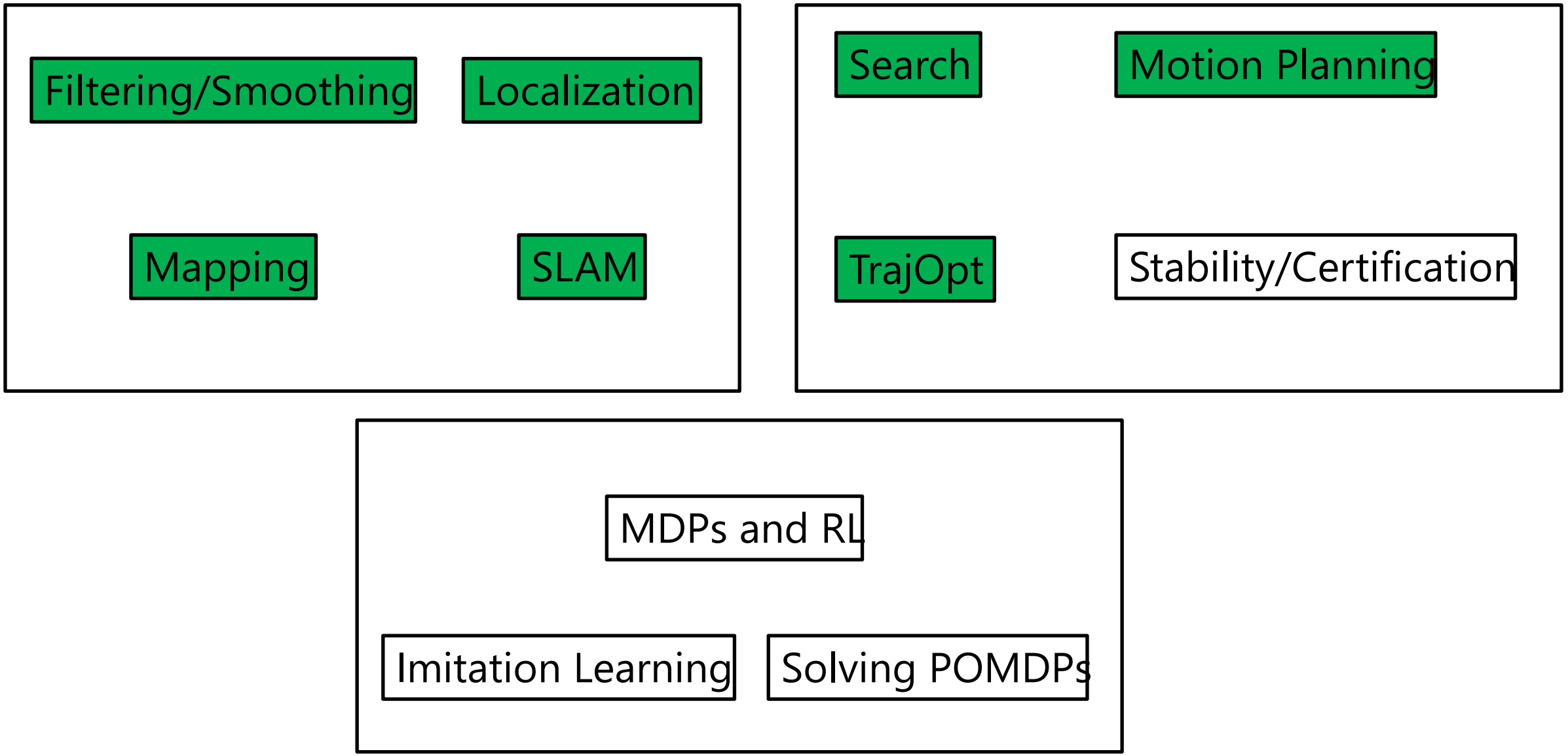
Robotics
Spring 2023

Abhishek Gupta

TAs: Yi Li, Srivatsa GS

Many slides courtesy of Sidd Srinivasa (UW), Pieter Abbeel (UC Berkeley), Igor Mordatch (Google)

Recap: Course Overview



Lecture Outline

Optimal Control: Collocation and Shooting

Lyapunov Stability

Reinforcement Learning

Zooming out: Optimal Control

- LQR is a special case of optimal control with analytic solution
- Optimal control more generally solves this problem

$$\min_{u_{1:T}} \sum_{t=1}^{T} g(x_t, u_t) + G(x_T, u_T)$$

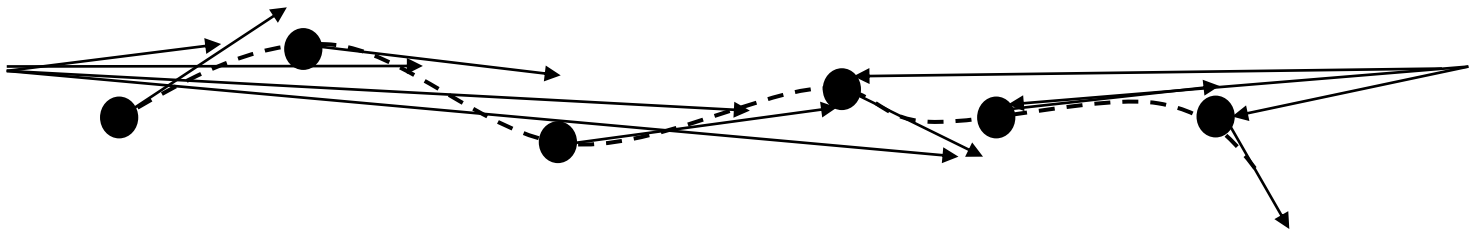
$$x_{t+1} = f(x_t, u_t)$$

- Challenge: often non-convex and challenging to optimize!

Two techniques for optimal control

- Shooting:
  - Optimize for actions such that cost is minimized and future states are implicitly defined by actions
    - LQR is a shooting method with linear dynamics and quadratic cost
- Collocation:
  - Optimize for actions **and** states such that cost is minimized and the dynamics are explicitly satisfied as a constaint

Shooting
(optimize actions)



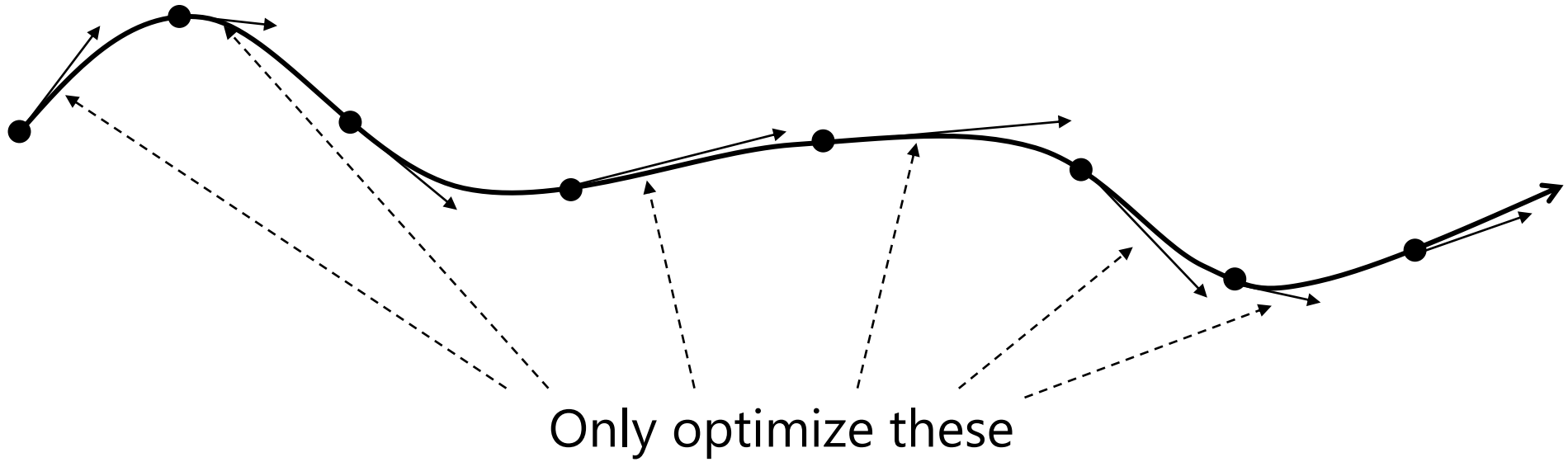
Collocation (also opt states)

Optimal Control -- Approaches

	Return open-loop controls $u_0, u_1, \ldots, u_H$	Return feedback policy $\pi_\theta(\cdot)$ (e.g. linear or neural net)
shooting	$\min_{u_0,u_1,\ldots,u_H} c(x_0,u_0) + c(f(x_0,u_0),u_1) + c(f(f(x_0,u_0),u_1),u_2) + \ldots$	$\min_\theta c(x_0,\pi_\theta(x_0)) + c(f(x_0,\pi_\theta(x_0)),\pi_\theta(f(x_0,\pi_\theta(x_0)))) + \ldots$
collocation	$\min_{x_0,u_0,x_1,u_1,\ldots,x_H,u_H} \sum_{t=0}^{H} c(x_t,u_t)$ s.t. $x_{t+1} = f(x_t,u_t) \quad \forall t$	$\min_{x_0,x_1,\ldots,x_H,\theta} \sum_{t=0}^{H} c(x_t,\pi_\theta(x_t))$ s.t. $x_{t+1} = f(x_t,\pi_\theta(x_t)) \quad \forall t$ --- $\min_{x_0,u_0,x_1,u_1,\ldots,x_H,u_H,\theta} \sum_{t=0}^{H} c(x_t,u_t)$ s.t. $x_{t+1} = f(x_t,u_t) \quad \forall t$ $u_t = \pi_\theta(x_t) \quad \forall t$

Optimal Control: Shooting

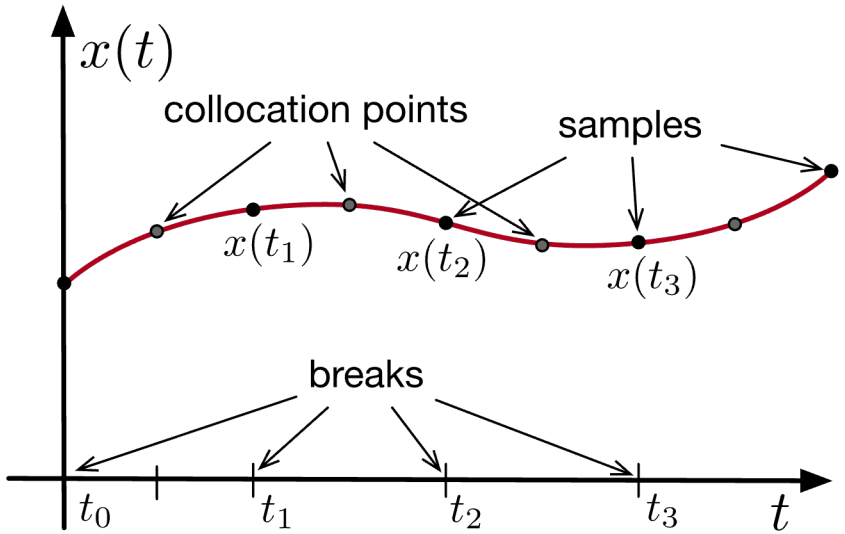
- Only optimize over actions to minimize cost



- Usually easier to specify valid domain, but worse conditioning

Optimal Control: Collocation

- Jointly optimize over both states and actions to minimize cost, subject to constraints



- Better numerical conditioning

(In)stability of Open-loop Shooting

- Let's reconsider:

$$\min_{u_0, u_1, \ldots, u_H} c(x_0, u_0) + c(f(x_0, u_0), u_1) + c(f(f(x_0, u_0), u_1), u_2) + \ldots$$

  - Rolling out u_0, u_1, …, u_H can often be unstable because small numerical errors (or noise) can amplify in case of unstable dynamics
  - In turn, this can make this formulation unstable to optimize

- Solutions:
  - During roll-out, use re-planning (model-predictive control)
  - **Use collocation**

Collocation versus Shooting

- Shooting:
  - Improve sequence of controls over time, at all times u (or pi) are meaningful
  - Often poorly conditioned (effect of early u so much higher than later u)
  - Not clear how to initialize in a way that nudges towards a goal state
- Collocation
  - Might converge to a local optimum that's infeasible, and until converged often not feasible
  - x provides decoupling between time-steps, making computation stable
  - Can initialize with simple linear interpolation or guess of good trajectory
- Iterative LQR?
  - Specific example of a shooting method, with linear controllers, and second order optimization

Let’s look a little closer

Solving Problems with Direct Shooting

Forward Shooting:

$$\min_{\mathbf{u}^0 \ldots \mathbf{u}^T} \sum_t C^t(\mathbf{x}^t), \quad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t)$$

Solution 1:
Linearize and perform iLQR/DDQP

Solution 3:
Formulate as a convex optimization problem, use convex solver (gurobi/snopt)

Solution 2:
If differentiable, direct backpropagation through time

Visualization of Direct Shooting

Visualization of Direct Shooting

Forward Shooting:

$$\min_{\mathbf{u}^0 \ldots \mathbf{u}^T} \sum_t C^t(\mathbf{x}^t), \quad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t)$$

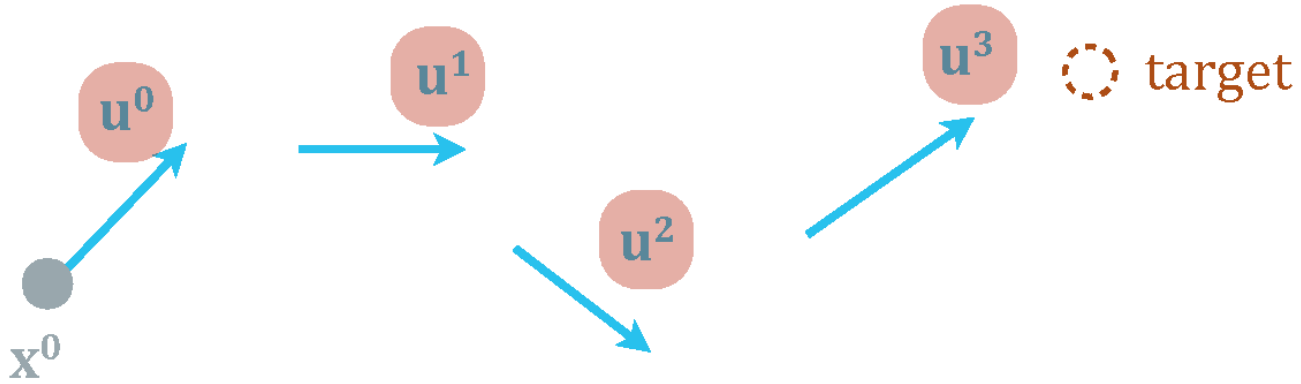
target

$\mathbf{x}^0$

Visualization of Direct Shooting

Forward Shooting:

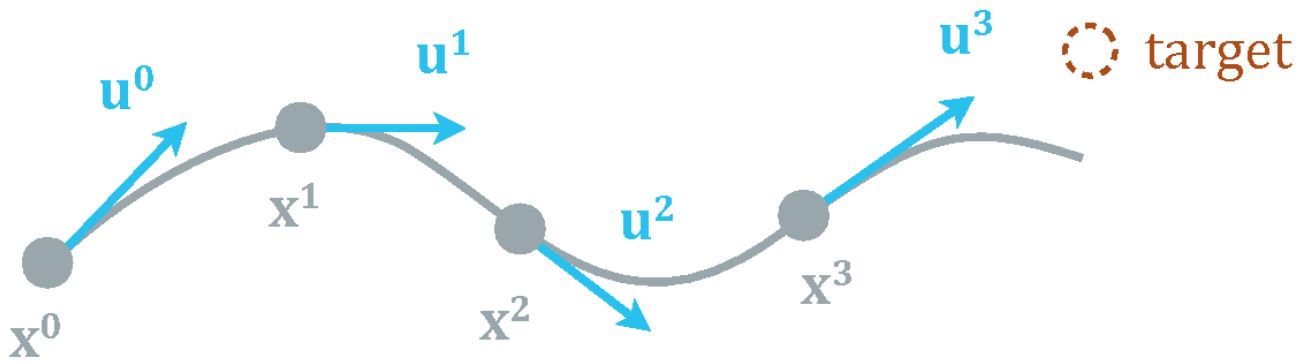
$$\min_{\mathbf{u}^0 \ldots \mathbf{u}^T} \sum_t C^t(\mathbf{x}^t), \quad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t)$$



Visualization of Direct Shooting

Forward Shooting:

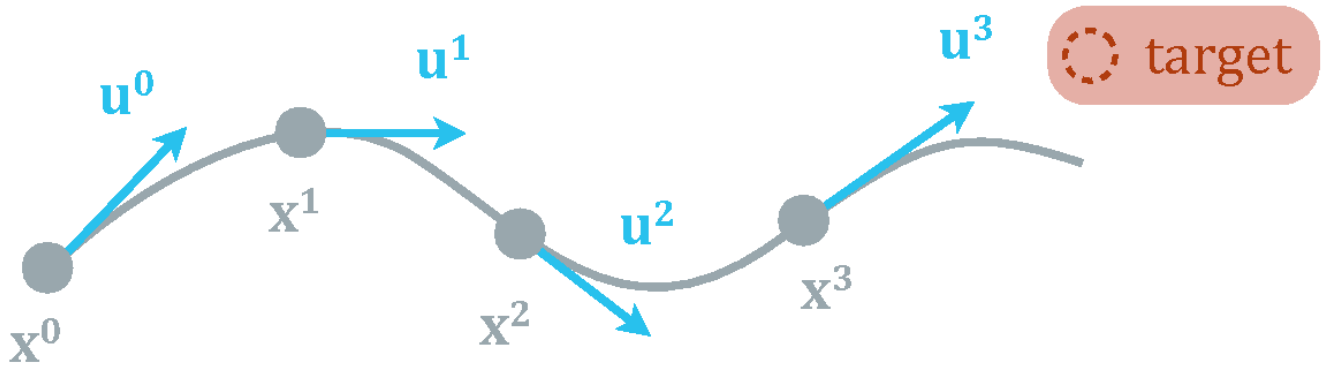
$$\min_{\mathbf{u}^0 \ldots \mathbf{u}^T} \sum_t C^t(\mathbf{x}^t), \qquad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t)$$



Visualization of Direct Shooting

Forward Shooting:

$$\min_{\mathbf{u}^0 \ldots \mathbf{u}^T} \sum_t C^t(\mathbf{x}^t), \qquad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t)$$



Poor Conditioning in Direct Shooting

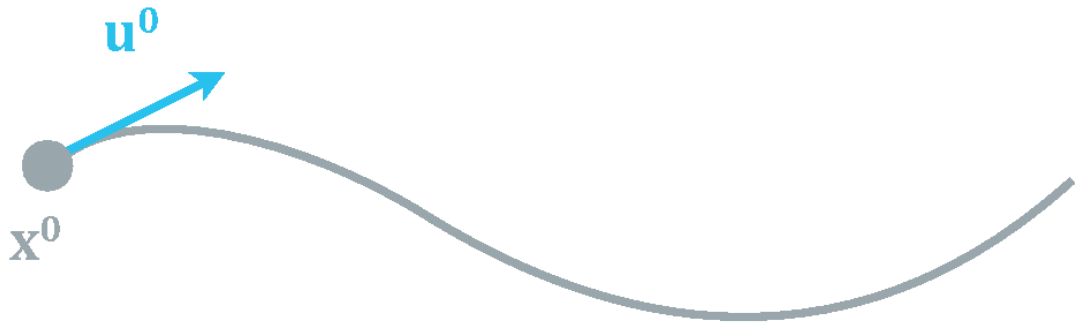
Forward Shooting:

$$\min_{\mathbf{u}^0 \ldots \mathbf{u}^T} \sum_t C^t(\mathbf{x}^t), \quad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t)$$

Poor Conditioning in Direct Shooting

Forward Shooting:

$$\min_{\mathbf{u}^0 \ldots \mathbf{u}^T} \sum_t C^t(\mathbf{x}^t), \quad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t)$$



Poor Conditioning in Direct Shooting

Forward Shooting:

$$\min_{\mathbf{u}^0 \ldots \mathbf{u}^T} \sum_t C^t(\mathbf{x}^t), \quad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t)$$

Poor Conditioning in Direct Shooting

Forward Shooting:

$$\min_{\mathbf{u}^0 \ldots \mathbf{u}^T} \sum_t C^t(\mathbf{x}^t), \quad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t)$$

$$\min_{\mathbf{u}_1, \ldots, \mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \cdots$$

$$\cdots + c(f(f(\ldots)\ldots), \mathbf{u}_T)$$

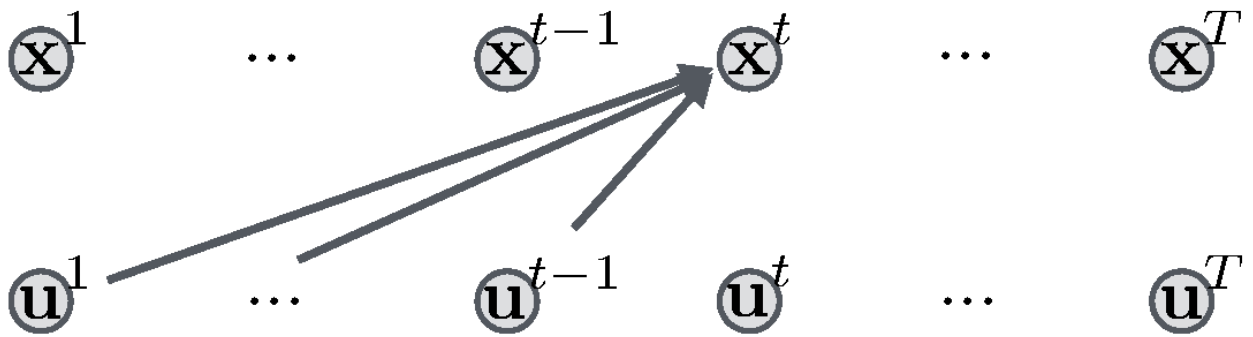
Poor Conditioning in Direct Shooting

Forward Shooting:

$$\min_{\mathbf{u}^0 \ldots \mathbf{u}^T} \sum_t C^t(\mathbf{x}^t), \quad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t)$$

$$\min_{\mathbf{u}_1, \ldots, \mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \cdots$$

$$\cdots + c(f(f(\ldots)\ldots), \mathbf{u}_T)$$



Narrow Feasible Region

Forward Shooting:

$$\min_{\mathbf{u}^0 \ldots \mathbf{u}^T} \sum_t C^t(\mathbf{x}^t), \quad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t)$$

Narrow Feasible Region

Forward Shooting:

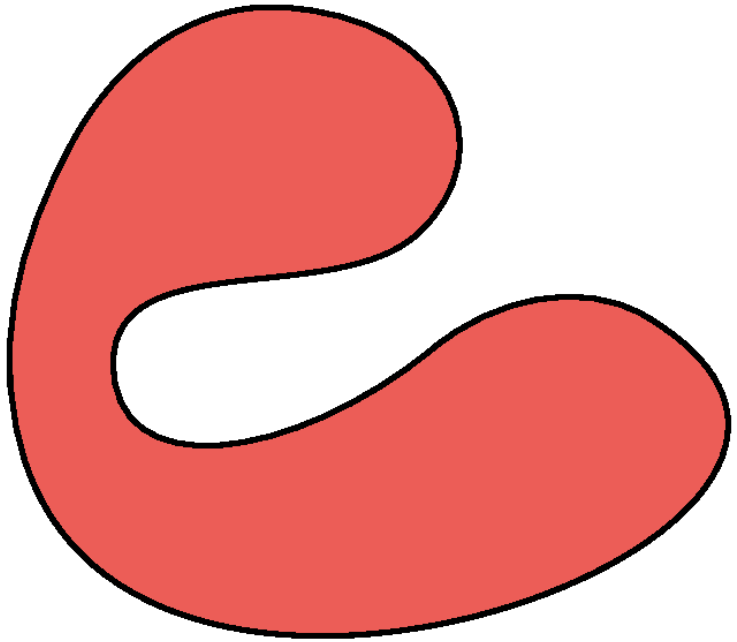
$$\min_{\mathbf{u}^0 \ldots \mathbf{u}^T} \sum_t C^t(\mathbf{x}^t), \quad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t)$$

implicit hard constraint

Narrow Feasible Region

Forward Shooting:

$$\min_{\mathbf{u}^0 \ldots \mathbf{u}^T} \sum_t C^t(\mathbf{x}^t), \qquad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t)$$

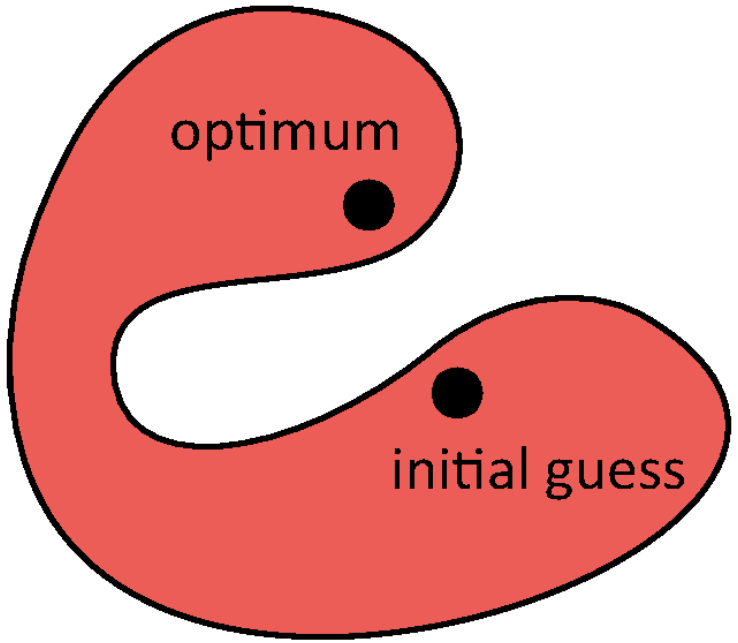


feasible region

Narrow Feasible Region

Forward Shooting:

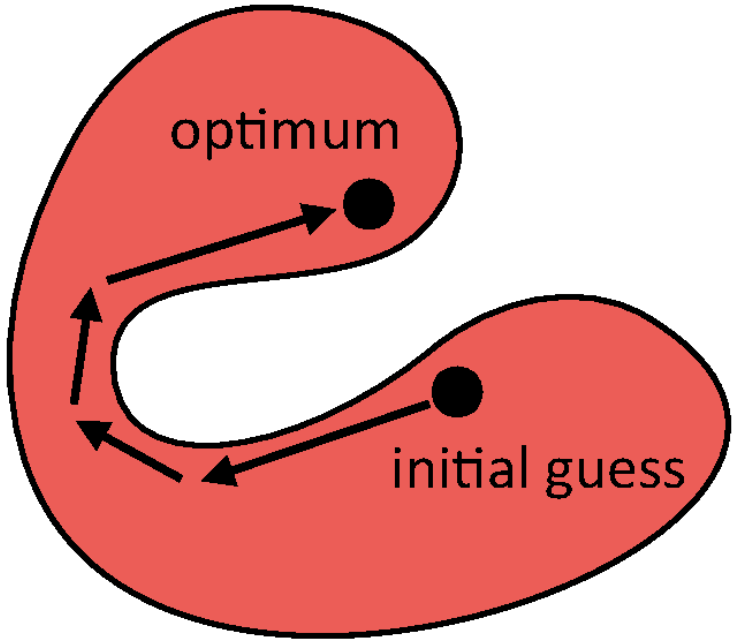
$$\min_{\mathbf{u}^0 \ldots \mathbf{u}^T} \sum_t C^t(\mathbf{x}^t), \quad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t)$$



Narrow Feasible Region

Forward Shooting:

$$\min_{\mathbf{u}^0 \ldots \mathbf{u}^T} \sum_t C^t(\mathbf{x}^t), \quad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t)$$

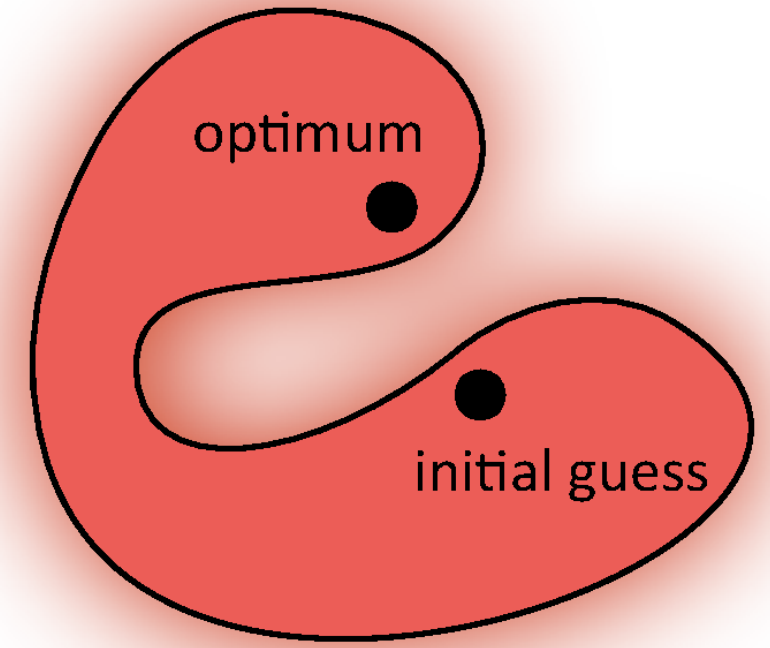


Narrow Feasible Region

Forward Shooting:

$$\min_{\mathbf{u}^0 \ldots \mathbf{u}^T} \sum_t C^t(\mathbf{x}^t), \qquad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t)$$

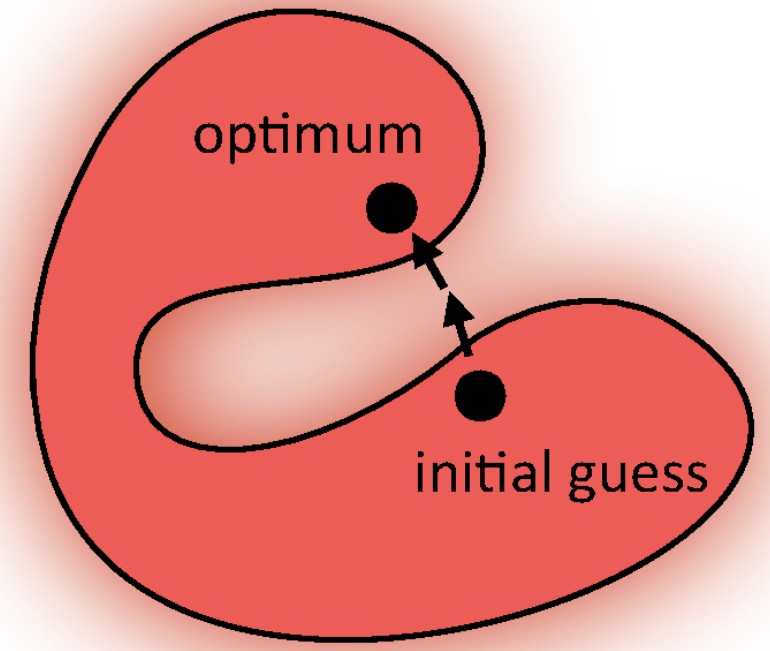
soft constraint



Narrow Feasible Region

Forward Shooting:

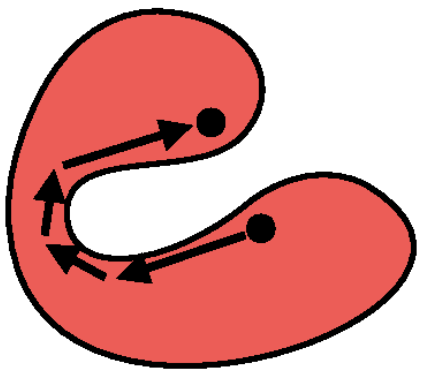
$$\min_{\mathbf{u}^0 \ldots \mathbf{u}^T} \sum_t C^t(\mathbf{x}^t), \quad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t)$$



Narrow Feasible Region

Forward Shooting:

$$\min_{\mathbf{u}^0 \ldots \mathbf{u}^T} \sum_t C^t(\mathbf{x}^t), \quad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t)$$



- Comes up as an issue in practice
  - collisions, falling down, etc…
- Prone to falling into local minima
- Makes solution sensitive to initial guess
- Initial guess from demonstrations and randomization helps

Comparison between Shooting and Collocation

shooting	$\min_{u_0, u_1, \ldots, u_H} c(x_0, u_0) + c(f(x_0, u_0), u_1) + c(f(f(x_0, u_0), u_1), u_2) + \ldots$
collocation	$\min_{x_0, u_0, x_1, u_1, \ldots, x_H, u_H} \sum_{t=0}^{H} c(x_t, u_t)$ s.t. $x_{t+1} = f(x_t, u_t) \quad \forall t$

Comparison between Shooting and Collocation

shooting	$\min_{u_0, u_1, \ldots, u_H} \; c(x_0, u_0) + c(f(x_0, u_0), u_1) + c(f(f(x_0, u_0), u_1), u_2) + \ldots$
collocation	$\min_{x_0, u_0, x_1, u_1, \ldots, x_H, u_H} \; \sum_{t=0}^{H} c(x_t, u_t)$ $\text{s.t.} \quad x_{t+1} = f(x_t, u_t) \;\; \forall t$

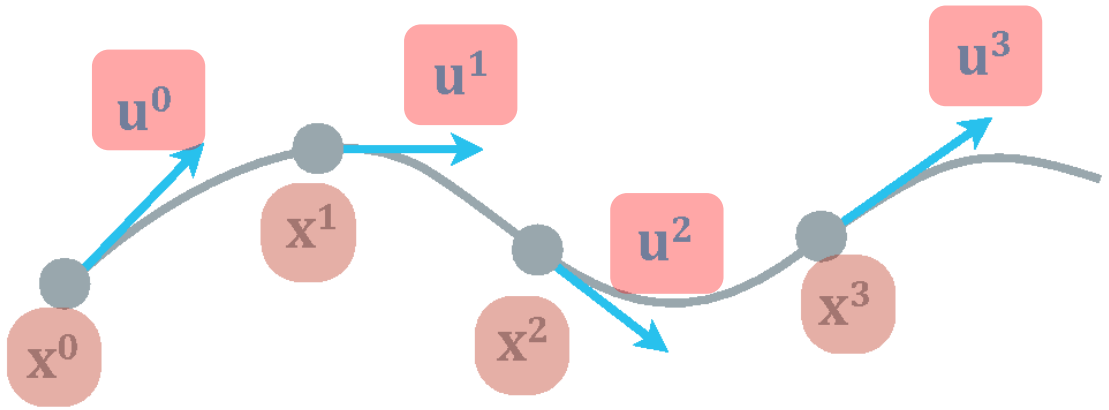
Let's think about collocation

shooting	$\min_{u_0, u_1, \ldots, u_H} c(x_0, u_0) + c(f(x_0, u_0), u_1) + c(f(f(x_0, u_0), u_1), u_2) + \ldots$
collocation	$\min_{x_0, u_0, x_1, u_1, \ldots, x_H, u_H} \sum_{t=0}^{H} c(x_t, u_t)$ s.t. $x_{t+1} = f(x_t, u_t) \quad \forall t$

Solving Collocation Problems

$$\min_{x_0, u_0, x_1, u_1, \ldots, x_H, u_H} \quad \sum_{t=0}^{H} c(x_t, u_t)$$

$$\text{s.t.} \quad x_{t+1} = f(x_t, u_t) \quad \forall t$$



Idea 1: Direct constrained optimization through forward dynamics

Idea 2: constrained optimization through inverse dynamics

Collocation with Inverse Dynamics

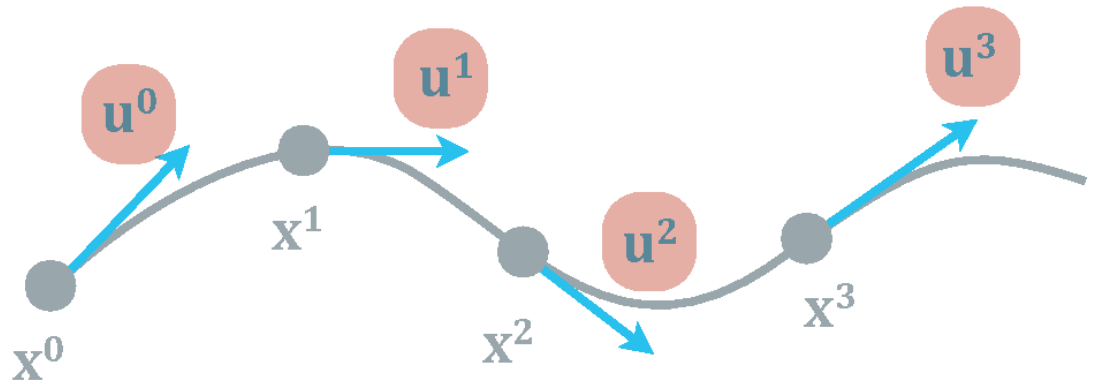
Forward Shooting:

$$\min_{\mathbf{u}^0 \ldots \mathbf{u}^T} \sum_t C^t(\mathbf{x}^t), \quad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t)$$

inverse dynamics function

Direct Collocation:

$$\min_{\mathbf{x}^0 \ldots \mathbf{x}^T} \sum_t C^t(\mathbf{x}^t), \; st \; f^{-1}(\mathbf{x}^t, \mathbf{x}^{t+1}) = \mathbf{u}^t \in \mathcal{U}$$



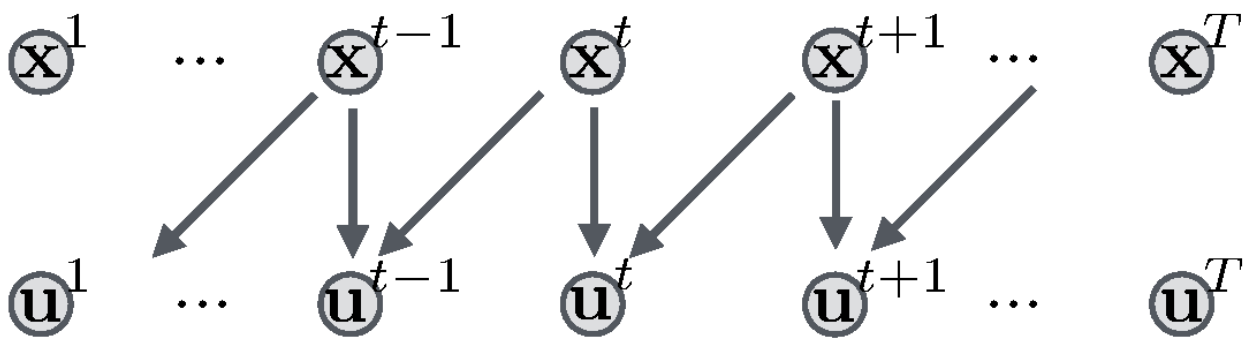
Collocation with Inverse Dynamics

Forward Shooting:

$$\min_{\mathbf{u}^0 \ldots \mathbf{u}^T} \sum_t C^t(\mathbf{x}^t), \quad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t)$$

Direct Collocation:

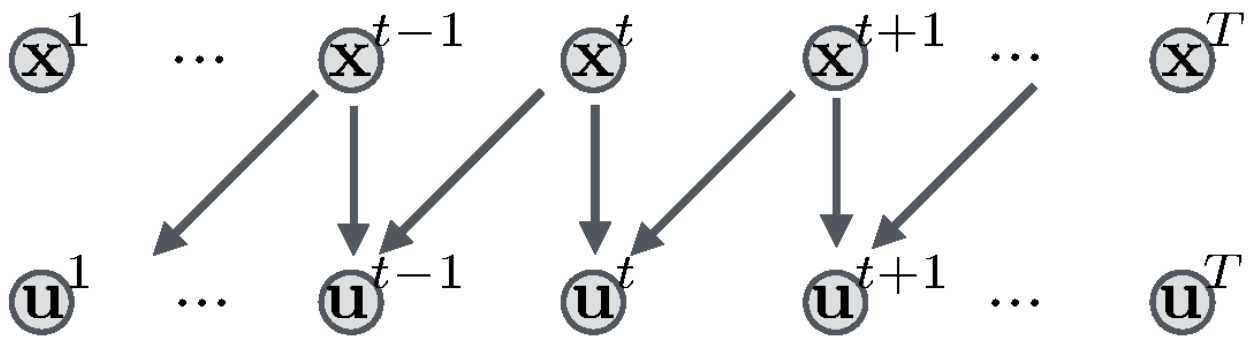
$$\min_{\mathbf{x}^0 \ldots \mathbf{x}^T} \sum_t C^t(\mathbf{x}^t), \ \ st \ f^{-1}(\mathbf{x}^t, \mathbf{x}^{t+1}) = \mathbf{u}^t \in \mathcal{U}$$



Advantages of Direct Collocation

Direct Collocation:

$$\min_{\mathbf{x}^0 \ldots \mathbf{x}^T} \sum_t C^t(\mathbf{x}^t), \;\; st \;\; f^{-1}(\mathbf{x}^t, \mathbf{x}^{t+1}) = \mathbf{u}^t \in \mathcal{U}$$



- Only pairwise dependencies
- Good conditioning
  - changing x^1 has similar effect as changing x^T
- No forward integration instability

Feasibility under Direct Collocation

Direct Collocation:

$$\min_{\mathbf{x}^0 \ldots \mathbf{x}^T} \sum_t C^t(\mathbf{x}^t), \quad st \; f^{-1}(\mathbf{x}^t, \mathbf{x}^{t+1}) = \mathbf{u}^t \in \mathcal{U}$$

Explicit rather than implicit constraint

Feasibility under Direct Collocation

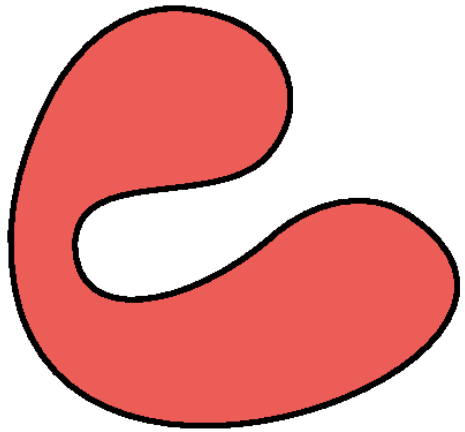
Direct Collocation:

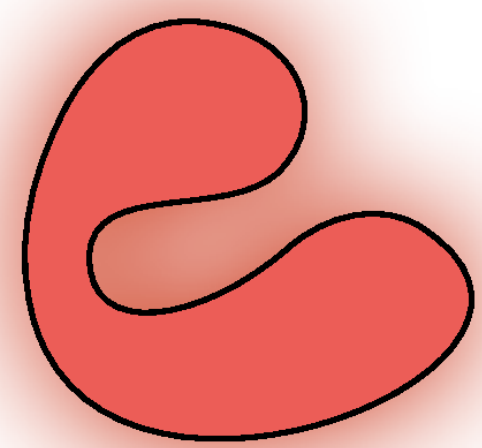
$$\min_{\mathbf{x}^0 \ldots \mathbf{x}^T} \sum_t C^t(\mathbf{x}^t), \quad st \; f^{-1}(\mathbf{x}^t, \mathbf{x}^{t+1}) = \mathbf{u}^t \in \mathcal{U}$$

Explicit rather than implicit constraint

Can be hard or soft

Less prone to local minima





Shooting vs Collocation

Forward Shooting:

$$\min_{\mathbf{u}^0 \ldots \mathbf{u}^T} \sum_t C^t(\mathbf{x}^t), \quad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t)$$

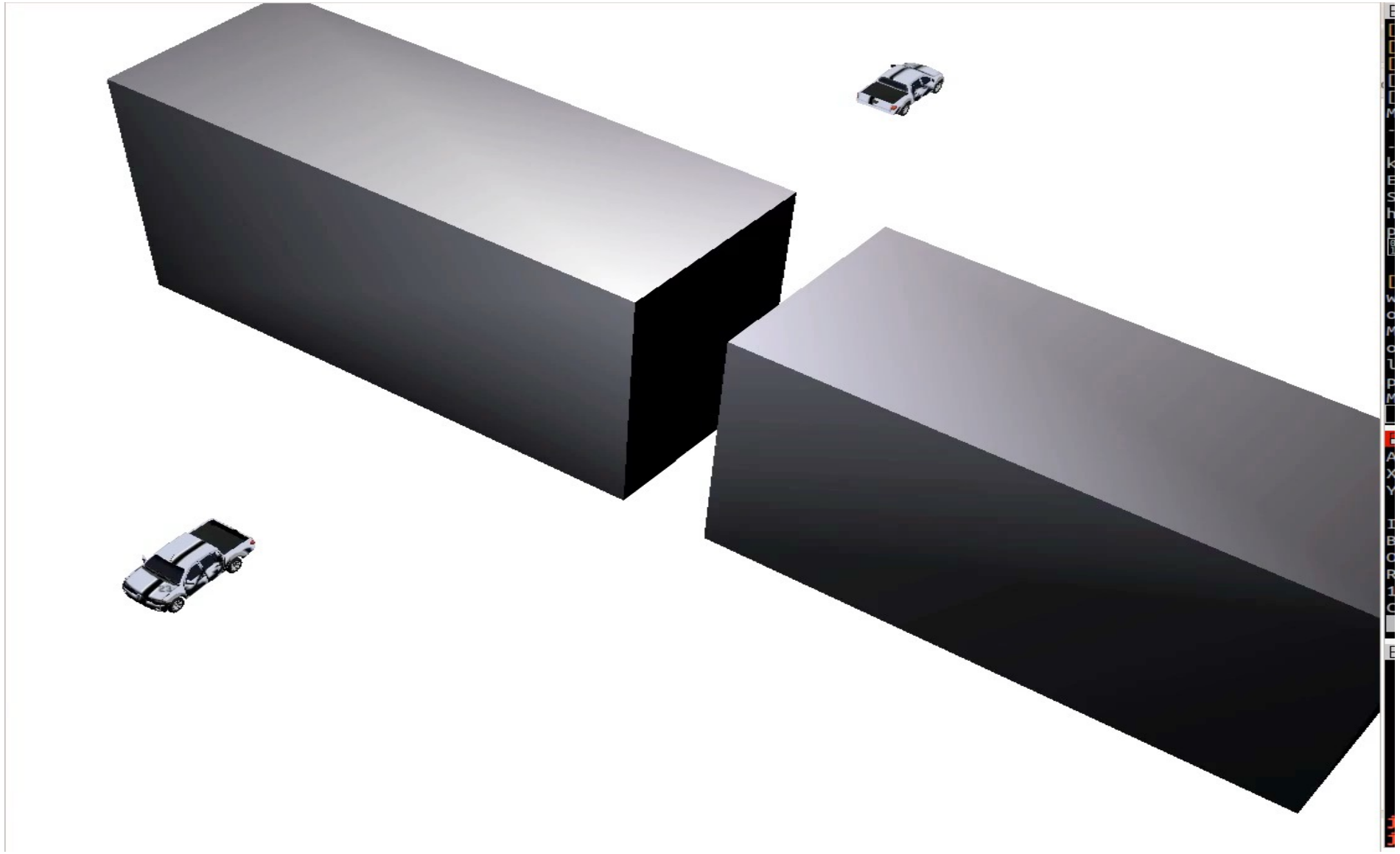
- Optimize over controls
- State trajectory is implicit
- Dynamics is an implicit constraint (always satisfied)

Direct Collocation:

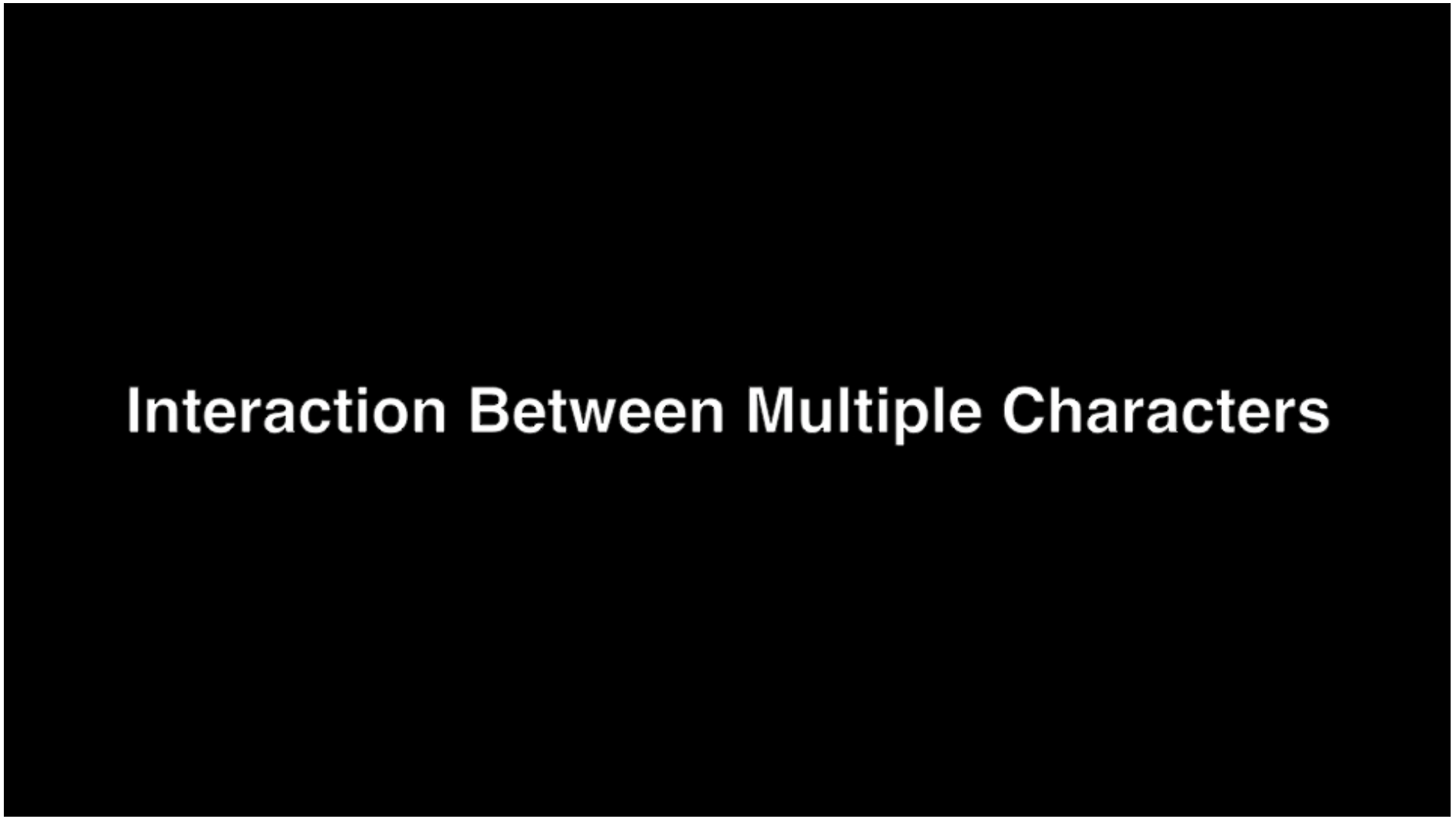
$$\min_{\mathbf{x}^0 \ldots \mathbf{x}^T} \sum_t C^t(\mathbf{x}^t), \ \ st \ \ f^{-1}(\mathbf{x}^t, \mathbf{x}^{t+1}) = \mathbf{u}^t \in \mathcal{U}$$

- Optimize over states
- Controls and forces are implicit
- Dynamics is an explicit constraint (can be soft)

Optimal control methods in action



Optimal control methods in action



Optimal control methods in action

Lecture Outline

Optimal Control: Collocation and Shooting

Lyapunov Stability

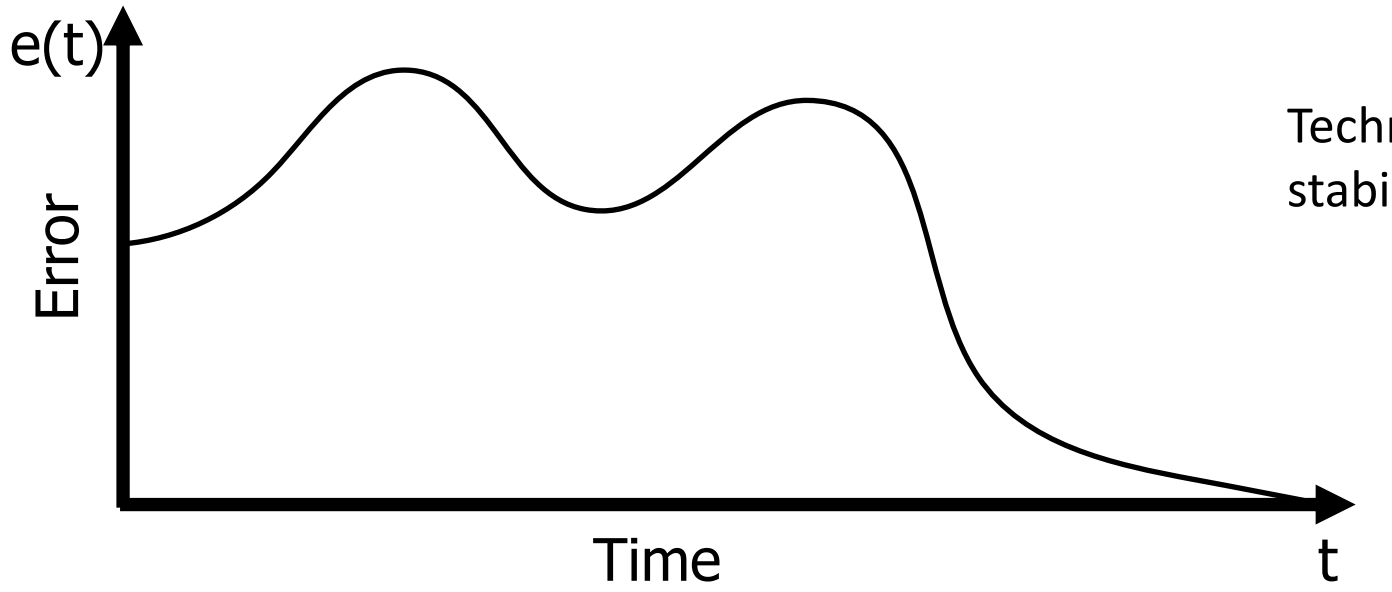
Reinforcement Learning

How can we prove that a controller is stable?

Lyapunov Stability

What is stability?

$$\lim_{t \to \infty} e(t) = 0$$



Technically: this is global asymptotic stability, there are other notions of stability

So we want both $e(t) \to 0$ and $\dot{e}(t) \to 0$

More technically precise definitions

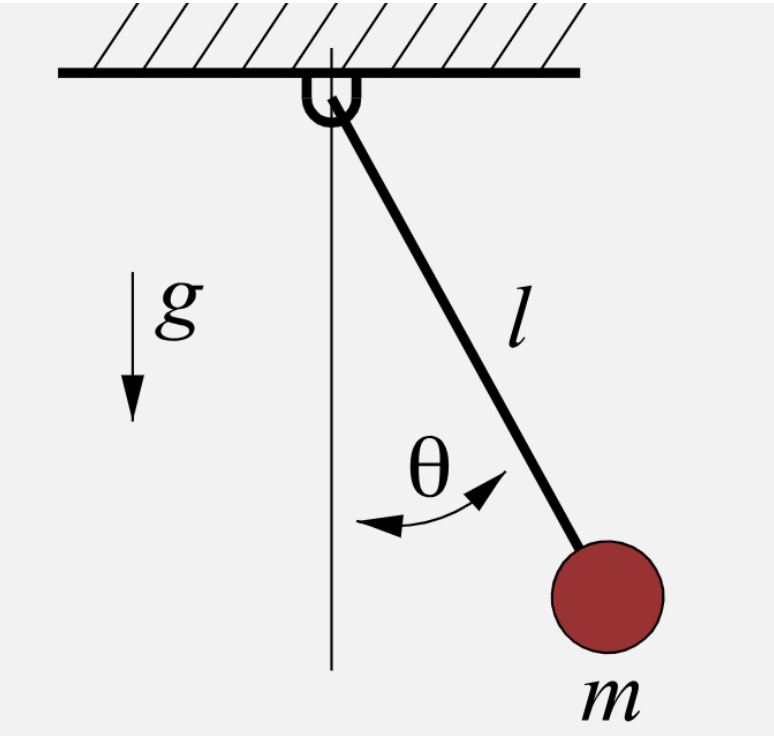
1. This equilibrium is said to be **Lyapunov stable**, if, for every $\epsilon > 0$, there exists a $\delta > 0$ such that, if $\|x(0) - x_e\| < \delta$, then for every $t \geq 0$ we have $\|x(t) - x_e\| < \epsilon$.
2. The equilibrium of the above system is said to be **asymptotically stable** if it is Lyapunov stable and there exists $\delta > 0$ such that if $\|x(0) - x_e\| < \delta$, then $\lim_{t \to \infty} \|x(t) - x_e\| = 0$.
3. The equilibrium of the above system is said to be **exponentially stable** if it is asymptotically stable and there exist $\alpha > 0, \beta > 0, \delta > 0$ such that if $\|x(0) - x_e\| < \delta$, then $\|x(t) - x_e\| \leq \alpha \|x(0) - x_e\| e^{-\beta t}$, for all $t \geq 0$.

Stable i.s.l: doesn't go unbounded

Asymptotically stable → converges

Exponentially stable → converges fast

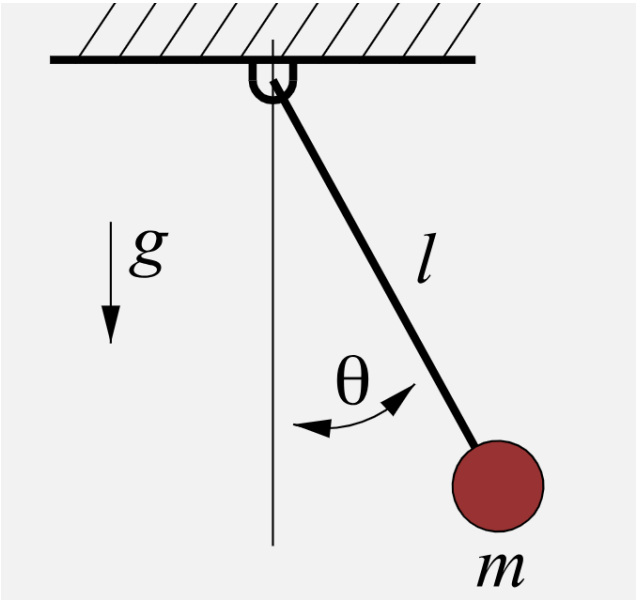
Let’s start with a simple system



$$ml^2\ddot{\theta} + mgl\sin\theta = u$$

Can derive via method of Lagrange or newtons laws of motion

Is a pendulum stable?



$$ml^2\ddot{\theta} + mgl\sin\theta = u$$

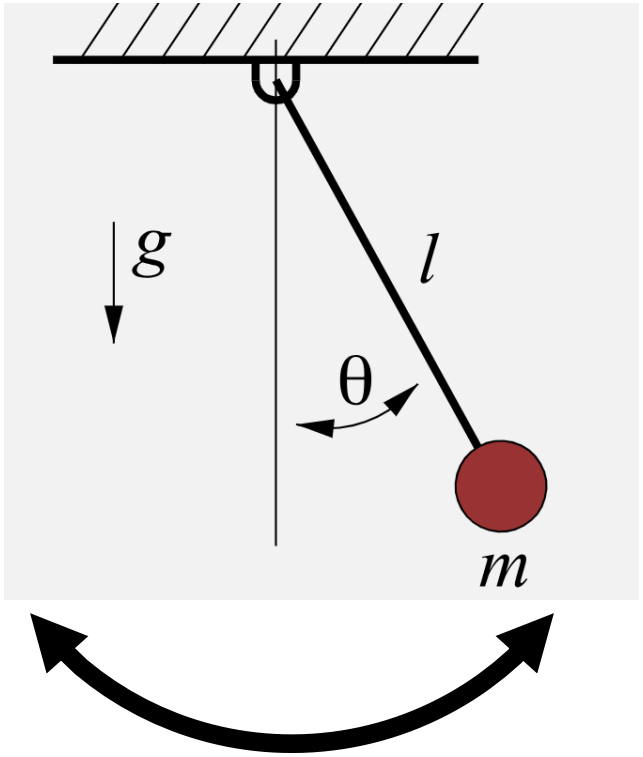
What control law should we use to stabilize the pendulum, i.e.

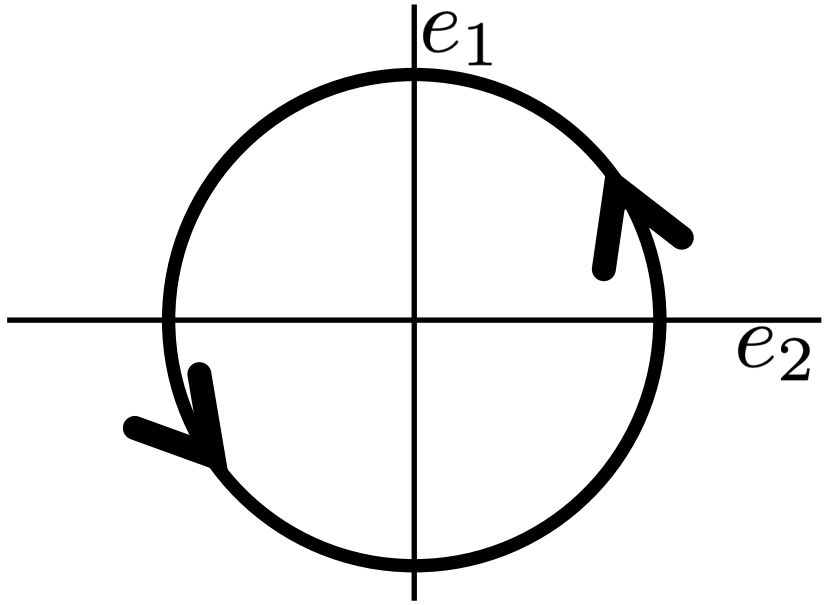
Choose $u = \pi(\theta, \dot{\theta})$ such that $\theta \to 0$

$\dot{\theta} \to 0$

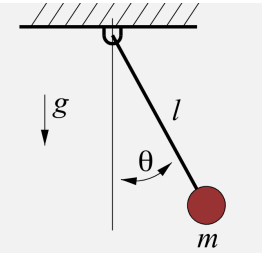
How does the passive error dynamics behave?

Set u=0. Dynamics is not stable, pendulum keeps oscillatin





How do we verify if a controller is stable?



$$ml^2\ddot{\theta} + mgl\sin\theta = u$$

Lets pick the following law:

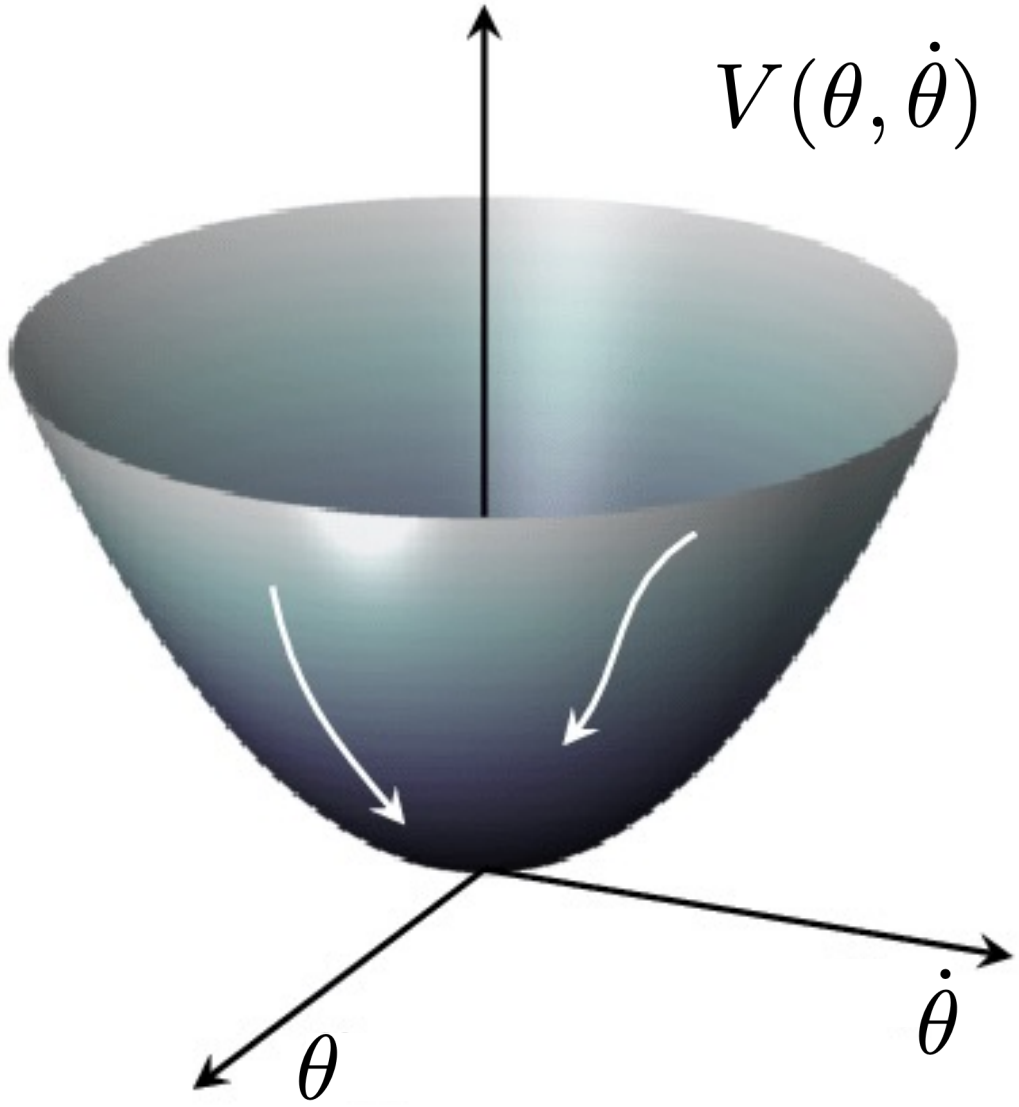
$$u = -K\dot{\theta}$$

Is this stable? How do we know?

We can simulate the dynamics from different start point and check….

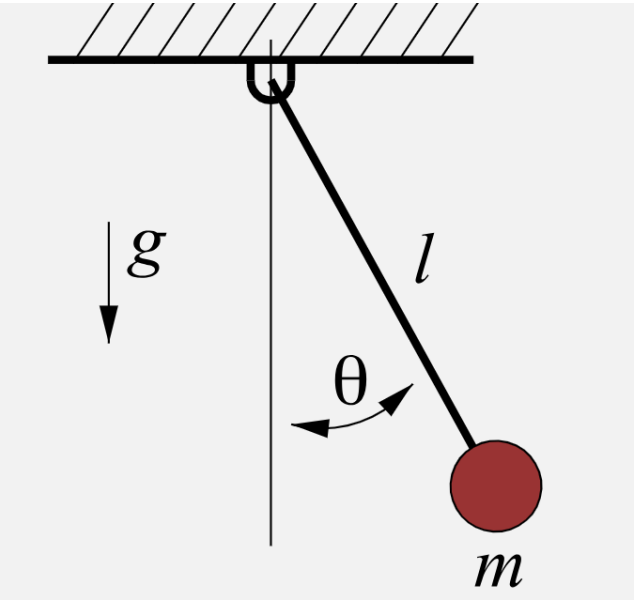
but how many points do we check? what if we miss some points?

Key Idea: Think about energy!



Make energy decay to 0 and stay there

$$V(\theta, \dot{\theta}) = \frac{1}{2} m l^2 \dot{\theta}^2 + mgl(1 - \cos\theta)$$
$$> 0$$



$$\dot{V}(\theta, \dot{\theta}) = ml^2 \dot{\theta}\ddot{\theta} + mgl(\sin\theta)\dot{\theta}$$
$$= \dot{\theta}(u - mgl\sin\theta) + mgl(\sin\theta)\dot{\theta}$$
$$= \dot{\theta}u$$

Choose a control law $u = -k\dot{\theta}$

$$\dot{V}(\theta, \dot{\theta}) = -k\dot{\theta}^2 < 0$$

Lyapunov function:
A generalization of energy

Lyapunov function for a closed-loop system

1. Construct an energy function that is always positive

$$V(x) > 0, \forall x$$

Energy is only 0 at the origin, i.e. $V(0) = 0$

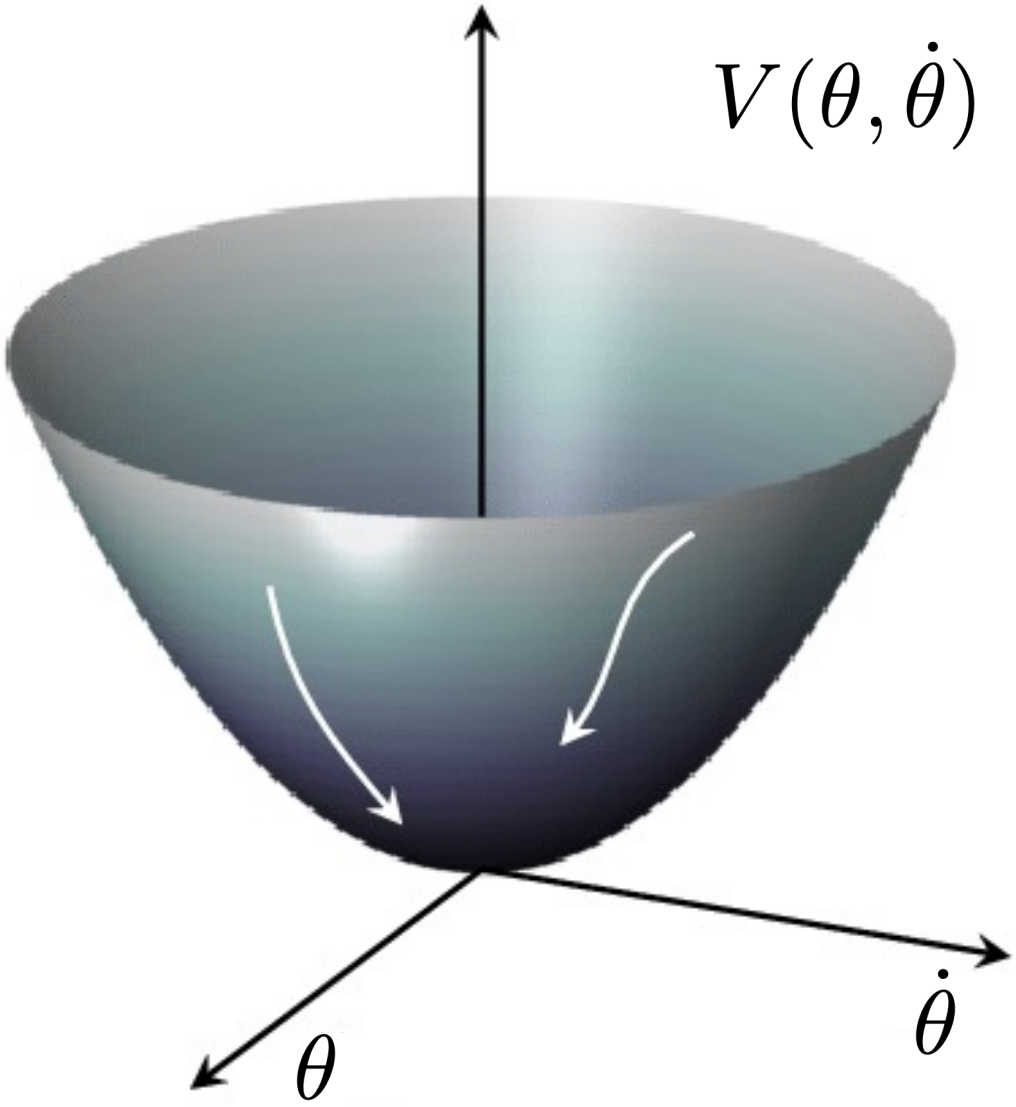
2. Choose a control law such that this energy always decreases

$$\dot{V}(x) < 0, \forall x$$

Energy rate is 0 at origin, i.e. $\dot{V}(0) = 0$

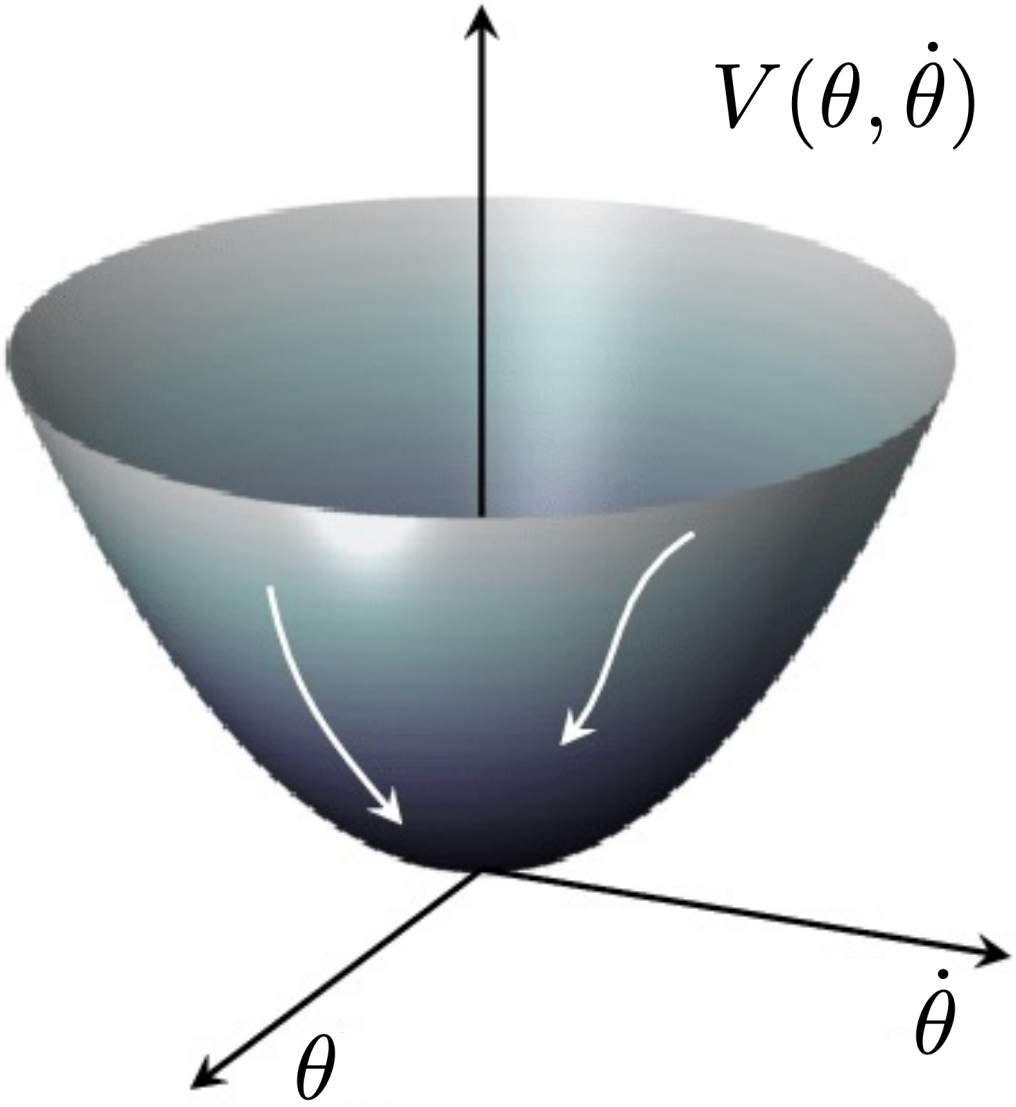
No matter where you start, energy will decay and you will reach 0!

Intuition for Lyapunov Functions



1. V can only be 0 at 0, and is always positive
2. dV/dt is negative, meaning V is going towards 0
   → since it is positive and can only be 0 at 0, eventually this will converge to 0

Why are Lyapunov functions useful?



Provides a way to guarantee stability of a system/controller without exhaustive forward simulation

Finding Lyapunov Functions

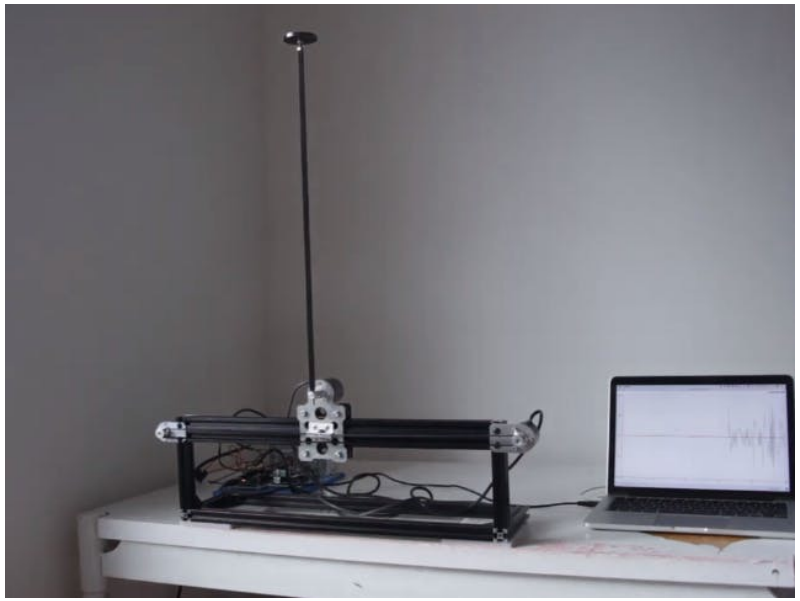
Strategy 1: Use domain knowledge and hand-design

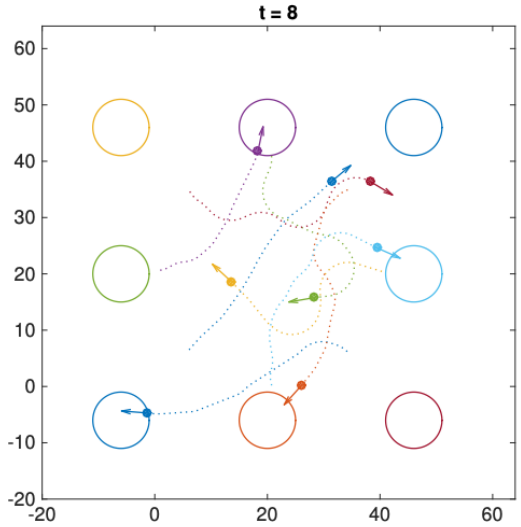
Strategy 2: For linear systems, can solve LMI to obtain a function

Strategy 3: For non-linear functions, can use ideas like sum of squares programming to find V

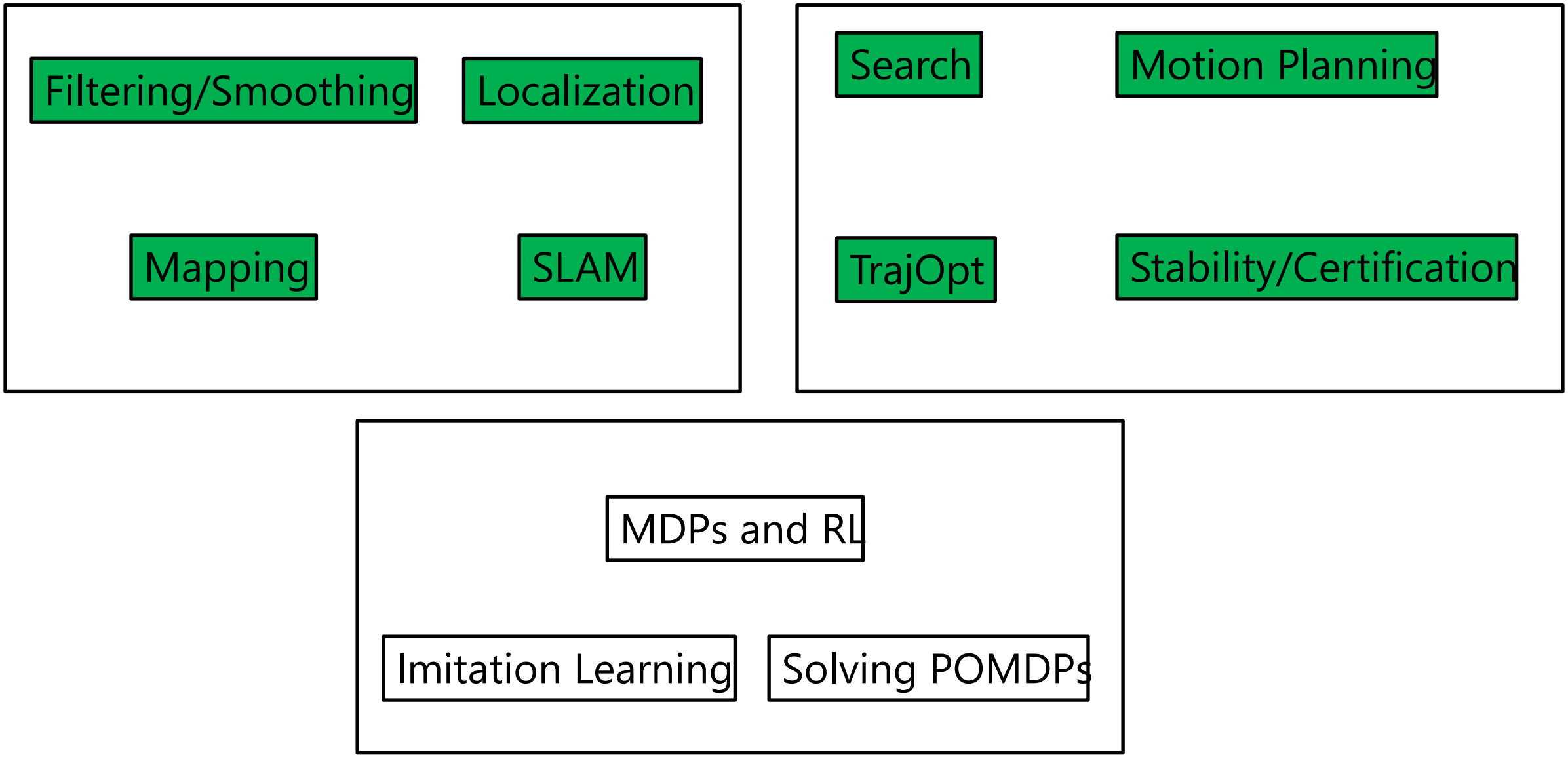
Strategy 4: Do bilevel optimization to find V or learn V

Uses of Lyapunov Functions





Recap: Course Overview



Lecture Outline

Optimal Control: Collocation and Shooting

Lyapunov Stability

Reinforcement Learning

When is optimal control + planning not enough?

Unstructured Environments

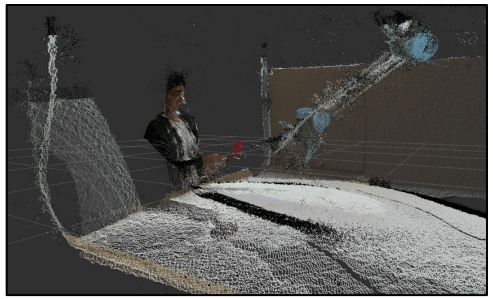
Diversity

Variability

Complexity

Challenging to model these environments

How does a typical robotics pipeline look?

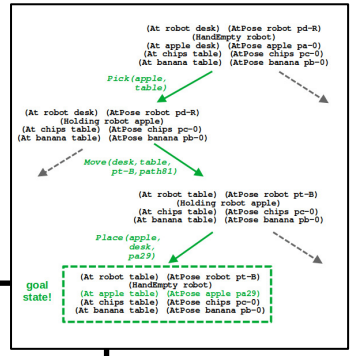


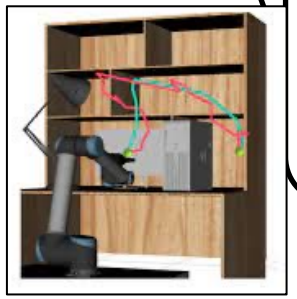
State Estimation

Modeling and Prediction

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} = \tau_g(\mathbf{q}) + \mathbf{B}\mathbf{u},$$

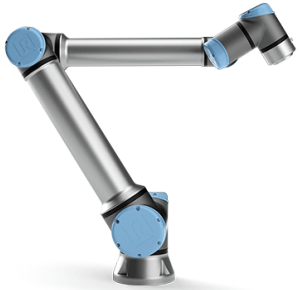
High-level planning

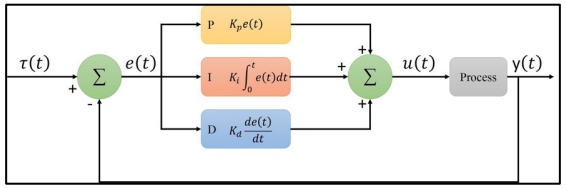




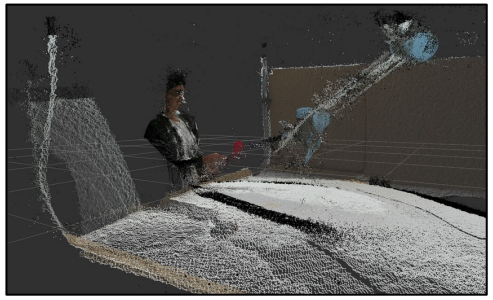
Low-level planning

Low-level control





Deep reinforcement learning pipeline for robotics



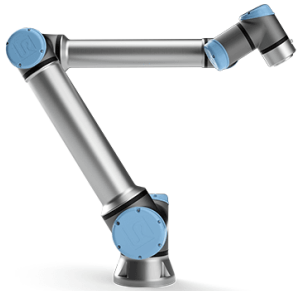
State Estimation

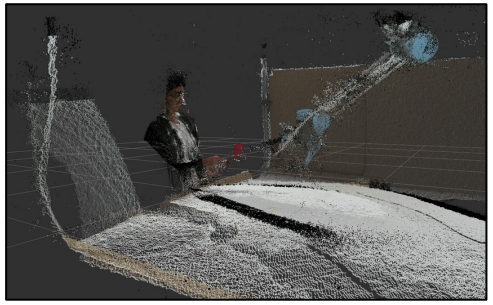
Modeling and Prediction

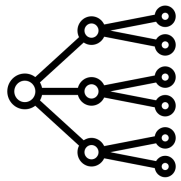
High-level planning

Low-level planning

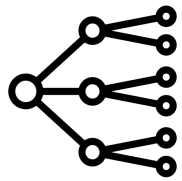
Low-level control

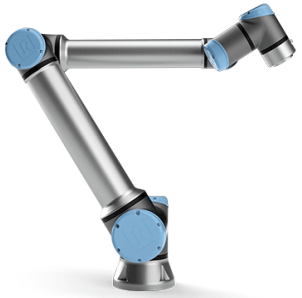






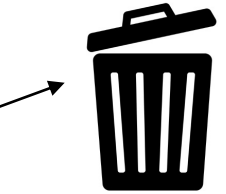
End to end policy – perception + control



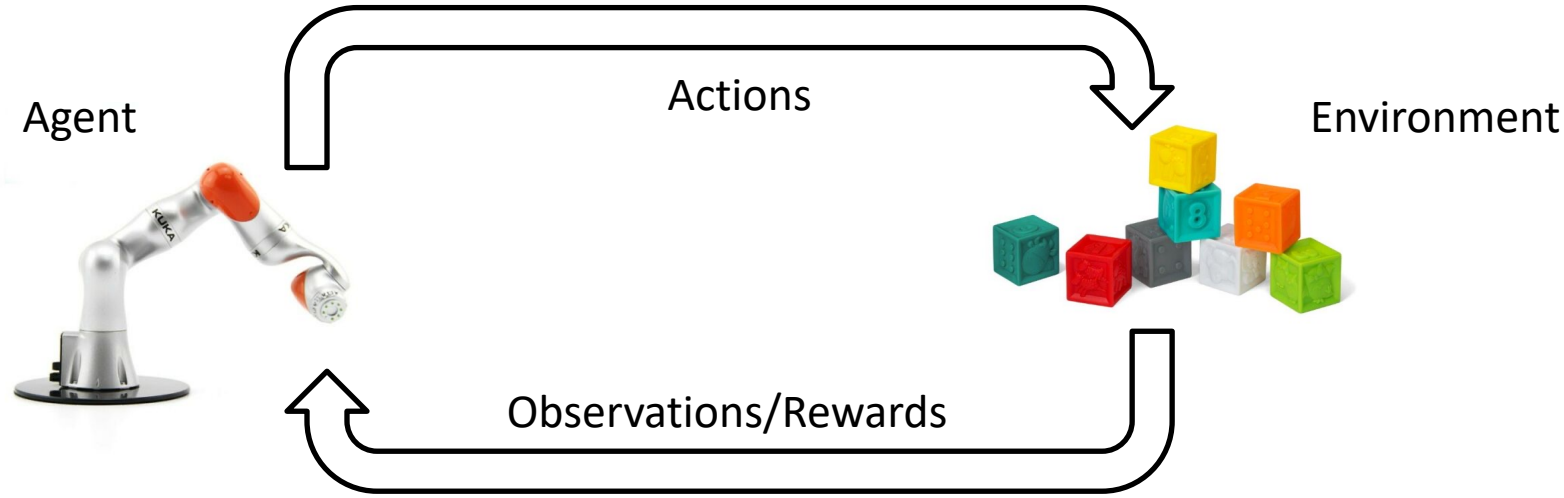


What is reinforcement learning?

Remove assumption for known environment model, learn directly from data



Modeling assumptions



Agent has:

- Sensing
- Actuation

Environment accepts:

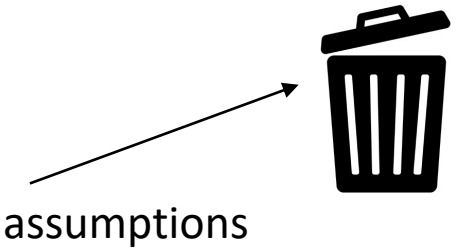
- Actions from agent

Produces

- Observations for sensors(usually unknown)

Why would you do this?

Remove assumption for known environment model, learn directly from data



Pros:

1. Continual improvement on deployment
2. Avoid significant modeling assumptions and simulation
3. Scale across tasks easily!

Cons:

1. Potentially prohibitive data requirements
2. Sometimes unstable, lacks guarantees
3. Poor extrapolation

Promising and useful tool in unstructured, dynamic environments

How does Optimal Control relate to Reinforcement Learning

Optimal Control

$$\min_{x,u} \int_0^X L(t, x(t), u(t)).dx$$

$$\text{w.r.t}$$

$$x'(t) = f(x(t), u(t))$$

- Often continuous time
- Minimize cost
- Known model and cost function (often differentiable/convex)

Reinforcement Learning

$$\max_{\theta} \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} r(s_t, a_t) \right]$$

- Often discrete time
- Maximize reward
- Unknown model and cost function (and non-differentiable)

Is it useful for robotics?

Robots that get better over time, adapting to new **environments**

iRobot

Is it useful for robotics?

Robots that get better over time, adapting to new **objects**

Kalashnikov et al 18

Is it useful for robotics?

Robots that get better over time, adapting to new **terrains**

Miki et al 22

Is it useful for robotics?

Robots that get better over time, adapting to new **tasks**

Gupta et al 21

Why should we care about RL?

Allows agents to continue improving/adapting on deployment with minimal human effort

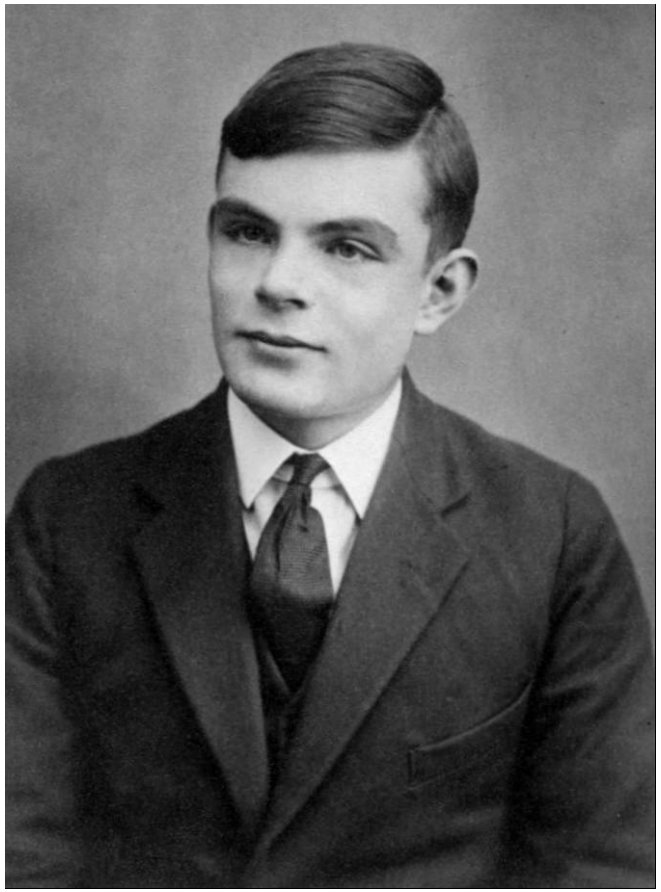
Locomotion

Manipulation

Agents can overfit to domains at test time rather than per-domain human design

Why should we care about RL?

Hypothesis: By designing algorithms that can improve themselves, we can reach fully intelligent systems



"Instead of trying to produce a programme to simulate the adult mind, why not rather try to produce one which simulates the child's? If this were then subjected to an appropriate course of education one would obtain the adult brain" – Alan Turing

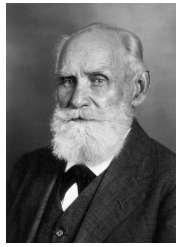
⇩

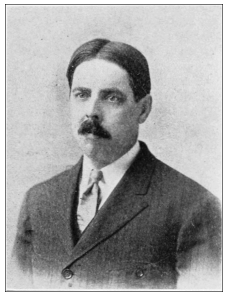
Rather than try to directly replicate behaviors, try to replicate adaptative learning mechanisms

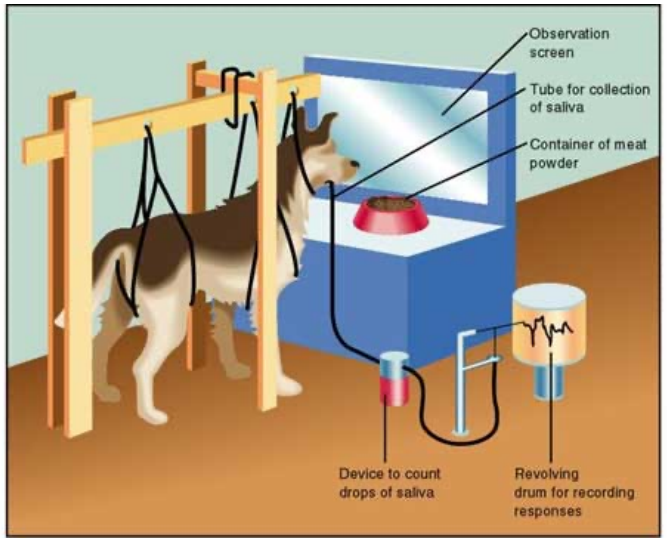
A Little History on Reinforcement Learning

Two distinct threads converged to give rise to modern RL

Animal Psychology







Optimal Control

$$\min_{x,u} \int_0^x L(t, x(t), u(t)).dx$$

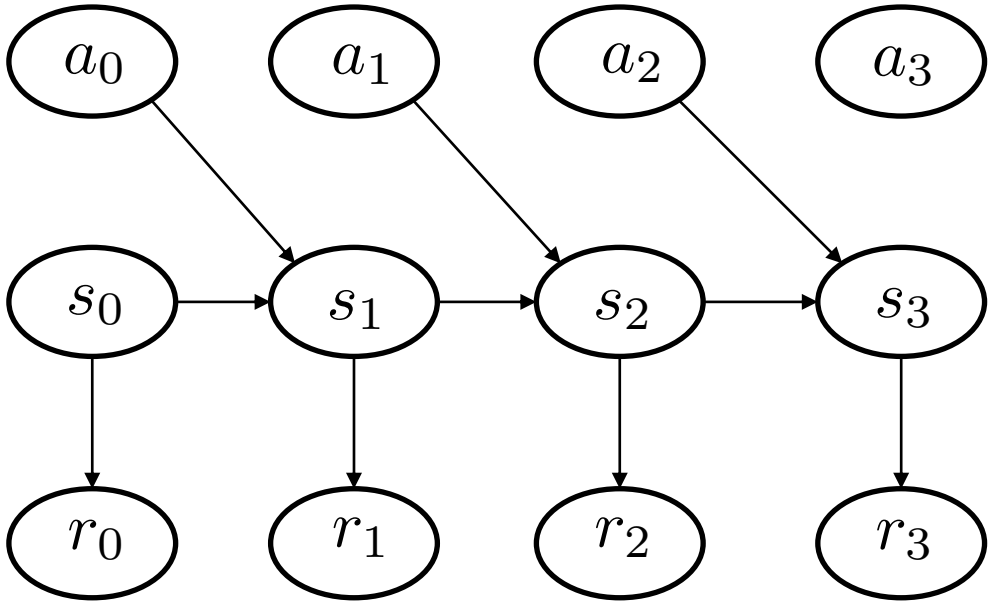
$$\text{w.r.t}$$

$$x'(t) = f(x(t), u(t))$$

Ideas from temporal difference learning/dynamic programming united these fields!

Sutton and Barto

Markov Decision Processes



States: $\mathcal{S}$

Actions: $\mathcal{A}$

Rewards: $\mathcal{R}$

Transition Dynamics - $p(s_{t+1}|s_t, a_t)$

Markov property $$p(s_1, s_2, s_3) = p(s_3|s_2)p(s_2|s_1)p(s_1)$$

Trajectory $$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T, a_T, r_T)$$

MDPs to the Real World

Task: Place kettle in sink

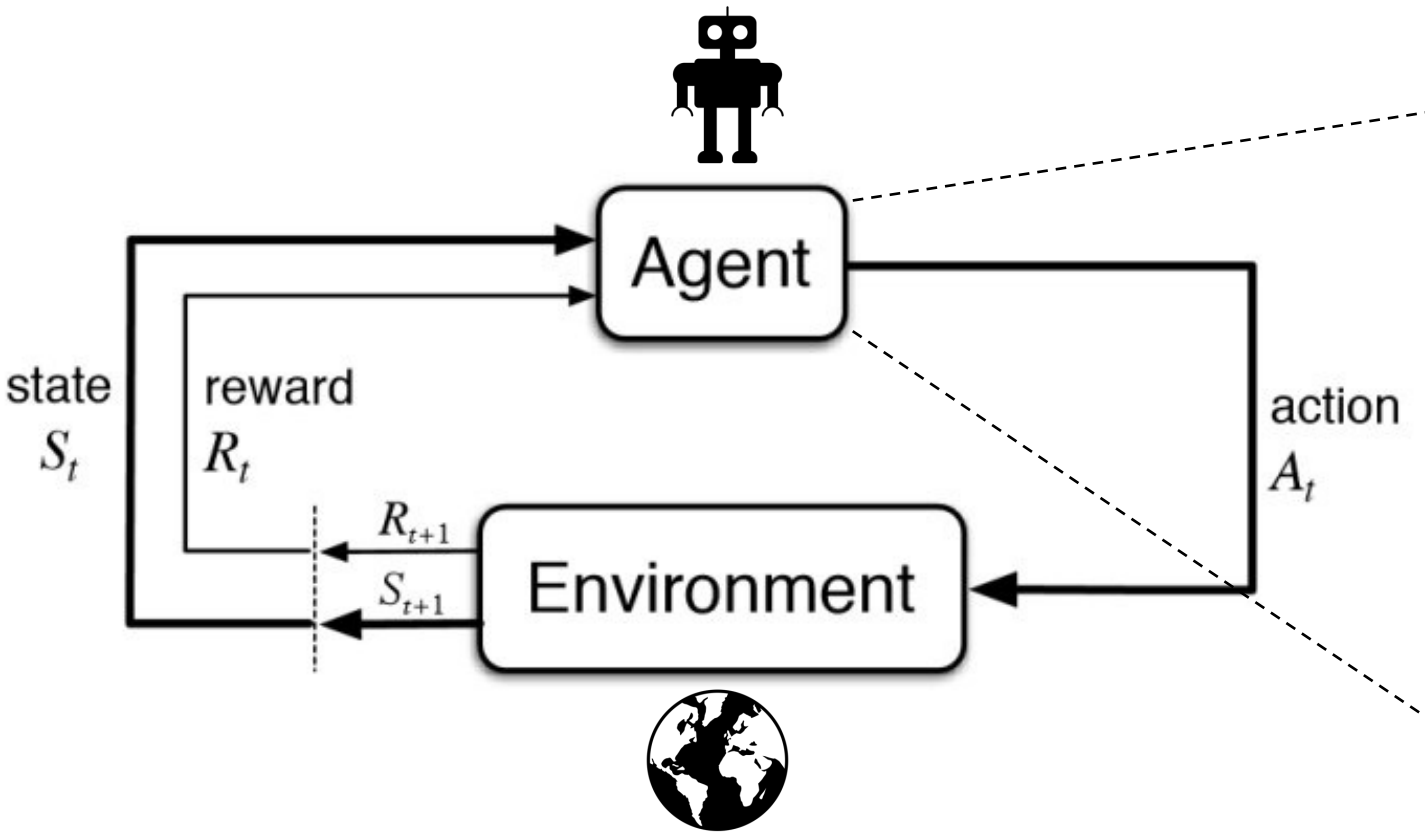
State: Camera Images / Joint Encoders

Action: Joint torques/velocities

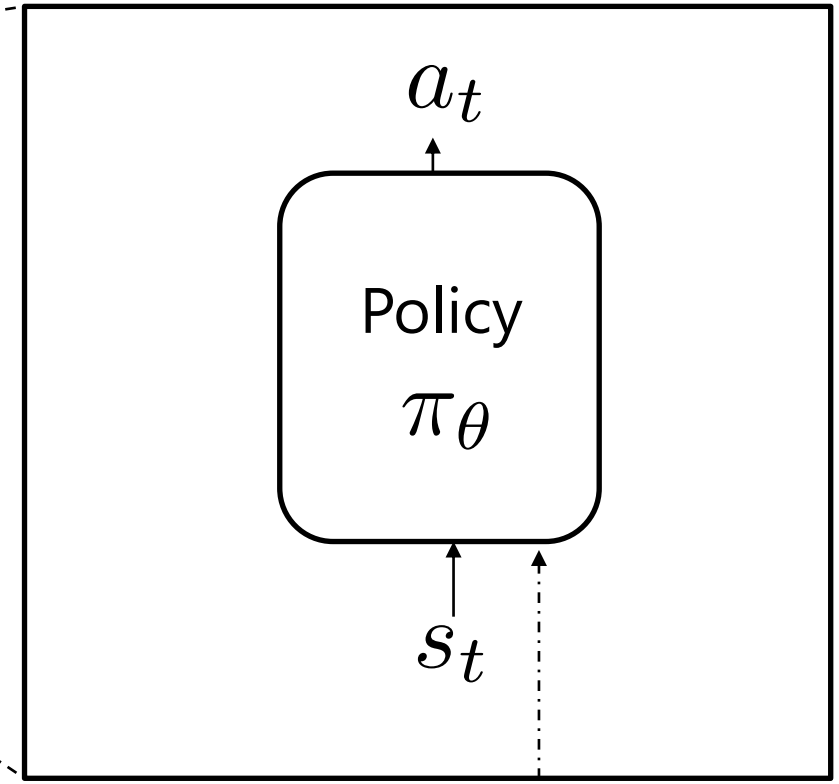
Reward: Distance from kettle to sink

Transition: World physics

Reinforcement Learning Formalism



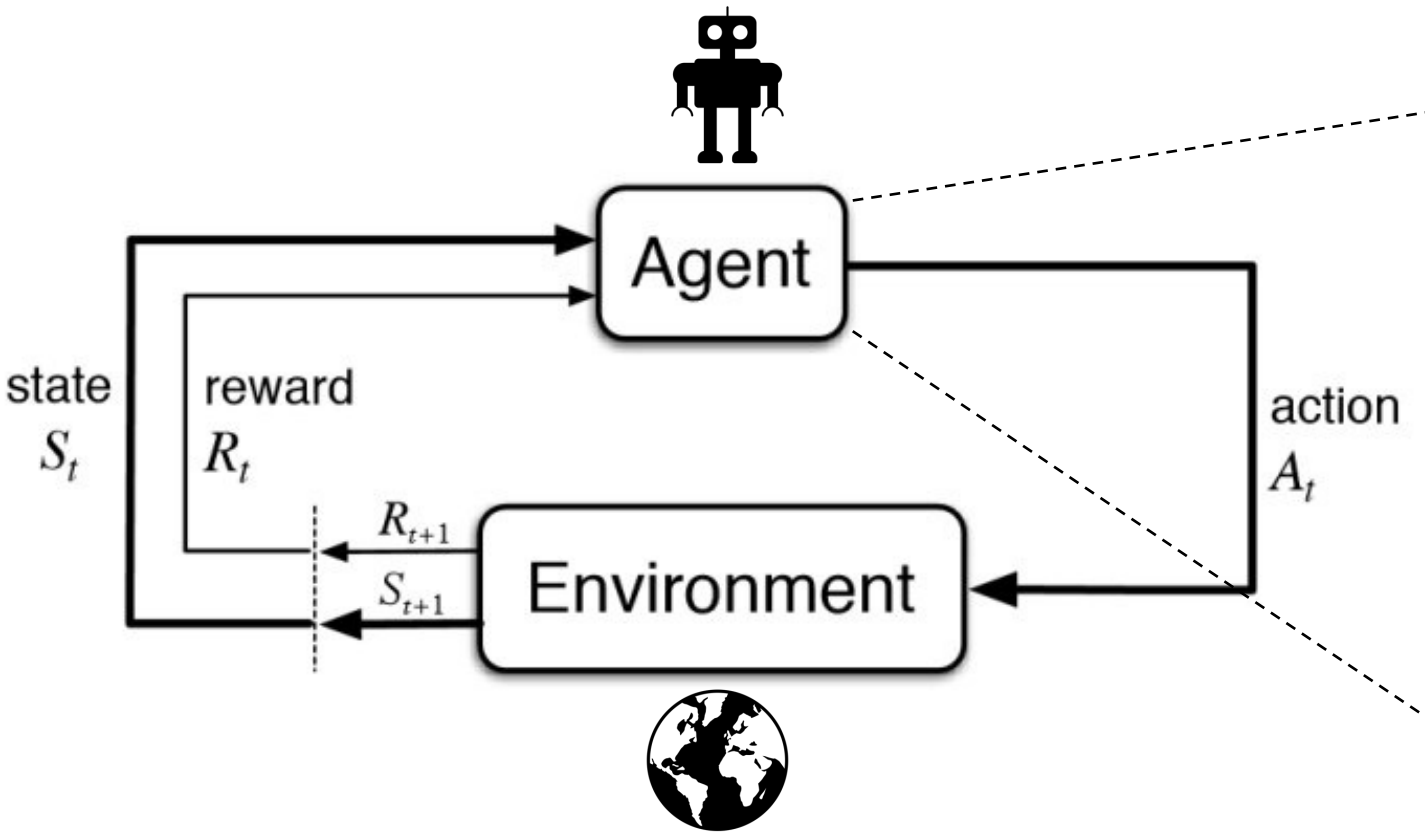
Rules for choosing actions



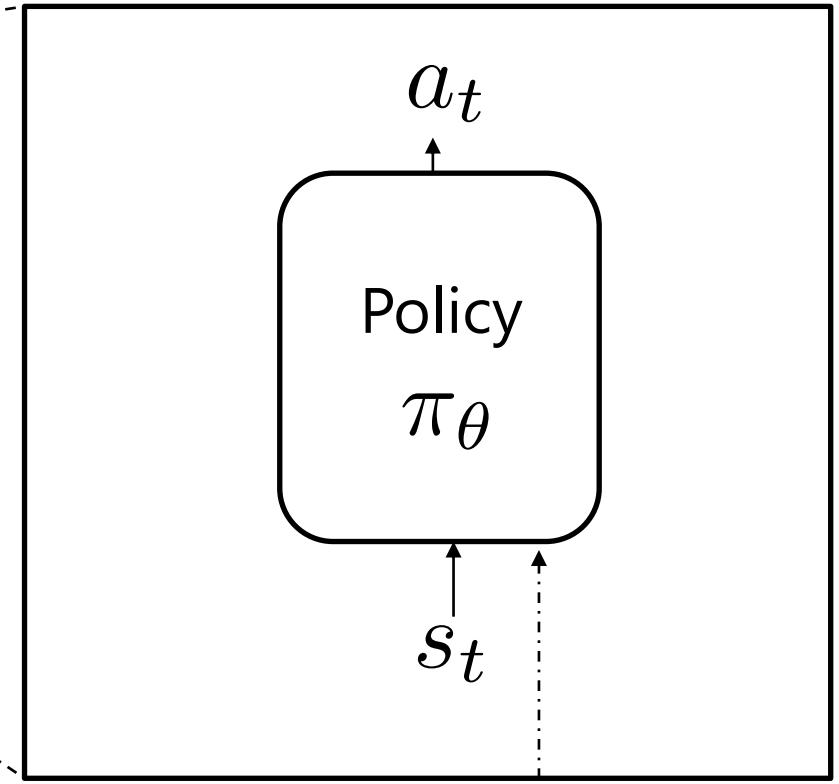
Maximize the sum of expected rewards under policy

Needs to be learned

Reinforcement Learning Formalism



Rules for choosing actions



$$\max_\theta \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} r(s_t, a_t) \right]$$

Trajectory sampled using policy

Needs to be learned

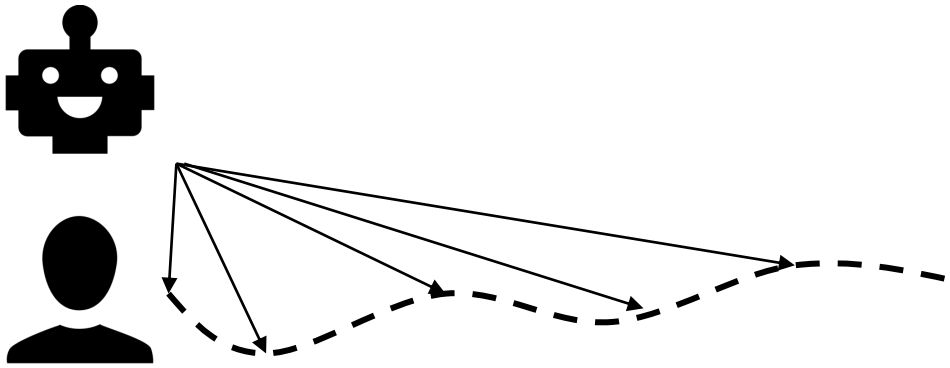
Why is this not just supervised learning?

Supervised Learning

$$\max_\theta \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\log \hat{p}_\theta(y|x)\right]$$

Sampling from expert

$D_{\mathrm{KL}}(p^*||p_\theta)$ IID

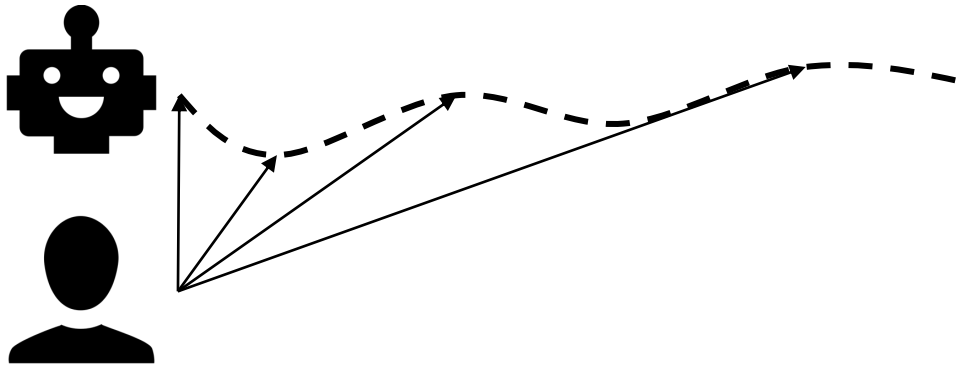


Reinforcement Learning

$$\max_\theta \mathbb{E}_{\tau\sim\pi_\theta} \left[\sum_{t=0}^{T} r(s_t, a_t)\right]$$

Sampling from policy

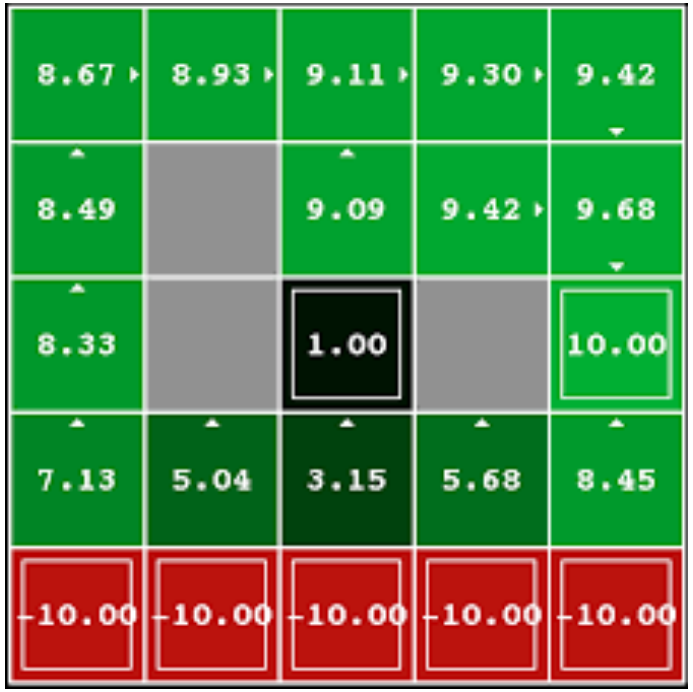
$D_{\mathrm{KL}}(p_\theta||p^*)$ Non-IID



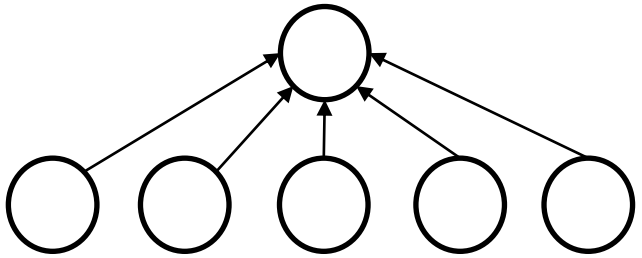
Main thing to learn - Policies

Policies are mappings from states to optimal actions

Tabular

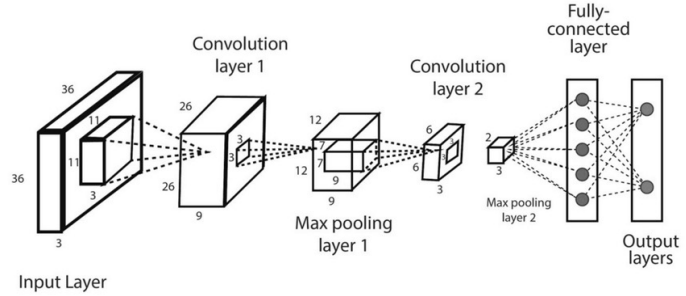


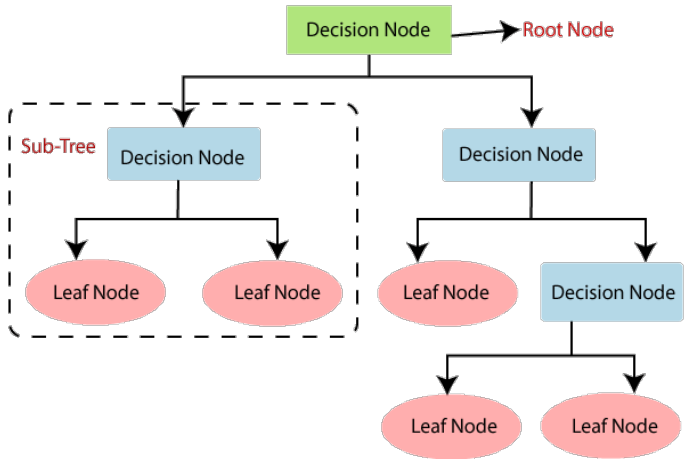
Linear



$$\pi(a|s) = \langle \phi(s,a), w \rangle$$

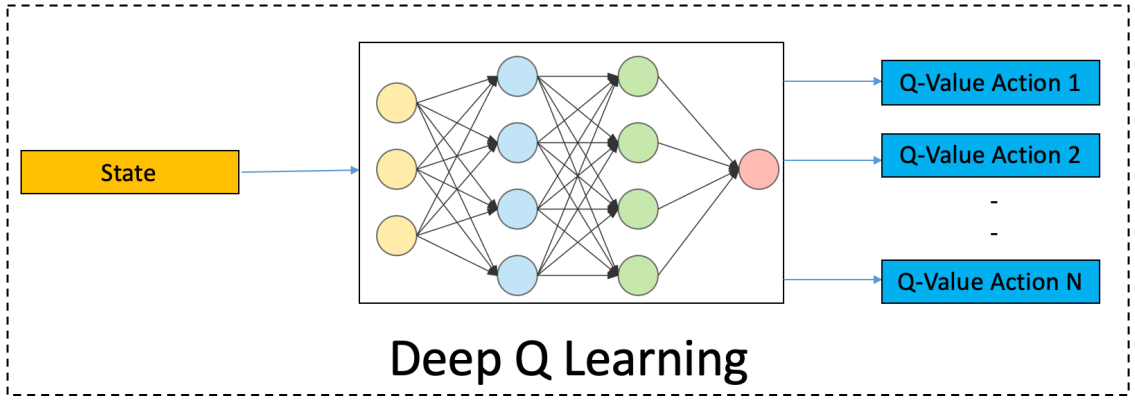
Arbitrary function approx



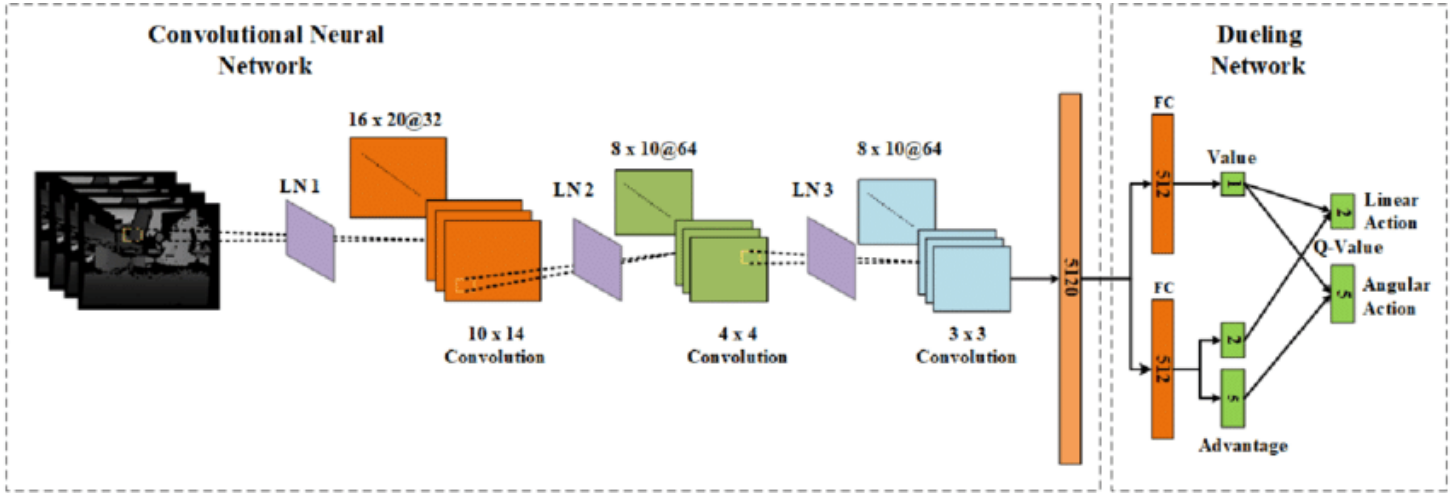


Ok so where does deep learning fit in?

Avoids expensive hand-design for adaptive agents, learn end-to-end: sensors → actions



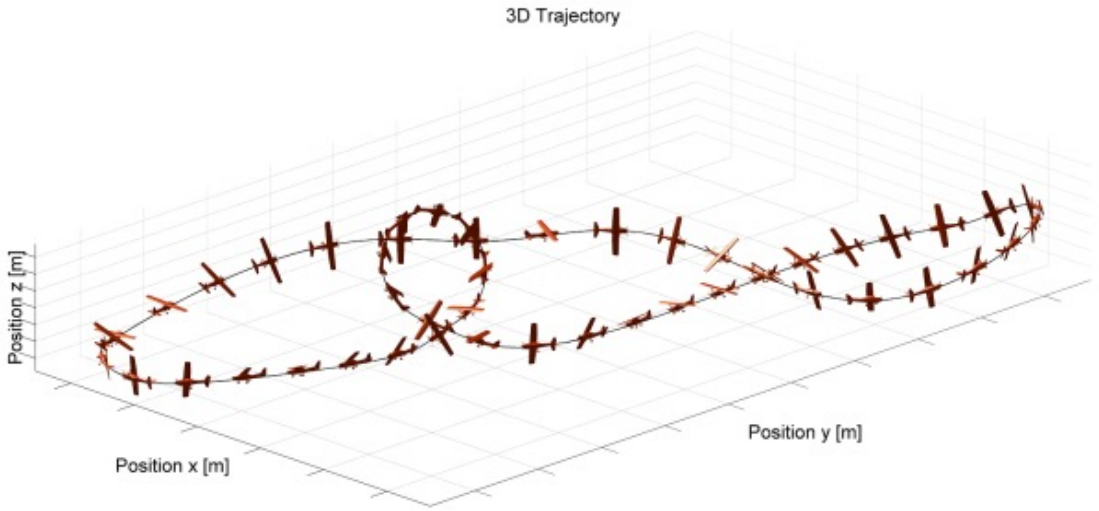
Policies/Q-values/model are represented as deep neural networks



Leads to non-trivial challenges in learning and optimization!

Where is Reinforcement Learning not useful?

Not the right call for very safety-critical, repetitive applications

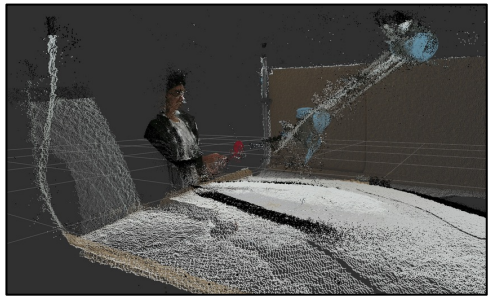


Where is Reinforcement Learning “potentially” useful?

Domains which have high diversity, yet relatively cheap autonomous data collection

But these domains are not as simple as just running RL algorithms!

A Closer Look at the Pipeline for Robotic Learning



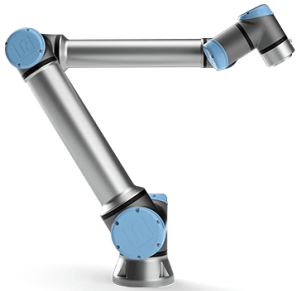
State Estimation

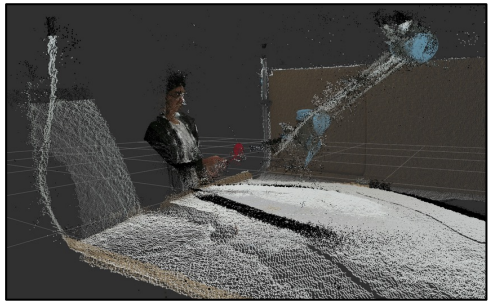
Modeling and Prediction

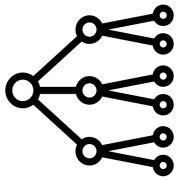
High-level planning

Low-level planning

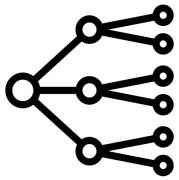
Low-level control

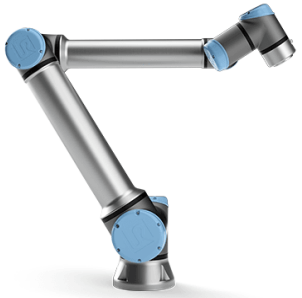






End to end policy – perception + control





Why might we not want to do this?

Modules compensate for each other

Avoids hand-designing and supervising interfaces

Often more performant/less biased

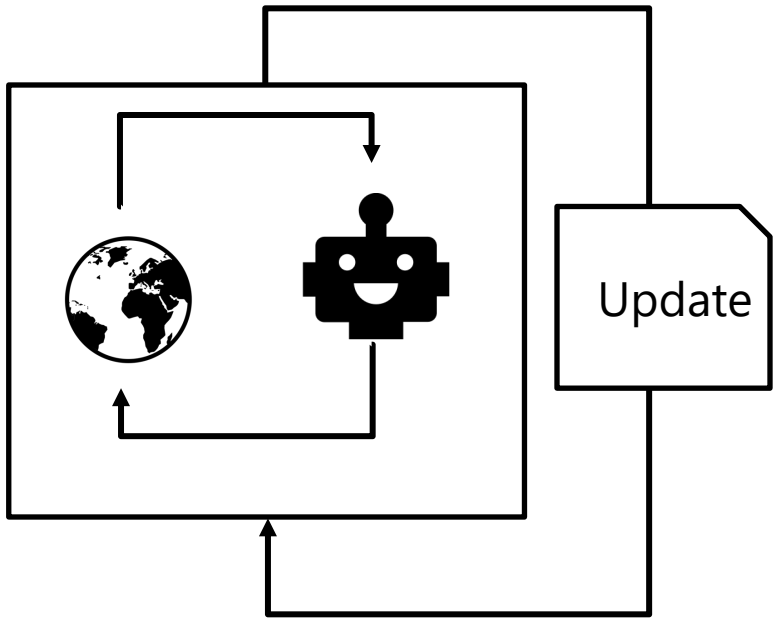
Lack of Interpretability

Lack of Reusability

Often data inefficient

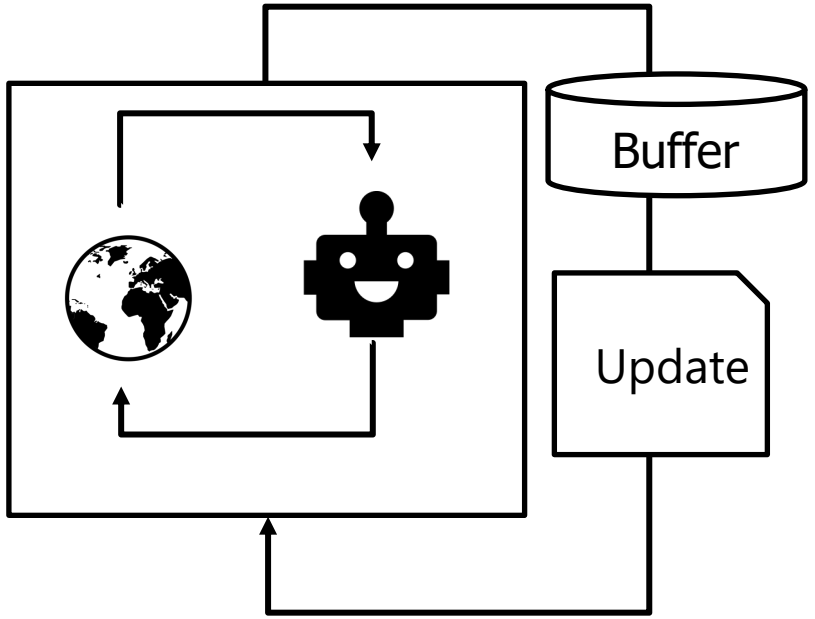
Learning Algorithms for Robotics

On-Policy Algorithms



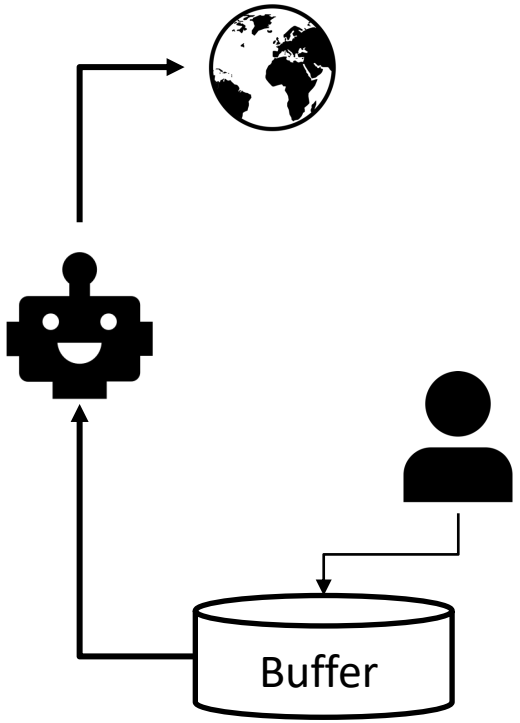
Simple, performant,
Data inefficient

Off-Policy Algorithms



Data-efficient,
sometimes unstable

Imitation Learning



Performant, efficient, but
compounding error and
expensive data collection

Recap: Course Overview