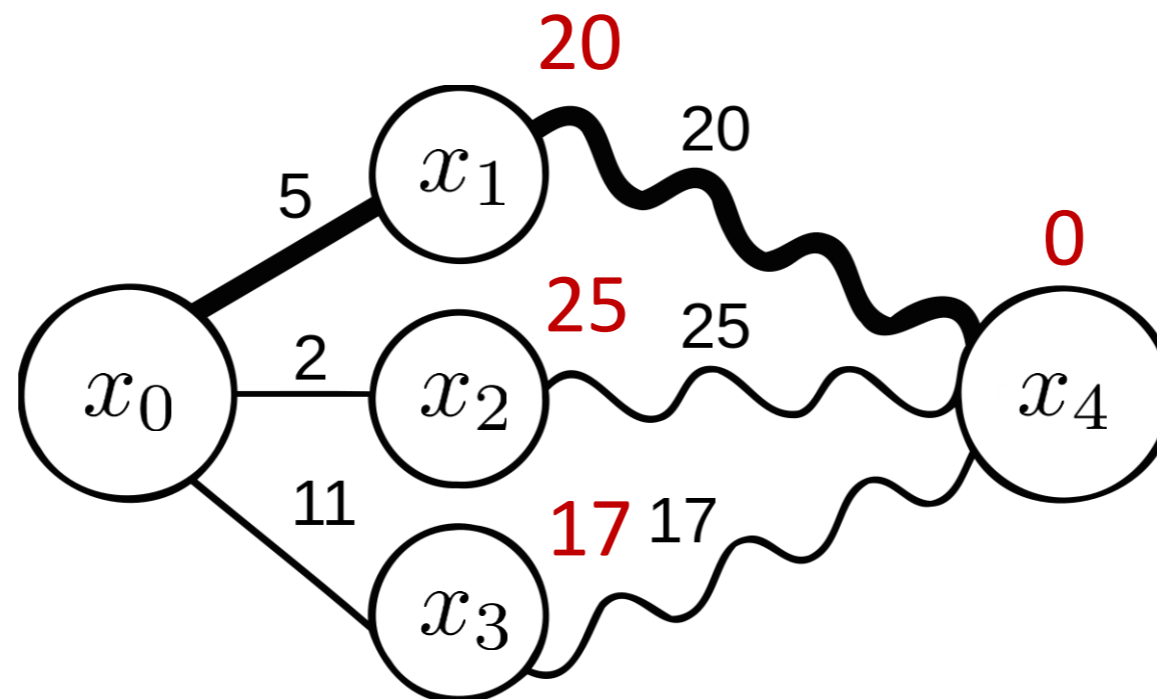


dynamic programming - algorithm

- J^* is the true unknown optimal cost-to-go that we want to find
- we can start with an initial guess \hat{J}^* (for example zero) and iterate until convergence

$$\hat{J}^*(\mathbf{x}_k) = \min_{\mathbf{u}_k \in U} [g(\mathbf{x}_k, \mathbf{u}_k) + \hat{J}^*(\mathbf{f}_d(\mathbf{x}_k, \mathbf{u}_k))]$$

starting from the goal where we know the cost-to-go!

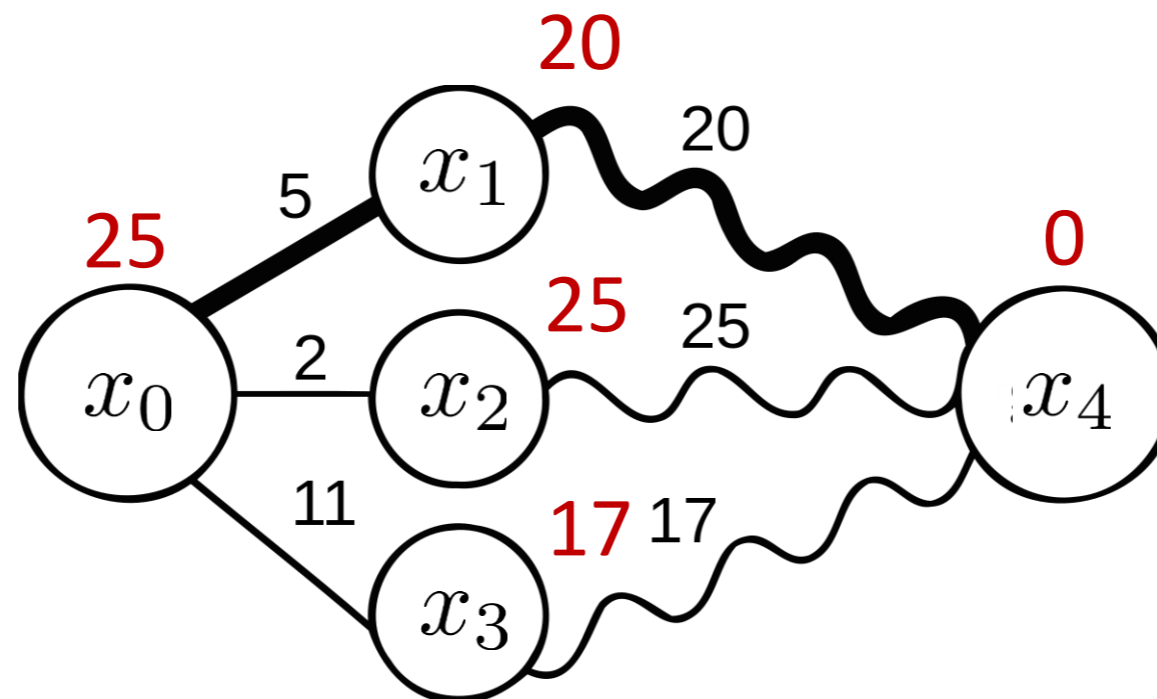


dynamic programming - algorithm

- J^* is the true unknown optimal cost-to-go that we want to find
- we can start with an initial guess \hat{J}^* (for example zero) and iterate until convergence

$$\hat{J}^*(\mathbf{x}_k) = \min_{\mathbf{u}_k \in U} [g(\mathbf{x}_k, \mathbf{u}_k) + \hat{J}^*(\mathbf{f}_d(\mathbf{x}_k, \mathbf{u}_k))]$$

starting from the goal where we know the cost-to-go!



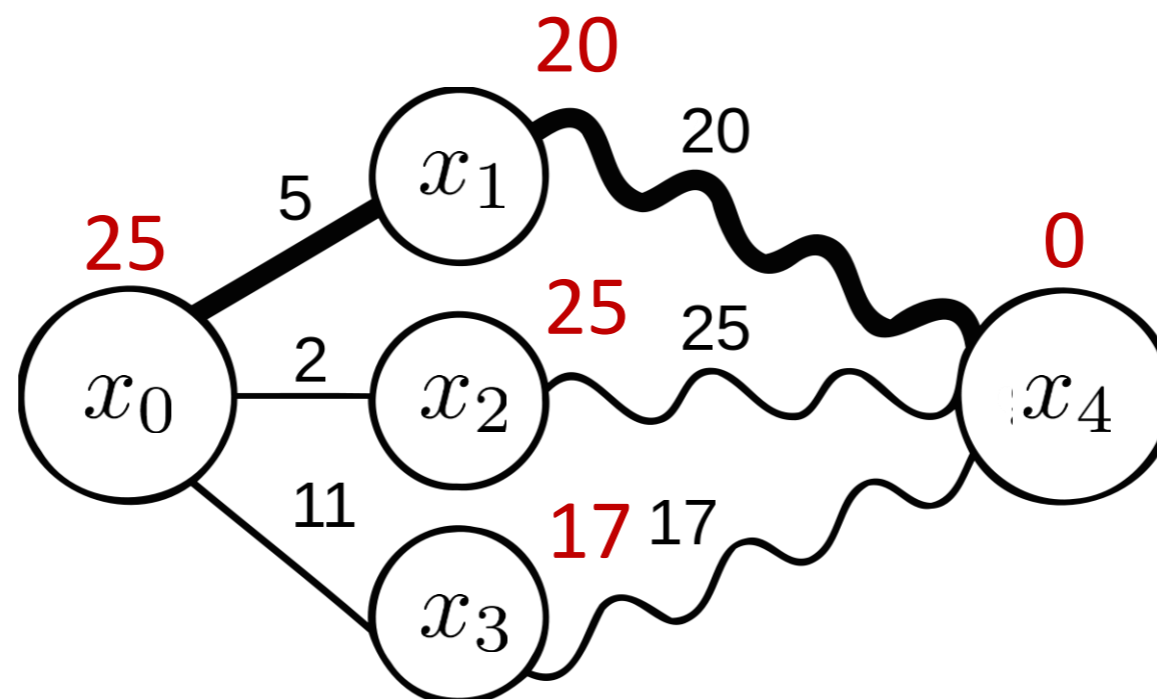
dynamic programming - algorithm

- after each iteration, \hat{J}^* approaches the true optimal cost to go

$$\hat{J}^* \rightarrow J^*$$

- at convergence, we can obtain the optimal control law $\mathbf{u}^*(\mathbf{x})$ as

$$\mathbf{u}^*(\mathbf{x}_k) = \underset{\mathbf{u}_k \in U}{\operatorname{arg\,min}} [g(\mathbf{x}_k, \mathbf{u}_k) + \hat{J}^*(\mathbf{f}_d(\mathbf{x}_k, \mathbf{u}_k))]$$



dynamic programming - algorithm

- after each iteration, \hat{J}^* approaches the true optimal cost to go

$$\hat{J}^* \rightarrow J^*$$

- at convergence, we can obtain the optimal control law $\mathbf{u}^*(\mathbf{x})$ as

$$\mathbf{u}^*(\mathbf{x}_k) = \underset{\mathbf{u}_k \in U}{\operatorname{arg\,min}} [g(\mathbf{x}_k, \mathbf{u}_k) + \hat{J}^*(\mathbf{f}_d(\mathbf{x}_k, \mathbf{u}_k))]$$

- plugging the optimal control law to the previous equation, we have that the variation of the **cost-to-go** depends only on the **running cost**

$$\hat{J}^*(\mathbf{x}_k) - \hat{J}^*(\mathbf{f}_d(\mathbf{x}_k, \mathbf{u}_k^*)) = g(\mathbf{x}_k, \mathbf{u}_k^*)$$

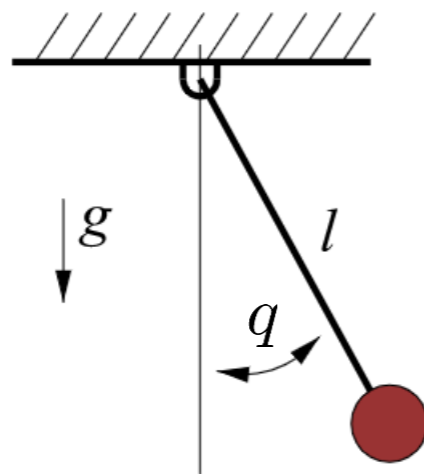
dynamic programming - grid world

- action $u \in \mathbf{U} = [\text{up, down, right, left}]$
- cost function $J(\mathbf{x}, \mathbf{u}) = \begin{cases} 1 & \mathbf{x}_k \neq \mathbf{x}_g \\ 0 & \mathbf{x}_k = \mathbf{x}_g \end{cases}$

				x_g

dynamic programming - pendulum

- suppose we want to find a feedback law $u(\mathbf{x})$ to bring a pendulum to the up equilibrium \mathbf{x}_e
- the state is $\mathbf{x} = (q, \dot{q})$ and the goal is defined as $\mathbf{x}_e = (\pi, 0)$
- we have an input constraint $|u| \leq 1$
- the generic $g(\mathbf{x}_k, \mathbf{u}_k)$ can be taken as the squared distance of \mathbf{x}_k from \mathbf{x}_e



$$I\ddot{q} + mg_0l \sin q + b\dot{q} = u$$

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

dynamic programming - pendulum

- we start discretizing the state and input space of our robot

$$q \in [-3.14, -3.10, \dots, 3.14]$$

$$\dot{q} \in [-10, -9.9, \dots, 10]$$

$$u \in [-1, -0.95, \dots, 1]$$

and discretizing $f(\cdot)$ over time

- the smaller is the discretization step used, the more accurate will be the final solution; obviously at the **expense** of the time required to solve the optimization problem
- $f_d(\cdot)$ will be a function that will map $f_k(\mathbf{x}_k, u_k)$ to a successor state $\mathbf{x}_{k+1} \in \mathbf{X}$

dynamic programming - limitations

- DP is a powerful optimization technique that has two major drawbacks that may prevent its use in practice:
 - **accuracy**: this is due to the discretization of the input and state of the robot
 - **scalability** (curse of dimensionality): as the dimension of the state and input spaces increases, the computational complexity may become prohibitive

HJB equation

- we can bring DP over a continuous state space and time, obtaining the **Hamilton - Jacobi - Bellman** (HJB) equation
- suppose we have a continuous system $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$ and a running additive cost

$$J(\mathbf{x}, \mathbf{u}) = \int_0^T g(\mathbf{x}, \mathbf{u}) dt + g(\mathbf{x}(T)) \quad \leftarrow \text{finite horizon formulation}$$

- the HJB equation assumes the form of

$$-\frac{\partial J^*(t, \mathbf{x})}{\partial t} = \min_{\mathbf{u}} \left[g(\mathbf{x}, \mathbf{u}) + \frac{\partial J^*(t, \mathbf{x})}{\partial \mathbf{x}} \mathbf{f}(\mathbf{x}, \mathbf{u}) \right]$$

(note that J^* depend also on time)

HJB equation - comparison

- DP and HJB are very similar

DP

$$\mathbf{x}_{k+1} = \mathbf{f}_d(\mathbf{x}_k, \mathbf{u}_k)$$

$$\sum_{k=0}^{N-1} g(\mathbf{x}_k, \mathbf{u}_k) + g_N(\mathbf{x}_N)$$

$$J^*(\mathbf{x}_k) = \min_{\mathbf{u}_k} [g(\mathbf{x}_k, \mathbf{u}_k) + J^*(\mathbf{f}_d(\mathbf{x}_k, \mathbf{u}_k))]$$

HJB

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

$$\int_0^T g(\mathbf{x}, \mathbf{u}) dt + g_T(\mathbf{x}(T))$$

$$-\frac{\partial J^*}{\partial t} = \min_{\mathbf{u}} [g(\mathbf{x}, \mathbf{u}) + \frac{\partial J^*}{\partial \mathbf{x}} \mathbf{f}(\mathbf{x}, \mathbf{u})]$$

HJB equation - derivation

- an “informal” derivation using DP and a discrete time system
- we can discretize the final time T in N pieces, using the discretization interval

$$\delta = \frac{T}{N}$$

- we can approximate the continuous time system and the cost as

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)\delta$$

$$J(\mathbf{x}, \mathbf{u}) = \sum_{k=0}^N g(\mathbf{x}_k, \mathbf{u}_k)\delta$$

- the DP equation is

$$\tilde{J}^*(k\delta, \mathbf{x}) = \min_{\mathbf{u}} [g(\mathbf{x}, \mathbf{u})\delta + J^*((k+1)\delta, \mathbf{x} + \mathbf{f}(\mathbf{x}, \mathbf{u})\delta)]$$

HJB equation - derivation

$$\tilde{J}^*(k\delta, \mathbf{x}) = \min_{\mathbf{u}} [g(\mathbf{x}, \mathbf{u})\delta + J^*((k+1)\delta, \mathbf{x} + \mathbf{f}(\mathbf{x}, \mathbf{u})\delta)]$$

- we can expand with Taylor the last term

$$\tilde{J}^*((k+1)\delta, \mathbf{x} + \mathbf{f}(\mathbf{x}, \mathbf{u})\delta) =$$

$$\tilde{J}^*(k\delta, \mathbf{x}) + \frac{\partial \tilde{J}^*}{\partial t} \delta + \frac{\partial \tilde{J}^*}{\partial \mathbf{x}} \mathbf{f}(\mathbf{x}, \mathbf{u})\delta$$

- and plug it back in the DP equation

$$\cancel{\tilde{J}^*(k\delta, \mathbf{x})} = \min_{\mathbf{u}} [g(\mathbf{x}, \mathbf{u})\delta + \cancel{\tilde{J}^*(k\delta, \mathbf{x})} + \frac{\partial J^*(k\delta, \mathbf{x})}{\partial t} \delta + \frac{\partial J^*(k\delta, \mathbf{x})}{\partial \mathbf{x}} \mathbf{f}(\mathbf{x}, \mathbf{u})\delta]$$

HJB equation - derivation

$$0 = \min_{\mathbf{u}} \left[g(\mathbf{x}, \mathbf{u})\delta + \frac{\partial J^*(k\delta, \mathbf{x})}{\partial t} \delta + \frac{\partial J^*(k\delta, \mathbf{x})}{\partial \mathbf{x}} \mathbf{f}(\mathbf{x}, \mathbf{u})\delta \right]$$



$$-\frac{\partial J^*(k\delta, \mathbf{x})}{\partial t} \delta = \min_{\mathbf{u}} \left[g(\mathbf{x}, \mathbf{u})\delta + \frac{\partial J^*(k\delta, \mathbf{x})}{\partial \mathbf{x}} \mathbf{f}(\mathbf{x}, \mathbf{u})\delta \right]$$

- we can now divide by δ and take the limit for $\delta \rightarrow 0$ obtaining

$$-\frac{\partial J^*(t, \mathbf{x})}{\partial t} = \min_{\mathbf{u}} \left[g(\mathbf{x}, \mathbf{u}) + \frac{\partial J^*(t, \mathbf{x})}{\partial \mathbf{x}} \mathbf{f}(\mathbf{x}, \mathbf{u}) \right]$$

HJB equation - solution

- if the system is control-affine

$$\dot{\mathbf{x}} = \mathbf{f}_1(\mathbf{x}) + \mathbf{f}_2(\mathbf{x})\mathbf{u}$$

and we restrict the running cost to be quadratic in the input \mathbf{u}

$$g(\mathbf{x}, \mathbf{u}) = g_1(\mathbf{x}) + \mathbf{u}'\mathbf{R}\mathbf{u}$$

we can rewrite the HJB equation as

$$-\frac{\partial J^*(t, \mathbf{x})}{\partial t} = \min_{\mathbf{u}} \left[g_1(\mathbf{x}) + \mathbf{u}'\mathbf{R}\mathbf{u} + \frac{\partial J^*(t, \mathbf{x})}{\partial \mathbf{x}} (\mathbf{f}_1(\mathbf{x}) + \mathbf{f}_2(\mathbf{x})\mathbf{u}) \right]$$

- we can solve it computing the gradient over \mathbf{u}

$$\mathbf{u}^* = -\frac{1}{2}\mathbf{R}^{-1}\mathbf{f}_2(\mathbf{x})' \frac{\partial J^*(t, \mathbf{x})'}{\partial \mathbf{x}}$$

↑
unknown

Linear Quadratic Regulator

- J^* can be found **numerically** also in the continuous formulation
- some problems can even have a **closed form solution** of the HJB equation

$$\mathbf{HJB} \quad \longrightarrow \quad -\frac{\partial J^*}{\partial t} = \min_{\mathbf{u}} [g(\mathbf{x}, \mathbf{u}) + \frac{\partial J^*}{\partial \mathbf{x}} \mathbf{f}(\mathbf{x}, \mathbf{u})]$$

- for linear system the resulting optimal control is called Linear Quadratic Regulator (**LQR**)
- for nonlinear system, one can solve a LQR problem for the linearized system at an equilibrium point
- this solution is valid only in a neighborhood of the equilibrium point (more on this later)

Linear Quadratic Regulator - derivation

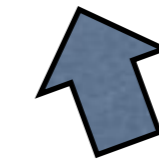
- consider the linear system

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$

- and the quadratic cost

$$J(\mathbf{x}, \mathbf{u}) = \mathbf{x}(T)' \mathbf{Q}_f \mathbf{x}(T) + \int_0^T (\mathbf{x}(t)' \mathbf{Q} \mathbf{x}(t) + \mathbf{u}(t)' \mathbf{R} \mathbf{u}(t)) dt$$

where $\mathbf{Q}_f, \mathbf{Q} \geq 0, \mathbf{R} > 0$ are usually diagonal



control effort

- the HJB equation is

$$0 = \min_{\mathbf{u}} [\mathbf{x}' \mathbf{Q} \mathbf{x} + \mathbf{u}' \mathbf{R} \mathbf{u} + \frac{\partial J^*}{\partial t} + \frac{\partial J^*}{\partial \mathbf{x}} (\mathbf{A} \mathbf{x} + \mathbf{B} \mathbf{u})]$$

with the boundary condition $J^*(T, \mathbf{x}) = \mathbf{x}(T)' \mathbf{Q}_f \mathbf{x}(T)$

Linear Quadratic Regulator - derivation

$$0 = \min_u [x' Q x + u' R u + \frac{\partial J^*}{\partial t} + \frac{\partial J^*}{\partial x} (A x + B u)]$$

- we can search for a solution of the form $J^*(t, x) = x' P(t) x$ obtaining

$$\frac{\partial J^*}{\partial t} = x' \dot{P}(t) x$$

$$\frac{\partial J^*}{\partial x} = 2x' P(t)$$

$$P(t) = P'(t) \geq 0$$

- the HJB becomes

$$0 = \min_u [x' Q x + u' R u + x' \dot{P}(t) x + 2x' P(t) (A x + B u)]$$

- we can obtain the minimum computing the gradient w.r.t u (being R non-singular)

$$2B P(t) x + 2u' R = 0$$

$$u = -R^{-1} B' P(t) x$$

to be found

Linear Quadratic Regulator - derivation

- substituting back u in the HJB equation we obtain

$$0 = x'(\dot{P}(t) + P(t)A + A'P(t) - P(t)BR^{-1}B'P(t) + Q)x$$

- in order to solve the HJB, $P(t)$ should satisfy the continuous time **Riccati equation**

$$\dot{P}(t) = -(P(t)A + A'P(t) - P(t)BR^{-1}B'P(t) + Q)$$

with the boundary condition $P(T) = Q_f$

- we can solve this numerically (different solvers are available)! Once $P(t)$ is found, the optimal control law becomes

$$u^* = -R^{-1}B'P(t)x$$

Linear Quadratic Regulator - infinite horizon

- we can solve the LQR in the **infinite horizon** case

$$J(\mathbf{x}, \mathbf{u}) = \int_0^{\infty} \mathbf{x}(t)' \mathbf{Q} \mathbf{x}(t) + \mathbf{u}(t)' \mathbf{R} \mathbf{u}(t) dt$$

- the cost to go does not depend anymore from time

$$\frac{\partial J^*}{\partial t} = 0$$

- the Riccati equation becomes the **Algebraic Riccati Equation (ARE)**

$$0 = -(\mathbf{P} \mathbf{A} + \mathbf{A}' \mathbf{P} - \mathbf{P} \mathbf{B} \mathbf{R}^{-1} \mathbf{B}' \mathbf{P} + \mathbf{Q})$$

and the controller becomes a static state feedback

$$\mathbf{u}^* = -\mathbf{R}^{-1} \mathbf{B}' \mathbf{P} \mathbf{x}$$



it does not depend on time anymore

Linear Quadratic Regulator - linearization

- for a nonlinear system, LQR is usually activated when the system is near an equilibrium point
- in this case, since LQR works for linear system, we need to perform a **linearization** of the state equations
- we start from the nonlinear system $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$ and we find an equilibrium point $(\boldsymbol{x}_e, \boldsymbol{u}_e)$ such that $\boldsymbol{f}(\boldsymbol{x}_e, \boldsymbol{u}_e) = 0$
- we then do a change of coordinates

$$\hat{\boldsymbol{x}} = \boldsymbol{x} - \boldsymbol{x}_e \quad \hat{\boldsymbol{u}} = \boldsymbol{u} - \boldsymbol{u}_e$$



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

Linear Quadratic Regulator - linearization

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

- and expand $f(x, u)$ with a first-order Taylor approximation

$$\dot{\hat{x}} = \cancel{f(x_e, u_e)} + \left. \frac{\partial f}{\partial x} \right|_{x_e, u_e} (x - x_e) + \left. \frac{\partial f}{\partial u} \right|_{x_e, u_e} (u - u_e)$$



$$\dot{\hat{x}} = A\hat{x} + B\hat{u}$$

- the resulting LQR controller assumes the form of

$$\hat{u} = -R^{-1}BP(x - x_e)$$

Linear Quadratic Regulator - Pendubot

- for underactuated robots, LQR is activated when the system is near an equilibrium point
- example: Pendubot, 2R robot in the vertical plane
- state: $\mathbf{x} = (q_1, q_2, \dot{q}_1, \dot{q}_2)$
- only the **first link** q_1 is actuated, q_2 is passive
- we want to swing-up the system from the stable equilibrium down-down (the two links point downward) $\mathbf{x}_0 = (0, 0, 0, 0)$, to the up-up equilibrium $\mathbf{x}_e = (\pi, 0, 0, 0)$
- design a LQR to work in the neighborhood of \mathbf{x}_e

Linear Quadratic Regulator - Pendubot

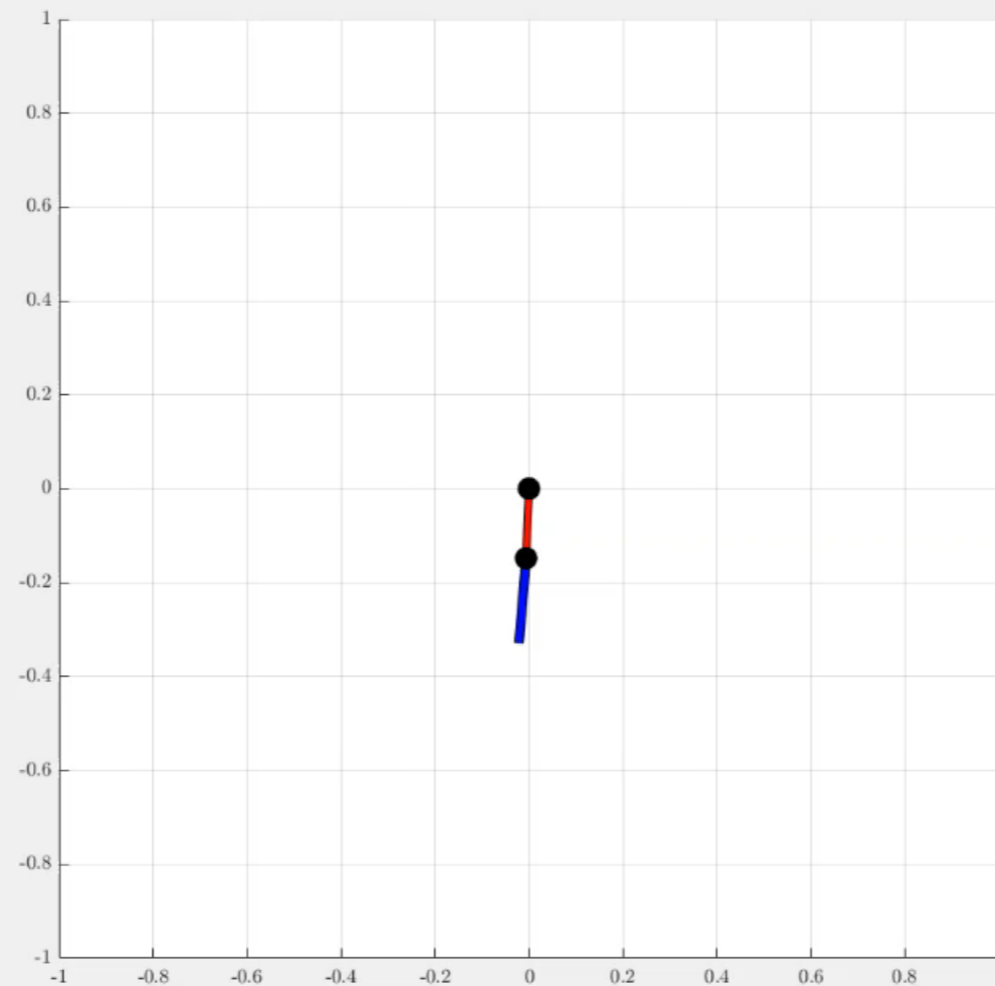
- starting from the initial state x_0 we can apply (for example) an energy based controller that can bring the system **near** the equilibrium point x_e (inside the **basin of attraction**)
- then we can **switch** the controller to LQR to stabilize the system around x_e
- to design the LQR, we need to **linearize** the Pendubot equations around x_e and choose $Q \geq 0, R > 0$

diagonal 4x4 matrix

scalar

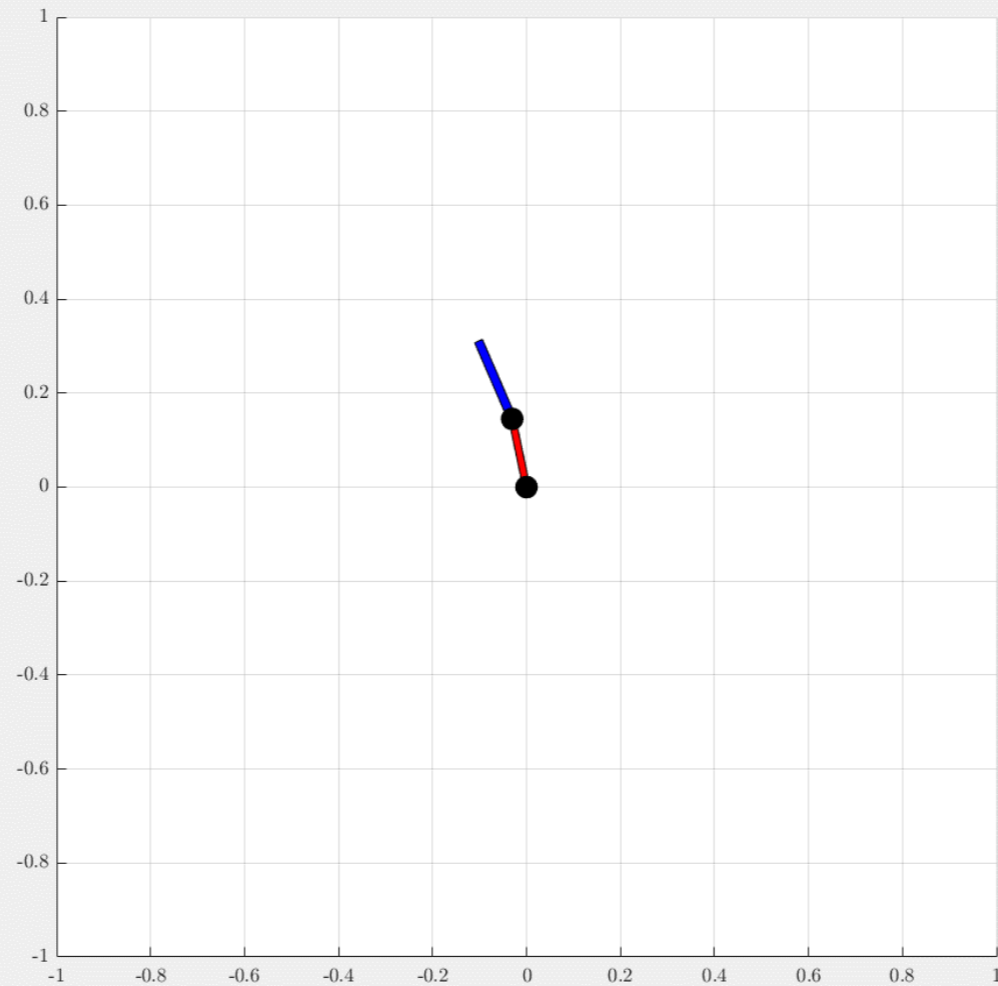
Linear Quadratic Regulator - Pendubot

- swing-up maneuver with an energy based controller and stabilization with the LQR



Linear Quadratic Regulator - Pendubot

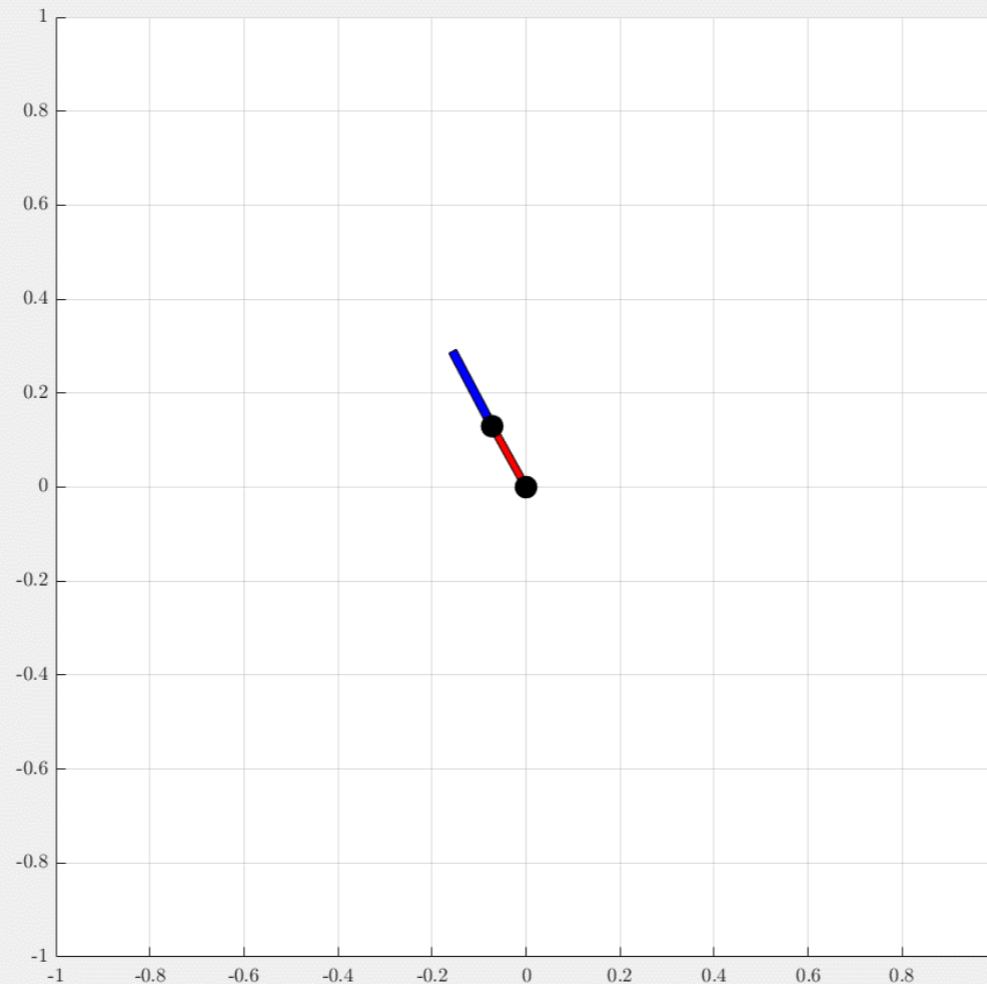
- LQR activated **inside** the basin of attraction



$$\mathbf{x}_0 = (3.35, 0.2011, -1.0, 0.017)$$

Linear Quadratic Regulator - Pendubot

- LQR activated **outside** the basin of attraction - the linearization (and hence the LQR) is no longer valid

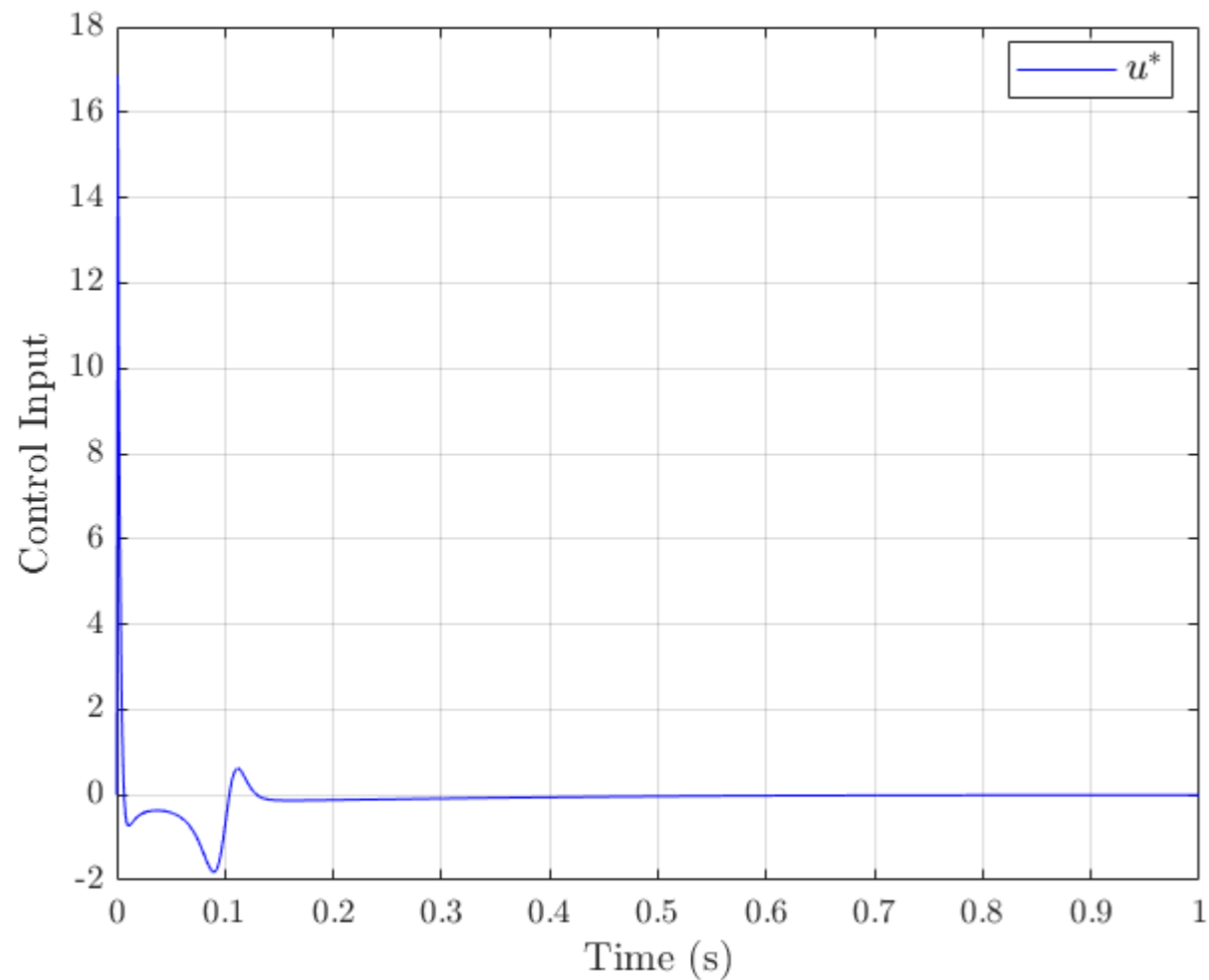


$$\mathbf{x}_0 = (3.7, -0.2, -2.0, 0.017)$$

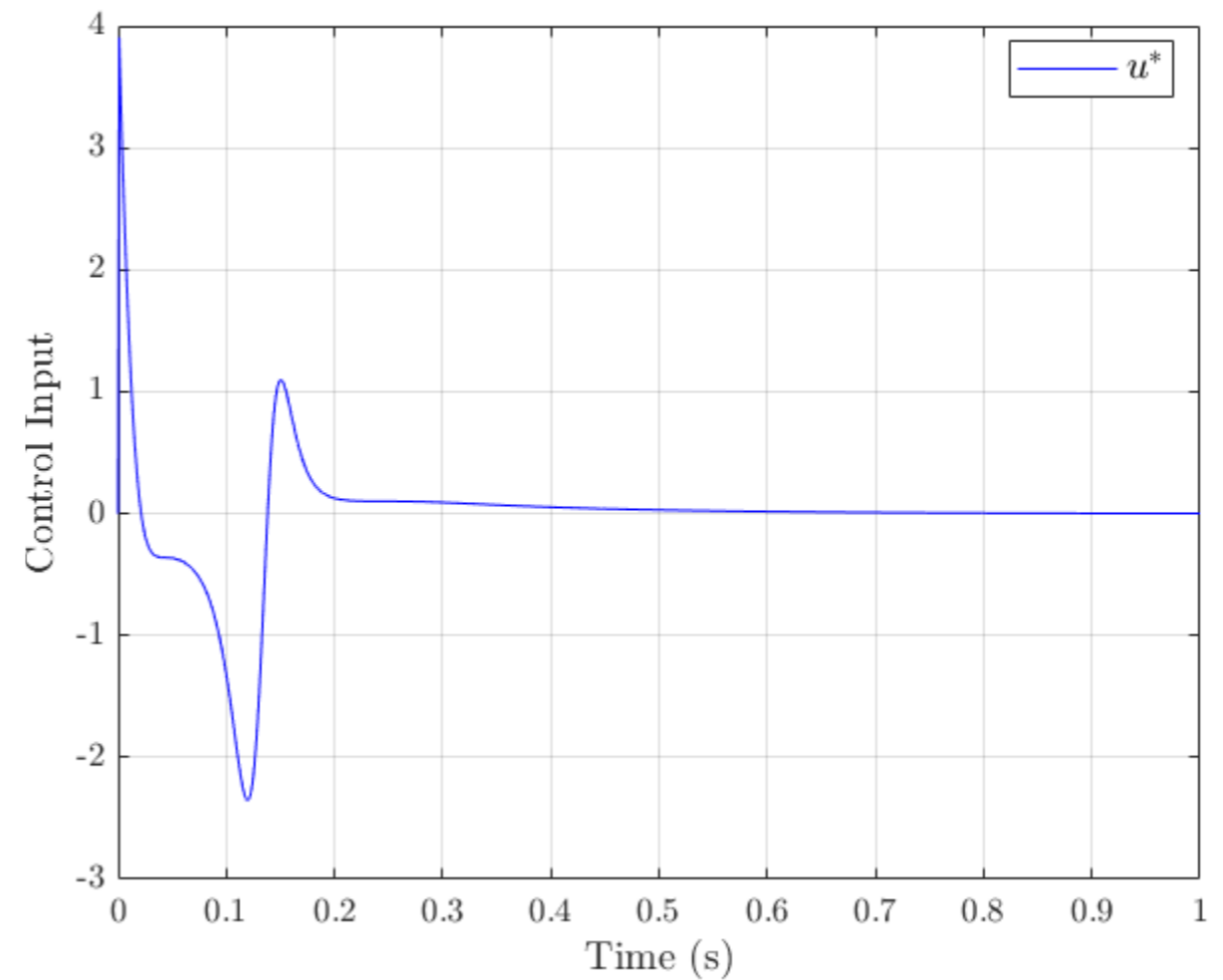
Linear Quadratic Regulator - Pendubot

- the control effort can be reduced increasing R

$$R = 0.1$$



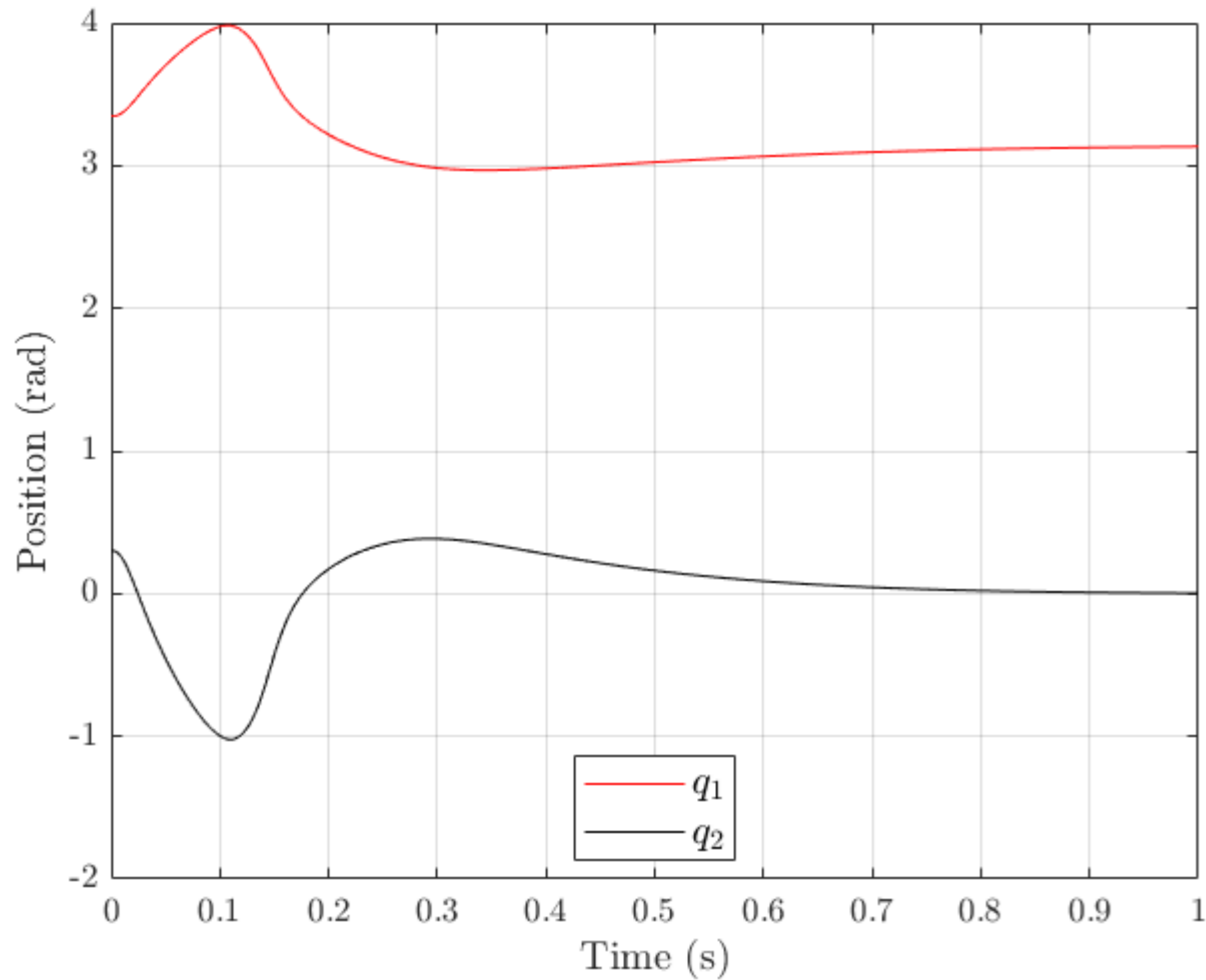
$$R = 2$$



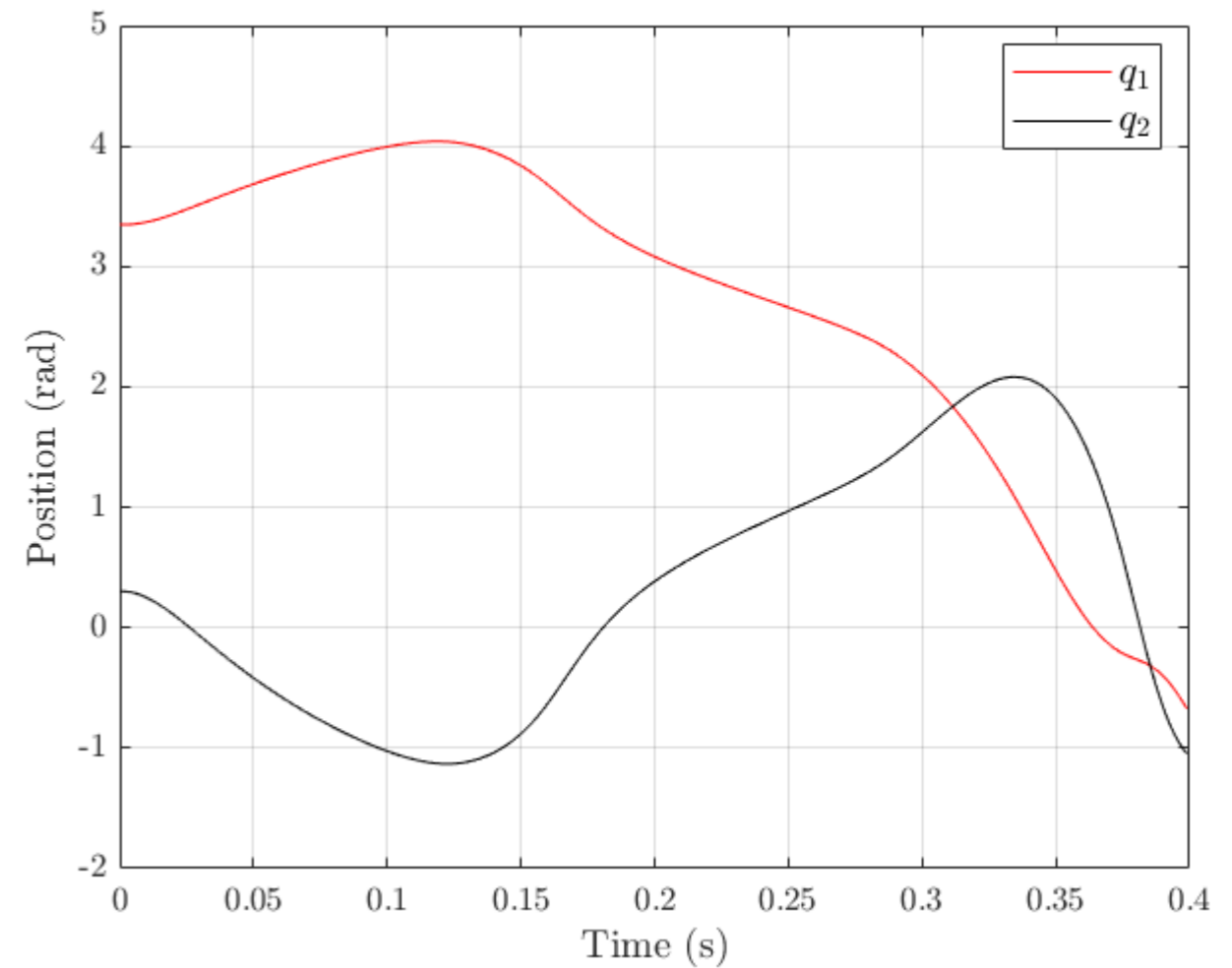
Linear Quadratic Regulator - Pendubot

- the **basin of attraction** (the validity of the LQR controller) changes w.r.t. the chosen weights Q, R

$R = 2$



$R = 5$



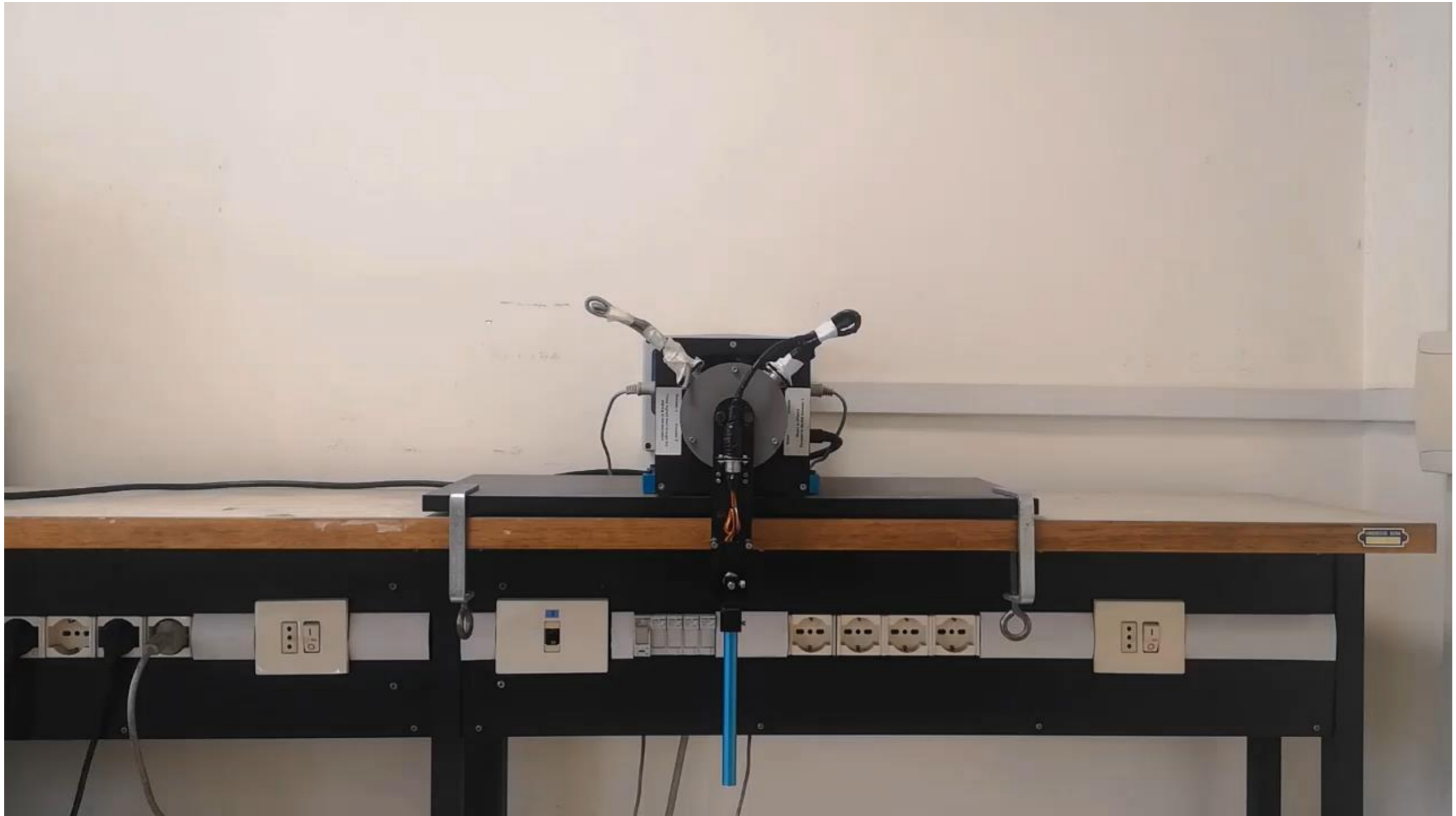
Linear Quadratic Regulator - Pendubot

- equilibrium up-up



Linear Quadratic Regulator - Pendubot

- equilibrium down-up



Linear Quadratic Regulator - Quadrotor

- hovering



summary

- some methods for finding a control law $u^*(x)$ that is optimal w.r.t to a cost function
- DP is only applicable on **small, finite** and **discrete** state and input spaces
- the HJB equation is the extension of the DP equation for state - input space and time
- the Riccati equation is a special solution of the HJB equation for linear system; the resulting LQR can be used as a control law for **nonlinear** systems, but it only works in the vicinity of the linearization point