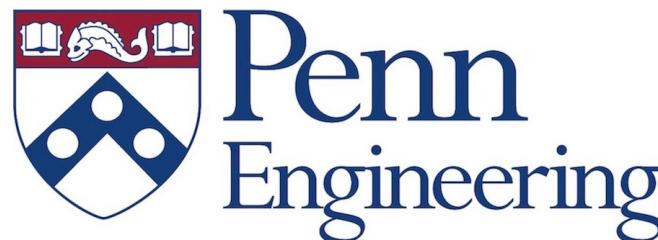


CIS 7000-04 / ESE 6800 Spring 2025: Intro To Imitation and Reinforcement Learning

Tue, Jan 21, 2025

Instructor: Dinesh Jayaraman

Assistant Professor, CIS



Module Goals (about 3 lectures)

We assume that:

- You understand machine learning at the level of an introductory class
e.g. CIS 5190
- You don't know much about imitation and reinforcement learning
- You know a little bit about robots

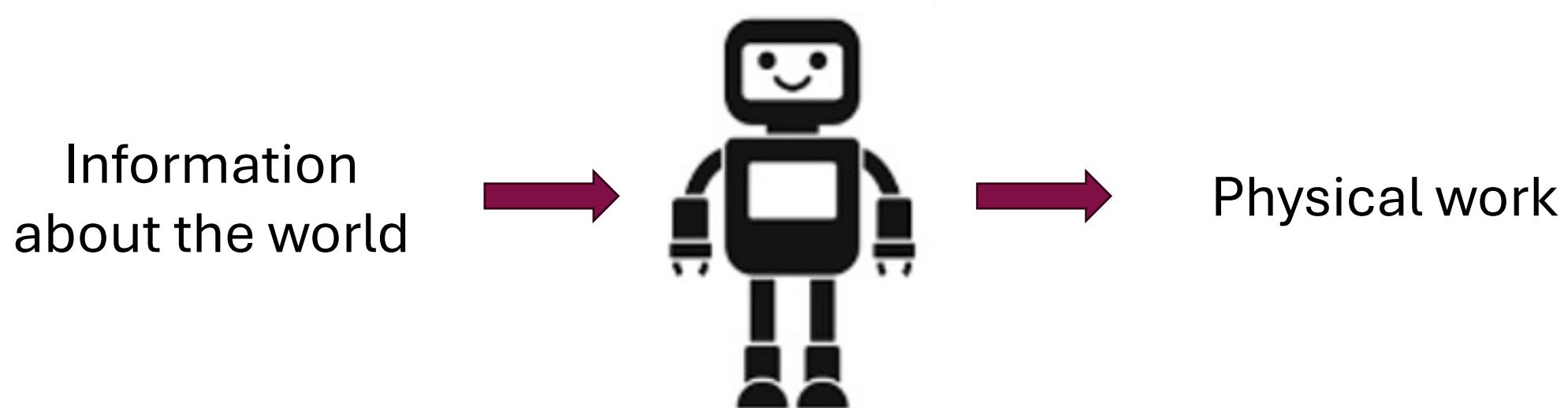
At the end, you will know:

- How to cast the problem of robotic control as a sequential decision making problem
- What the key ideas of imitation learning and reinforcement learning are, and sketches of some basic methods for each
- Broad landscape of imitation and RL, and names and self-study references for common approaches

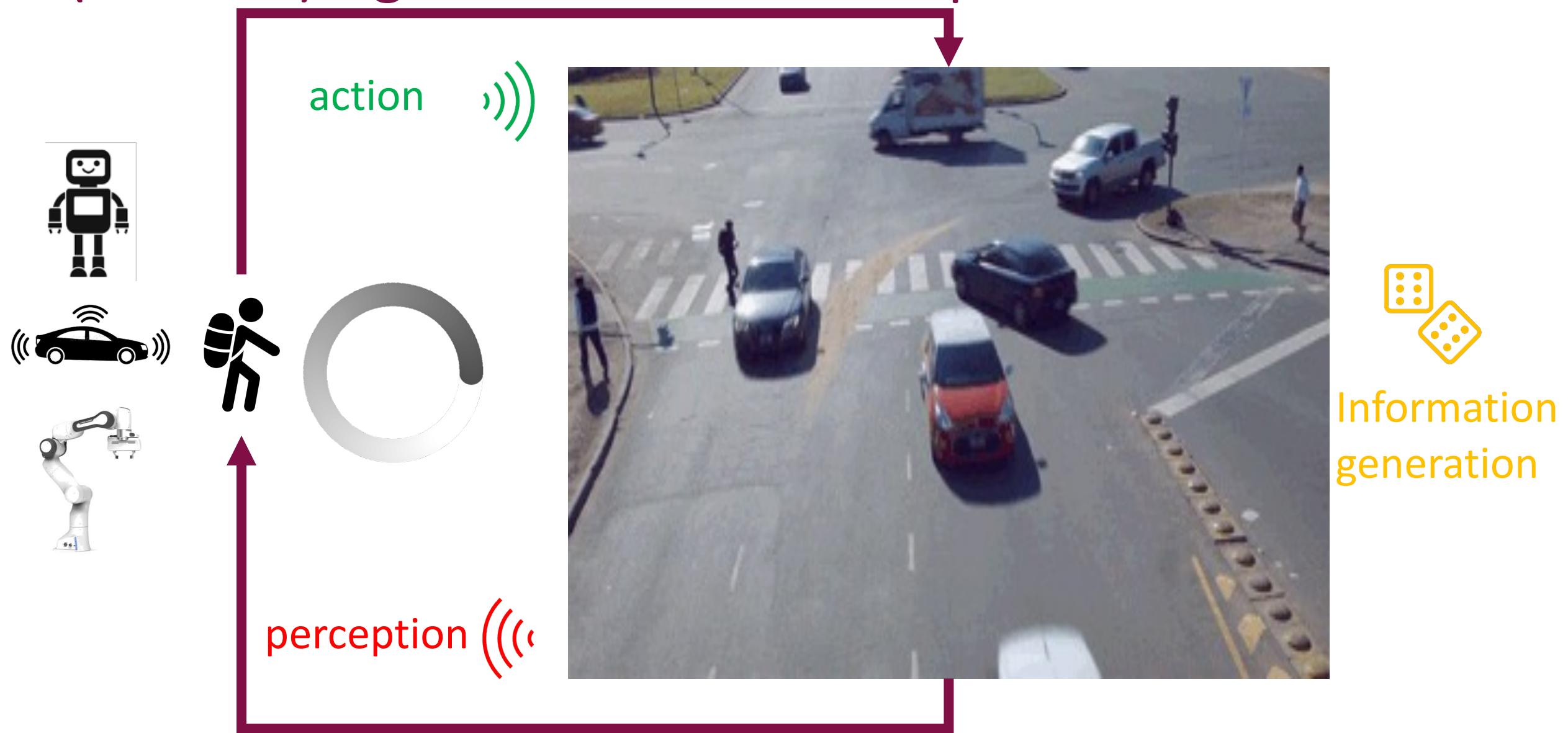
Plan

- What Does Robot Control Look Like? Perception-Action Loops
- How Markov Decision Processes Model Robot Control
- What If Dynamics and Reward are Unknown
 - Imitation Learning
 - Reinforcement Learning
 - Policy Gradients
 - Q Learning, DDPG ...
 - Model-based RL (stretch goal)

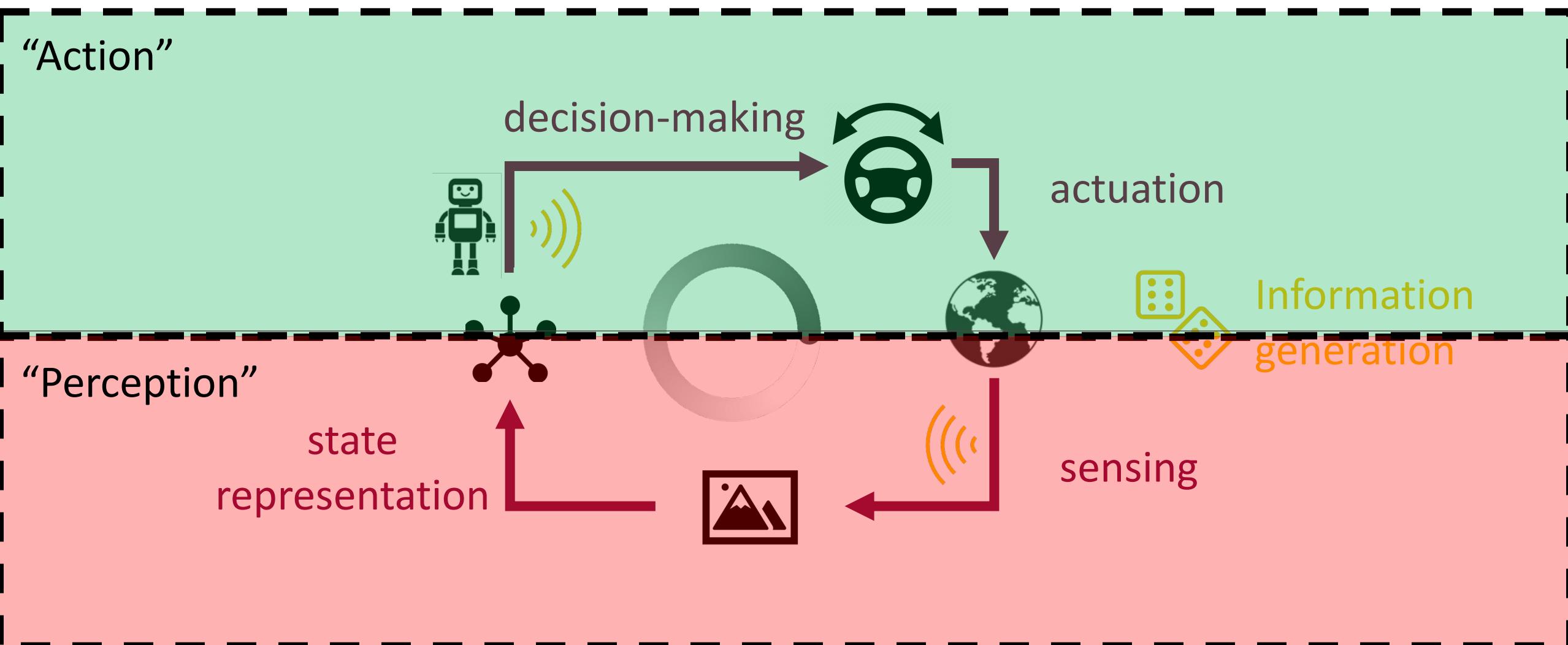
Robots: Information → Physical Work



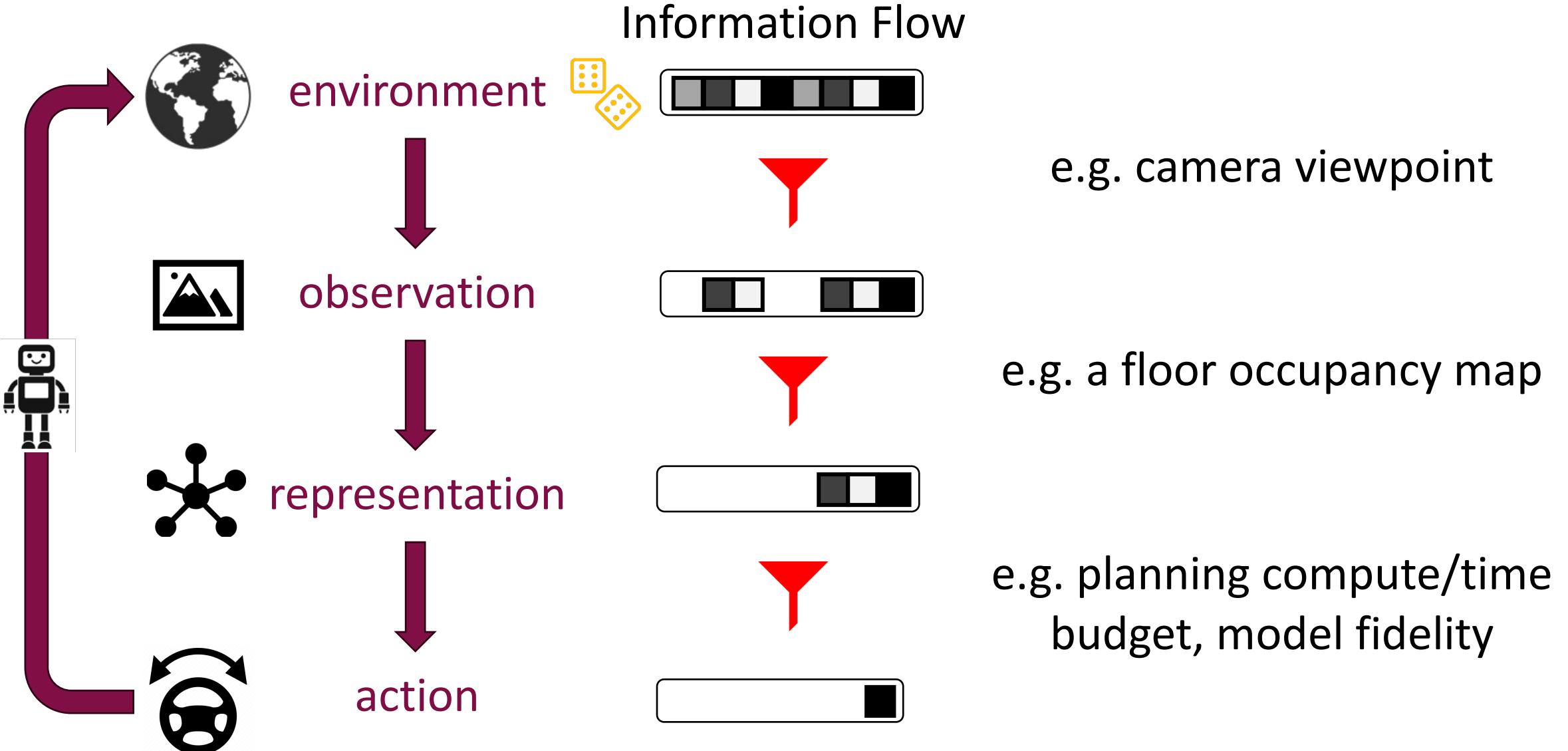
(Robotic) Agent ↔ World: Perception & Action



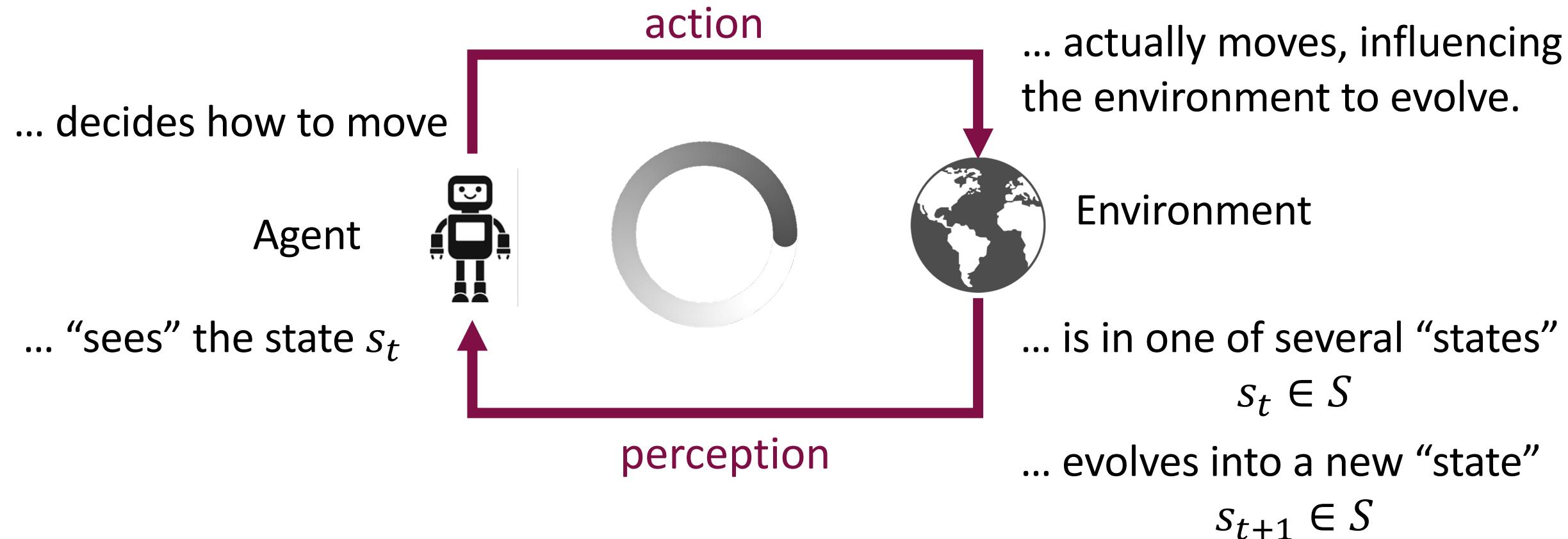
Zooming Into Perception-Action Loops



Selective Processing of Information



The Perception-Action Loop (“Fully Observed”)



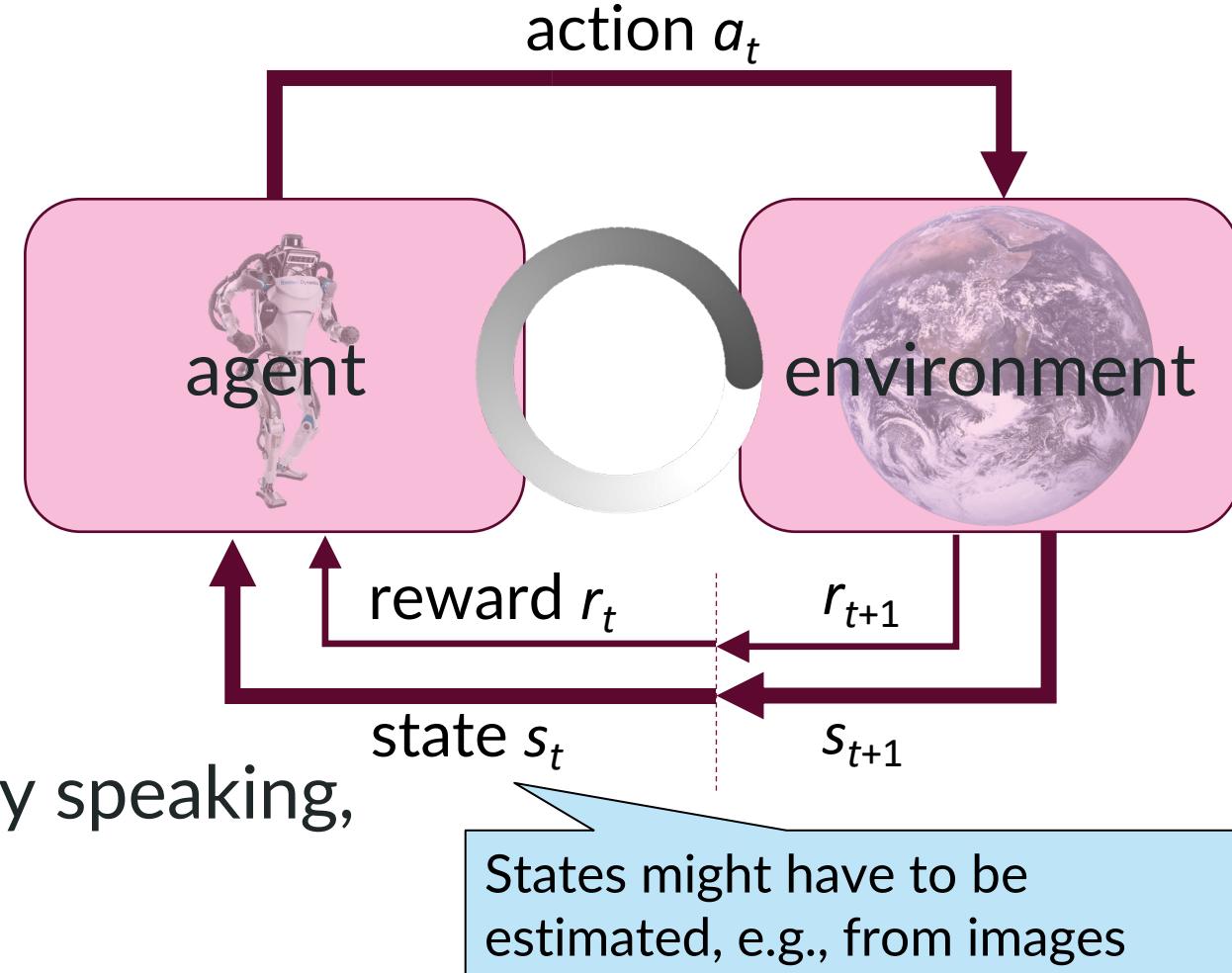
What Robots Can Learn From Data

- **Policies:** mappings from perceived state s_t to action commands a_t
- **Dynamics models:** models of how agent actions influence the evolution of the environment state
- **Reward functions:** a score indicating how well the robot is performing a task.
- **State Representations:** an encoding of raw sensory inputs
- **What to Sense**
- **“Common-Sense Knowledge”**

Casting Robot Control As A Sequential Decision Making Problem

Markov Decision Process Formulation of Control

- Agent receives observations (state $s_t \in S$) and feedback (reward r_t) from the world
- Agent takes action $a_t \in A$
- Agent receives updated state s_{t+1} and reward r_{t+1}
- Agent's goal is to maximize, loosely speaking, "expected rewards in the future".

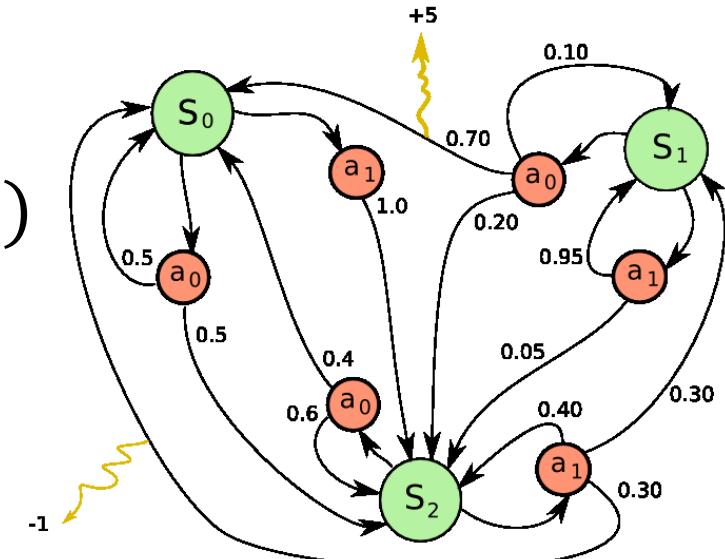


MDP Formulation: Notations

An MDP (S, A, P, R, γ) is defined by:

- Set of states $s \in S$
- Set of actions $a \in A$
- Transition function or “dynamics model” $P(s' | s, a)$
 - Probability $P(s' | s, a)$ that a from s leads to s'
- Reward function $r_t = R(s, a, s')$
- Discount factor $\gamma < 1$, expressing how much we care about the future (vs. immediate rewards)
- “utility” = *discounted* future reward sum $\sum_t \gamma^t r_{t+1}$
- Goal: maximize *expected* utility

Example



Robots Make Sequential Decisions Too!

Must make a sequence of decisions to maximize some success measure/"reward", which is a cumulative effect of the full sequence.



Actions a_t : muscle contractions

Observations s_t : sight, smell

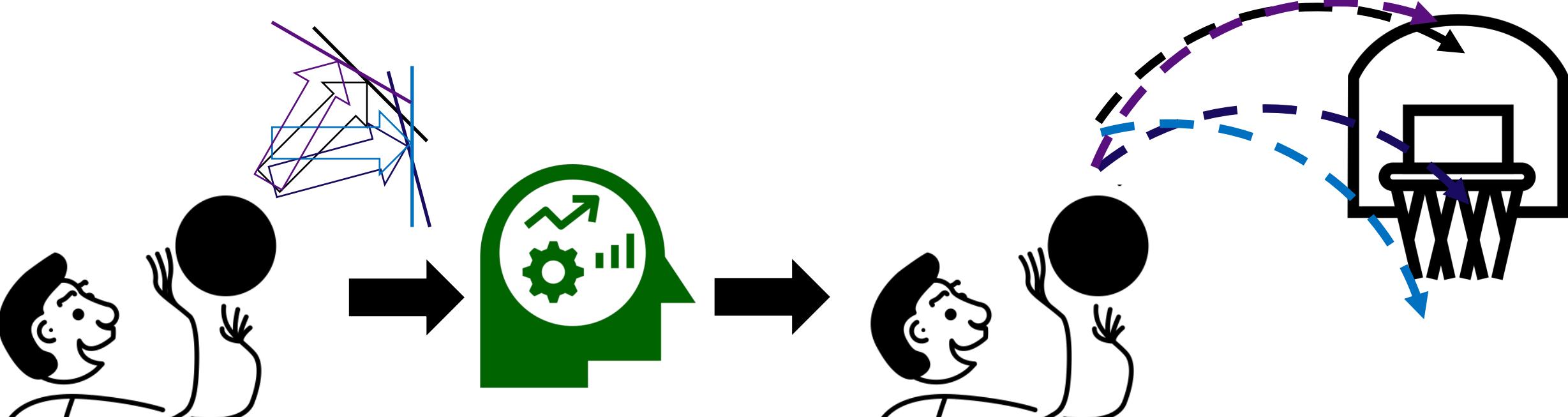
Reward r_t : food

motor current or torque

camera images

average speed

Traditional Model-Based Planning (an example)



1. Sample actions and forecast their effects using $P(s'|s, a)$.
2. Select action with best forecasted outcome $\sum_t \gamma^t r_{t+1}$.

If you can predict how the environment will evolve in response to an action (and score those predictions), you can select good actions.

Traditional Model-Based Controllers

- Broadly, traditional approaches rely on having near-accurate predictive “models” of the environment. E.g. LQR (linear quadratic regulators), MPC (model predictive control), H-infinity control, Policy Iteration (dynamic programming) ...
- The quality of the synthesized controller depends directly on the quality of the environment model.

Towards Imitation and Reinforcement Learning

- But in practice, it is too strong of an assumption that $P(\cdot)$ and $R(\cdot)$ are known in advance.
- If we assume no knowledge of $P(\cdot)$ and $R(\cdot)$, we are in the position of an agent dropped into a completely unknown environment with a completely unknown task. How could this even work?

Sidenote: In real problems, we have *some* knowledge of $P(\cdot)$ and $R(\cdot)$, but this assumption helps study algorithms for the most general setting.

Towards Imitation and Reinforcement Learning

- How might learning even work with unknown dynamics and rewards?
 - **Reinforcement Learning:**
 - No idea about either $P(\cdot)$ or $R(\cdot)$ at the start of training, but we do have the ability to try things out in the world, each time experiencing the effects of $P(\cdot)$ and $R(\cdot)$.
 - Key Unique Issues:
 - Exploration: How to acquire useful experience to select a good policy?
 - Credit Assignment: How to figure out which parts of experience were “good” (likely to lead to good outcomes) and which were “bad”?
 - More on these later
 - **Imitation Learning:**
 - No idea about either $P(\cdot)$ or $R(\cdot)$, but largely circumvents above RL issues by having a teacher show us the way!
 - Rather than figure out both how to gather experiences and how to optimize policies based on them, we *only* optimize policies.

Side Note 1: Partially Observed MDPs (POMDPs)

- Most robotics problems are *partially observed* i.e., you cannot observe the full Markov state (or you can only observe it noisily).
 - E.g. an autonomous vehicle cannot see a pedestrian beyond a neighbor car.
- Handling this correctly requires a change to the basic MDP formulation; we will not cover this in this tutorial, but instead return to it as our readings during the course require.

Side Note 2: Other Assumptions in the MDP framework

- Time discretization -> continuous-time MDPs etc.
- Instantaneous s_t, a_t at time t . -> real-time MDP, delay-aware MDPs etc.
- Simple scalar rewards -> constrained MDPs etc.

...

Imitation Learning Through Behavior Cloning

Solving sequential decision making problems with supervised learning!

Which year is this from?

"This review investigates two recent developments in artificial intelligence and neural computation: learning from imitation and the development of humanoid robots. It will be postulated that the study of imitation learning offers a promising route to gain new insights into mechanisms of perceptual motor control that could ultimately lead to the creation of autonomous humanoid robots."

Schaal, S (1999). Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences* 3:233-242.

Is Imitation Learning the Route to Humanoid Robots?

Stefan Schaal

sschaal@usc.edu

<http://www-slab.usc.edu/sschaal>

Computer Science and Neuroscience, HNB-103, University of Southern California, Los Angeles, CA 90089-2520
Kawato Dynamic Brain Project (ERATO/JST), 2-2 Hikaridai, Seika-cho, Soraku-gun, 619-02 Kyoto, Japan

Summary

This review investigates two recent developments in artificial intelligence and neural computation: learning from imitation and the development of humanoid robots. It will be postulated that the study of imitation learning offers a promising route to gain new insights into mechanisms of perceptual motor control that could ultimately lead to the creation of autonomous humanoid robots. Imitation learning focuses on three important

Robots then

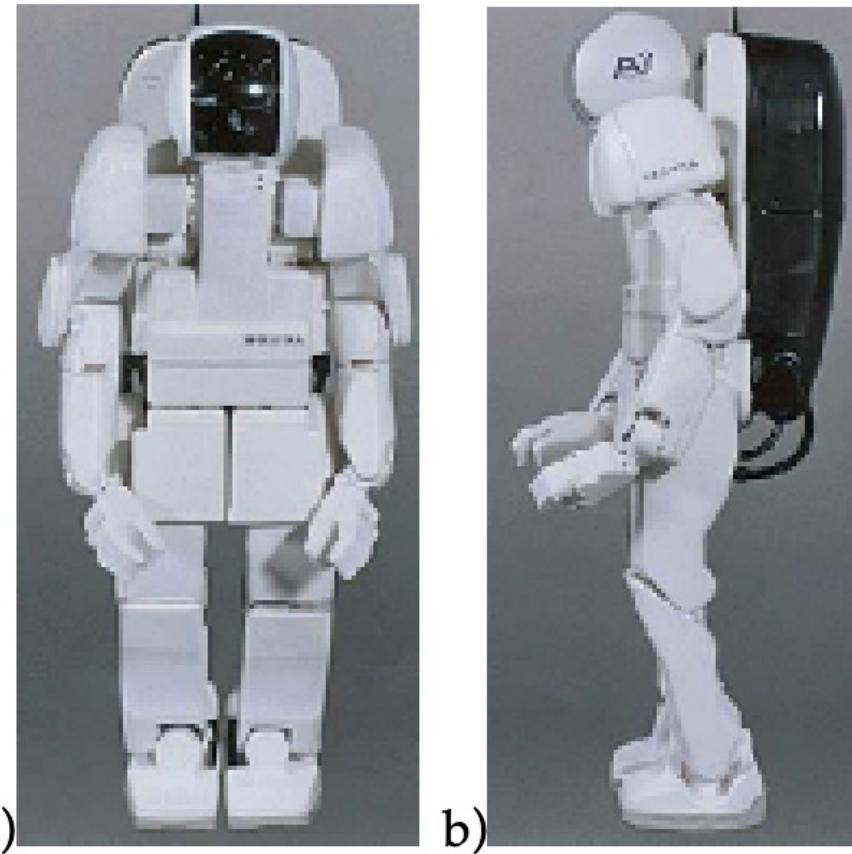


Figure 1: Honda Humanoid Robot in a) frontal, and b) side view.

Fast forward to 2024



FULLY LEARNED, FAST,
TION

Video from Figure.ai

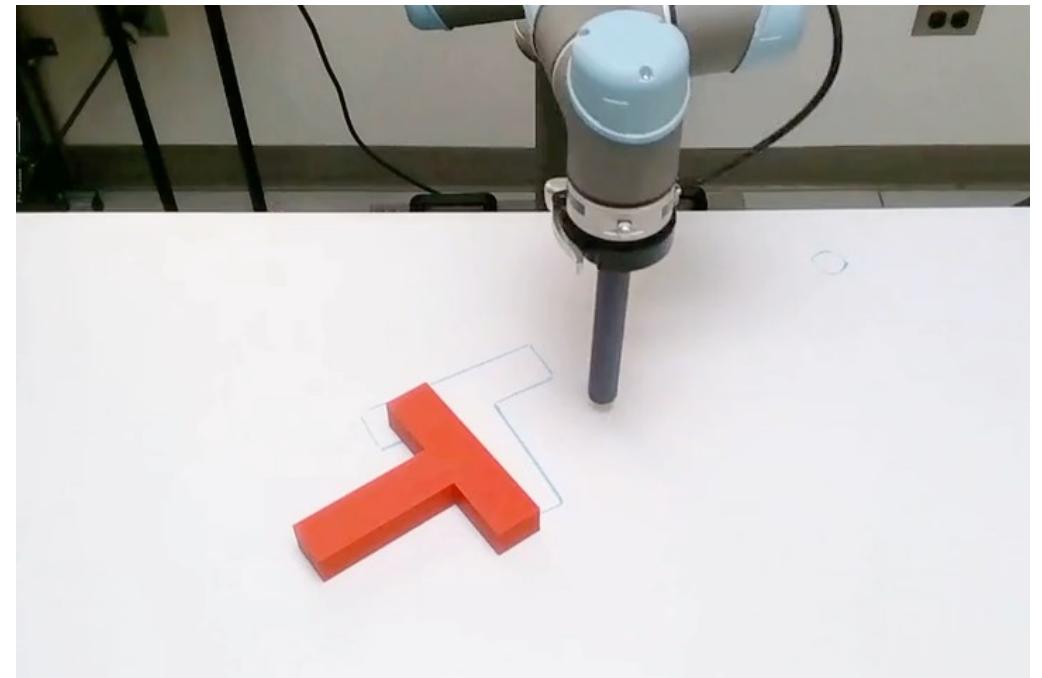


Video from Tesla

Exciting progress on imitation learning



Video from ALOHA (Zhao et al., 2023)



Video from Diffusion Policy (Chi et al., 2023)

When you have lots of data



Video Credit: DeepMind



Video Credit: Physical Intelligence

“Policies” for Sequential Decision Making

For any input state of the system, the ML policy model maps it to a decision.

- This motivates the following input-output structure of the model:
 - Input: state observation, like sight and smell for the dog.
 - Output: actions, like muscle contractions.

This mapping from input states to a probability distribution over output actions (or sometimes just a single deterministic action) is called a decision-making “policy”, often denoted π .

Supervised learning of Action Policies?

- Given the current “state” x , make a decision $\hat{y} = \max_y \pi_\theta(y|x)$.
 - Supervision => labels for “good” decisions that maximize rewards.
 - So, we’d like to have some dataset of (state x , good decision y^*) pairs. Then we could try running supervised learning just as always.
- For the sequential decision making problem, we will use the notation:
 - state input s instead of x ,
 - action output a instead of y .
 - We will often subscript these items with time indices as s_t, a_t etc.

Behavior Cloning (BC)



expert

observed states
 s_1, s_2, \dots, s_H
 a_1, a_2, \dots, a_H
actions

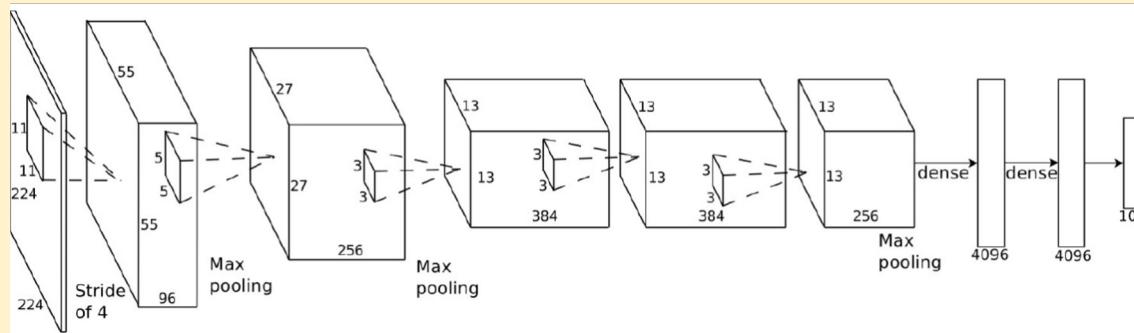
training data

supervised learning

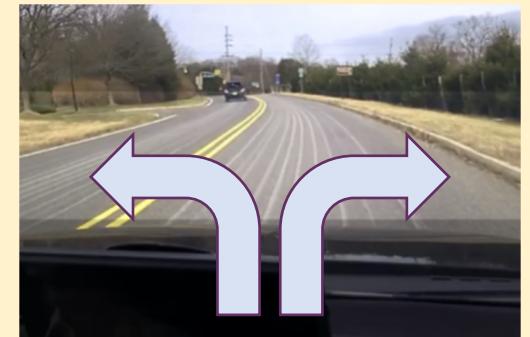
policy
 $\pi_\theta(a_t | s_t)$



observed state s_t



convolutional network



action a_t

An “end-to-end” policy

Behavior Cloning Objective Function

Supervised maximum-likelihood objective to map from states to expert actions.

$$Loss = -\frac{1}{N} \sum_{i=1}^N \left(\sum_{t=1}^T \log \pi_\theta(a_{i,t} | s_{i,t}) \right)$$

Demonstration data Expert actions

trajectories time

Could minimize by following the gradient:

$$\frac{1}{N} \sum_{i=1}^N \left(\sum_{t=1}^T \nabla_\theta \log \pi_\theta(a_{i,t} | s_{i,t}) \right)$$

Likelihood gradient:
“Change the policy to
make these actions
more likely”.

Behavior Cloning (BC) Objective Function

Supervised maximum-likelihood objective to map from states to expert actions.

$$Loss = -\frac{1}{N} \sum_{i=1}^N \left(\sum_{t=1}^T \log \pi_\theta(a_{i,t} | s_{i,t}) \right)$$

Demonstration data Expert actions
trajectories time

- **Violates i.i.d. assumptions of supervised learning:** learner's prediction affects future input observations / states during execution of the learned policy. Will return to this later.
- **No connection to the task reward function?** Best, we can say is, if $R(\cdot)$ is bounded e.g. $[0, -1]$, then a policy with error rate ϵ on the expert data incurs a reward penalty $< O(T^2\epsilon)$

Does this really work?

ALVINN: AN AUTONOMOUS LAND VEHICLE IN A NEURAL NETWORK

Dean A. Pomerleau
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

ABSTRACT

ALVINN (Autonomous Land Vehicle In a Neural Network) is a 3-layer back-propagation network designed for the task of road following. Currently ALVINN takes images from a camera and a laser range finder as input and produces as output the direction the vehicle should travel in order to follow the road. Training has been conducted using simulated road images. Successful tests on the Carnegie Mellon autonomous navigation test vehicle indicate that the network can effectively follow real roads under certain field conditions. The representation developed to perform the task differs dramatically when the network is trained under various conditions, suggesting the possibility of a novel adaptive autonomous navigation system capable of tailoring its processing to the conditions at hand.

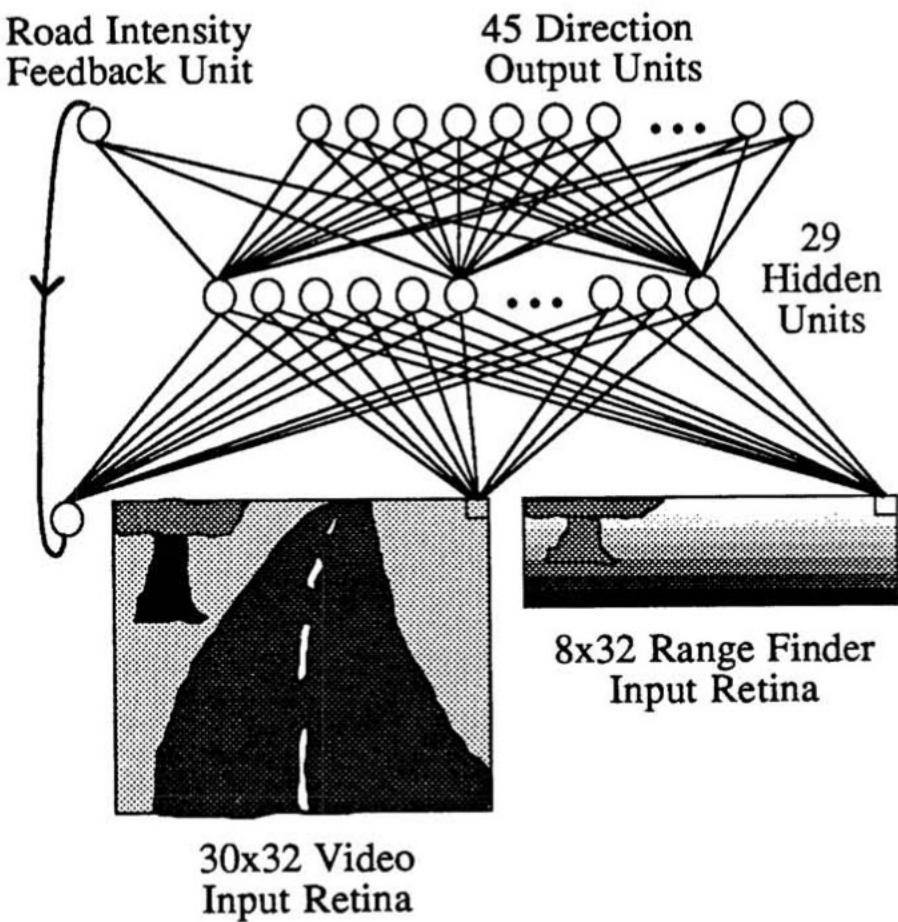


Figure 1: ALVINN Architecture

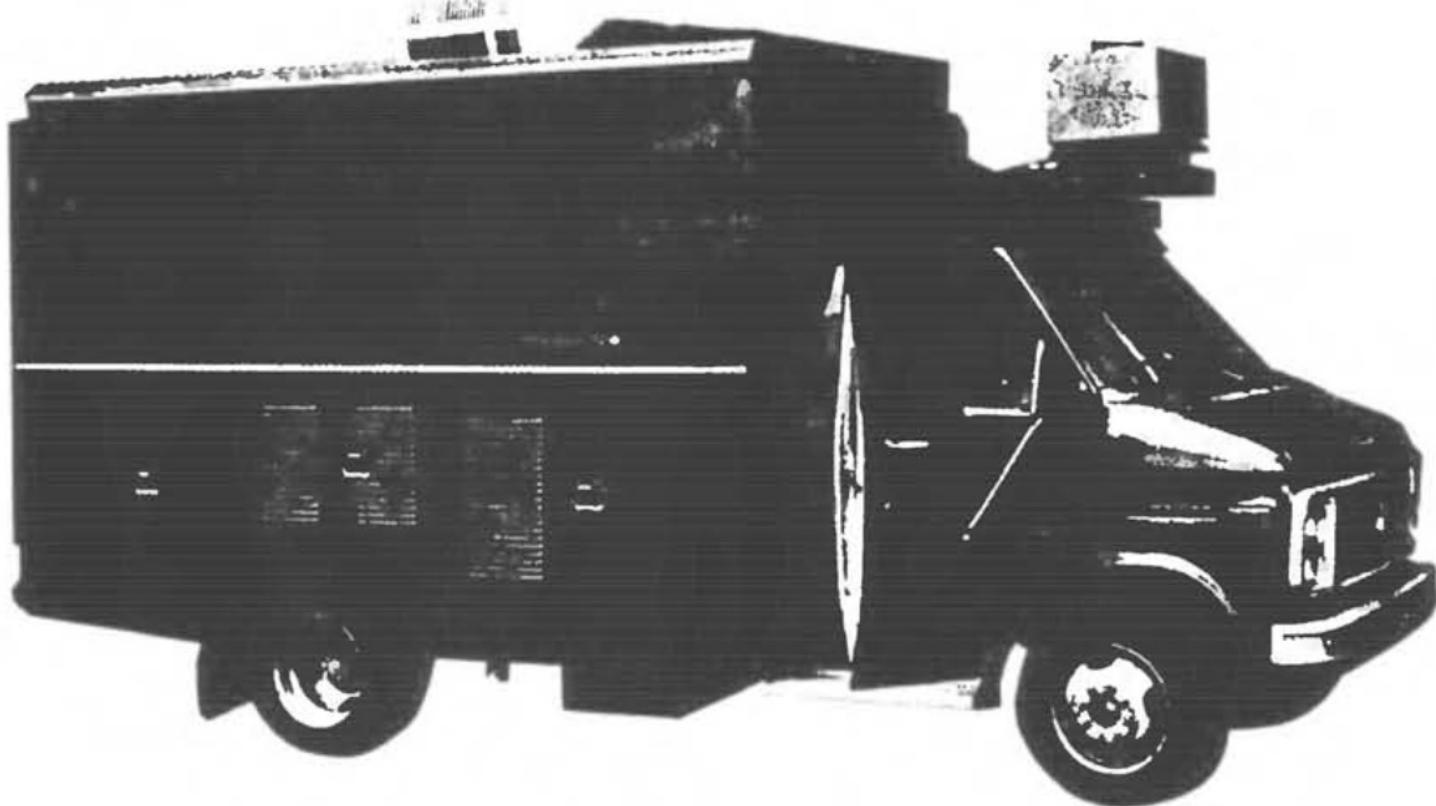


Figure 3: NAVLAB, the CMU autonomous navigation test vehicle.

"the network can accurately drive the NAVLAB at a speed of 1/2 meter per second along a 400 meter path through a wooded area of the CMU campus under sunny fall conditions"

“No Hands Across America” – 1995!

[No Hands Across America](#)
[Home Page](#)



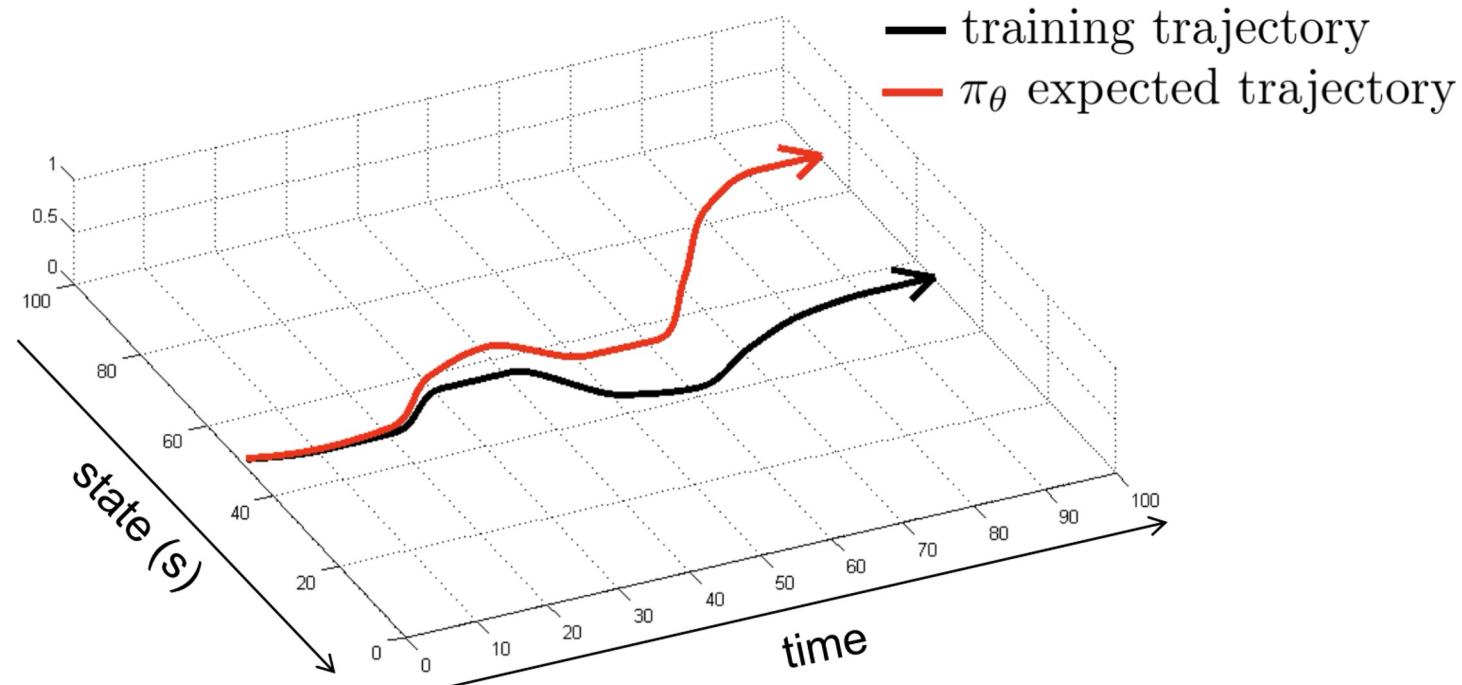
“On July 23, 1995 Research Scientist Dean Pomerleau and Ph.D. student Todd Jochem, both from the [Robotics Institute](#) of [Carnegie Mellon University](#) in Pittsburgh, PA, will begin a trans-continental journey in their 1990 Pontiac Trans Sport. If all goes well, the two will end up in San Diego, CA on July 30. What makes this trip special is that the vehicle will drive itself most of the way.”

TRIP COMPLETE !!!

2797/2849 miles (98.2%)

Distribution Shift in BC

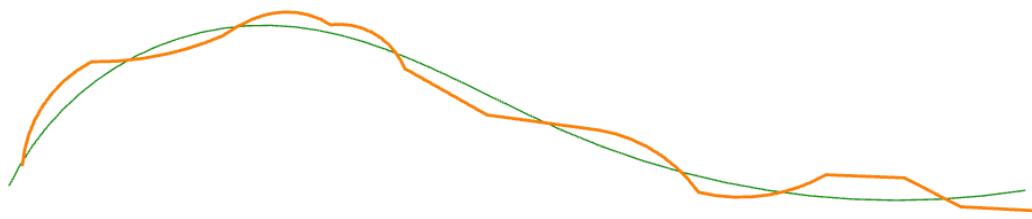
The policy is trained on *demonstration data* that is different from the data it encounters in the world.



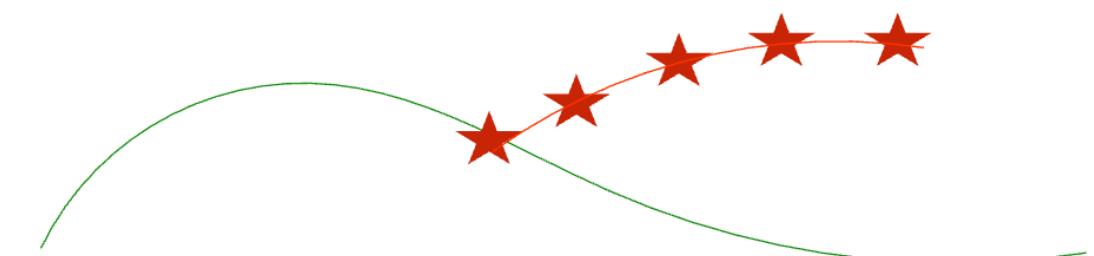
The cloned policy is imperfect; this leads to “compounding” errors, and the agent soon encounters unfamiliar states, leading to failure.

Note how these errors arise from ignoring the the *sequential, interconnected* nature of the task. Past decisions influence future states!

Compounding Error In Behavior Cloning



Independent-in-time errors



Compounding error

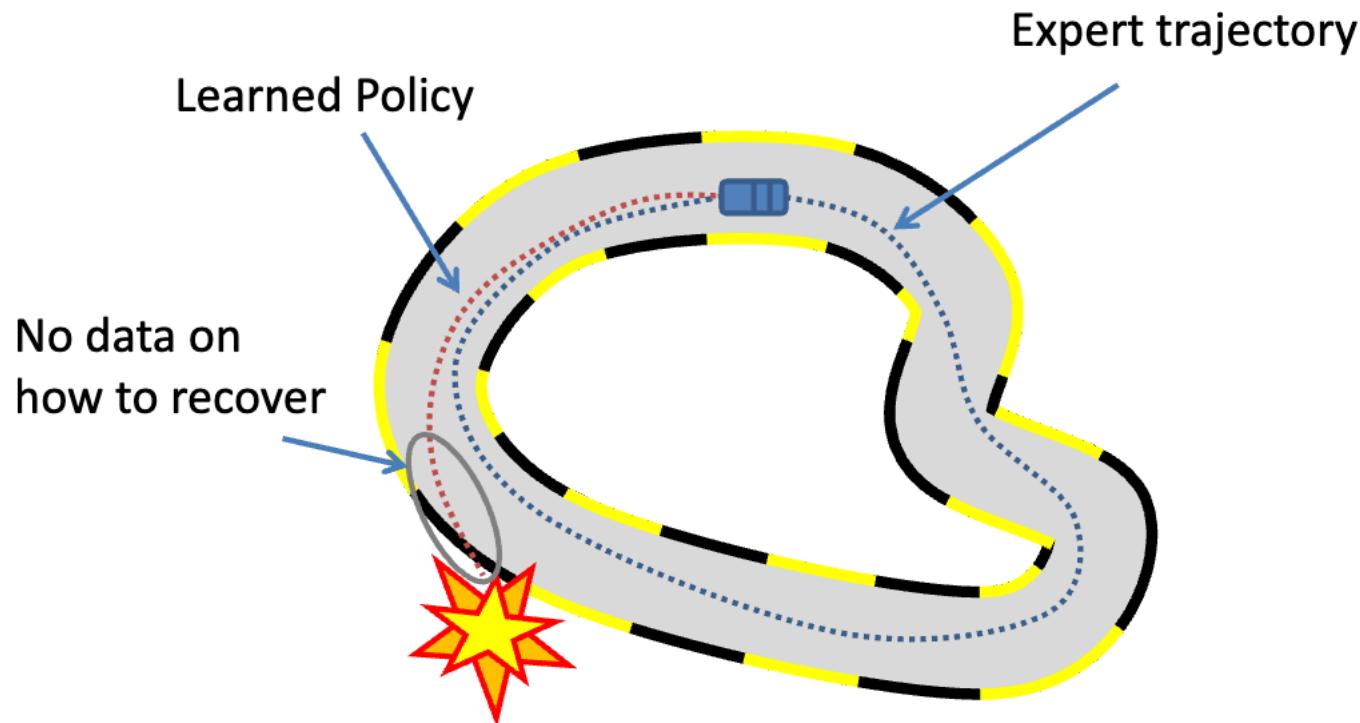
Images: Katerina Fragkiadaki

[Ross and Bagnell 2010, Efficient Reductions for Imitation Learning](#)

“When driving for itself, the network may occasionally stray from the center of road and so must be prepared to recover by steering the vehicle back to the center of the road.”

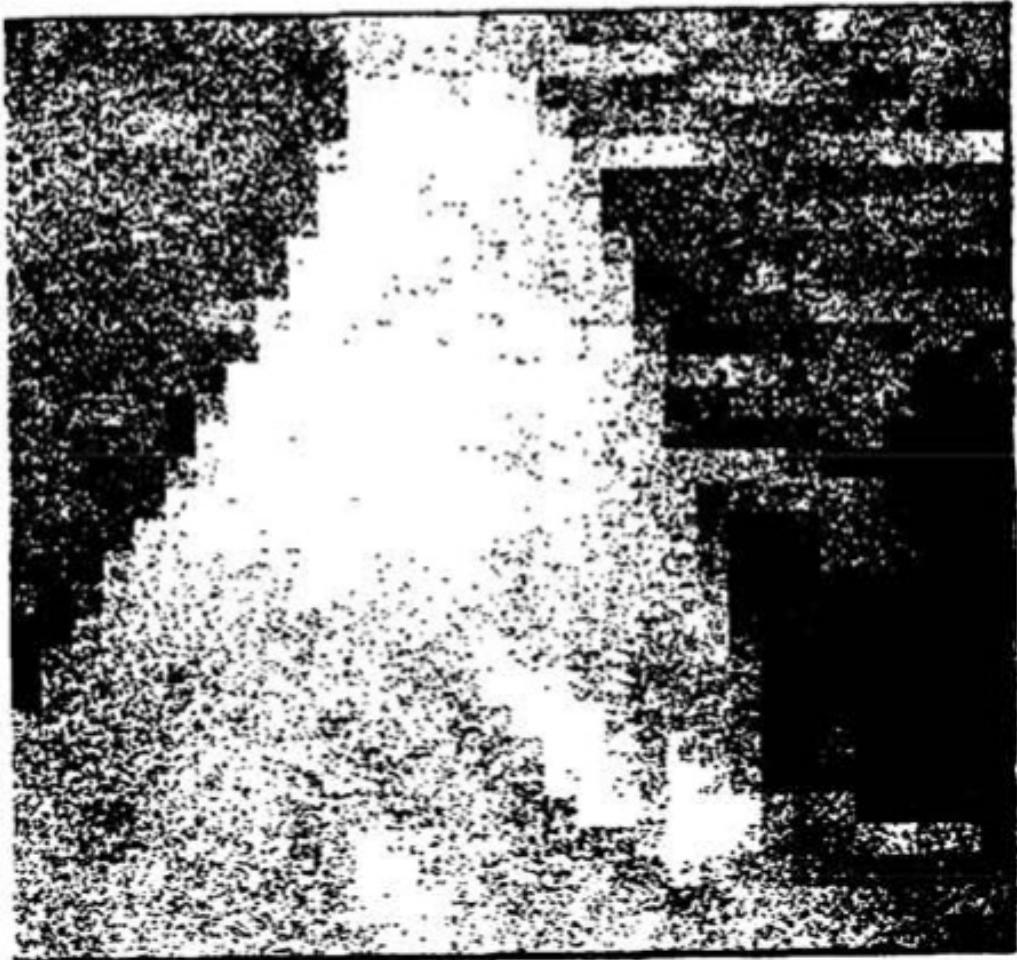
- Pomerleau '89

What's Missing: “Recovery” Data

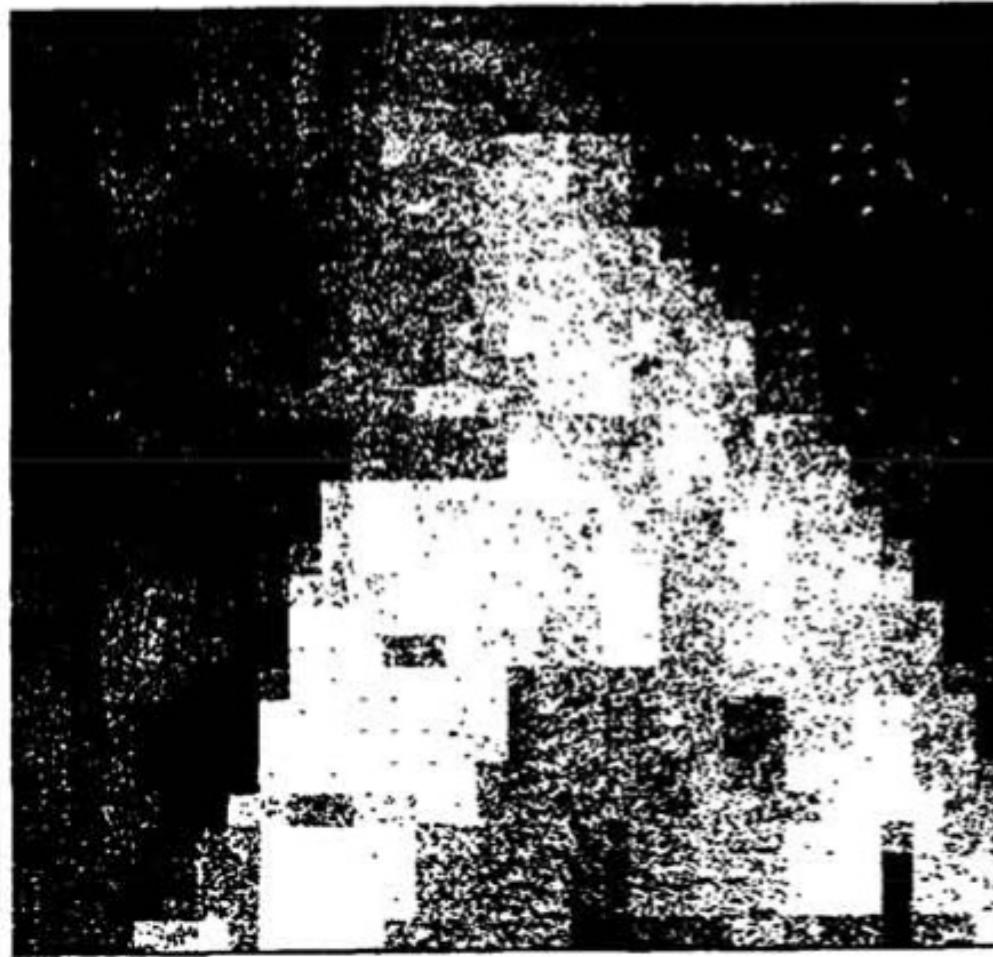


Images: Katerina Fragkiadaki

ALVINN involved Simulated(!) Recovery Data



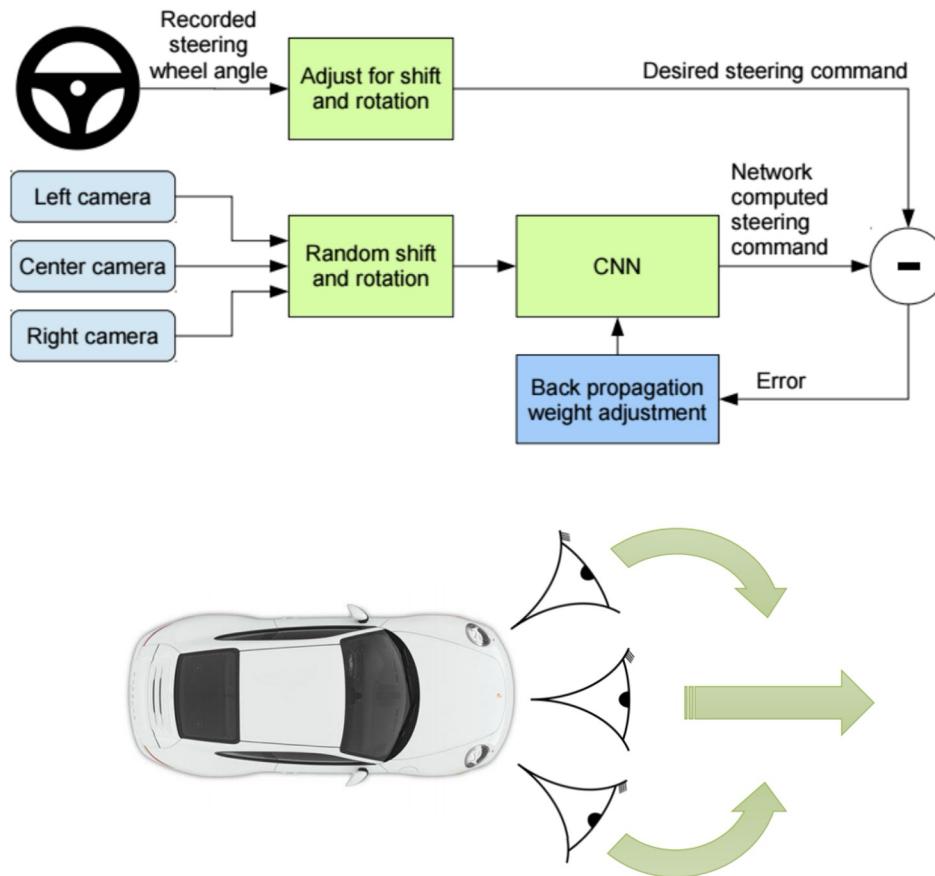
Real Road Image



Simulated Road Image

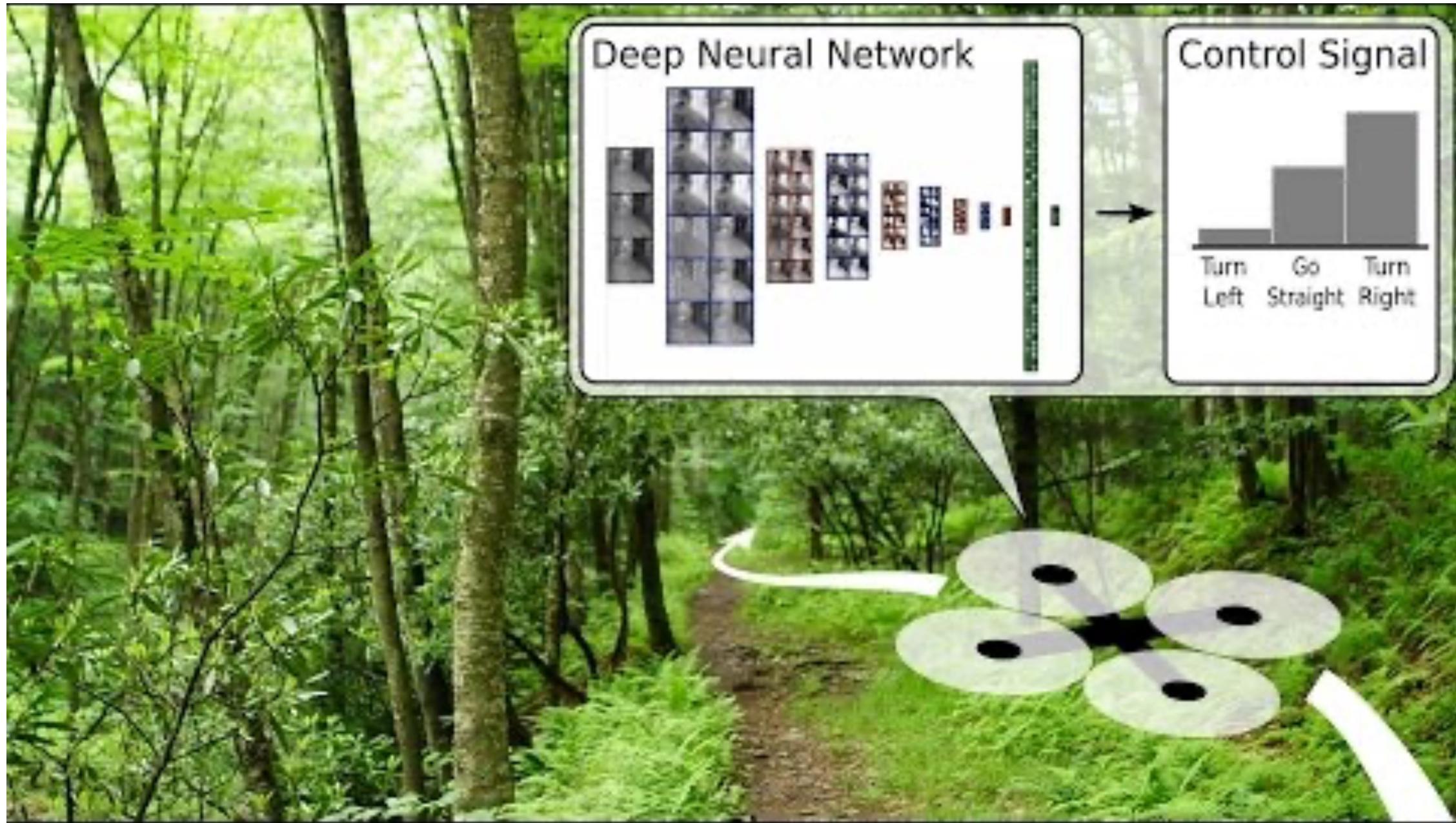
proceedings.neurips.cc/paper/1988/file/812b4ba287f5ee0bc9d43bbf5bbe87fb-Paper.pdf

Clever Recovery Data Collection For Navigation



[Dave-2 A Neural Network Drives A Car - YouTube](#)

Hack to handle limited distributional shift for this specific MDP ... exploits the geometry of the state and action space.



Interactive Online BC: DAGGER [Ross et al, 2011](#)

A general trick for handling distributional shift: requery expert on new states encountered by the initial cloned policy upon execution, then retrain.

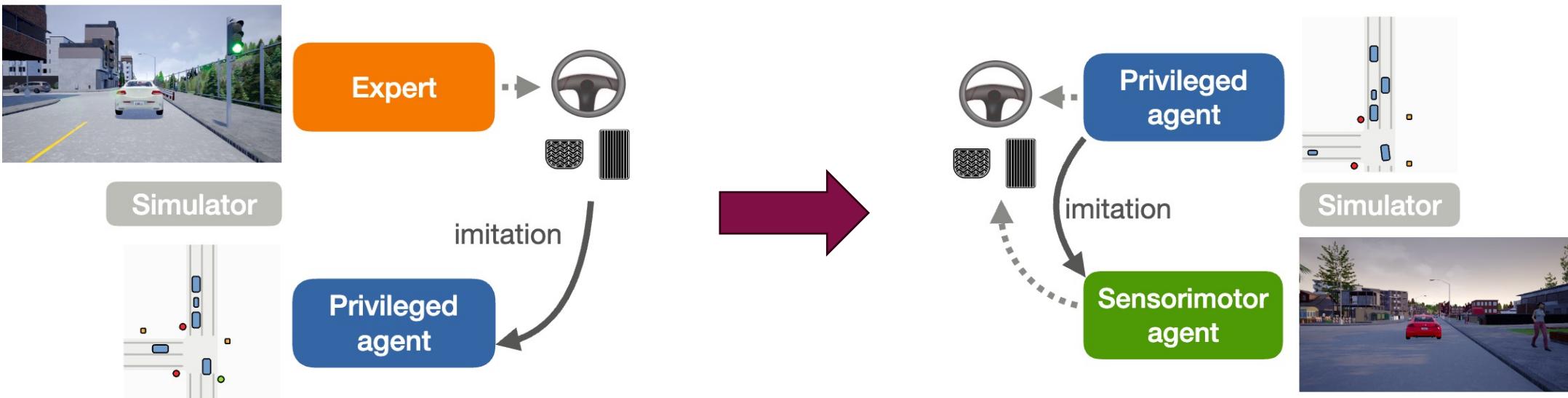
- 
1. Train $\pi_\theta(a_t|s_t)$ from expert data $\mathcal{D} = \{s_1, a_1, \dots, s_N, a_N\}$
 2. Execute / “Rollout” π_θ to get dataset $\mathcal{D}_\pi = \{s_1^{new}, \dots, s_M^{new}\}$
 3. Ask expert to label each state in \mathcal{D}_π with actions a_t^{new}
 4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

Assumes it is okay to keep asking the expert all through the training process.
“Queryable experts”. Might not always be practical.

See also (for an offline, non-interactive variant): Lee et al 2017, [DART](#): Disturbances for Augmenting Robot Trajectories

Other Policies as “Experts”

- Rather than humans, sometimes the “expert” may be another policy. E.g. a policy trained from privileged observations, or a hand-scripted policy, or even a “traditional” model-based policy operating on the system state.
- E.g. “learning by cheating” [[Chen et al, 2020](#)] in simulation:
 - first train an expert policy to operate from simulator state rather than images (e.g. with reinforcement learning, or even a hand-coded program)
 - Train a vision-based policy to imitate the expert.



BC is akin to how language models are (first) trained

- Language models (LMs) e.g. the GPT family perform “next-token prediction”
 - At test time, they autoregressively predict the next token conditioned on their own previous outputs.

LM_{θ} (next response token \hat{r}_i | previous response tokens $\hat{r}_{<i}$; prompt p)

- At training time, they are trained with “teacher forcing” on a human-generated text dataset \mathcal{D} .

$$\text{LM Loss} = -\mathbb{E}_{\mathcal{D}} \left[\sum_{i=1,2,\dots} \log LM_{\theta}(\hat{r}_i | r_{<i}; p) \right]$$

Compare this to:

$$\text{BC Loss} = -\frac{1}{N} \sum_{i=1}^N \left(\sum_{t=1}^T \log \pi_{\theta}(a_{i,t} | s_{i,t}) \right)$$

- Incidentally, after pre-training as above, LMs are typically finetuned with “reinforcement learning from human feedback” (RLHF).