A reinforcement learning perspective on industrial model predictive control

Upper Bound 2024

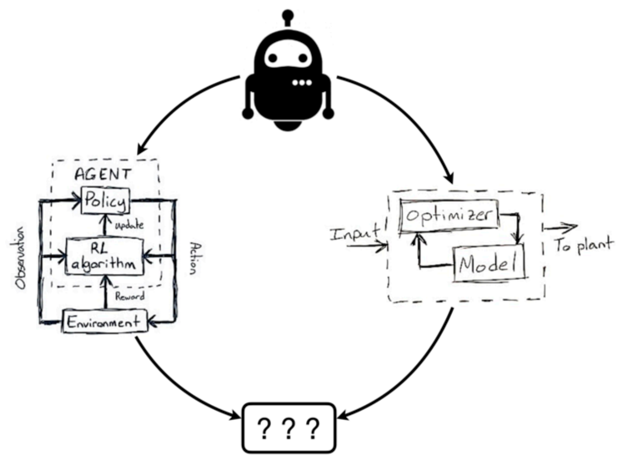
AUTHOR
Nathan Lawrence

AFFILIATION
UBC mathematics

PUBLISHED
May 25, 2024

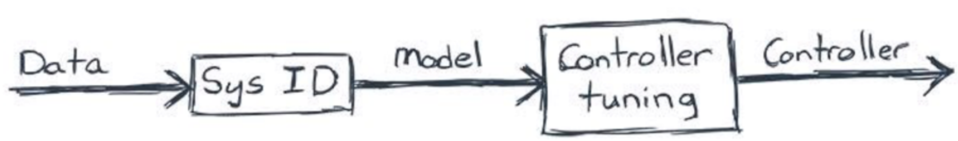
ABSTRACT
Online version of the presentation given at Upper Bound 2024. Code and slides available here.

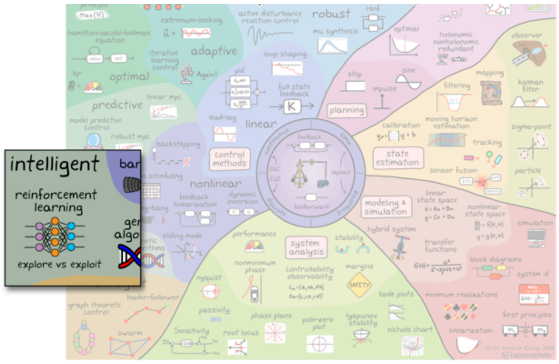
Sketches–Brian Douglas



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…





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, discuss with your neighbor :-)

Cheat sheet! (github)

Reinforcement learning

Agents and environments

Apply torque

Observe

- Angle
- Angular velocity

Environment

s_t – state

a_t – action

Actions and states lead to new states

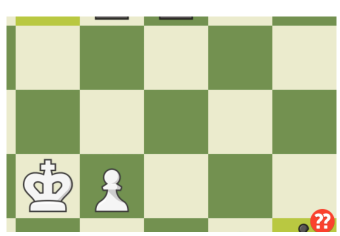
$$s_{t+1} \sim p\left(s_{t+1} | s_t, a_t\right)$$

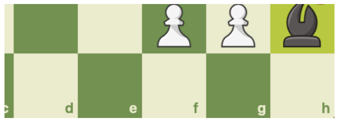
Continuing forever gives a trajectory

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

Most trajectories are “bad”

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





Agent

The agent is the learner:

- Decides which actions to take
- Improves its behavior

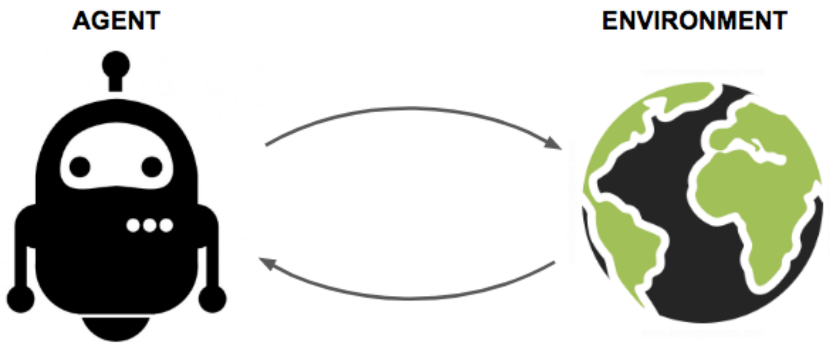


Figure: L. Weng (2018)

Reward guides learning:

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

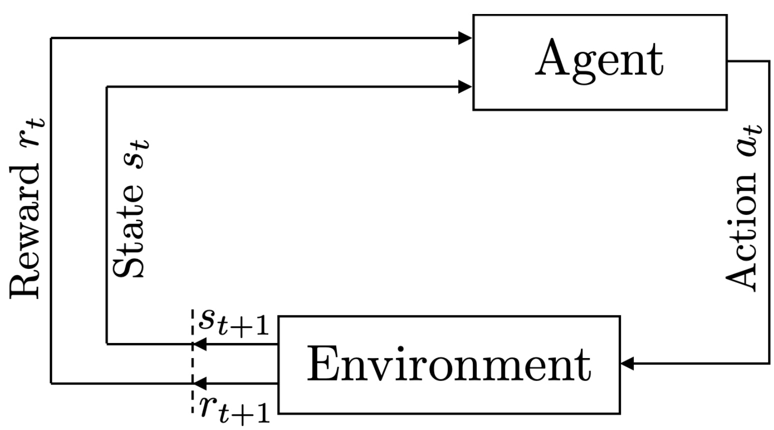


Figure: Sutton and Barto (2018)

What are we really trying to solve?

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

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

Agent chooses actions from a **policy**

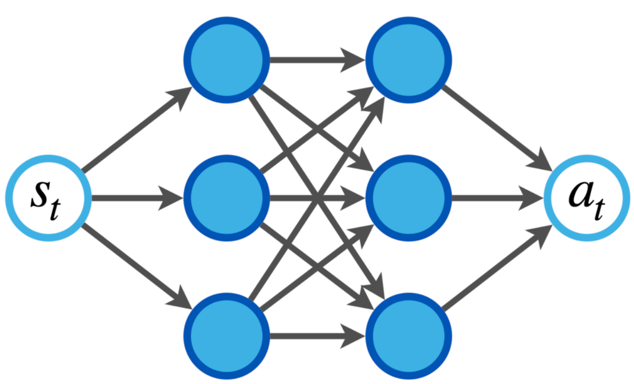
$$a_t \sim \pi\left(a_t|s_t\right)$$

Rewards assigned by a function

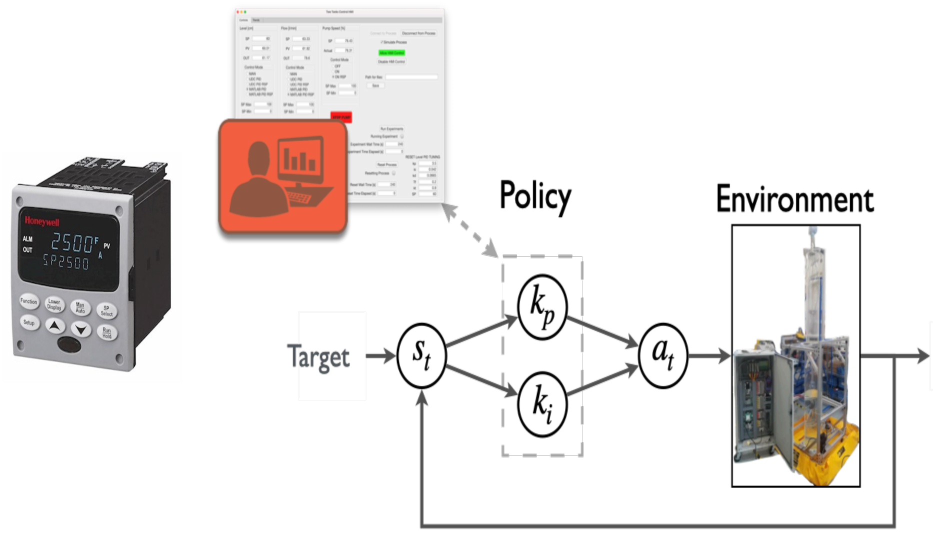
$$r_t = r(s_t, a_t)$$

The space of policies is vast

- Completely random
- Industrial control module
- Neural network



Restricting the policy space is practical



See Lawrence et al. (2022)

We want the “best” policy!

- Take $a \in \operatorname{argmax}_a \{r(s,a)\}$?

$\longrightarrow$ too shortsighted

- Maximize $r_0 + r_1 + r_2 + \ldots$?

$\longrightarrow$ not enough urgency

(Also, might diverge, which is bad.)

Discounted **return**

✅ Pick $\gamma \in [0,1]$ and consider

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

- A loonie now?
- Or γ^t cents later?

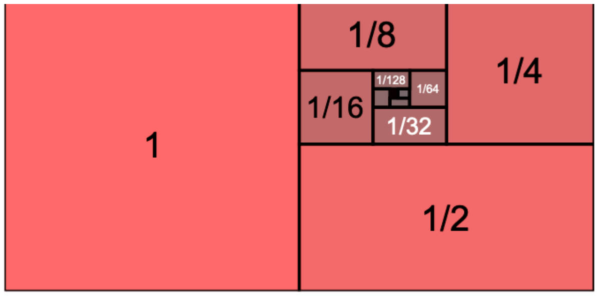
Discounted returns converge

- Typically, $\gamma \in (0,1)$
- Assume all rewards fit inside $[-\bar{r}, \bar{r}]$

Return is Upper Bounded across all possible trajectories

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

Geometric series (Wikipedia)



The reinforcement learning problem

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

Why is this hard/impossible?

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

Wishful thinking

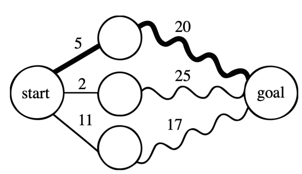
What if we had an optimal trajectory?

That is,

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

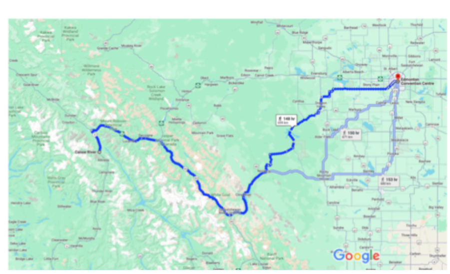
Then the trajectory starting at $s_t^\star$ should still be optimal

$$s_t^\star, a_t^\star, r_t^\star, s_{t+1}^\star, \ldots$$



Optimal substructure (Wikipedia)

Self-consistency is key



Value functions

For a given policy π...

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

Aka "Q-function"

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

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

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

Abstracting the objective through value functions

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

What's the big deal with these magical functions?

Observe: for any specific policy π

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

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

What's the big deal with these magical functions?

Averaging...

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

What's the big deal with these magical functions?

Similarly for Q^π...

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

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 for any policy!

Optimal policy

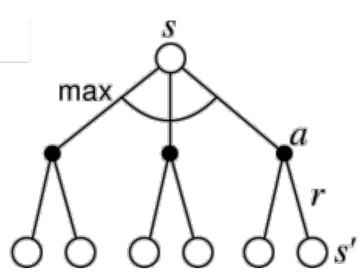
- $\pi^\star$ – optimal policy
- We don't actually have $\pi^\star$, but it's fun to imagine...
- $\pi^\star$ solves the following:

$$
V^\star(s) = \max_\pi V^\pi(s) \qquad Q^\star(s,a) = \max_\pi Q^\pi(s,a)
$$

Bellman optimality equation

Given $V^\star$...

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



One-step planning is long-term optimal

Bellman optimality equation

Given $Q^\star$...

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

...Even easier!

Therefore,

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

Learn $Q^\star$ and then maximize it!

Evaluate, improve, repeat...

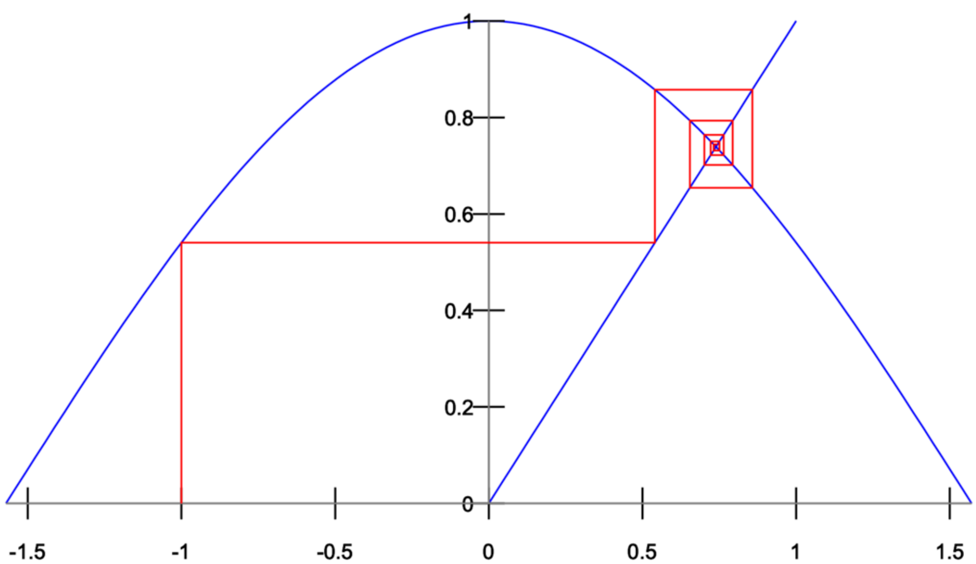
Like $V^\star$, $Q^\star$ satisfies a neat optimality condition:

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

"Plug this magical function $Q^\star$ into the RHS produces the same function"

Fixed-point iteration

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



Close your eyes and exclaim "Bellman!"

Fixed-point iteration in value space

- Let's just take the operator

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

and apply fixed-point iteration!

- …"Just"?
- The RHS is an **idealized** update scheme and key source of inspiration

Fixed-point aspirations

If we have a policy π and some oracle that tells us Q^π…

Then take

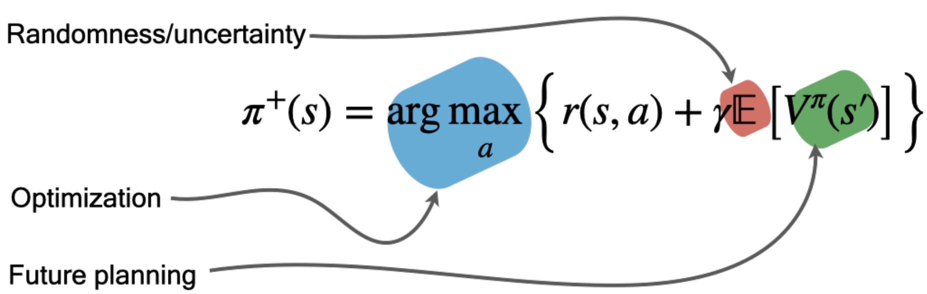
$$\begin{aligned}\pi^+(s) &= \arg\max_a Q^\pi(s,a) \\ &= \arg\max_a \left\{ r(s,a) + \gamma \mathbb{E}_{s' \sim p(s'|s,a)} \left[V^\pi(s') \right] \right\}\end{aligned}$$

- Then π^+ is at least as rewarding as π!
- When improvement is no longer possible, we have

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

Three important approximations

See works by Dimitri Bertsekas (most recently Bertsekas (2023))



There is a lot of jargon to get into:

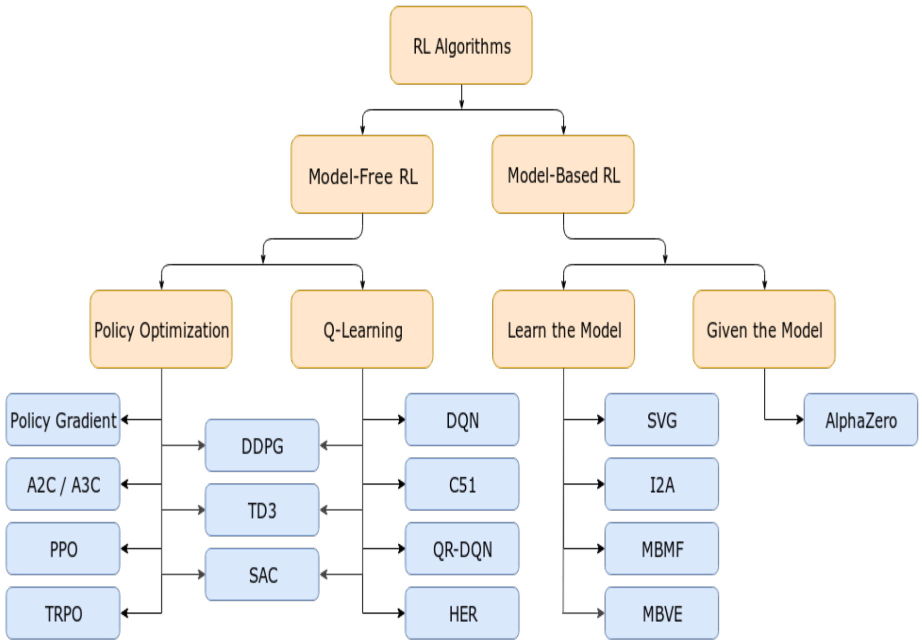
- Policy evaluation
- Policy iteration
- Value iteration
- Temporal-difference learning, Monte Carlo…
- On-policy, off-policy…

- SARSA, Q-Learning, policy gradients, etc…
- "REINFORCE": *RE*ward *I*ncrement = *N*onnegative *F*actor $\times$ *O*ffset *R*einforcement $\times$ *C*haracteristic *E*ligibility (Williams (1992))

We're focusing on high-level ideas and need to get to MPC

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

Figure: Spinning Up



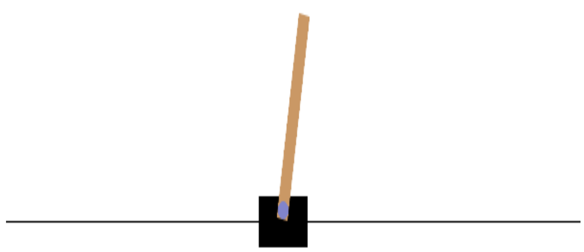
Break

Open RL Benchmark (Huang et al. 2024)

Examples

Learning to balance

Cartpole



Don't fall!

Deep Q-networks (DQNs)

For a **finite** set of actions $\{a_1, a_2, \ldots, a_d\}$, use a neural network to define

Mnih et al. (2013)

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

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

Optimization is trivial:

Optimization is trivial:

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

But approximation is difficult

Playing Atari with Deep Reinforcement Learning

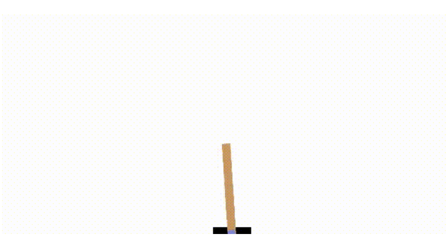
Volodymyr Mnih Koray Kavukcuoglu David Silver Alex Graves Ioannis Antonoglou

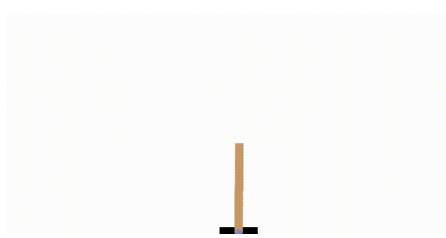
Daan Wierstra Martin Riedmiller

Equation 3: Approximately respect the Bellman equation $[q_1, \ldots, q_d] \approx r(s, a) + \gamma \max\{q'_1, \ldots q$

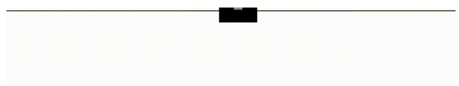
Algorithm 1 Deep Q-learning with Experience Replay

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 ϵ 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

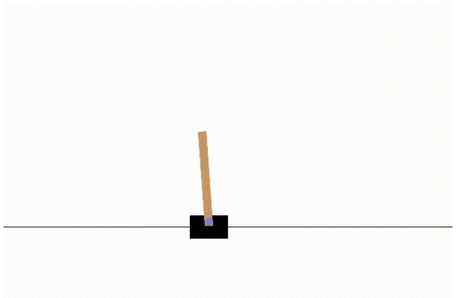




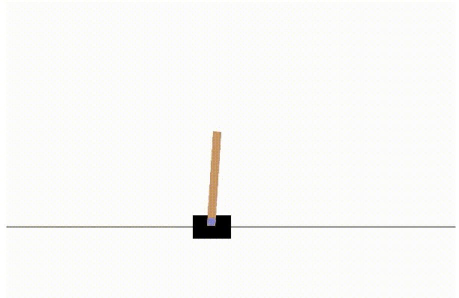
Initial



After some time...

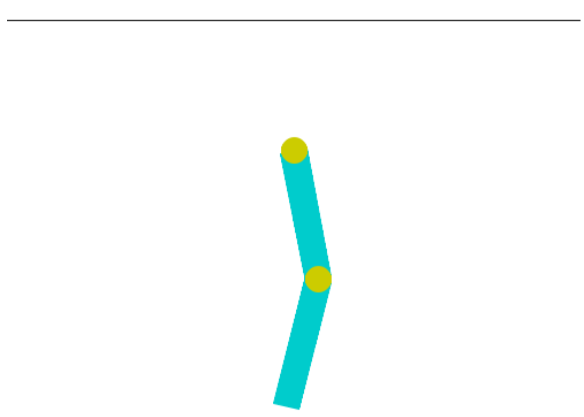


Almost there...



Final

Acrobot

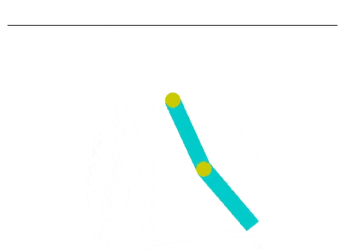


~~Touch the line!~~

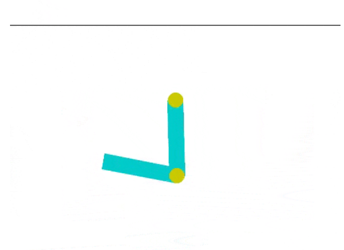
Stay above the line!

Applying DQN...

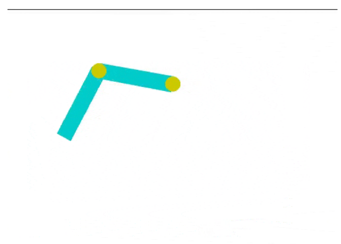
Standing upright would be better than all of these...



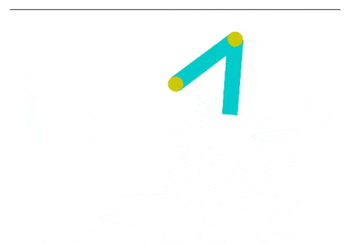
Initial



After some time...



Almost there...



Final

Finer control requires continuous actions

What we want:

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

What DQN delivers:

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

Finer control requires continuous actions

Consider two separate networks with parameters θ, ϕ:

- Policy π_θ (aka "actor")
- Q-network Q_ϕ (aka "critic")

Idea: Use Q_ϕ as a loss function for π_θ:

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

Then

$$\max_a Q^\pi(s, a) \approx Q_\phi(s, \pi_\theta(s))$$

An easy-to-evaluate approximation of the argmax operation!

Disclaimer

The simplest vanilla implementations don'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

DDPG (Lillicrap et al. 2015), TD3 (Fujimoto, van Hoof, and Meger 2018), SAC (Haarnoja et al. 2019)

Acrobot cont'd

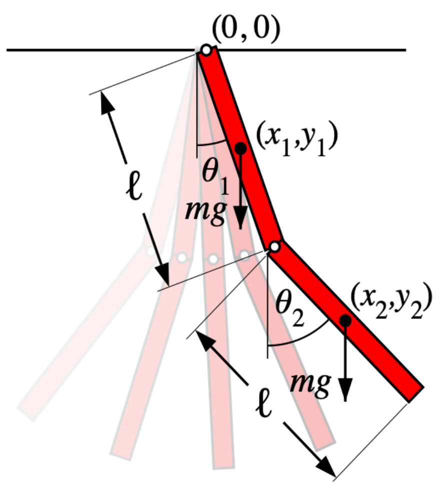
Let's assume we have some RL oracle:

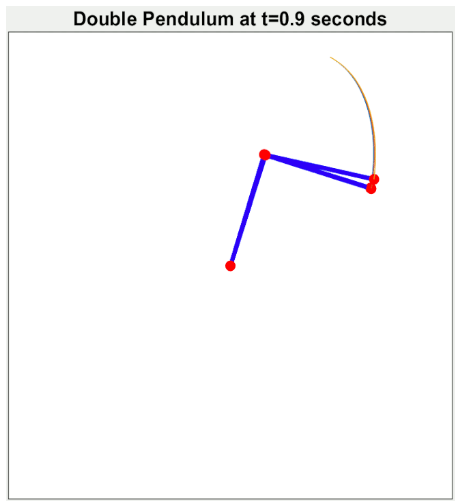
Given an environment and sufficiently powerful networks, it does a reasonable job at solving the principal RL problem

We isolate two components:

- Reward function
- Discount factor γ

Try to formulate a reward function



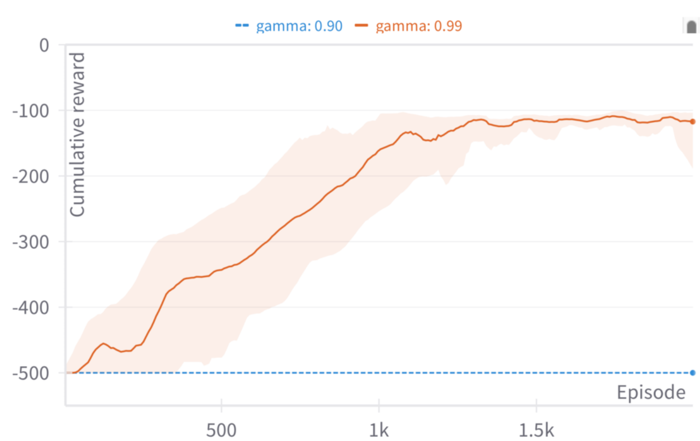


[Double pendulum [Wikipedia]]

Default reward

- 0 if above line
- -1 otherwise

"gamma" = γ (discount factor)

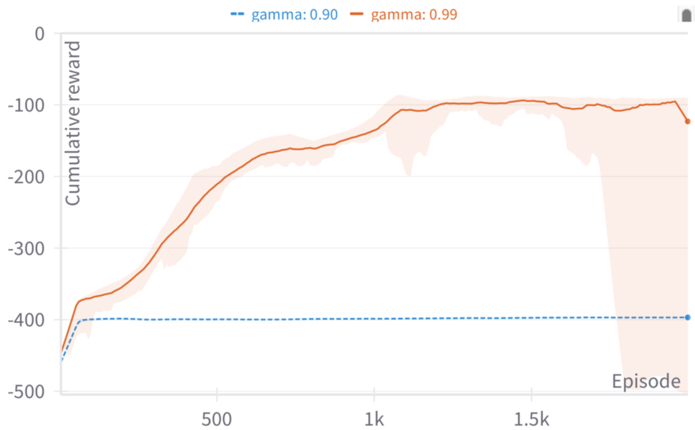


2 reward

ℓ^2 reward

- Negative 2-norm of:
  - normalized velocities
  - $\cos(\pi) - \cos(\theta_1)$, $\cos(0) - \cos(\theta_2)$

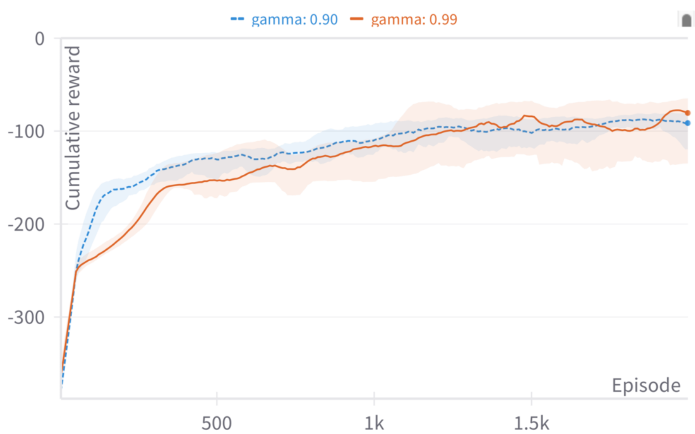
"gamma" = γ (discount factor)



"Height" reward

- Height of the outer link's endpoint

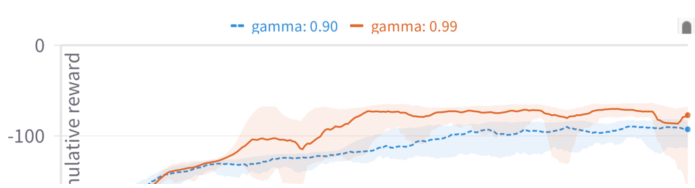
"gamma" = γ (discount factor)

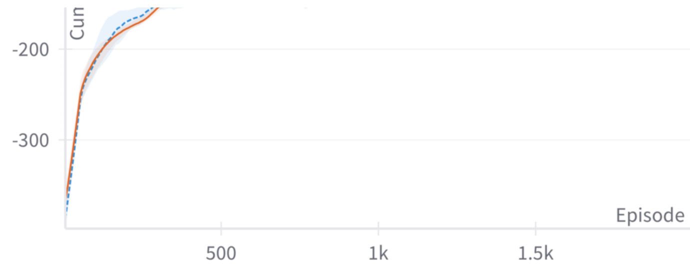


ℓ^∞ reward

- Negative ∞-norm of:
  - Deviation from maximum height
  - Normalized velocities

"gamma" = γ (discount factor)





	default	ℓ^2	height	ℓ^∞
$\gamma = 0.90$				
$\gamma = 0.99$				

1 of 4^8 configurations. Sampled via random.org.

Bloopers

I modified the default environment: default reward, spaces, sampling time

Action space:

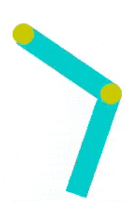
- Old: $\{-1, 0, 1\}$ (discrete)
- Intermediate: $[-1, 1]$ (continuous, restricted)
- New: $[-2, 2]$ (continuous, expanded)

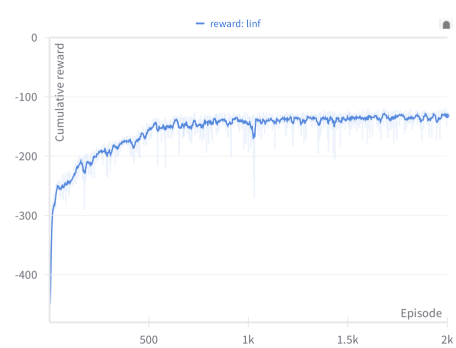
Sampling time:

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

Restricted action space

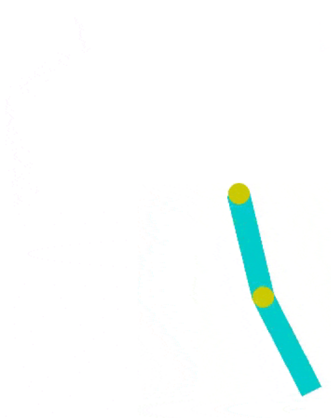
ℓ^∞ reward, $\gamma = 0.95$, 5hz

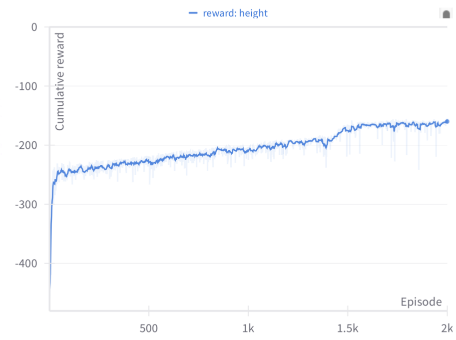




Restricted action space cont'd

"Height" reward, $\gamma = 0.99$, 5hz





An analytic solution

What if we know something about the environment?

Aside: Some notation

- RL maximizes reward
- Control minimizes cost
- RL uses states and actions s_t, a_t
- Control ... x_t, u_t

Cheat sheet! (github)

Just know we are talking about the same objects in spirit up to a simple sign flip

The original problem

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

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

Let's grossly simplify the problem

$$
\begin{aligned}
\text{minimize} \quad & \sum_{t=0}^{\infty} \gamma^t \left(x_t^2 + \rho u_t^2\right) \\
\text{where} \quad & x_{t+1} = a x_t + b u_t \\
& u_t = -k x_t
\end{aligned}
$$

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

Assign some values for simplicity

Focus on γ and k:

See appendix for general formulas

$$
\begin{aligned}
\text{minimize} \quad & \sum_{t=0}^{\infty} \gamma^t \left(x_t^2 + u_t^2\right) \\
\text{where} \quad & x_{t+1} = 1.1 x_t + 0.5 u_t \\
& u_t = -k x_t \\
& x_0 = 1.0
\end{aligned}
$$

Trajectories are easy to compute

Note $k = 0$ would be disastrous.

$$
\begin{aligned}
x_{t+1} &= 1.1 x_t + 0.5 u_t \\
u_t &= -k x_t
\end{aligned}
\quad \Longrightarrow \quad
\begin{aligned}
x_1 &= 1.1 - 0.5k \\
&\vdots \\
x_{t+1} &= (1.1 - 0.5k)^{t+1} \\
u_t &= -k(1.1 - 0.5k)^t
\end{aligned}
$$

Costs are easy to compute

$$
\begin{aligned}
\sum_{t=0}^{\infty} \gamma^t \left(x_t^2 + u_t^2\right) &= 1 + \gamma(1.1 - 0.5k)^2 + \gamma^2(1.1 - 0.5k)^4 + \ldots \\
&\quad + k^2 + \gamma k^2 (1.1 - 0.5k)^2 + \ldots \\
&= \sum_{t=0}^{\infty} \gamma^t (1.1 - 0.5k)^{2t} (1 + k^2)
\end{aligned}
$$

$$
\begin{aligned}
x_{t+1} &= (1.1 - 0.5k)^{t+1} x_0 \\
u_t &= -k(1.1 - 0.5k)^t x_0 \\
x_0 &= 1.0
\end{aligned}
$$

Quadratic value function

By properties of geometric series:

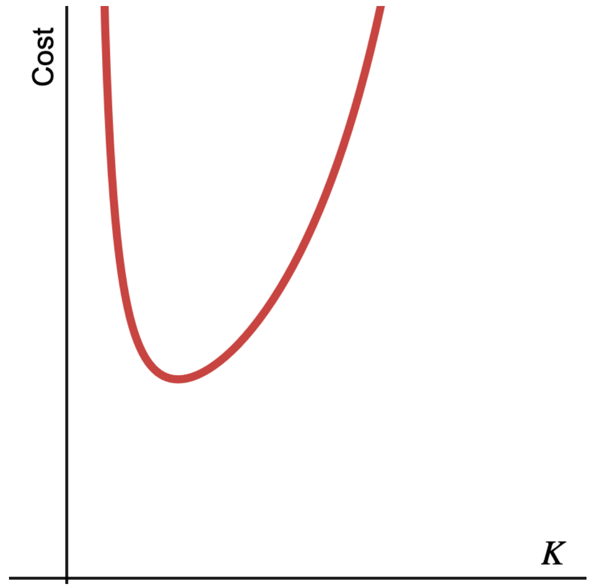
$\sum_{i=0}^{\infty} \alpha \beta^i = \alpha \frac{1}{1-\beta}$

$$
\text{return} = \sum_{t=0}^{\infty} \gamma^t (1.1 - 0.5k)^{2t} (1 + k^2)
$$

$$= \frac{1+k^2}{1-\gamma(1.1-0.5k)^2}$$

A 1-D optimization problem

$$\min_k \frac{1+k^2}{1-\gamma(1.1-0.5k)^2}$$



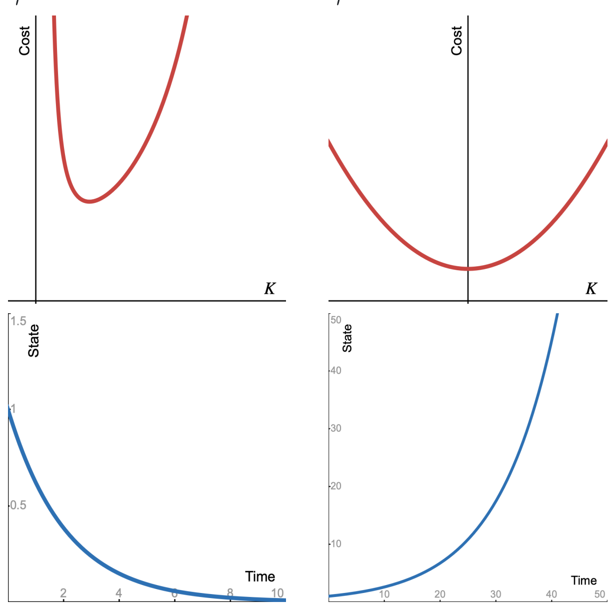
Desmos [graph]

System: $x_{t+1} = 1.1x_t + 0.5u_t$ (unstable)

$\gamma = 1$ | $\gamma = 0$

Value function

Closed-loop trajectory



Wait, are linear controllers optimal?

YES!

$$\begin{aligned} \min_{u_0, u_1, \ldots} \quad & \sum_{t=0}^{\infty} \gamma^t \left(x_t^2 + u_t^2\right) \\ \text{where} \quad & x_{t+1} = ax_t + bu_t \\ & \cancel{u_t = -kx_t} \end{aligned}$$

Define

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

$$\Downarrow \text{(fact)}$$

$$V^\star(x) = Px^2$$

The optimal value function is:

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

Bellman gives us a single variable problem:

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

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

$$u = -\frac{\gamma ab P}{1 + \gamma b^2 P} x$$

$$\Downarrow$$

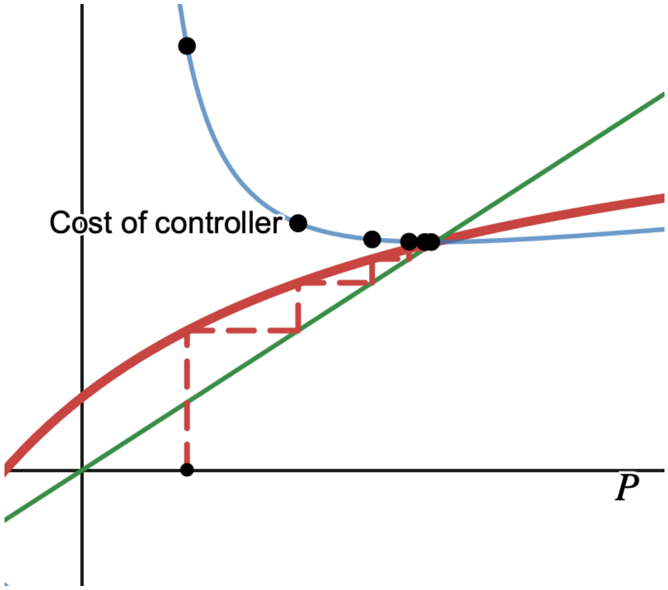
Linear controller...but what is P?

Plug u back into the Bellman equation to find:

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

A fixed point in P!

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



Desmos [link]

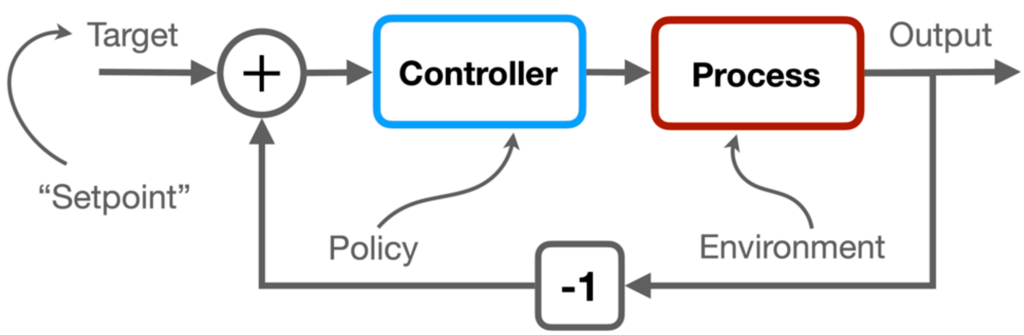
Recap lessons from RL section

- Interplay between reward design and “shortsighted” vs. “farsighted” objectives
- Fixed points:
  - Principled theoretical target
  - Practical approximations in value/policy space
  - Tractable in simple scalar case (+ visualizations)

Process control

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…



Linear policies

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)

"Input"

"Output"

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

Feedback control setup

- $y_{sp} =$ desired value (sp = "setpoint")
- $y =$ measured value
- $e = y_{sp} - y$

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

Proportional control

Consider the policy

$$u(t) = k_p e(t)$$

Proportional-Integral control

Consider the policy

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

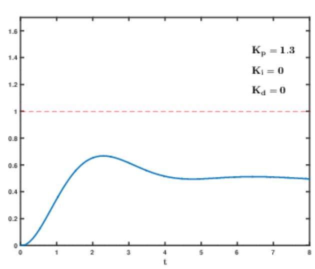
The magic of integral action

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

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

4. The integral term must have a finite limit
5. Zero offset! ($e(t) \to 0$)

Proportional-Integral-Derivative control



Proportional: Go towards the setpoint

Integral: Stay at the setpoint

Derivative: Don't overshoot the setpoint

PID controller (Wikipedia)

PID summary

Pros

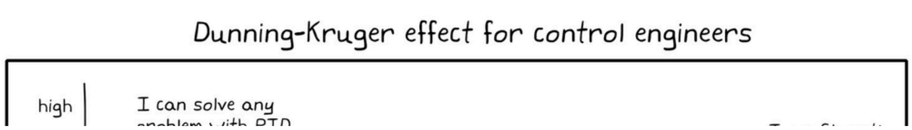
+ "Simple" structure
+ Widely used
+ Stable, robust, offset-free tracking

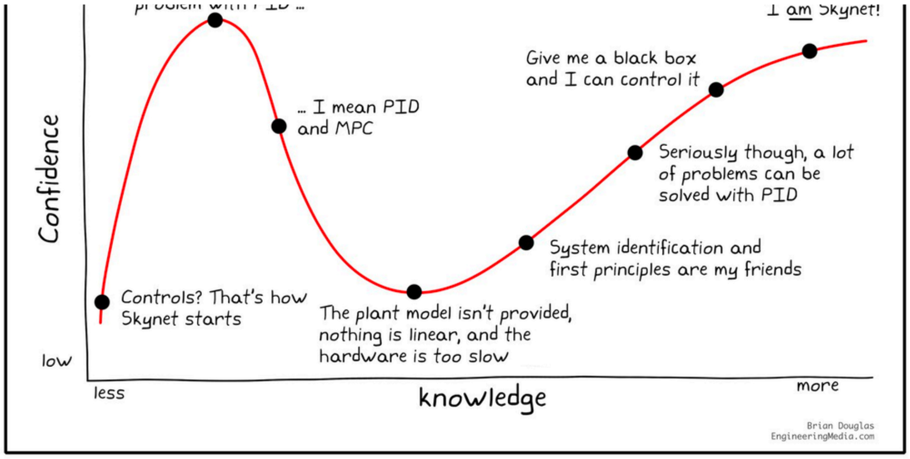
Cons

- "Simple" systems
- Can be difficult to tune
- Awkward in the face of constraints

See addendum for details/experiments dealing with multiloop PID

Break





LQR

Scalable design

$$\begin{array}{ll} \min\limits_{u_0,u_1,\ldots} & \sum\limits_{t=0}^{\infty} \gamma^t \left(x_t^T M x_t + u_t^T R u_t \right) \\ \text{where} & x_{t+1} = A x_t + B u_t \end{array}$$

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

We already solved this in the scalar case!

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

General solution

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

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

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

Example is in the Julia package ControlSystems.jl. Other options include Matlab's Control System Toolbox, or Python Control Systems

```
using ControlSystems

Pd = c2d(ss(P),ts)
```

Library.

```
A, B = Pd.A, Pd.B
M, R = I, I

K = lqr(Discrete, A, B, M, R)

u(x,t)  = -K*x
```

Aside about discounting

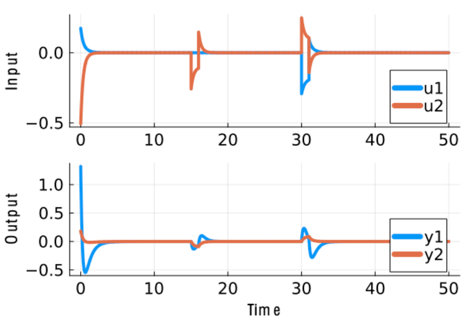
Standard LQR solvers: $(A, B, M, R) \to K$

Discounted LQR: Use

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

As $\gamma \to 0$...

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



- 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

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

Globally optimal?

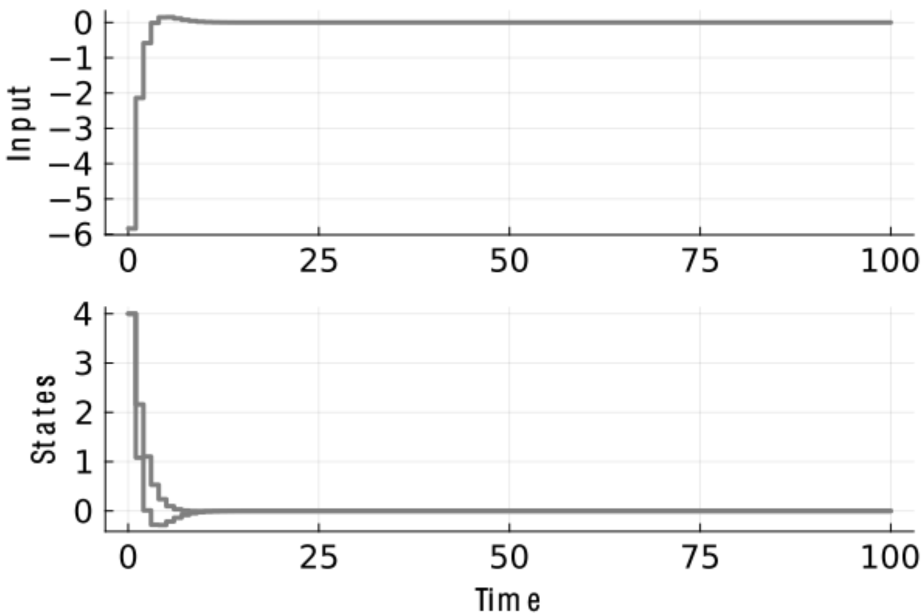
Consider the system

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

Unstable $\rightarrow$ control is needed to achieve equilibrium

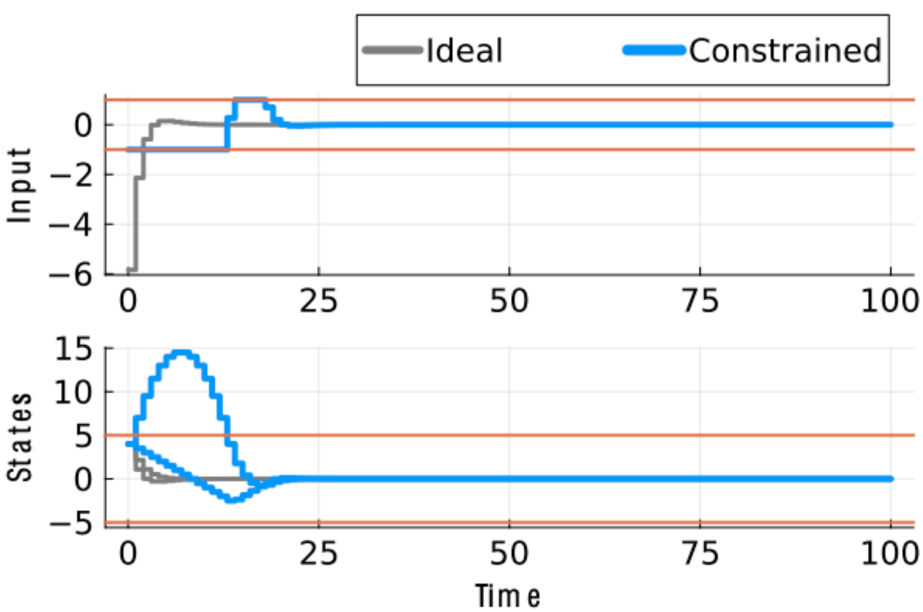
Globally optimal?

LQR, business as usual



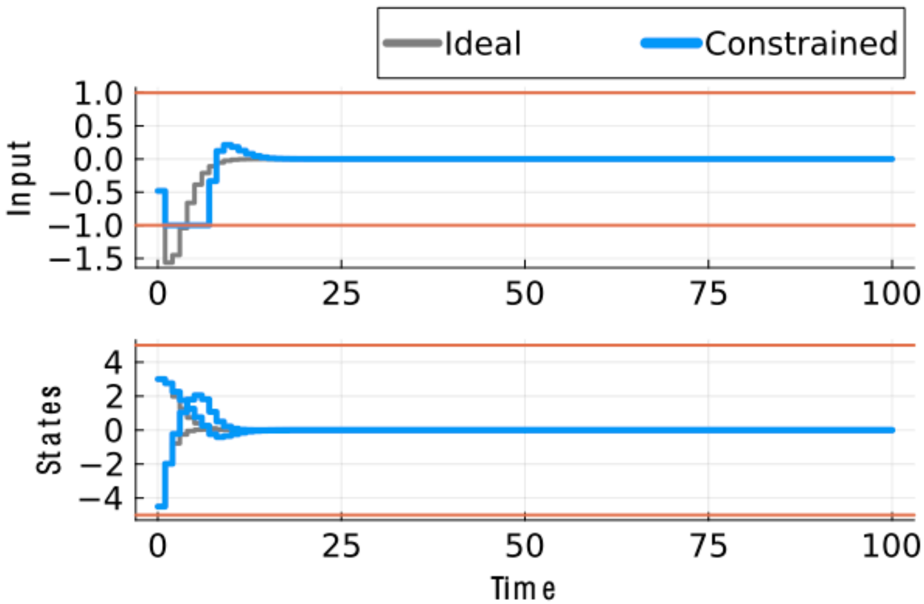
Globally optimal?

Suppose only actions in the set $\{u : \|u\|_\infty \leq 1\}$ are possible
And we want the states to stay in $\{x : \|x\|_\infty \leq 5\}$

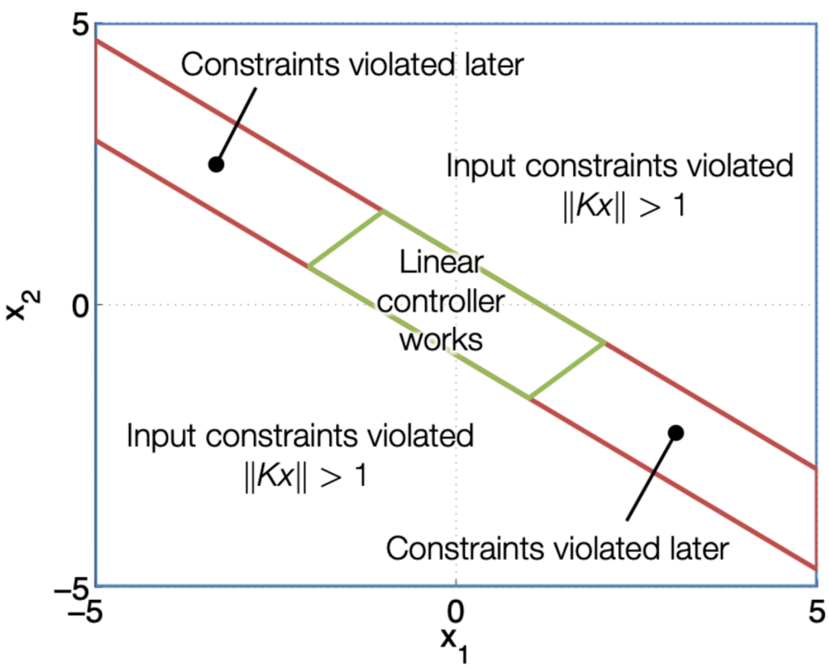


Globally optimal?

The actions can start out feasible, then become infeasible later



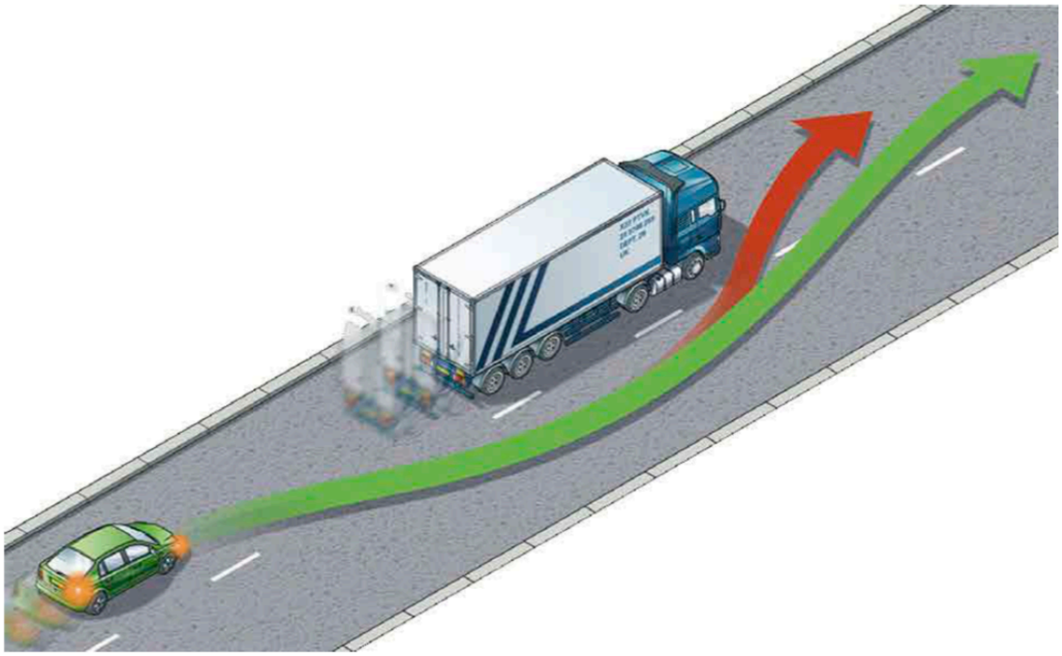
Locally optimal



See Borrelli, Bemporad, and Morari (2017) or slides

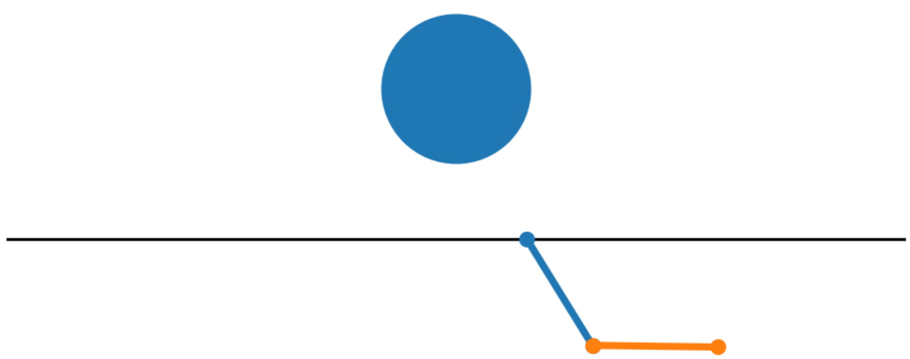
Nonlinear policies

What if a controller could anticipate an obstacle?



From this...

To this...



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

Model predictive control

Problem formulation

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

$$\begin{aligned} \min_{u_0, u_1, \ldots u_{N-1}} \quad & \sum_{t=0}^{N-1} x_t^T M x_t + u_t^T R u_t \\ \text{where} \quad & x_{t+1} = Ax_t + Bu_t \\ & x_t \in \mathcal{X} \\ & u_t \in \mathcal{U} \end{aligned}$$

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

Problem formulation

Applying the optimal inputs $u_0^\star, u_1^\star, \ldots$ 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

The receding horizon idea

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

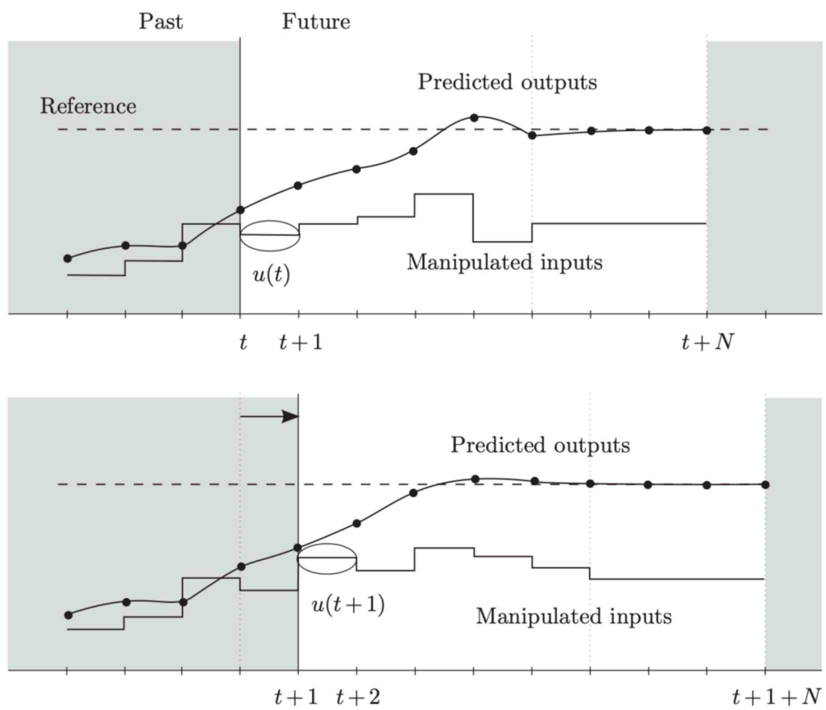
$$\begin{aligned} \min_{u_0, u_1, \ldots u_{N-1}} \quad & \sum_{k=0}^{N-1} x_k^T M x_k + u^T R u_k \\ \text{where} \quad & x_0 = s_t \\ & x_{k+1} = Ax_k + Bu_k \\ & x_k \in \mathcal{X} \\ & u_k \in \mathcal{U} \end{aligned}$$

The receding horizon idea

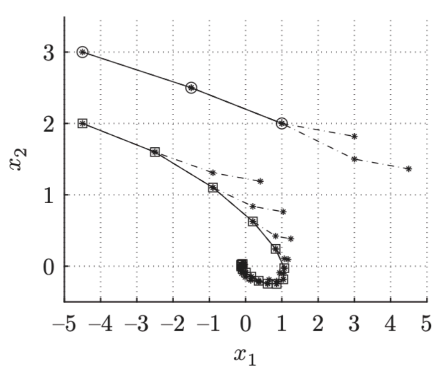
MPC controller

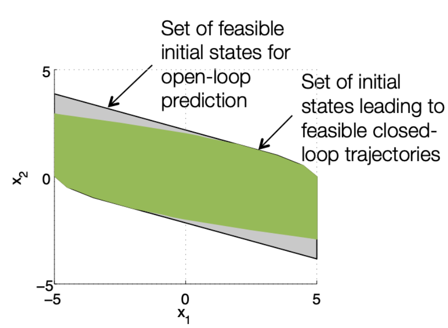
1. Initialize state $x_0 = s$
2. Solve the MPC optimization problem
3. Apply $u_0^\star$ to the system
4. Update system state $s \leftarrow s'$

The receding horizon idea



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



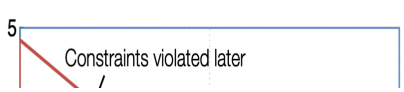


Solid - realized closed-loop trajectories

Dash - predicted trajectory

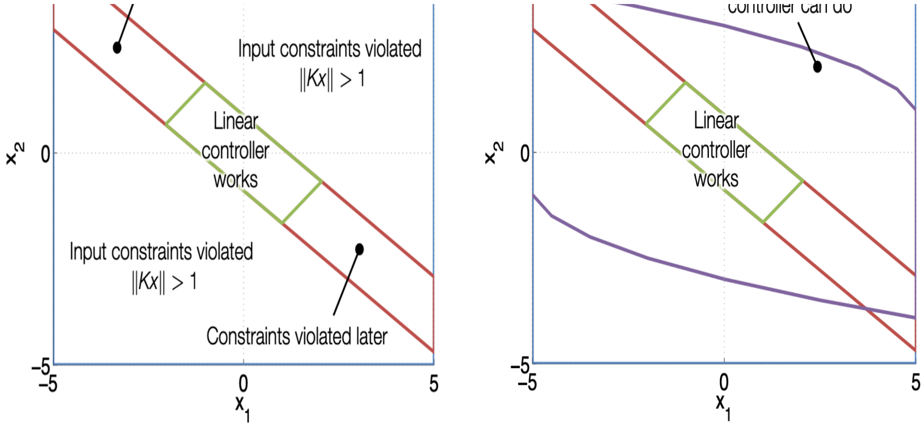
See Borrelli, Bemporad, and Morari (2017), Chapter 12

LQR



MPC

Check addendum to see why MPC is a nonlinear controller



Other Formats
RevealJS
PDF

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)

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

- Infeasibility
- Instability

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

1. Terminal constraint
2. Terminal cost

$$\begin{aligned}
\min_{u_0, u_1, \ldots u_{N-1}} \quad & \sum_{k=0}^{N-1} \left(x_k^T M x_k + u_k^T R u_k \right) + \; x_N^T P x_N \\
\text{where} \quad & x_0 = s_t \\
& x_{k+1} = A x_k + B u_k \\
& x_k \in \mathcal{X}, u_k \in \mathcal{U} \\
& x_N \in \mathcal{X}_f
\end{aligned}$$

How to design terminal cost?

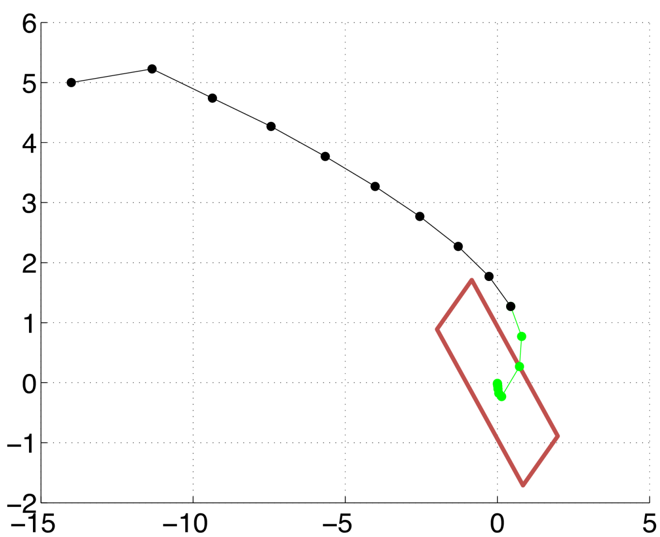
LQR!

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

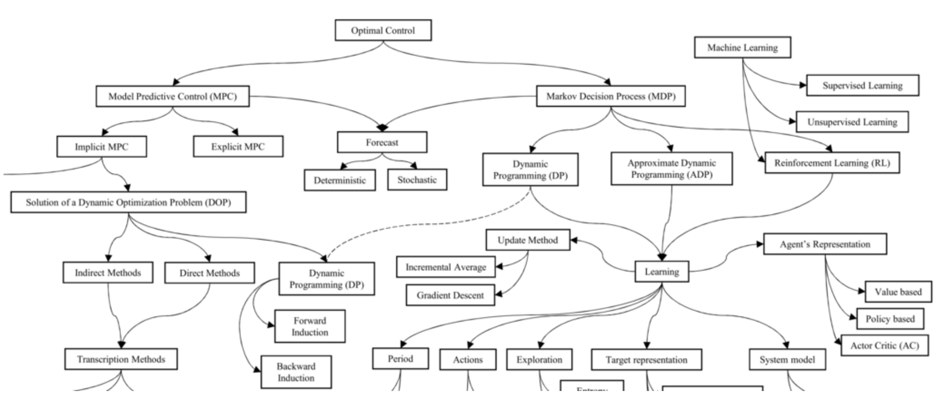
Final objective

$$
\begin{aligned}
\min_{u_0, u_1, \dots u_{N_c-1}} \quad & \sum_{k=0}^{N-1} \left(x_k^T M x_k + u_k^T R u_k\right) + x_N^T P x_N \\
\text{where} \quad & x_0 = s_t \\
& x_{k+1} = A x_k + B u_k \\
& x_k \in \mathcal{X}, \quad u_k \in \mathcal{U} \\
& u_k = -K x_k, \quad N_c \le k < N
\end{aligned}
$$

Mimic infinite horizon behavior



MPC + RL



Arroyo et al. (2022)

Main motivation

MPC

+ Safety by design
+ Modularity

- Manual design
- Rigid

RL

+ Model-free
+ Flexible objectives

- Safety constraints
- Slowish learning

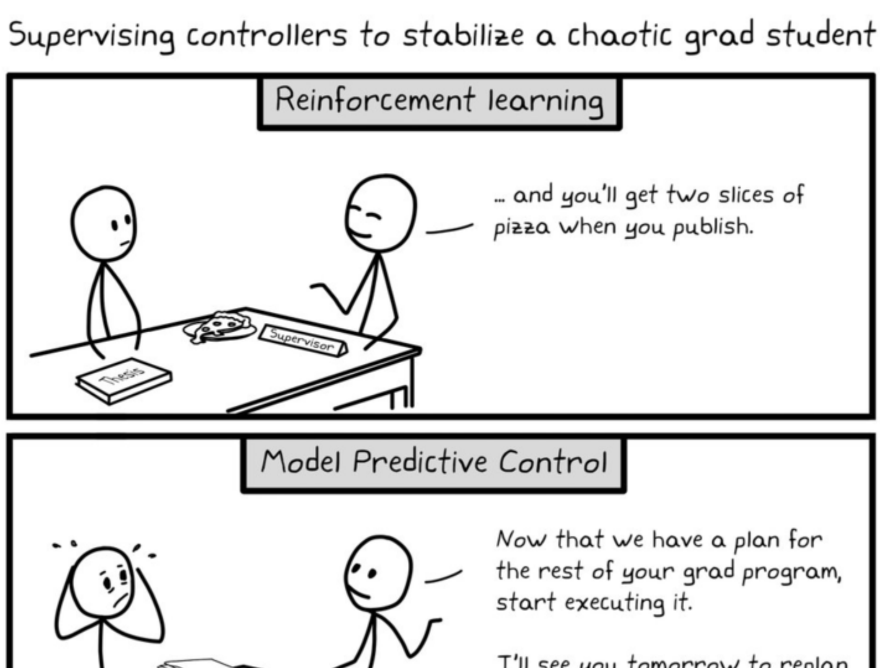
MPC + value function

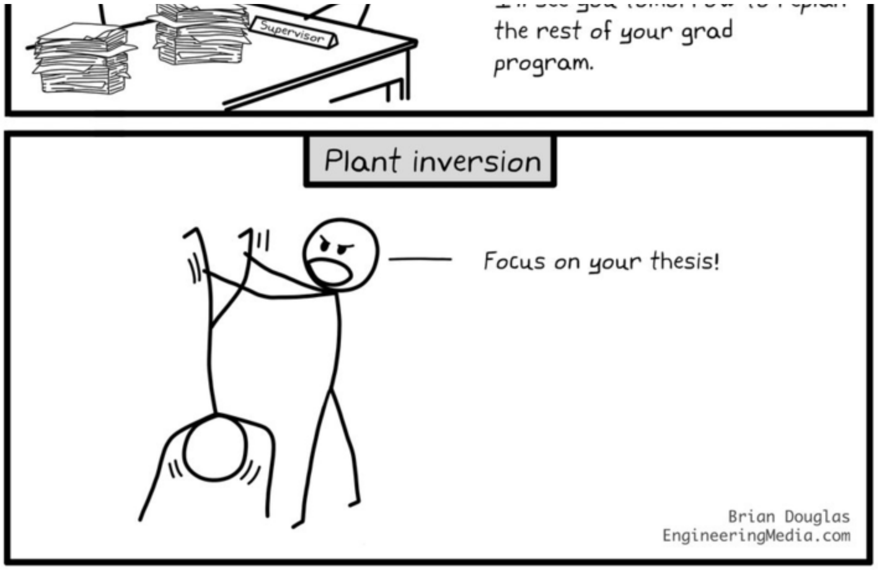
$$
\begin{aligned}
\min_{u_0, u_1, \ldots u_{N-1}} \quad & \sum_{k=0}^{N-1} \ell(x_k, u_k) \quad + \quad \overbrace{V_\theta(x_N)}^{\text{Learnable residual}} \\
\text{where} \quad & x_0 = s_t \\
& x_{k+1} = f(x_k, u_k) \\
& \underbrace{x_k \in \mathcal{X}, \quad u_k \in \mathcal{U}}_{\text{Prior engineering}}
\end{aligned}
$$

A special case

$$
\begin{aligned}
\min_{u_0} \quad & \ell(x_0, u_0) + V_\theta(x_1) \\
\text{where} \quad & x_0 = s_t \\
& x_1 = f(x_0, u_0) \\
& x_1 \in \mathcal{X}, \quad u_0 \in \mathcal{U}
\end{aligned}
$$

Break





Implementation

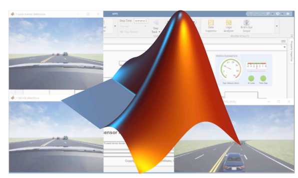
MPC frameworks

OpEn

Sopasakis, Fresk, and Patrinos (2020)

robust optimal control toolbox

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]

MPC frameworks

[do-mpc]:

- Open source
- Modular
- Python interface
- Fast

robust optimal control toolbox

CasADi

Fiedler et al. (2023)

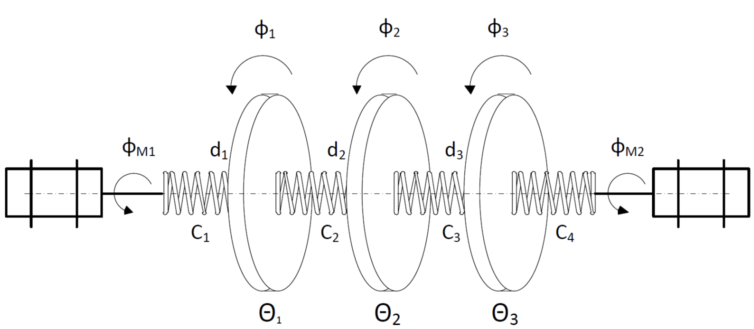
Andersson et al. (2019)

Example: Triple mass spring system

See notebook from do-mpc for full code samples. We only show snippets of the key destinations.

Create model

```
import do_mpc
from casadi import *

model_type = 'continuous' # either 'discrete' or 'continuou
model = do_mpc.model.Model(model_type)

dphi = model.set_variable(var_type='_x', var_name='dphi', s
# Two states for the desired (set) motor position:
phi_m_1_set = model.set_variable(var_type='_u', var_name='p
phi_m_2_set = model.set_variable(var_type='_u', var_name='p
```



Right-hand-side equation

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

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

```
Theta_1 = model.set_variable('parameter', 'Theta_1')
Theta_2 = model.set_variable('parameter', 'Theta_2')
Theta_3 = model.set_variable('parameter', 'Theta_3')

c, d = np.array([2.697,  2.66,  3.05, 2.86])*1e-3, np.array
dphi_next = vertcat(
```

```
    -c[0]/Theta_1*(phi_1-phi_1_m)-c[1]/Theta_1*(phi_1-phi_2
    -c[1]/Theta_2*(phi_2-phi_1)-c[2]/Theta_2*(phi_2-phi_3)-
    -c[2]/Theta_3*(phi_3-phi_2)-c[3]/Theta_3*(phi_3-phi_2_r
)

model.set_rhs('dphi', dphi_next)
model.setup()
```

Create controller

```
mpc = do_mpc.controller.MPC(model)

setup_mpc = {
    'n_horizon': 20,
    't_step': 0.1,
    'n_robust': 1,

    'store_full_solution': True,
}
mpc.set_param(**setup_mpc)
```

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

```
mpc.setup()
```

Define simulator

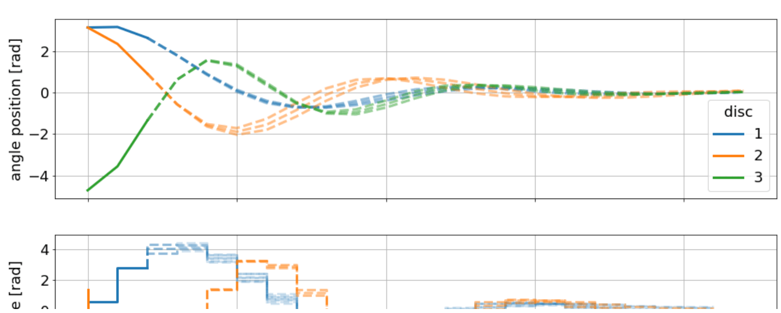
Either use the same model inside the MPC or define a different model to simulate the “true” system:

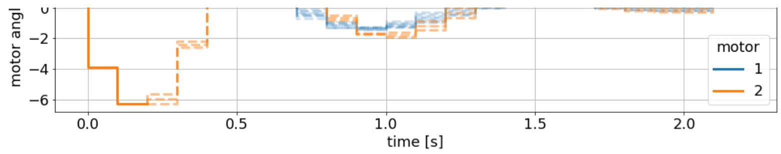
- Simplified MPC model
- Complex “true” simulator model

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

Run the control loop

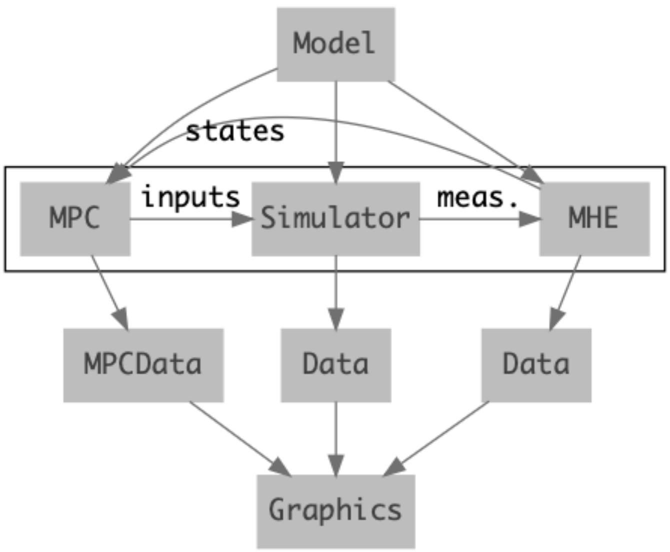
```
for i in range(20):
    u0 = mpc.make_step(x0)
    x0 = simulator.make_step(u0)
```



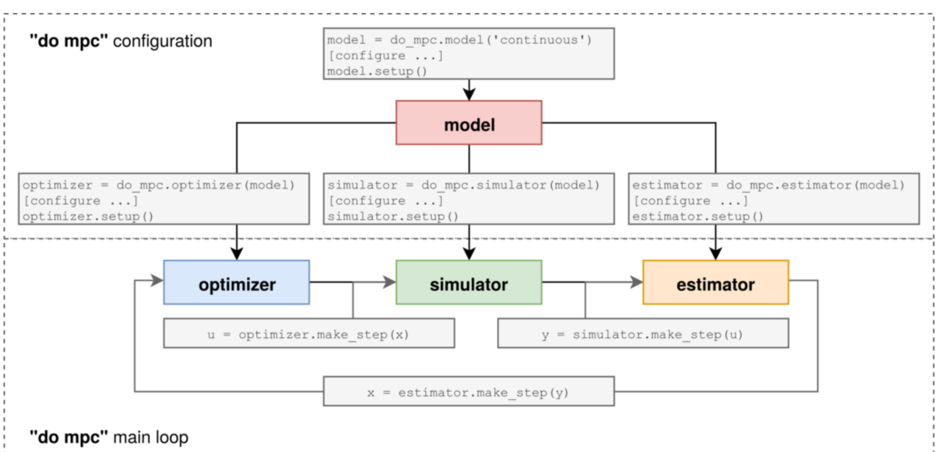


Creating these gifs is easy with do-mpc's `Graphics` and `Data` modules

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



do-mpc summary



RL frameworks

Raffin et al. (2021)

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

Huang et al. (2022)

OpenAI Spinning Up

Spinning Up

acme

Hoffman et al. (2022)

TorchRL

Bou et al. (2023)

Tianshou

J. Weng et al. (2022)

RLlib

Liang et al. (2018)

(...A really long list here)

RL frameworks

CleanRL:

- Self-contained implementations
- Rapid prototyping
- Thorough documentation and benchmarking
- `Gym` for environments and `wandb` for tracking

Gymnasium

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

Weights & Biases

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

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

Combining RL and MPC

Recall what we're after:

$$\begin{aligned} \min_{u_0, u_1, \ldots u_{N-1}} \quad & \sum_{k=0}^{N-1} \ell(x_k, u_k) \quad + \quad \overbrace{V_\theta(x_N)}^{\text{Learnable residual}} \\ \text{where} \quad & x_0 = s_t \\ & x_{k+1} = f(x_k, u_k) \\ & \underbrace{x_k \in \mathcal{X}, \quad u_k \in \mathcal{U}}_{\text{Prior engineering}} \end{aligned}$$

Combining RL and MPC

CleanRL:

1. Value function learning
2. Environment implementation

do-mpc:

1. Optimization module
2. Simulation

1. RL value function in do-mpc

Value function training

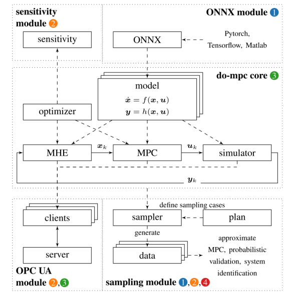
```
data = rb.sample(args.batch_size)
with torch.no_grad():
    next_state_actions, next_state_log_pi, _ = actor.get_ac
    qf1_next_target = qf1_target(data.next_observations, ne
    qf2_next_target = qf2_target(data.next_observations, ne
    min_qf_next_target = torch.min(qf1_next_target, qf2_nex
    next_q_value = (data.rewards.flatten() +
        (1 - data.dones.flatten()) * args.gamma * (min_qf_n

qf1_a_values = qf1(data.observations, data.actions).view(-1
qf2_a_values = qf2(data.observations, data.actions).view(-1
qf1_loss = F.mse_loss(qf1_a_values, next_q_value)
qf2_loss = F.mse_loss(qf2_a_values, next_q_value)
qf_loss = qf1_loss + qf2_loss

# optimize the model
q_optimizer.zero_grad()
qf_loss.backward()
q_optimizer.step()
```

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()`

ONNX



Fiedler et al. (2023)

1. RL value function in do-mpc

Implement MPC with terminal value function:

```
mpc = do_mpc.controller.MPC(model)

mpc.settings.n_horizon = 1
lterm = model.aux['cost']

terminal_converter = do_mpc.sysid.ONNXConversion(value_onnx
def terminal_casadi(x):
    terminal_converter.convert(x = x.T, goal=np.zeros(x.T.s
    return terminal_converter['terminal_cost']
mterm = -terminal_casadi(model.x['x'])

mpc.set_objective(lterm=lterm, mterm=mterm)

mpc.bounds['lower','_u','u'] = -0.5
mpc.bounds['upper','_u','u'] =  0.5

mpc.setup()
```

1. RL value function in do-mpc

Run MPC + value function controller in CleanRL:

```
x0 = estimator.make_step(obs["observation"])
action = mpc.make_step(x0)

# A quick way of incorporating exploration
with torch.no_grad():
    if global_step % args.policy_frequency == 0: # optiona
        noise = actor._explore_noise(obs).numpy()
        action += noise
        action = np.float32(action.clip(env.action_space.lo

next_ob, reward, done, info = env.step(action)
```

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

2. Gym wrapper for do-mpc simulation

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

```
import gymnasium as gym

env = gym.make("LunarLander-v2", render_mode="human")
observation, info = env.reset()

for _ in range(1000):
    action = env.action_space.sample()  # agent policy that
    observation, reward, terminated, truncated, info = env.

    if terminated or truncated:
        observation, info = env.reset()

env.close()
```

Environments follow a basic blueprint:

```
from gymnasium import spaces

class CustomEnv(gym.Env):
  def __init__(self, arg1, arg2, ...):
    super(CustomEnv, self).__init__()
    self.action_space = spaces.Discrete(N_DISCRETE_ACTIONS)
    self.observation_space = spaces.Box(low=0, high=255, sh

  def step(self, action):
    ...
    return observation, reward, done, info

  def reset(self):
    ...
    return observation
  def render(self, mode='human'):
    ...
  def close (self):
    ...
```

Create Gym environment that queries do-mpc simulation:

```
class DoMPCEnv(gym.Env):
    """
    Gym environment that uses do-mpc for carrying out simul
    """
    def __init__(self, simulator:do_mpc.simulator.Simulator
                 num_steps=100):
        super().__init__()
        ...
```

```python
    def step(self, action):
        # simplified version --- hides some processing step

        self.t += 1
        self.state = self.simulator.make_step(action)
        info = self._get_info()
        reward, terminated, truncated = info["reward"], inf

        return self.state, reward, terminated, truncated, i
    ...
```

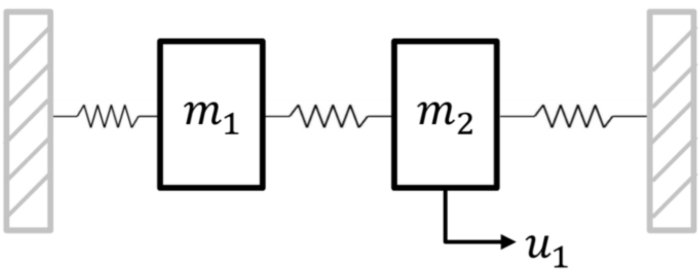
Example: Oscillating masses

State: Position and velocity of each mass

Action: Force applied to m_2

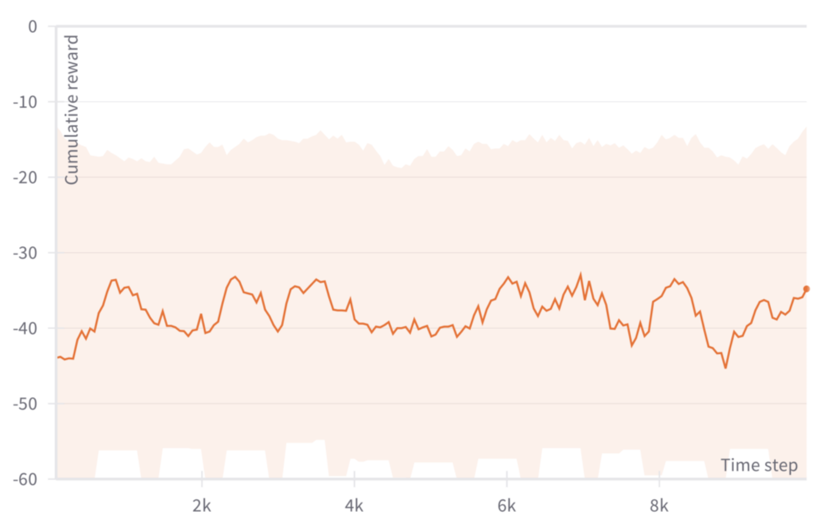
MPC cost: $\|x\|^2$

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



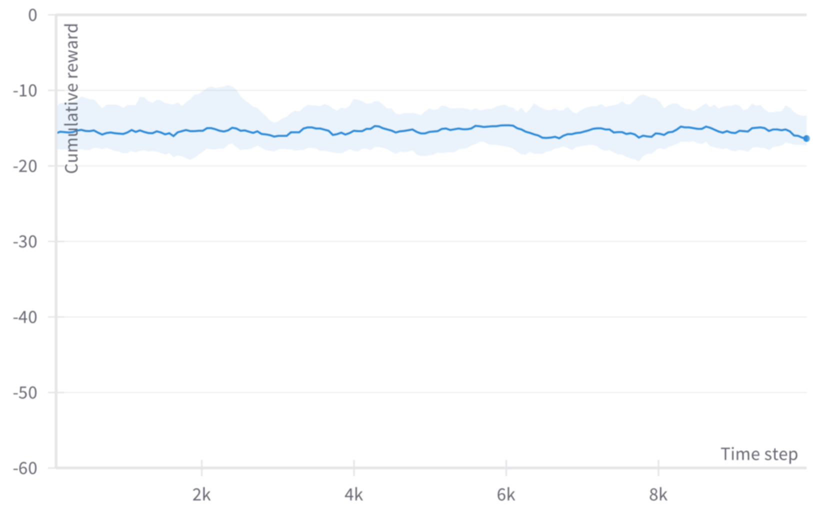
“Bad” model

- A model such that using it for MPC results in poor/inconsistent rewards
- Prediction horizon $N = 10$

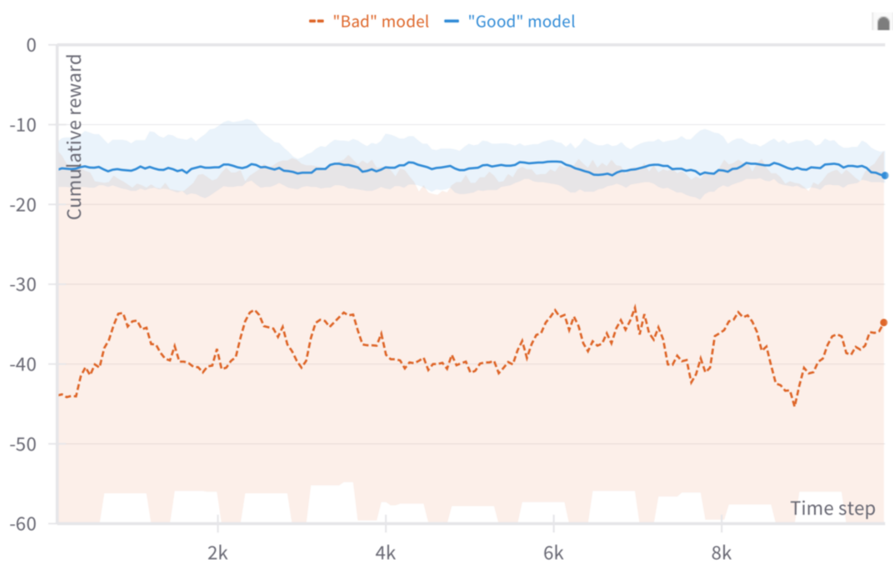


“Good” model

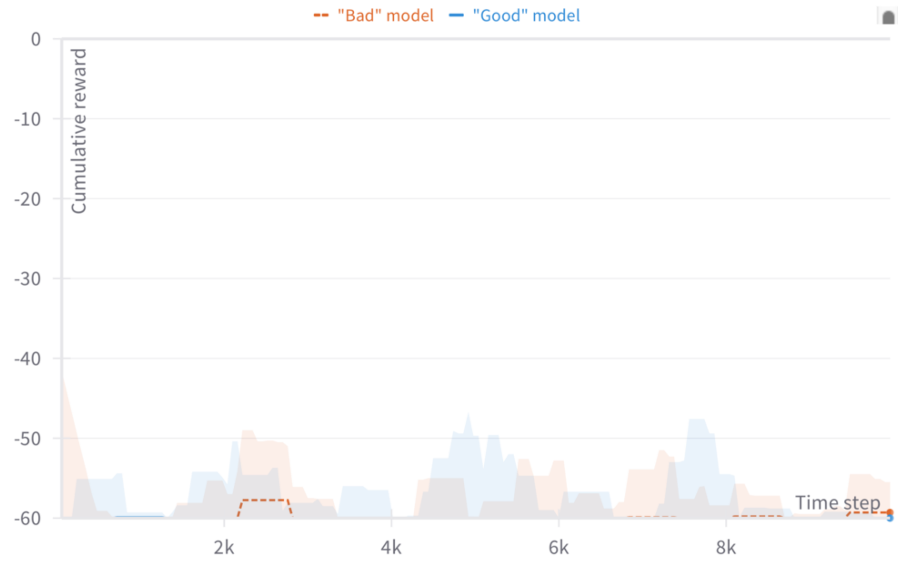
- Well, the MPC policy performs well



N=10

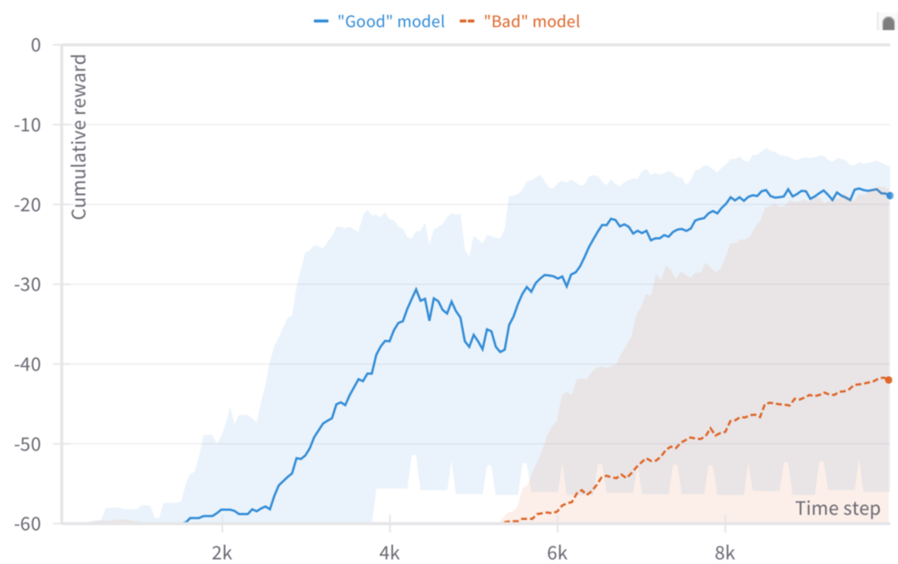


N=1



RL + MPC, nominal result

$N = 1$ plus learned terminal value function



Summary so far

Pros

+ Augmented MPC improves over time
+ Somewhat overcomes deficiencies of a “bad” model or short prediction horizon

Cons

- Learning is slow/delayed
- Significant variation

Sparse rewards

We gave the agent a binary reward

Why is that a challenge for the agent? Pros/cons of such a reward?

Pros

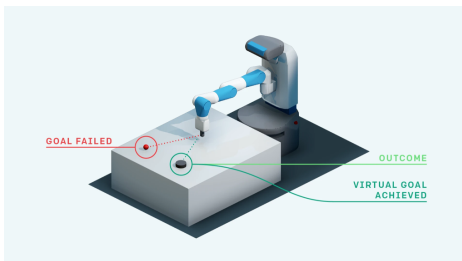
+ Simple to define
+ Many ways of completing the task

Cons

- Degenerate datasets
- Wasted exploration

Hindsight experience replay

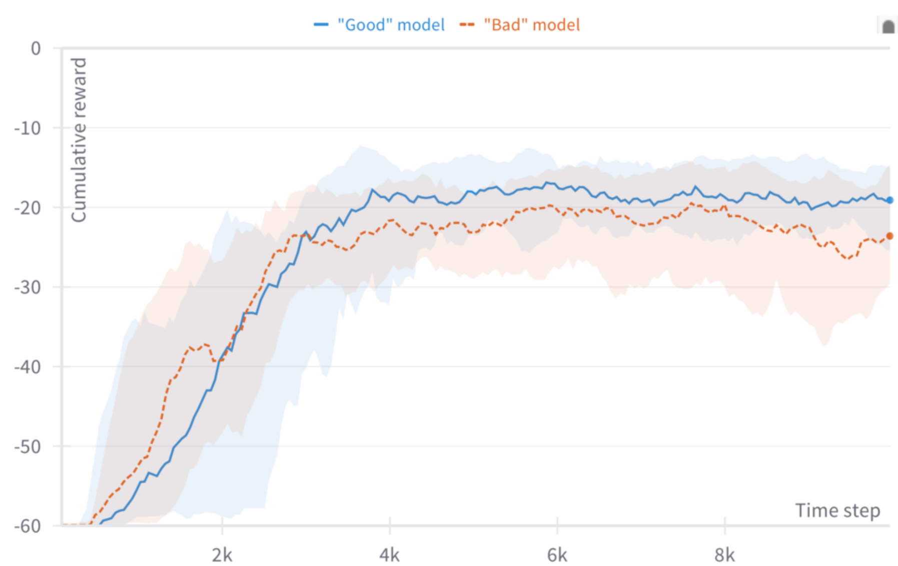
- Intuition: Failure is informative
- Idea: Relabel trajectories
    - Pretend end-state is goal-state
    - Rebalance the data
    - Learn from failure



See Andrychowicz et al. (2017) and blog post

RL + MPC with reward relabeling

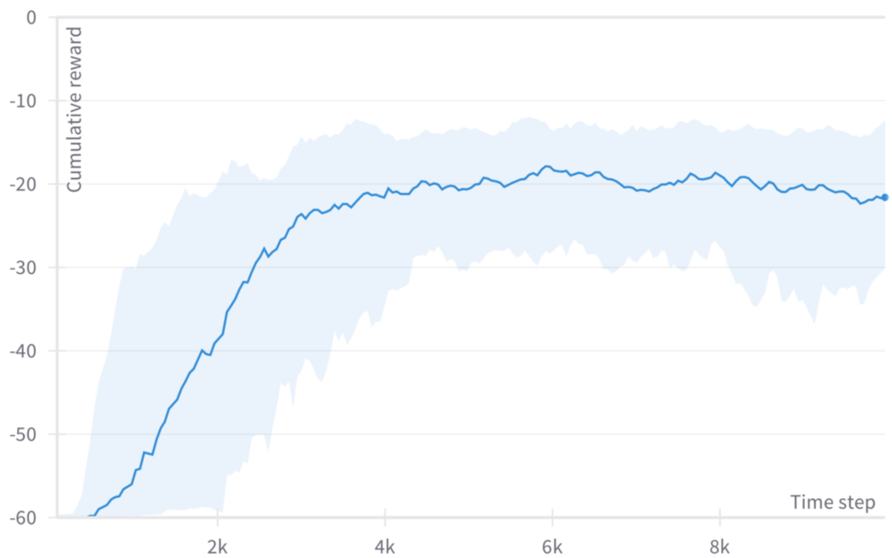
$N = 1$ plus learned terminal value function



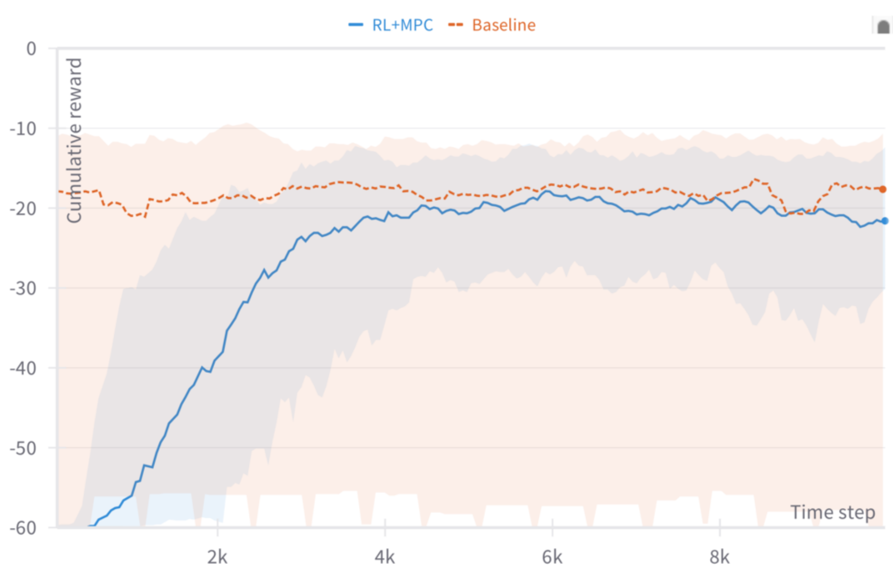
Combine the results

Combine the results

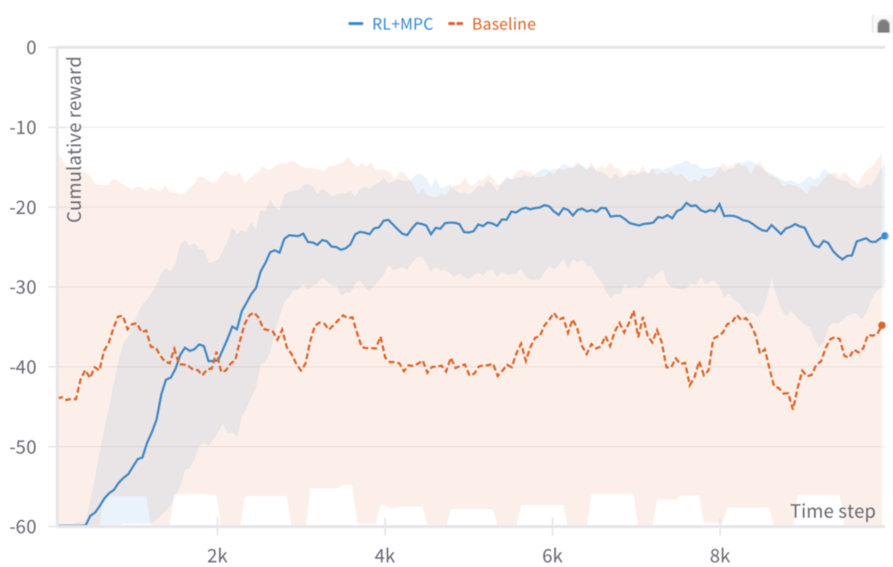
Represents a continuum of model fidelities



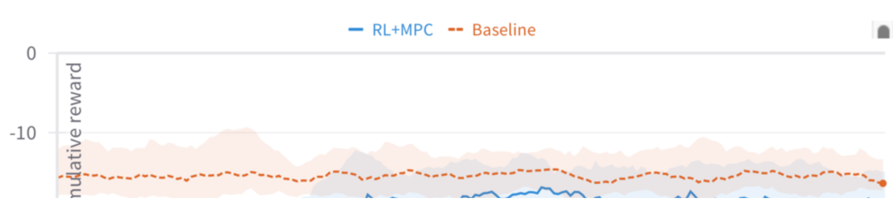
Baseline = combined MPC results with “good”/“bad” model

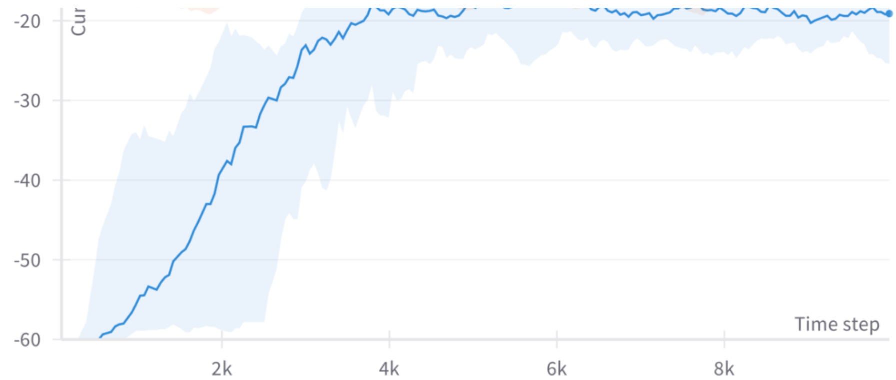


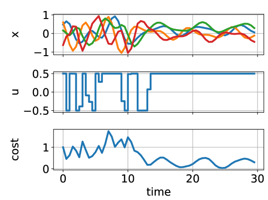
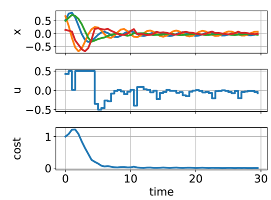
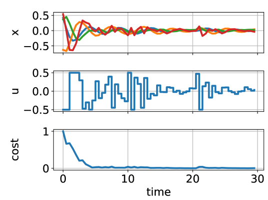
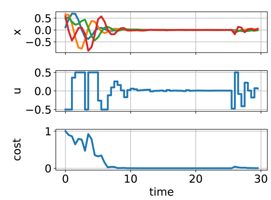
More practically, RL + MPC with a poor model and short prediction horizon compensates for nominal MPC with a poor model



A “good” model of course improves both



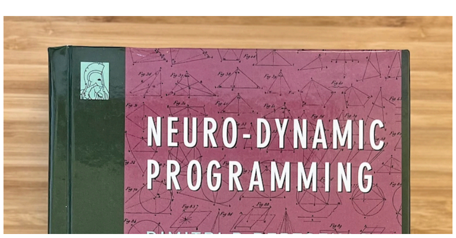


	"Bad" model	"Good" model
No relabeling		
Relabeling		

Summary

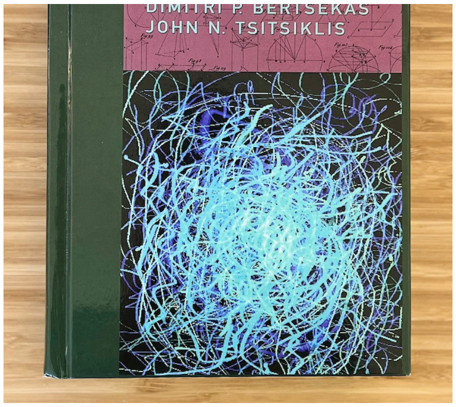
- Adding an RL element to MPC can:
  - Compensate for a "bad" model
  - Incorporate high-level objectives
- Adding an MPC element to RL can:
  - Enable safer operations
  - Improve sample complexity (pretraining—not tested here, but the obvious thing to do)

1996



2022

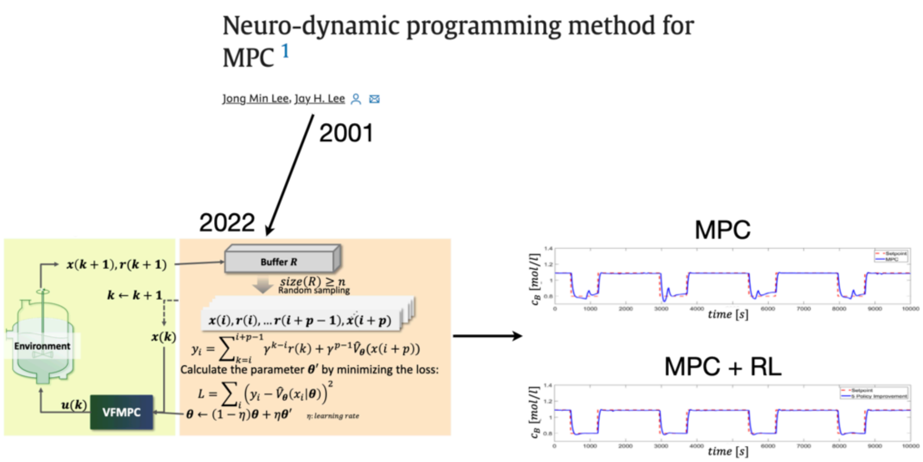
See Bertsekas and Tsitsiklis (1996), Bertsekas (2022)



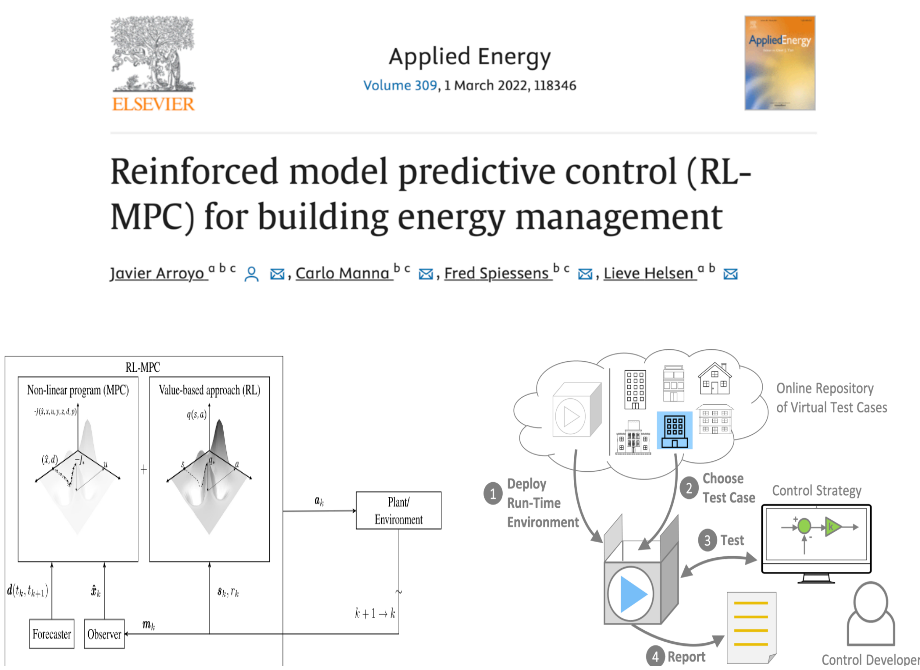
Classic, rigorous

Contemporary, intuitive

See Lee and Lee (2001), Yoo (2022)



See Arroyo et al. (2022) and BOPEST



This is a popular idea

Value Function Approximation and Model Predictive Control

Mingyuan Zhong*, Mikala Johnson*, Yuval Tassa†, Tom Erez† and Emanuel Todorov*

(2013)

Plan Online, Learn Offline: Efficient Learning and Exploration via Model-Based Control

Kendall Lowrey*[1] Aravind Rajeswaran*[1] Sham Kakade[1] Emanuel Todorov[1,2] Igor Mordatch[3]

(2019)

Learning Lyapunov terminal costs from data for complexity reduction in nonlinear model predictive control

Shokhjakhon Abdufattokhov | Mario Zanon | Alberto Bemporad

Last week (2024)

Deep Value Model Predictive Control

*Farbod Farshidian[1], *David Hoeller[1,2], Marco Hutter[1]

(2019)

Practical Reinforcement Learning For MPC: Learning from sparse objectives in under an hour on a real robot

Napat Karnchanachari [1 2 *] PATK@ETHZ.CH
Miguel I. Valls [1 *] MIGUEL.DELAIGLESIAVALLS@SEVENSENSE.CH
David Hoeller [2 3] DHOELLER@ETHZ.CH
Marco Hutter [2] MAHUTTER@ETHZ.CH

(2020)

(See also Bhardwaj, Choudhury, and Boots (2020))

Thank you

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

Slides and code

Link: https://github.com/NPLawrence/RL-MPC-tutorial

Experiment data

- Scalar LQR plotting tool on Desmos
- Experiments on wandb:
  - SAC (Acrobot)
  - DQN (Cartpole & Acrobot)
  - RL+MPC

References

Abdufattokhov, Shokhjakhon, Mario Zanon, and Alberto Bemporad. 2024. “Learning Lyapunov Terminal Costs from Data for Complexity Reduction in Nonlinear Model Predictive Control.” *International Journal of Robust and Nonlinear Control*. https://doi.org/10.1002/rnc.7411.

Amos, Brandon, Ivan Dario Jimenez Rodriguez, Jacob Sacks, Byron Boots, and J. Zico Kolter. 2019. “Differentiable MPC for End-to-end Planning and Control.” arXiv. https://doi.org/10.48550/arXiv.1810.13400.

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.

Andrychowicz, Marcin, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. 2017. “Hindsight Experience Replay.” In *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/hash/453fadbd8a1a3af50a9df4df899537b5-Abstract.html.

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.

Astrom, Karl Johan, Karl Henrik Johansson, and Qing-Guo Wang. 2001. “Design of Decoupled PID Controllers for MIMO Systems.” In *Proceedings of the 2001 American Control Conference.(Cat. No. 01CH37148)*, 3:2015–20. IEEE.

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. 2022. *Lessons from AlphaZero for Optimal, Model Predictive, and Adaptive Control*. Athena Scientific.

and Adaptive Control. Athena Scientific.

———. 2023. *A Course in Reinforcement Learning*. Athena Scientific.

Bertsekas, Dimitri, and John N Tsitsiklis. 1996. *Neuro-Dynamic Programming*. Athena Scientific.

Bhardwaj, Mohak, Sanjiban Choudhury, and Byron Boots. 2020. "Blending MPC & Value Function Approximation for Efficient Reinforcement Learning." *arXiv:2012.05909 [Cs]*. http://arxiv.org/abs/2012.05909.

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.

Elnawawi, Shams, Lim C. Siang, Daniel L. O'Connor, and R. Bhushan Gopaluni. 2022. "Interactive Visualization for Diagnosis of Industrial Model Predictive Controllers with Steady-State Optimizers." *Control Engineering Practice* 121: 105056. https://doi.org/10.1016/j.conengprac.2021.105056.

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.

Farshidian, Farbod, David Hoeller, and Marco Hutter. 2019. "Deep Value Model Predictive Control." arXiv. http://arxiv.org/abs/1910.03358.

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.

Forbes, Michael G., Rohit S. Patwardhan, Hamza Hamadah, and R. Bhushan Gopaluni. 2015. "Model Predictive Control in Industry: Challenges and Opportunities." *IFAC-PapersOnLine* 48 (8): 531–38. https://doi.org/10.1016/j.ifacol.2015.09.022.

Fujimoto, Scott, Herke van Hoof, and David Meger. 2018. "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.

Gros, Sebastien, and Mario Zanon. 2020. "Data-Driven Economic NMPC Using Reinforcement Learning." *IEEE Transactions on Automatic Control* 65 (2): 636–48. https://doi.org/10.1109/TAC.2019.2913768.

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

Nikola Momchev, Danila Sinopalnikov, Piotr Stanczyk, et al. 2022. "Acme: A Research Framework for Distributed Reinforcement Learning." arXiv. https://doi.org/10.48550/arXiv.2006.00979.

Huang, Shengyi, Rousslan Fernand Julien 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.

Karnchanachari, Napat, Miguel Iglesia Valls, David Hoeller, and Marco Hutter. 2020. "Practical Reinforcement Learning for Mpc: Learning from Sparse Objectives in Under an Hour on a Real Robot." In *Learning for Dynamics and Control*, 211–24. PMLR.

Lawrence, Nathan P., Michael G. Forbes, Philip D. Loewen, Daniel G. McClement, Johan U. Backström, and R. Bhushan Gopaluni. 2022. "Deep Reinforcement Learning with Shallow Controllers: An Experimental Application to PID Tuning." *Control Engineering Practice* 121: 105046. https://doi.org/10.1016/j.conengprac.2021.105046.

Lee, Jong Min, and Jay H. Lee. 2001. "Neuro-Dynamic Programming Method for MPC." *IFAC Proceedings Volumes* 34 (25): 143–48. https://doi.org/10.1016/S1474-6670(17)33814-4.

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.

Lowrey, Kendall, Aravind Rajeswaran, Sham Kakade, Emanuel Todorov, and Igor Mordatch. 2019. "Plan Online, Learn Offline: Efficient Learning and Exploration via Model-Based Control." arXiv. https://doi.org/10.48550/arXiv.1811.01848.

Maeder, Urban, Francesco Borrelli, and Manfred Morari. 2009. "Linear Offset-Free Model Predictive Control." *Automatica* 45 (10): 2214–22. https://doi.org/10.1016/j.automatica.2009.06.005.

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.

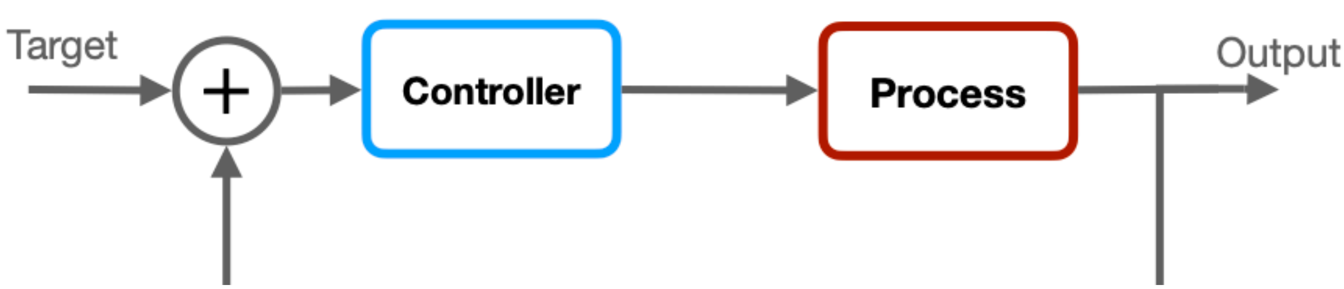
Nejatbakhsh Esfahani, Hossein, Arash Bahari Kordabad, Wenqi Cai, and Sebastien Gros. 2023. "Learning-Based State Estimation and Control Using MHE and MPC Schemes with Imperfect Models." *European Journal of Control* 73: 100880. https://doi.org/10.1016/j.ejcon.2023.100880.

Control 73: 100880. https://doi.org/10.1016/j.ejcon.2023.100880.
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/.
Williams, Ronald J. 1992. "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning." *Machine Learning* 8 (3): 229–56.
https://doi.org/10.1007/BF00992696.
Yoo, Haeun. 2022. "Strategies to Apply Reinforcement Learning for Advanced Chemical Process Control."
Zhong, Mingyuan, Mikala Johnson, Yuval Tassa, Tom Erez, and Emanuel Todorov. 2013. "Value Function Approximation and Model Predictive Control." In *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, 100–107. Singapore, Singapore: IEEE. https://doi.org/10.1109/ADPRL.2013.6614995.

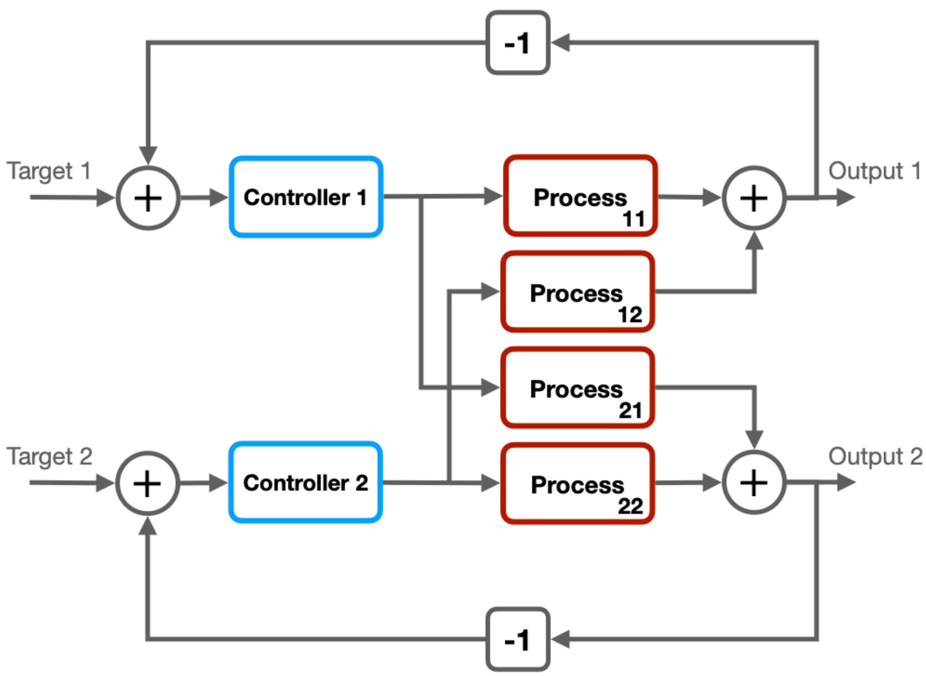
Addendum

Multiloop PID

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



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

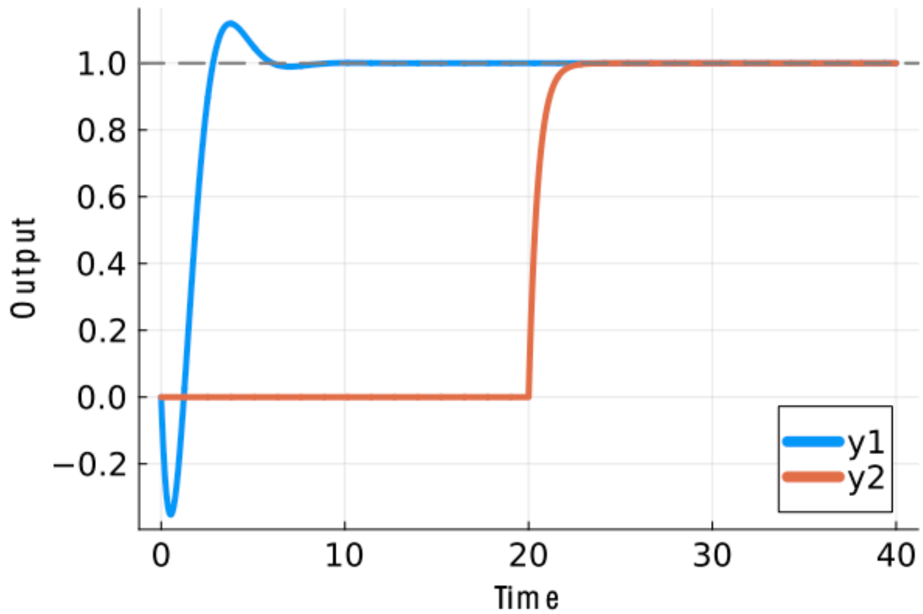


We could continue with PID, but generally:

1. Doing so won't advance our agenda here
2. There are limitations to PID:
   - PID is usually used to regulate a single variable system
   - New tuning parameters get introduced with multivariable methods
   - Nonetheless you should still know about PID!

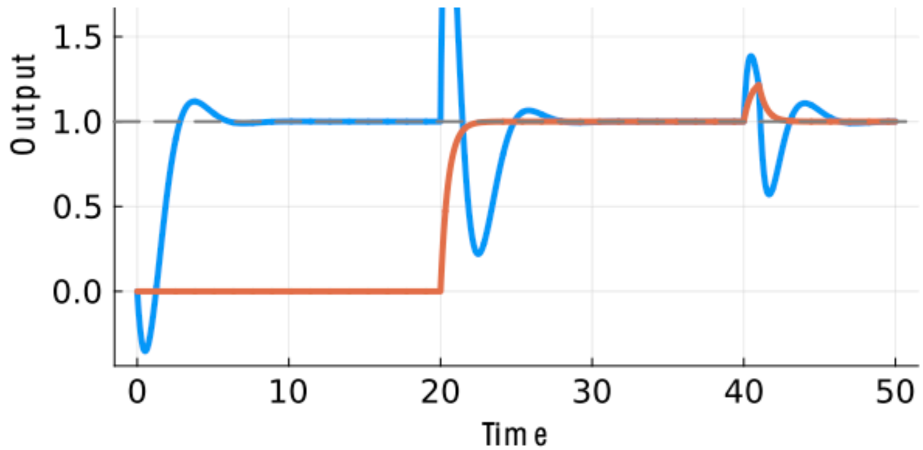
Can we tune controllers independently?

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



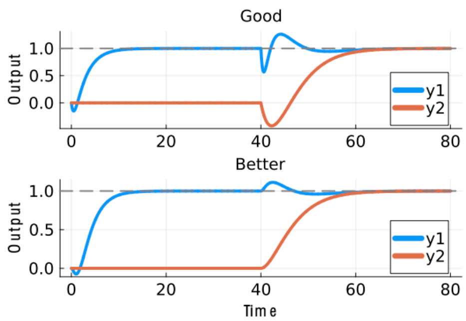
Can we tune controllers independently?

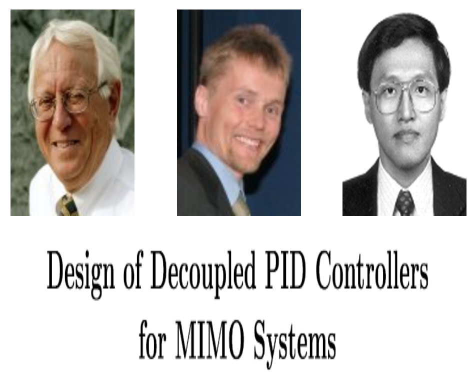
Those controllers can be disastrous if one process feeds into the other



Can we tune controllers independently?

... sort of

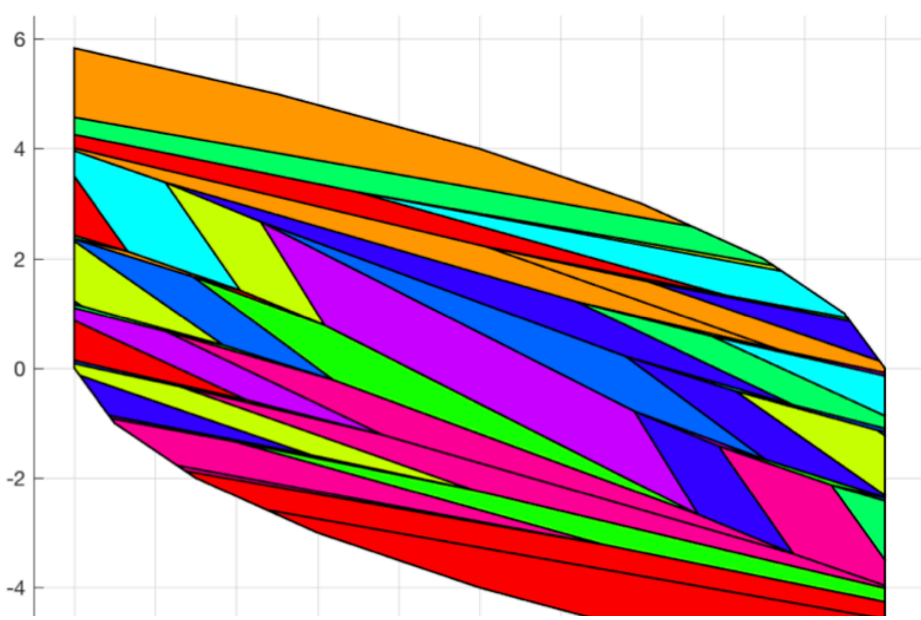


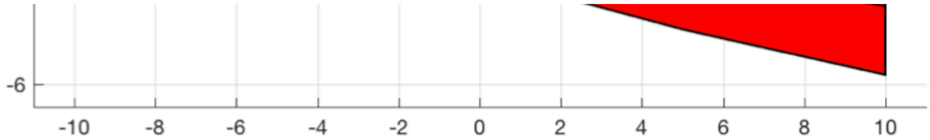


Astrom, Johansson, and Wang (2001)

We need a scalable approach to handling loop interactions!

MPC control law is nonlinear





Why?

MPC = Multi-parametric quadratic programming

Original problem:

$$\begin{aligned} \min_{u_0, u_1, \ldots u_{N-1}} \quad & \sum_{k=0}^{N-1} x_k^T M x_k + u^T R u_k \\ \text{where} \quad & x_0 = s_t \\ & x_{k+1} = A x_k + B u_k \\ & x_k \in \mathcal{X} \\ & u_k \in \mathcal{U} \end{aligned}$$

Unroll the dynamics:

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

$$x_1 = Bu_0$$

$$x_2 = Bu_1 + ABu_0$$

$$x_3 = Bu_2 + ABu_1 + A^2 Bu_0$$

$$x_n = \begin{bmatrix} B & AB & \cdots & A^{N-1}B \end{bmatrix} \begin{bmatrix} u_{N-1} \\ u_{N-2} \\ \vdots \\ u_0 \end{bmatrix}$$

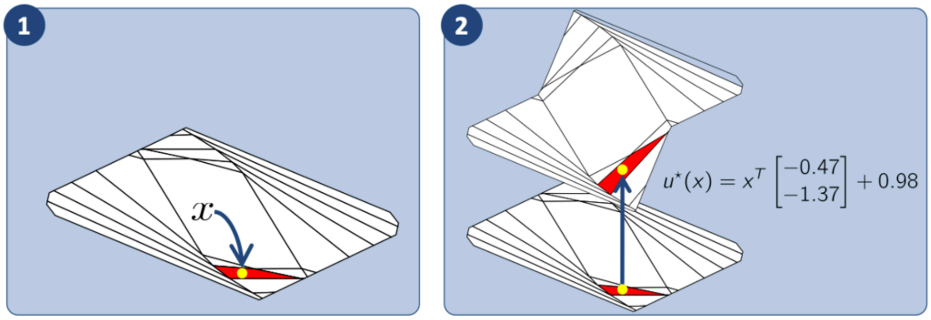
Algebra:

$$\begin{aligned} \min_{U} \quad & \frac{1}{2} U^T H U + s_t^T F U \\ \text{where} \quad & GU \leq W + E s_t \end{aligned}$$

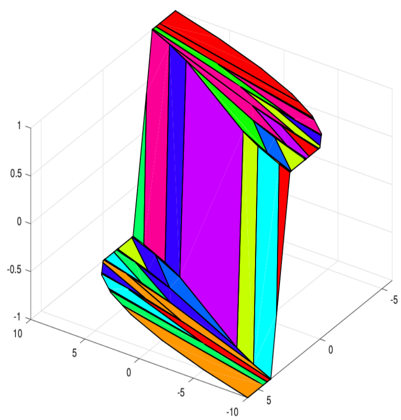
See Bemporad et al. (2002)

KKT conditions:

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

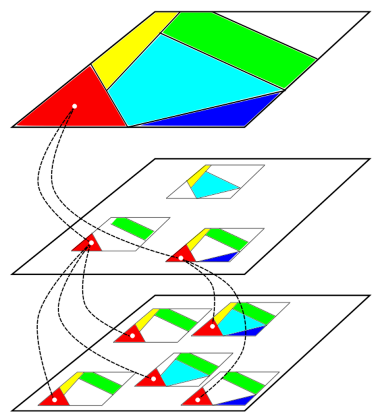


MPC = Continuous piecewise affine



MPC control law

🤝



ReLU DNN

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

Other things I ignored or didn't get to

- State estimation (Nejatbakhsh Esfahani et al. 2023)
- Differentiable MPC (Gros and Zanon 2020; Amos et al. 2019)
- Offset-free tracking (Maeder, Borrelli, and Morari 2009)
- Industrial control: theory vs practice (Forbes et al. 2015; Elnawawi et al. 2022)

Appendix

Scalar LQR general formulas

Trajectories are easy to compute:

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

Returns are easy to compute

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

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

Quadratic value function

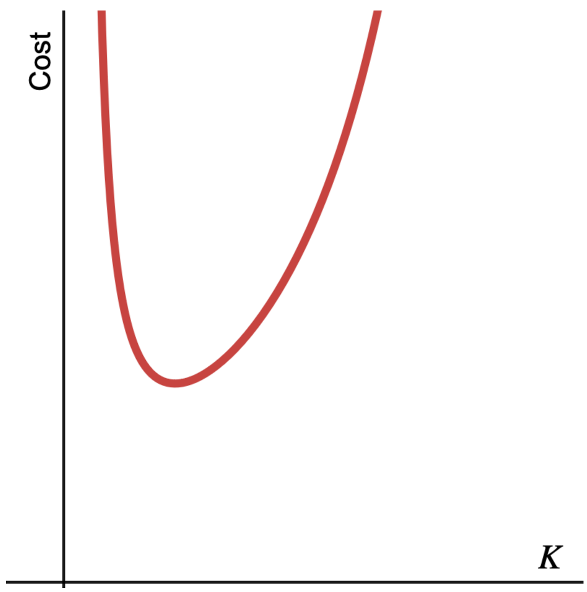
By properties of geometric series:

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

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

A 1-D optimization problem

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



Desmos [graph]