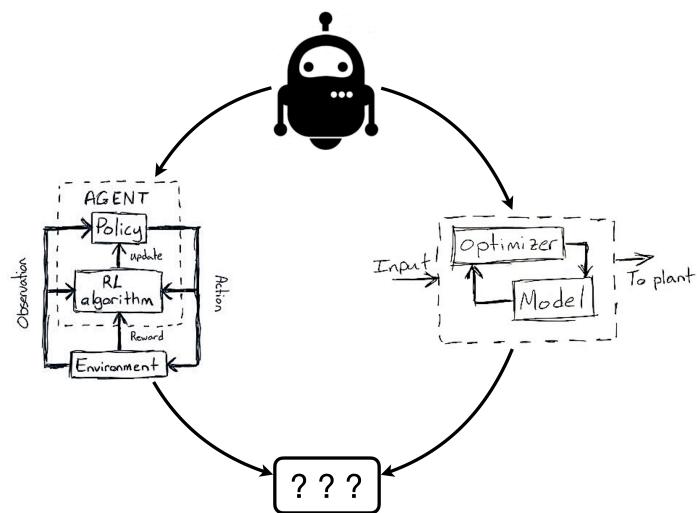


A reinforcement learning perspective on industrial model predictive control

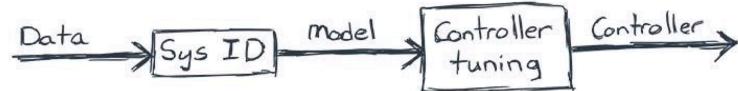
Upper Bound 2024

Nathan Lawrence

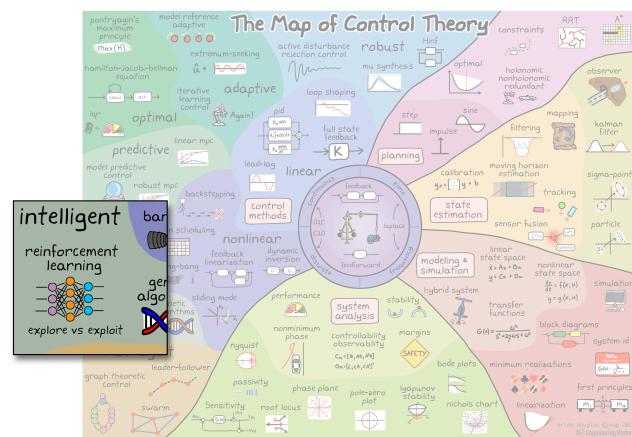


Day-to-day life depends on safely regulating a system around a constant value:

- 1. Cruise control
- 2. Temperature
- 3. Concentrations
- 4. Levels
- 5. Moisture
- 6. Etc...



3



4

Today

Combine reinforcement learning (RL) & model predictive control (MPC)!

- RL, MPC, and some stuff in-between pertaining to process control
- How to implement these ideas
- Emphasis on intuition rather than rigor
- Ask questions and be ready to discuss things with your neighbor



Cheat sheet!

I hope this works for you!

5

Reinforcement learning

7

Agents and environments

8

Environment

Apply torque



Observe

- Angle
- Angular velocity

9

Environment

s_t – state

a_t – action

Actions and states lead to new states

$$s_{t+1} \sim p(s_{t+1} | s_t, a_t)$$



10

Environment

Continuing forever gives a trajectory

$$s_0, a_0, s_1, a_1, \dots, s_t, a_t, s_{t+1}, \dots$$

Most trajectories are “bad”

We want a system that discovers “good” behavior in the environment



11

Agent

The agent is the learner:

- Decides which actions to take
- Improves its behavior

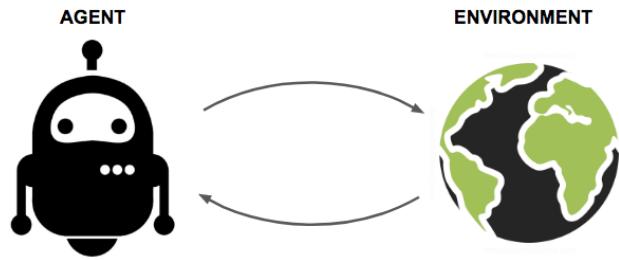


Figure: L. Weng (2018)

12

Agent

Reward guides learning:

- A scalar feedback signal
- Indicates which states & actions are good

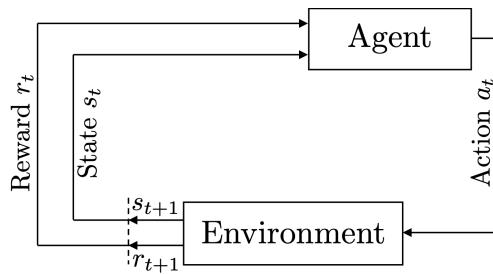


Figure: Sutton and Barto (2018)

13

What are we really trying to solve?

14

The agent-environment interface yields the trajectory

$$s_0, a_0, r_0, s_1, a_1, r_1 \dots, s_t, a_t, r_t, s_{t+1}, \dots$$

States governed by

$$s_{t+1} \sim p(s_{t+1} | s_t, a_t)$$

Agent chooses actions from a **policy**

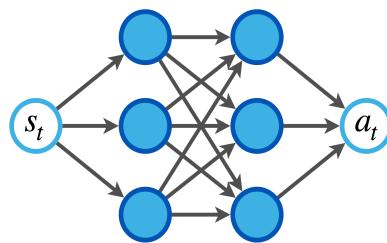
$$a_t \sim \pi(a_t | s_t)$$

Rewards assigned by a function

$$r_t = r(s_t, a_t)$$

The space of policies is vast

- Completely random
- An industrial control module
- Neural network



We want the “b

16

- Take $a \in \operatorname{argmax}_a \{ r(s, a) \}$?
→ too shortsighted
- Maximize $r_0 + r_1 + r_2 + \dots$?
→ might diverge

Consider the discounted **return**

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$$

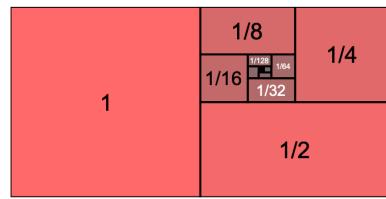
where $\gamma \in [0, 1]$

17

- Typically, $\gamma \in (0, 1)$
- Assume r is bounded by \bar{r}

Return is bounded across all possible trajectories

$$|G_t| \leq \frac{\bar{r}}{1 - \gamma}$$



[Geometric series \(Wikipedia\)](#)

18

The reinforcement learning problem

$$\text{maximize} \quad J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right]$$

19

The reinforcement learning problem

$$\text{maximize } J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right]$$

Why is this hard/impossible?

- Infinite horizon
- Search space
- Randomness
- No system description

20

Wishful thinking

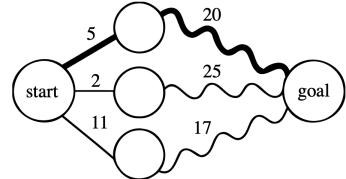
21

What if we had an optimal trajectory?

$$s_0^*, a_0^*, r_0^*, s_1^*, a_1^*, r_1^* \dots, s_t^*, a_t^*, r_t^*, s_{t+1}^*, \dots$$

Then the trajectory starting at s_t^* should still be optimal

$$s_t^*, a_t^*, r_t^*, s_{t+1}^*, \dots$$



[Optimal substructure \(Wikipedia\)](#)

22

Chess example

(placeholder – think about planning backwards from checkmate)

23

- Define the **state-action value** to be

$$Q(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 = s, a_0 = a \right]$$

- Similarly, the (state) **value** averages over the policy:

$$\begin{aligned} V(s) &= \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 = s \right] \\ &= \mathbb{E}_{a \sim \pi(a|s)} [Q(s, a)] \end{aligned}$$

24

Abstracting the objective through value functions

$$J(\pi) = \mathbb{E}_{s_0 \sim p(s_0)} [V(s_0)]$$

25

What's the big deal with these magical functions?

Observe

$$\begin{aligned} G_t &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \\ &= r_t + \gamma (r_{t+1} + \gamma r_{t+2} + \dots) \\ &= r_t + \gamma G_{t+1} \end{aligned}$$

Compresses an infinite number of actions into a simple scalar recursion!

26

What's the big deal with these magical functions?

Averaging...

$$\begin{aligned} V(s) &= \mathbb{E}_\pi [G_t | s_t = s] \\ &= \mathbb{E}_\pi [r_t + \gamma G_{t+1} | s_t = s] \\ &= \mathbb{E}_{a \sim \pi(a|s)} [r(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s,a)} [V(s')]] \end{aligned}$$

27

What's the big deal with these magical functions?

Similarly for Q ...

$$\begin{aligned} Q(s, a) &= r(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s,a)} [V(s')] \\ &= r(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s,a), a' \sim \pi(a'|s')} [Q(s', a')] \end{aligned}$$

28

What's the big deal with these magical functions?

These are the **Bellman equations** for V and Q

$$\begin{aligned} V(s) &= \mathbb{E} \left[r(s, a) + \gamma V(s') \right] \\ Q(s, a) &= r(s, a) + \gamma \mathbb{E} \left[Q(s', a') \right] \end{aligned}$$

Bellman equation holds

29

Bellman optimality equation

An optimal policy π^* solves the following:

$$V^*(s) = \max_{\pi} V(s) \quad Q^*(s, a) = \max_{\pi} Q(s, a)$$

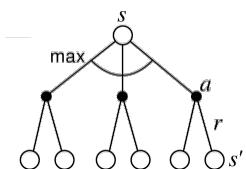
We don't actually have π^* , but it's fun to imagine...

30

Bellman optimality equation

Given V^* ...

$$\begin{aligned} V^*(s) &= \mathbb{E}_{a \sim \pi^*(a|s)} [r(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s,a)} [V^*(s')]] \\ &= \max_a \{ r(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s,a)} [V^*(s')] \} \end{aligned}$$



One-step planning is lc

Bellman optimality equation

Given Q^* ...

$$V^*(s) = \max_a Q^*(s, a)$$

...Even easier!

Learn Q^* and then ma

Evaluate, improve, repeat...

33

Like V^* , Q^* satisfies a neat optimality condition:

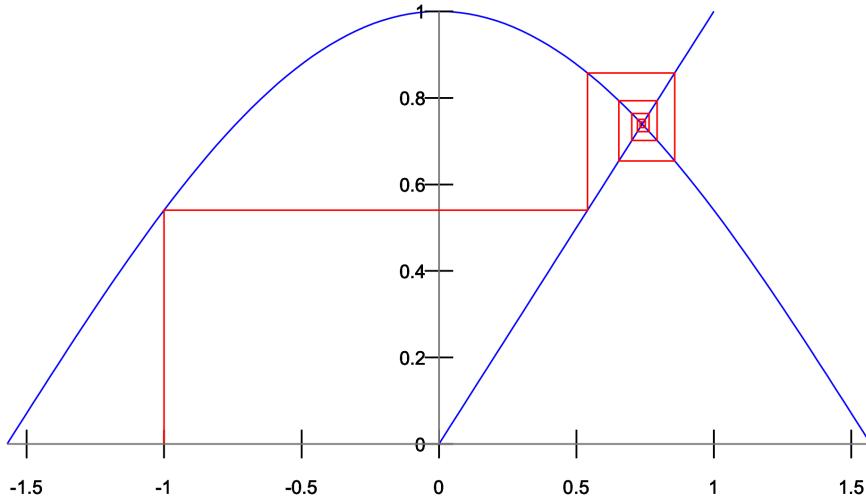
$$\begin{aligned} Q^*(s, a) &= r(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s,a), a' \sim \pi^*(a'|s')} [Q^*(s', a')] \\ &= r(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s,a)} \left[\max_{a'} Q^*(s', a') \right] \end{aligned}$$

“Plug this magical function Q^* into the RHS produces the same function”

34

Fixed-point iteration

Iterating $q^{k+1} = B(q^k)$ may converge to some q where $q = B(q)$



35

- Let's just take the operator

$$B(Q) = r(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s, a)} \left[\max_{a'} Q(s', a') \right]$$

and apply fixed-point iteration!

- The RHS is an **idealized** update scheme and key source of inspiration

36

Fixed-point aspirations

If we have a policy π and some oracle that tells us Q , take

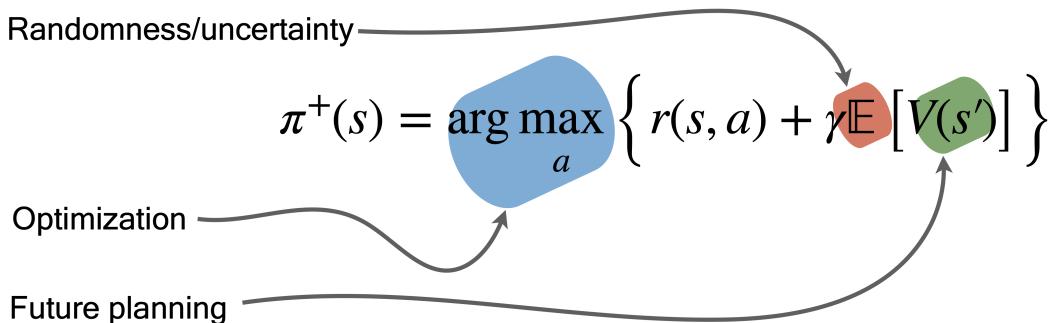
$$\begin{aligned}\pi^+(s) &= \arg \max_a Q(s, a) \\ &= \arg \max_a \left\{ r(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s,a)} [V(s')] \right\}\end{aligned}$$

Then π^+ is at least as good as π ! That is, $Q^+ = B(Q) \geq Q$.

There are many ways to proceed: Policy iteration, value iteration, and we haven't even mentioned policy evaluation. We're focusing on high-level ideas and avoiding introducing jargon, so see Sutton and Barto (2018) for a full treatment.

37

Three important approximations



To move things along we will work through some examples instead of focusing on the minutiae of common RL algorithms

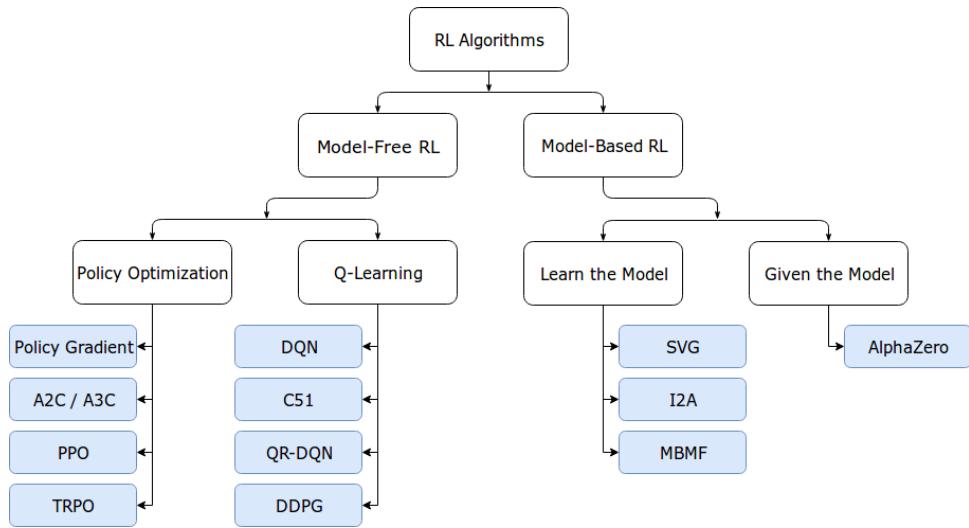


Figure: Spinning Up

39

Examples

41

Learning to balance

42

Cartpole



43

Deep Q-networks (DQNs)

For a **finite** set of actions, define the neural network

$$\text{DQN}(s) = \begin{bmatrix} q_1 \\ \vdots \\ q_d \end{bmatrix}$$

where each q_i is an approximation of $Q(s, a_i)$

Optimization is trivial:

$$\begin{aligned}\pi(s) &= \arg \max \text{DQN}(s) \\ &= \arg \max \{q_1, \dots, q_d\} \\ &\approx \arg \max_a Q(s, a_i)\end{aligned}$$

Playing Atari with Deep Reinforcement Learning

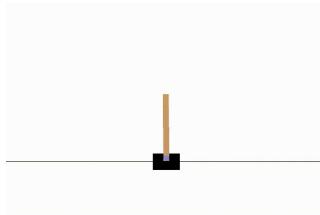
Volodymyr Mnih Koray Kavukcuoglu David Silver Alex Graves Ioannis Antonoglou
Daan Wierstra Martin Riedmiller

Algorithm 1 Deep Q-learning with Experience Replay

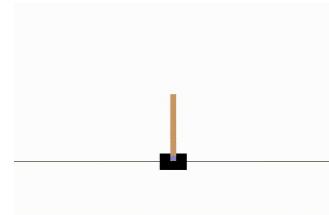
```
Initialize replay memory  $\mathcal{D}$  to capacity  $N$ 
Initialize action-value function  $Q$  with random weights
for episode = 1,  $M$  do
    Initialise sequence  $s_1 = \{x_1\}$  and preprocessed sequenced  $\phi_1 = \phi(s_1)$ 
    for  $t = 1, T$  do
        With probability  $\epsilon$  select a random action  $a_t$ 
        otherwise select  $a_t = \max_a Q^*(\phi(s_t), a; \theta)$ 
        Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$ 
        Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$ 
        Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $\mathcal{D}$ 
        Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $\mathcal{D}$ 
        Set  $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$ 
        Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$  according to equation 3
    end for
end for
```

Equation 3: Approximately respect the Bellman equation

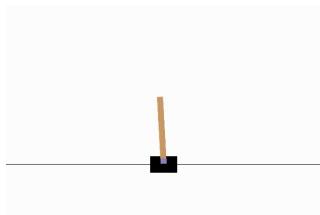
$$[q_1, \dots, q_d] \approx r(s, a) + \gamma \max \{q'_1, \dots, q'_d\}$$



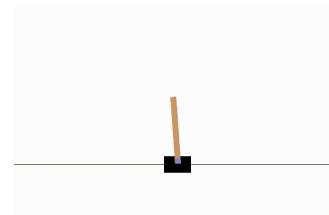
Initial



After some time...



Almost there...



Final

46

Acrobot



Touch the line!

47

Acrobot



Touch the line!
Stay above the line!

48

Applying DQN...



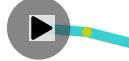
Initial



After some time...



Almost there...



Final

Standing upright would be better than all of these...

49

Finer control requires continuous actions

What we want:

$$\pi^*(s) = \arg \max_a Q^*(s, a)$$

What DQN delivers:

$$\begin{aligned}\pi(s) &= \arg \max \{q_1, \dots, q_d\} \\ &\approx \arg \max \{Q^*(s, a_1), \dots, Q^*(s, a_d)\}\end{aligned}$$

50

Finer control requires continuous actions

Consider two separate networks:

- Policy π_θ (aka “actor”)
- Q -network Q_φ (aka “critic”)

Idea: Use Q_φ as a loss function for π_θ :

$$\text{maximize}_\theta \quad \mathbb{E} [Q_\varphi(s, \pi_\theta(s))]$$

51

Then

$$\max_a Q(s, a) \approx Q(s, \pi_\theta(s))$$

An easy-to-evaluate approximation

52

Disclaimer

This doesn't actually work.

See the DDPG, TD3, SAC papers for all the tricks/hacks that make this idea work.

- Replay buffers
- Target networks
- Exploration / exploitation
- Double Q -learning
- Delayed updates
- Smoothing

Acrobot cont'd

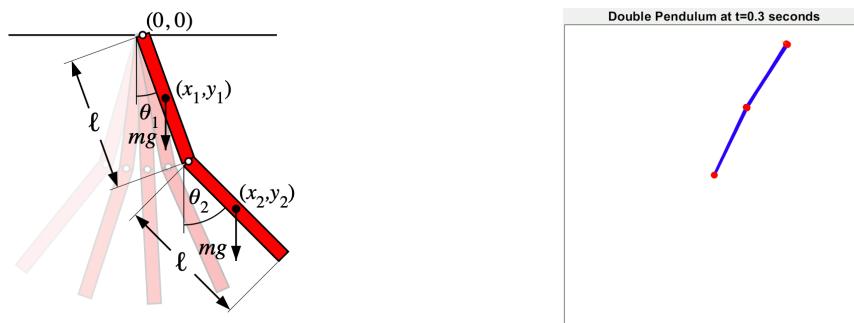
Let's assume we have some RL oracle: Given an environment and sufficiently powerful networks, it does a reasonable job at solving the principal RL problem

We isolate two components:

- Reward function
- Discount factor γ

54

Todo: Try to formulate a reward function



[Double pendulum \[Wikipedia\]](#)

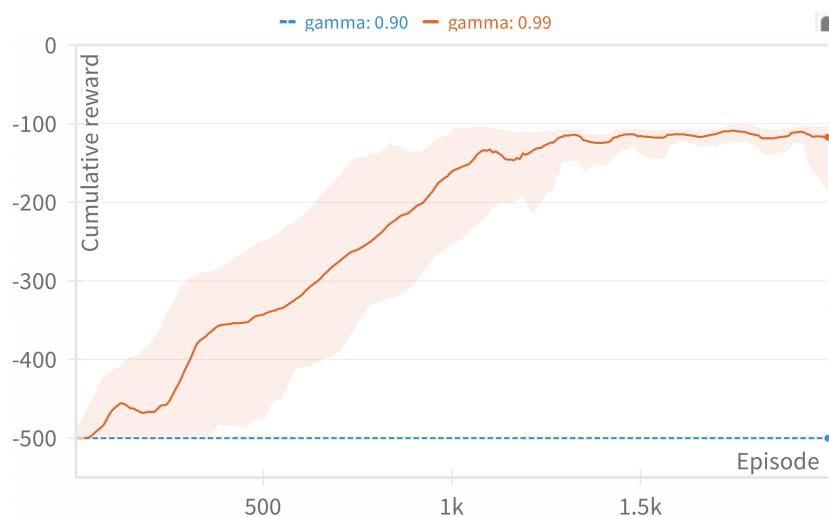
55

Reward functions

- Default: 0 if above line; -1 otherwise
- ℓ^2 : Negative 2-norm of
 - normalized velocities
 - $-1 - \cos(q_1), 1 - \cos(q_2)$
- Height
- ℓ^∞ : Negative ∞ -norm of
 - Normalized velocities
 - Normalized height

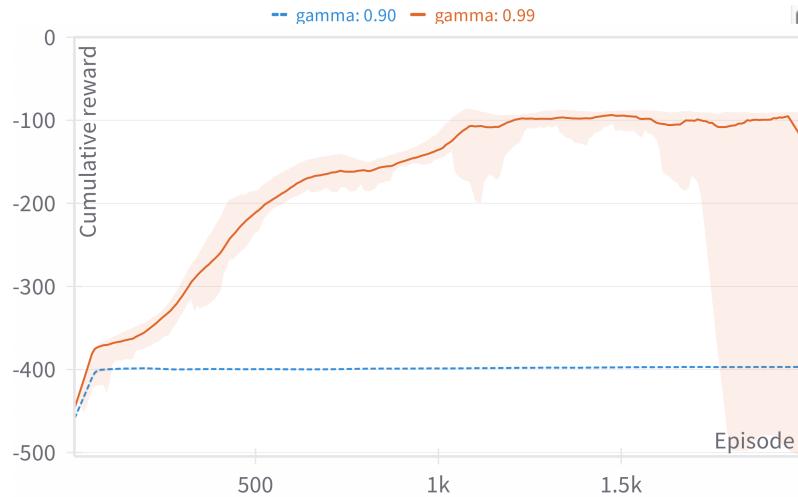
56

Default reward



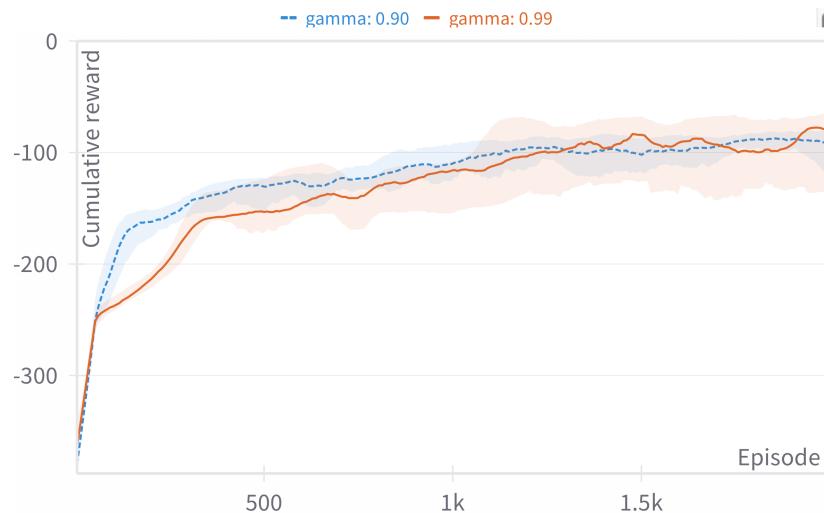
57

ℓ^2 reward



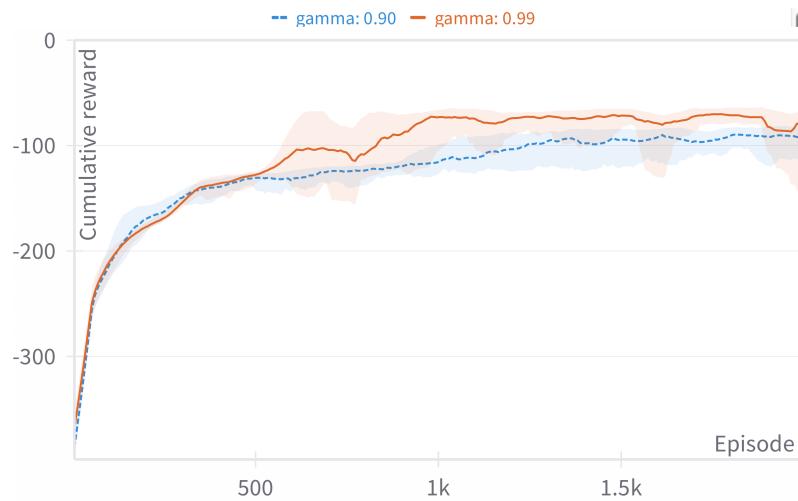
58

“Height” reward

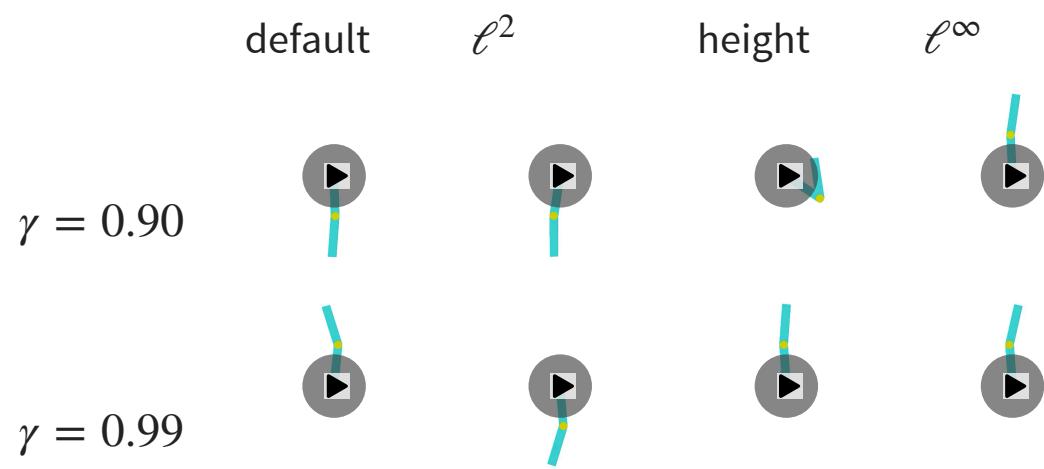


59

ℓ^∞ reward



60



1 of 4^8 configurations. Sampled via [random.org](#).

61

Bloopers

I modified the [default environment](#): default reward, spaces, sampling time

Action space:

- Old: $\{-1, 0, 1\}$ (discrete)
- Intermediate: $[-1, 1]$ (continuous, restricted)
- New: $[-2, 2]$ (continuous, expanded)

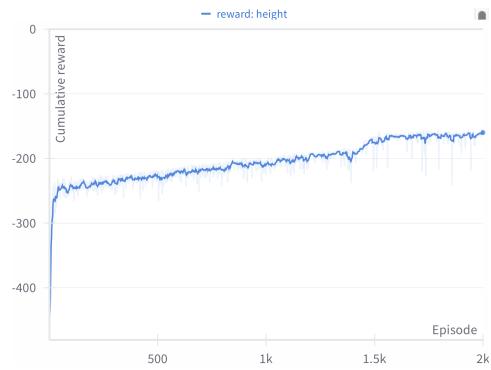
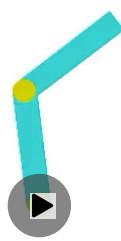
Sampling time:

- Old: $dt = 0.2$ seconds
- New: $dt = 0.1$ seconds

62

Restricted action space

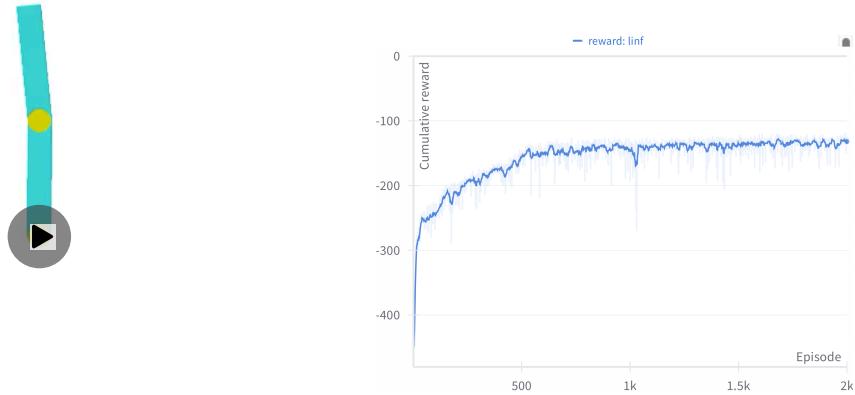
“Height” reward, $\gamma = 0.99$, 5hz



63

Restricted action space cont'd

ℓ^∞ reward, $\gamma = 0.95$, 5hz



64

An analytic solution

65

What if we know something about the environment?

Let's grossly simplify the problem:

$$\text{maximize } \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right]$$

- Unknown dynamics
- Unstructured policy
- Possibly unknown reward
- Everything is random

66

Aside: Some notation

(placeholder – some discussion about RL and control notation should go somewhere...)

67

What if we know something about the environment?

Let's grossly simplify the problem:

$$\begin{aligned} \text{minimize} \quad & \sum_{t=0}^{\infty} \gamma^t (x_t^2 + \rho u_t^2) \\ \text{where} \quad & x_{t+1} = ax_t + bu_t \\ & u_t = -kx_t \end{aligned}$$

- Linear, scalar dynamics
- Linear policy
- Quadratic cost
- Deterministic

68

Quadratic value function

Trajectories are easy to compute:

$$\begin{aligned} x_{t+1} = ax_t + bu_t &\implies x_{t+1} = (a - bk)x_t \\ u_t = -kx_t &\qquad\qquad\qquad \vdots \\ &x_{t+1} = (a - bk)^{t+1}x_0 \\ &u_t = -k(a - bk)^t x_0 \end{aligned}$$

69

Quadratic value function

Returns are easy to compute:

$$\begin{aligned} \sum_{t=0}^{\infty} \gamma^t (x_t^2 + \rho u_t^2) &= x_0^2 + \gamma(a - bk)^2 x_0^2 + \gamma^2(a - bk)^4 x_0^2 + \dots \\ &\quad + \rho k^2 x_0^2 + \gamma \rho k^2 (a - bk)^2 x_0^2 + \dots \\ &= \sum_{t=0}^{\infty} \gamma^t (a - bk)^{2t} (1 + \rho k^2) x_0^2 \end{aligned}$$

$$\begin{aligned} x_{t+1} &= (a - bk)^{t+1} x_0 \\ u_t &= -k(a - bk)^t x_0 \end{aligned}$$

70

Quadratic value function

By properties of geometric series:

$$\begin{aligned} \text{return} &= \sum_{t=0}^{\infty} \gamma^t (a - bk)^{2t} (1 + \rho k^2) x_0^2 \\ &= \frac{1 + \rho k^2}{1 - \gamma(a - bk)^2} x_0^2 \end{aligned}$$

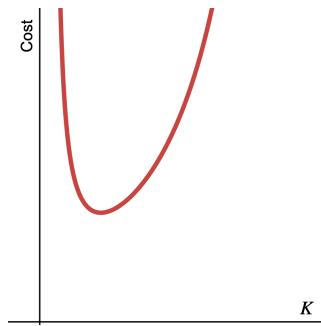
$$\sum_{i=0}^{\infty} \alpha \beta^i = \alpha \frac{1}{1-\beta}$$

71

Quadratic value function

A 1-D optimization problem in K :

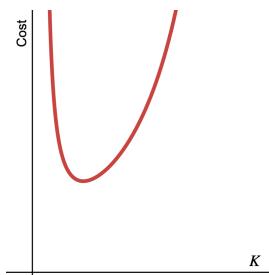
$$\min_K \frac{1 + \rho K^2}{1 - \gamma(a - bK)^2}$$



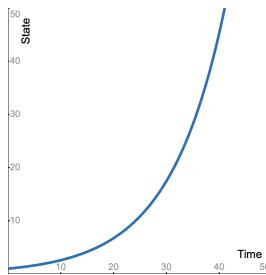
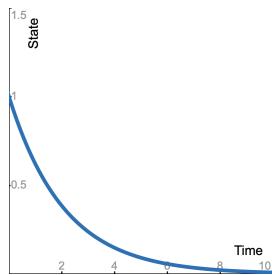
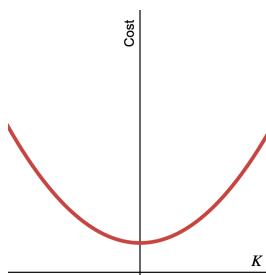
[To graph]

72

$$\gamma = 1$$



$$\gamma = 0$$



73

Wait, are linear controllers optimal?

YES!

$$\min_{u_0, u_1, \dots} \sum_{t=0}^{\infty} \gamma^t (x_t^2 + \rho u_t^2)$$

where $x_{t+1} = ax_t + bu_t$

~~$u_t = -kx_t$~~

74

Define

$$V^*(x) = \min_{\substack{u_0, u_1, \dots \\ x_{t+1} = ax_t + bu_t \\ x_0 = x}} \sum_{t=0}^{\infty} \gamma^t (x_t^2 + \rho u_t^2)$$

\Downarrow (fact)

$$V^*(x) = Px^2$$

The optimal value function is:

1. Quadratic
2. Parameterized by some $P > 0$

Bellman gives us a single variable problem:

$$Px^2 = \min_u \{x^2 + \rho u^2 + \gamma P(ax + bu)^2\}$$

$$\Downarrow \text{solve } 0 = \nabla_u \{ \dots \}$$

$$u = -\frac{\gamma abP}{\rho + \gamma b^2 P}x$$

\Downarrow

Linear controller...but what is P ?

Plug u back into the Bellman equation to find:

$$P = 1 + \gamma a^2 P - \frac{\gamma^2 (abP)^2}{\rho + \gamma b^2 P}$$

A fixed point in P !

Finding this fixed point in **parameter space** gives the optimal solution in value **function space**

Recap lessons from RL section

(placeholder + 5 min break slide)

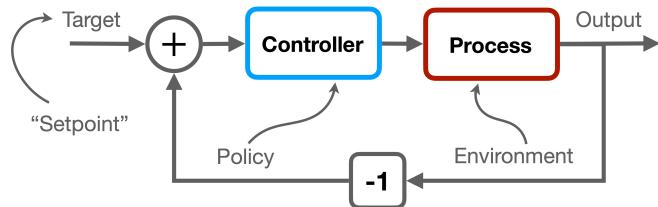
79

Process control

81

A common control task is to bring a system to a constant value:

- 1. Cruise control
- 2. Temperature
- 3. Concentrations
- 4. Levels
- 5. Moisture
- 6. Etc...



82

Linear policies

83

PID control

“Based on a survey of over eleven thousand controllers in the refining, chemicals and pulp and paper industries, 97% of regulatory controllers utilize a PID feedback control algorithm.”

– Desborough and Miller (2002), also Åström and Murray (2021)

85

The system settles at zero...how do we get it to settle somewhere else?

86

Proportional control

y_{sp} = desired value (sp = “setpoint”)

y = measured value

$$e = y_{sp} - y$$

We want $e \rightarrow 0$ as $t \rightarrow \infty$

87

Proportional control

Consider the policy

$$u = k_p e$$



88

Proportional-Integral control

Consider the policy

$$u(t) = k_p e(t) + k_i \int_0^t e(\tau) d\tau$$



89

The magic of integral action

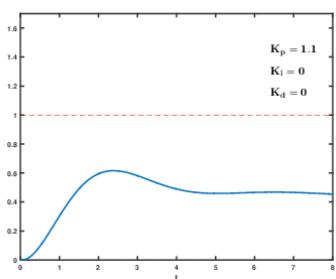
1. Assume that k_p, k_i are chosen such that the system is stable
2. Then $u(t) \rightarrow \bar{u}, e(t) \rightarrow \bar{e}$
3. We can write

$$\bar{u} = k_p \bar{e} + k_i \lim_{t \rightarrow \infty} \int_0^t e(\tau) d\tau$$

4. The integral term must converge
5. Zero offset!

90

Proportional-Integral-Derivative control



Proportional: Go towards the setpoint

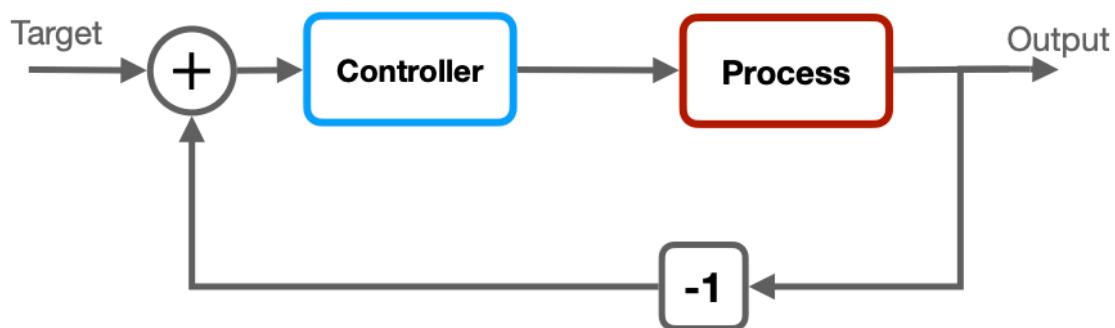
Integral: Stay at the setpoint

Derivative: Don't overshoot the setpoint

PID controller (Wikipedia)

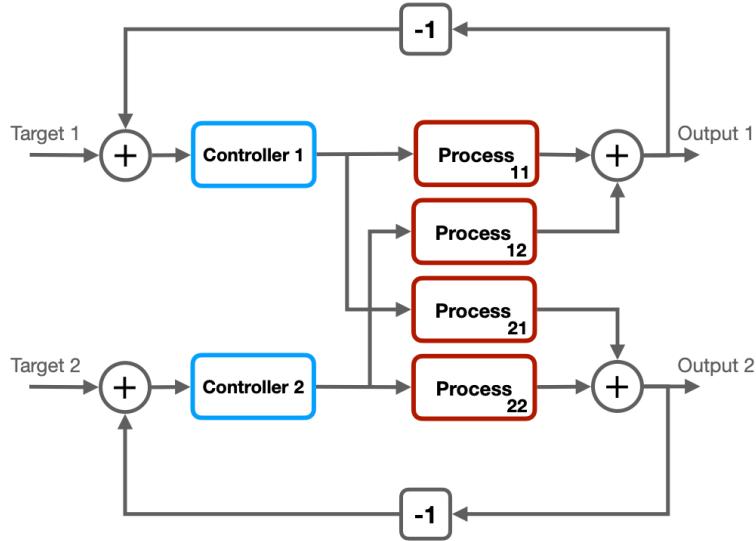
91

PID controllers are great for tasks with a single input and single output.



92

But often times we want to control a system with many input and output variables.



93

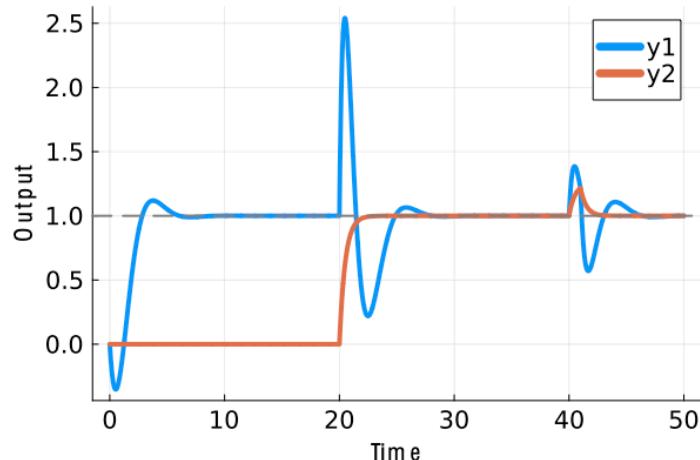
Can we tune controllers independently?

Take two **independent** processes and design individual controllers

94

Can we tune controllers independently?

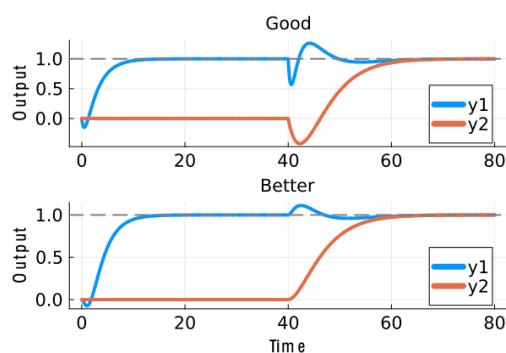
Those controllers can be disastrous if one process feeds into the other



95

Can we tune controllers independently?

... sort of



Design of Decoupled PID Controllers
for MIMO Systems

Karl Johan Åström¹, Karl Henrik Johansson², and Qing-Guo Wang³

We need a scalable approach to handling loop interactions!

96

LQR

98

$$\min_{u_0, u_1, \dots} \sum_{t=0}^{\infty} \gamma^t (x_t^T M x_t + u_t^T R u_t)$$

where $x_{t+1} = Ax_t + Bu_t$

- **Linear:** Dynamics
- **Quadratic:** Cost (and value)
- **Regulator:** Keep state x_t around 0

We already solved this in the scalar case!

$$M \geq 0, R >, \gamma \in [0, 1]$$

99

General solution

1. There is $P > 0$ such that $V^*(x) = x^T Px$. Then via the **Bellman equation**

$$x_t^T Px_t = \min_u \left\{ x_t^T M x_t + u_t^T R u_t + \gamma (Ax + Bu)^T P (Ax + Bu) \right\}$$

100

General solution

1. Apply the **Bellman equation** with $V^*(x) = x^T Px$

101

General solution

1. Apply the **Bellman equation** with $V^*(x) = x^T Px$
2. Enforce

$$0 = \nabla_u \{ \dots \}$$

102

General solution

1. Apply the **Bellman equation** with $V^*(x) = x^T Px$
2. Enforce $0 = \nabla_u \{ \dots \}$
3. Obtain $u = -Kx$ where

$$K = \gamma(R + \gamma B^T P B)^{-1} B^T P A$$

103

General solution

1. Apply the **Bellman equation** with $V^*(x) = x^T Px$
2. Enforce $0 = \nabla_u \{ \dots \}$
3. Obtain $u = -\gamma(R + \gamma B^T PB)^{-1} B^T PAx$
4. P satisfies the **Discrete Algebraic Riccati Equation**

$$P = M + \gamma A^T PA - \gamma^2 A^T PB(R + \gamma B^T PB)^{-1} B^T PA$$

104

- LQR is a tidy and globally optimal solution for controlling multivariable systems!
- It comes standard in any control systems library

```
1 using ControlSystems
2
3 Pd = c2d(ss(P),ts)
4 A, B = Pd.A, Pd.B
5 M, R = I, I
6
7 K = lqr(Discrete, A, B, M, R)
8
9 u(x,t) = -K*x
```

Example is in the Julia package [ControlSystems.jl](#). Other options include Matlab's [Control System Toolbox](#), or [Python Control Systems Library](#).

105

Aside about discounting

Standard LQR solvers: $(A, B, M, R) \rightarrow K$

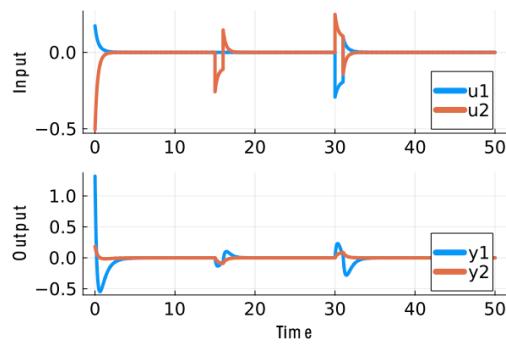
Discounted LQR: Use

$$\left(\sqrt{\gamma}A, B, M, \frac{1}{\gamma}R \right)$$

As $\gamma \rightarrow 0$...

- “Ignore the state transition matrix”
- “Apply infinite weight to control actions”
- ...Unstable controller

106



- The controller quickly brings the system to 0
- A random disturbance in u_2 occurs at $t = 15$, affecting y_1, y_2
- The controller brings y_1 and y_2 back to equilibrium

107

Globally optimal?

All systems are subject to constraints:

- Finite resources & money
- Limited actuation

LQR assumes:

- Any control action is permissible
- Any intermediate state is acceptable

108

Globally optimal?

Consider the system

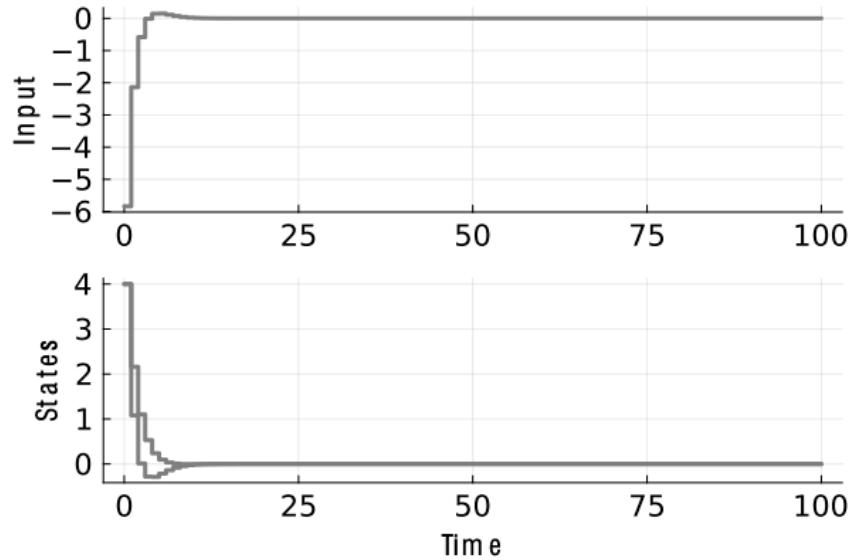
$$x_{t+1} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} x_t + \begin{bmatrix} 1 \\ 0.5 \end{bmatrix} u_t$$

Unstable → control is needed to achieve equilibrium

109

Globally optimal?

LQR, business as usual

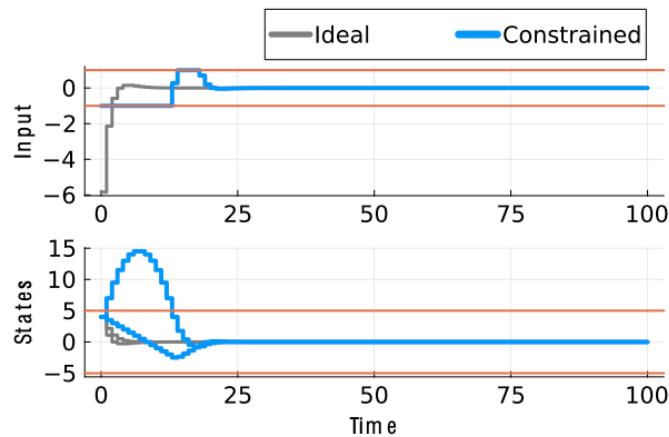


110

Globally optimal?

Suppose only actions in the set $\{u: \|u\|_\infty \leq 1\}$ are possible

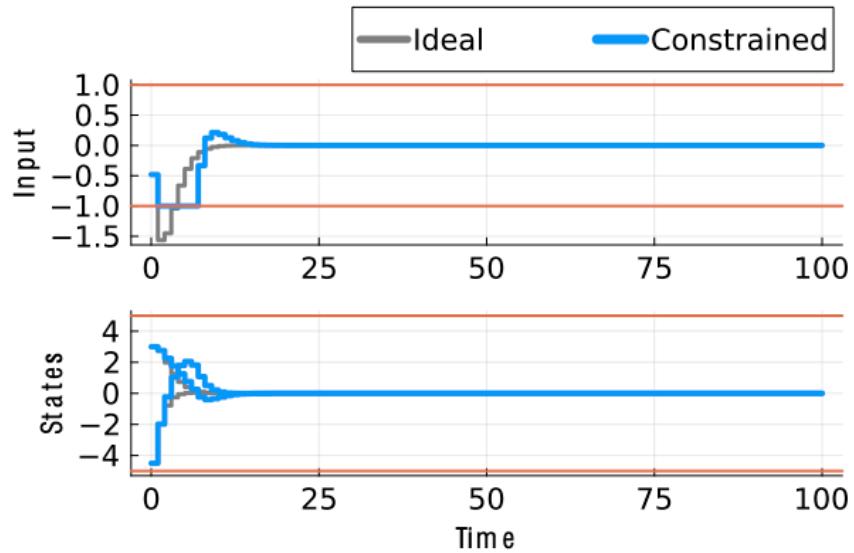
And we want the states to stay in $\{x: \|x\|_\infty \leq 5\}$



111

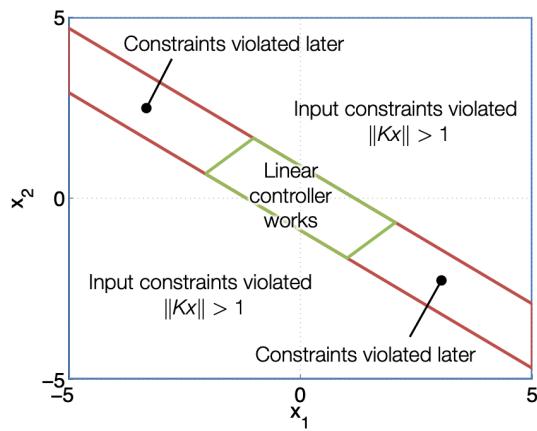
Globally optimal?

The actions can start out feasible, then become infeasible later



112

Locally optimal

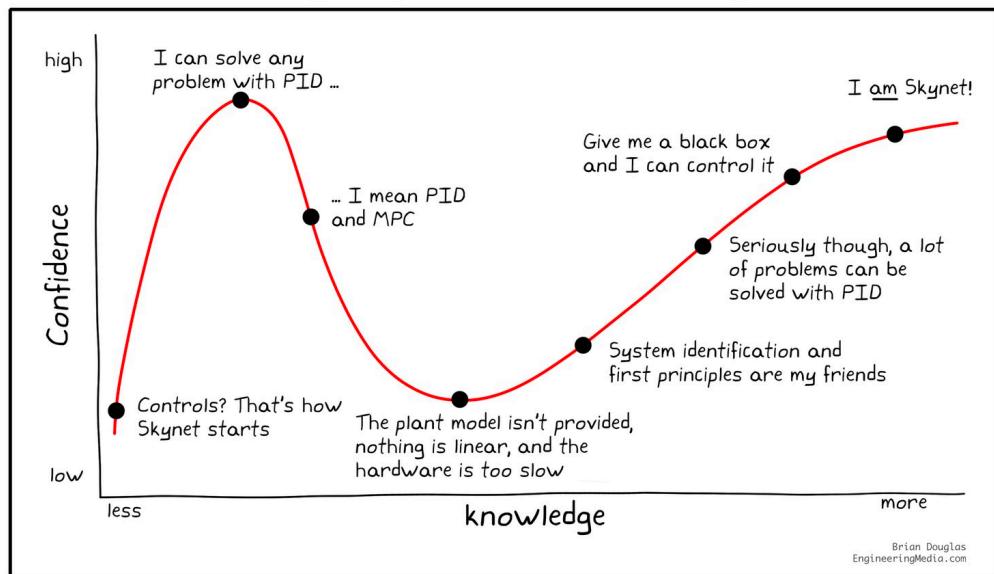


See Borrelli, Bemporad, and Morari (2017) or [slides](#)

113

Break

Dunning-Kruger effect for control engineers



114

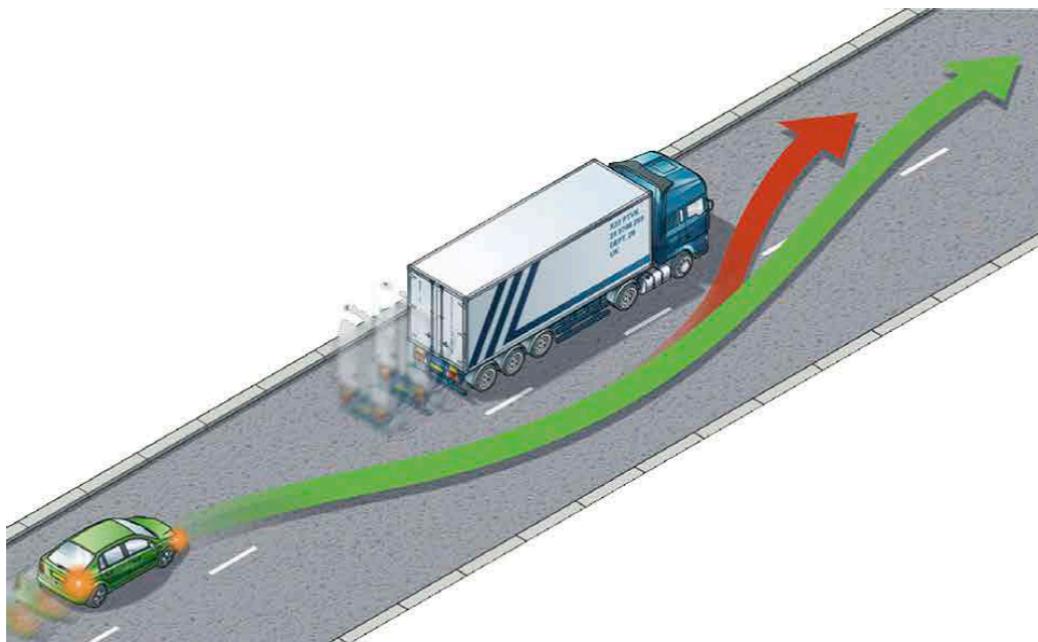
Nonlinear policies

116

What if a controller could anticipate an obstacle?

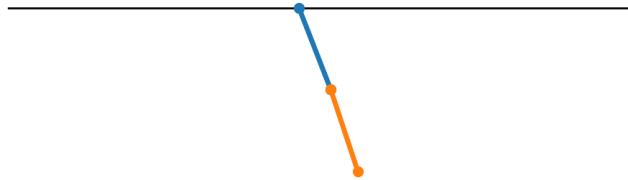
117

What if a controller could anticipate an obstacle?



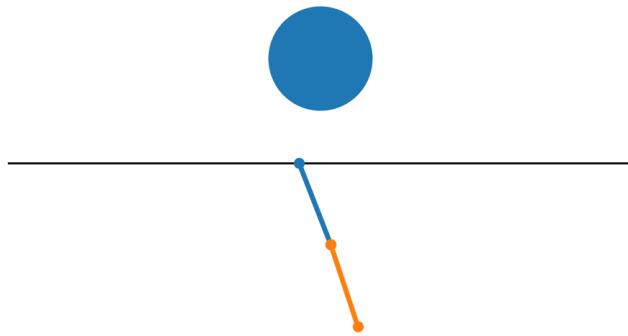
118

From this...



119

To this...



120

Anticipating constraints is inherently nonlinear

Suppose x_t is “close” to upper constraint c :

- Take conservative actions near constraint
- More freedom away from constraints

LQR:

- Follow $u_t = -Kx_t$ no matter what

121

Model predictive control

123

Problem formulation

MPC is a common-sense strategy of making decisions by predicting the future

$$\min_{u_0, u_1, \dots, u_{N-1}} \sum_{t=0}^{N-1} x_t^T M x_t + u_t^T R u_t$$

$$\text{where } x_{t+1} = Ax_t + Bu_t$$

$$x_t \in \mathcal{X}$$

$$u_t \in \mathcal{U}$$

\mathcal{X}, \mathcal{U} are often box constraints: $u_{\min} \leq u_t \leq u_{\max} \forall t$. Like LQR, $M \geq 0, R > 0$.

¹²⁴

Problem formulation

Applying the optimal inputs $u_0^\star, u_1^\star, \dots$ is an **open-loop** strategy

- Model errors compound
- Unexpected disturbances will go unchecked
- Will need to solve the MPC problem after N time steps anyway

¹²⁵

The receding horizon idea

At each time step, re-initialize the MPC problem with the state s_t from the environment:

$$\min_{u_0, u_1, \dots, u_{N-1}} \sum_{k=0}^{N-1} x_k^T M x_k + u^T R u_k$$

where $x_0 = s_t$

$$x_{k+1} = Ax_k + Bu_k$$

$$x_k \in \mathcal{X}$$

$$u_k \in \mathcal{U}$$

126

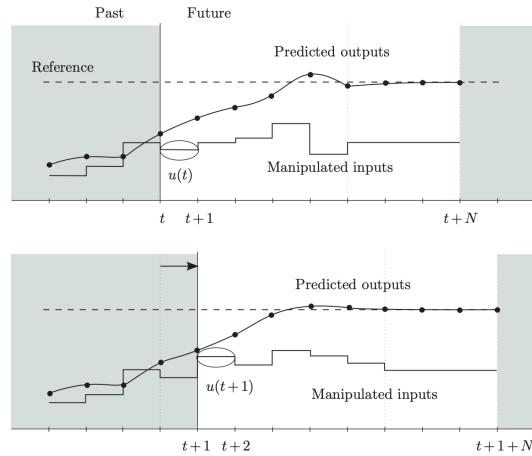
The receding horizon idea

MPC controller

1. Initialize state $x_0 = s$
2. Solve the MPC optimization problem
3. Apply u_0^\star to the system
4. Update system state $s \leftarrow s'$

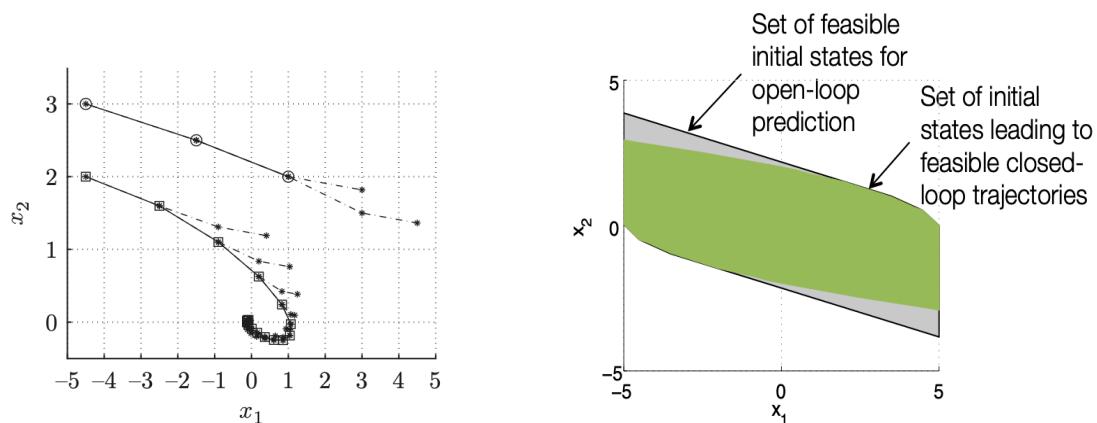
127

The receding horizon idea



See Borrelli, Bemporad, and Morari (2017), Chapter 12

128

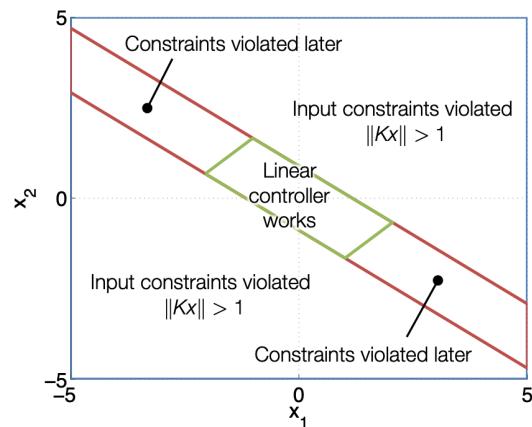


Solid - realized closed-loop trajectories
Dash - predicted trajectory

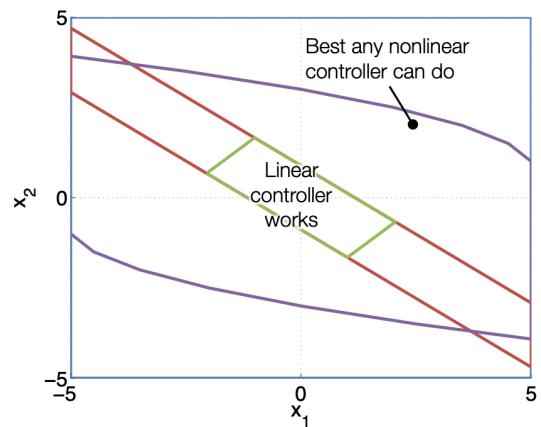
See Borrelli, Bemporad, and Morari (2017), Chapter 12

129

LQR

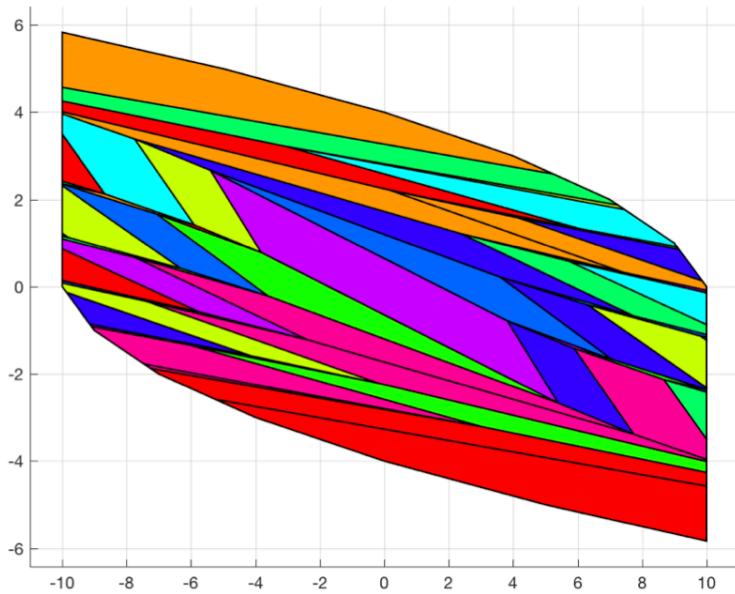


MPC



130

MPC control law is nonlinear



131

Why?

MPC = Multi-parametric quadratic programming

132

Original problem:

$$\min_{u_0, u_1, \dots, u_{N-1}} \sum_{k=0}^{N-1} x_k^T M x_k + u^T R u_k$$

$$\text{where } x_0 = s_t$$

$$x_{k+1} = Ax_k + Bu_k$$

$$x_k \in \mathcal{X}$$

$$u_k \in \mathcal{U}$$

133

Unroll the dynamics:

$$x_{k+1} = Ax_k + Bu_k$$

134

Unroll the dynamics:

$$x_{k+1} = Ax_k + Bu_k \text{ and } x_0 = 0 \text{ for ease}$$

135

Algebra:

$$\min_U \quad \frac{1}{2} U^T H U + s_t^T F U$$

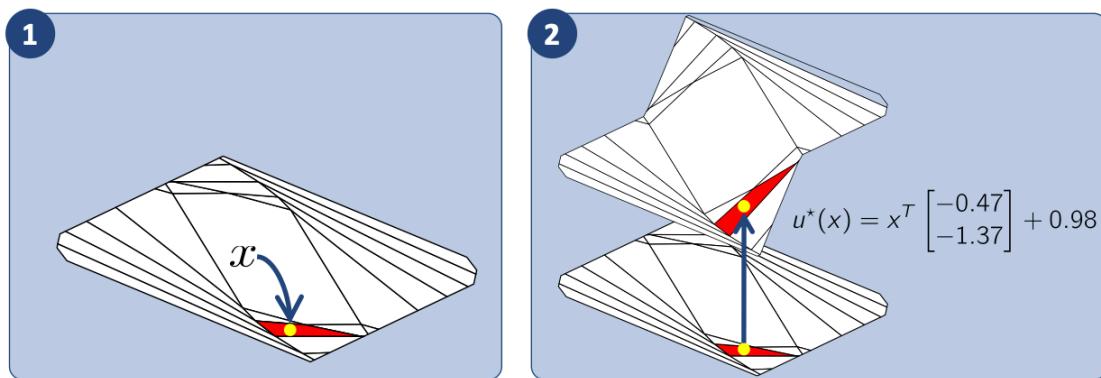
where $GU \leq W + Es_t$

See Bemporad et al. (2002)

136

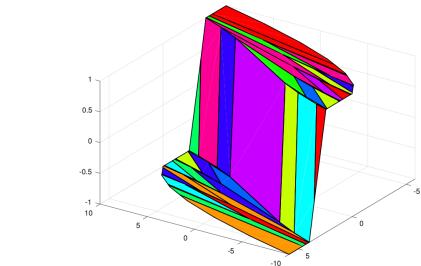
KKT conditions:

U^\star and associated Lagrange multipliers are affine in s_t

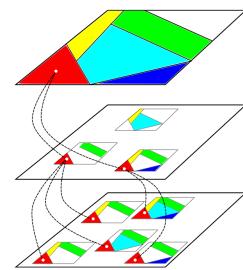


137

MPC = Continuous piecewise affine



MPC control law



ReLU DNN

See Bemporad et al. (2002), Karg and Lucia (2020), Montufar et al. (2014)

138

Stability issues

“In the engineering literature it is often assumed (tacitly and incorrectly) that a system with optimal control law is necessarily stable.”

– Kalman (1960)

139

Repeatedly implementing a finite horizon solution on an infinite horizon problem leads to “surprises”

- Infeasibility
- Instability

140

How can we solve an infinite horizon optimal control problem with finite resources?

1. Terminal constraint
2. Terminal cost

$$\min_{u_0, u_1, \dots, u_{N-1}} \sum_{k=0}^{N-1} (x_k^T M x_k + u_k^T R u_k) + x_N^T P x_N$$

where $x_0 = s_t$

$$x_{k+1} = Ax_k + Bu_k$$

$$x_k \in \mathcal{X}, u_k \in \mathcal{U}$$

$$x_N \in \mathcal{X}_f$$

How to design terminal cost?

LQR!

1. Obtain P by solving $\text{lqr}(A, B, M, R)$
2. Embed a fictitious LQR controller $u_t = -Kx_t$ into MPC after N_c time steps

Final objective

$$\min_{u_0, u_1, \dots, u_{N_c-1}} \sum_{k=0}^{N-1} (x_k^T M x_k + u_k^T R u_k) + x_N^T P x_N$$

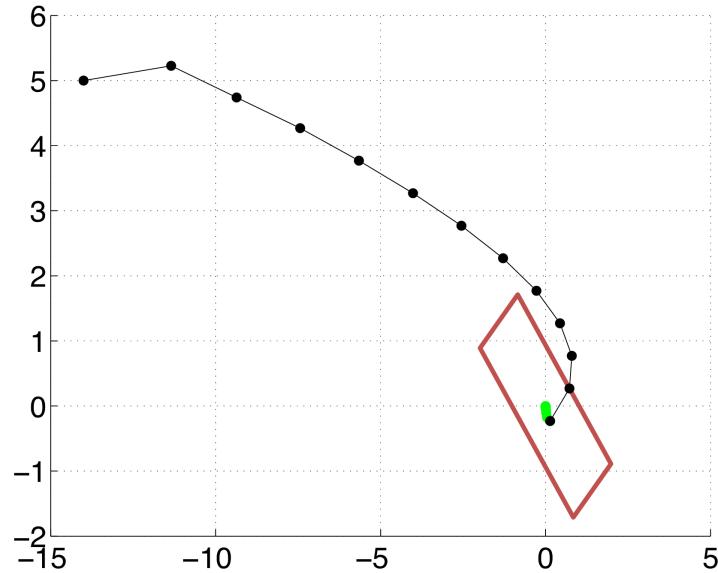
where $x_0 = s_t$

$$x_{k+1} = Ax_k + Bu_k$$

$$x_k \in \mathcal{X}, \quad u_k \in \mathcal{U}$$

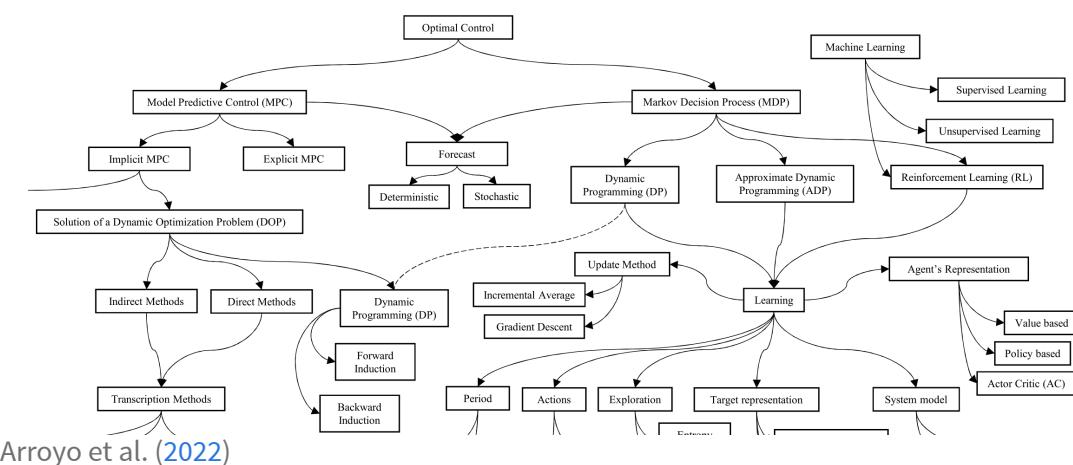
$$u_k = -Kx_k, \quad N_c \leq k < N$$

Mimic infinite hori



144

MPC + RL



146

Main motivation

MPC

- + Safety by design
- + Modularity
- Manual design
- Rigid

RL

- + Model-free
- + Flexible objectives
- Safety constraints
- Slowish learning

147

MPC + value function

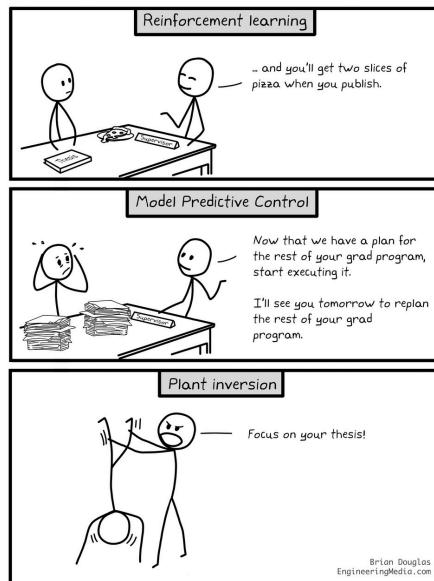
$$\min_{u_0, u_1, \dots, u_{N-1}} \sum_{k=0}^{N-1} \ell(x_k, u_k) + V_\theta(x_N)$$

where $x_0 = s_t$
 $x_{k+1} = f(x_k, u_k)$
 $x_k \in \mathcal{X}, \quad u_k \in \mathcal{U}$

148

Break

Supervising controllers to stabilize a chaotic grad student



149

Implementation

151

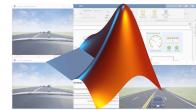
MPC frameworks



Sopasakis, Fresk, and Patrinos (2020)



Fiedler et al. (2023)



Model Predictive Control Toolbox

acados—a modular open-source framework for fast embedded optimal control

Verschueren et al. (2022)

A software framework for embedded nonlinear model predictive control using a gradient-based augmented Lagrangian approach (GRAMPC)

Englert et al. (2019)

MATMPC - A MATLAB Based Toolbox for Real-time Nonlinear Model Predictive Control

Chen et al. (2019)

MPCTools: Nonlinear Model Predictive Control Tools for CasADi (Python Interface)

MPC Tools

152

MPC frameworks

[do-mpc](#):

- Open source
- Modular

- Python interface
- Fast



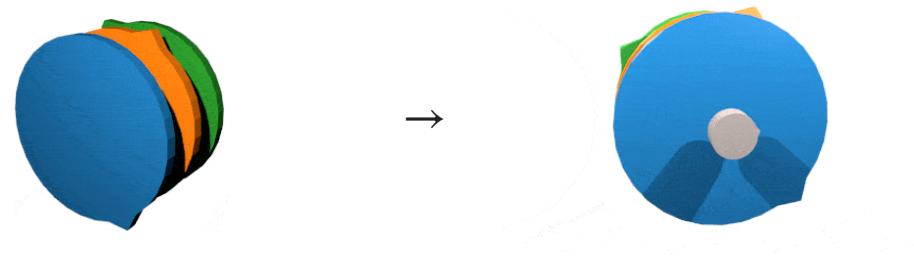
Fiedler et al. (2023)



Andersson et al. (2019)

153

Example: Triple mass spring system

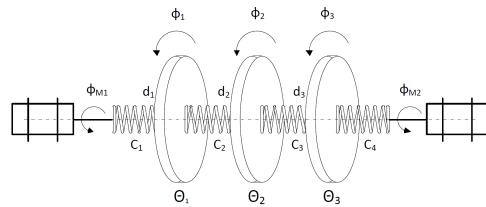


See [notebook](#) from [do-mpc](#) for full code samples. We only show snippets of the key destinations.

154

Create model

```
1 import do_mpc
2 from casadi import *
3
4 model_type = 'continuous' # either 'discrete' or 'continuous'
5 model = do_mpc.model.Model(model_type)
6
7 dphi = model.set_variable(var_type='_x', var_name='dphi', shape=(3,
8 # Two states for the desired (set) motor position:
9 phi_m_1_set = model.set_variable(var_type='_u', var_name='phi_m_1_s'
10 phi_m_2_set = model.set_variable(var_type='_u', var_name='phi_m_2_s'
```



155

Right-hand-side equation

Define the states, inputs, parameters, and function composing an ODE

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

```
1 Theta_1 = model.set_variable('parameter', 'Theta_1')
2 Theta_2 = model.set_variable('parameter', 'Theta_2')
3 Theta_3 = model.set_variable('parameter', 'Theta_3')
4
5 c = np.array([2.697, 2.66, 3.05, 2.86])*1e-3
6 d = np.array([6.78, 8.01, 8.82])*1e-5
7
8 dphi_next = vertcat(
9     -c[0]/Theta_1*(phi_1-phi_1_m)-c[1]/Theta_1*(phi_1-phi_2)-d[0]/Theta_1*(phi_1-phi_3)
10    -c[1]/Theta_2*(phi_2-phi_1)-c[2]/Theta_2*(phi_2-phi_3)-d[1]/Theta_2*(phi_2-phi_2_m)
11    -c[2]/Theta_3*(phi_3-phi_2)-c[3]/Theta_3*(phi_3-phi_2_m)-d[2]/Theta_3*(phi_3-phi_3_m)
12 )
13
14 model.set_rhs('dphi', dphi_next)
15 model.setup()
```

156

Create controller

```
1 mpc = do_mpc.controller.MPC(model)
2
3 setup_mpc = {
4     'n_horizon': 20,
5     't_step': 0.1,
6     'n_robust': 1,
7     'store_full_solution': True,
8 }
9 mpc.set_param(**setup_mpc)
```

(Defining constraints, the objective function, and even uncertain parameters, all follow a similar workflow)

```
1 mpc.setup()
```

157

Define simulator

Either use the same model inside the MPC or define a different model to simulate the “true” system:

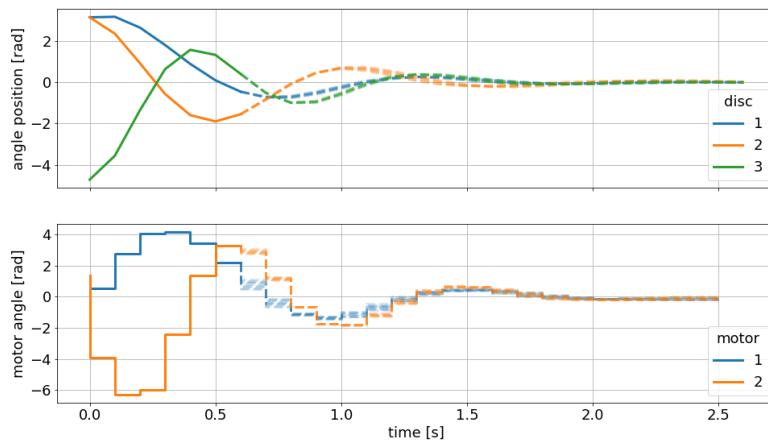
- Simplified MPC model
- Complex “true” simulator model

```
1 simulator = do_mpc.simulator.Simulator(model)
```

158

Run the control loop

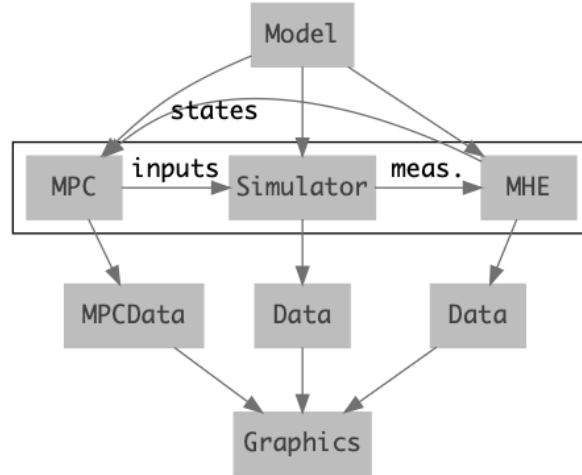
```
1 for i in range(20):
2     u0 = mpc.make_step(x0)
3     x0 = simulator.make_step(u0)
```



159

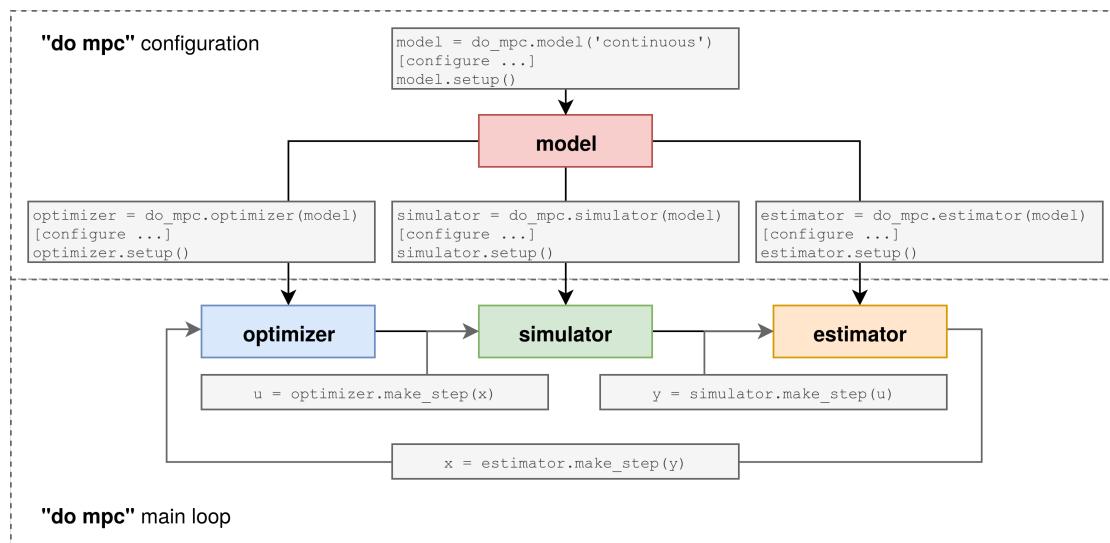
Creating these gifs is easy with do-mpc's **Graphics** and **Data** modules

```
1 mpc_graphics = do_mpc.graphics.Graphics(mpc.data)
2 sim_graphics = do_mpc.graphics.Graphics(simulator.data)
```



160

do-mpc summary



161

RL frameworks



CleanRL: High-quality Single-file Implementations of Deep Reinforcement Learning Algorithms

Raffin et al. (2021)

Huang et al. (2022)



Spinning Up



Hoffman et al. (2022)



Bou et al. (2023)



J. Weng et al. (2022)



Liang et al. (2018)

(...A really long list [here](#)) 162

RL frameworks

[CleanRL](#):

- Self-contained implementations
- Rapid prototyping
- Thorough documentation and benchmarking
- [Gym](#) for environments and [wandb](#) for tracking



Towers et al. (2023), Brockman et al. (2016)



Biewald et al. (2020)

CleanRL:

- 1 algorithm gets 1 file
- Read, learn, and modify in a linear fashion
- 300-400 lines of code
 - Including all utilities

Modular libraries:

model.learn()



```
1. stable_baselines3/ppo/ppo.py — 315 LOC, 51 lines of docstring (LOD)
2. stable_baselines3/common/on_policy_algorithm.py — 280 LOC, 49 LOD
3. stable_baselines3/common/base.class.py — 819 LOC, 231 LOD
4. stable_baselines3/common/utils.py — 506 LOC, 195 LOD
5. stable_baselines3/common/env_util.py — 157 LOC, 43 LOD
6. stable_baselines3/common/atari_wrappers.py — 249 LOC, 84 LOD
7. stable_baselines3/common/vec_env/_init__.py — 73 LOC, 24 LOD
8. stable_baselines3/common/vec_env/dummy_vec_env.py — 126 LOC, 25 LOD
9. stable_baselines3/common/vec_env/base_vec_env.py — 375 LOC, 112 LOD
10. stable_baselines3/common/vec_env/util.py — 77 LOC, 31 LOD
11. stable_baselines3/common/vec_env/vec_frame_stack.py — 65 LOC, 14 LOD
12. stable_baselines3/common/vec_env/stacked_observations.py — 267 LOC, 74 LOD
```

⋮

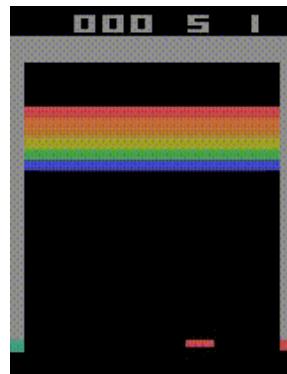
Total lines of code: 7759

164

Like with the MPC packages, we’re choosing the best one for our purposes—the other ones are absolutely worth looking into!

165

Break?



[Open RL Benchmark](#) (Huang et al. 2024)

166

Combining RL and MPC

Recall what we're after:

$$\min_{u_0, u_1, \dots, u_{N-1}} \sum_{k=0}^{N-1} \ell(x_k, u_k) + \overbrace{V_\theta(x_N)}^{\text{Learnable residual}}$$

where $x_0 = s_t$
 $x_{k+1} = f(x_k, u_k)$
 $\underbrace{x_k \in \mathcal{X}, u_k \in \mathcal{U}}_{\text{Prior engineering}}$

167

Combining RL and MPC

CleanRL:

1. Value function learning
2. Environment implementation

do-mpc:

1. Optimization module
2. Simulation

168

1. RL value function in do-mpc

Value function training

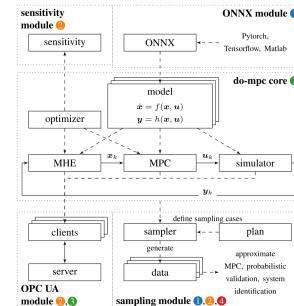
```
1 data = rb.sample(args.batch_size)
2 with torch.no_grad():
3     next_state_actions, next_state_log_pi, _ = actor.get_action(data)
4     qf1_next_target = qf1_target(data.next_observations, next_state_actions)
5     qf2_next_target = qf2_target(data.next_observations, next_state_actions)
6     min_qf_next_target = torch.min(qf1_next_target, qf2_next_target)
7     next_q_value = (data.rewards.flatten() +
8                      (1 - data.dones.flatten()) * args.gamma * (min_qf_next_target))
9
10    qf1_a_values = qf1(data.observations, data.actions).view(-1)
11    qf2_a_values = qf2(data.observations, data.actions).view(-1)
12    qf1_loss = F.mse_loss(qf1_a_values, next_q_value)
13    qf2_loss = F.mse_loss(qf2_a_values, next_q_value)
14    qf_loss = qf1_loss + qf2_loss
15
16    # optimize the model
17    q_optimizer.zero_grad()
18    qf_loss.backward()
19    q_optimizer.step()
```

169

1. RL value function in do-mpc

Export value function to do-mpc:

1. Export PyTorch model to **ONNX** using `torch.onnx.export()`
2. Export ONNX model to CasADI `do_mpc.sysid.ONNXConversion()`



Fiedler et al. (2023)

170

1. RL value function in do-mpc

Implement MPC with terminal value function:

```
1 mpc = do_mpc.controller.MPC(model)
2
3 mpc.settings.n_horizon = 1
4 lterm = model.aux['cost']
5
6 terminal_converter = do_mpc.sysid.ONNXConversion(value_onnx)
7 def terminal_casadi(x):
8     terminal_converter.convert(x = x.T, goal=np.zeros(x.T.shape))
9     return terminal_converter['terminal_cost']
10 mterm = -terminal_casadi(model.x['x'])
11
12 mpc.set_objective(lterm=lterm, mterm=mterm)
13
14 mpc.bounds['lower', '_u', 'u'] = -0.5
15 mpc.bounds['upper', '_u', 'u'] = 0.5
16
17 mpc.setup()
```

171

1. RL value function in do-mpc

Run MPC + value function controller in CleanRL:

```
1 x0 = estimator.make_step(obs["observation"])
2 action = mpc.make_step(x0)
3
4 # A quick way of incorporating exploration
5 with torch.no_grad():
6     if global_step % args.policy_frequency == 0: # optional: add so
7         noise = actor._explore_noise(obs).numpy()
8         action += noise
9         action = np.float32(action.clip(env.action_space.low, env.a
10
11 next_ob, reward, done, info = env.step(action)
```

Next, what is `env.step()`?

172

2. Gym wrapper for do-mpc simulation

Zooming out a bit, a basic RL loop looks like this:

```
1 import gymnasium as gym
2
3 env = gym.make("LunarLander-v2", render_mode="human")
4 observation, info = env.reset()
5
6 for _ in range(1000):
7     action = env.action_space.sample() # agent policy that uses th
8     observation, reward, terminated, truncated, info = env.step(act
9
10    if terminated or truncated:
11        observation, info = env.reset()
12
13 env.close()
```

173

Environments follow a basic blueprint:

```
1 from gymnasium import spaces
2
3 class CustomEnv(gym.Env):
4     def __init__(self, arg1, arg2, ...):
5         super(CustomEnv, self).__init__()
6         self.action_space = spaces.Discrete(N_DISCRETE_ACTIONS)
7         self.observation_space = spaces.Box(low=0, high=255, shape=(HEI
8
9     def step(self, action):
10        ...
11        return observation, reward, done, info
12    def reset(self):
13        ...
14        return observation
15    def render(self, mode='human'):
16        ...
17    def close (self):
18        ...
```

174

Create Gym environment that queries do-mpc simulation:

```
1 class DoMPCEnv(gym.Env):
2     """
3         Gym environment that uses do-mpc for carrying out simulations
4     """
5     def __init__(self, simulator:do_mpc.simulator.Simulator,
6                  num_steps=100):
7         super().__init__()
8         ...
9
10    def step(self, action):
11        # simplified version --- hides some processing steps
12
13        self.t += 1
14        self.state = self.simulator.make_step(action)
15        info = self._get_info()
16        reward, terminated, truncated = info["reward"], info["dista
17
18        return self.state, reward, terminated, truncated, info
19        ...
```

175

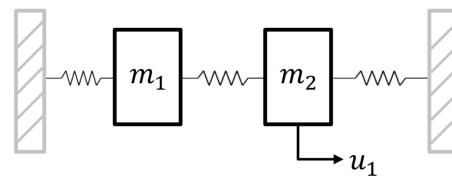
Example: Oscillating masses

State: Position and velocity of each mass

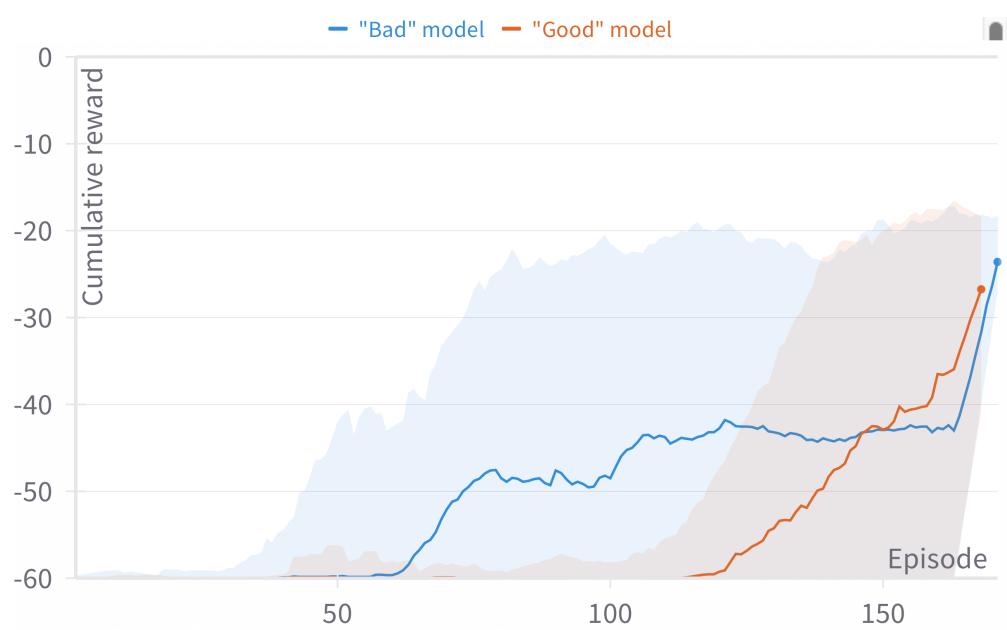
Action: Force applied to m_2

MPC cost: $\|x\|^2$

Reward: 0 if $\|x\|_\infty \leq \epsilon$; -1 otherwise



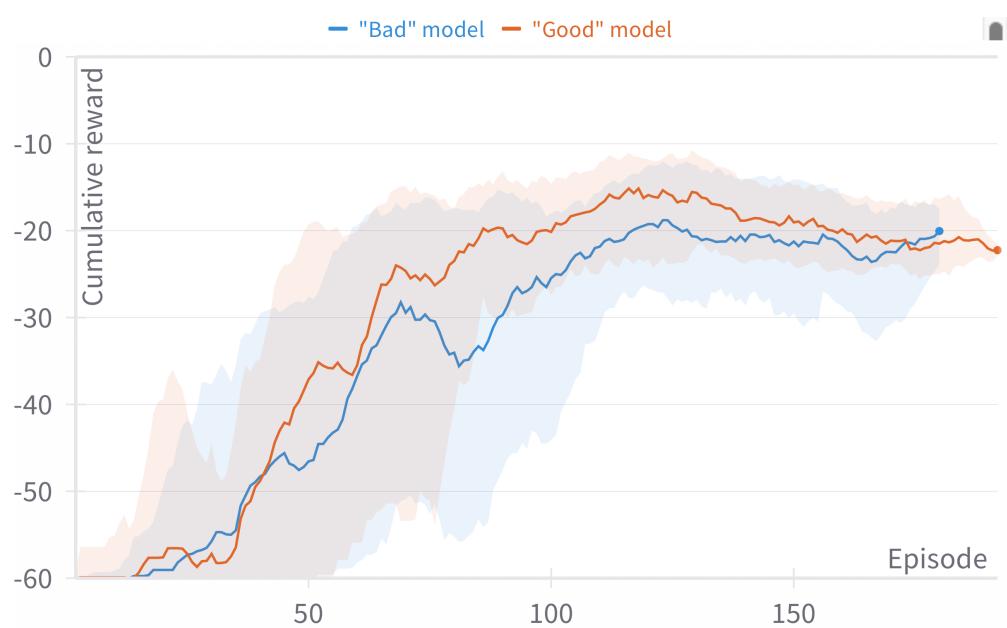
176



177

(placeholder—hindsight experience replay and learning from sparse rewards)

178

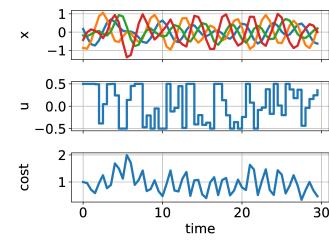
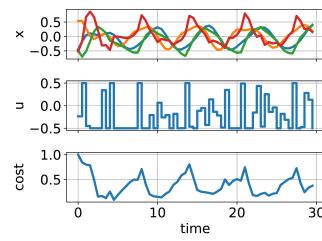


179

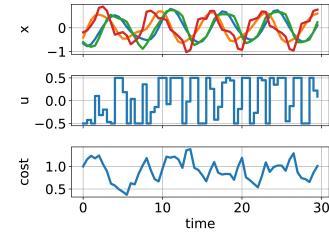
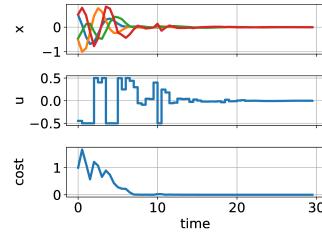
“Bad” model

“Good” model

No relabeling



Relabeling

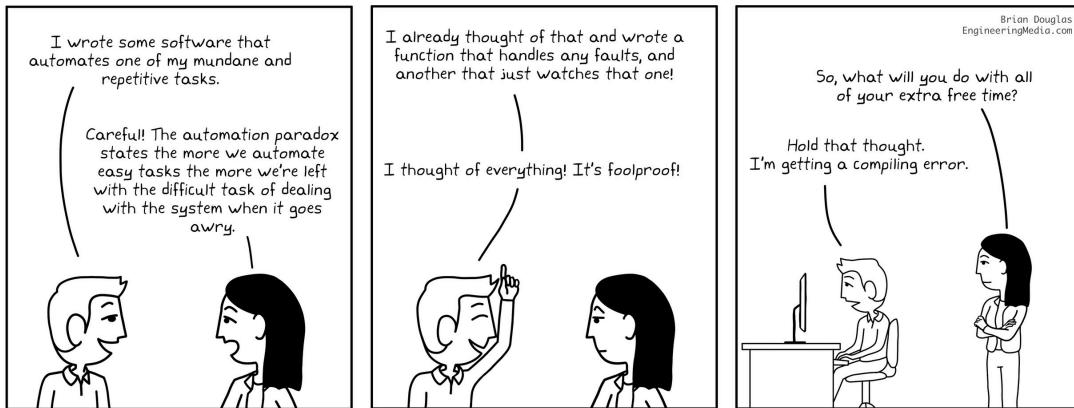


180

Studies in the literature

181

Automation Paradox



182

Addendum

184

Nonlinear MPC problem formulation

185

State estimation, offset-free tracking

186

Differentiable MPC

187

Industrial control: theory vs practice

(placeholder — thinking of Michael's paper, also Siang/Sham's)

188

Thank you

190

- Martha White & Upper Bound organizers
- Philip Loewen
- Bhushan Gopaluni
- Michael Forbes
- Shuyuan Wang
- Thiago da Cunha Vasco
- DAIS Lab
- NSERC

191

Slides and code



Link: <https://github.com/NPLawrence/upperbound-tutorial>

192

References

- Andersson, Joel A E, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. 2019. “CasADI – A Software Framework for Nonlinear Optimization and Optimal Control.” *Mathematical Programming Computation* 11 (1): 1–36. <https://doi.org/10.1007/s12532-018-0139-4>.
- Arroyo, Javier, Carlo Manna, Fred Spiessens, and Lieve Helsen. 2022. “Reinforced Model Predictive Control (RL-MPC) for Building Energy Management.” *Applied Energy* 309: 118346. <https://doi.org/10.1016/j.apenergy.2021.118346>.
- Åström, Karl Johan, and Richard M Murray. 2021. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton university press.
- Bemporad, Alberto, Manfred Morari, Vivek Dua, and Efstratios N Pistikopoulos. 2002. “The Explicit Linear Quadratic Regulator for Constrained Systems.”
- Bertsekas, Dimitri. 2023. *A Course in Reinforcement Learning*. Athena Scientific.
- Biewald, Lukas et al. 2020. “Experiment Tracking with Weights and Biases.” *Software Available from Wandb. Com* 2 (5). <https://wandb.ai/site/>.
- Borrelli, Francesco, Alberto Bemporad, and Manfred Morari. 2017. *Predictive Control for Linear and Hybrid Systems*. Cambridge University Press.
- Bou, Albert, Matteo Bettini, Sebastian Dittert, Vikash Kumar, Shagun Sodhani, Xiaomeng Yang, Gianni De Fabritiis, and Vincent Moens. 2023. “TorchRL: A Data-Driven Decision-Making Library for PyTorch.” arXiv.

<https://doi.org/10.48550/arXiv.2306.00577>.

- Brockman, Greg, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. “OpenAI Gym.” <https://github.com/openai/gym>.
- Chen, Yutao, Mattia Bruschetta, Enrico Picotti, and Alessandro Beghi. 2019. “MATMPC - A MATLAB Based Toolbox for Real-time Nonlinear Model Predictive Control.” In *2019 18th European Control Conference (ECC)*, 3365–70. Naples, Italy: IEEE. <https://doi.org/10.23919/ECC.2019.8795788>.
- Desborough, Lane, and Randy Miller. 2002. “Increasing Customer Value of Industrial Control Performance Monitoring -Honeywell’s Experience.” *AIChE Symposium Series* 98.
- Englert, Tobias, Andreas Völz, Felix Mesmer, Sönke Rhein, and Knut Graichen. 2019. “A Software Framework for Embedded Nonlinear Model Predictive Control Using a Gradient-Based Augmented Lagrangian Approach (GRAMPC).” *Optimization and Engineering* 20 (3): 769–809. <https://doi.org/10.1007/s11081-018-9417-2>.
- Fiedler, Felix, Benjamin Karg, Lukas Lüken, Dean Brandner, Moritz Heinlein, Felix Brabender, and Sergio Lucia. 2023. “Do-Mpc: Towards FAIR Nonlinear and Robust Model Predictive Control.” *Control Engineering Practice* 140: 105676. <https://doi.org/10.1016/j.conengprac.2023.105676>.
- Fujimoto, Scott, Herke van Hoof, and David Meger. 2018-07-10/2018-07-15. “Addressing Function Approximation Error in Actor-Critic Methods.” In *Proceedings of the 35th International Conference on Machine Learning*, 80:1587–96. Proceedings of Machine Learning Research. PMLR. <https://proceedings.mlr.press/v80/fujimoto18a.html>.

Haarnoja, Tuomas, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, et al. 2019. “Soft Actor-Critic Algorithms and Applications.” *arXiv:1812.05905 [Cs, Stat]*. <http://arxiv.org/abs/1812.05905>.

Hoffman, Matthew W., Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Nikola Momchev, Danila Sinopalnikov, Piotr Stańczyk, et al. 2022. “Acme: A Research Framework for Distributed Reinforcement Learning.” *arXiv*. <https://doi.org/10.48550/arXiv.2006.00979>.

Huang, Shengyi, Rousslan Fernand Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G. M. Araújo. 2022. “CleanRL: High-quality Single-file Implementations of Deep Reinforcement Learning Algorithms.” *Journal of Machine Learning Research* 23 (274): 1–18. <http://jmlr.org/papers/v23/21-1342.html>.

Huang, Shengyi, Quentin Gallouédec, Florian Felten, Antonin Raffin, Rousslan Fernand Julien Dossa, Yanxiao Zhao, Ryan Sullivan, et al. 2024. “Open RL Benchmark: Comprehensive Tracked Experiments for Reinforcement Learning.” *arXiv*. <https://doi.org/10.48550/arXiv.2402.03046>.

Kalman, Rudolf Emil. 1960. “Contributions to the Theory of Optimal Control” 5 (2): 102–19.

Karg, Benjamin, and Sergio Lucia. 2020. “Efficient Representation and Approximation of Model Predictive Control Laws via Deep Learning.” *IEEE Transactions on Cybernetics* 50 (9): 3866–78. <https://doi.org/10.1109/TCYB.2020.2999556>.

Liang, Eric, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. 2018. “RLLib: Abstractions for

- Distributed Reinforcement Learning.” In *Proceedings of the 35th International Conference on Machine Learning*, 3053–62. PMLR.
<https://proceedings.mlr.press/v80/liang18b.html>.
- Lillicrap, Timothy P., Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. “Continuous Control with Deep Reinforcement Learning.” <https://doi.org/10.48550/ARXIV.1509.02971>.
- Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. “Playing Atari with Deep Reinforcement Learning.” arXiv. <https://doi.org/10.48550/arXiv.1312.5602>.
- Montufar, Guido F, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. 2014. “On the Number of Linear Regions of Deep Neural Networks.” *Advances in Neural Information Processing Systems* 27.
- Raffin, Antonin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. 2021. “Stable-Baselines3: Reliable Reinforcement Learning Implementations.” *Journal of Machine Learning Research* 22 (268): 1–8.
<http://jmlr.org/papers/v22/20-1364.html>.
- Sopasakis, P., E. Fresk, and P. Patrinos. 2020. “OpEn: Code Generation for Embedded Nonconvex Optimization.” In *IFAC World Congress*. Berlin.
- Sutton, Richard S., and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. Second edition. Adaptive Computation and Machine Learning Series. Cambridge, Massachusetts: The MIT Press.
- Towers, Mark, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, et al. 2023. “Gymnasium.” Zenodo.

<https://doi.org/10.5281/zenodo.8127026>.

- Verschueren, Robin, Gianluca Frison, Dimitris Kouzoupis, Jonathan Frey, Niels Van Duijkeren, Andrea Zanelli, Branimir Novoselnik, Thivaharan Albin, Rien Quirynen, and Moritz Diehl. 2022. “Acados—a Modular Open-Source Framework for Fast Embedded Optimal Control.” *Mathematical Programming Computation* 14 (1): 147–83. <https://doi.org/10.1007/s12532-021-00208-8>.
- Weng, Jiayi, Huayu Chen, Dong Yan, Kaichao You, Alexis Duburcq, Minghao Zhang, Yi Su, Hang Su, and Jun Zhu. 2022. “Tianshou: A Highly Modularized Deep Reinforcement Learning Library.” arXiv. <https://doi.org/10.48550/arXiv.2107.14171>.
- Weng, Lilian. 2018. “A (Long) Peek into Reinforcement Learning.” *Lilianweng.github.io*.
<https://lilianweng.github.io/posts/2018-02-19-rl-overview/>.