Overview of Robot Decision Making

Prof. Yuke Zhu

Fall 2023

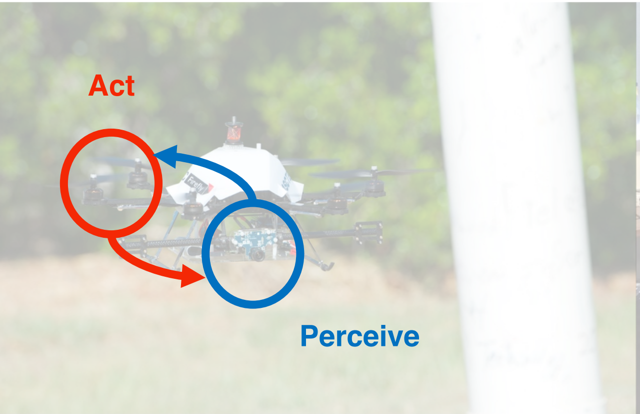
Logistics

- Grades for project proposals are out.
- PyTorch tutorial next Wednesday
  - Over Zoom during Soroush’s office hours
  - Video recording will be available on Canvas

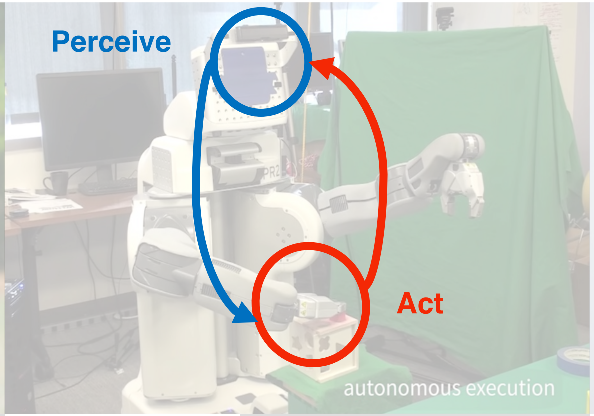
Today's Agenda

- What is Robot Decision Making?
- Mathematical Framework of Sequential Decision Making
- Learning for Decision Making
    - Online: model-free vs. model-based RL
    - Offline: imitation as supervised learning, offline RL
- Research Frontiers
    - Human in the loop, Learning to learn, Task and motion planning…

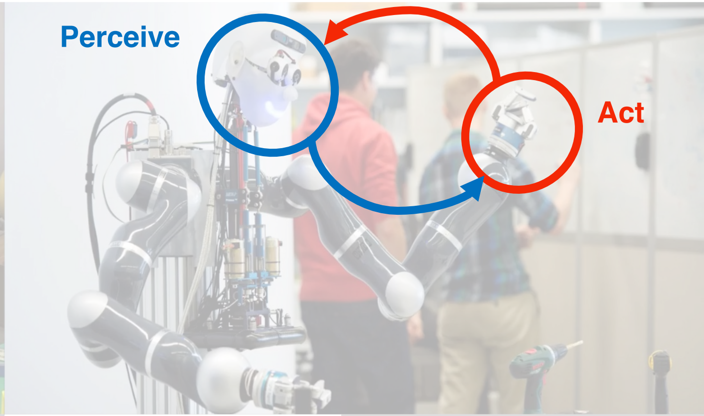
Robot Learning is to close the perception-**action** loop.



[Sa et al. IROS 2014]



[Levine et al. JMLR 2016]

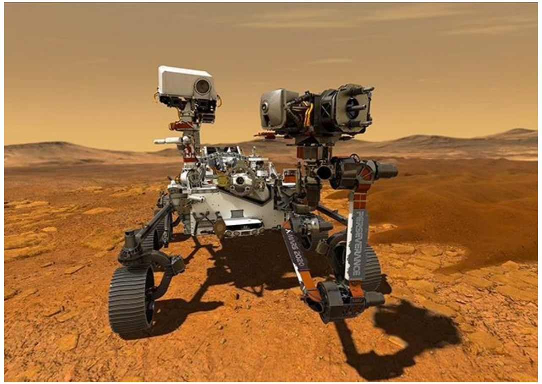


[Bohg et al. ICRA 2018]

What is Robot Decision Making?

Choosing the actions a robot performs in the physical world…

Assistive Robots (Companions)

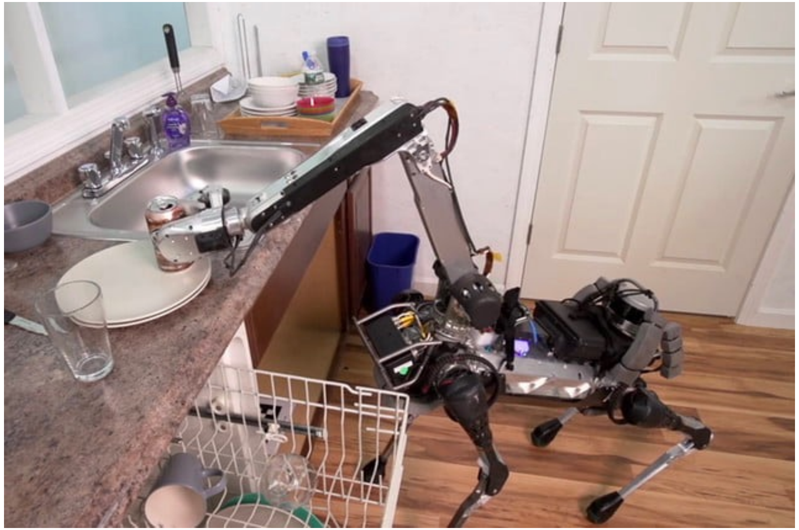


Outer Space (Explorers)

Autonomous Driving (Transporters)

What is Robot Decision Making?

Choosing the actions a robot performs in the physical world…

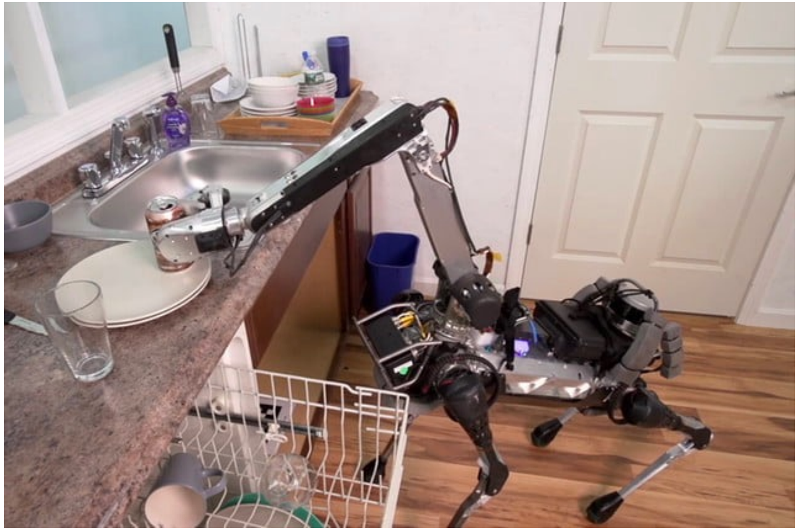


[Source: Boston Dynamics]

- Behaviors can't be easily programmed
- Imperfect sensing and actuation
- Safety and robustness under uncertainty

Robot Decision Making vs. Playing Games

Robot decision making is **embodied**, **active**, and **environmentally situated**.

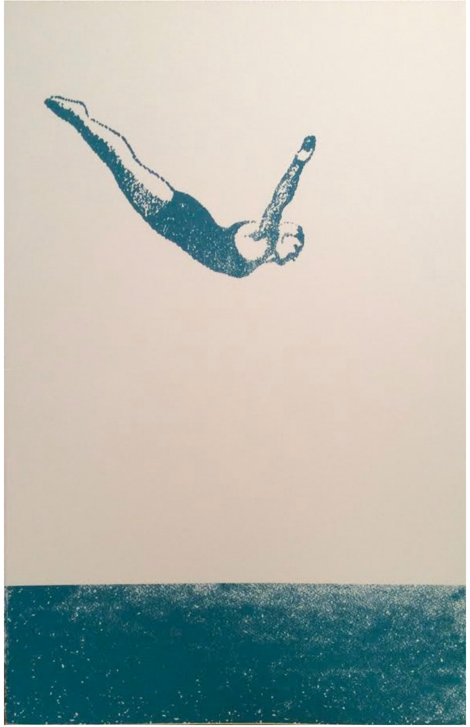


[Source: Boston Dynamics]

[Source: DeepMind's AlphaGo]

Before We Dive In…

- This lecture is intended to provide a **high-level, bird-eye view** on (robot) decision making.
- The goal is not to go through all technical details:
  - We will re-visit them through paper reading in the following weeks.
  - Study the parts that you are less familiar with from online resources.
- Take related courses and read textbooks to learn this subject in depth (see the last slide).



Mathematical Framework: Markov Decision Processes

A **Markov Decision Process** is defined by a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

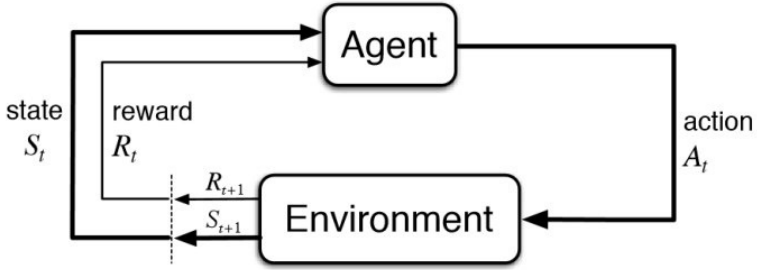
$\mathcal{S}$: state space $(s_t \in \mathcal{S})$

$\mathcal{A}$: action space $(a_t \in \mathcal{A})$

$\mathcal{P}$: transition probability $\mathcal{P}^a_{ss'} = \Pr[s_{t+1} \mid s_t, a_t]$

$\mathcal{R}$: reward function $r(s, a) = \mathbb{E}[r_{t+1} | s = s_t, a = a_t]$

γ : a discount factor $\gamma \in [0, 1]$



Mathematical Framework: Markov Decision Processes

A **Markov Decision Process** is defined by a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

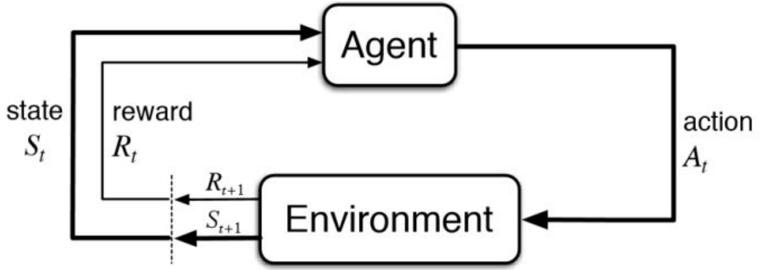
$\mathcal{S}$: state space $(s_t \in \mathcal{S})$

$\mathcal{A}$: action space $(a_t \in \mathcal{A})$

$\mathcal{P}$: transition probability $\mathcal{P}^a_{ss'} = \Pr[s_{t+1} \mid s_t, a_t]$

$\mathcal{R}$: reward function $r(s, a) = \mathbb{E}[r_{t+1} | s = s_t, a = a_t]$

γ : a discount factor $\gamma \in [0, 1]$




```
for i in range(1000):
    action = np.random.randn(env.robots[0].dof) # sample random action
    obs, reward, done, info = env.step(action)  # take action in the environment
    env.render()  # render on display
```

Mathematical Framework: Markov Decision Processes

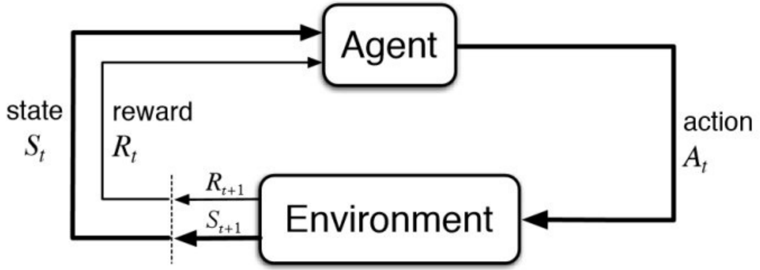
A **Markov Decision Process** is defined by a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

A **policy** maps states to actions $\pi : \mathcal{S} \rightarrow \mathcal{A}$

Goal of (robot) decision making

Choose policy that maximizes cumulative reward

$$\pi^* = \arg\max_{\pi} \mathbb{E}[\sum_{t \geq 0} \gamma^t r(s_t, \pi(s_t))]$$



Mathematical Framework: Markov Decision Processes

We define two functions given a policy π

Value function: the expected cumulative discounted reward when acting according to the policy from a given state

$$V^\pi(s) = \mathbb{E}\Big[\sum_{t=0} \gamma^t r(s_t, \pi(s_t)) \,|\, s_0 = s\Big]$$

Q function: the expected cumulative discounted reward when acting according to the policy from a given state **and taking a given action**

$$Q^\pi(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^\pi(s')$$

Mathematical Framework: Markov Decision Processes

We define two functions given a policy π

V*(s) = max_a Q*(s, a)   * means optimal

Value function: the expected cumulative discounted reward when acting according to the policy from a given state

$$V^\pi(s) = \mathbb{E}[\sum_{t=0} \gamma^t r(s_t, \pi(s_t)) \mid s_0 = s]$$

Q function: the expected cumulative discounted reward when acting according to the policy from a given state **and taking a given action**

$$Q^\pi(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^\pi(s')$$

Mathematical Framework: Markov Decision Processes

We define two functions given a policy π

Value function: the expected cumulative discounted reward when acting according to the policy from a given state

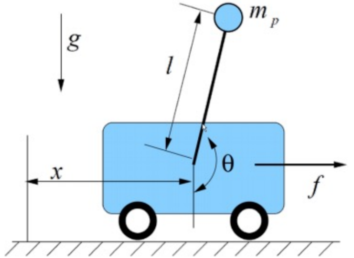
$$V^\pi(s) = \mathbb{E}\big[\sum_{t=0} \gamma^t r(s_t, \pi(s_t)) \,|\, s_0 = s\big]$$

Q function: the expected cumulative discounted reward when acting according to the policy from a given state **and taking a given action**

π*(a|s) = arg max_a Q*(s, a)

$$Q^\pi(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^\pi(s')$$

Solving MDPs with Known Models



When we know the model of the MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

Use ideas from

Dynamic Programming

Value Iteration

1. Estimate optimal value function
2. Compute optimal policy from optimal value function

Initialize $V(s)$ to arbitrary values
Repeat
  For all $s \in S$
    For all $a \in \mathcal{A}$
      $Q(s,a) \leftarrow E[r|s,a] + \gamma \sum_{s' \in S} P(s'|s,a)V(s')$
    $V(s) \leftarrow \max_a Q(s,a)$
Until $V(s)$ converge

using model

Policy Iteration

1. Start with random policy
2. Iteratively improve it until convergence to optimal policy

Initialize a policy π' arbitrarily
Repeat
  $\pi \leftarrow \pi'$
  Compute the values using π by
    solving the linear equations
      $V^{\pi}(s) = E[r|s,\pi(s)] + \gamma \sum_{s' \in S} P(s'|s,\pi(s))V^{\pi}(s')$
  Improve the policy at each state
    $\pi'(s) \leftarrow \arg\max_a (E[r|s,a] + \gamma \sum_{s' \in S} P(s'|s,a)V^{\pi}(s'))$
Until $\pi = \pi'$

using model

Solving MDPs with Known Models

When we know the model of the MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

Optimal Control (LQR)

Assume **linear transitions** and **quadratic reward functions**

A special case: exact solution π^* is easily to solve

Linear transition $s_{t+1} = A_t s_t + B_t a_t$

Quadratic reward $r(s_t, a_t) = -s_t^\top U_t s_t - a_t^\top W_t a_t$

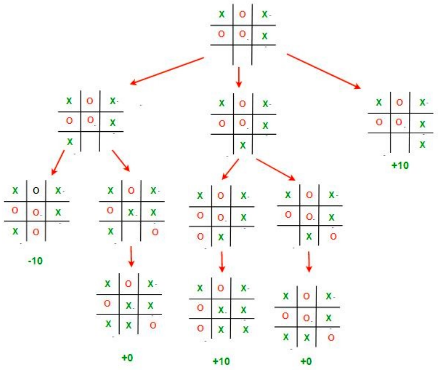
always negative

Extensions: LQG (Gaussian noise), iLQR (non-linear transition)

Sampling-based Planning

Evaluate outcomes of sampled actions with models

Choose the action that leads to the best (predicted) outcome



Monte-Carlo Tree Search (MCTS) for Tic-Tac-Toe

Solving MDPs with Learned Models

A key role of learning in model-based approaches

Model is known in restricted domains: **games**, **simulated robots**, **simple mechanics**

When model is not known, we can **learn the model from data**.

agent's experience

$$\tau = \{(s_i, a_i, r_i) \,|\, i = 0, \ldots, H\}$$

learned model

$$\hat{\mathcal{P}}$$

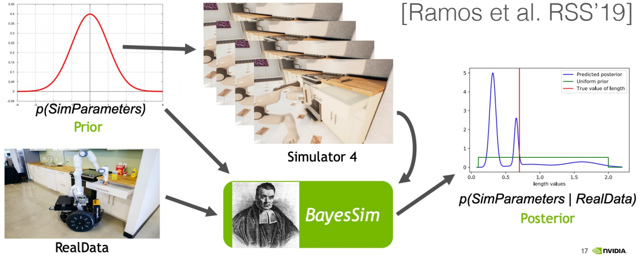
Can be represented by **Gaussian Processes**, **Neural Networks**, **GMMs**, etc.

Use **planning** and **optimization** methods for known models (previous two slides)

model-based RL

Solving MDPs with Learned Models

System Identification



[Ramos et al. RSS'19]

Model structure is known (e.g., simulator). We tune some model parameters (e.g., mass and friction).

$$\Pr[\mu \mid \mathcal{D}]$$

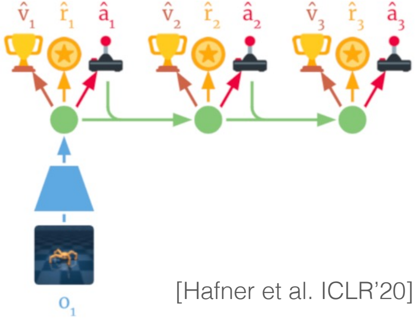
Sensor-Space Model

[Finn et al. ICRA'17]

Predicting future raw sensory data

$$f(s_{t+1} \mid s_t, a_t)$$

Latent-Space Model



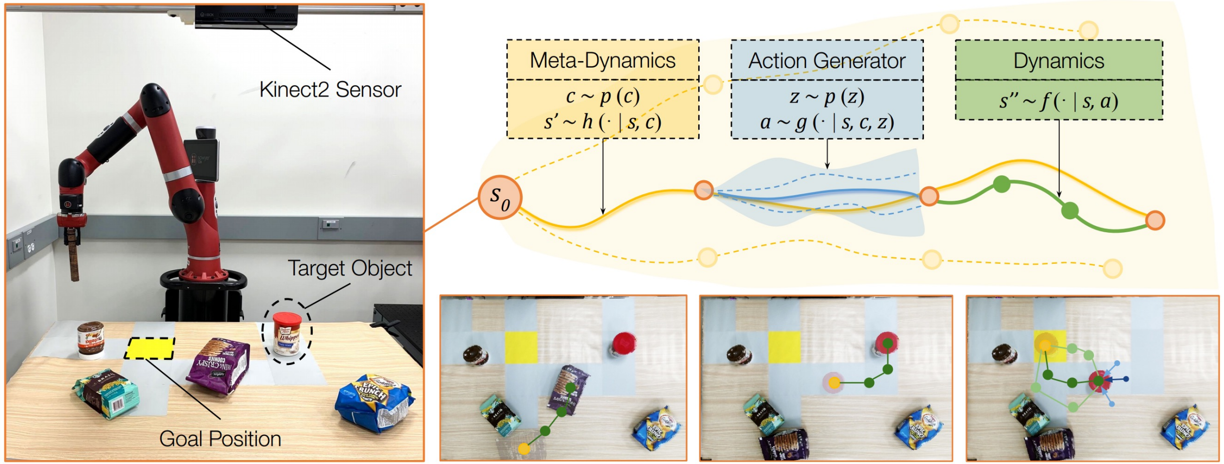
[Hafner et al. ICLR'20]

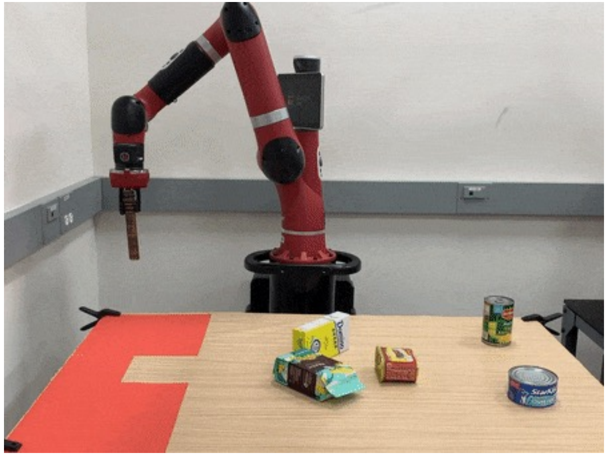
Learn behavior in imagination

Predicting future latent state

$$h_t = g(s_t) \quad f(h_{t+1} \mid h_t, a_t)$$

Examples of Model-Based Reinforcement Learning





"Dynamics Learning with Cascaded Variational Inference for Multi-Step Manipulation." Fang, Zhu, Garg, Savarese, Fei-Fei, CoRL 2019

Solving MDPs without Models

When model is unknown and hard to estimate, we can **learn policy directly** from the agent's trajectories τ from interacting with an MDP.

agent's experience

$$\tau = \{(s_i, a_i, r_i) \,|\, i = 0, \ldots, H\}$$

model-free RL

optimal policy

$$\pi^* = \arg\max_{\pi} \mathbb{E}[\sum_{t \geq 0} \gamma^t r(s_t, \pi(s_t))]$$

Solving MDPs without Models

Model-free

Value-based RL

Week 6 Tue

Deep Q-Network (DQN):

Represent Q with neural networks

Optimality condition (Bellman equation)

$$Q^*(s,a) = r(s,a) + \gamma\,\mathbb{E}_{s'|s,a}[\max_{a'} Q^*(s',a')]$$

$$\pi^*(a|s) = \arg\max_{a'} Q^*(s,a')$$

Q-learning rule (temporal different learning)

$$Q(s_t,a_t) \leftarrow (1-\alpha)\cdot \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}}\right)}^{\text{learned value}}$$

Solving MDPs without Models

policy parameterized by θ

Total reward of trajectory

Objective Function $J(\theta) = \mathbb{E}_{\tau \sim p(\tau|\theta)}[r(\tau)]$

trajectories under policy π_θ

Model-free

Policy-Gradient RL

policy gradient theorem

Week 6 Tue

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi[Q^\pi(s,a)\nabla_\theta \log \pi_\theta(a|s)]$$

Different ways of computing the Q values lead to different PG variants: Monte-Carlo estimates (REINFORCE), learning value functions (Actor-Critic)

Solving MDPs without Models

Model-free

Policy-Gradient RL

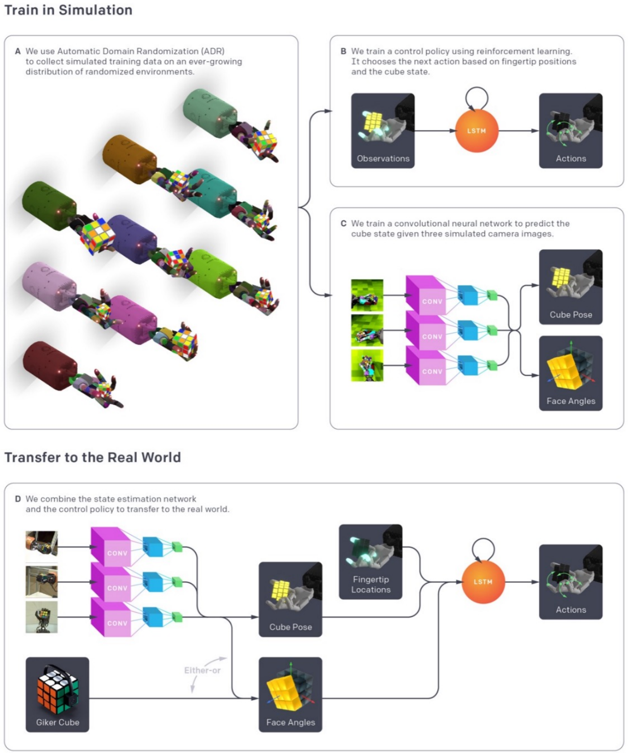
Week 6 Tue

$$
\begin{aligned}
\nabla \mathbb{E}_\pi \left[r(\tau) \right] &= \nabla \int \pi(\tau) r(\tau) d\tau \\
&= \int \nabla \pi(\tau) r(\tau) d\tau \\
&= \int \pi(\tau) \nabla \log \pi(\tau) r(\tau) d\tau
\end{aligned}
$$

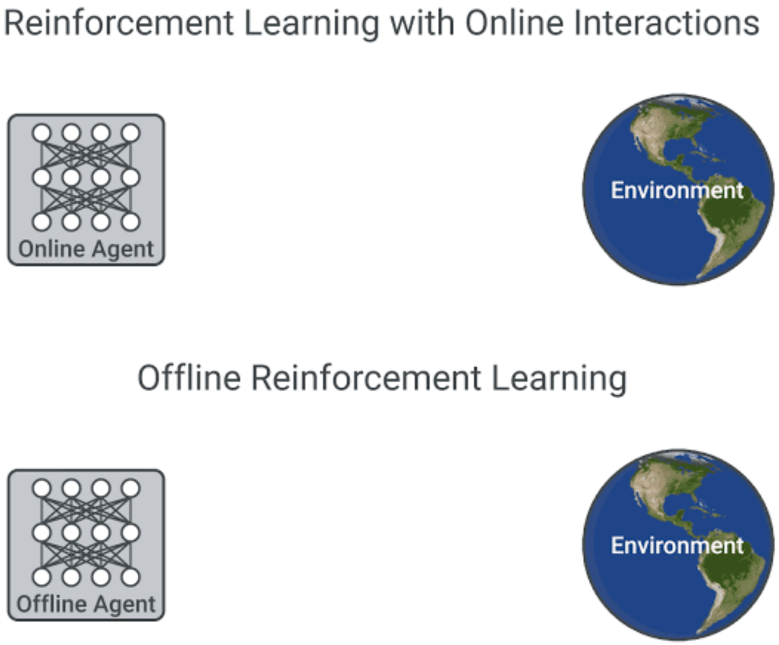
$$
\nabla \mathbb{E}_\pi \left[r(\tau) \right] = \mathbb{E}_\pi \left[r(\tau) \nabla \log \pi(\tau) \right]
$$

Examples of Model-Free Reinforcement Learning

“Solving Rubik’s Cube with a robot hand.” OpenAI 2019



Online vs Offline Reinforcement Learning



The agent **interact with the environment** to collect data directly in the real world or through simulation.

The agent only uses **previously collected data**. It does not **interact with the environment**.

[Source: offline-rl.github.io]

Solving MDPs without Models

Model-free Value-based RL

- ✓ Can learn Q function from any interaction data, not just trajectories gathered using the current policy (“off-policy” algorithm)
- ✓ Relatively data-efficient (can reuse old interaction data)
- ✗ Need to optimize over actions: hard to apply to continuous action spaces
- ✗ Optimal Q function can be complicated, hard to learn

Model-free Policy-Gradient RL

- ✓ Learns policy directly – often more stable
- ✓ Works for continuous action spaces
- ✗ Needs data from current policy to compute policy gradient (“on-policy” algorithm) – data inefficient
- ✗ Gradient estimates can be very noisy

Mathematical Framework: Markov Decision Processes

Reinforcement learning optimizes policy by **trial and error** in an MDP.

Goal: To maximize the long-term rewards

$\mathcal{S}$: state space $(s_t \in \mathcal{S})$

$\mathcal{A}$: action space $(a_t \in \mathcal{A})$

$\mathcal{P}$: transition probability $\mathcal{P}^a_{ss'} = \Pr[s_{t+1} \mid s_t, a_t]$

$\mathcal{R}$: reward function $r(s, a) = \mathbb{E}[r_{t+1} | s = s_t, a = a_t]$

γ : a discount factor $\gamma \in [0, 1]$

Fundamental assumption of RL: **reward function**

Mathematical Framework: Markov Decision Processes

Imitation learning optimizes policy by **imitating the expert** in an MDP.

Goal: To match the behavioral distributions

$\mathcal{S}$: state space $(s_t \in \mathcal{S})$

$\mathcal{A}$: action space $(a_t \in \mathcal{A})$

$\mathcal{P}$: transition probability $\mathcal{P}^a_{ss'} = \Pr[s_{t+1} \mid s_t, a_t]$

~~$\mathcal{R}$: reward function $r(s,a) = \mathbb{E}[r_{t+1} | s = s_t, a = a_t]$~~

~~γ : a discount factor $\gamma \in [0,1]$~~

$\mathcal{D}$: set of demonstrations drawn from the expert policy π_E

Mathematical Framework: Markov Decision Processes

Imitation learning optimizes policy by **imitating the expert** in an MDP.

Goal: To match the behavioral distributions

Two basic ideas

- Direct estimation of the expert policy from expert data (behavioral cloning)
- Reconstruct a reward function (inverse RL) and then learn a policy from the reward (RL)

Imitation as Supervised Learning

Idea 1: Direct estimation of the expert policy from expert data

Week 7 Tue

This can be cast as a **supervised learning** problem, called **behavioral cloning**

action from expert policy

$$\pi^* = \arg\min_{\pi} \sum_{s_t \in \mathcal{D}} L\Big(\pi(s_t), \pi_E(s_t)\Big)$$

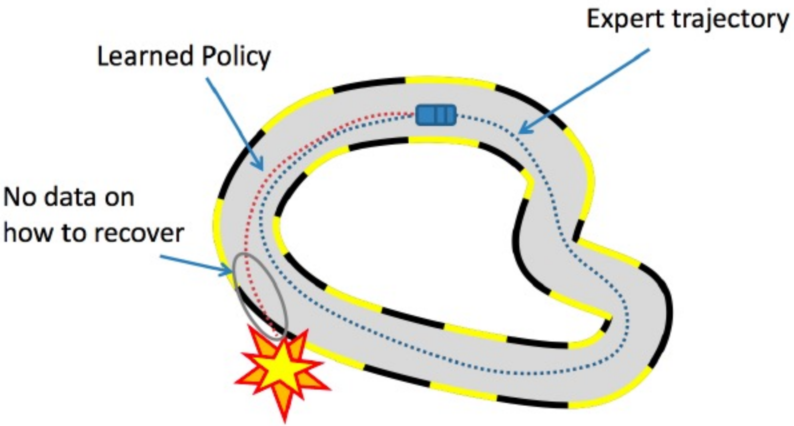
Distance metric that measures the discrepancy between the expert action and the policy action (e.g., KL-divergence)

Imitation as Supervised Learning

Idea 1: Direct estimation of the expert policy from expert data

Week 7 Tue

This can be cast as a **supervised learning** problem, called **behavioral cloning**

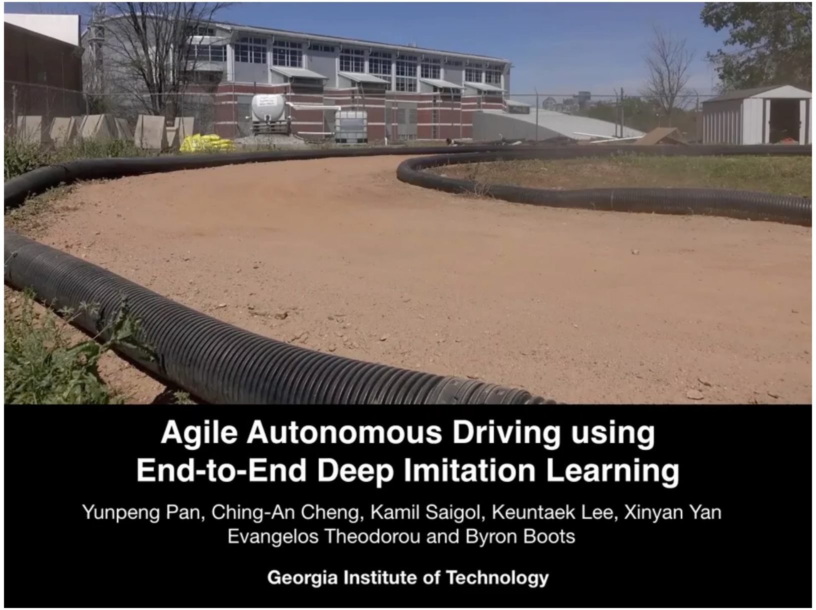


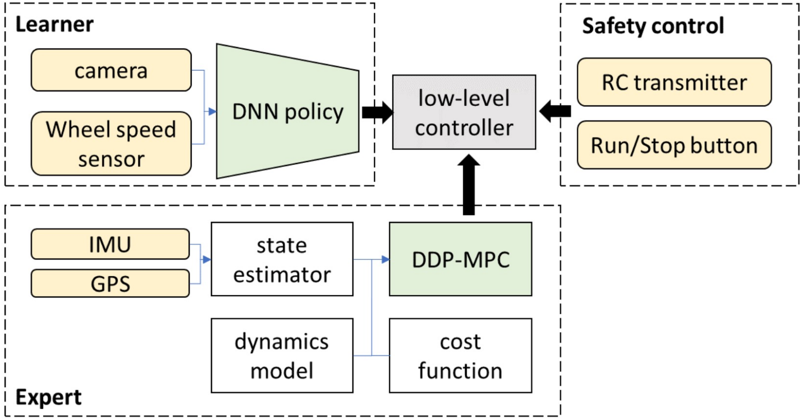
What can go wrong?

compounding errors

How to fix: Asking expert for more data (DAgger)

Examples of Supervised Imitation Learning





"Agile Autonomous Driving using End-to-End Deep Imitation Learning"
Pan, Cheng, Saigol, Lee, Yan, Theodorou, Boots. RSS 2018

Inverse Reinforcement Learning

Idea 2: Reconstruct a reward function and then learn a policy from the reward

Solving full-fledged RL in the inner loop

- Collect expert demonstrations: $D = \{\tau_1, \tau_2, \ldots, \tau_m\}$
- In a loop:
  - Learn reward function: $r_\theta(s_t, a_t)$
  - Given the reward function r_θ, learn π policy using RL
  - Compare π with π^* (expert's policy)
  - STOP if π is satisfactory

To solve efficiently, IRL methods often assume:

- ❖ Known dynamics (for comparing π and π^* efficiently)
- ❖ Linear reward function $r(s, a) = w^\top \phi(s)$

Problem: IRL is generally ill-posed – many reward functions under which the expert policy is optimal.

How can we address it?

Examples of Inverse Reinforcement Learning

The International Journal of Robotics Research OnlineFirst, published on June 23, 2010 as doi:10.1177/0278364910371999

ijrr

The International Journal of Robotics Research
000(00) 1–31

DOI: 10.1177/0278364910371999
ijr.sagepub.com
SAGE

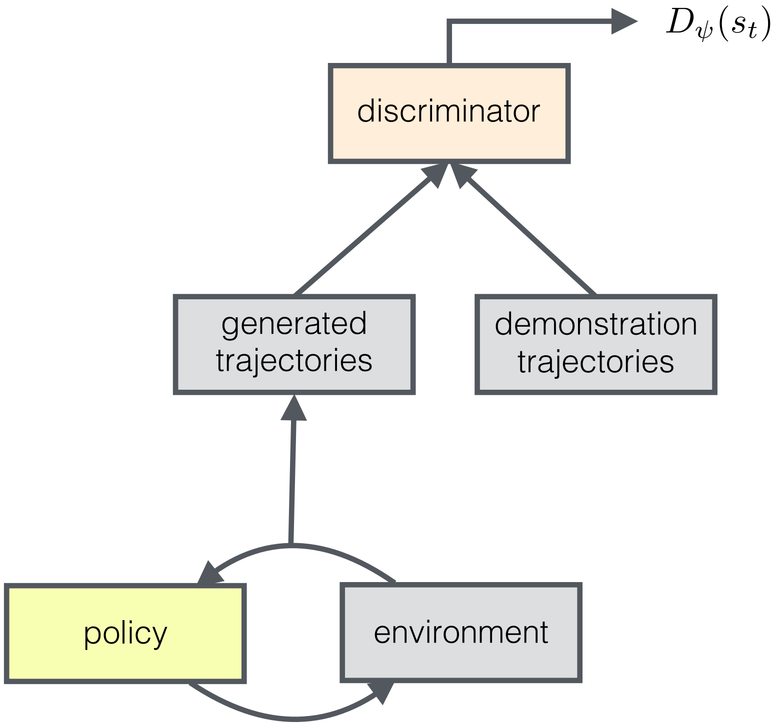
Autonomous Helicopter Aerobatics through Apprenticeship Learning

Pieter Abbeel[1], Adam Coates[2] and Andrew Y. Ng[2]

Abstract

Autonomous helicopter flight is widely regarded to be a highly challenging control problem. Despite this fact, human experts can reliably fly helicopters through a wide range of maneuvers, including aerobatic maneuvers at the edge of the helicopter's capabilities. We present apprenticeship learning algorithms, which leverage expert demonstrations to efficiently learn good controllers for tasks being demonstrated by an expert. These apprenticeship learning algorithms have enabled us to significantly extend the state of the art in autonomous helicopter aerobatics. Our experimental results include the first autonomous execution of a wide range of maneuvers, including but not limited to in-place flips, in-place rolls, loops and hurricanes, and even auto-rotation landings, chaos and tic-tocs, which only exceptional human pilots can perform. Our results also include complete airshows, which require autonomous transitions between many of these maneuvers. Our controllers perform as well as, and often even better than, our expert pilot.

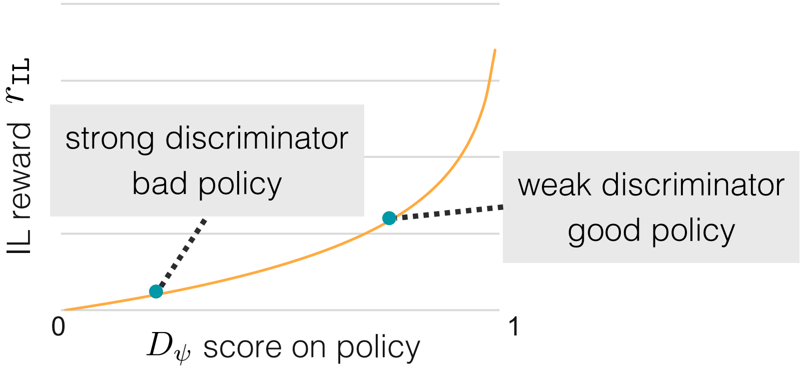
Adversarial Imitation Learning



discriminator objective

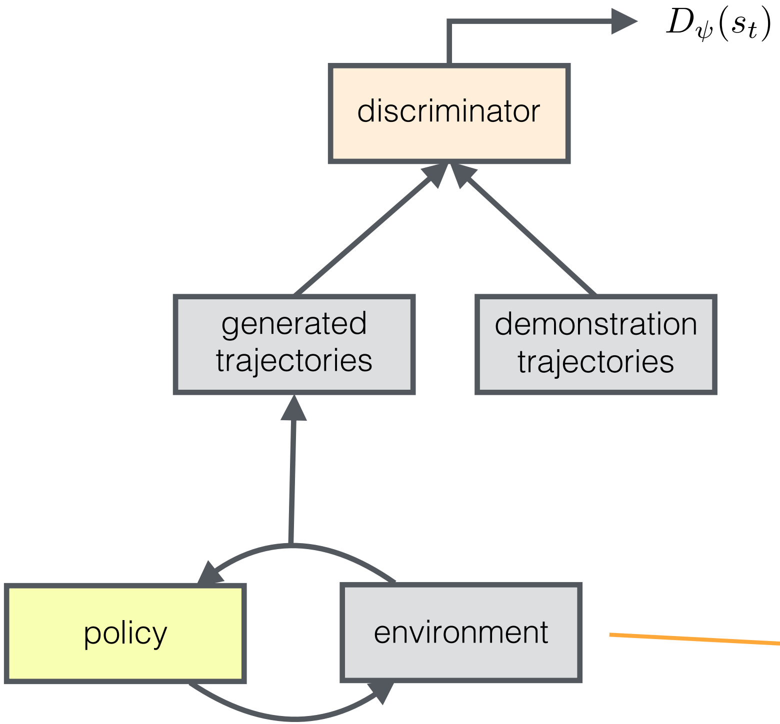
D_ψ predicts **0** if policy and **1** if demo

IL reward: $r_{\text{IL}}(s_t, a_t) = -\log(1 - D_\psi(s_t))$



[Goodfellow et al. 2014; Ho & Ermon, 2016]

Adversarial Imitation Learning



discriminator objective

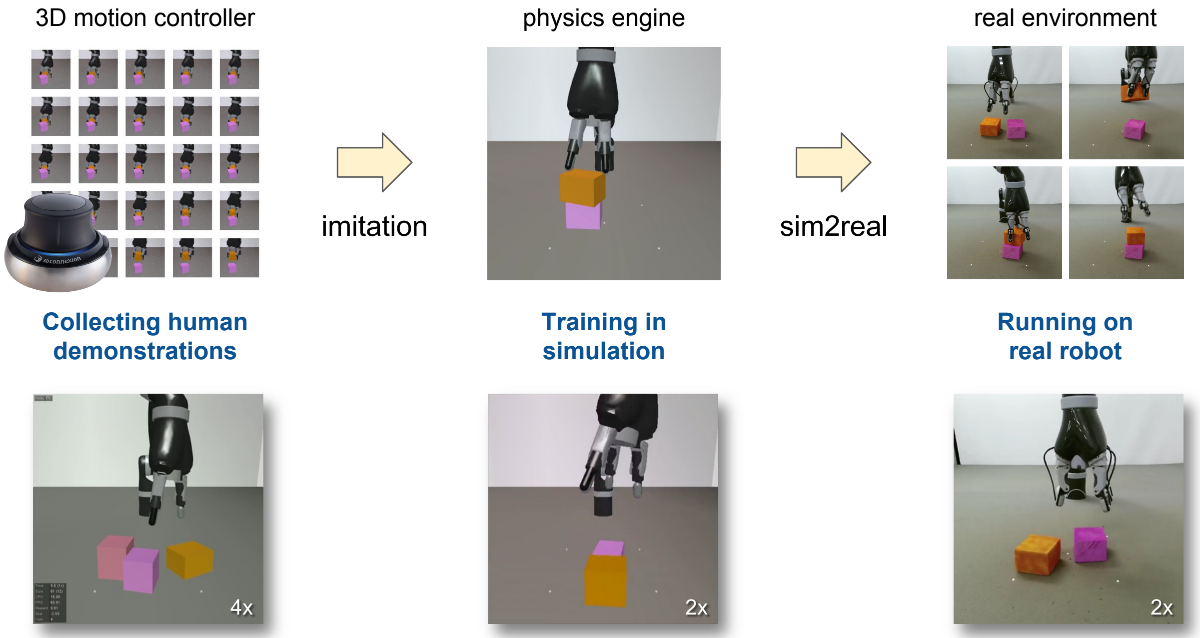
D_ψ predicts **0** if policy and **1** if demo

IL reward: $r_{\text{IL}}(s_t, a_t) = -\log(1 - D_\psi(s_t))$

- Represent **complex reward function** by neural networks
- More **iterative approaches** to update reward and policy (no need to run full RL before updating the reward function)
- We don't know the dynamics but have access to a **simulator** to compare with π and π^*.

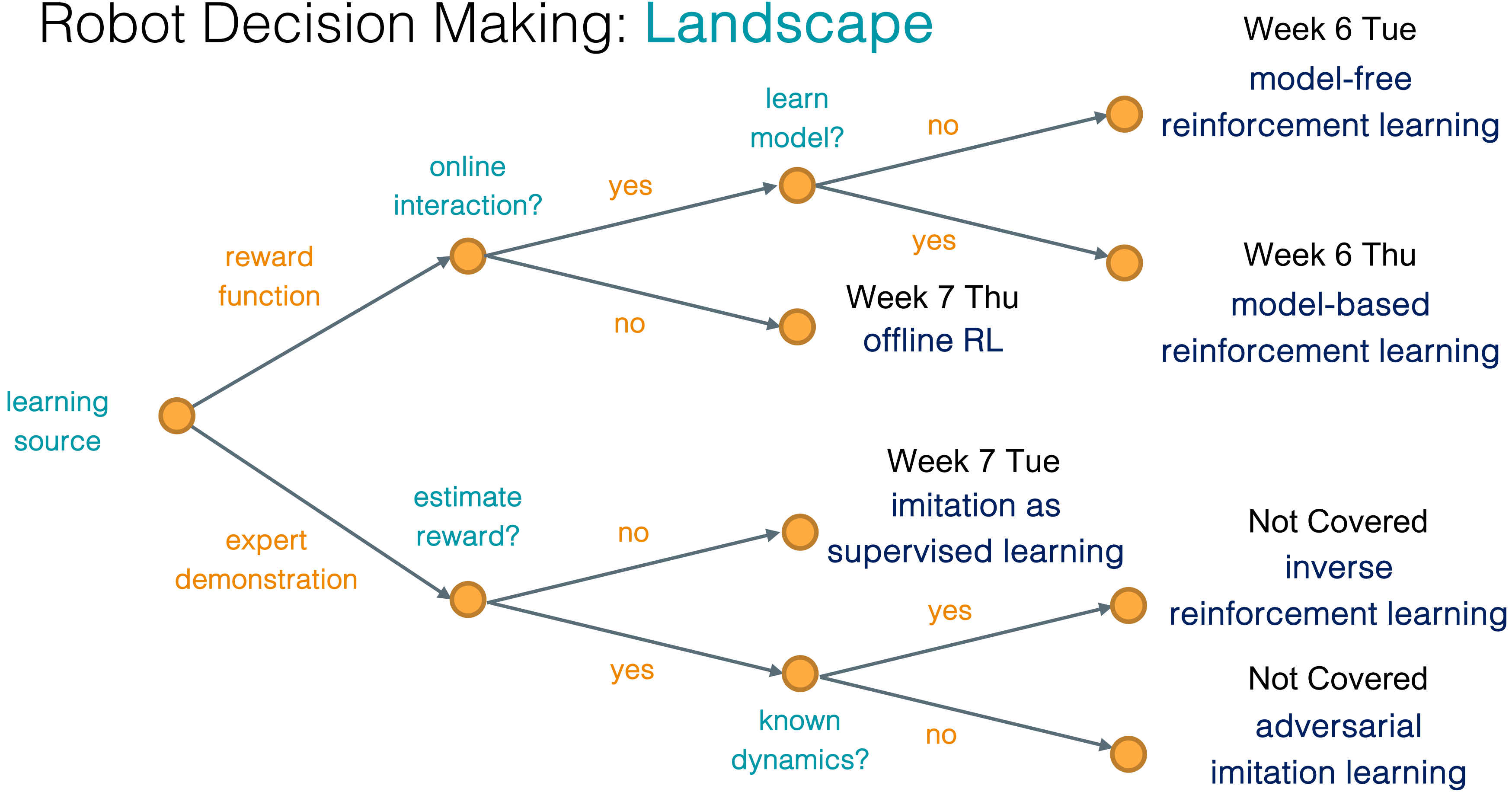
[Goodfellow et al. 2014; Ho & Ermon, 2016]

Examples of Adversarial Imitation Learning

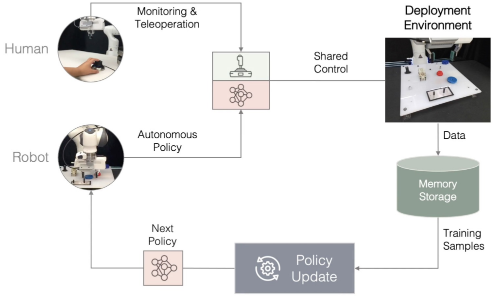


“Reinforcement and Imitation Learning for Diverse Visuomotor Skills.” Zhu et al. RSS 2018

Robot Decision Making: Landscape



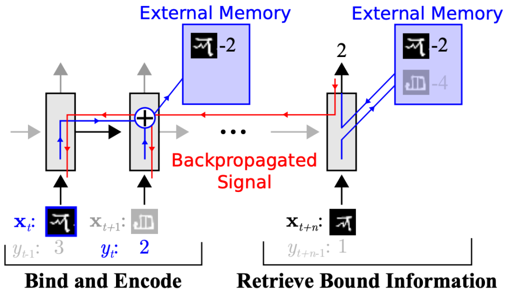
Robot Decision Making: Frontiers



[Liu et al. 2022]

Week 8 Tue

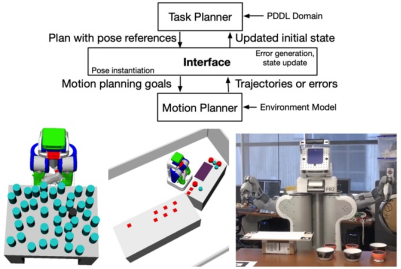
Human-in-the-Loop Learning



[Santoro et al. 2016]

Week 8 Thu

Learning to Learn



[Srivastava et al. 2014]

Week 9 Tue

Task and Motion Planning

Resources

Related courses at UTCS

- CS342: Neural Networks
- CS394R: Reinforcement Learning: Theory and Practice

Other Course Materials and Textbooks

- UCL Course on RL by David Silver
- Berkeley CS 294: Deep Reinforcement Learning
- Reinforcement Learning: An Introduction, Sutton and Barto
- Reinforcement Learning and Optimal Control, Bertsekas