Robot Learning: Algorithms and Applications

Abdeslam Boularias

Carnegie Mellon University

National Robotics Engineering Center

Outline

1. Overview
2. Optimal control
3. Inverse optimal control
4. Grasping
5. Manipulation
6. Navigation

Outline

1. **Overview**
2. Optimal control
3. Inverse optimal control
4. Grasping
5. Manipulation
6. Navigation

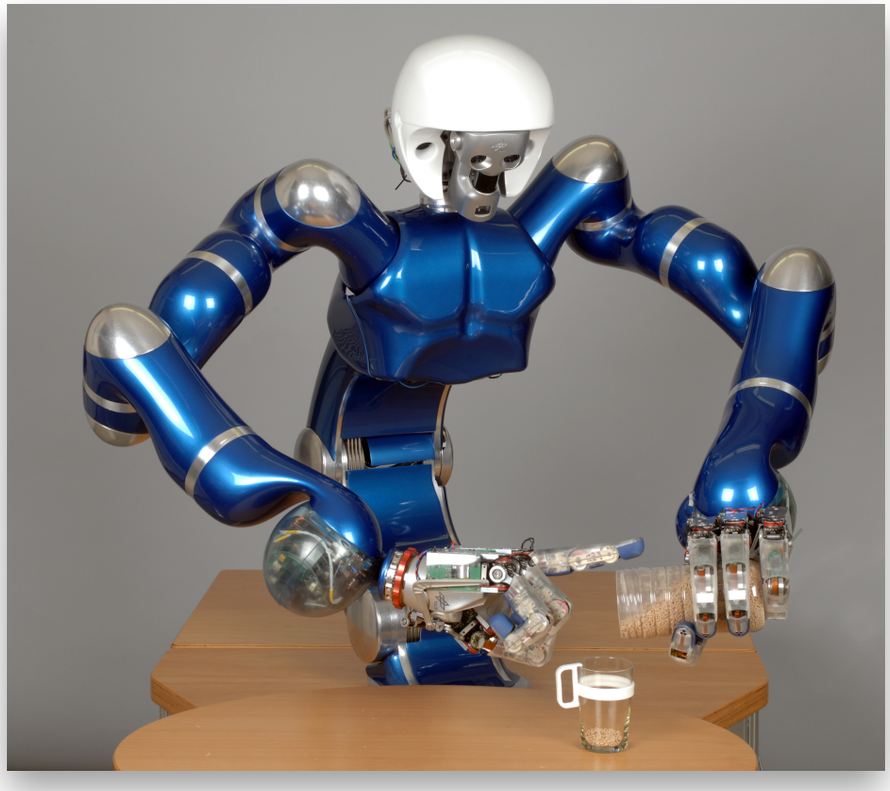
DARPA Robotics Challenge 2015

DARPA Robotics Challenge

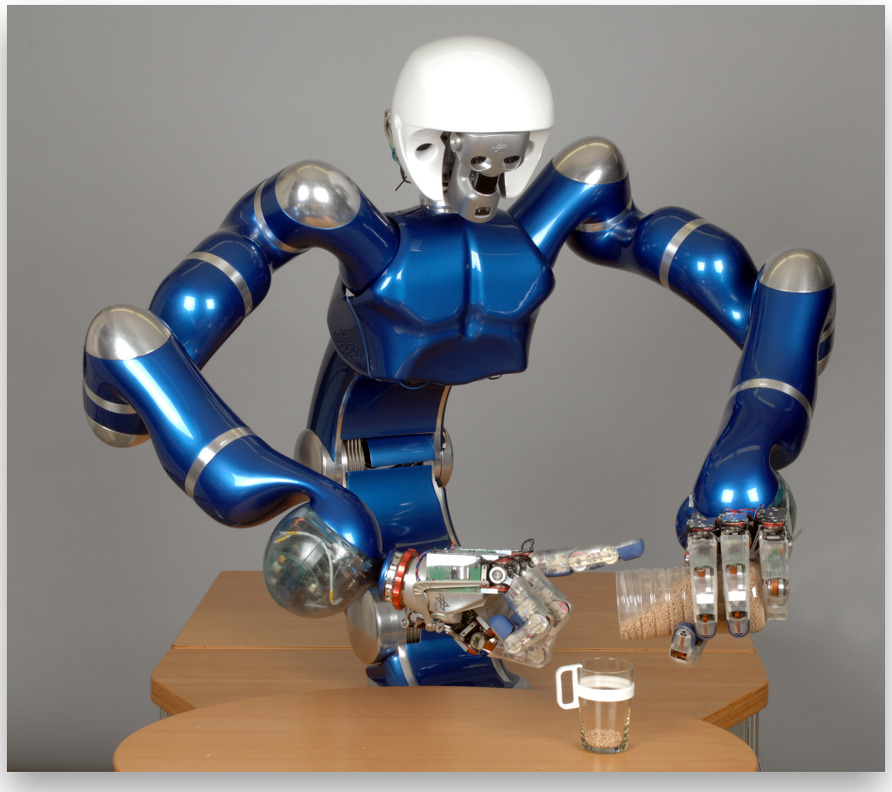
PIs: Tony Stentz, Alonzo Kelly, Herman Herman, Eric Meyhofer
Systems Lead: David Stager

DARPA PM: Dr. Gill Pratt

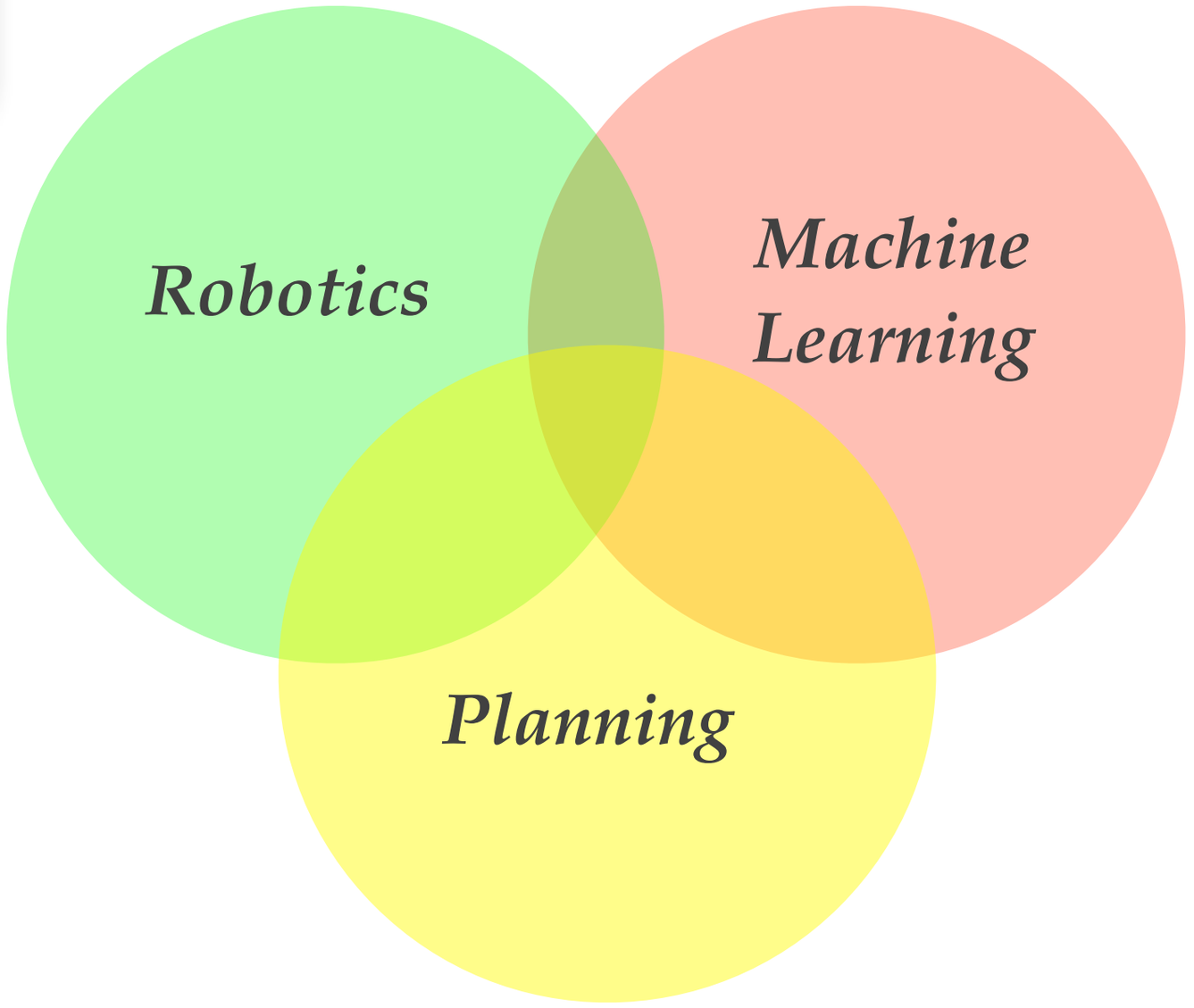
Unstructured Environments



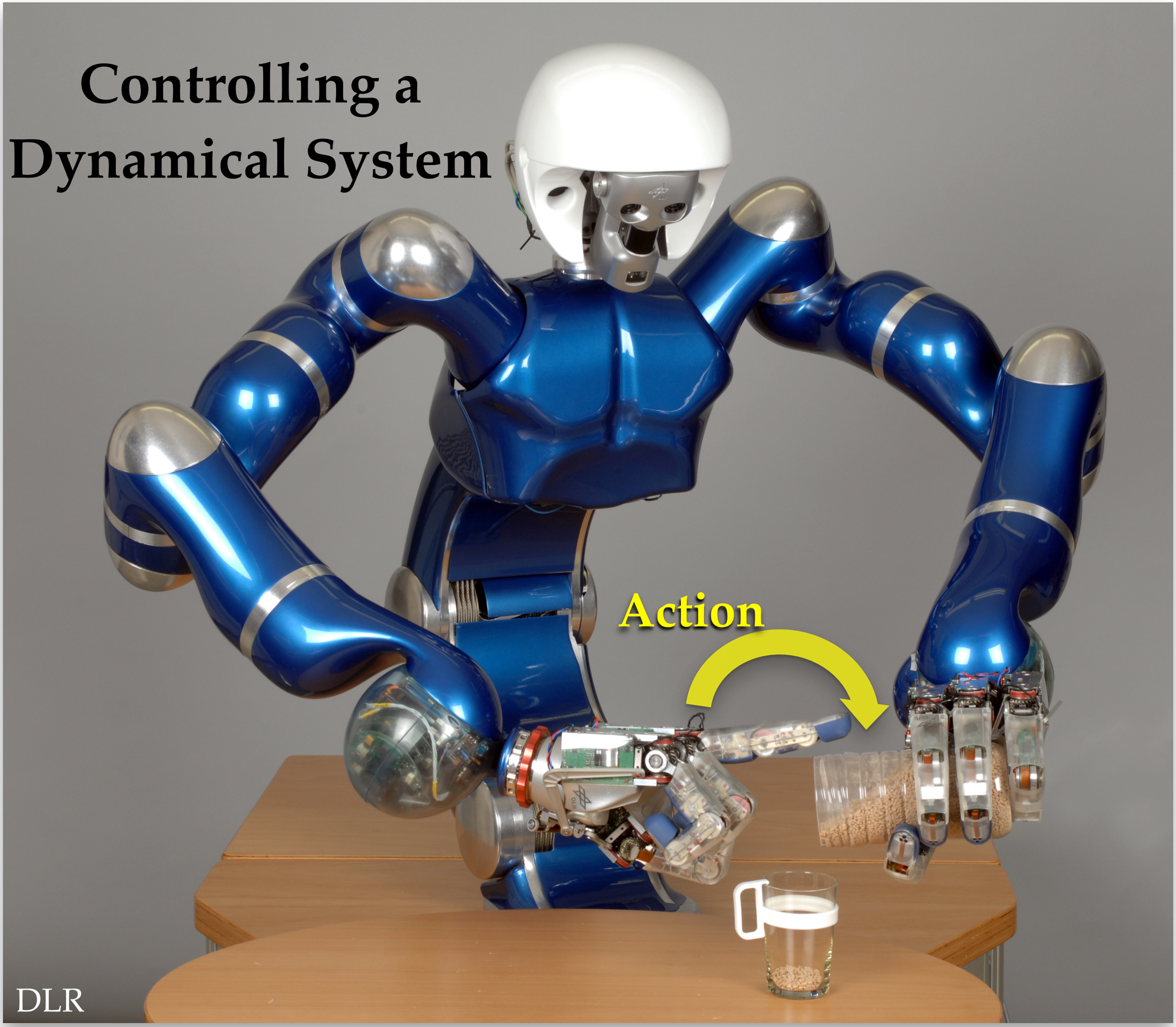
How can a robot learn to perform complex tasks from data?



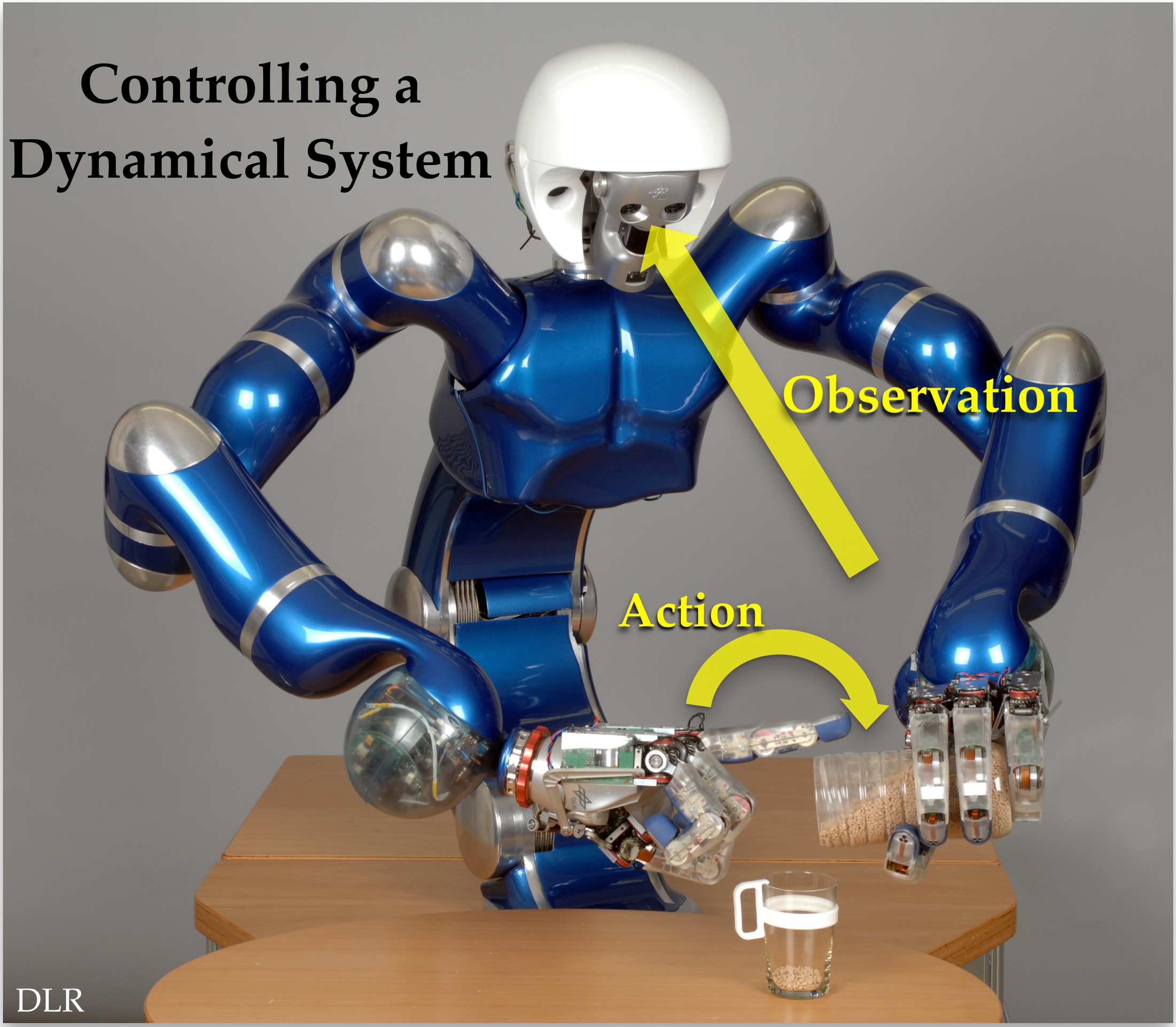
How can a robot learn to perform complex tasks from data?



Controlling a Dynamical System



Controlling a Dynamical System



Outline

1. Overview

2. Optimal control

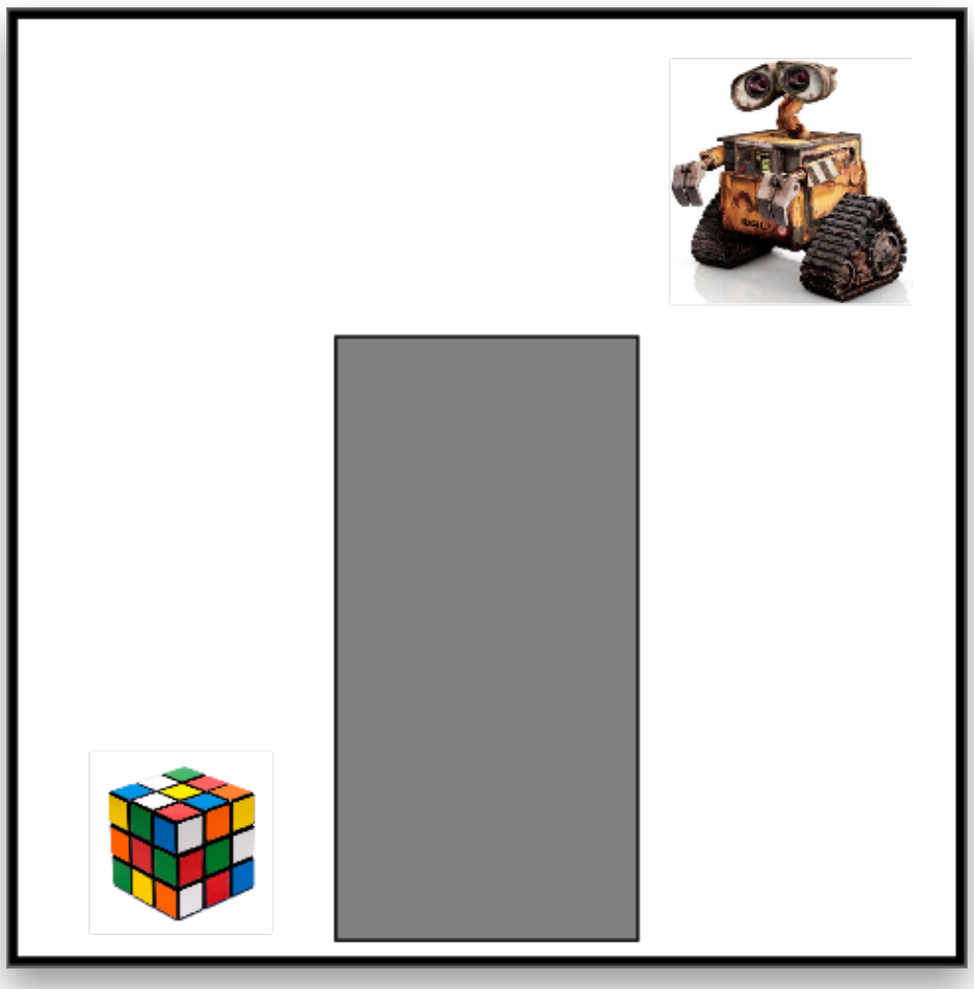
3. Inverse optimal control

4. Grasping

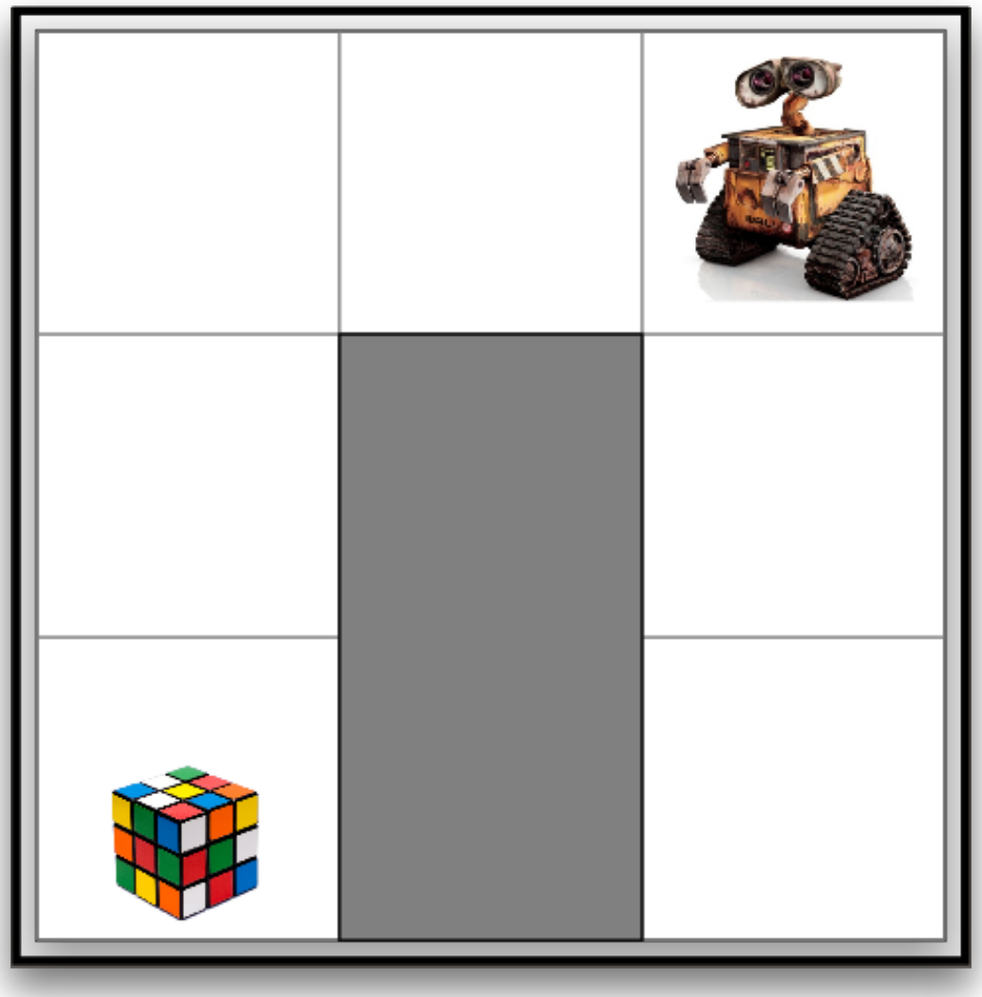
5. Manipulation

6. Navigation

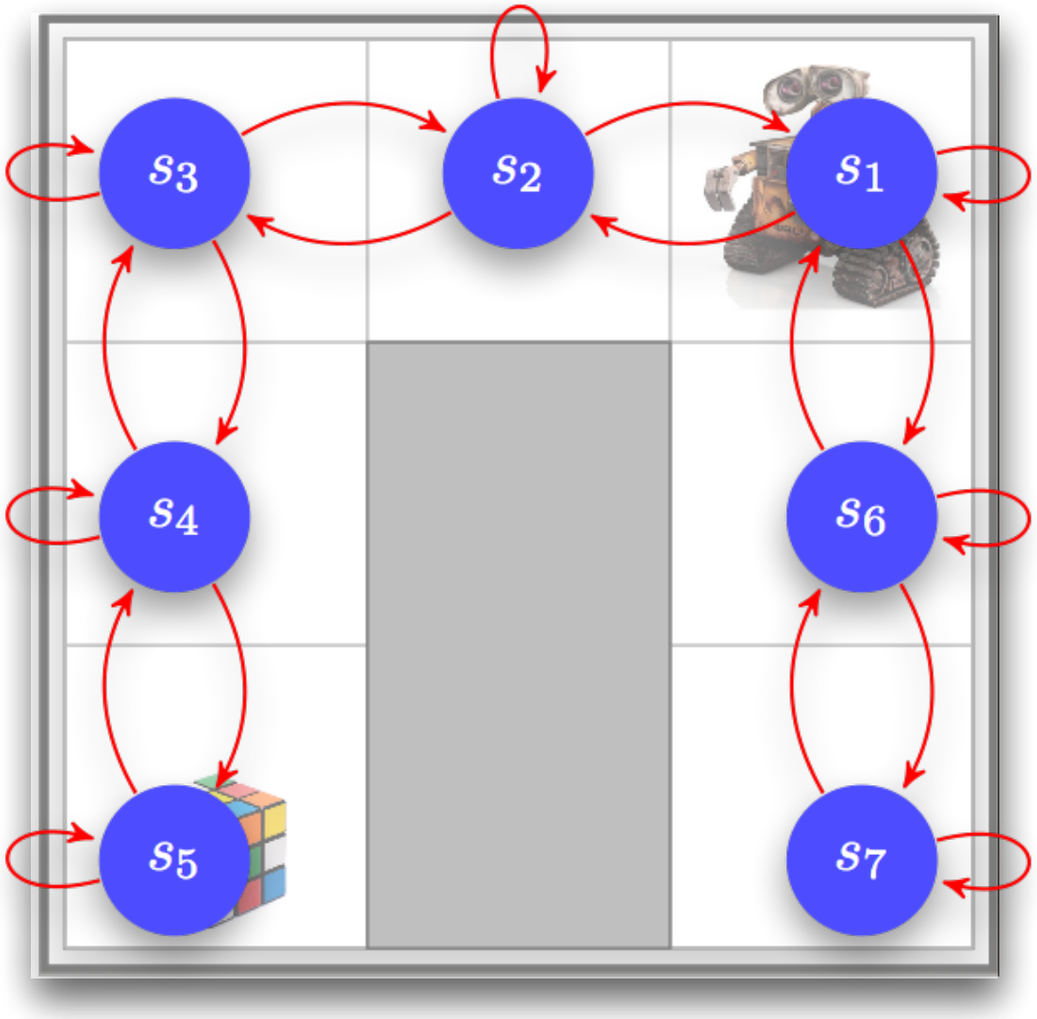
Path planning: a simple control problem



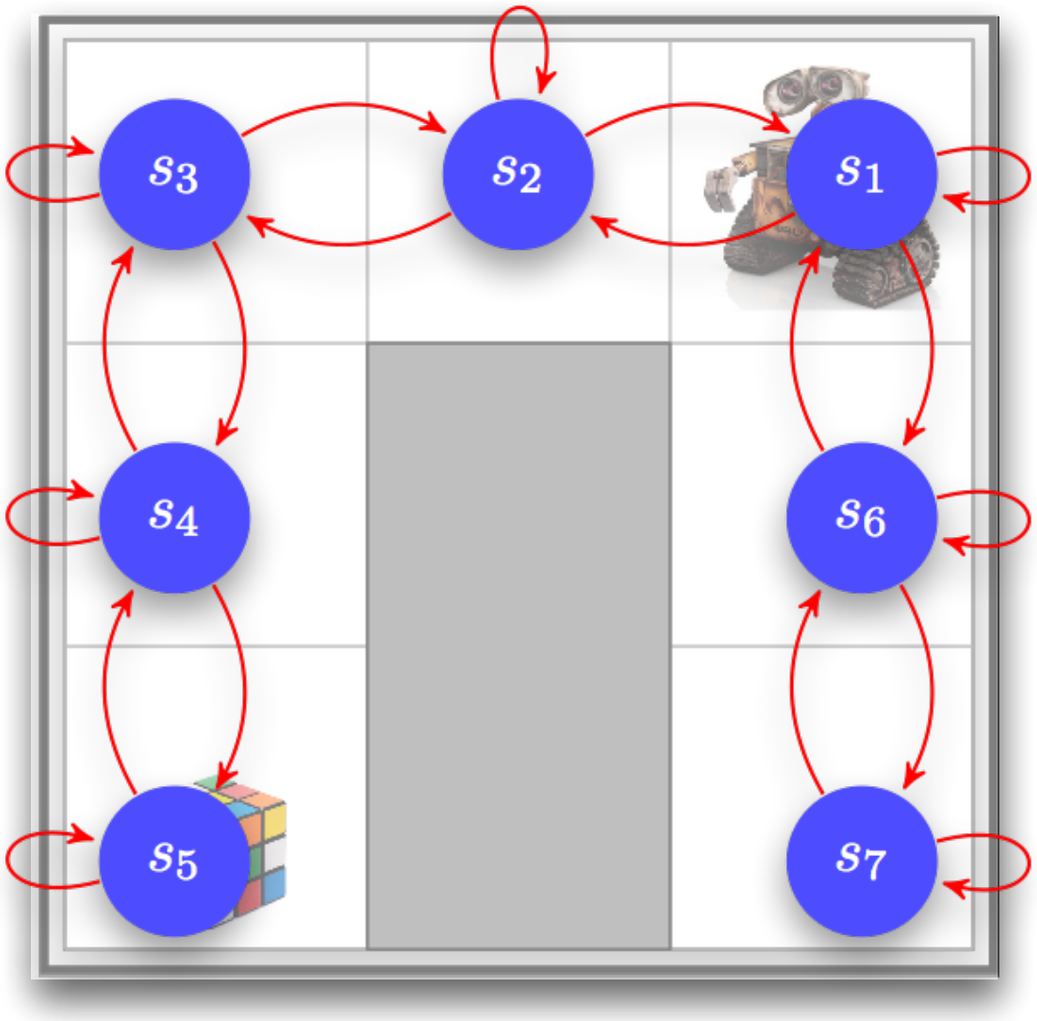
Path planning: a simple control problem



States and Actions



States and Actions



Andrey Markov
(1856–1922)

Markov Decision Process (MDP)

Notations

- S: set of states (e.g. position and velocity of the robot)
- A: set of actions (e.g. force)
- T: stochastic transition function

$$T(s, a, s') = Pr(s_{t+1} = s' | s_t = s, a_t = a)$$

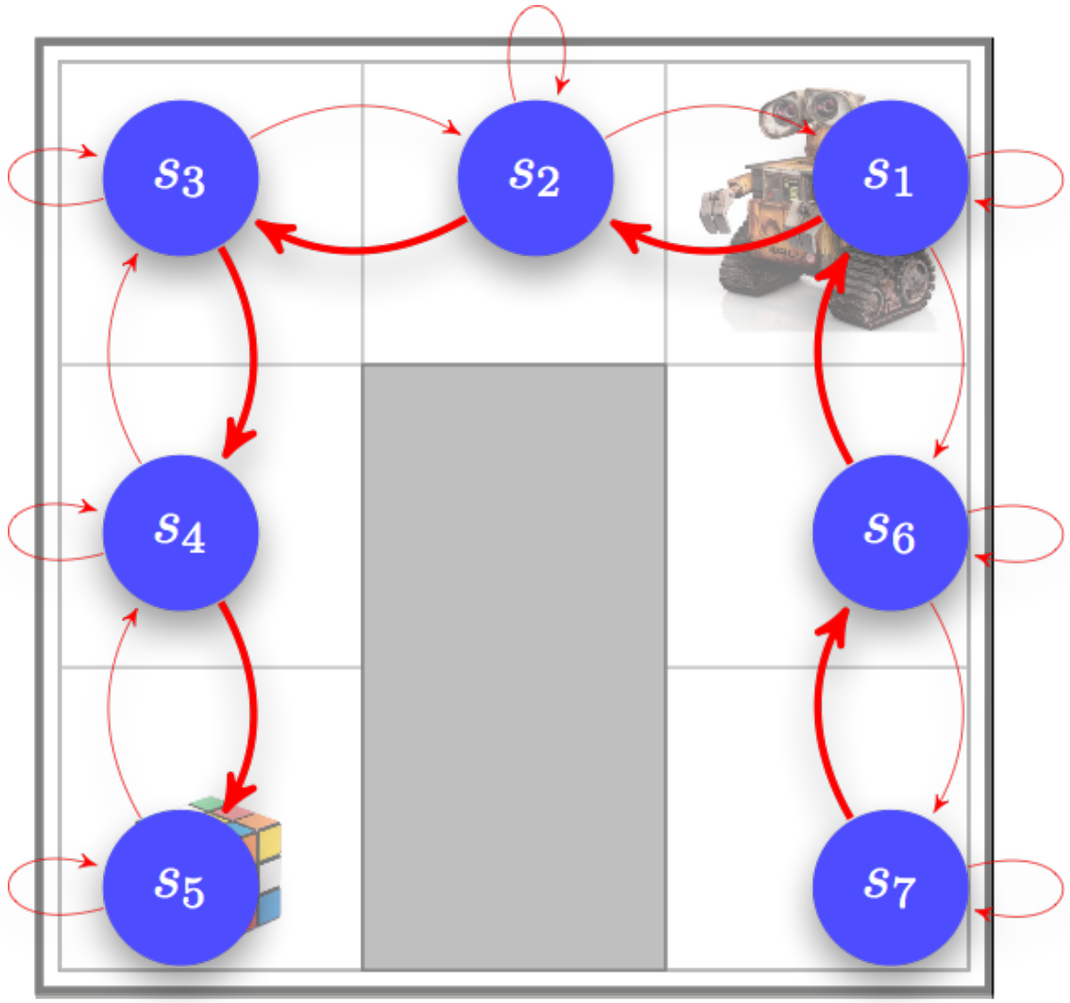
next state **current state** **current action**

- R: reward (or cost) function, $R(s, a) \in \mathbb{R}$

Policies (strategies)

A policy is a function π that maps each state to an action,

$$\pi : \mathcal{S} \to \mathcal{A}.$$



Value function

The ***value*** of policy π is the sum of rewards that one expects to gain by following it:

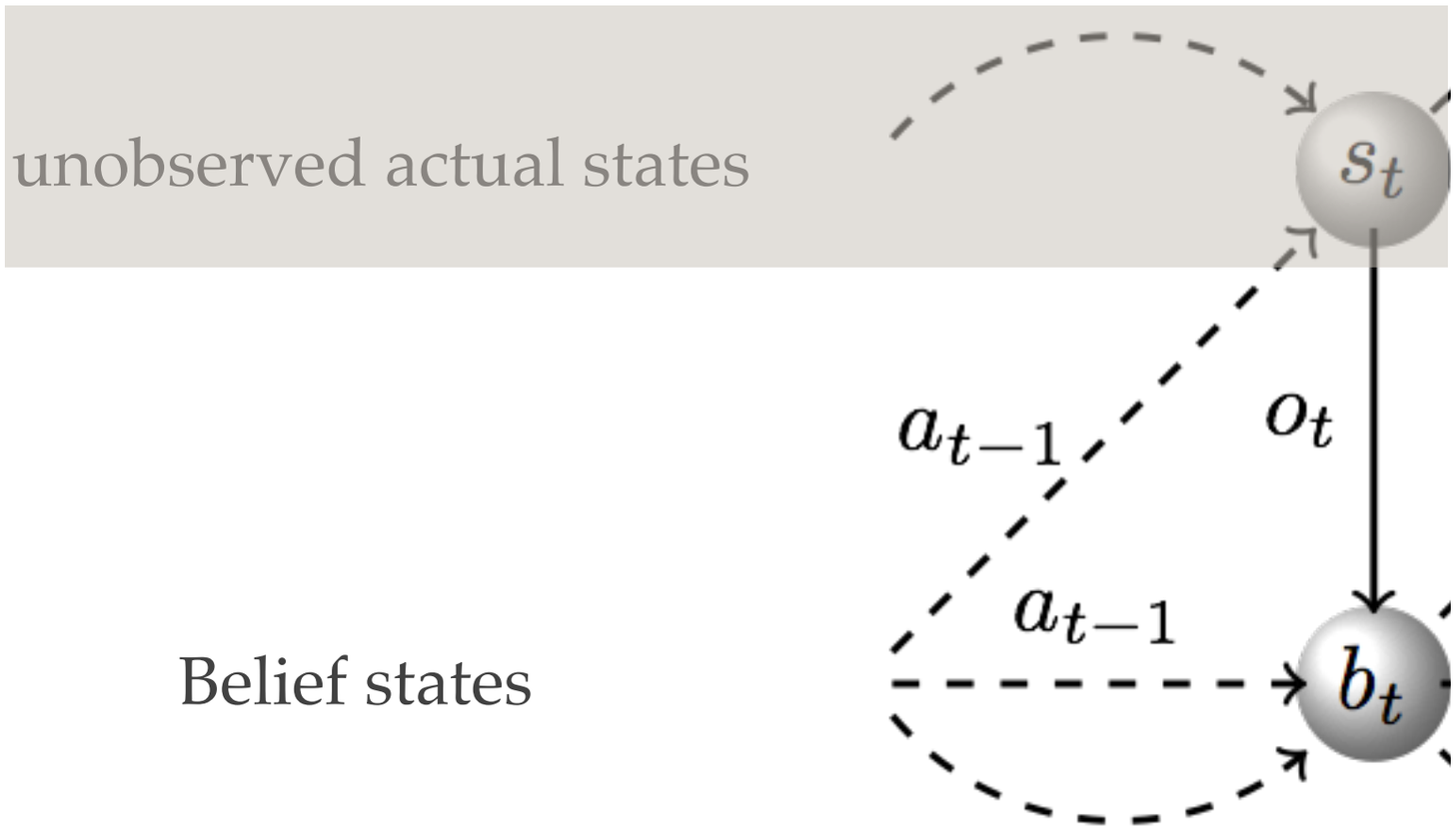
$$V(\pi) = \sum_{t=0}^{H} \mathbb{E}_{s_t} \Big[R(s_t, \pi(s_t)) \Big]$$

reward

Goal: finding an **optimal policy**

Partially Observable Markov Decision Process

- Observations are ***partial*** and ***noisy***.
- States cannot be precisely known.
- ***Belief state:*** a probability distribution on states

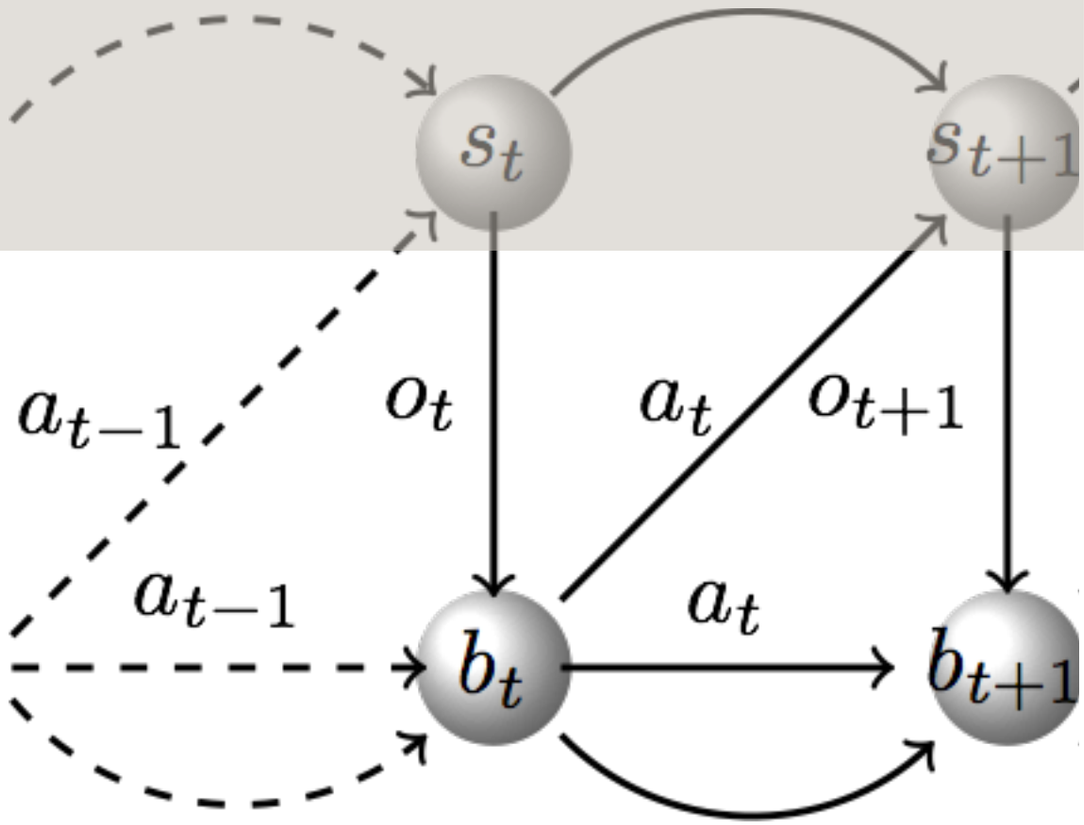


Partially Observable Markov Decision Process

- Observations are ***partial*** and ***noisy***.
- States cannot be precisely known.
- ***Belief state:*** a probability distribution on states

unobserved actual states

Belief states

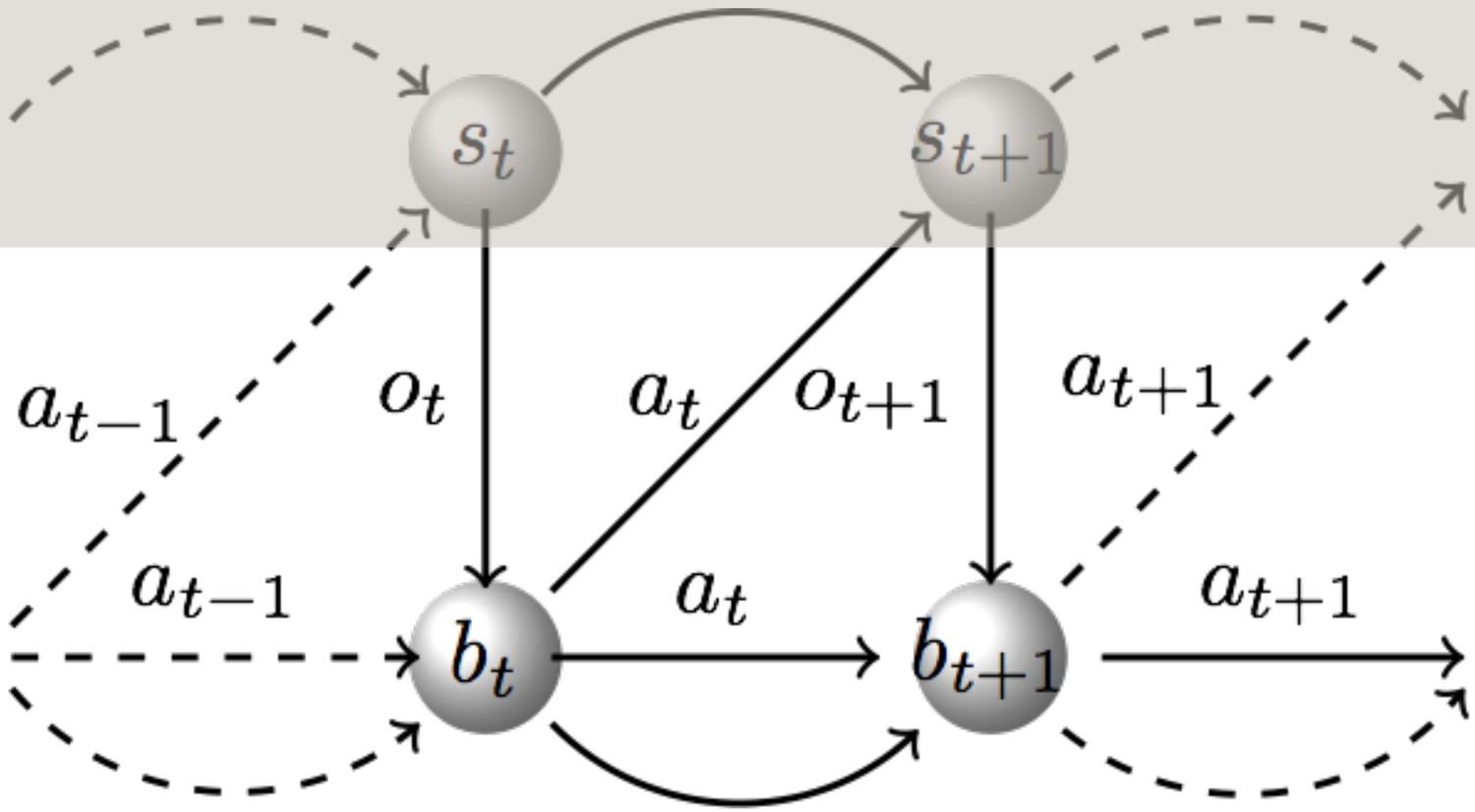


Partially Observable Markov Decision Process

- Observations are ***partial*** and ***noisy***.
- States cannot be precisely known.
- ***Belief state:*** a probability distribution on states

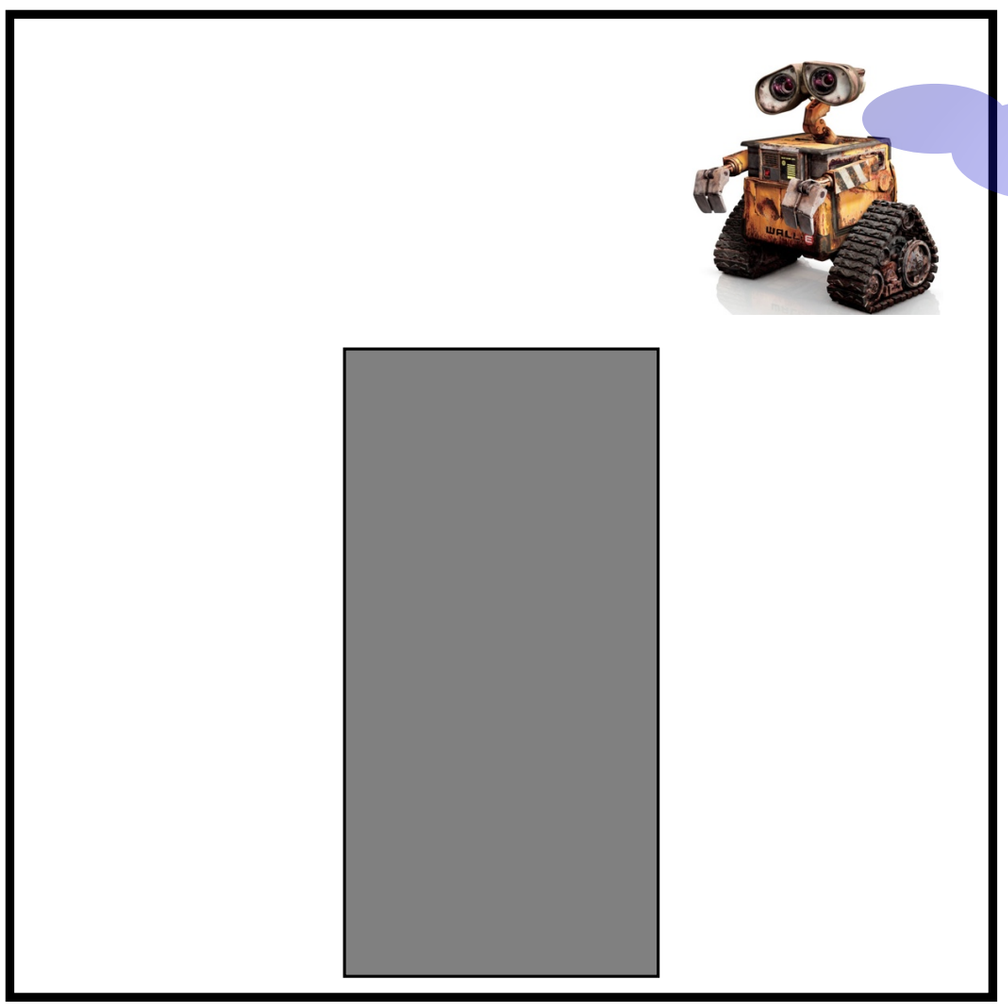
unobserved actual states

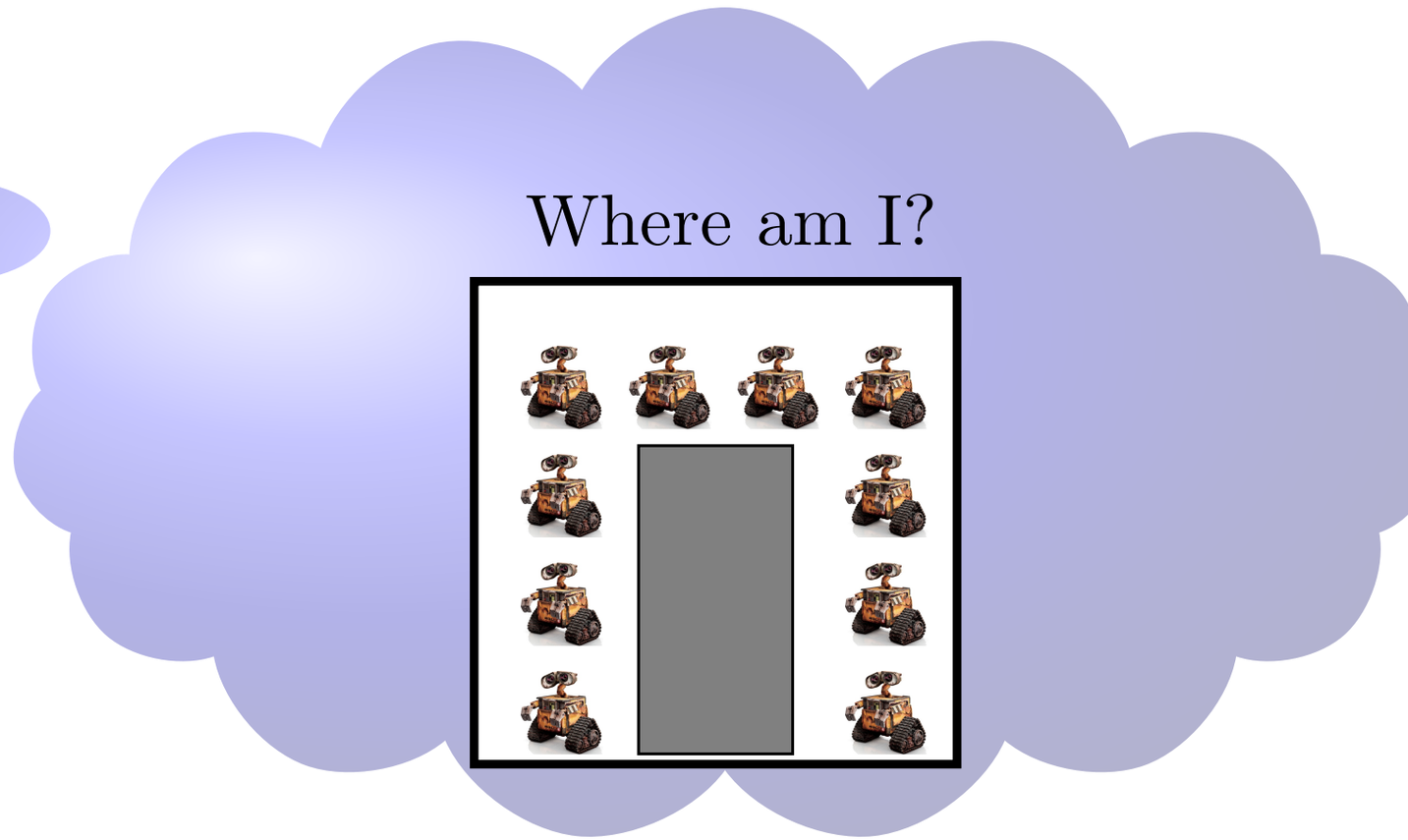
Belief states



Partially Observable Markov Decision Process

Example: the robot can sense an obstacle only after bumping into it.

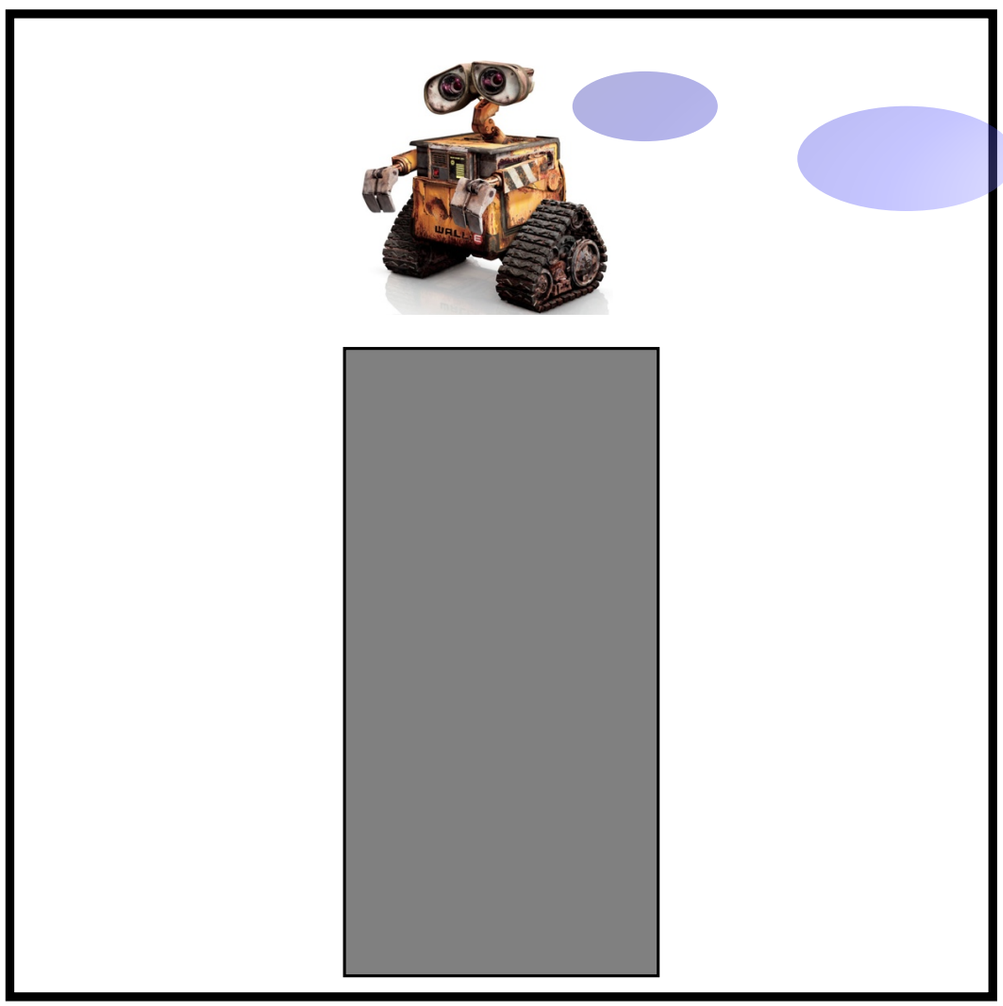


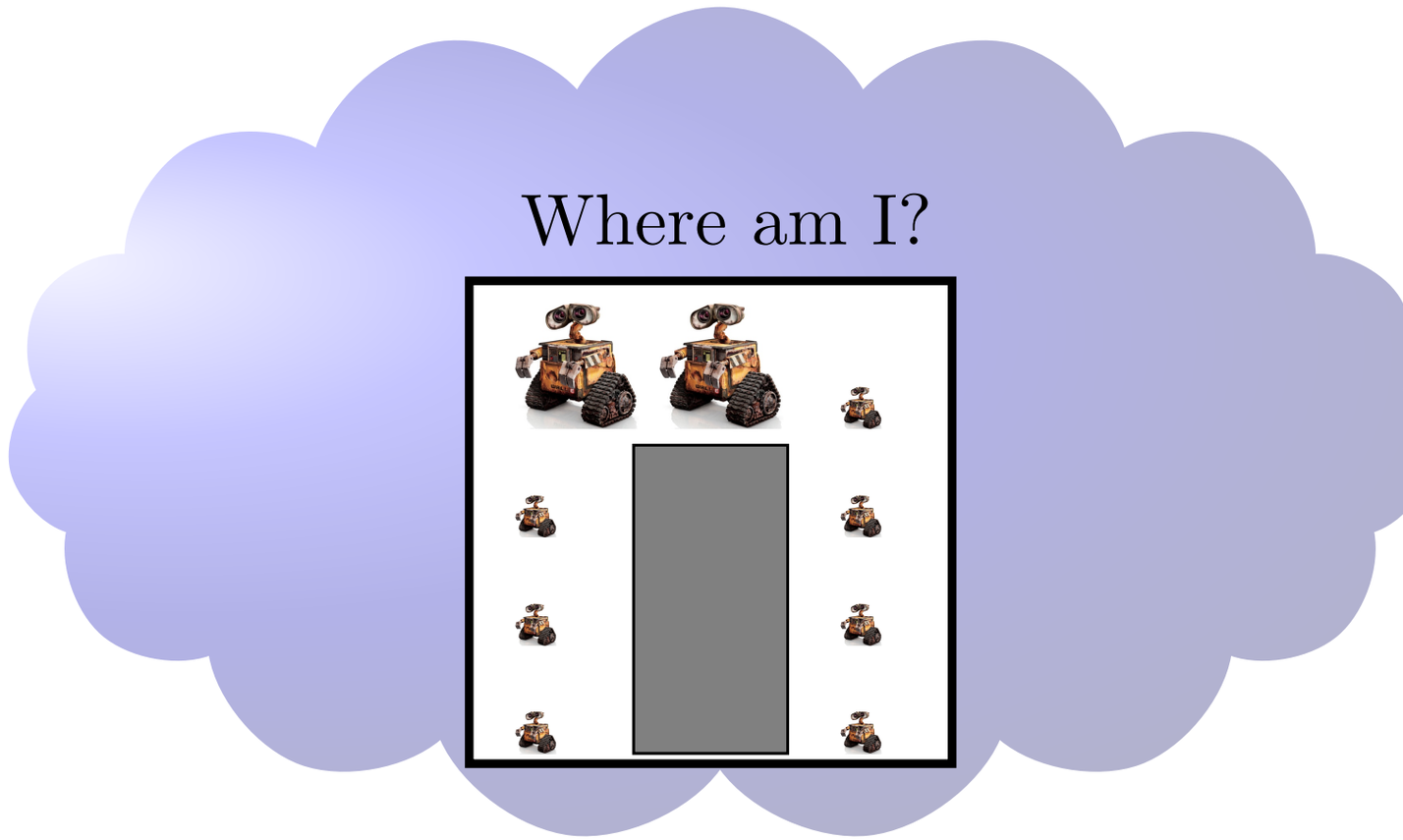


Belief state

Partially Observable Markov Decision Process

Example: the robot can sense an obstacle only after bumping into it.

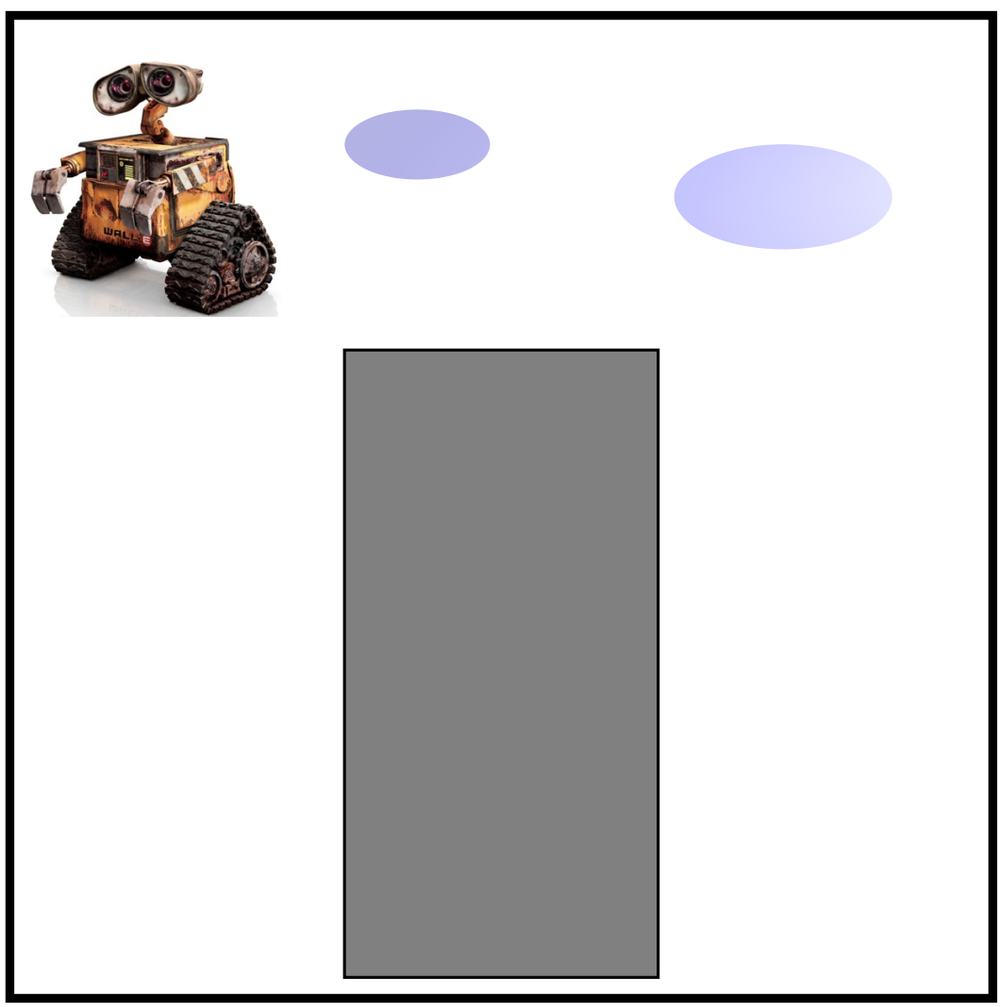


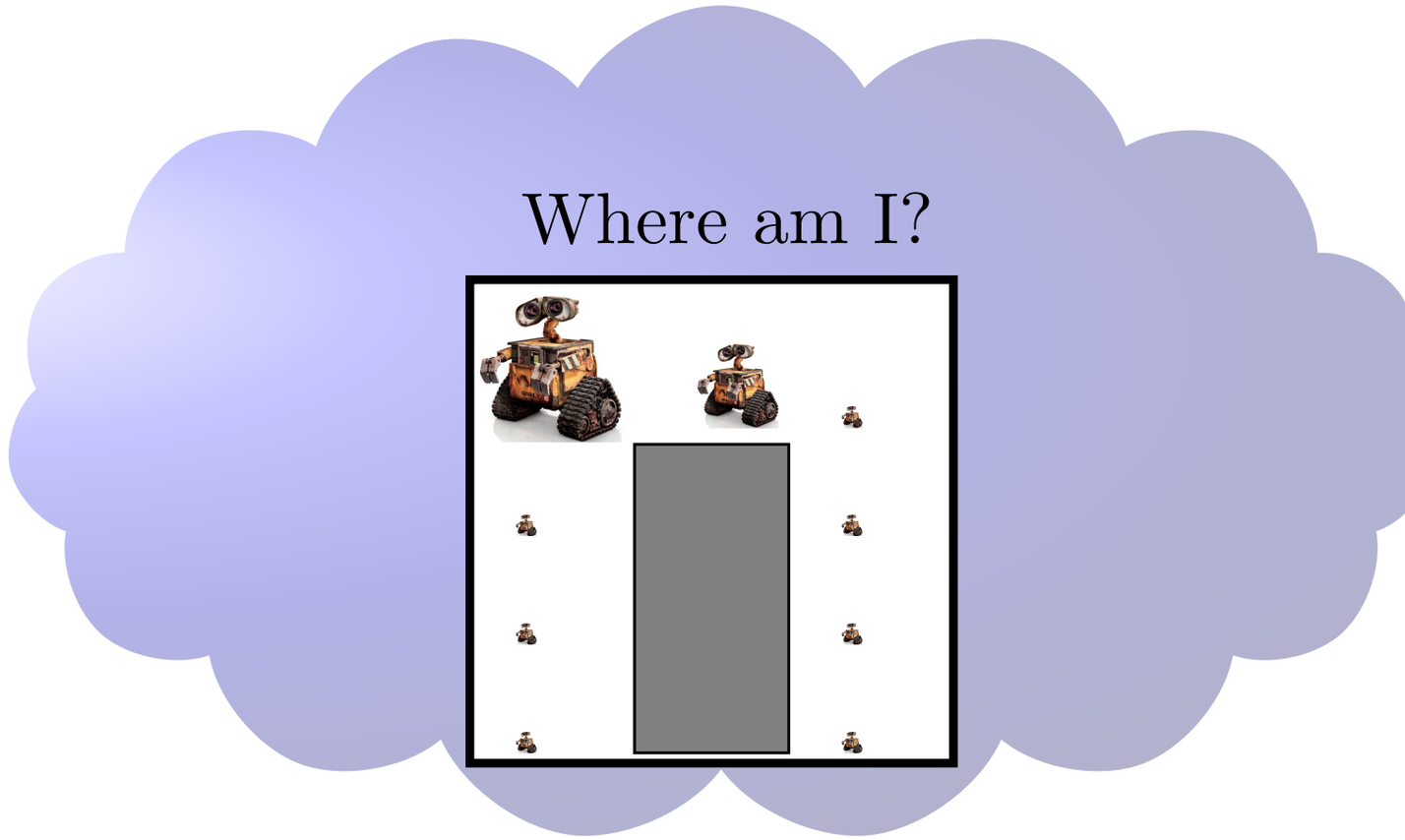


Belief state

Partially Observable Markov Decision Process

Example: the robot can sense an obstacle only after bumping into it.

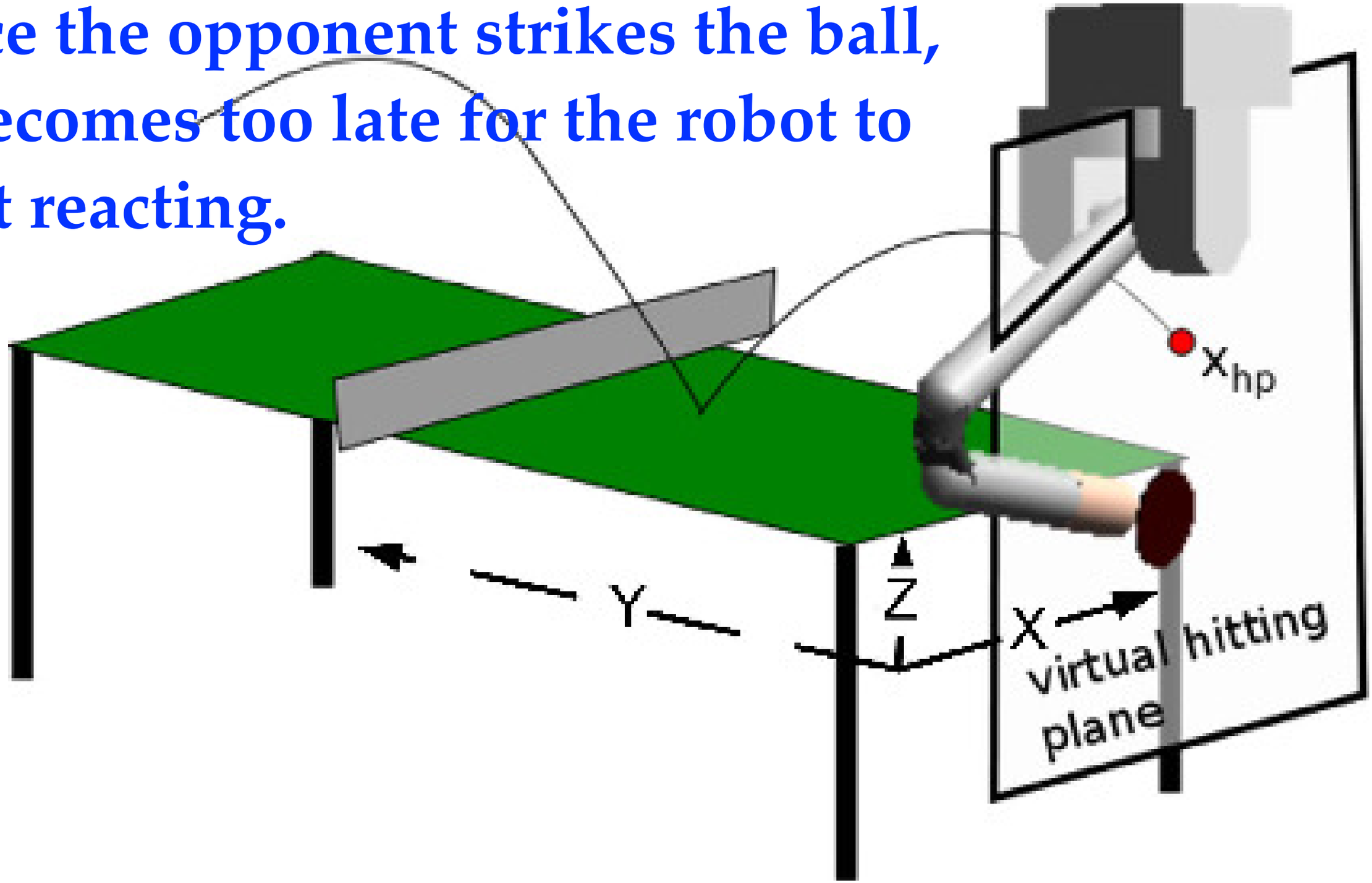




Belief state

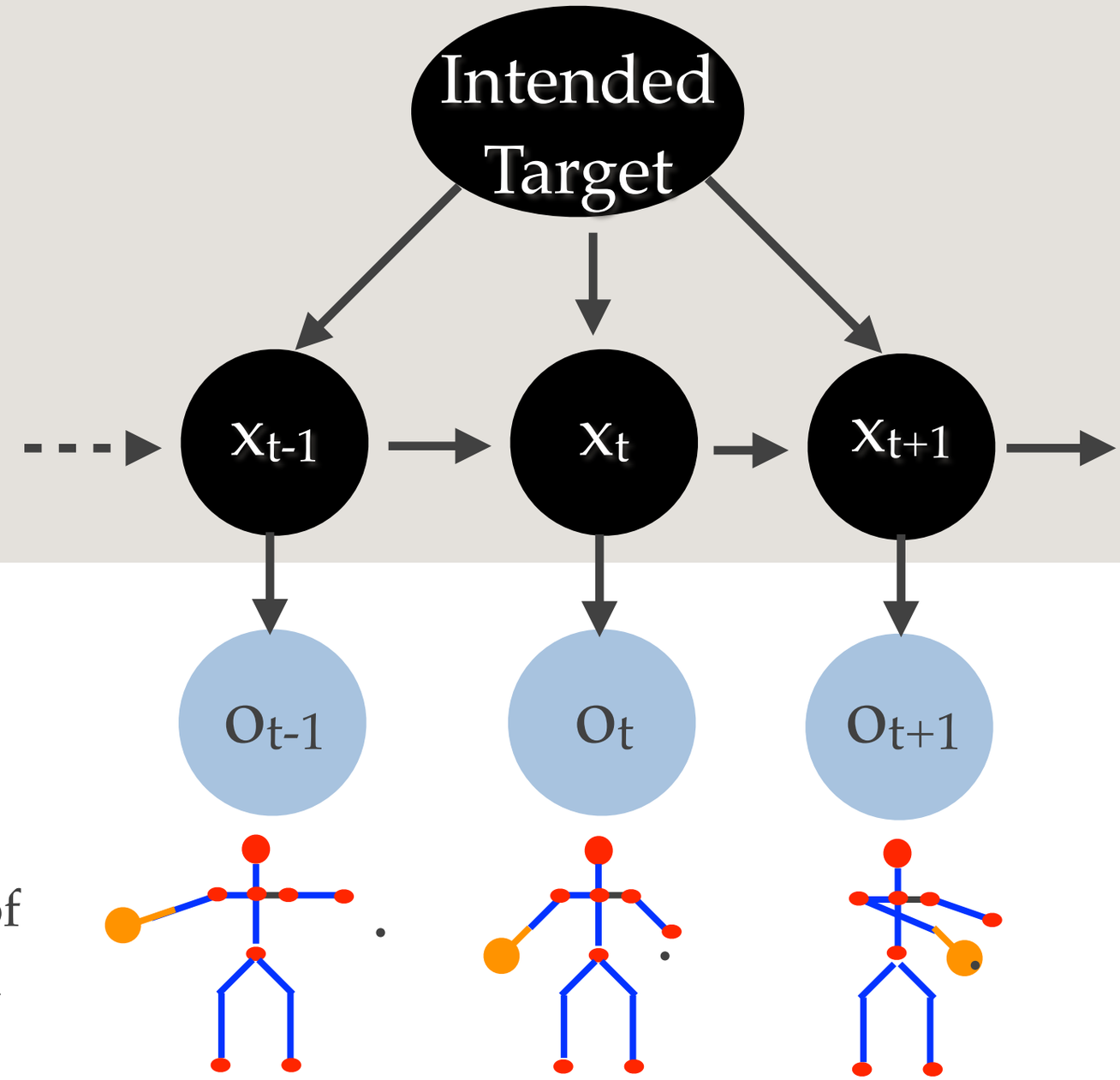
The target of the ball should be predicted in advance

Once the opponent strikes the ball, it becomes too late for the robot to start reacting.



Probabilistic graphical model of intention-driven dynamics

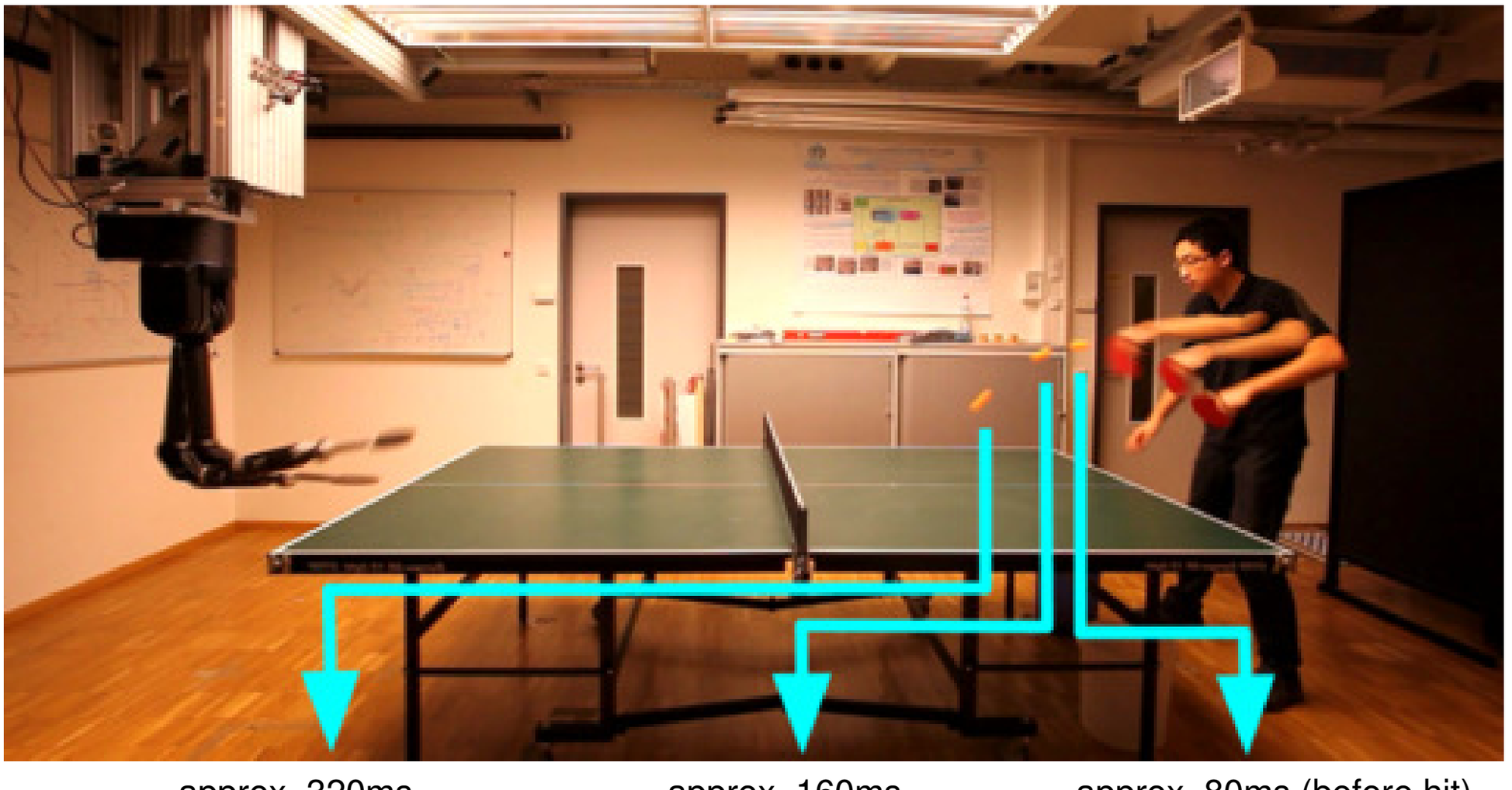
Hidden variables

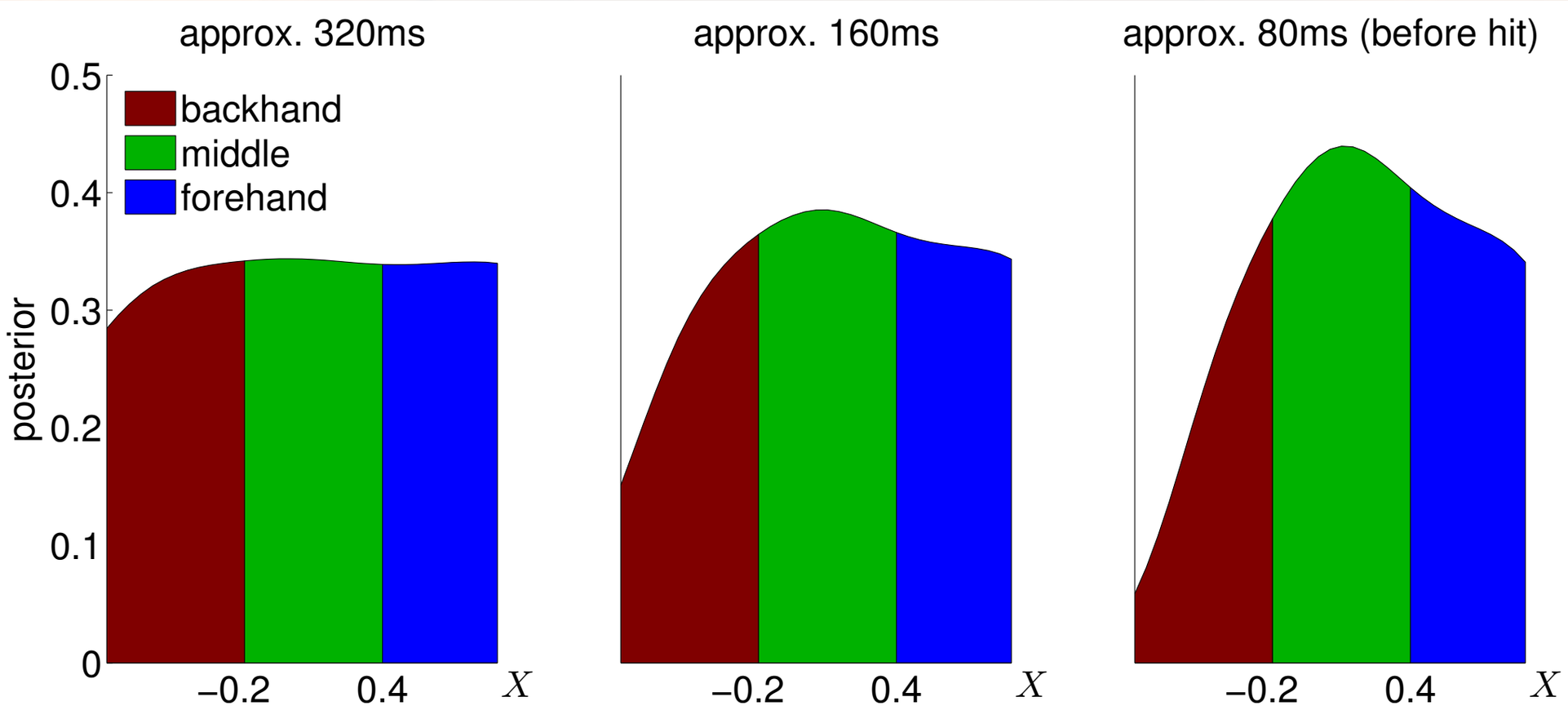


Observations

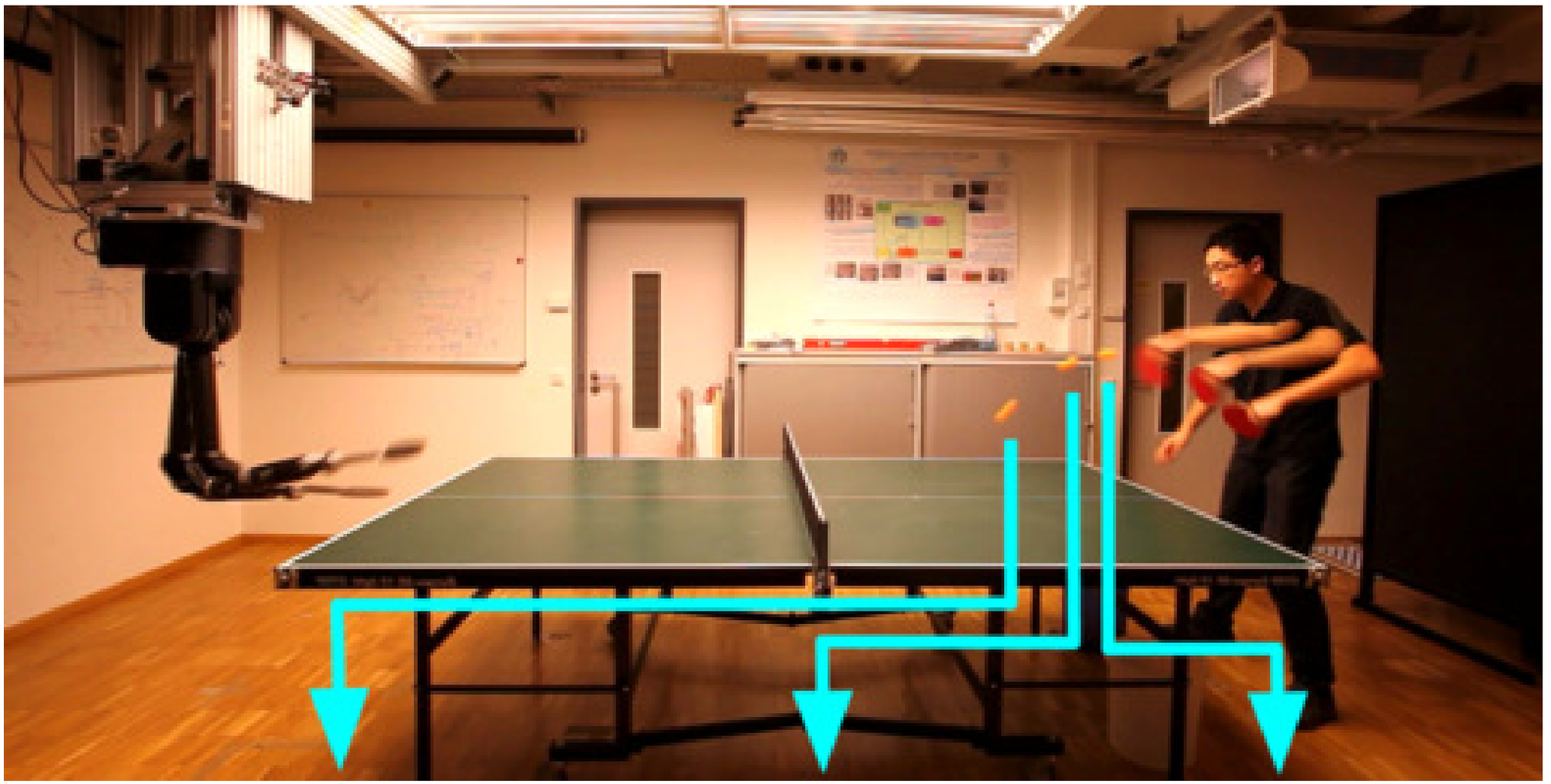
Positions of the ball, the racket and joints of the opponent, tracked using a *Kinect* camera

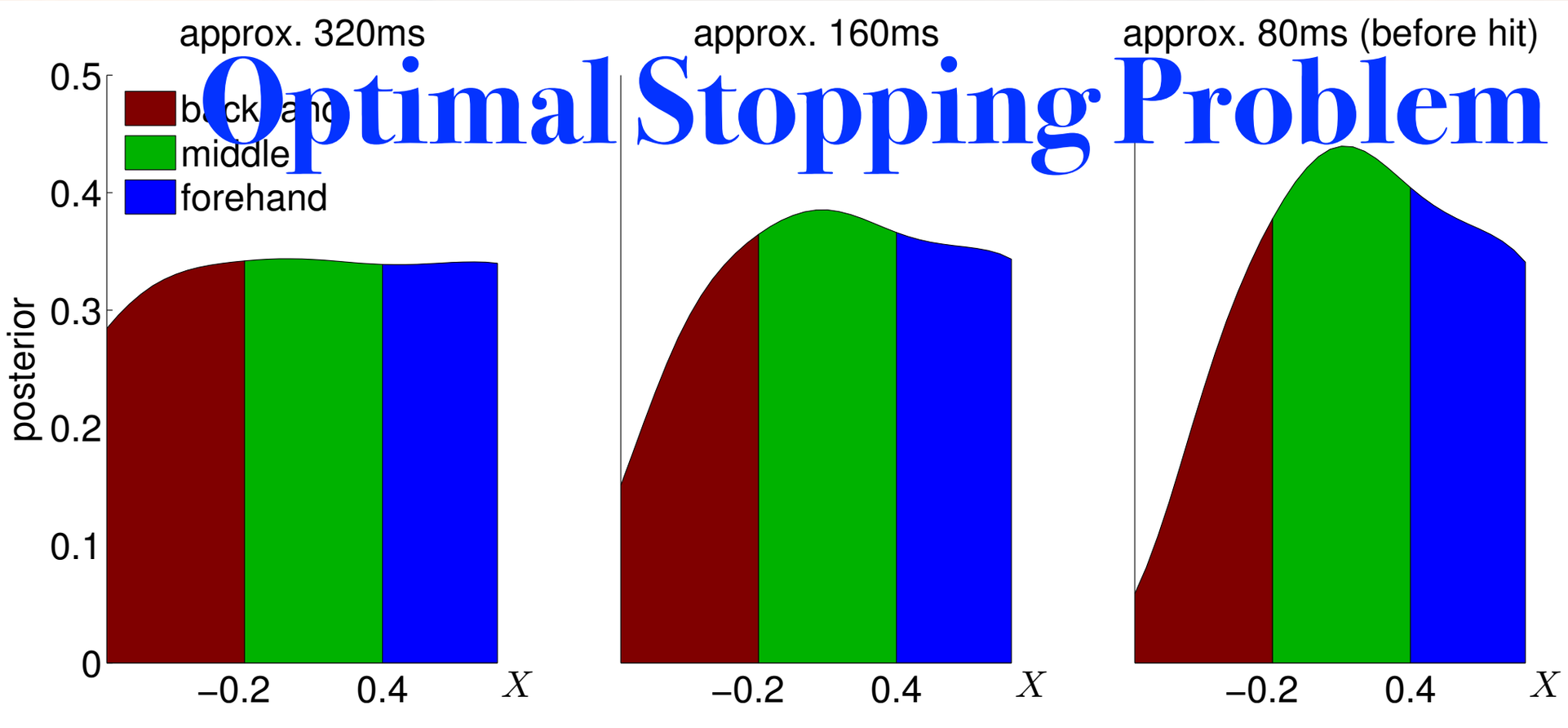
Observations o_t are generated according to a *Gaussian Process*.




approx. 320ms
approx. 160ms
approx. 80ms (before hit)
backhand
middle
forehand
posterior
0.5
0.4
0.3
0.2
0.1
0
−0.2
0.4
X

Optimal Stopping Problem





A Monte-Carlo planning algorithm

Sample current state and intention $s_t = [x_t, \text{target}]$

A Monte-Carlo planning algorithm

Sample current state and intention $s_t = [x_t, \text{target}]$

Sample subsequent state $x_{t+1} \sim P(\,.\,|\,s_t)$

A Monte-Carlo planning algorithm

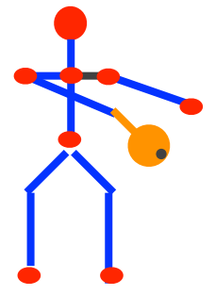
Sample current state and intention $s_t = [x_t, \text{target}]$

↓

Sample subsequent state $x_{t+1} \sim P(\,.\,|\,s_t)$

↓

Sample subsequent observation $o_{t+1} \sim P(\,.\,|\,x_{t+1})$



A Monte-Carlo planning algorithm

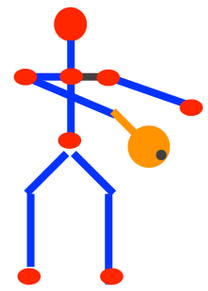
Sample current state and intention $s_t = [x_t, \text{target}]$

↓

Sample subsequent state $x_{t+1} \sim P(\,.\,|\,s_t)$

↓

Sample subsequent observation $o_{t+1} \sim P(\,.\,|\,x_{t+1})$



↓

Update belief b_{t+1} provided observation o_{t+1}

A Monte-Carlo planning algorithm

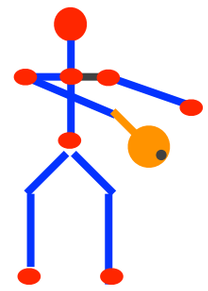
Sample current state and intention $s_t = [x_t, \text{target}]$

↓

Sample subsequent state $x_{t+1} \sim P(\,.\,|\,s_t)$

↓

Sample subsequent observation $o_{t+1} \sim P(\,.\,|\,x_{t+1})$



↓

Update belief b_{t+1} provided observation o_{t+1}

↓

Predict the expected performance $V(b_{t+1})$

A Monte-Carlo planning algorithm

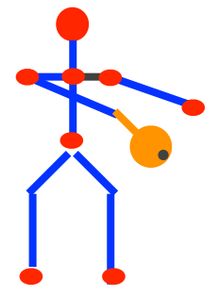
Sample current state and intention $s_t = [x_t, \text{target}]$

↓

Sample subsequent state $x_{t+1} \sim P(\,.\,|\,s_t)$

↓

Sample subsequent observation $o_{t+1} \sim P(\,.\,|\,x_{t+1})$



↓

Update belief b_{t+1} provided observation o_{t+1}

↓

Predict the expected performance $V(b_{t+1})$

↓

enough samples?

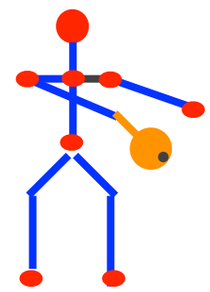
No

A Monte-Carlo planning algorithm

Sample current state and intention $s_t = [x_t, \text{target}]$

Sample subsequent state $x_{t+1} \sim P(\,.\,|\,s_t)$

Sample subsequent observation $o_{t+1} \sim P(\,.\,|\,x_{t+1})$



Update belief b_{t+1} provided observation o_{t+1}

Predict the expected performance $V(b_{t+1})$

enough samples?

No

Yes

$V(b_t) > V(b_{t+1})$

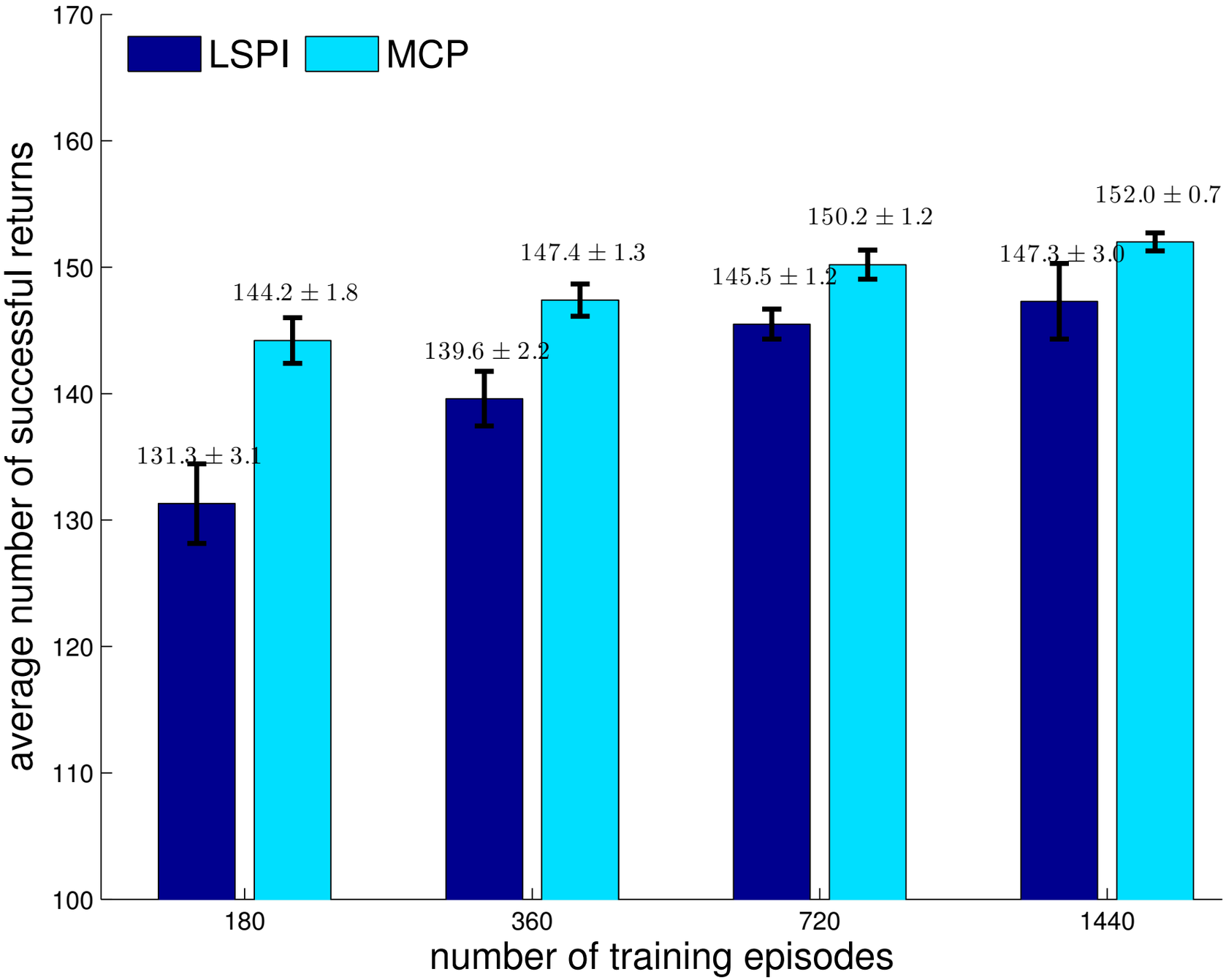
No

Wait and see

Yes

strike back!

Average number of successful returns



Z. Wang, A. Boularias *et al.* (2015) in *Artificial Intelligence Journal.*

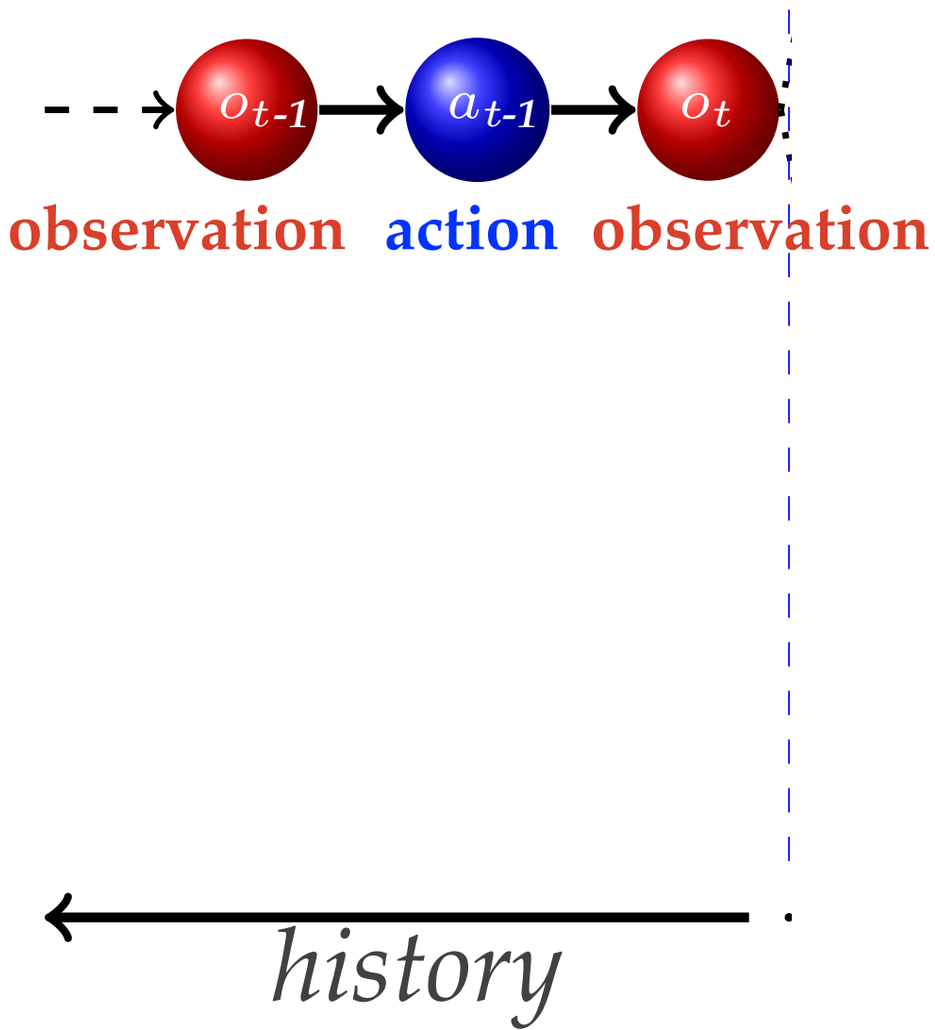
The Curse of Dimensionality

- Dimension of the belief state is **exponential** in the number of state variables.
- **Planning time** is polynomial in the dimension of the belief state.

Richard Bellman (1957)

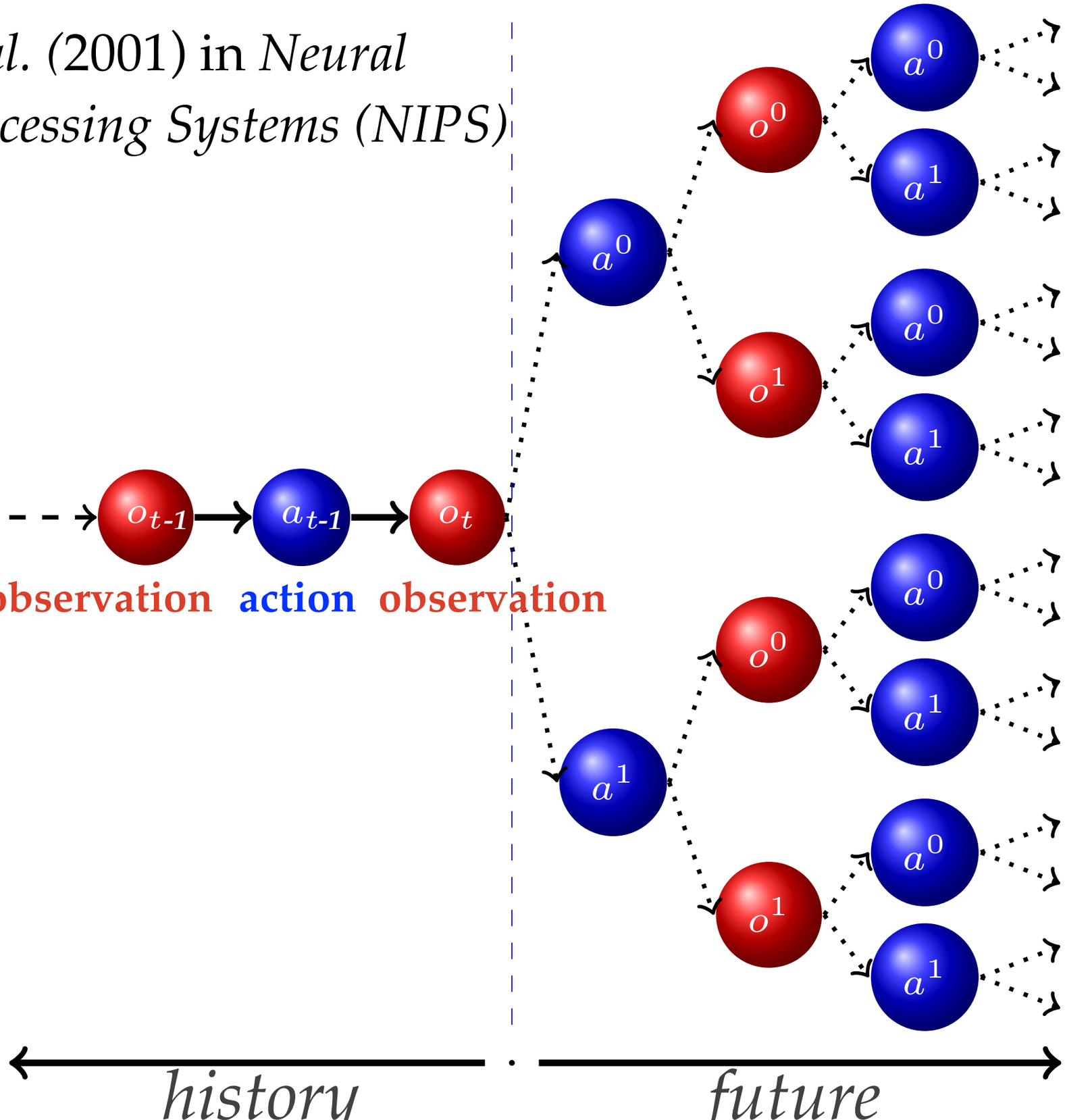
Predictive State Representations

M. Littman *et al.* (2001) in *Neural Information Processing Systems (NIPS)*



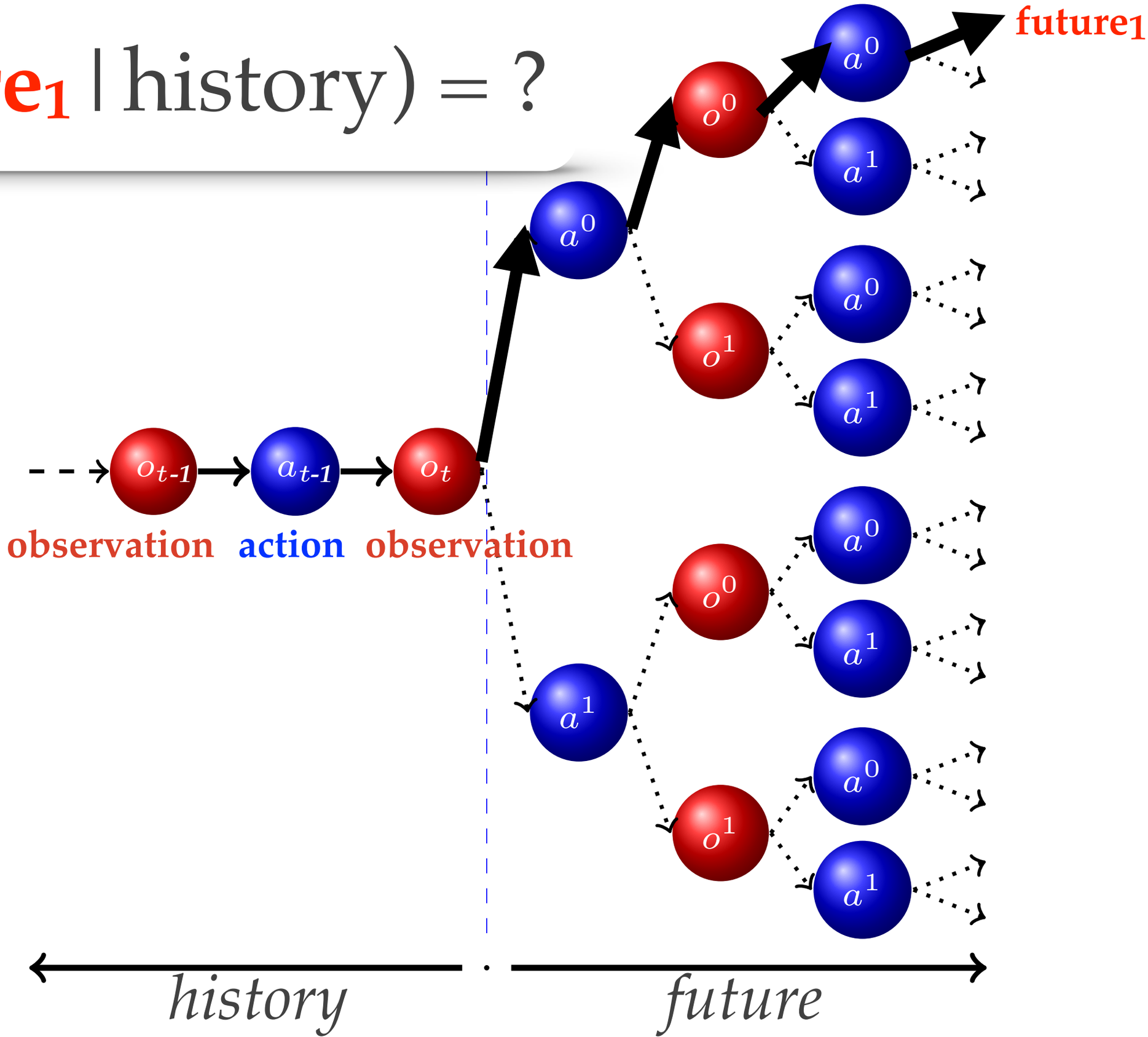
Predictive State Representations

M. Littman *et al.* (2001) in *Neural Information Processing Systems (NIPS)*



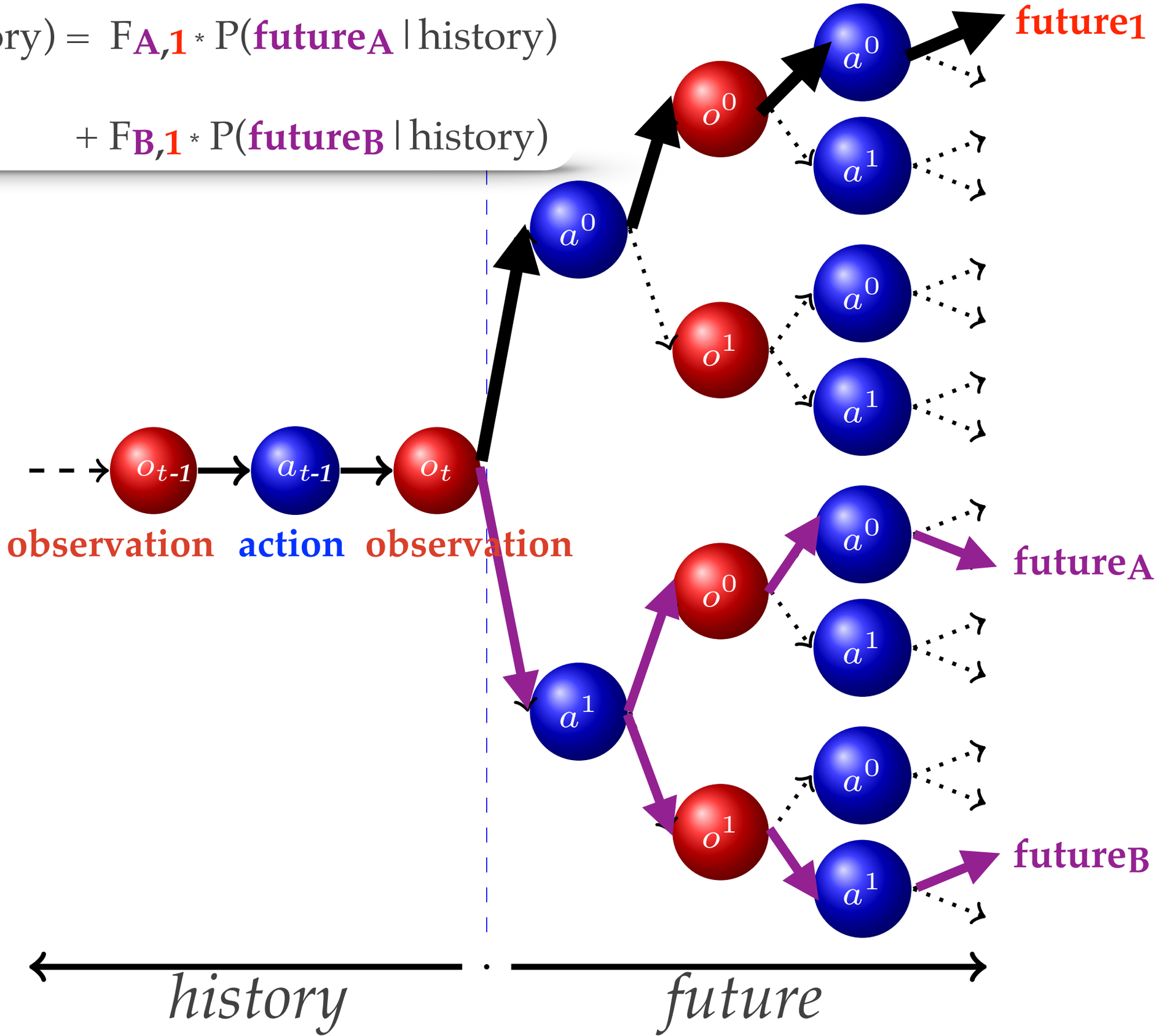
Predictive State Representations

$$P(\textbf{future}_1 \mid \text{history}) = ?$$



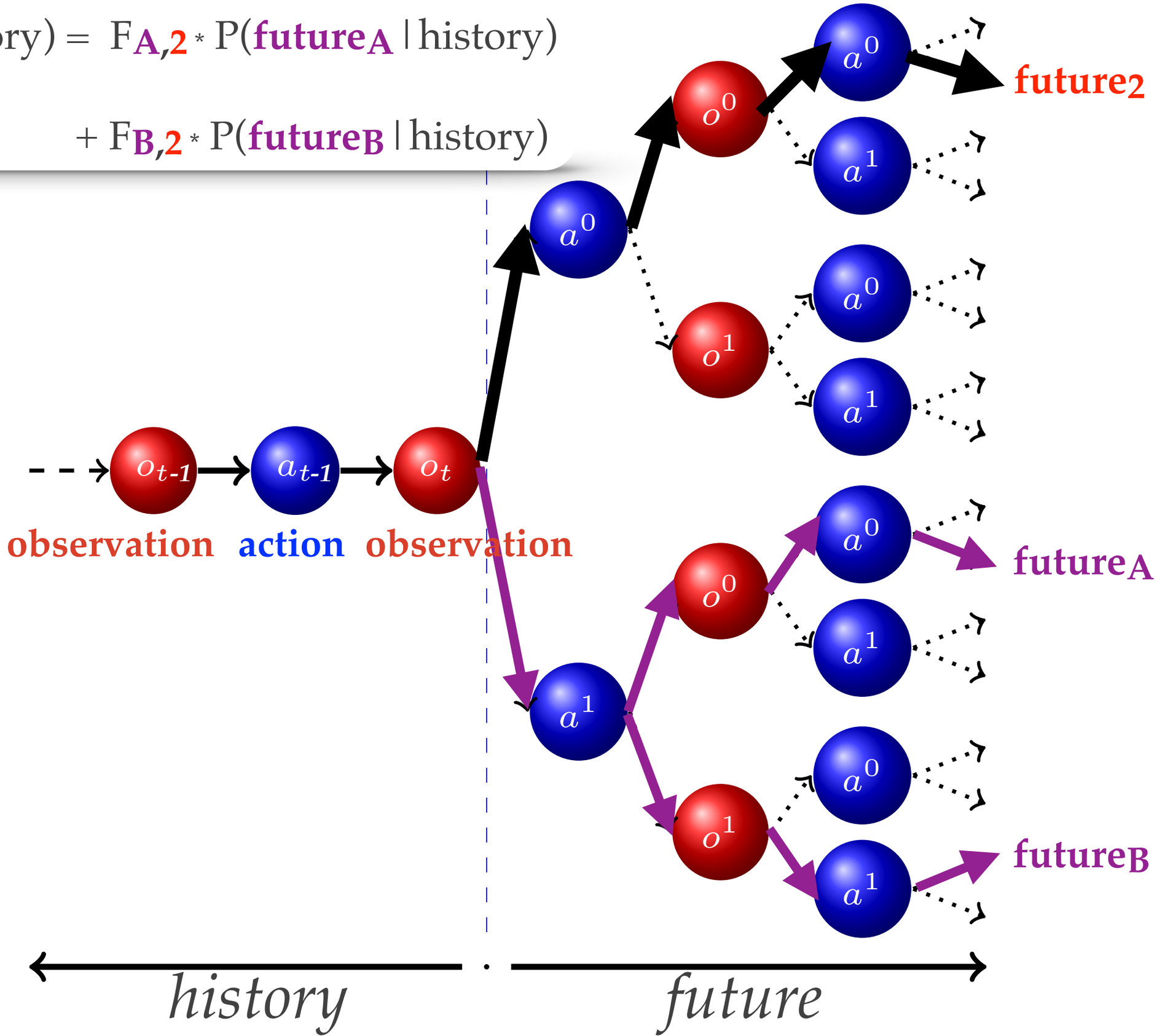
Predictive State Representations

$$P(\textbf{future}_1 \mid \text{history}) = F_{\textbf{A},1} * P(\textbf{future}_\textbf{A} \mid \text{history}) + F_{\textbf{B},1} * P(\textbf{future}_\textbf{B} \mid \text{history})$$



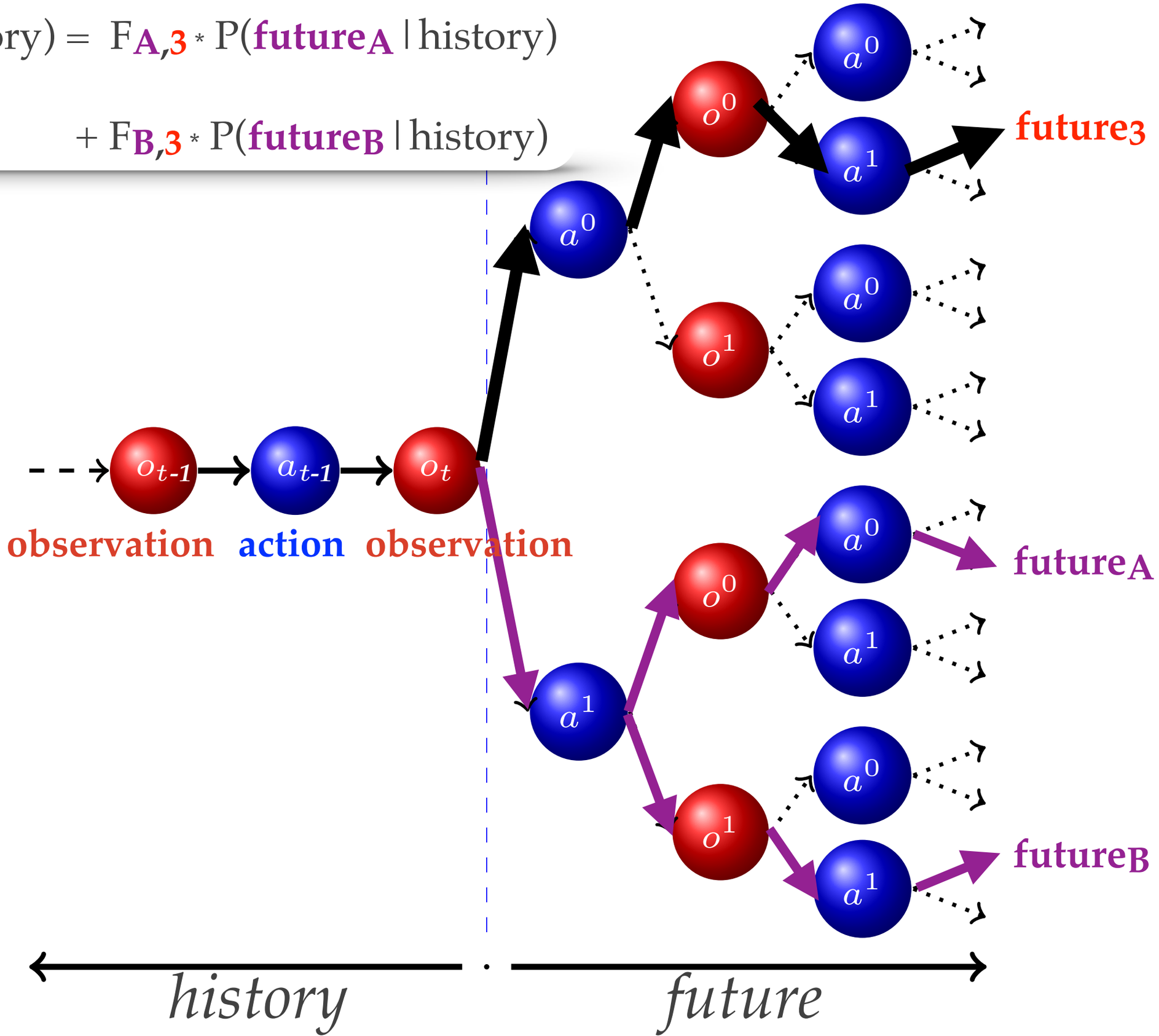
Predictive State Representations

$$P(\textbf{future}_2 \mid \text{history}) = F_{\textbf{A},2} * P(\textbf{future}_\textbf{A} \mid \text{history}) + F_{\textbf{B},2} * P(\textbf{future}_\textbf{B} \mid \text{history})$$



Predictive State Representations

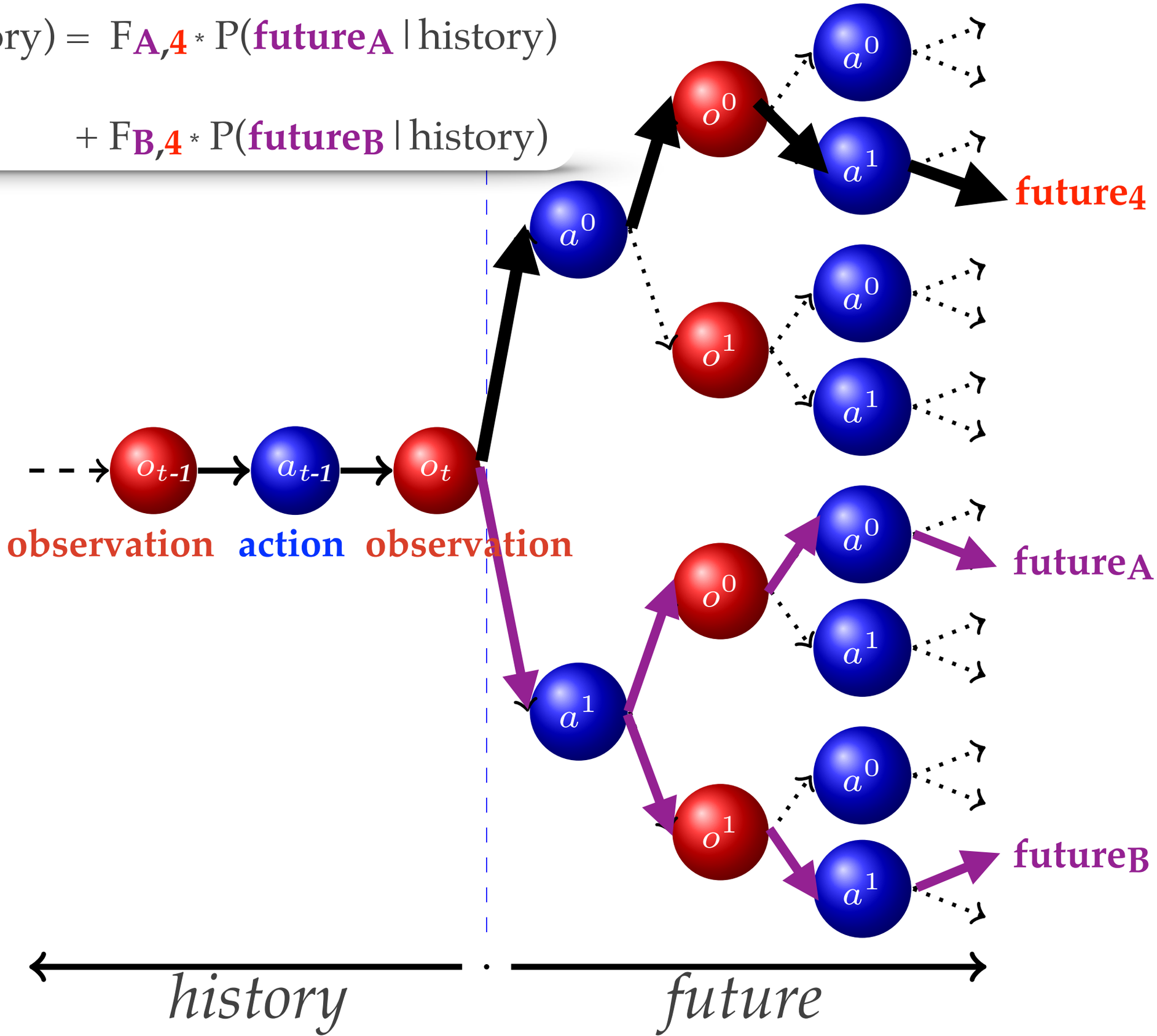
$$P(\textbf{future}_3 \mid \text{history}) = F_{\textbf{A},3} * P(\textbf{future}_\textbf{A} \mid \text{history}) + F_{\textbf{B},3} * P(\textbf{future}_\textbf{B} \mid \text{history})$$



Predictive State Representations

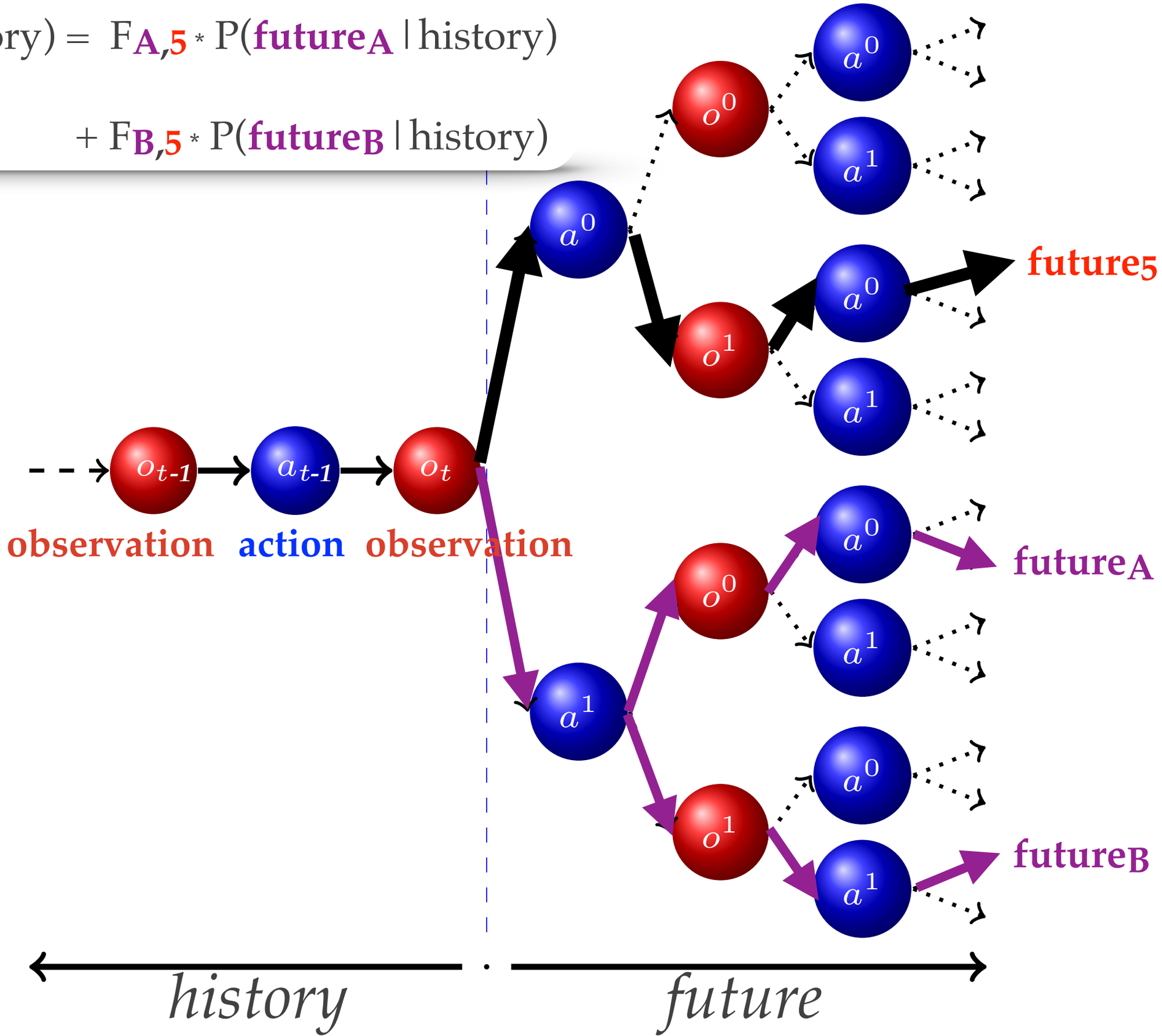
$$P(\textbf{future}_4 \mid \text{history}) = F_{\textbf{A},4} * P(\textbf{future}_\textbf{A} \mid \text{history})$$

$$+ F_{\textbf{B},4} * P(\textbf{future}_\textbf{B} \mid \text{history})$$



Predictive State Representations

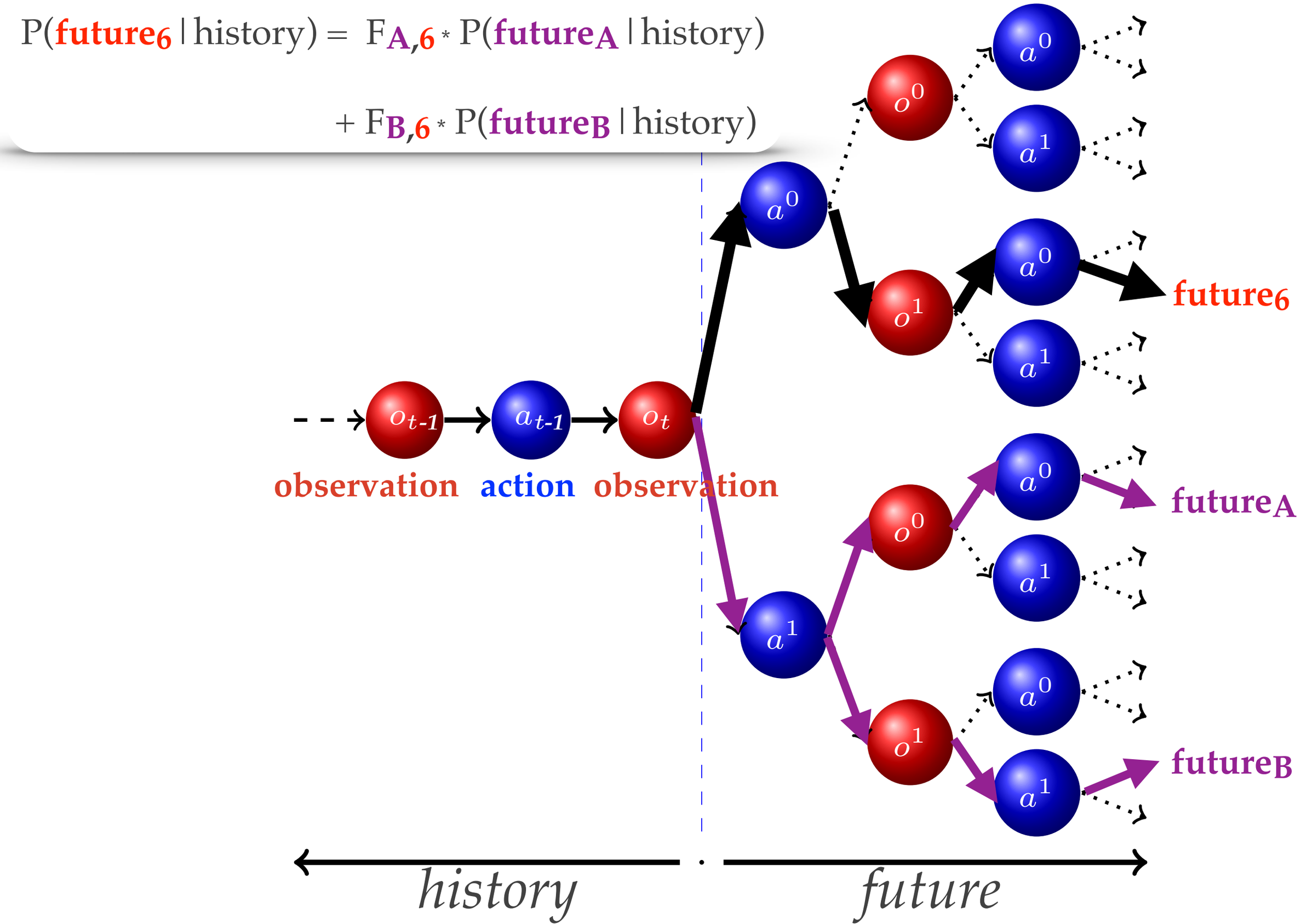
$$P(\textbf{future}_5 \mid \text{history}) = F_{\textbf{A},5} * P(\textbf{future}_\textbf{A} \mid \text{history}) + F_{\textbf{B},5} * P(\textbf{future}_\textbf{B} \mid \text{history})$$



Predictive State Representations

$$P(\textbf{future}_6 \mid \text{history}) = F_{\textbf{A},6} * P(\textbf{future}_\textbf{A} \mid \text{history})$$

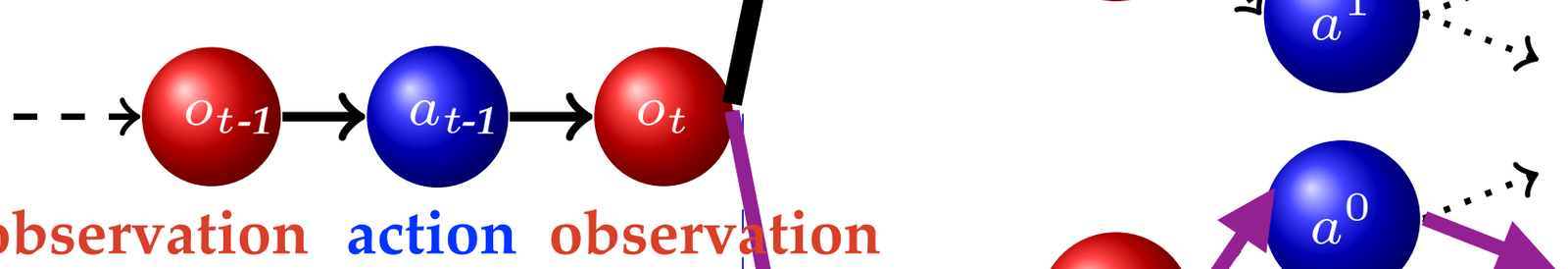
$$+ F_{\textbf{B},6} * P(\textbf{future}_\textbf{B} \mid \text{history})$$



Predictive State Representations

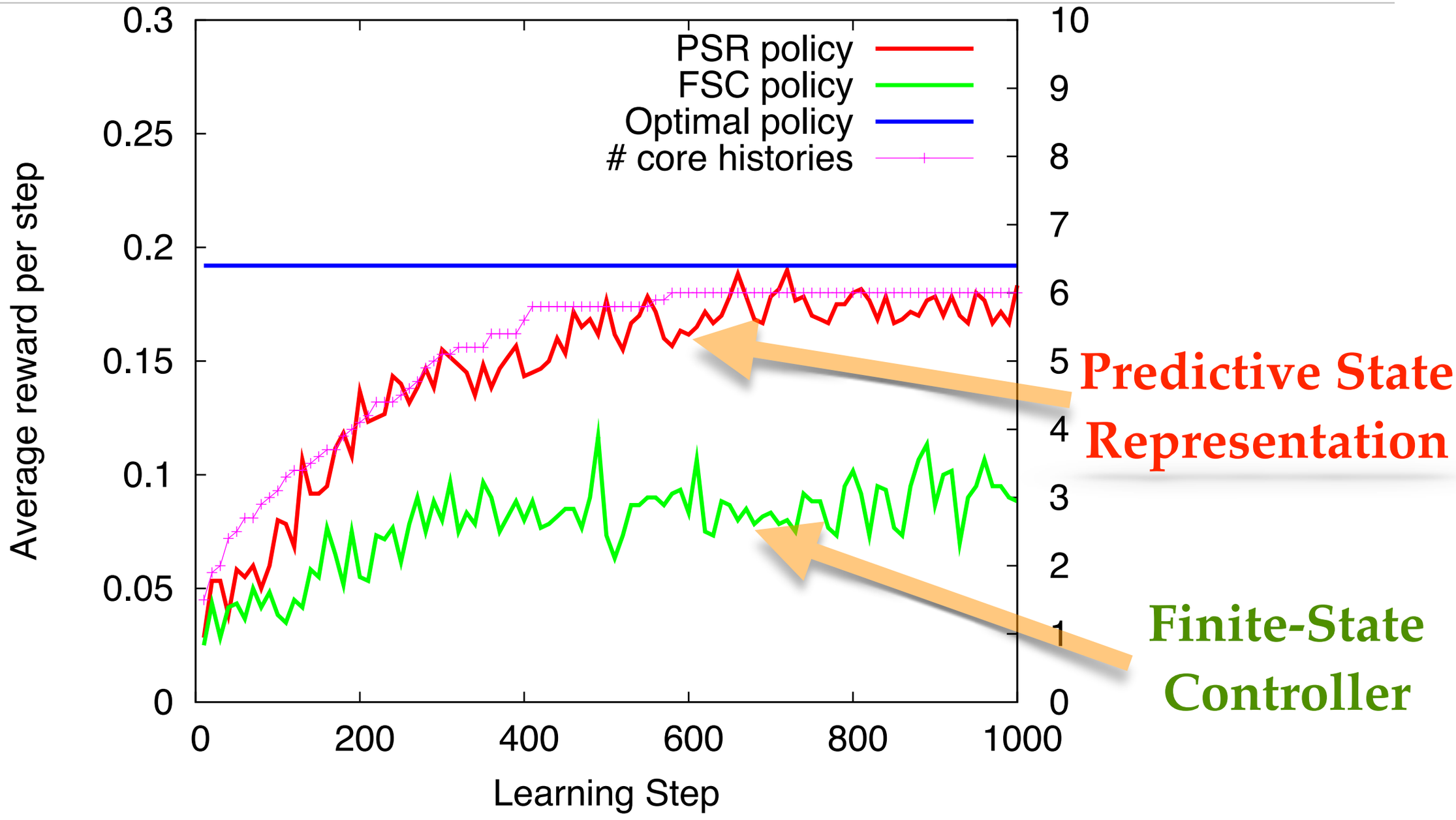
$$P(\textbf{future}_6 \mid \text{history}) = F_{\textbf{A},6} * P(\textbf{future}_\textbf{A} \mid \text{history}) + F_{\textbf{B},6} * P(\textbf{future}_\textbf{B} \mid \text{history})$$

Belief state



No hidden variables!

Reinforcement Learning with Predictive State Representations

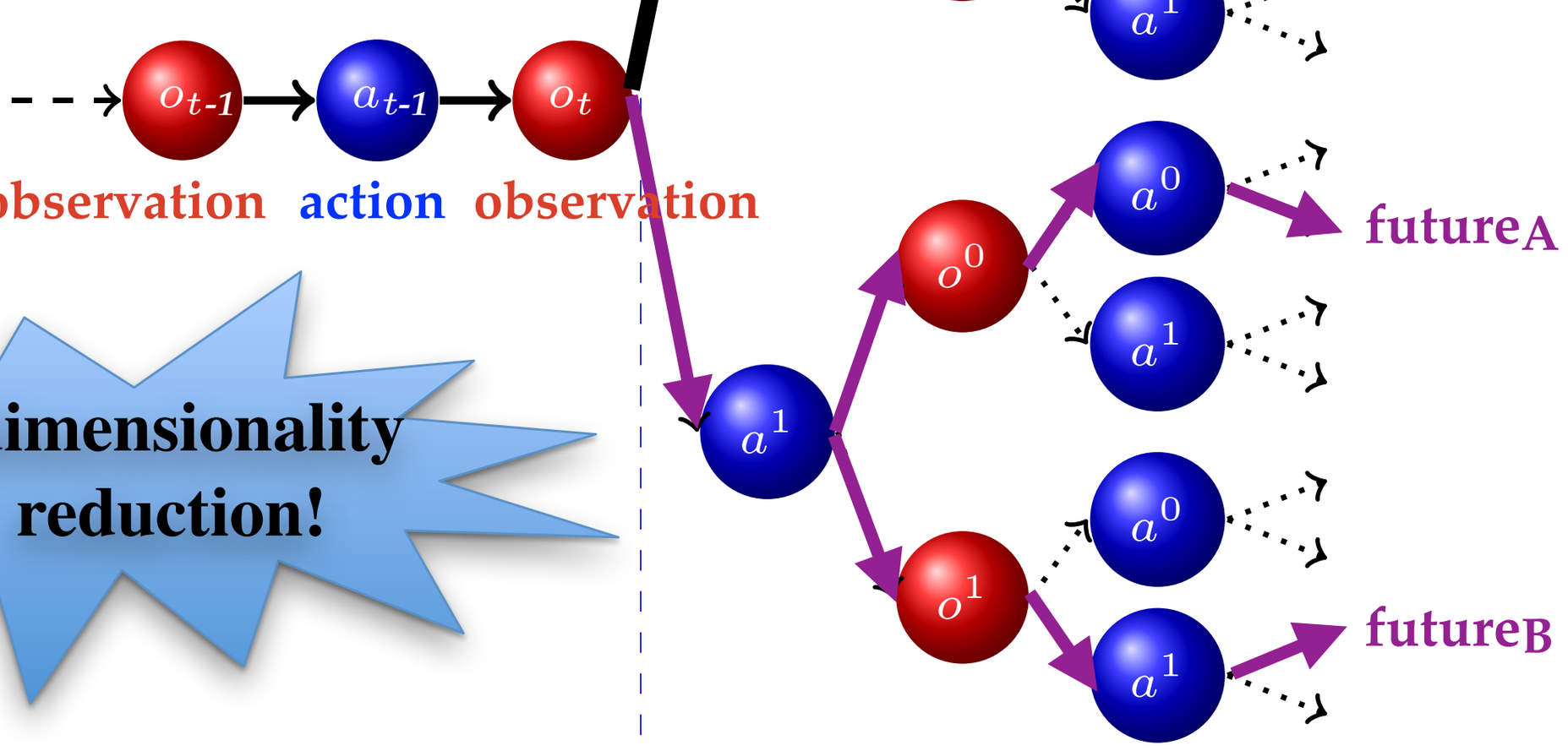


A. Boularias *et al.* (2009) in *International Conference on Machine Learning (ICML)*

Predictive State Representations

$$P(\textbf{future}_6 \mid \text{history}) = F_{A,6} * P(\textbf{future}_A \mid \text{history}) + F_{B,6} * P(\textbf{future}_B \mid \text{history})$$

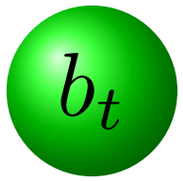
Belief state



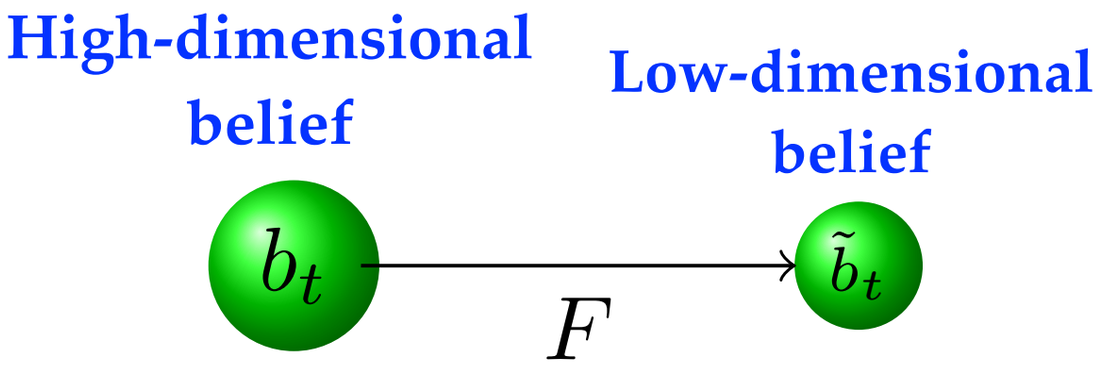
No dimensionality reduction!

Belief State Compression

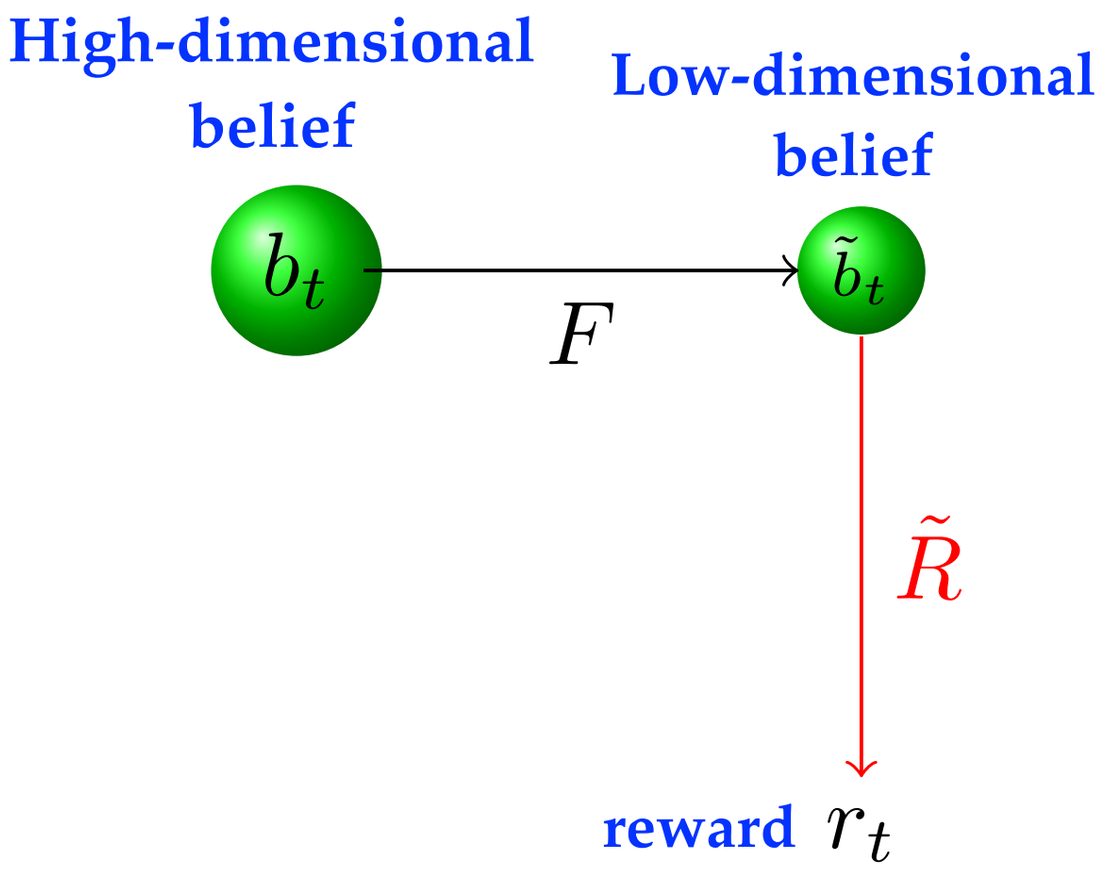
High-dimensional belief



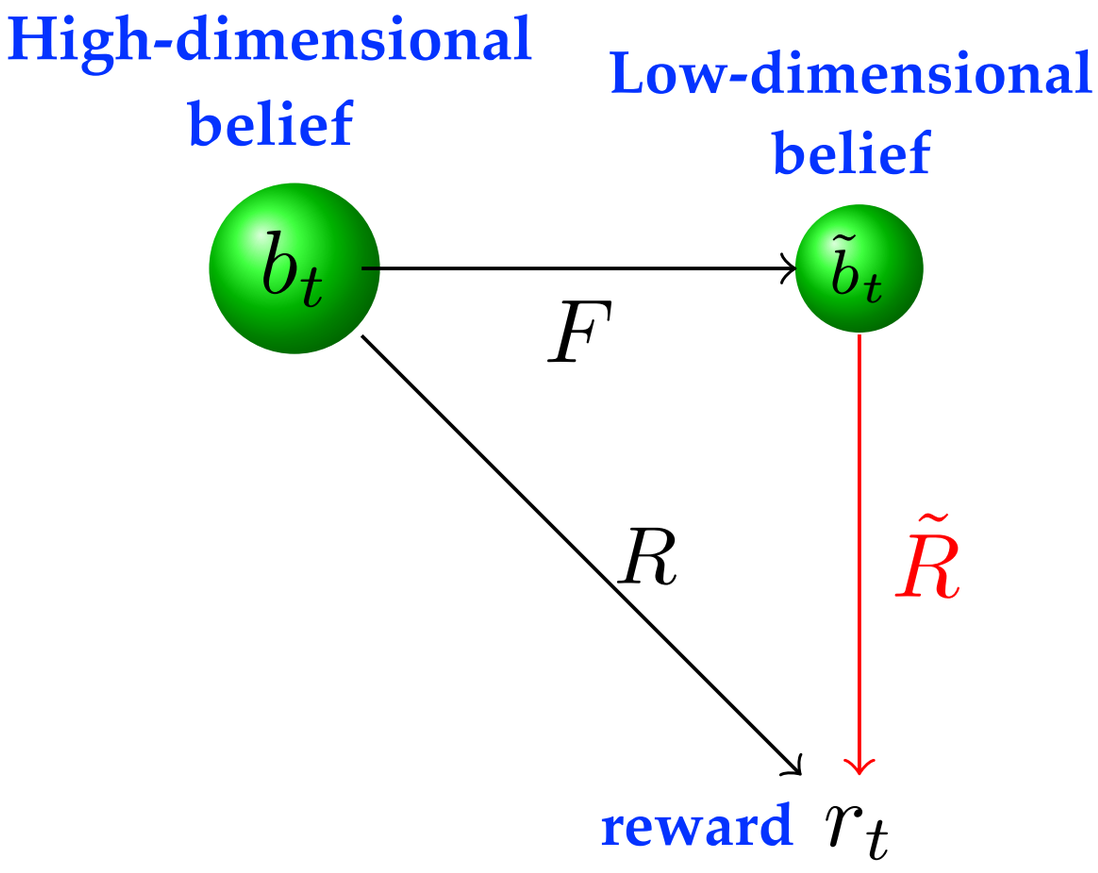
Belief State Compression



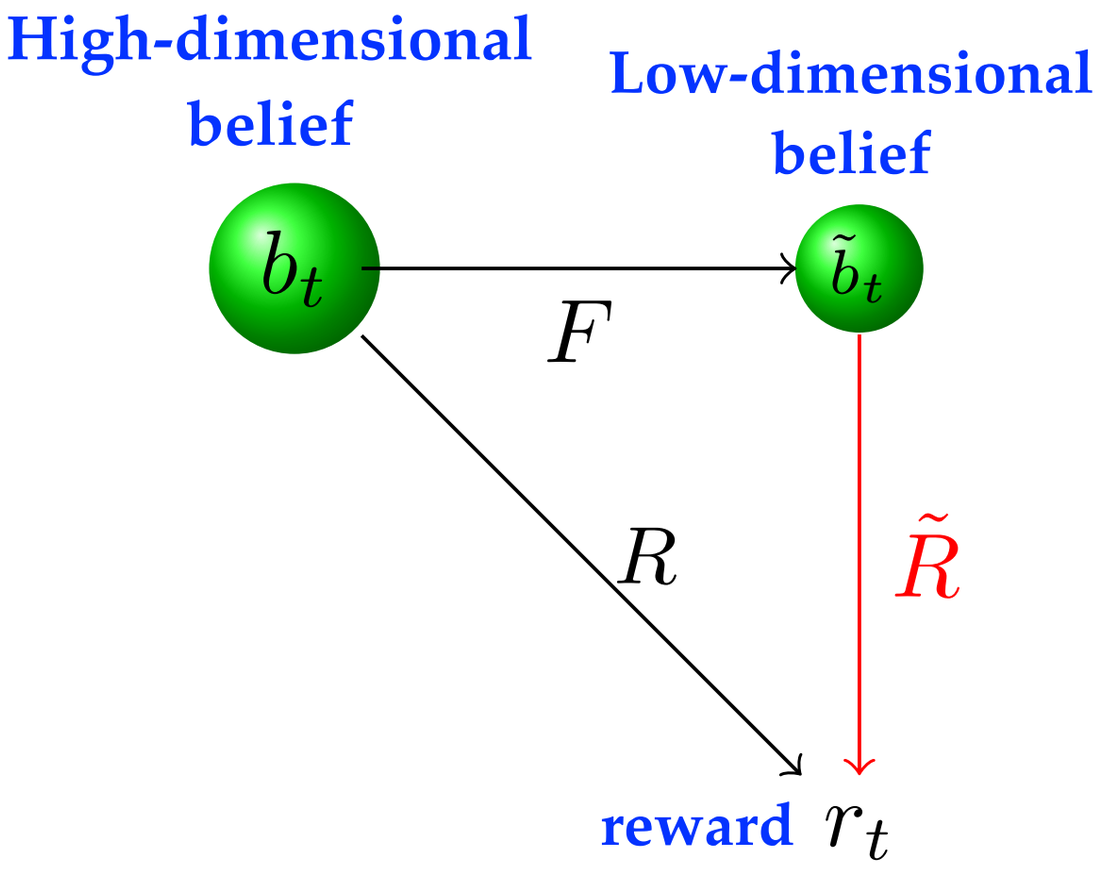
Belief State Compression



Belief State Compression

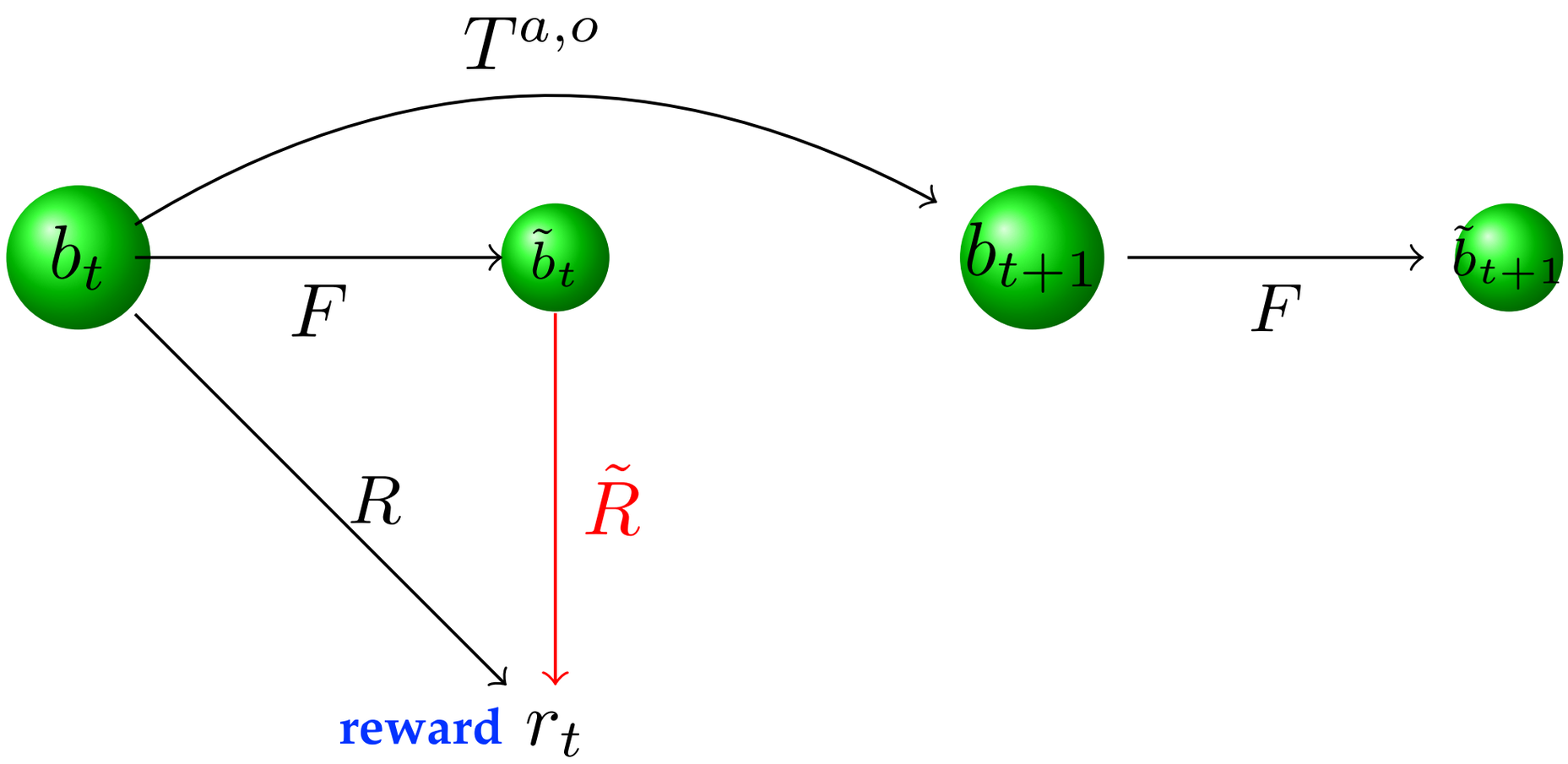


Belief State Compression



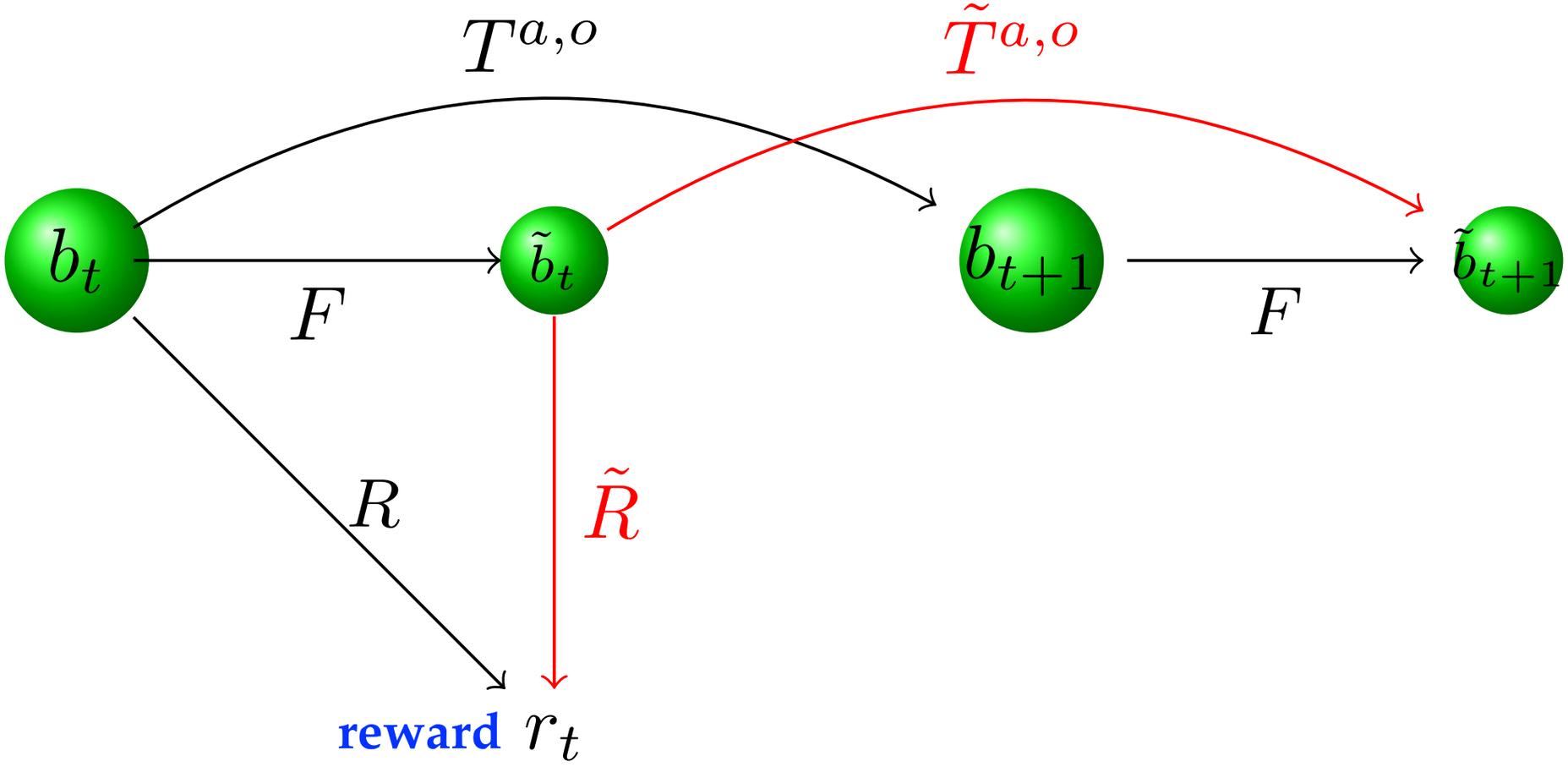
Minimize $\|F\tilde{R} - R\|_\infty$ for $\tilde{R}$

Belief State Compression



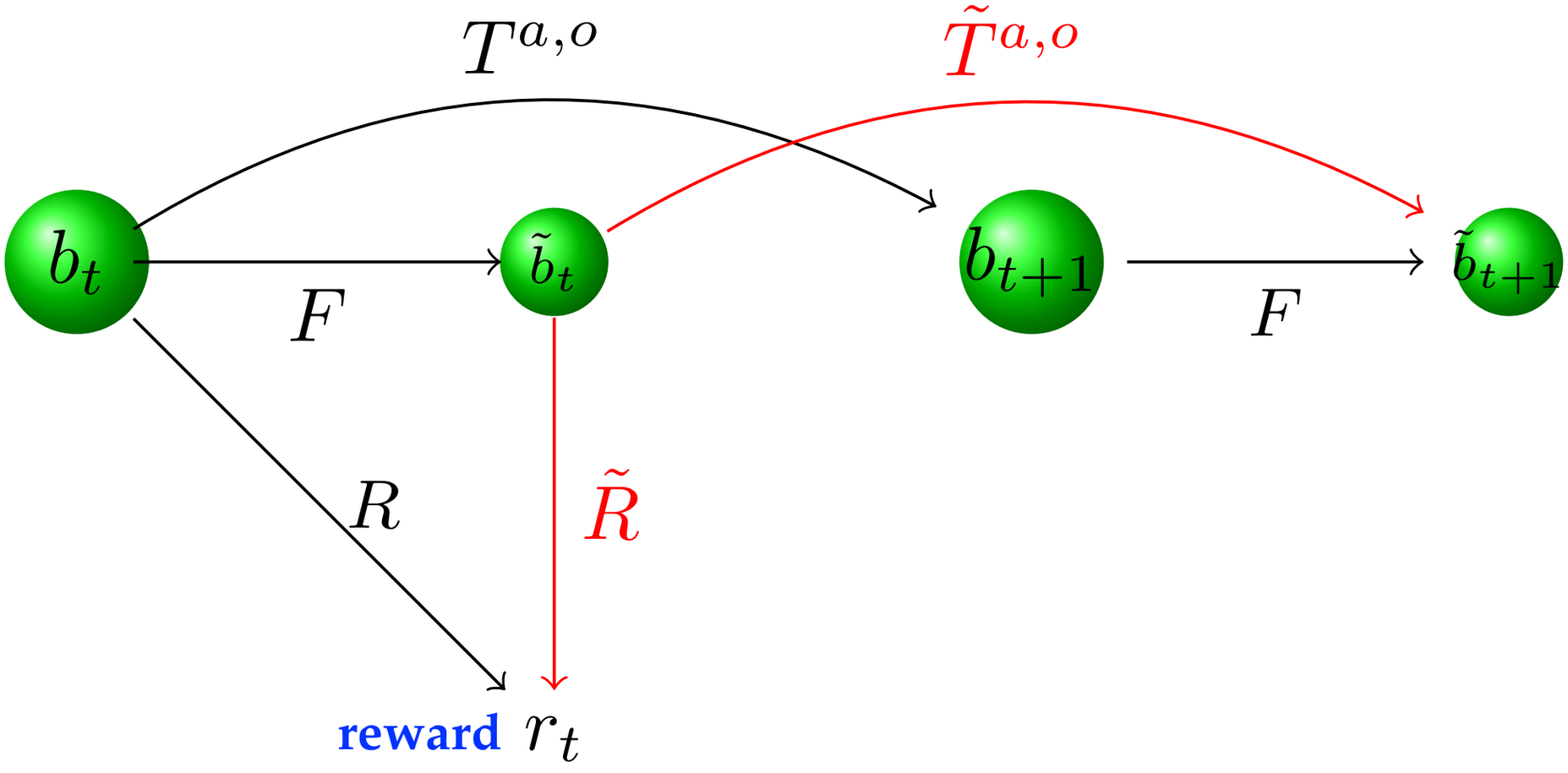
Minimize $\|F\tilde{R} - R\|_\infty$ for $\tilde{R}$

Belief State Compression



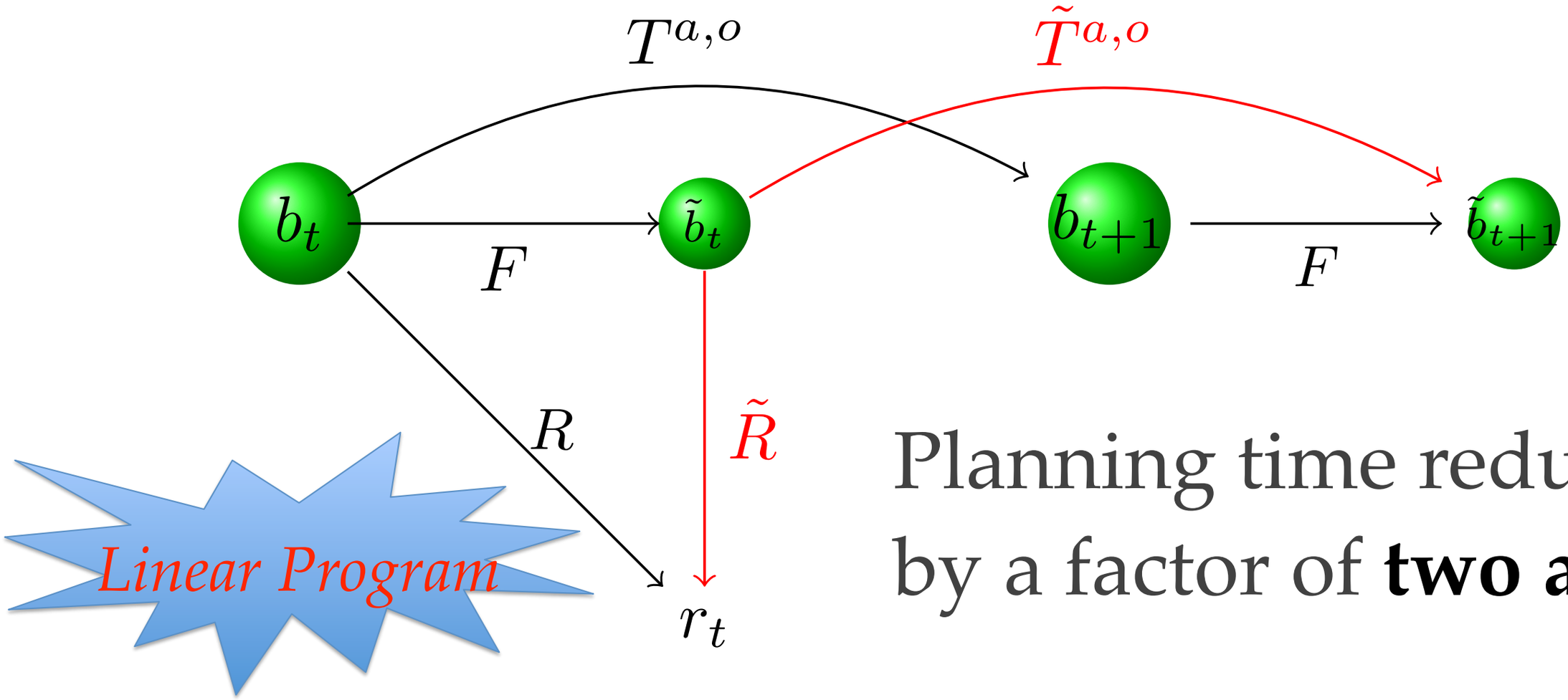
Minimize $\|F\tilde{R} - R\|_\infty$ for $\tilde{R}$

Belief State Compression



Minimize $\|F\tilde{R} - R\|_\infty$ for $\tilde{R}$ **and** $\|F\tilde{T}^{a,o} - T^{a,o}F\|_\infty$ for $\tilde{T}^{a,o}$

Belief State Compression



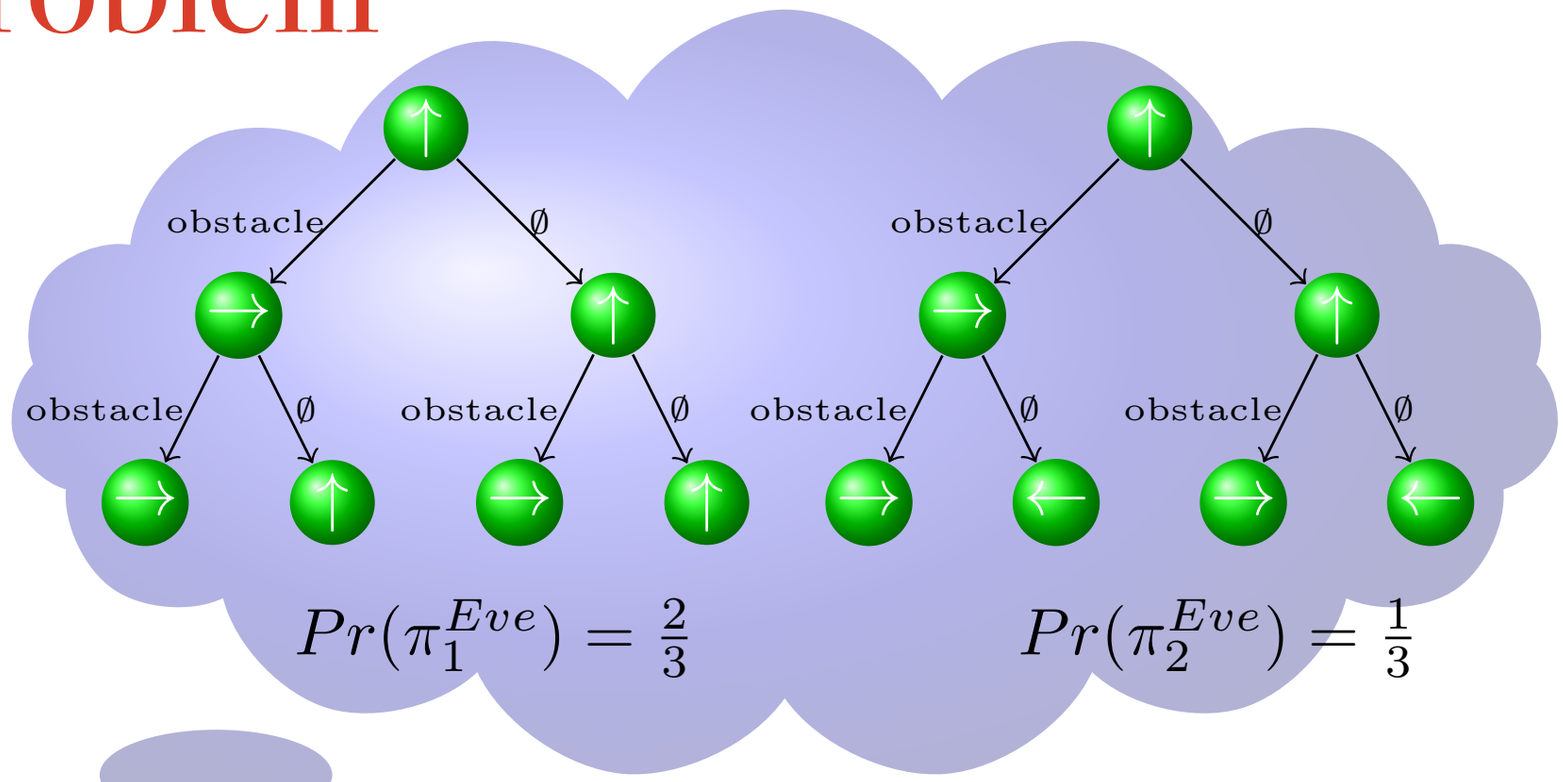
Planning time reduced by a factor of **two at least**

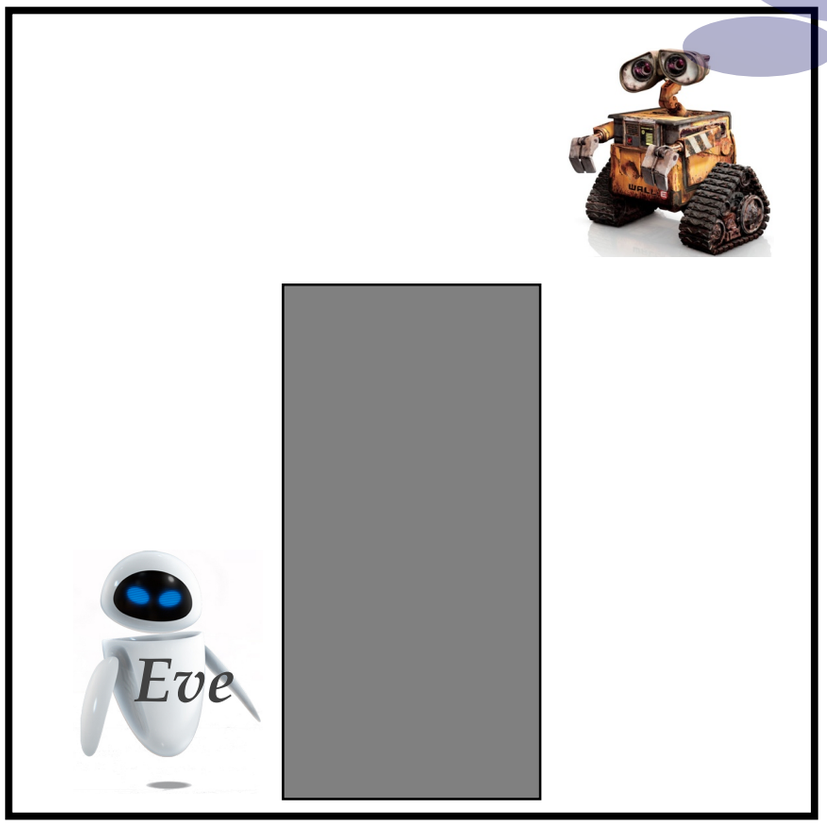
Minimize $\|F\tilde{R} - R\|_\infty$ for $\tilde{R}$ **and** $\|F\tilde{T}^{a,o} - T^{a,o}F\|_\infty$ for $\tilde{T}^{a,o}$

A. Boularias *et al.* (2008) in *International Conference on Machine Learning and Applications (ICMLA)*

Decentralized Partially Observable Markov Decision Process

The meeting problem

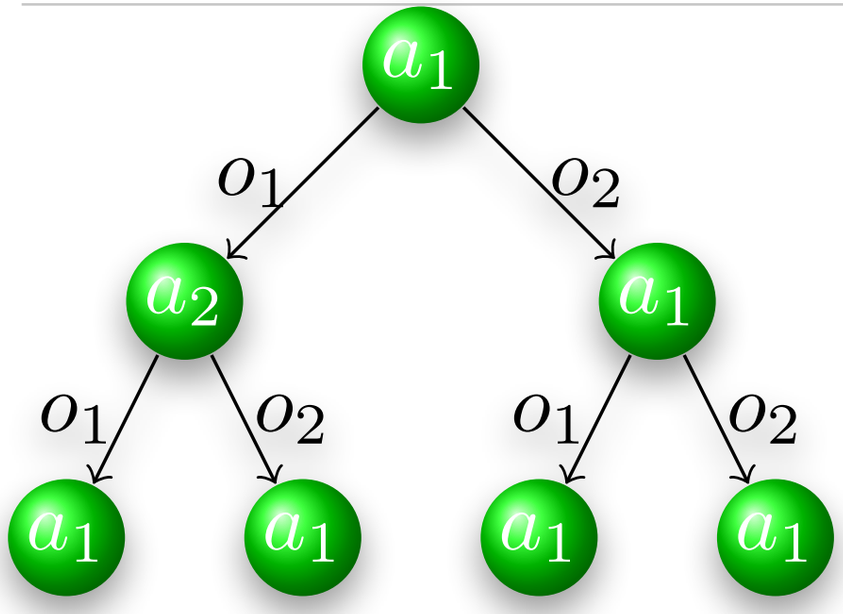




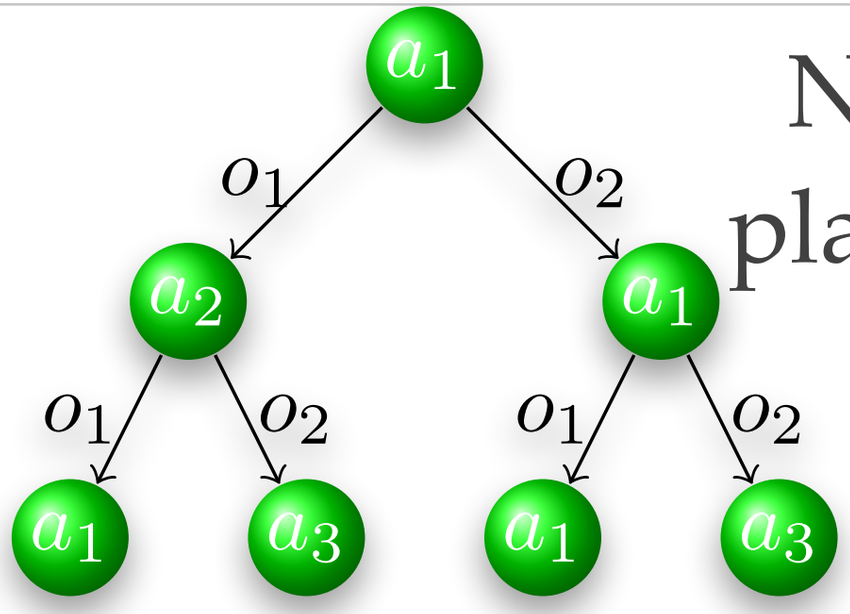
Probability distribution on the policies (strategies) of the other robot

PIXAR®

The number of policies is exponential in the planning horizon

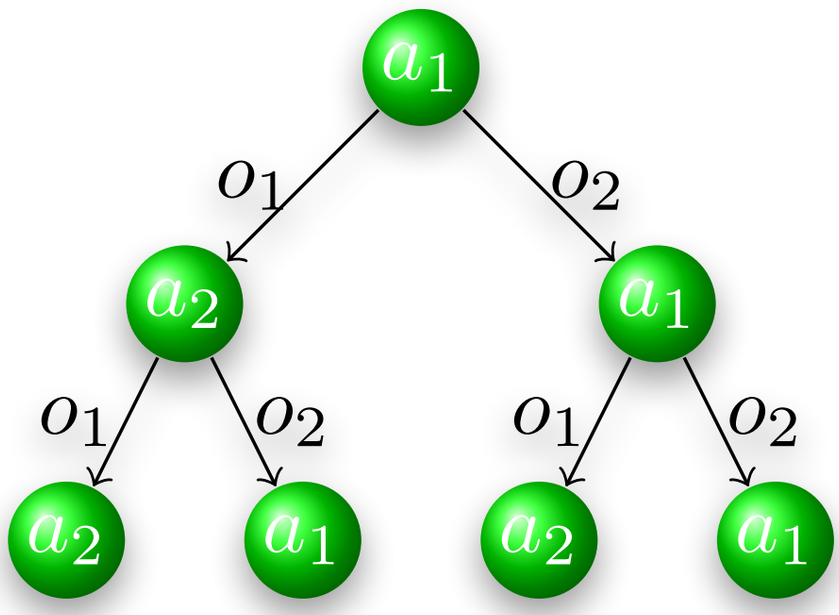


π_1

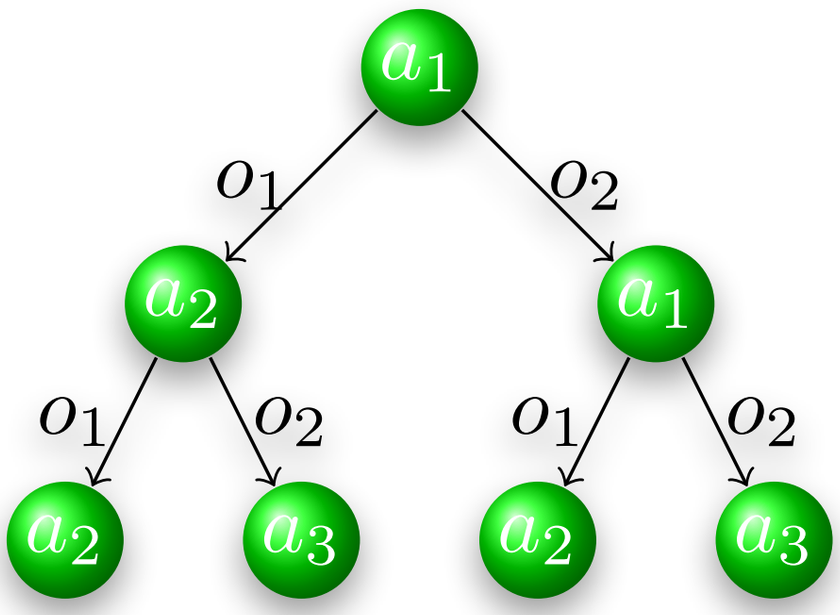


π_2

NEXP-complete planning problem!



π_3



π_4

Policy Space Compression

$$
U = \begin{array}{c}
\\
\pi_1 \\
\pi_2 \\
\pi_3 \\
\pi_4
\end{array}
\begin{array}{c}
\begin{array}{cccccccc}
a_1o_1a_2o_1a_1 & a_1o_1a_2o_1a_2 & a_1o_1a_2o_2a_1 & a_1o_1a_2o_2a_3 & a_1o_2a_1o_1a_1 & a_1o_2a_1o_1a_2 & a_1o_2a_1o_2a_1 & a_1o_2a_1o_2a_3
\end{array} \\
\begin{pmatrix}
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1
\end{pmatrix}
\end{array}
\quad \text{dimension} = 4
$$

Policy Space Compression

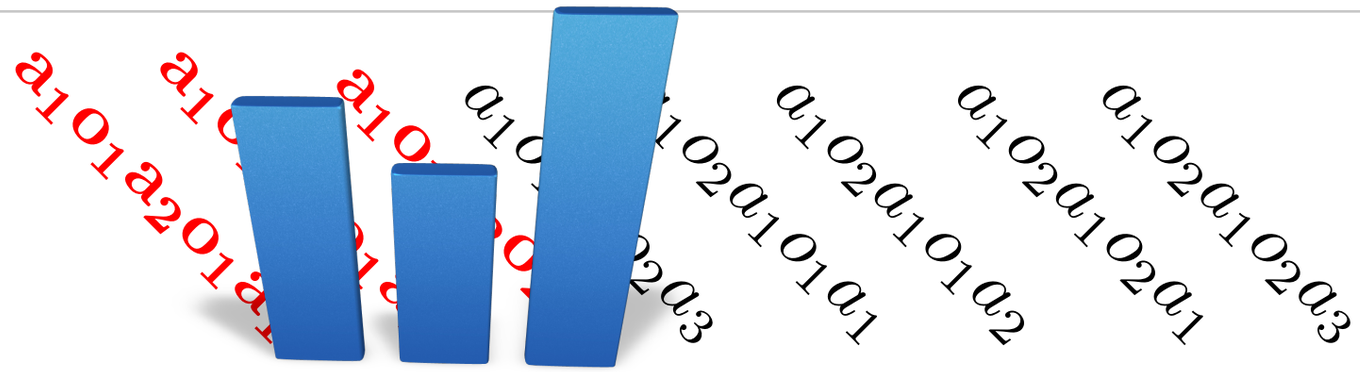
$$U = \begin{array}{c} \\ \pi_1 \\ \pi_2 \\ \pi_3 \\ \pi_4 \end{array} \begin{array}{c} \begin{array}{ccccccccc} \mathbf{a_1o_1a_2o_1a_1} & \mathbf{a_1o_1a_2o_1a_2} & \mathbf{a_1o_1a_2o_2a_1} & a_1o_1a_2o_2a_3 & a_1o_2a_1o_1a_1 & a_1o_2a_1o_1a_2 & a_1o_2a_1o_2a_1 & a_1o_2a_1o_2a_3 \end{array} \\ \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 & 1 & 0 & 1 & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & 1 & 1 & 0 & 0 & 1 \\ \mathbf{0} & \mathbf{1} & \mathbf{1} & 0 & 0 & 1 & 1 & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \end{array} \begin{array}{c} \\ \\ \\ \\ \text{dimension} = 4 \end{array}$$

Policy Space Compression

$$U = \begin{array}{c} \\ \pi_1 \\ \pi_2 \\ \pi_3 \\ \pi_4 \end{array} \begin{array}{c} \begin{array}{ccccccccc} \mathbf{a_1o_1a_2o_1a_1} & \mathbf{a_1o_1a_2o_1a_2} & \mathbf{a_1o_1a_2o_2a_1} & a_1o_1a_2o_2a_3 & a_1o_2a_1o_1a_1 & a_1o_2a_1o_1a_2 & a_1o_2a_1o_2a_1 & a_1o_2a_1o_2a_3 & \\ \end{array} \\ \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 & 1 & 0 & 1 & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & 1 & 1 & 0 & 0 & 1 \\ \mathbf{0} & \mathbf{1} & \mathbf{1} & 0 & 0 & 1 & 1 & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \end{array} \begin{array}{l} \\ \\ \\ \\ \text{dimension} = 4 \end{array}$$

$$= \underbrace{\begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} \\ \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{1} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \end{pmatrix}}_{F} \times \underbrace{\begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & -1 & 0 & 0 & 1 & -1 \end{pmatrix}}_{\text{constant weights}} \begin{array}{l} \\ \\ \text{dimension} = 3 \end{array}$$

Policy Space Compression



$$U = \begin{matrix} \pi_1 \\ \pi_2 \\ \pi_3 \\ \pi_4 \end{matrix} \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 & 1 & 0 & 1 & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & 1 & 1 & 0 & 0 & 1 \\ \mathbf{0} & \mathbf{1} & \mathbf{1} & 0 & 0 & 1 & 1 & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \quad \text{dimension} = 4$$

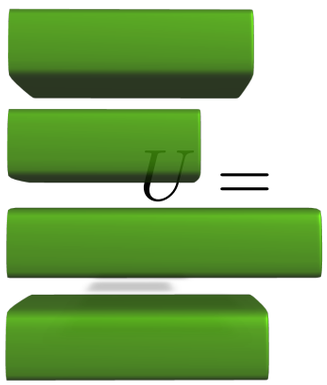
$$= \underbrace{\begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} \\ \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{1} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \end{pmatrix}}_{F} \times \underbrace{\begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & -1 & 0 & 0 & 1 & -1 \end{pmatrix}}_{\text{constant weights}} \quad \text{dimension} = 3$$

Policy Space Compression

$$U = \begin{array}{c} \\ \pi_1 \\ \pi_2 \\ \pi_3 \\ \pi_4 \end{array} \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 & 1 & 0 & 1 & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & 1 & 1 & 0 & 0 & 1 \\ \mathbf{0} & \mathbf{1} & \mathbf{1} & 0 & 0 & 1 & 1 & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \quad \text{dimension} = 4$$

$$= \underbrace{\begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} \\ \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{1} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \end{pmatrix}}_{F} \times \underbrace{\begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & -1 & 0 & 0 & 1 & -1 \end{pmatrix}}_{\text{constant weights}} \quad \text{dimension} = 3$$

Policy Space Compression



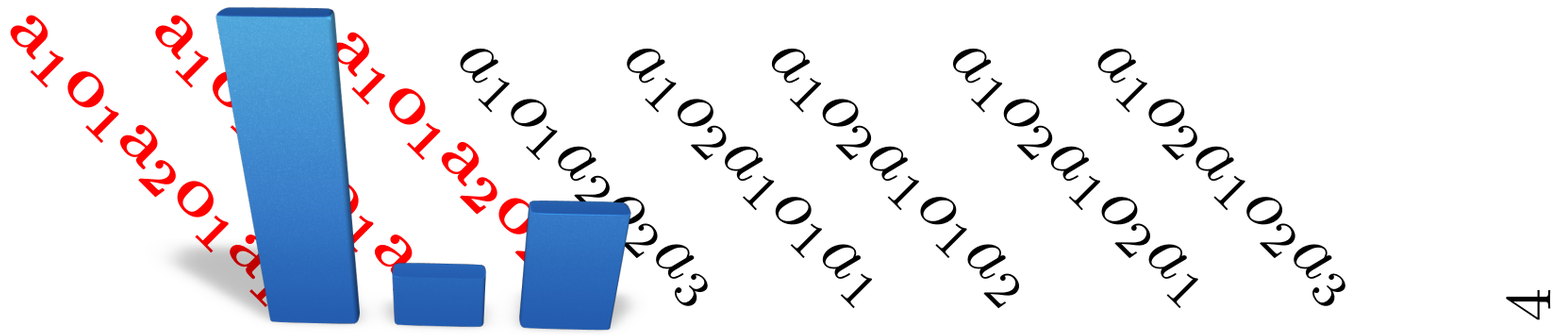
$$U = \begin{array}{c} \\ \pi_1 \\ \pi_2 \\ \pi_3 \\ \pi_4 \end{array} \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 & 1 & 0 & 1 & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & 1 & 1 & 0 & 0 & 1 \\ \mathbf{0} & \mathbf{1} & \mathbf{1} & 0 & 0 & 1 & 1 & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \quad \text{dimension} = 4$$

$$= \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} \\ \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{1} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & -1 & 0 & 0 & 1 & -1 \end{pmatrix} \quad \text{dimension} = 3$$

F

constant weights

Policy Space Compression



$$U = \begin{array}{c} \\ \pi_1 \\ \pi_2 \\ \pi_3 \\ \pi_4 \end{array} \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 & 1 & 0 & 1 & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & 1 & 1 & 0 & 0 & 1 \\ \mathbf{0} & \mathbf{1} & \mathbf{1} & 0 & 0 & 1 & 1 & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \quad \text{dimension} = 4$$

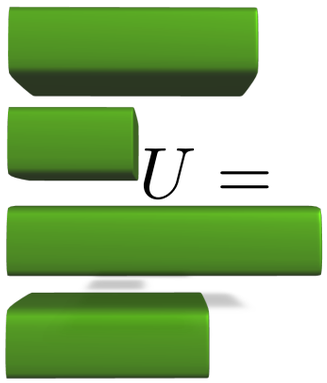
$$= \underbrace{\begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} \\ \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{1} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \end{pmatrix}}_{F} \times \underbrace{\begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & -1 & 0 & 0 & 1 & -1 \end{pmatrix}}_{\text{constant weights}} \quad \text{dimension} = 3$$

Policy Space Compression

$$U = \begin{array}{c} \\ \pi_1 \\ \pi_2 \\ \pi_3 \\ \pi_4 \end{array} \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 & 1 & 0 & 1 & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & 1 & 1 & 0 & 0 & 1 \\ \mathbf{0} & \mathbf{1} & \mathbf{1} & 0 & 0 & 1 & 1 & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \quad \text{dimension} = 4$$

$$= \underbrace{\begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} \\ \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{1} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \end{pmatrix}}_{F} \times \underbrace{\begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & -1 & 0 & 0 & 1 & -1 \end{pmatrix}}_{\text{constant weights}} \quad \text{dimension} = 3$$

Policy Space Compression

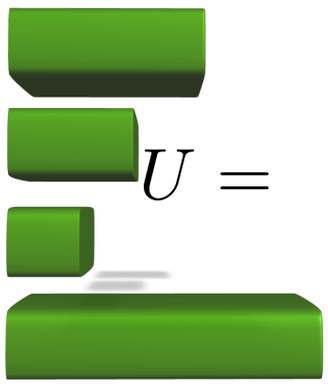


$$U = \begin{array}{c} \\ \pi_1 \\ \pi_2 \\ \pi_3 \\ \pi_4 \end{array} \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 & 1 & 0 & 1 & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & 1 & 1 & 0 & 0 & 1 \\ \mathbf{0} & \mathbf{1} & \mathbf{1} & 0 & 0 & 1 & 1 & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \quad \text{dimension} = 4$$

$$= \underbrace{\begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} \\ \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{1} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \end{pmatrix}}_{F} \times \underbrace{\begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & -1 & 0 & 0 & 1 & -1 \end{pmatrix}}_{\text{constant weights}} \quad \text{dimension} = 3$$

Policy Space Compression

A distribution on sequences (columns) is more compact than a distribution on policies (rows).



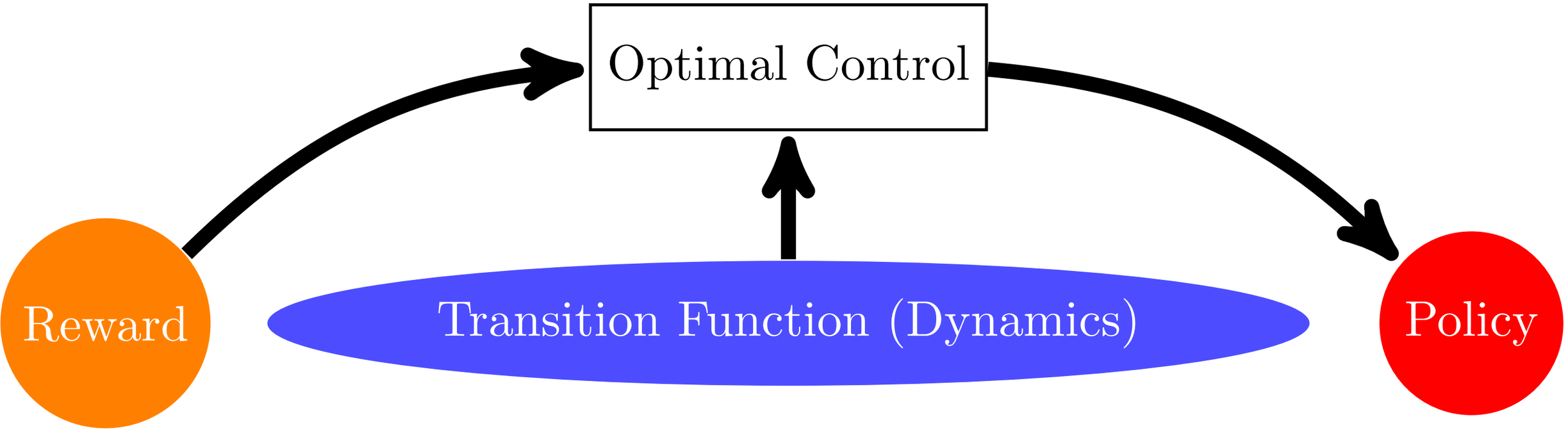
$$U = \begin{array}{c} \\ \pi_1 \\ \pi_2 \\ \pi_3 \\ \pi_4 \end{array} \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 & 1 & 0 & 1 & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & 1 & 1 & 0 & 0 & 1 \\ \mathbf{0} & \mathbf{1} & \mathbf{1} & 0 & 0 & 1 & 1 & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \begin{array}{l} \\ \\ \\ \\ \text{dimension} = 4 \end{array}$$

$$= \underbrace{\begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} \\ \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{1} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \end{pmatrix}}_{F} \times \underbrace{\begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & -1 & 0 & 0 & 1 & -1 \end{pmatrix}}_{\text{constant weights}} \begin{array}{l} \\ \\ \\ \text{dimension} = 3 \end{array}$$

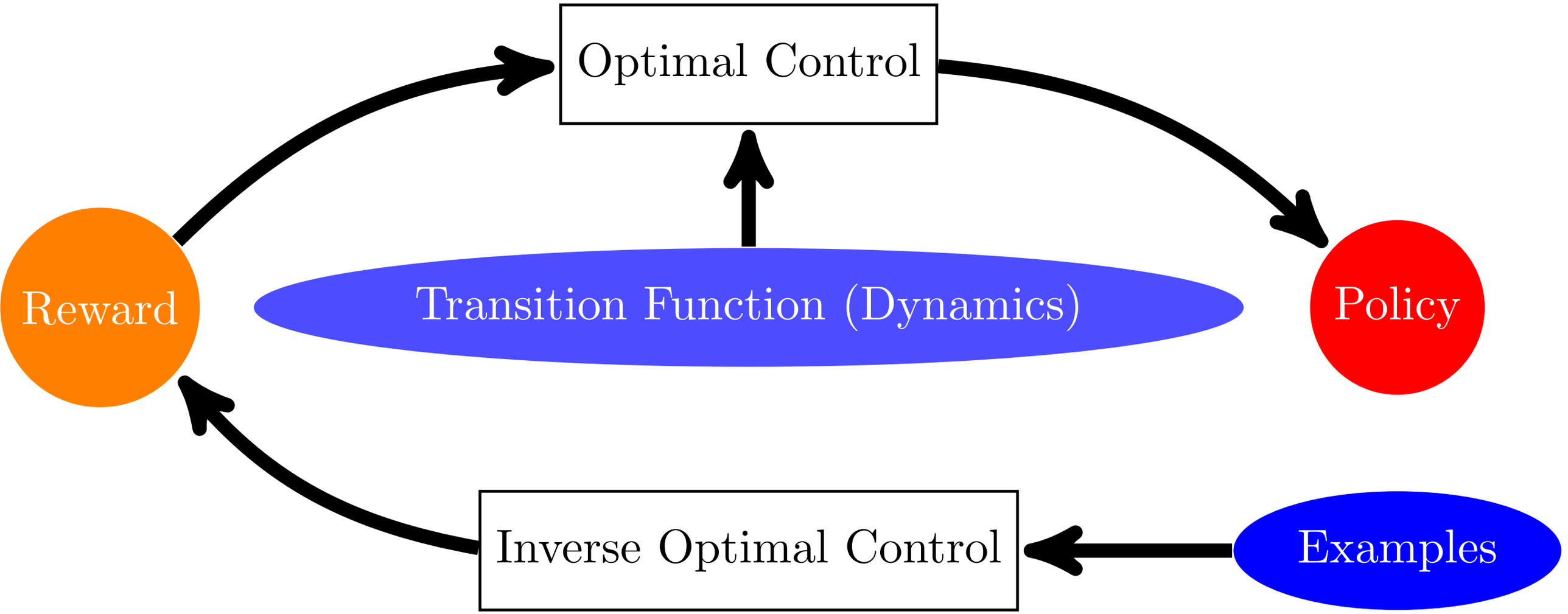
Policy Space Compression

		Decision Trees		Predictive Representations	
Problem	Horizon	planning time	number of trees	planning time	number of sequences
Multiagent Tiger	2	0.20	(27,27)	**0.17**	(18,18)
	3	2.29	(675,675)	**1.79**	(90,90)
	4	-	-	**534.90**	(540,540)
Multiagent Broadcasting Channel	2	0.12	(8,8)	**0.14**	(8,8)
	3	0.46	(72,72)	**0.36**	(24,24)
	4	17.59	(1800,1458)	**4.59**	(80,80)

A. Boularias *et al.* (2008) in *International Conference on Automated Planning and Scheduling (ICAPS)*



Designing a useful reward function for complex behaviors is a tedious task.



Designing a useful reward function for complex behaviors is a tedious task.

Outline

1. Overview

2. Optimal control

3. Inverse optimal control

4. Grasping

5. Manipulation

6. Navigation

Inverse Optimal Control

<u>Assumption:</u> The reward is a linear function of state-action features

$$R \stackrel{def}{=} w^T \phi$$

reward — *unknown weights* — *features (e.g. velocity, energy, distance from goal, ..)*

Inverse Optimal Control

<u>Assumption:</u> The reward is a linear function of state-action features

$$R \stackrel{def}{=} w^T \phi$$

Value of policy **π**

$$V(\pi) = \sum_{t=0}^{H} \mathbb{E}_{s_t}[R(s_t, \pi(s_t))]$$

Inverse Optimal Control

<u>Assumption:</u> The reward is a linear function of state-action features

$$R \stackrel{def}{=} w^T \phi$$

Value of policy **π**

$$V(\pi) = \sum_{t=0}^{H} \mathbb{E}_{s_t}[R(s_t, \pi(s_t))]$$

$$= \sum_{k=1}^{n} w_k \underbrace{\sum_{t=0}^{H} \mathbb{E}_{s_t}[\phi_k(s_t, a_t)]}_{\phi(\pi)} = w^T \phi(\pi)$$

expected features under π

(e.g. expected energy, distance, etc.)

Inverse Optimal Control: Problem Statement

Given an expert's policy π^*, find reward weights $\mathbf{w}$ such that:

$$w^T \phi(\pi^*) \geq \max_{\pi} w^T \phi(\pi)$$

Value of the expert's policy *Value of an arbitrary policy*

In other terms, expert's policy π^* has the highest possible value.

Inverse problems are generally ill-posed

Given an expert's policy π^*, find reward weights w such that:

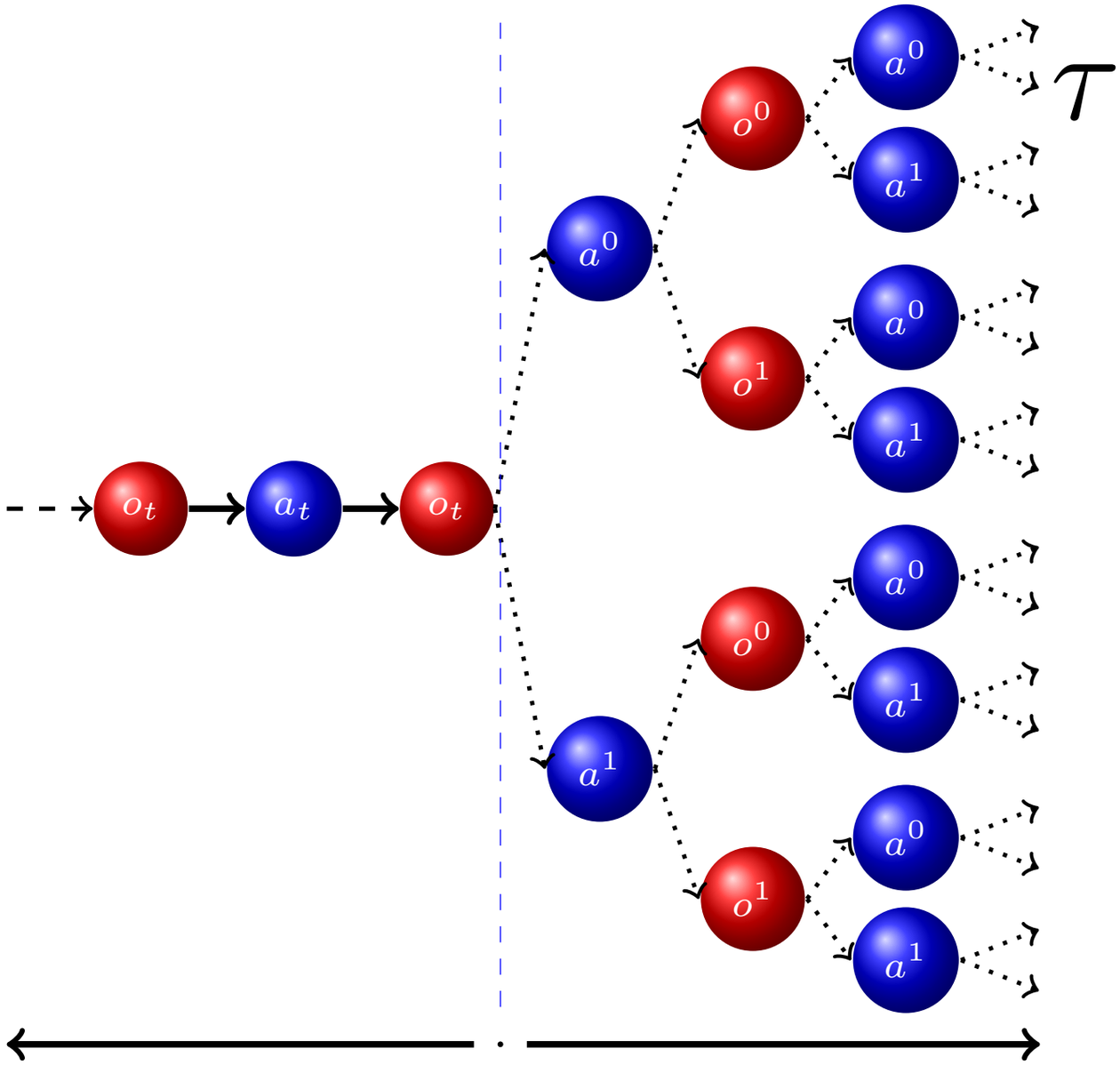
$$w^T \phi(\pi^*) \geq \max_{\pi} w^T \phi(\pi)$$

Value of the expert's policy

Value of an arbitrary policy

Relative Entropy Inverse Optimal Control

Find P, a probability distribution on state-action paths τ



Relative Entropy Inverse Optimal Control

Find P, a probability distribution on state-action paths τ

$$\int_\tau P(\tau) d\tau = 1 \quad \forall \tau : P(\tau) \geq 0$$

$$\| \mathbb{E}_{\tau \sim P}[\phi(\tau)] - \phi(\pi^*) \| \leq \epsilon$$

Expected features under P

Expected features in the expert's demonstration

Relative Entropy Inverse Optimal Control

Find P, a probability distribution on state-action paths τ

Solve $\min_P D_{KL}(P\|Q)$

Reference distribution

Subject to: $\int_\tau P(\tau)d\tau = 1 \quad \forall\tau : P(\tau) \geq 0$

convex

$$\|\mathbb{E}_{\tau\sim P}[\phi(\tau)] - \phi(\pi^*)\| \leq \epsilon$$

Expected features under P

Expected features in the expert's demonstration

Relative Entropy Inverse Optimal Control

Solution

Reference distribution

$$P(\tau|w) \propto Q(\tau) \exp\left(w^T \phi(\tau)\right)$$

Expected return

Reward weights **w** are obtained by gradient descent

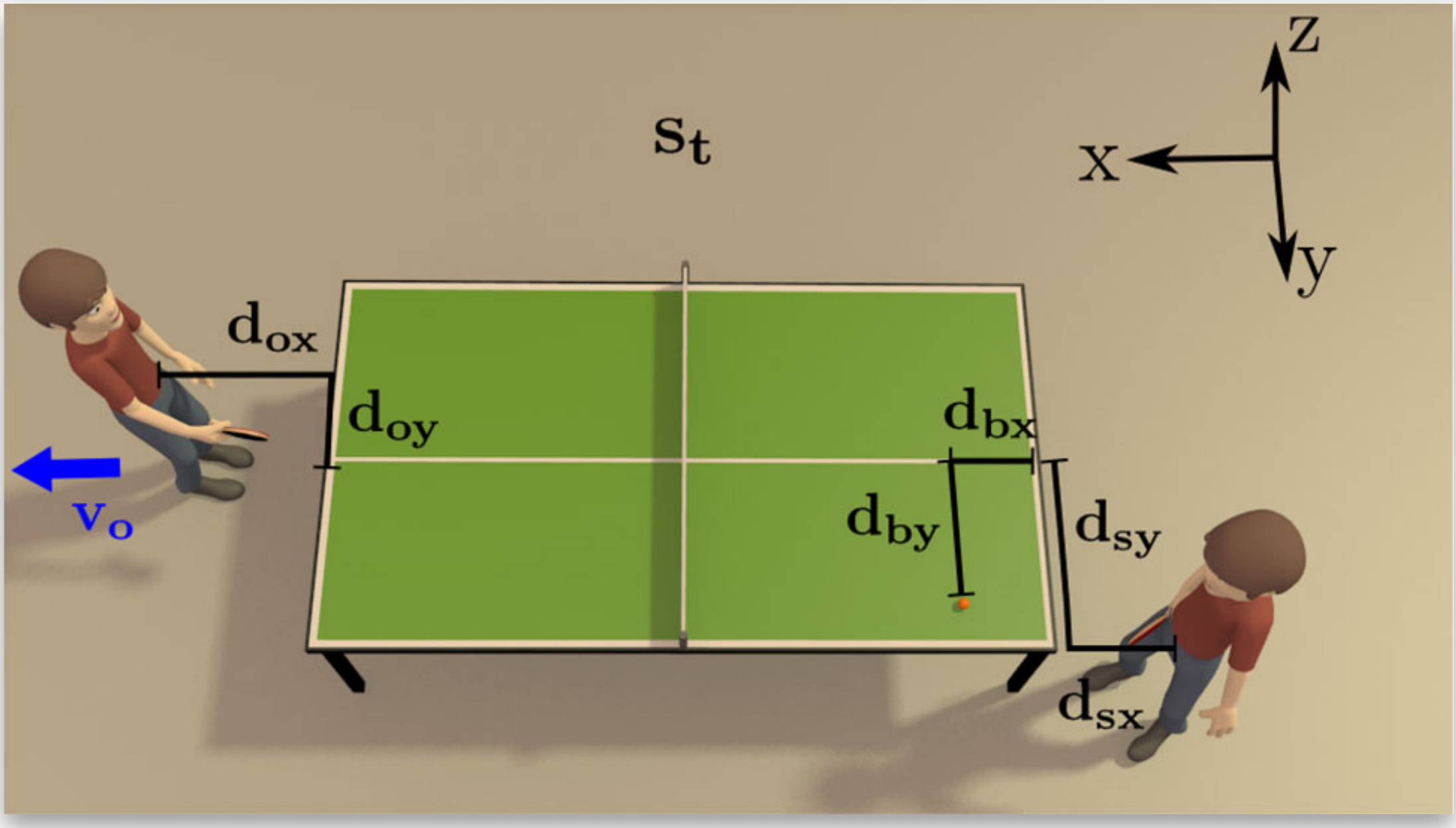
Robot Table Tennis: Learning to Imitate an Expert Player

Goal: Learn a reward function from demonstrations of a professional player

Trajectories of the ball and the bodies of the players were captured using infrared markers.

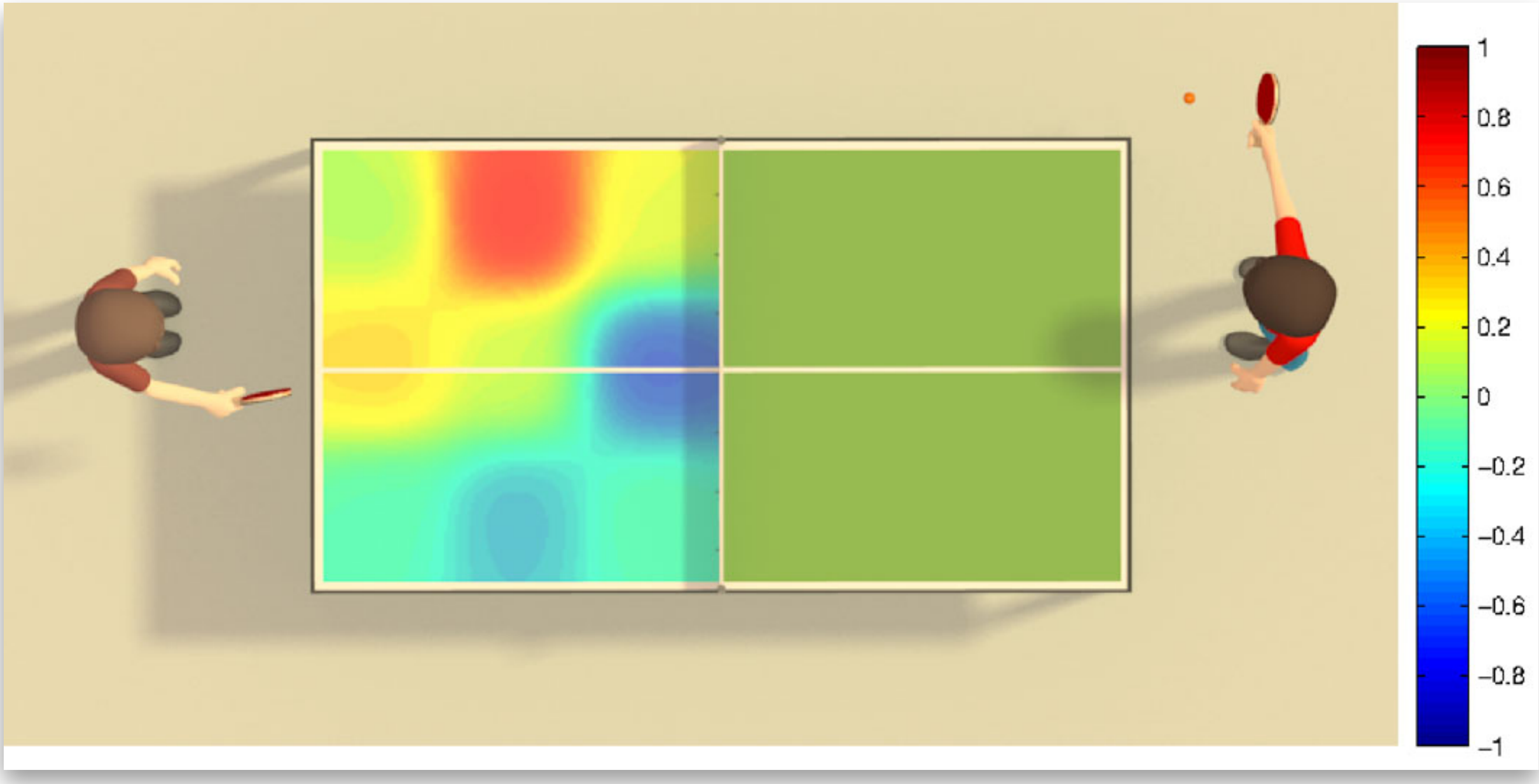
Robot Table Tennis: Learning to Imitate an Expert Player

State $\mathbf{s_t}$: position of the ball + positions of the players at time $\mathbf{t}$



Learned reward function

Red indicates good location for bouncing the ball.



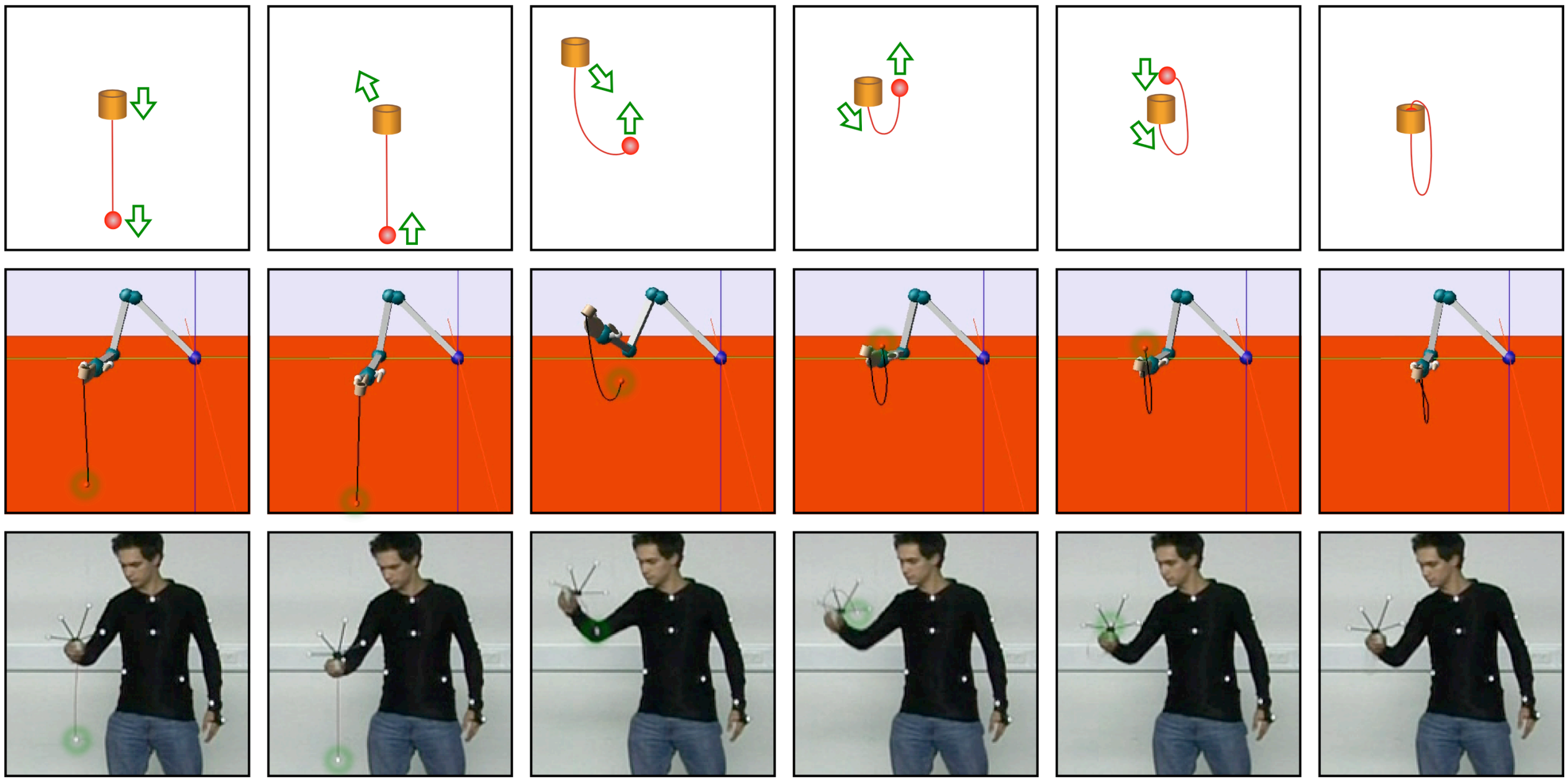
Robot Table Tennis: Learning to Imitate an Expert Player

The learned reward model was used to predict the skill levels of different players (without looking at the scores).

The most skilled players obtained the highest rewards.

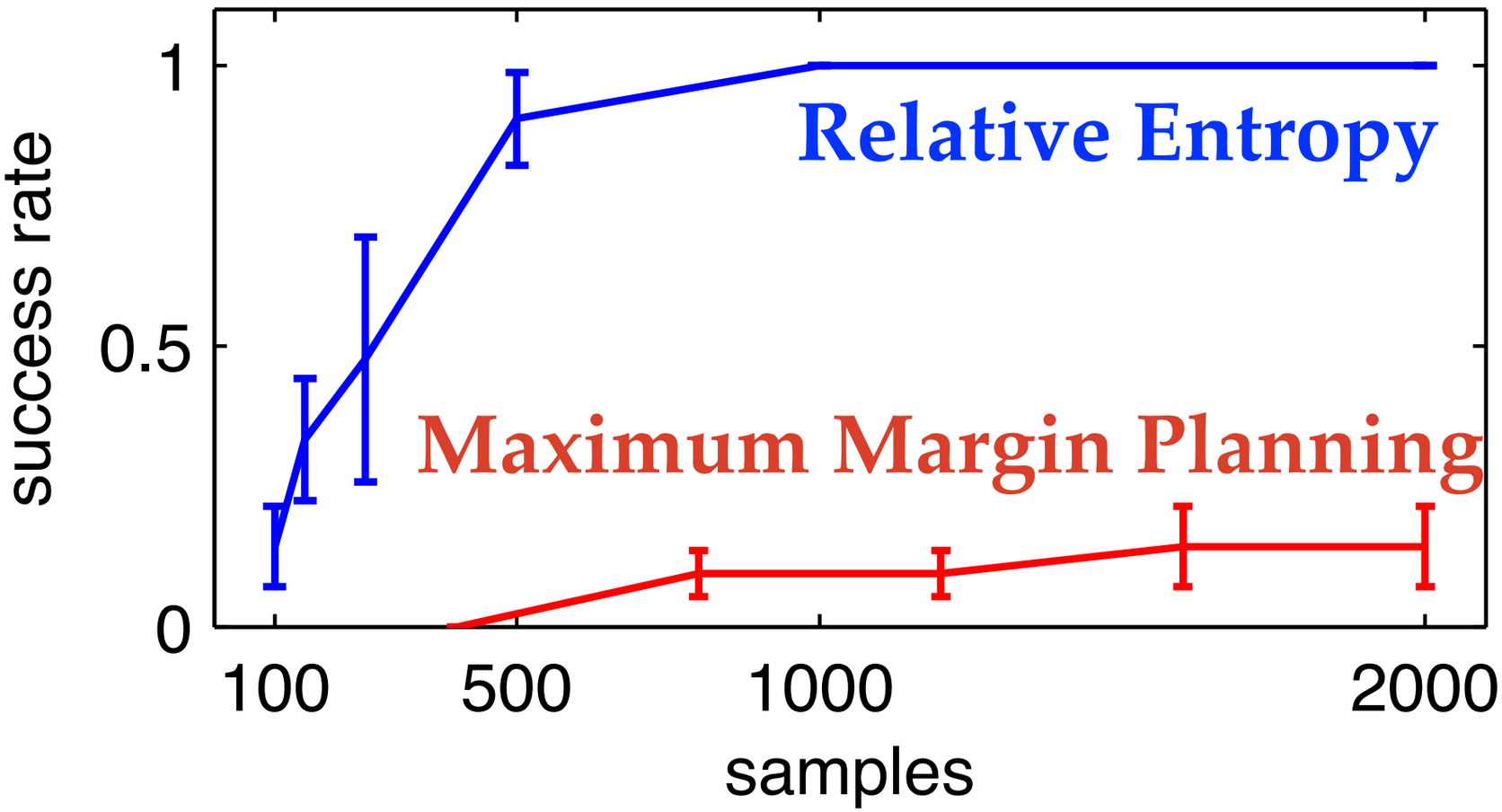
K. Muelling, A. Boularias *et al.* (2014) in *Biological Cybernetics 108(5): 603-619.*

Example: Ball-in-a-Cup game (*Kendama*)



A human expert (Jens Kober) providing a demonstration

Example: Ball-in-a-Cup game (*Kendama*)



A. Boularias *et al.* (2011) in *Artificial Intelligence and Statistics (AISTATS).*

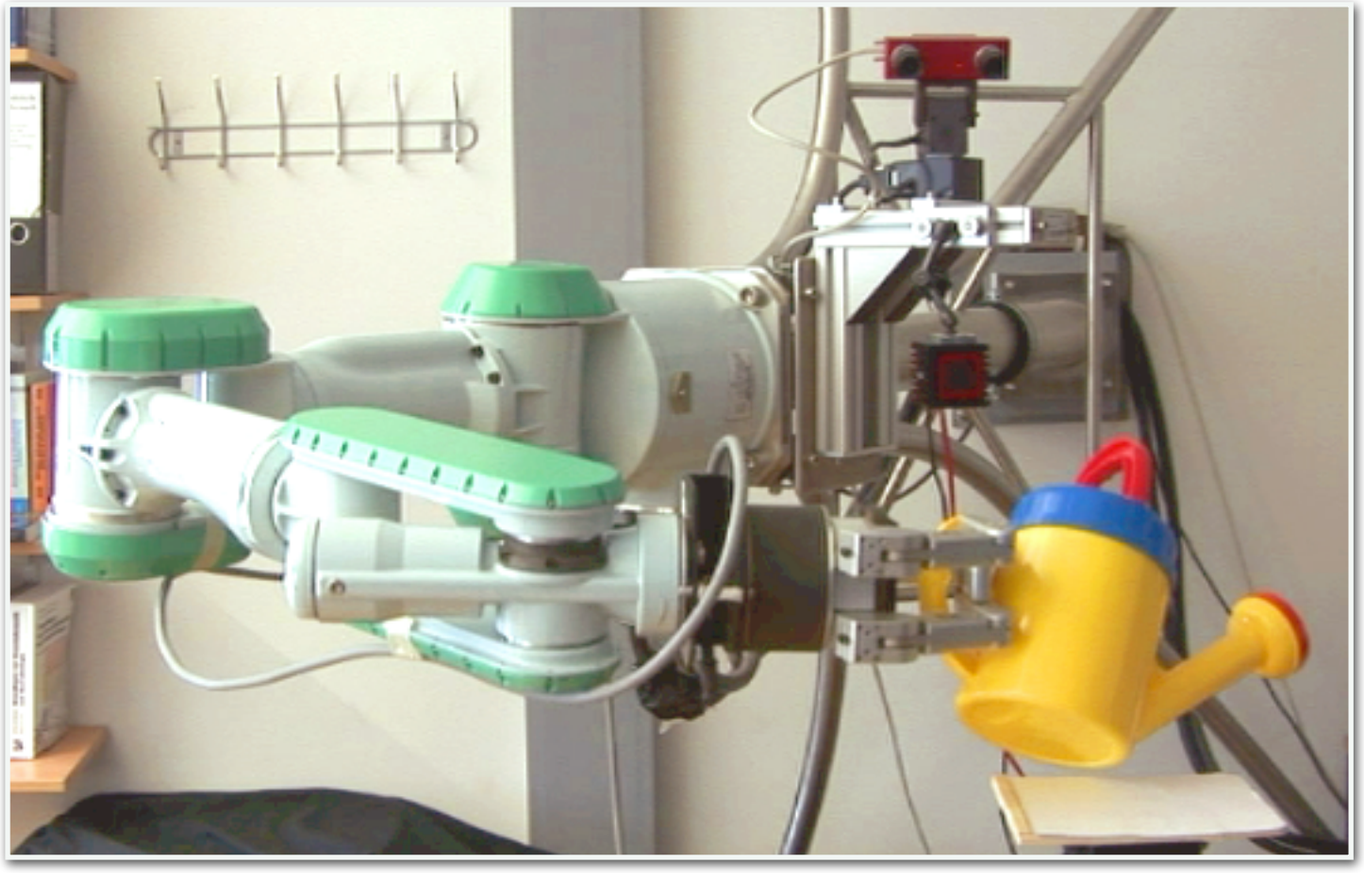
Outline

1. Overview
2. Optimal control
3. Inverse optimal control
4. **Grasping**
5. Manipulation
6. Navigation

Purposeful Grasping

Parameters of a grasping action (position and rotation of the hand) should be **chosen depending on the intended goal**.

Example: Use the handle if you plan to pour water.



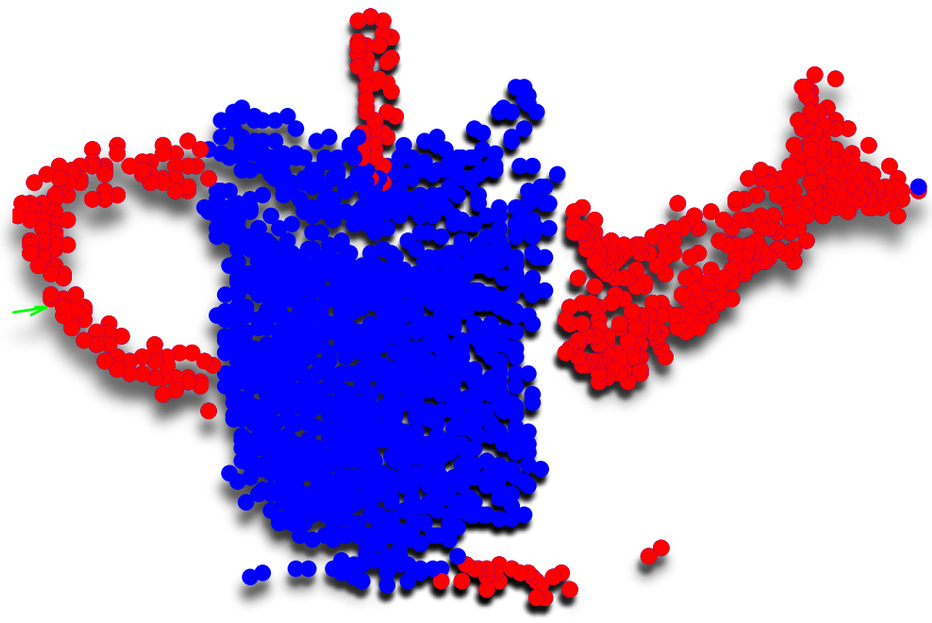
Purposeful Grasping

Barrett® hand

3D image of an unknown object

Purposeful Grasping

Barrett® hand

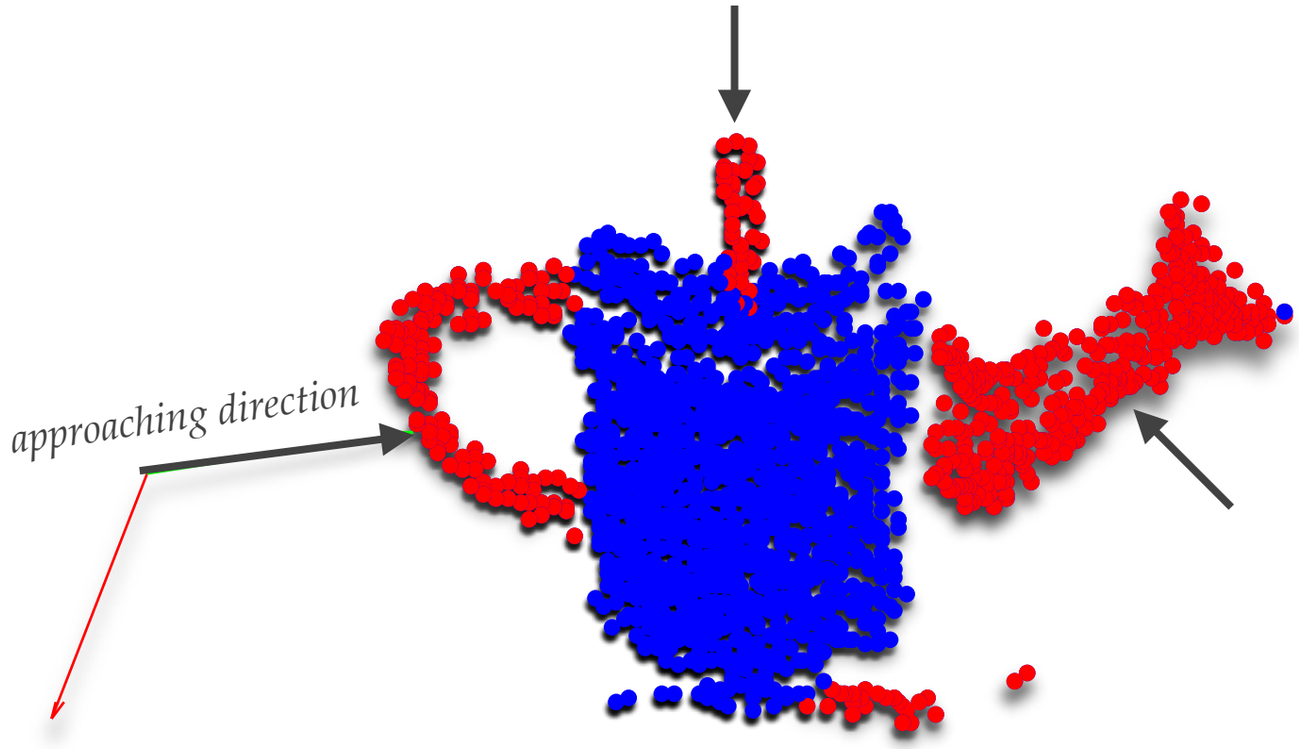


segmented object

Vision: Segment the object into parts

Purposeful Grasping

Barrett® hand



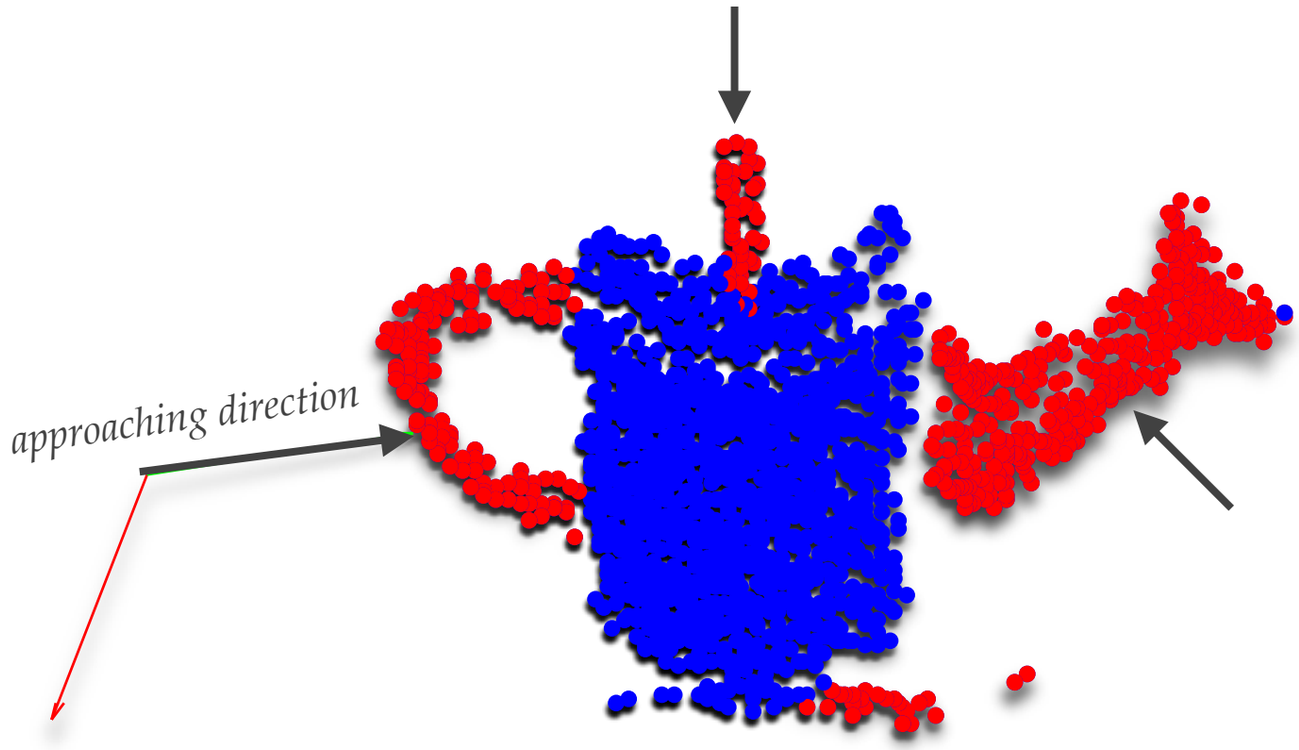
segmented object

Vision: Segment the object into parts

Planning: Simulate grasping actions for each part

Purposeful Grasping

Barrett® hand



segmented object

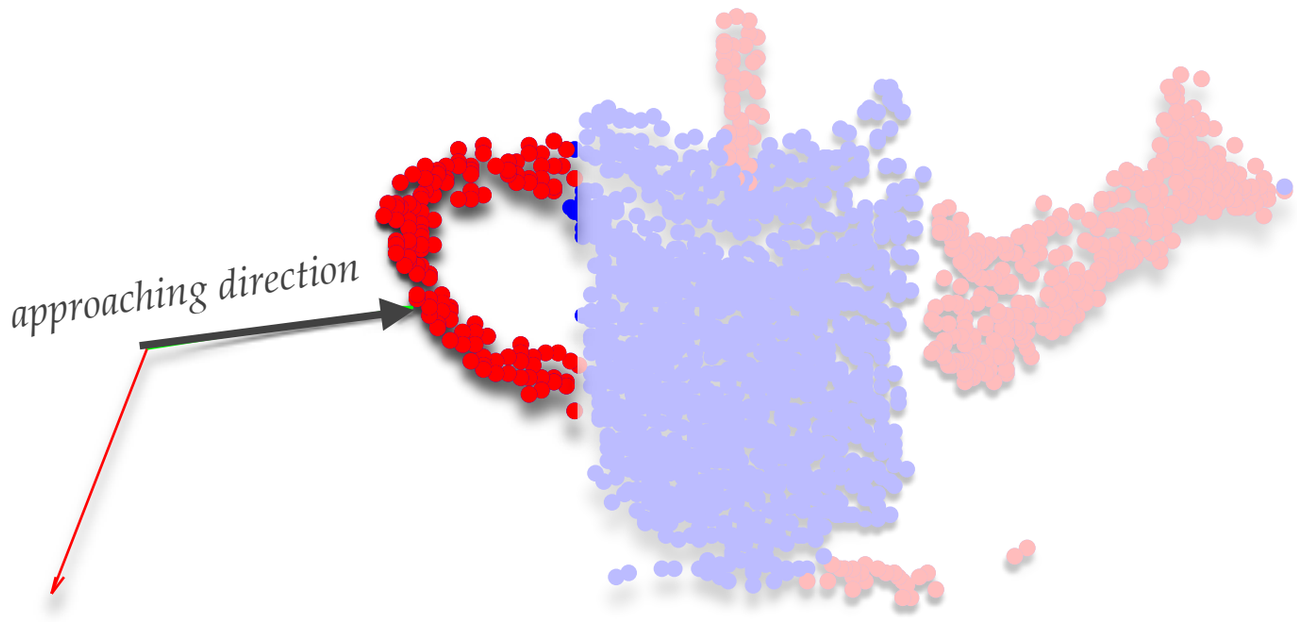
Vision: Segment the object into parts

Planning: Simulate grasping actions for each part

Planning-driven Vision:
Segment the object according to the intended goal

Purposeful Grasping

Barrett® hand



segmented object

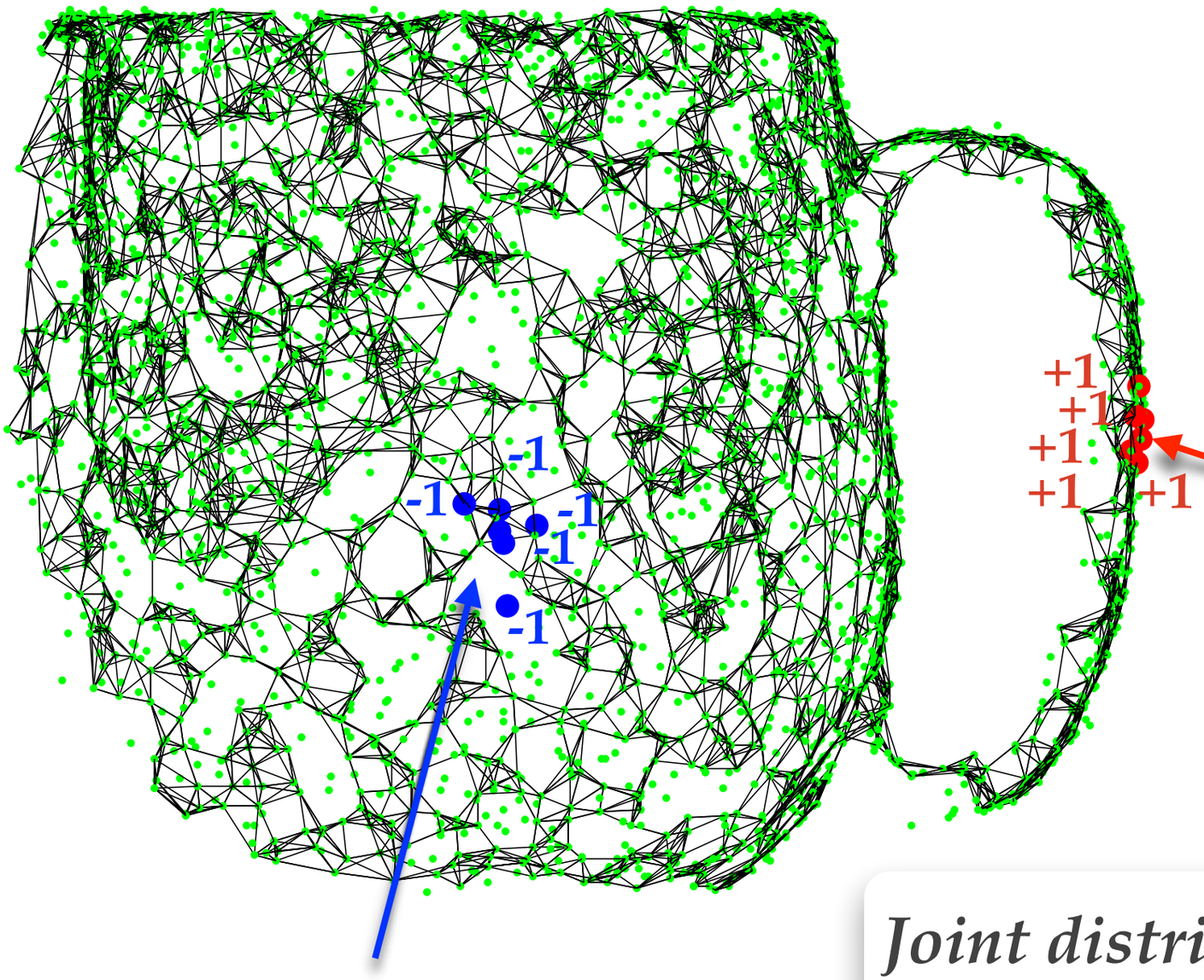
Vision: Segment the object into parts

↓

Planning: Simulate grasping actions for each part

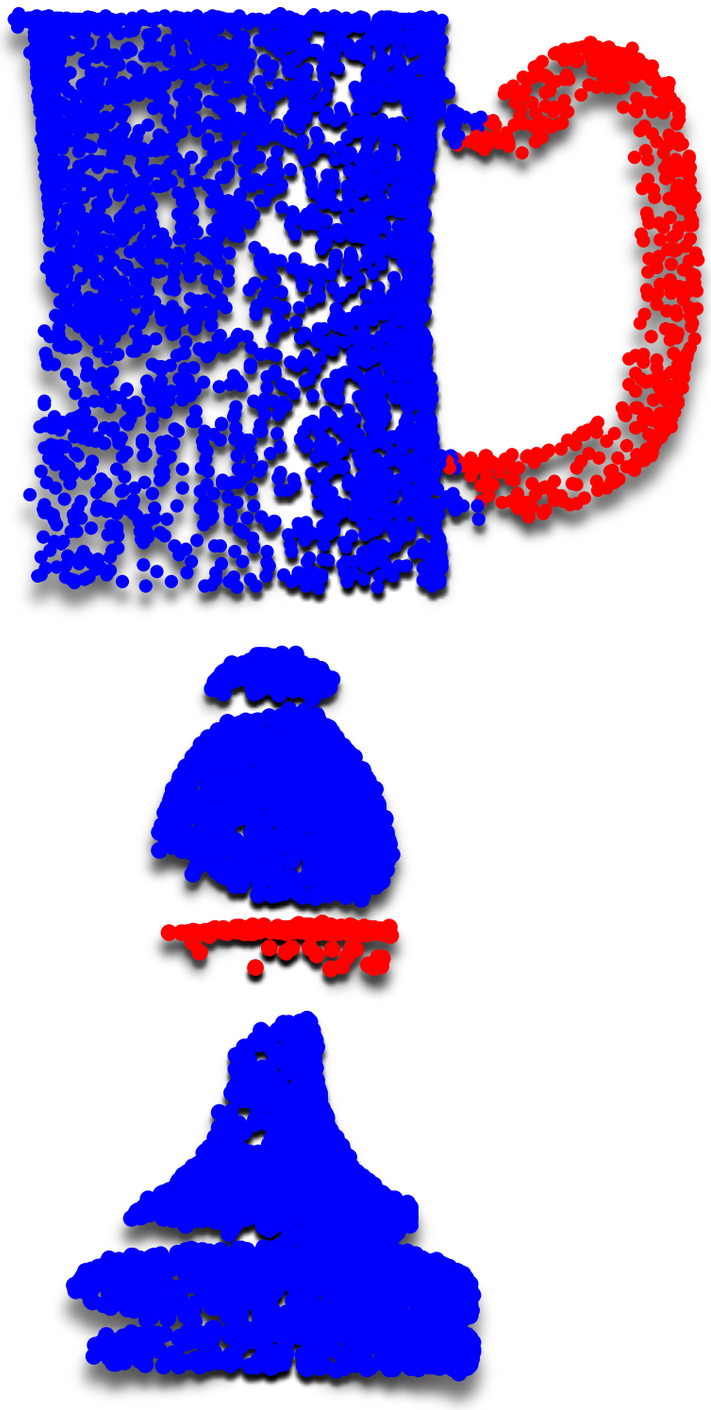
Planning-driven Vision:
Segment the object according to the intended goal

Learning Grasping Points with Associative Markov Networks

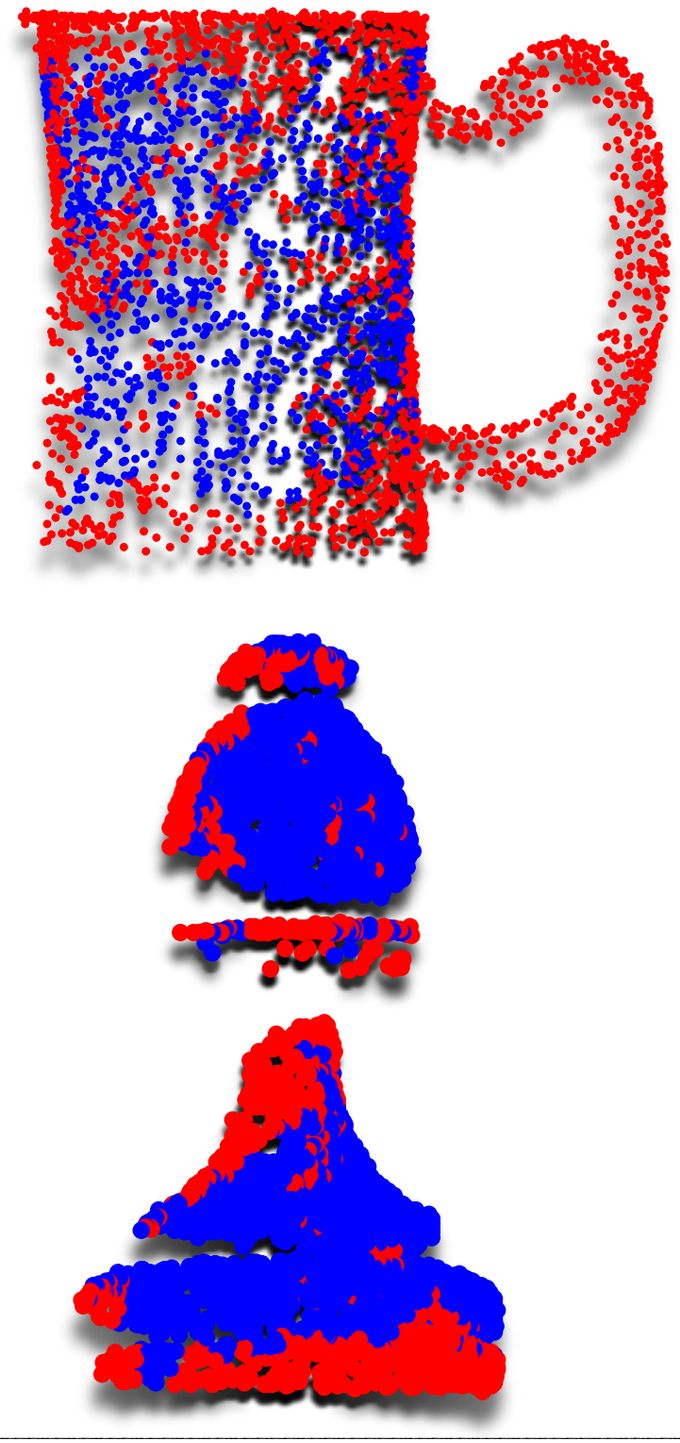


Joint distribution of the labels of all points

$$P(Y) \propto \exp\left(\sum_{i \in \mathcal{V}} y_i w_{node}^T \phi_i + \sum_{\substack{(i,j) \in \mathcal{E} \\ y_i = y_j}} w_{edge}^T \phi_{ij}\right)$$



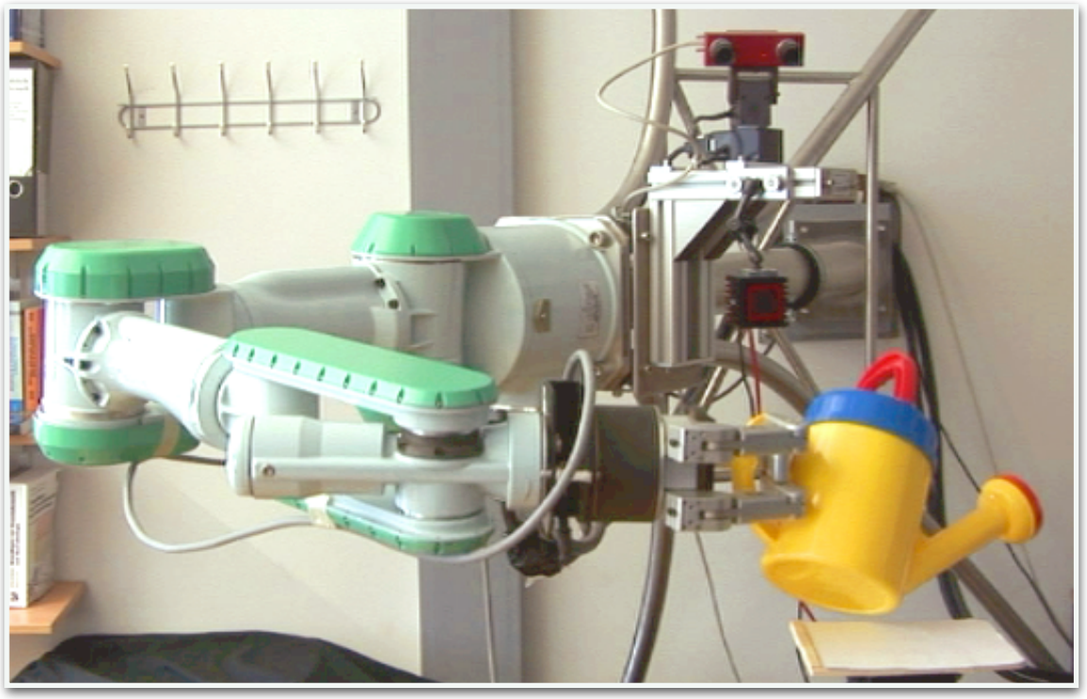
Associative Markov Networks



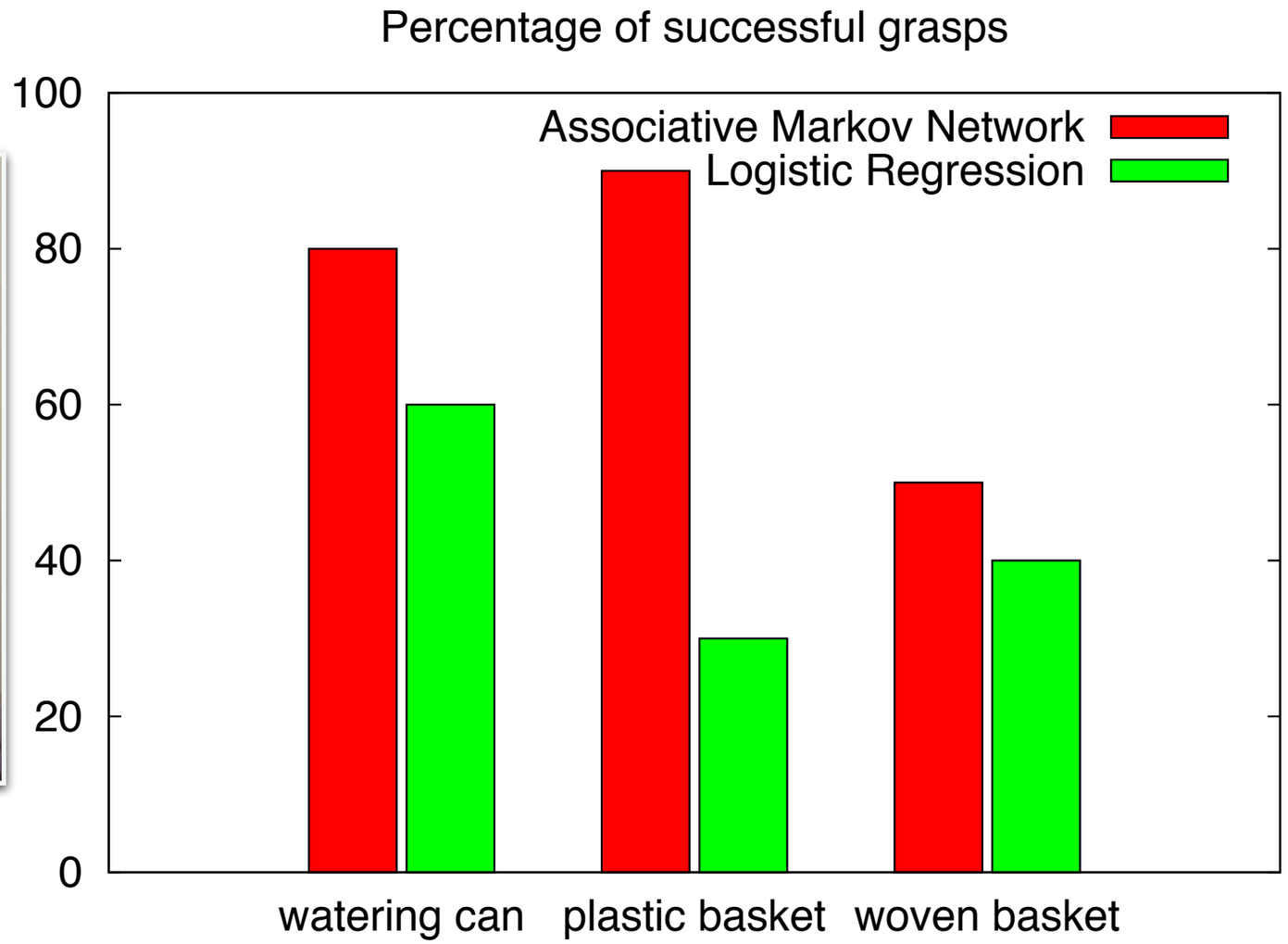
Logistic Regression

A. Boularias *et al.* (2011) in *IEEE International Conference on Intelligent Robots and Systems (IROS)*

Results



Setup



A. Boularias *et al.* (2011) in *IEEE International Conference on Intelligent Robots and Systems (IROS)*

An Autonomous Robot for Rubble Removal

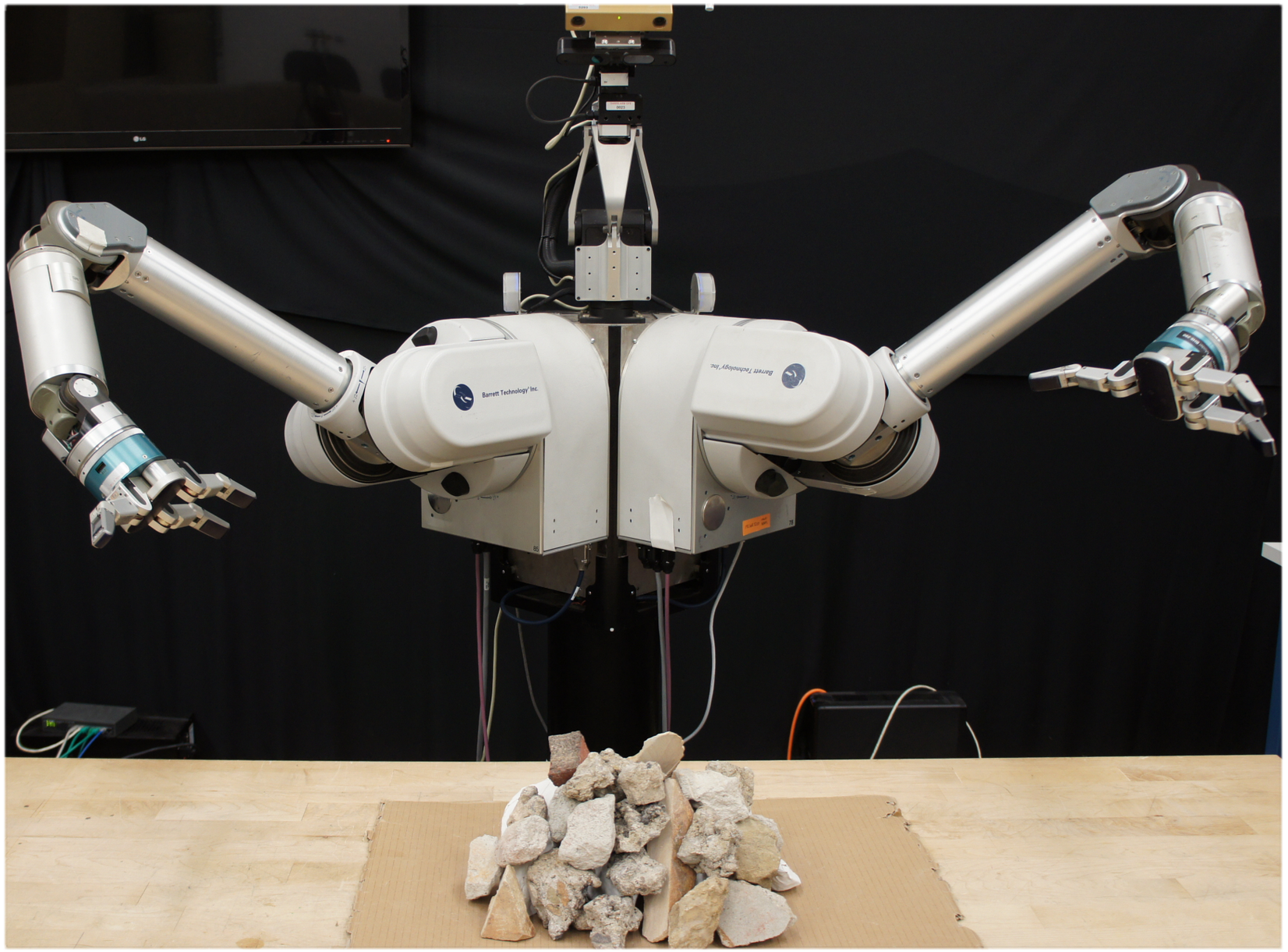
Source: Amedeo Troiani/Getty Images Europe

Source: AFP

Rubble removal is a major challenge in search-and-rescue missions

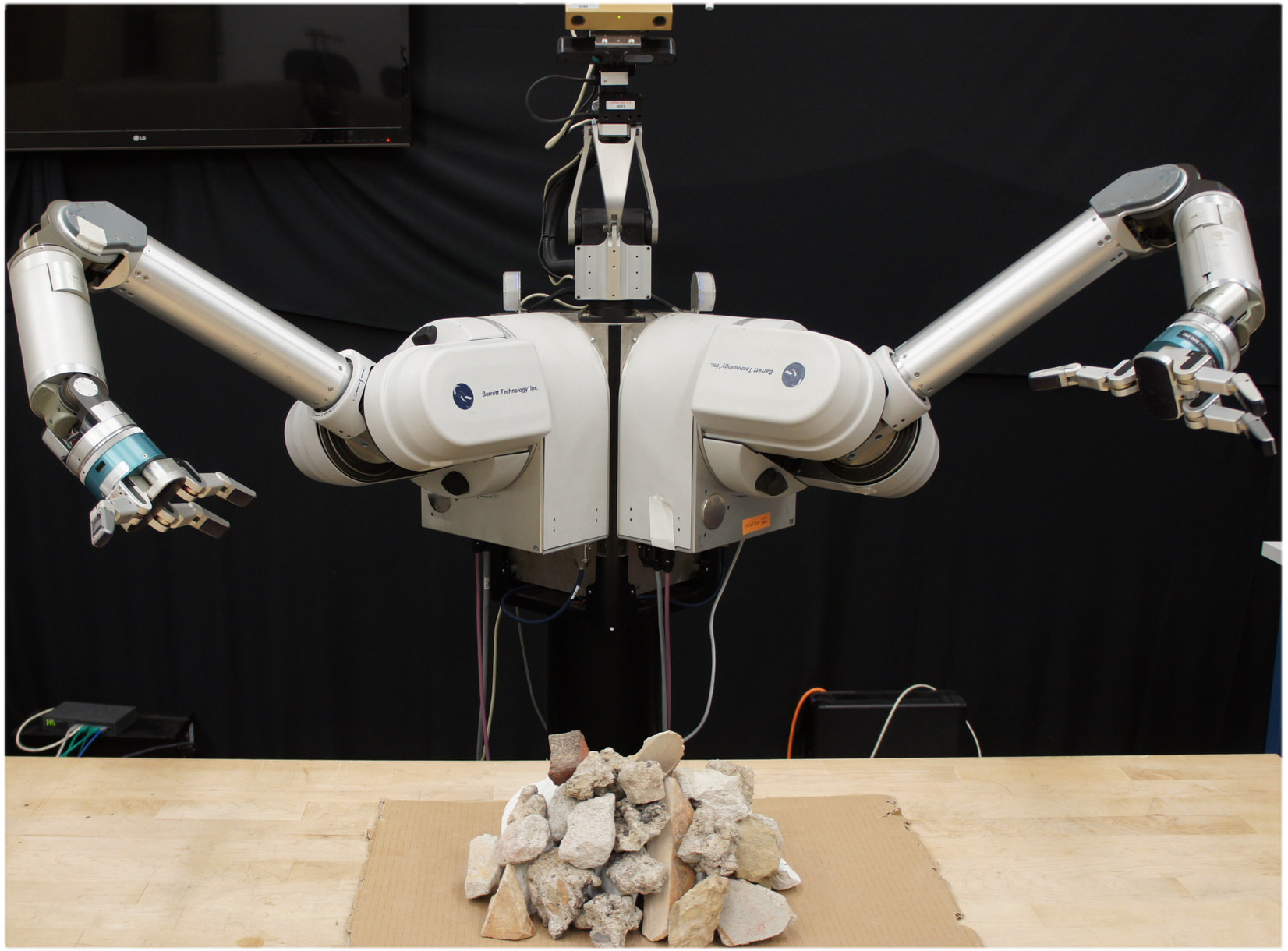
Tele-operation is tedious and requires a human expert

An Autonomous Robot for Rubble Removal



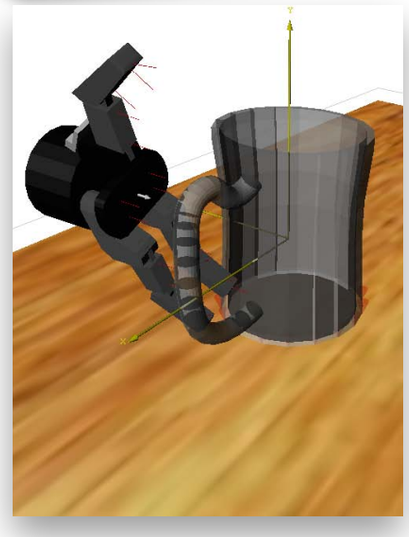
Two *Barrett*® arms and hands with a *Kinect*® camera

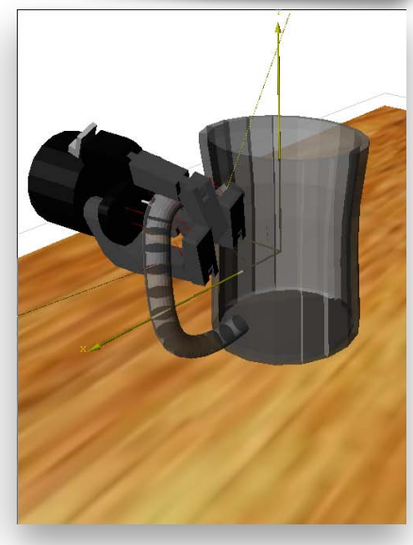
An Autonomous Robot for Rubble Removal



Two *Barrett*® arms and hands with a *Kinect*® camera

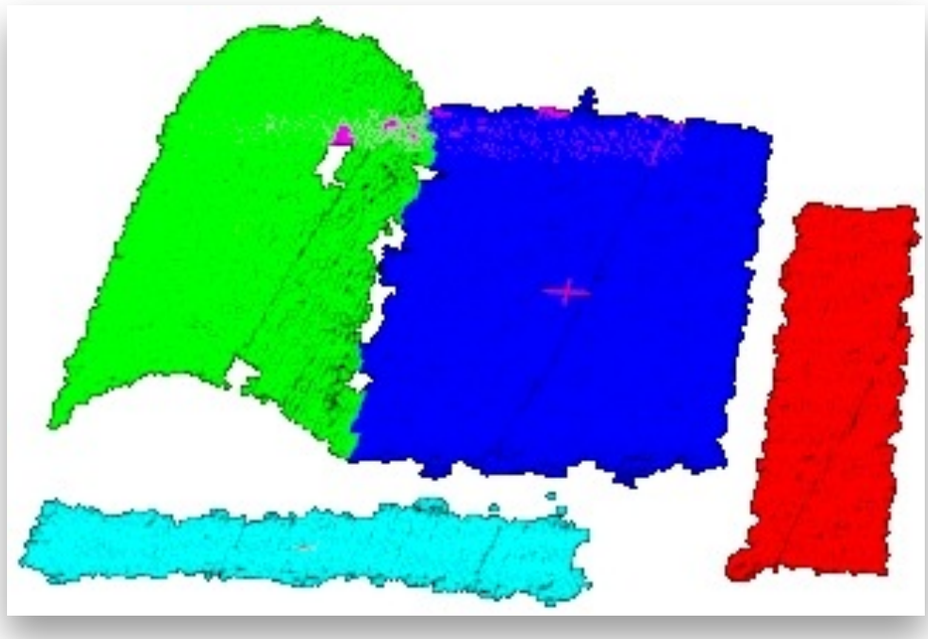
Most autonomous grasping techniques use **models** of the objects

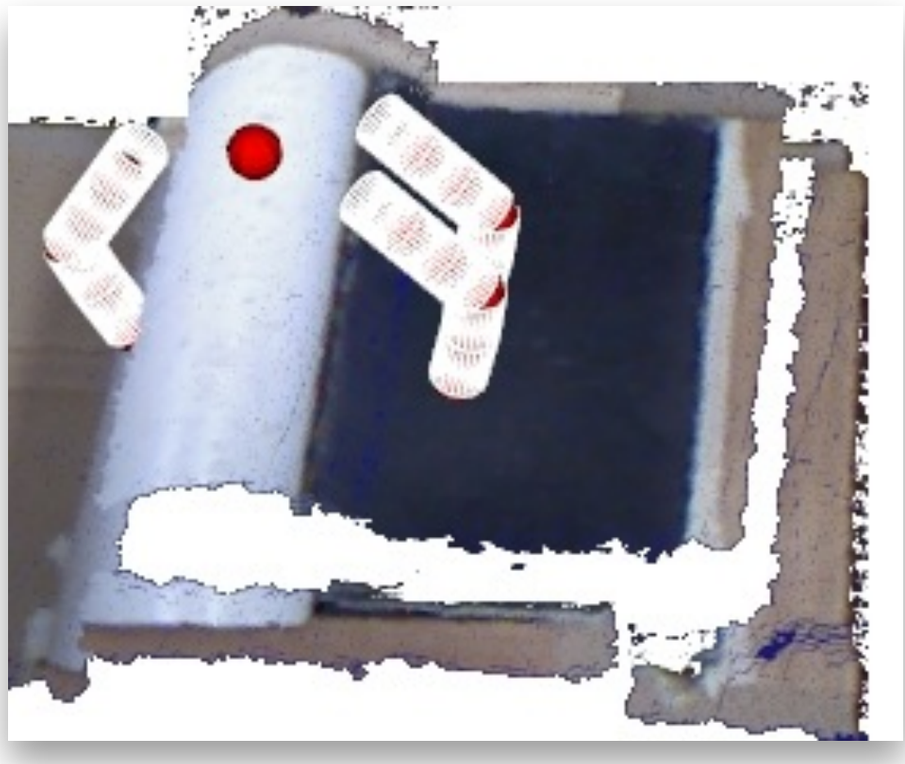




Objects found in rubble, such as rocks, are **irregular** and **unknown** to the robot. Therefore, we cannot rely on models!

Grasping Regular Objects: A Simple Heuristic



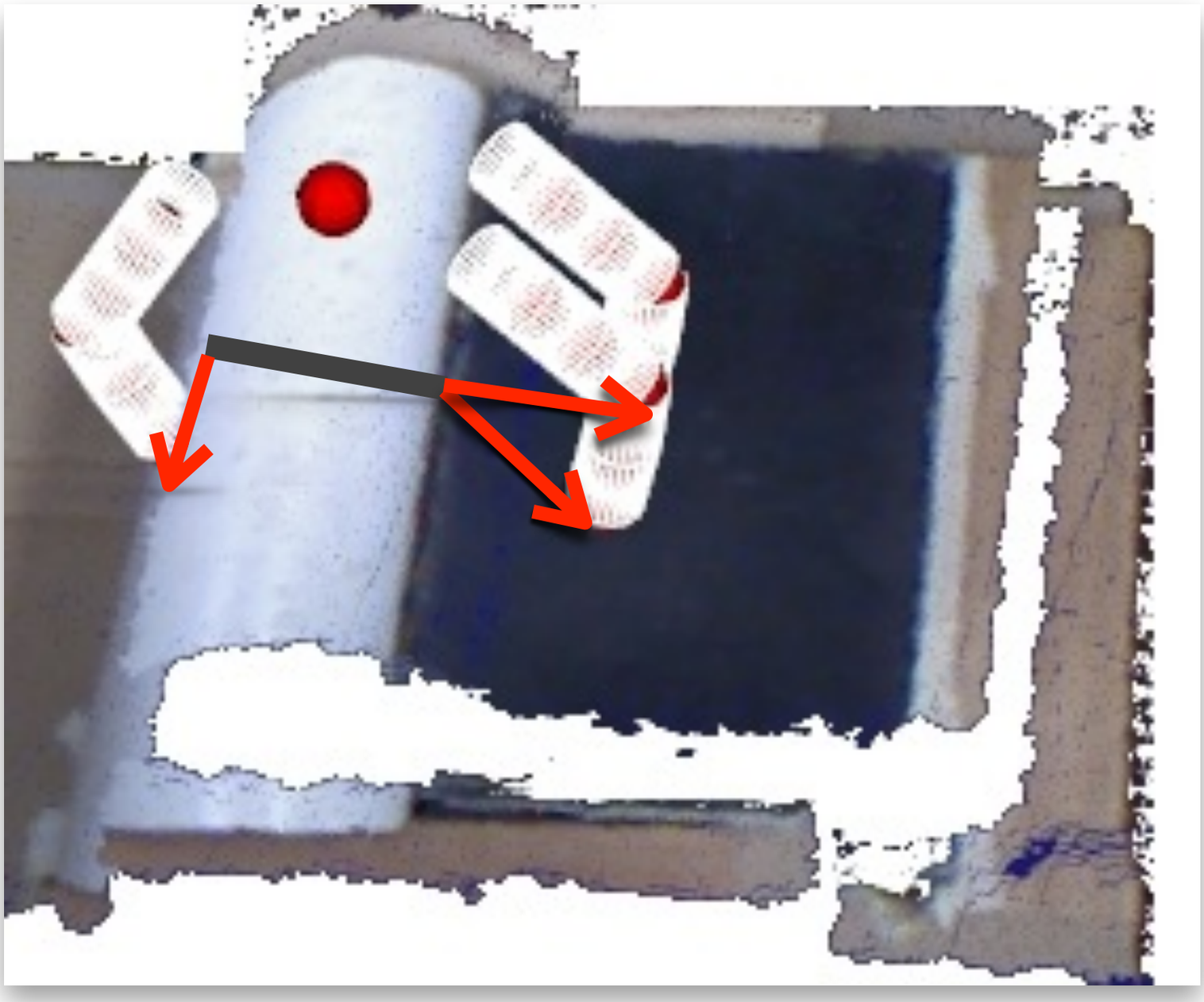


Take a 3D image of the scene

Segment the 3D point cloud into facets

Simulate grasping actions for each facet by checking for collisions

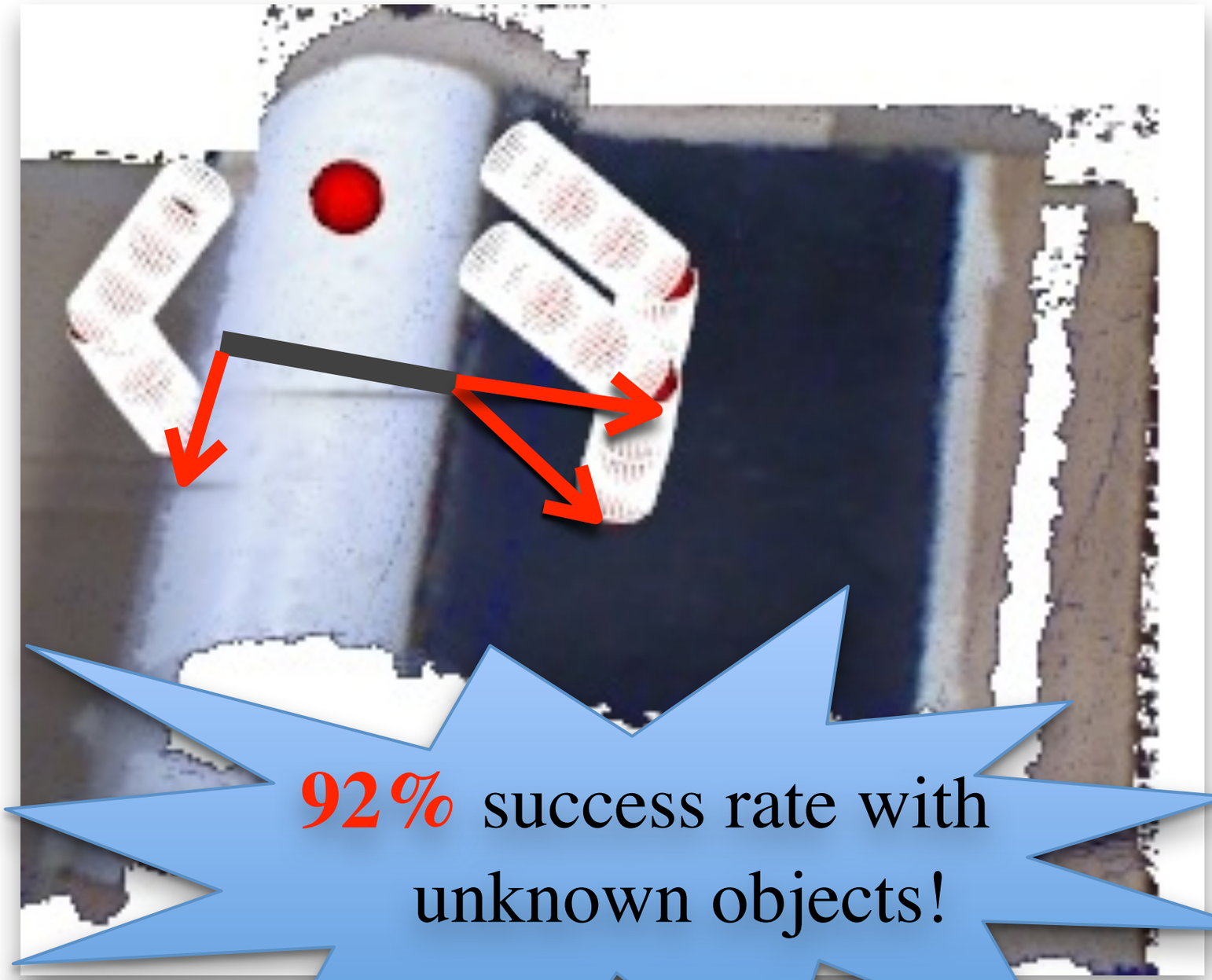
Grasping Regular Objects: A Simple Heuristic



Calculate the angles between the fingertips of the robotic hand and the extreme points of the object

Execute the grasp that has the maximum contact angles

Grasping Regular Objects: A Simple Heuristic



92% success rate with unknown objects!

Calculate the angles between the fingertips of the robotic hand and the extreme points of the object

Execute the grasp that has the maximum contact angles

A. Boularias *et al.* (2014) in *Conference of the Association for the Advancement of Artificial Intelligence (AAAI)*

Grasping Irregular Objects

The number of grasping actions that need to be simulated and evaluated is **very high** (thousands) when the objects are irregular

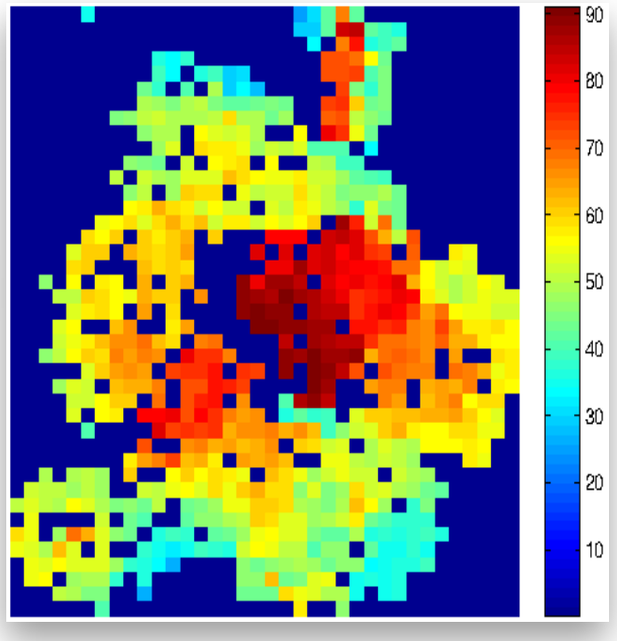
Simulation is **too slow** (0.1 second per action) for real-time requirements

Grasping Irregular Objects

Idea: Learn to predict the outcome of the simulation

Pile of rocks

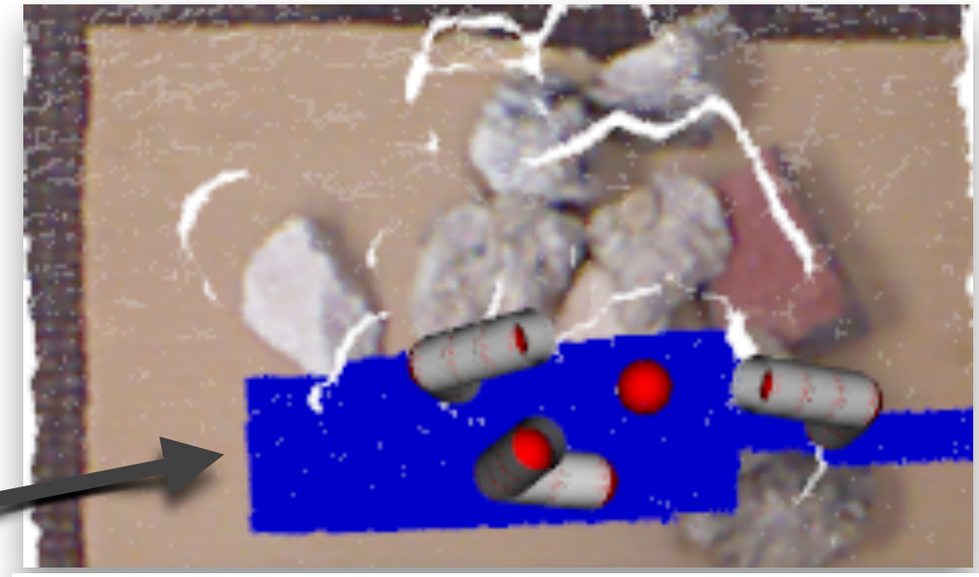
Real-time prediction

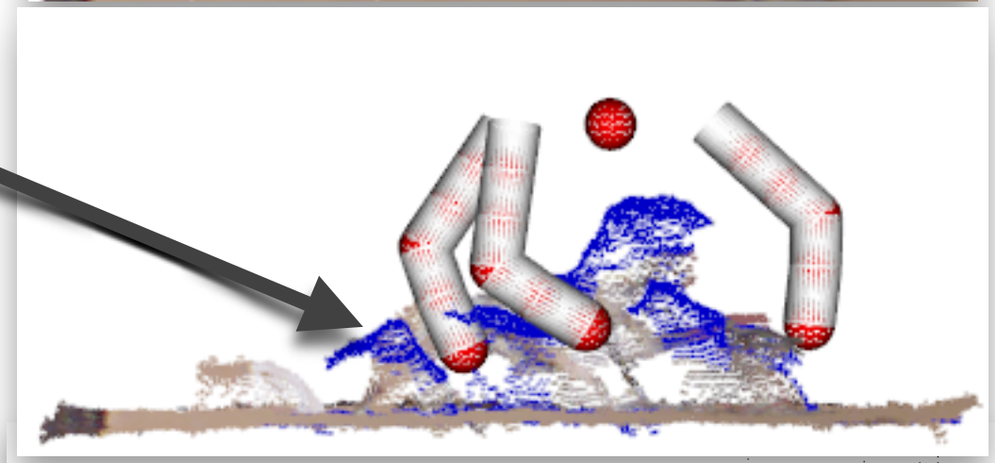


Predicted success probabilities
using *k-Nearest Neighbors*

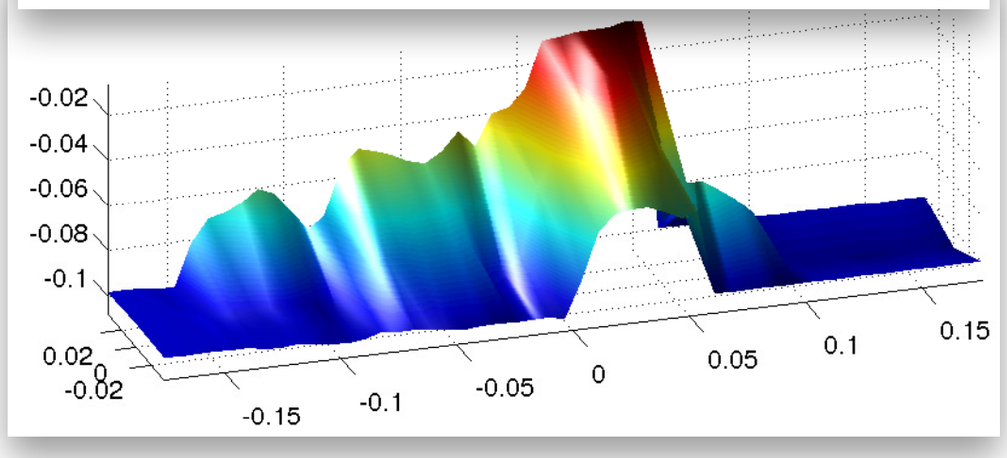
Features of Grasping Actions

Extract all the points in the 3D cloud that may collide with the robot's hand (the **blue strip** in the figures)

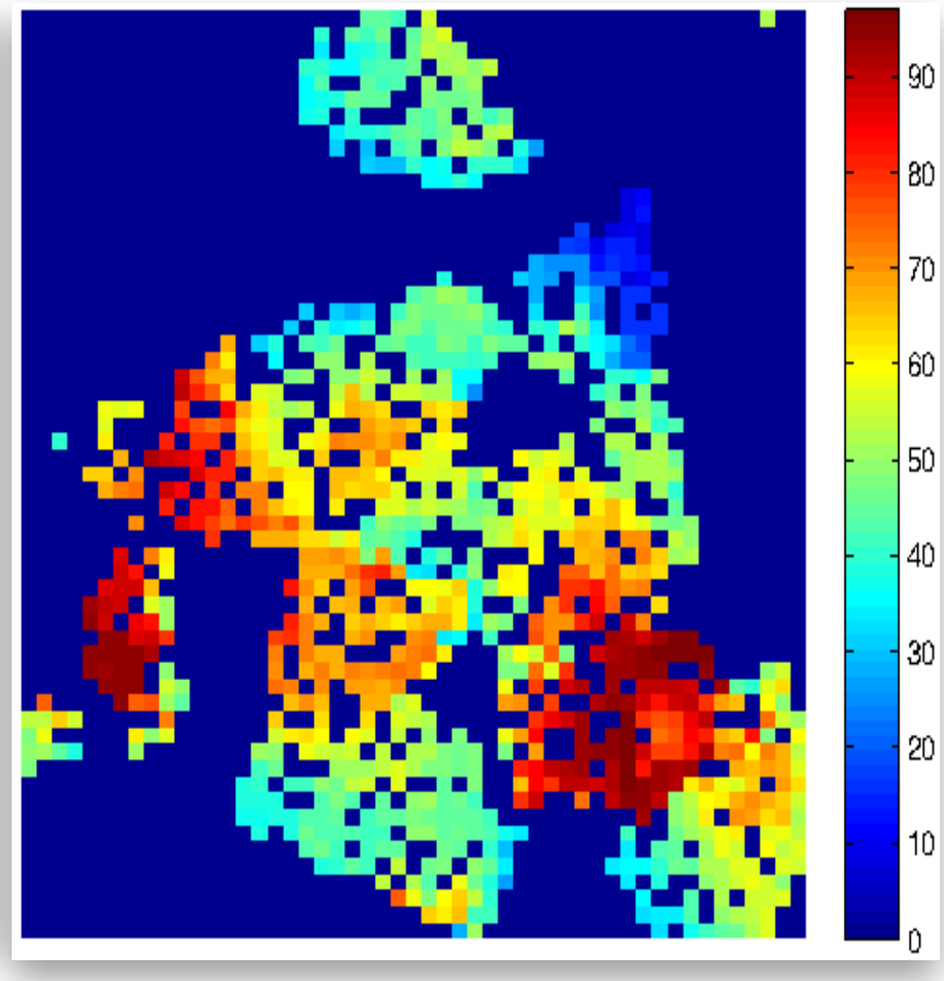




Feature matrix: elevations of the points of collision

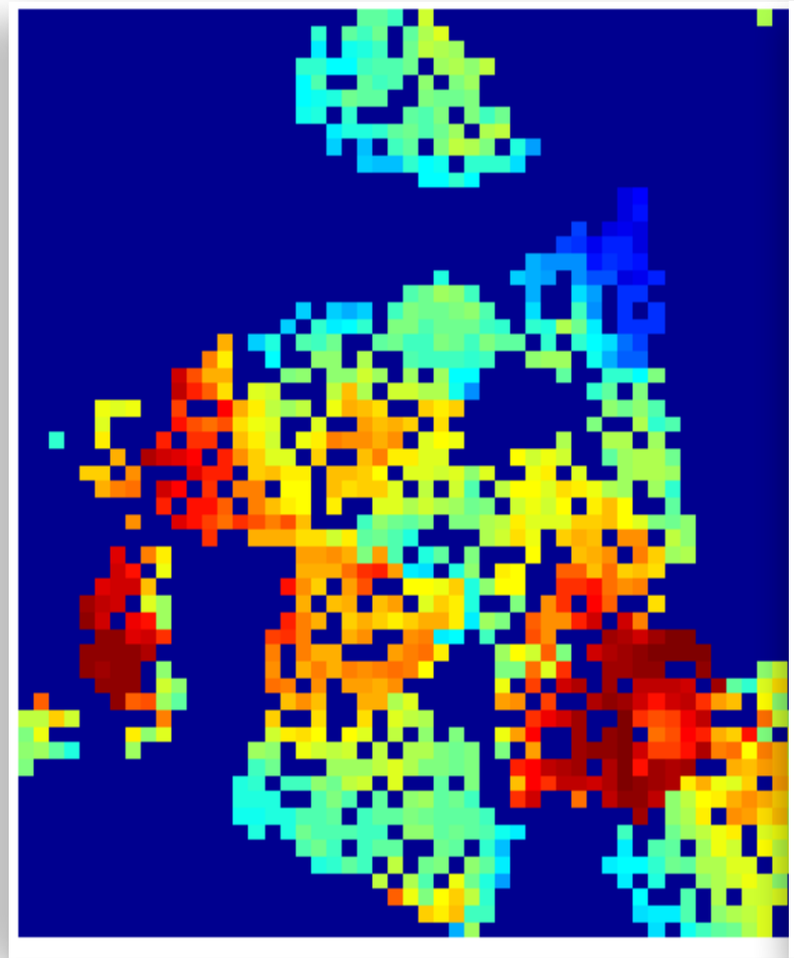


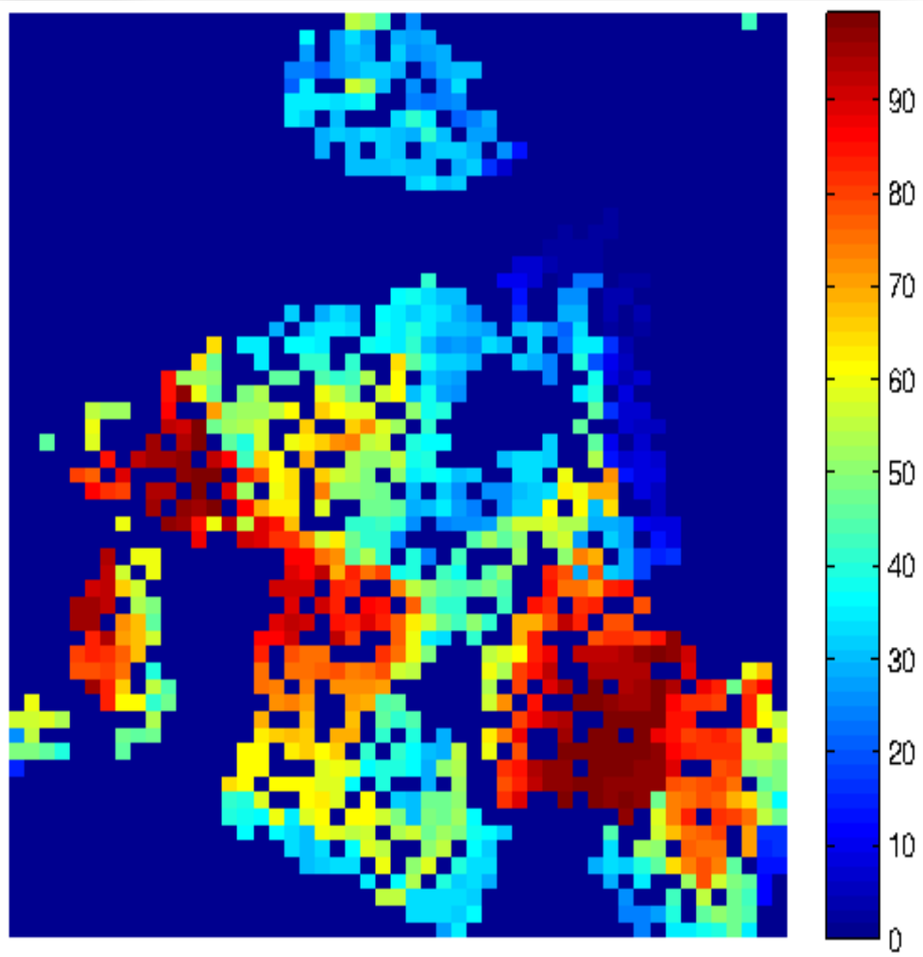
Predicting Success Probabilities of Grasping Action



Learned probabilities,
obtained in **2 seconds**
with *k-NN*

Predicting Success Probabilities of Grasping Action

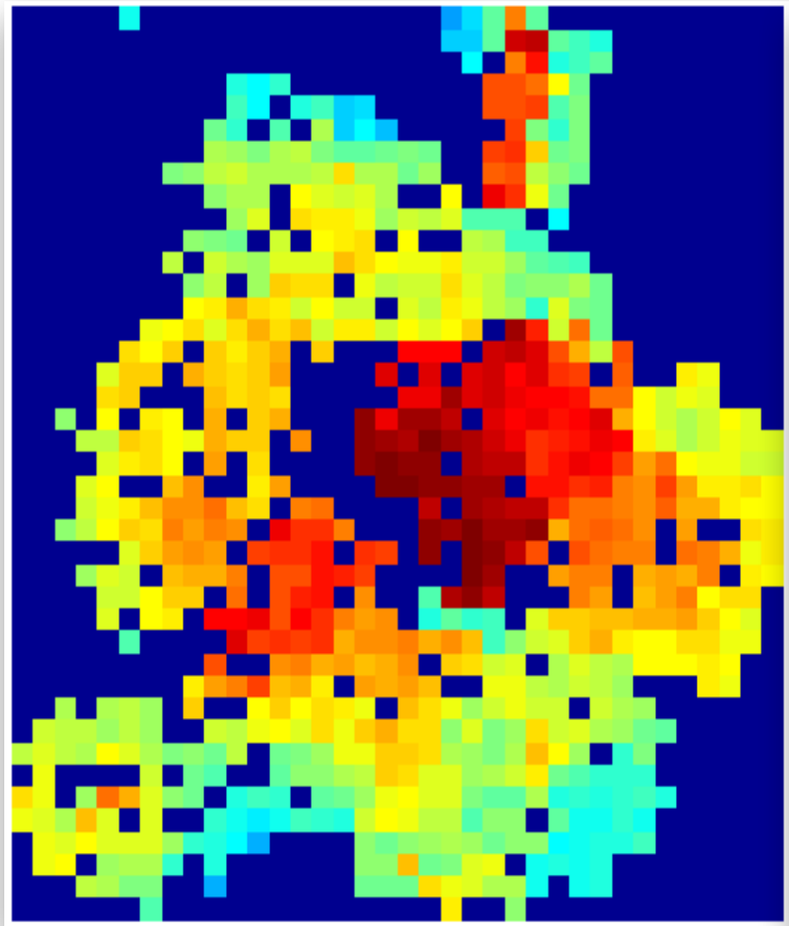


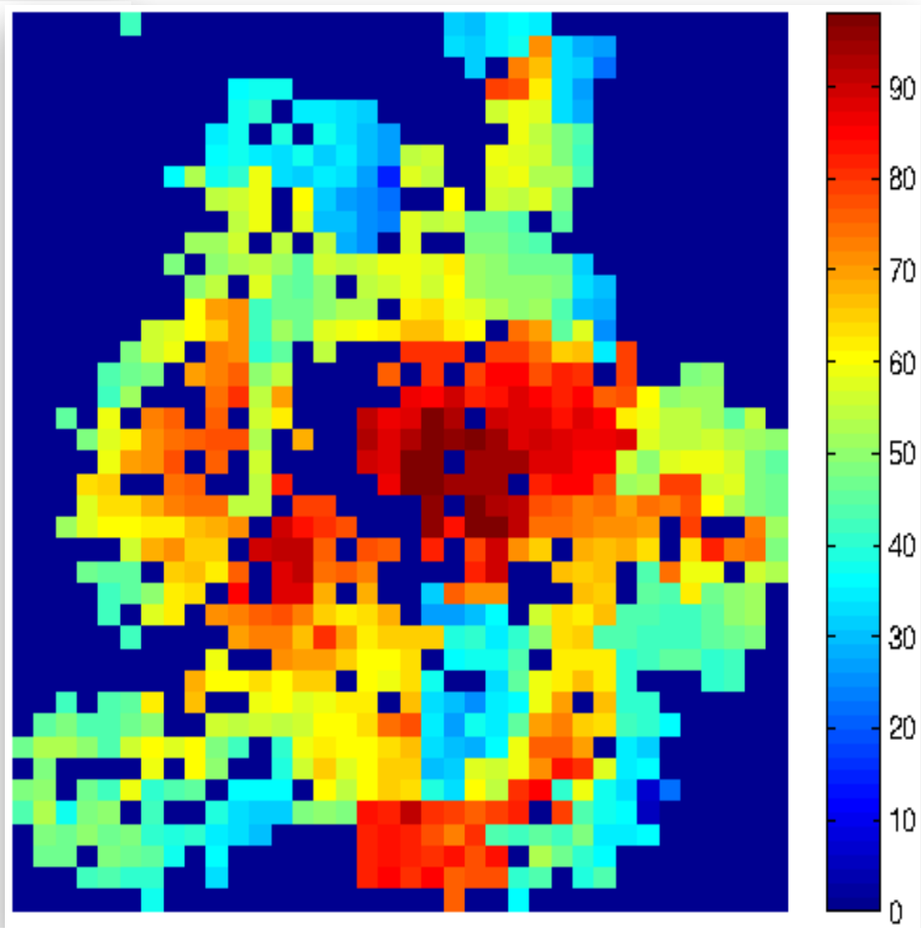


Learned probabilities, obtained in **2 seconds** with *k-NN*

Ground truth, obtained in **200 seconds** from simulations

Predicting Success Probabilities of Grasping Action

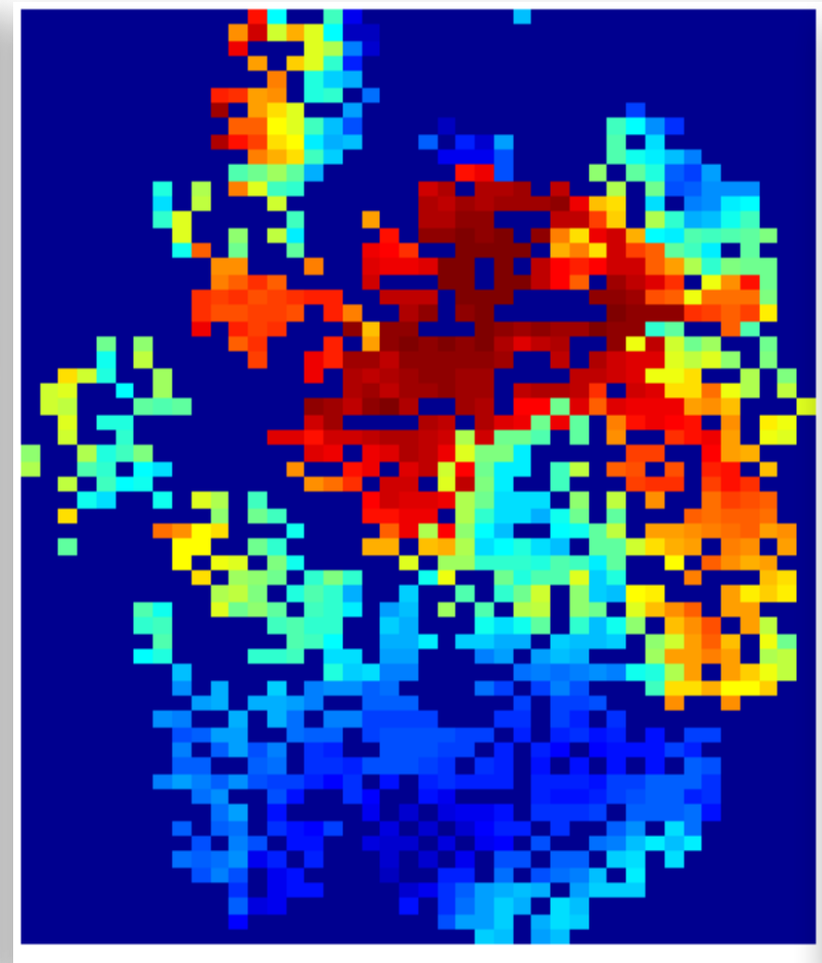


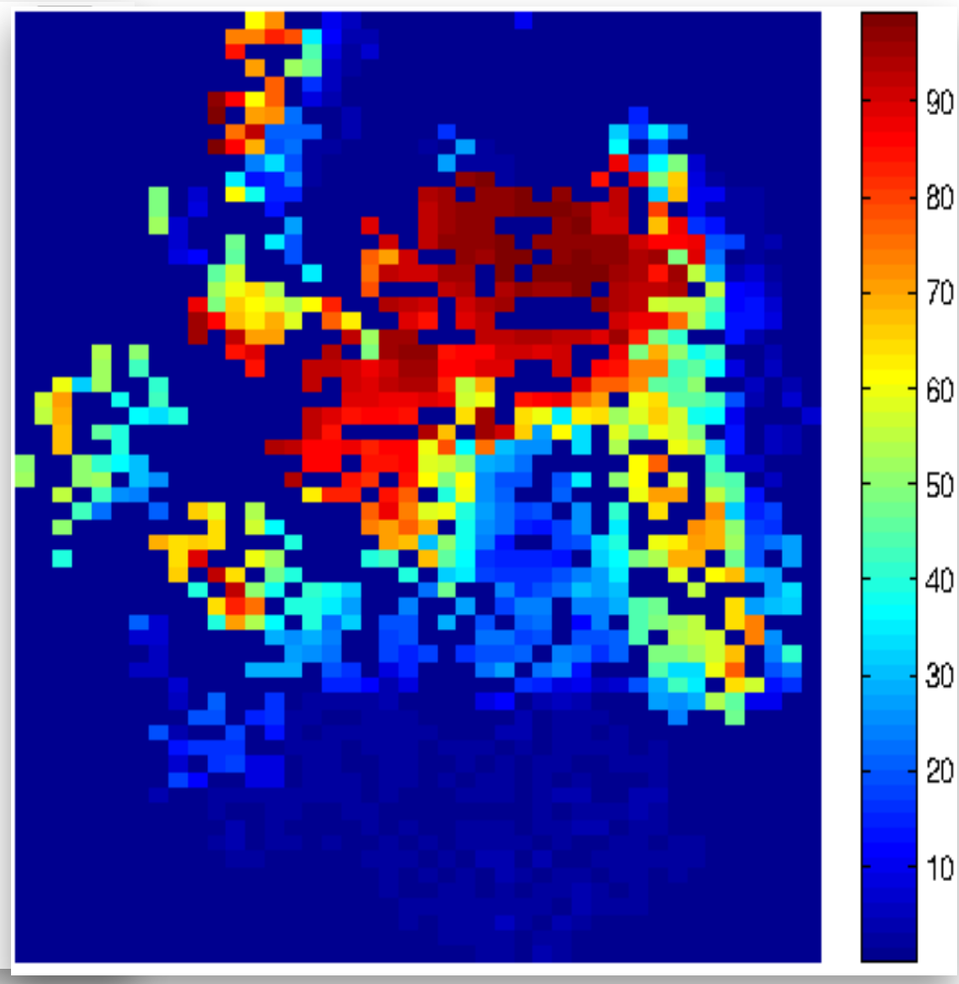


Learned probabilities, obtained in **2 seconds** with *k-NN*

Ground truth, obtained in **200 seconds** from simulations

Predicting Success Probabilities of Grasping Action

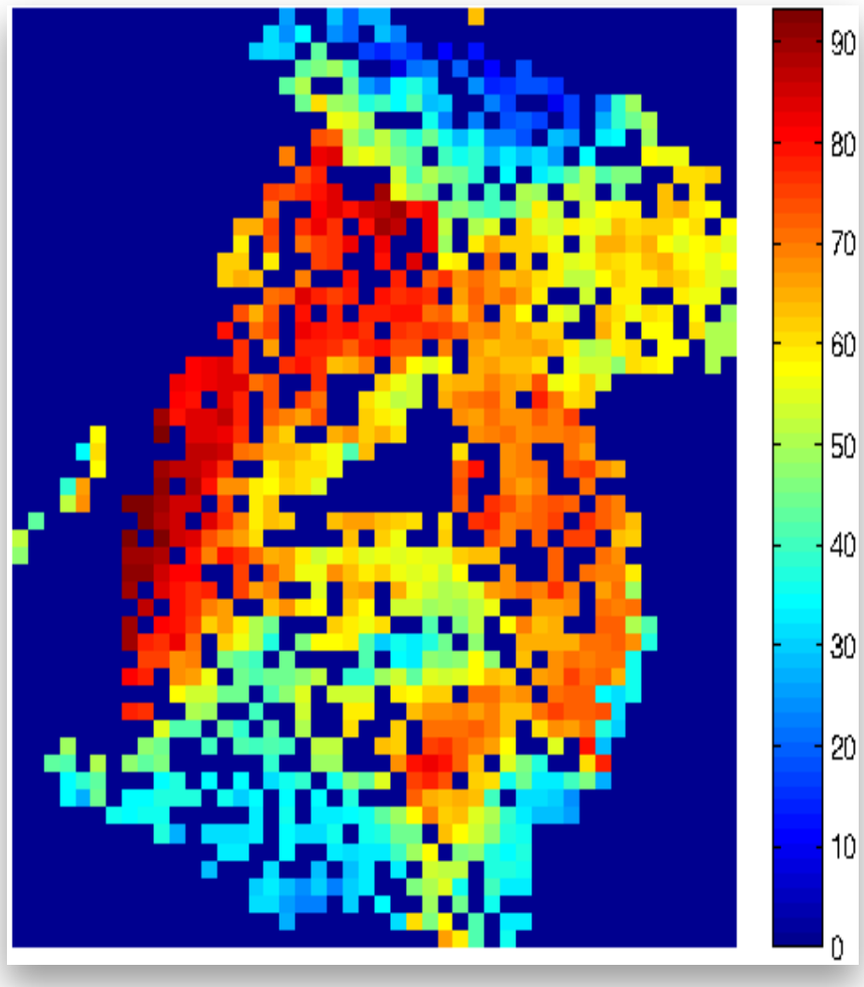




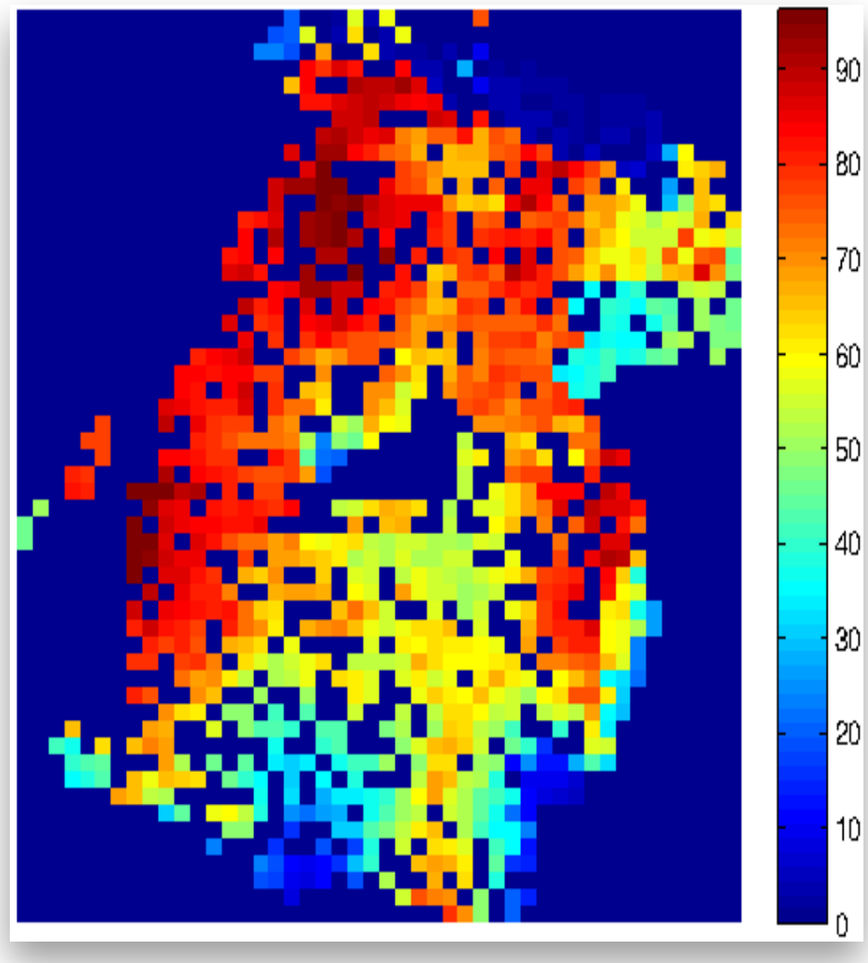
Learned probabilities, obtained in **2 seconds** with *k-NN*

Ground truth, obtained in **200 seconds** from simulations

Predicting Success Probabilities of Grasping Action

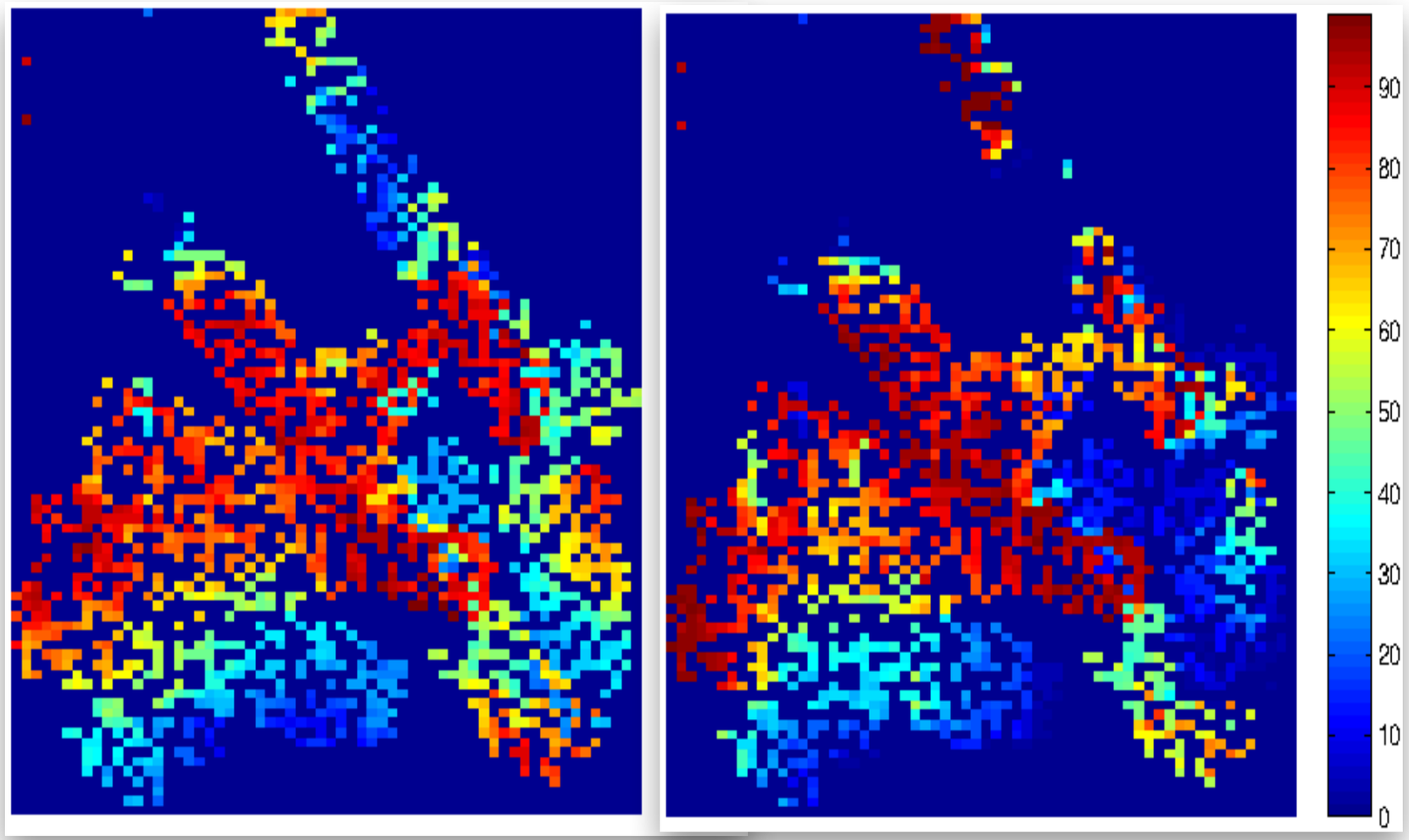


Learned probabilities, obtained in **2 seconds** with *k-NN*



Ground truth, obtained in **200 seconds** from simulations

Predicting Success Probabilities of Grasping Action



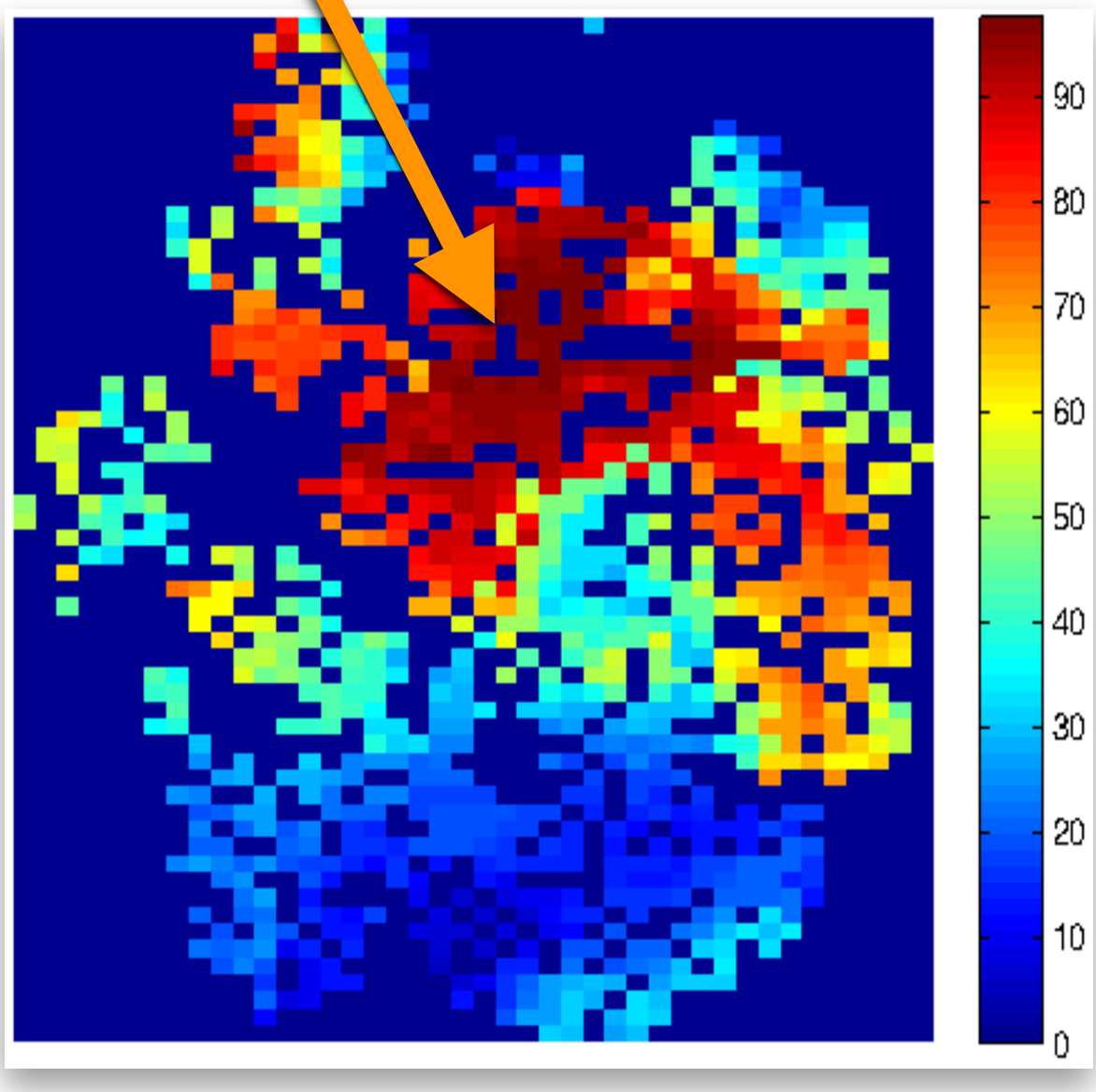
Learned probabilities, obtained in **2 seconds** with *k-NN*

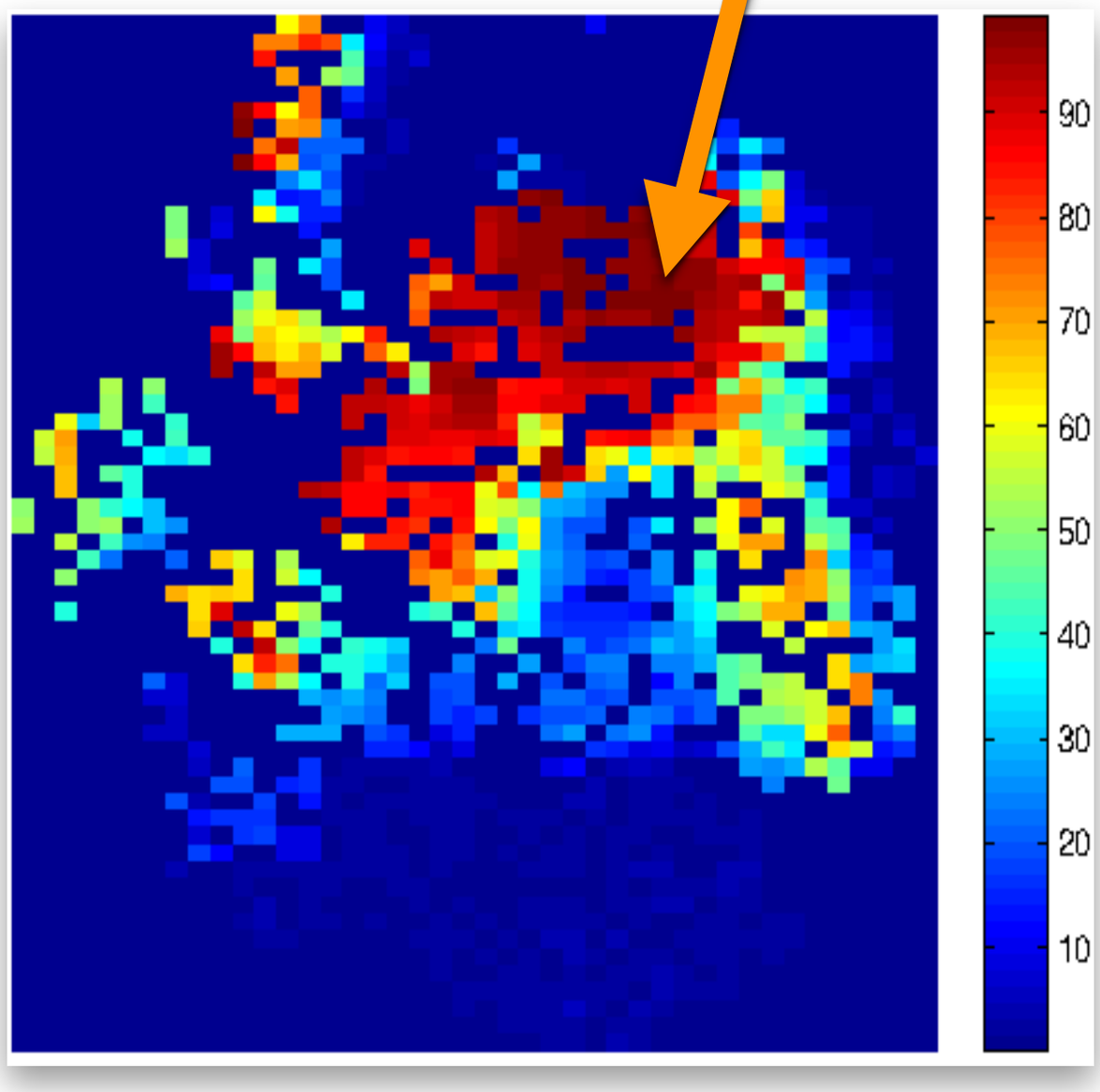
Ground truth, obtained in **200 seconds** from simulations

Best grasping point according to the learned probabilities

≠

Best grasping point according to the simulator

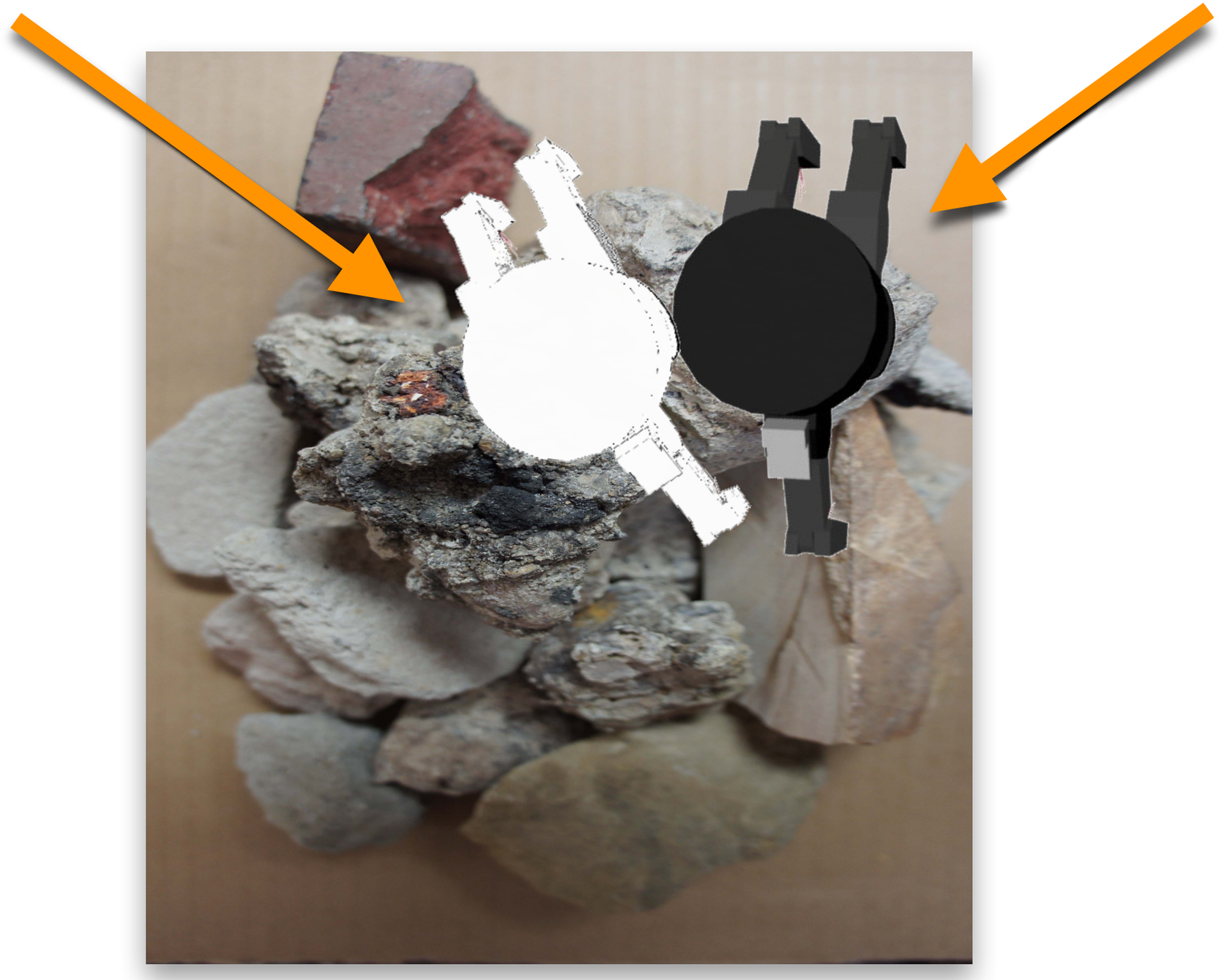




Best grasping point according to the learned probabilities

≠

Best grasping point according to the simulator

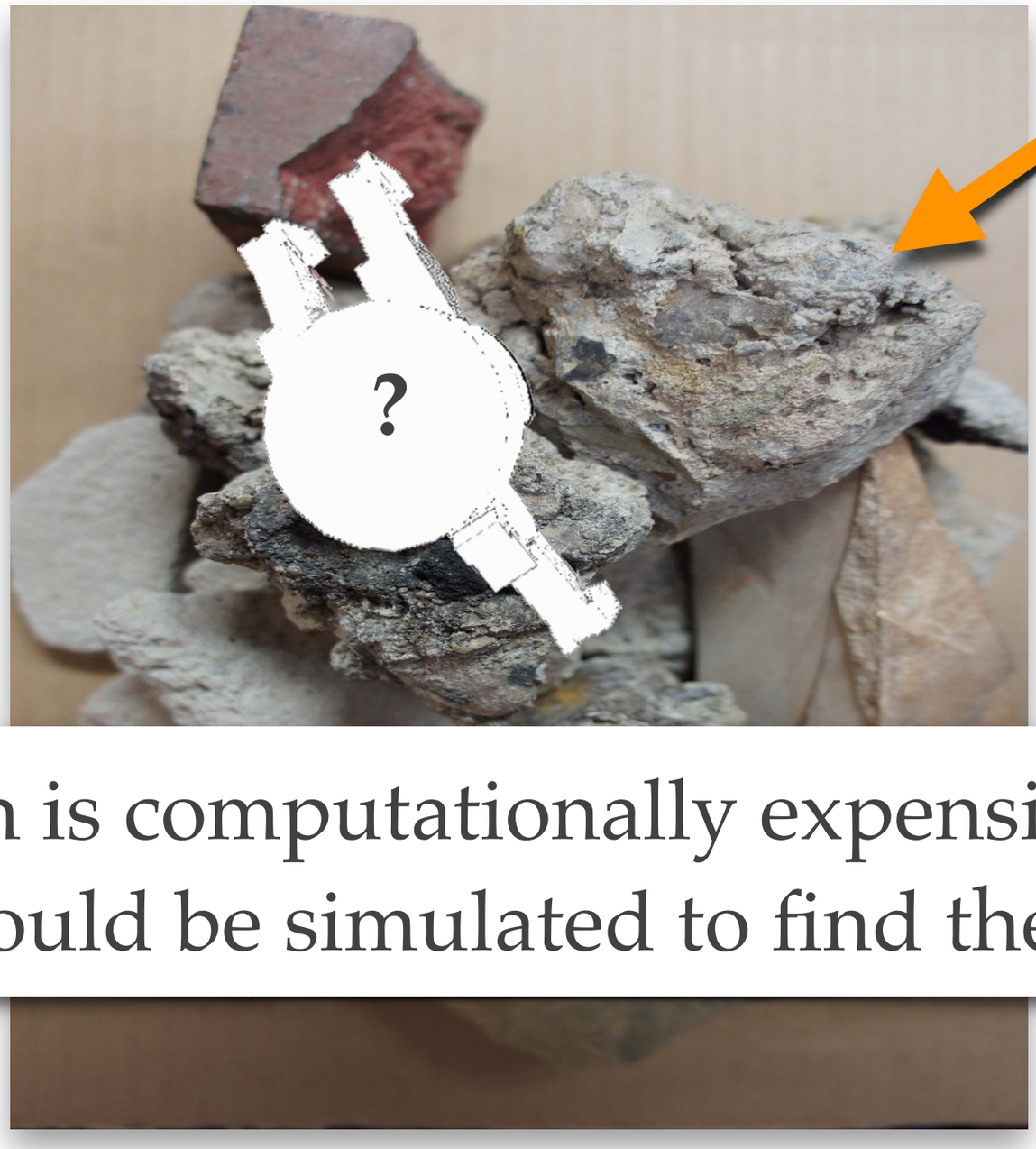


Fine-tuning in Simulation

Goal: best grasping point in simulation

Search, in simulation, for the best action by starting from the best action according to the learned probabilities

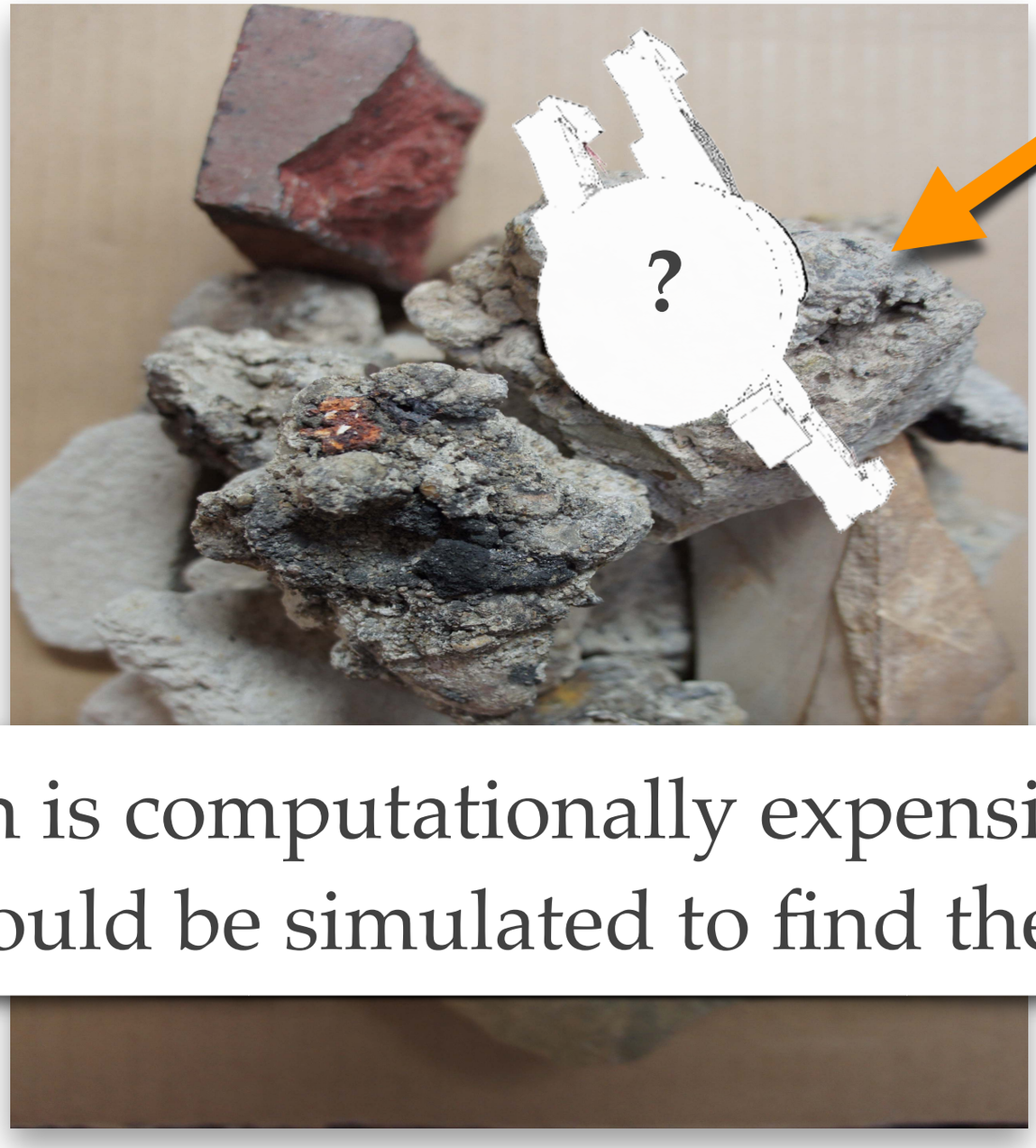
Fine-tuning in Simulation



Best grasping point according to the simulator

Simulation is computationally expensive, which actions should be simulated to find the best one?

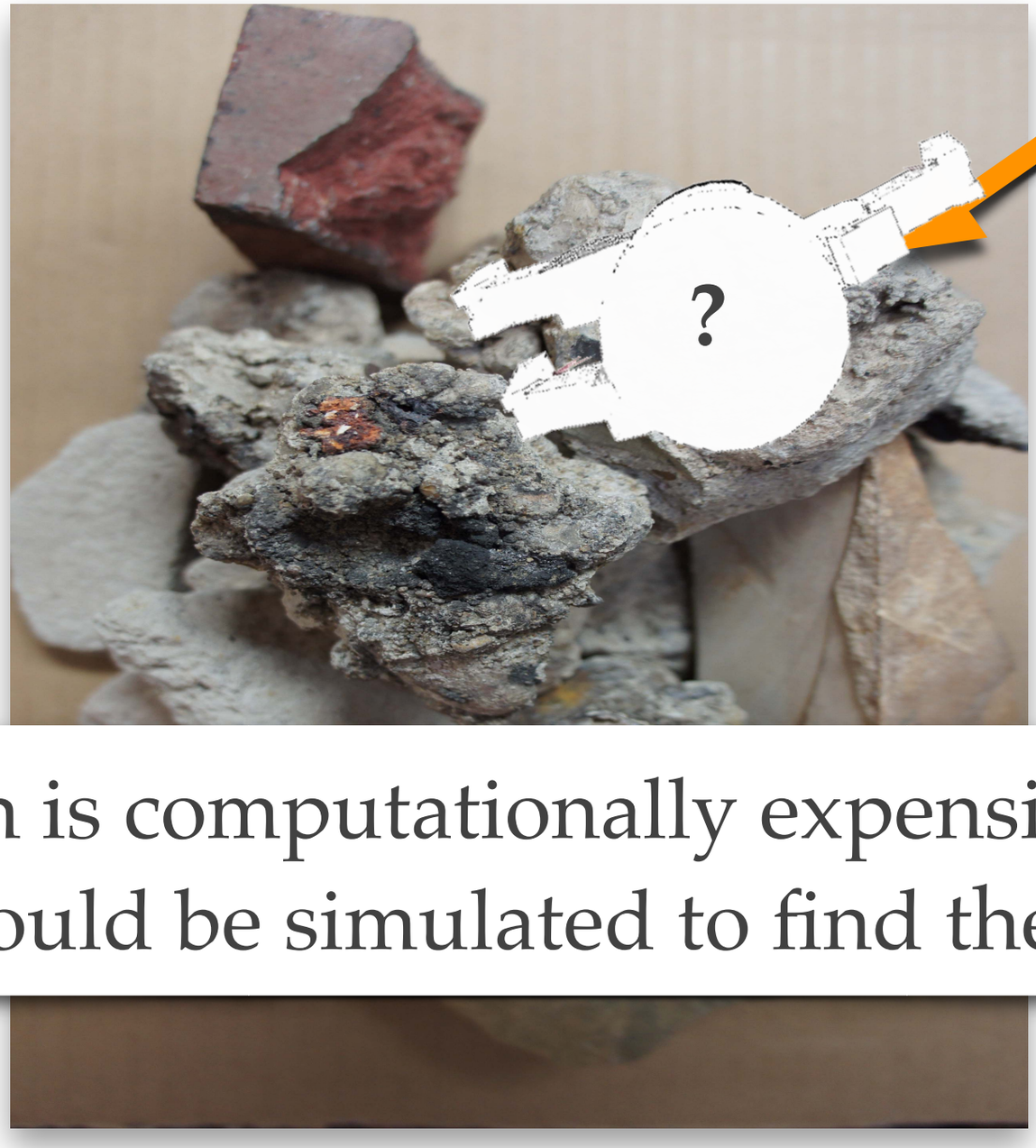
Fine-tuning in Simulation



Best grasping point according to the simulator

Simulation is computationally expensive, which actions should be simulated to find the best one?

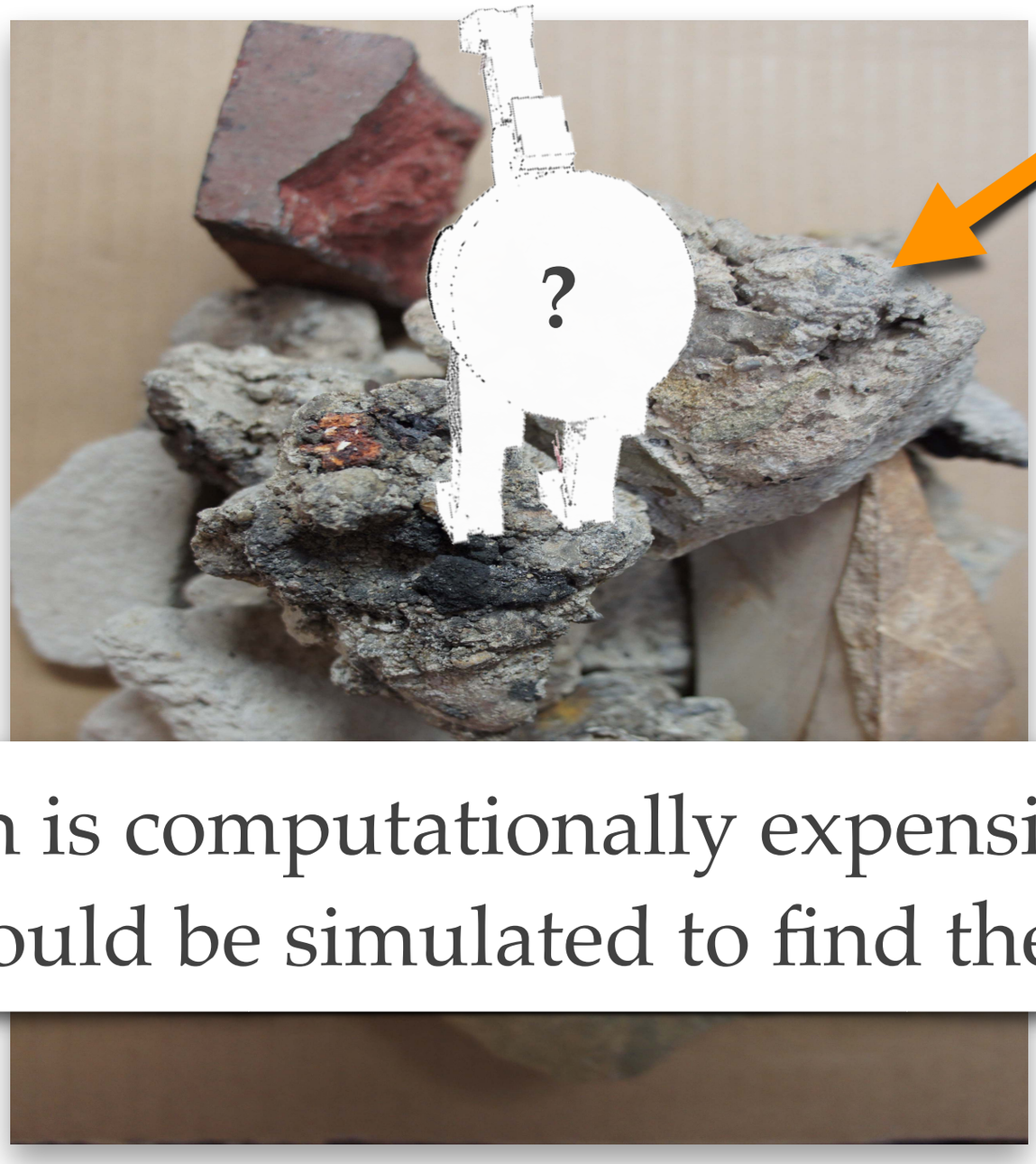
Fine-tuning in Simulation



Best grasping point according to the simulator

Simulation is computationally expensive, which actions should be simulated to find the best one?

Fine-tuning in Simulation



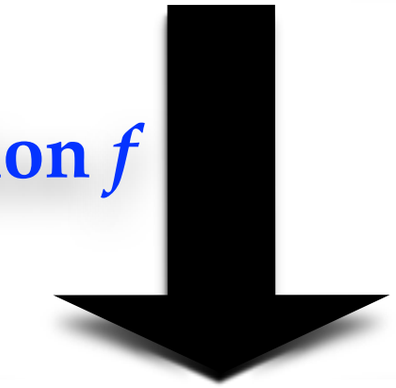
Best grasping point according to the simulator

Simulation is computationally expensive, which actions should be simulated to find the best one?

Black-box Optimization

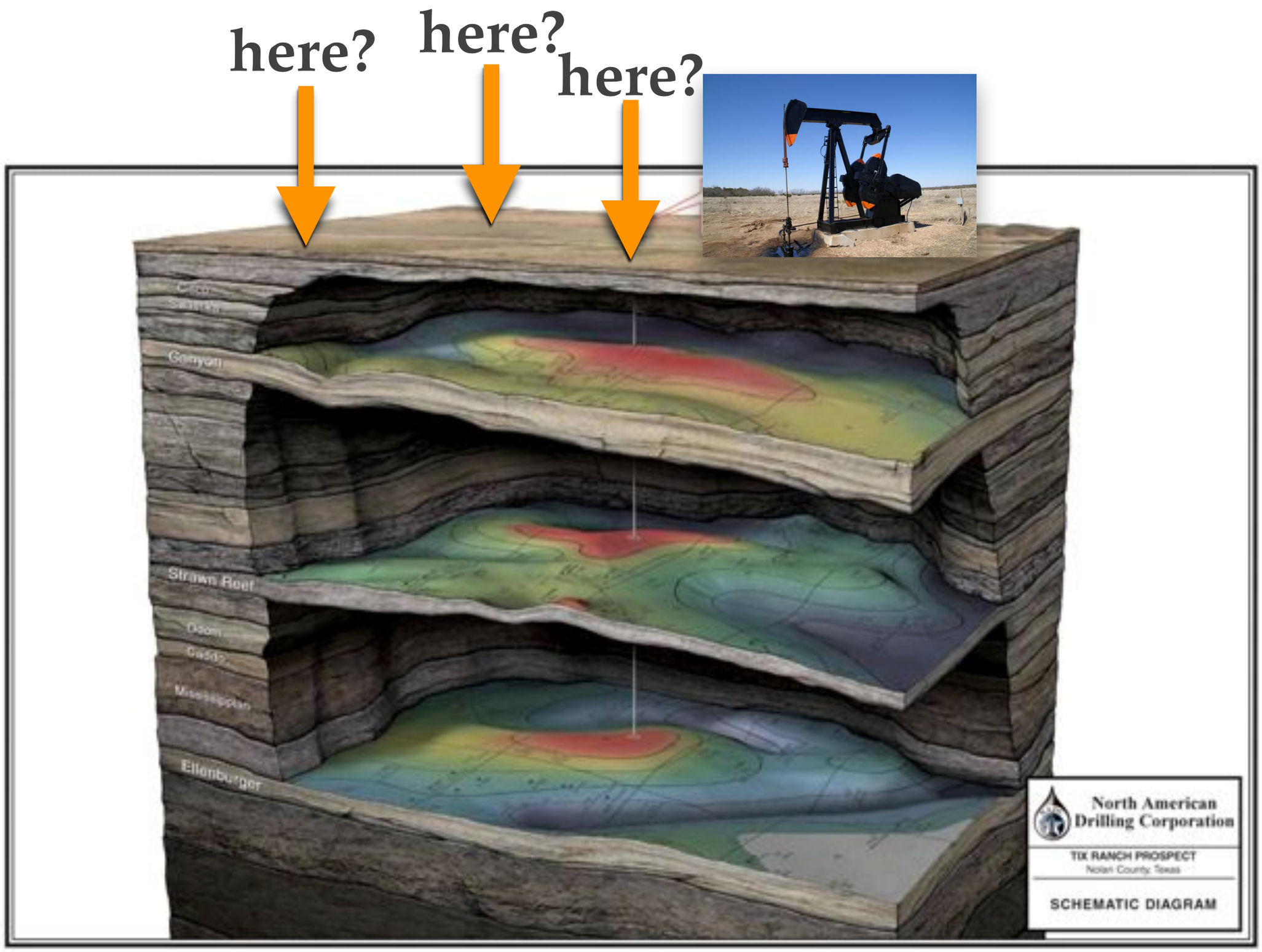
Input: position and rotation of the robotic hand

Unknown function *f*

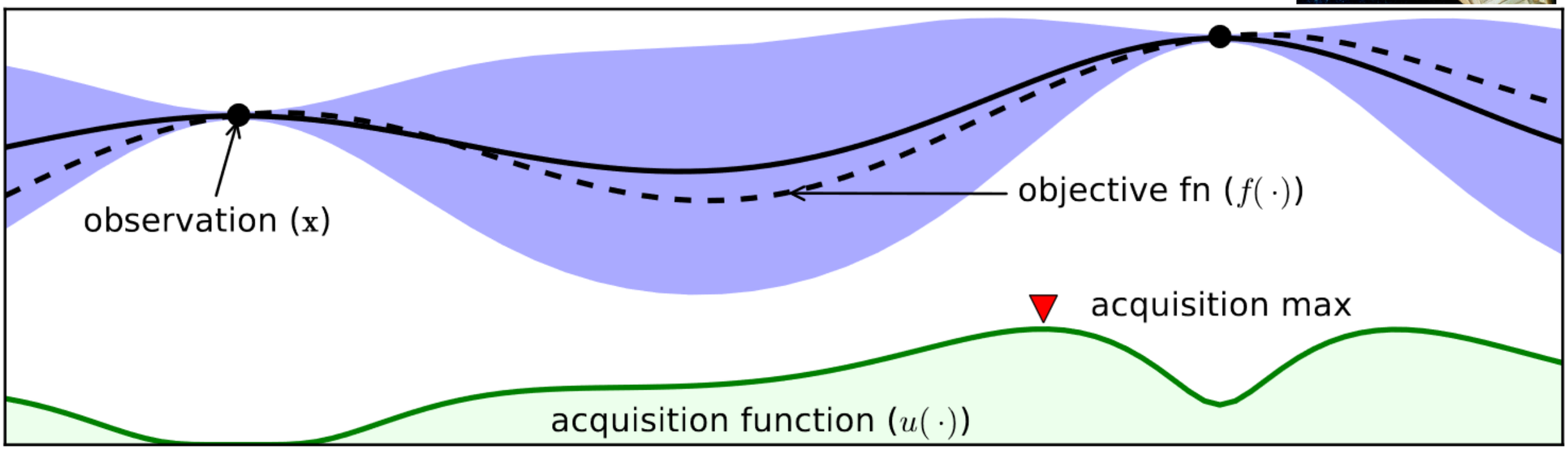


Output: stability of the simulated grasping action

Kriging: Where to drill next?



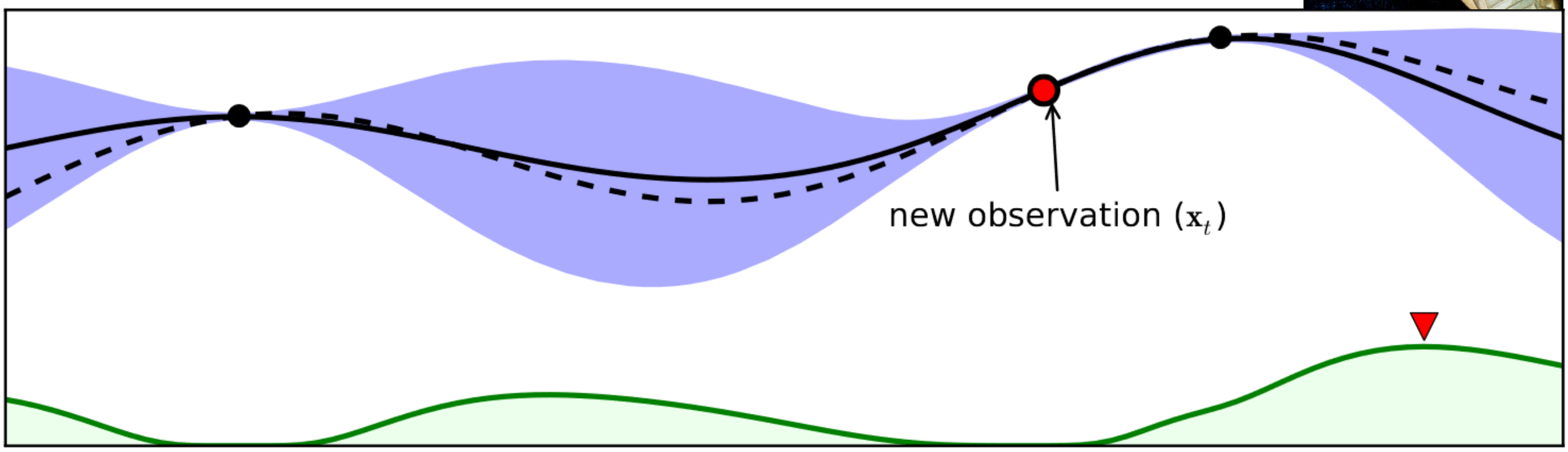
Gaussian Process



courtesy of advancedoptimizationatharvard.wordpress.com

Compute a posterior probability distribution ***p*** on all possible objective functions ***f***, given all the grasping actions that have been simulated and evaluated so far.

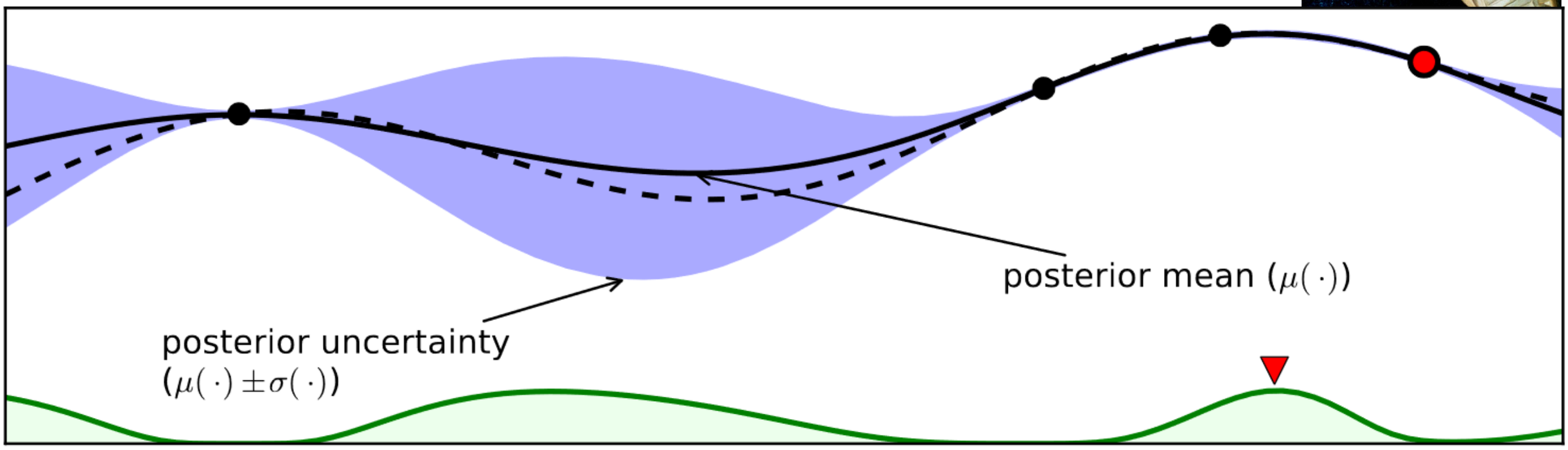
Gaussian Process



courtesy of advancedoptimizationatharvard.wordpress.com

Compute a posterior probability distribution ***p*** on all possible objective functions ***f***, given all the grasping actions that have been simulated and evaluated so far.

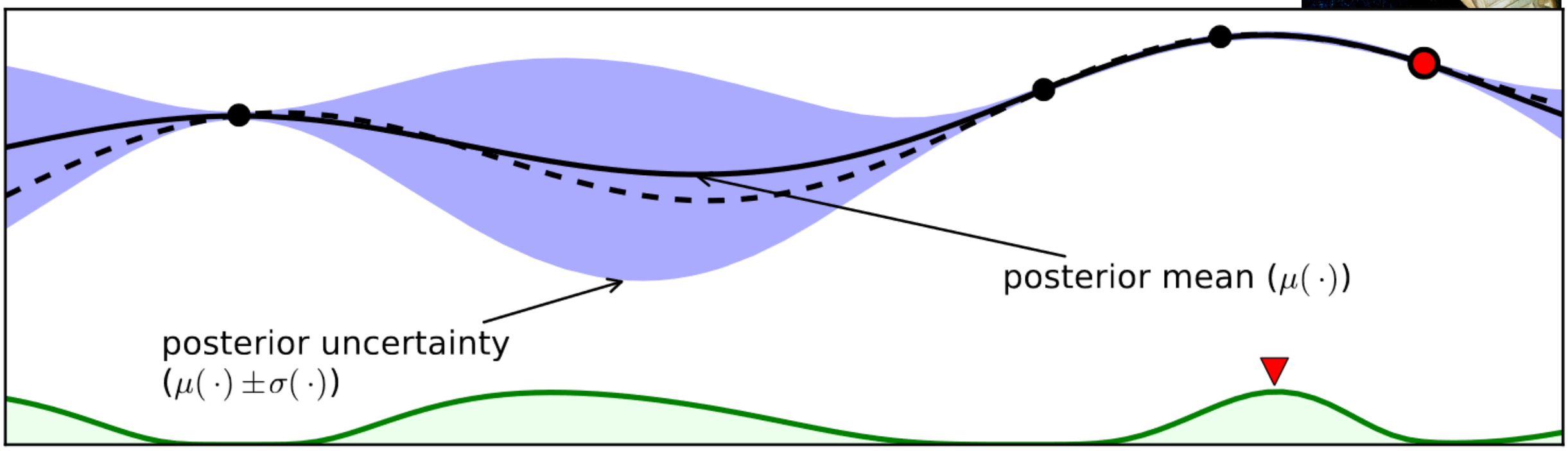
Gaussian Process



courtesy of advancedoptimizationatharvard.wordpress.com

Compute a posterior probability distribution ***p*** on all possible objective functions ***f***, given all the grasping actions that have been simulated and evaluated so far.

Gaussian Process



How should we choose the next point to evaluate?

Greedy Entropy Search

Compute a distribution P_{max} on the optimal action x:

$$P_{max}(x) \triangleq P\Big(x = \arg\max_{\tilde{x} \in \mathbb{R}^n} f(\tilde{x})\Big)$$
$$= \int_{f:\mathbb{R}^n \to \mathbb{R}} p(f) \Pi_{\tilde{x} \in \mathbb{R}^n - \{x\}} \Theta\Big(f(x) - f(\tilde{x})\Big) \mathrm{d}f$$

Heaviside step function

Greedy Entropy Search

Compute a distribution P_{max} on the optimal action x:

$$P_{max}(x) \triangleq P\Big(x = \arg\max_{\tilde{x} \in \mathbb{R}^n} f(\tilde{x})\Big)$$
$$= \int_{f:\mathbb{R}^n \to \mathbb{R}} p(f) \Pi_{\tilde{x} \in \mathbb{R}^n - \{x\}} \Theta\Big(f(x) - f(\tilde{x})\Big) \mathrm{d}f$$

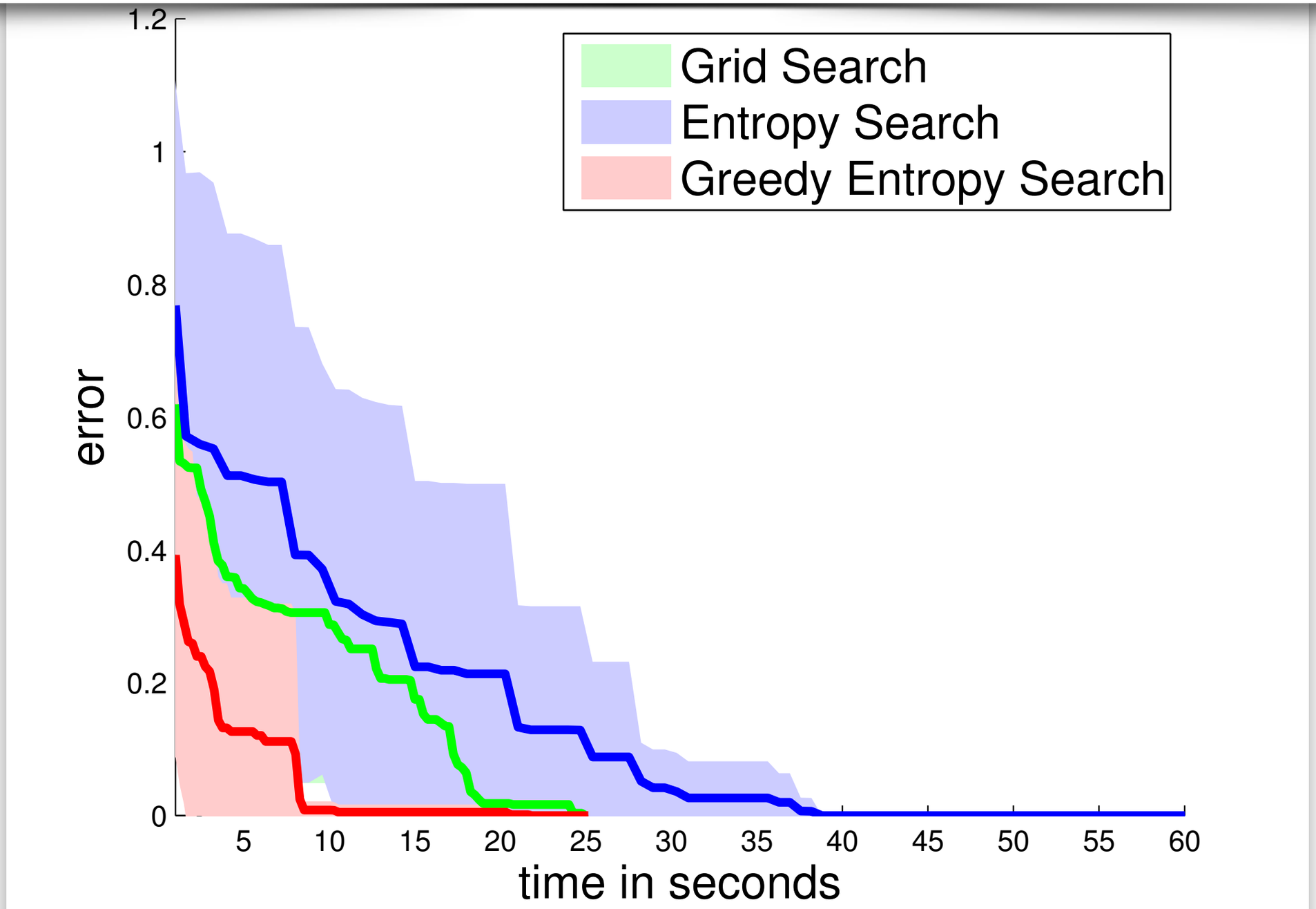
Heaviside step function

The next action x to evaluate is the one that contributes the most to the entropy of P_{max}, i.e. the one with that maximizes

$$-P_{max}(x) \log\Big(P_{max}(x)\Big)$$

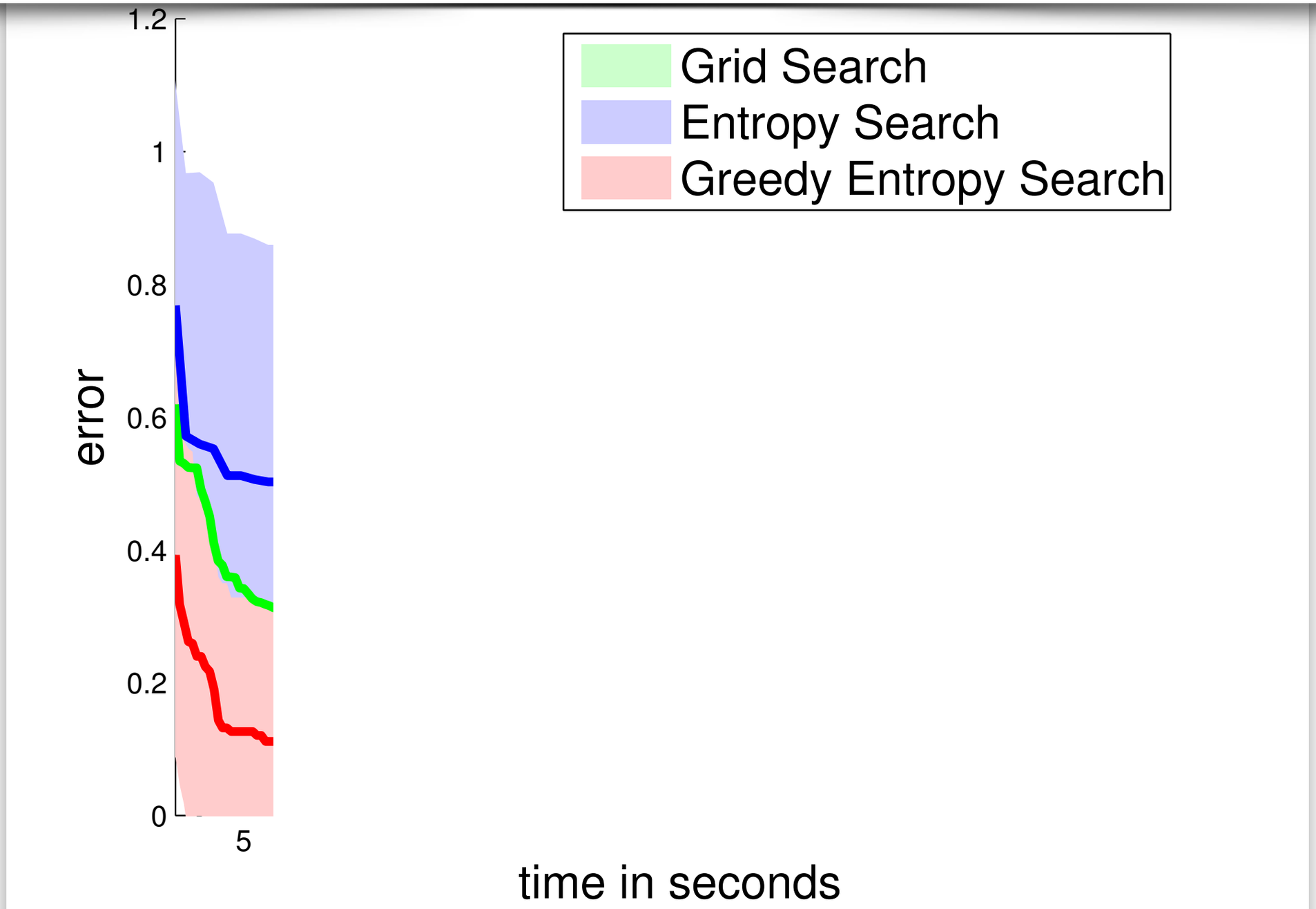
Anytime Optimization

Error = (value of the **actual best action**) – (value of the **best action found**)



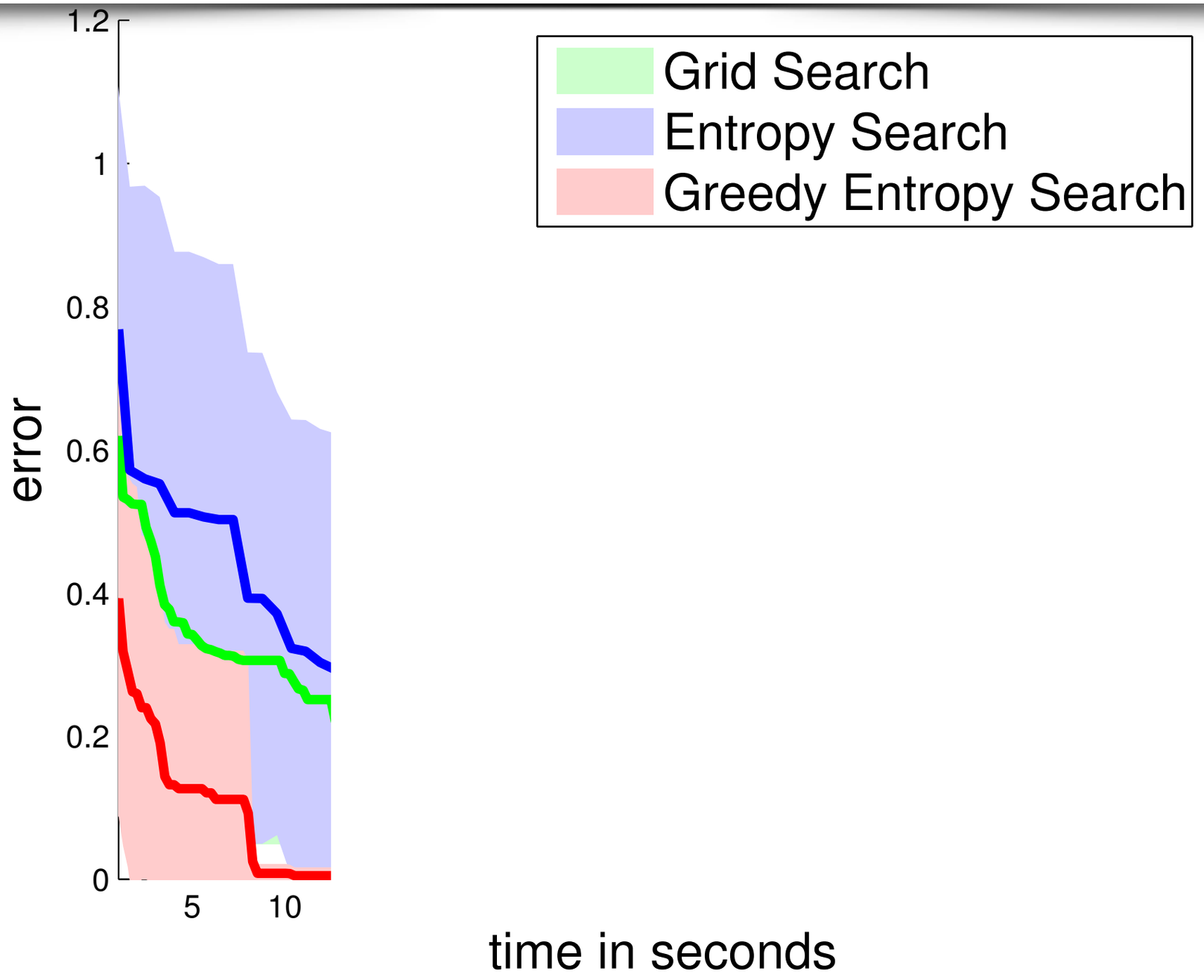
Anytime Optimization

Error = (value of the **actual best action**) – (value of the **best action found**)



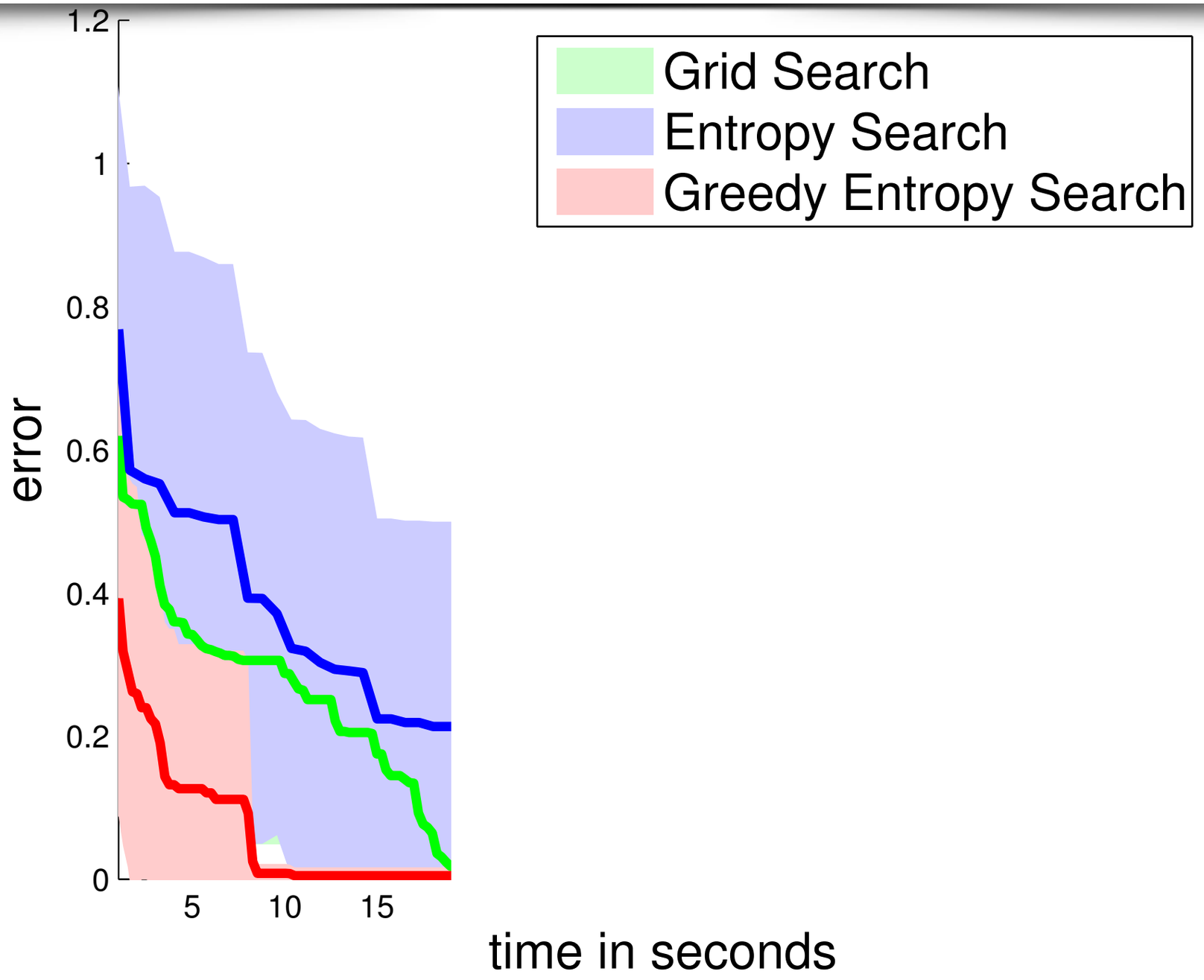
Anytime Optimization

Error = (value of the **actual best action**) – (value of the **best action found**)



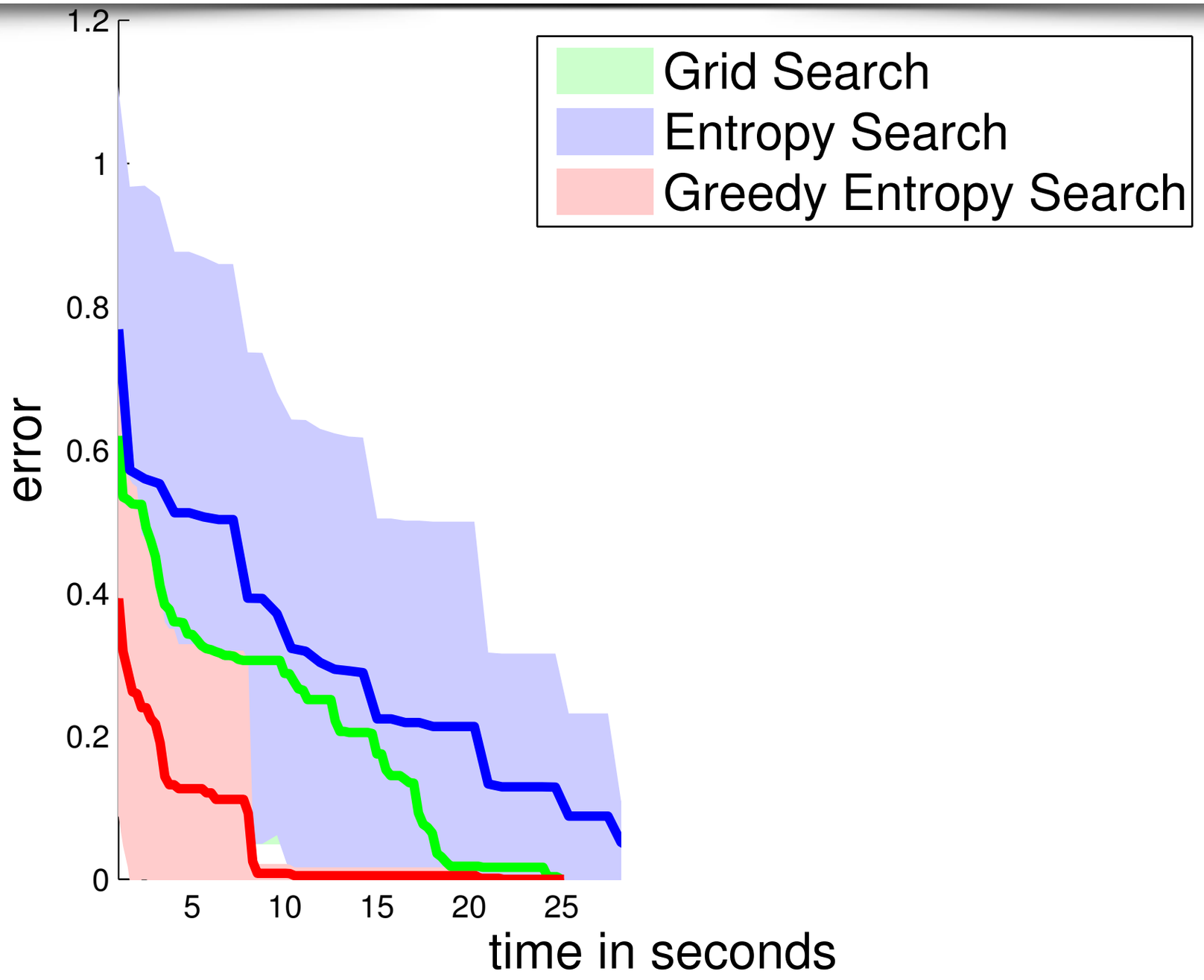
Anytime Optimization

Error = (value of the **actual best action**) – (value of the **best action found**)



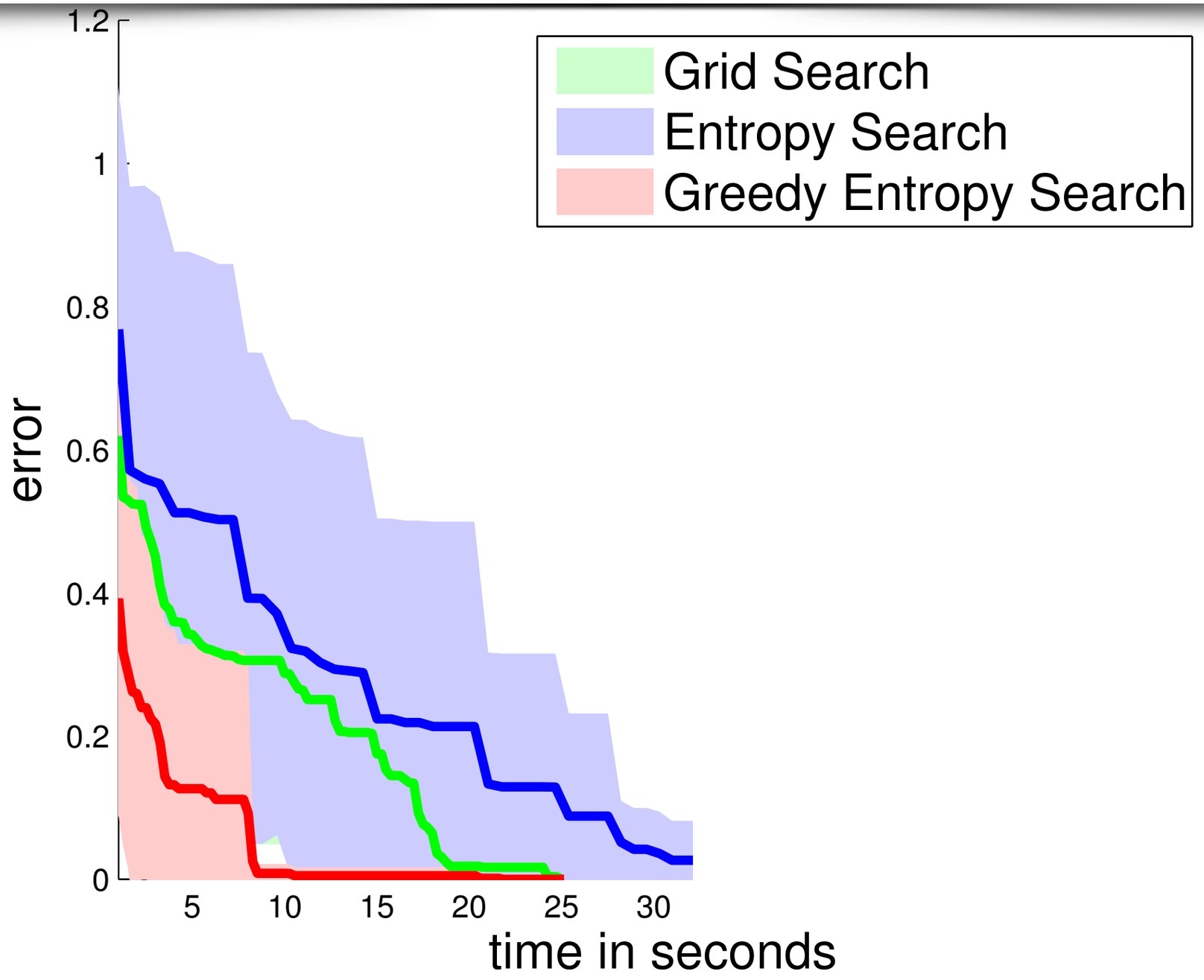
Anytime Optimization

Error = (value of the **actual best action**) – (value of the **best action found**)



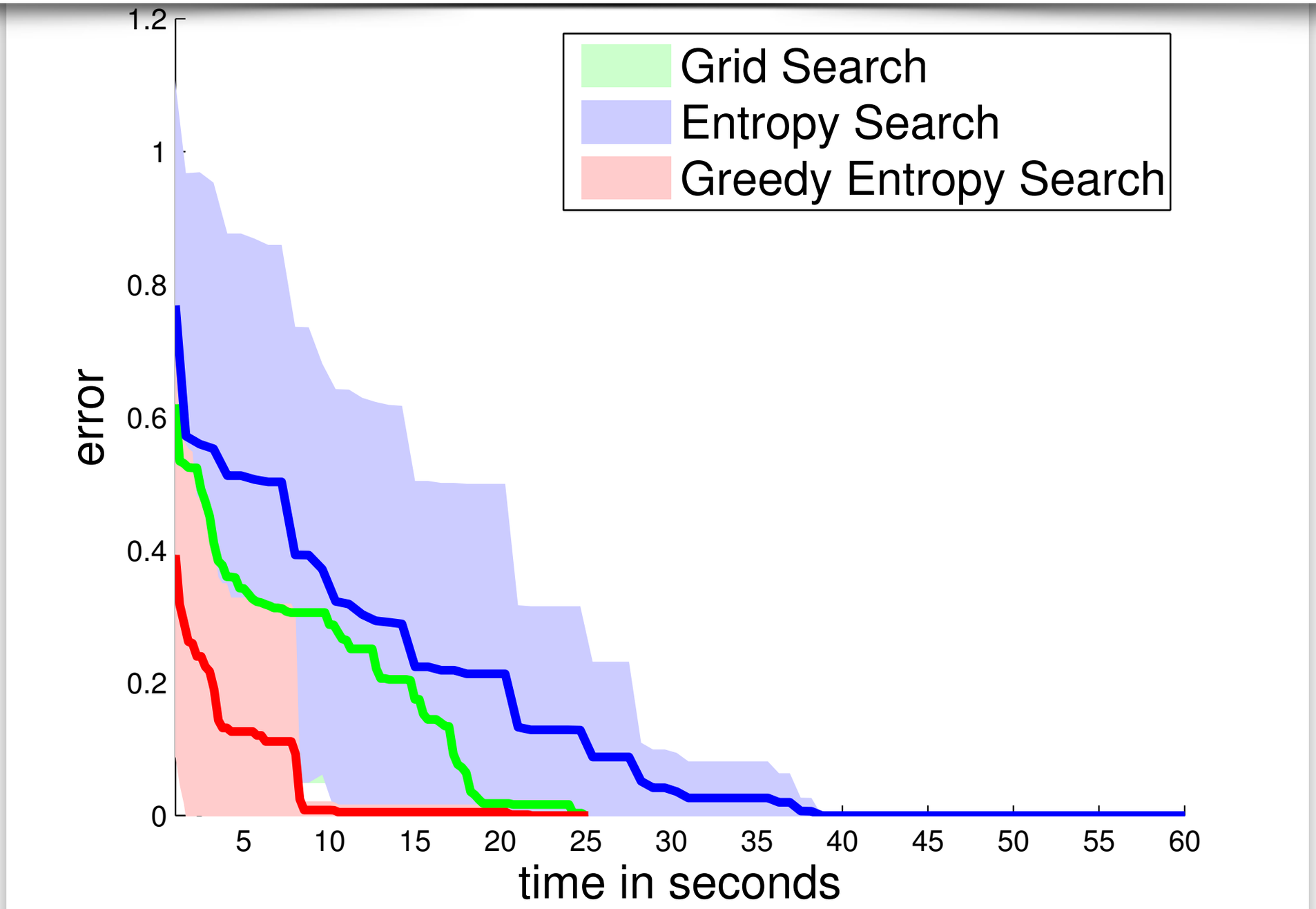
Anytime Optimization

Error = (value of the **actual best action**) – (value of the **best action found**)



Anytime Optimization

Error = (value of the **actual best action**) – (value of the **best action found**)



Results of Experiments on Grasping Rocks

34% success rate without learning, using only the centers of the rocks, and without a time budget

A. Boularias *et al.* (2014) in *Conference of the Association for the Advancement of Artificial Intelligence (AAAI)*

Results of Experiments on Grasping Rocks

34% success rate without learning, using only the centers of the rocks, and without a time budget

56% success rate with learning, Bayesian optimization, and a time budget of **1 second**

A. Boularias *et al.* (2014) in *Conference of the Association for the Advancement of Artificial Intelligence (AAAI)*

Results of Experiments on Grasping Rocks

34% success rate without learning, using only the centers of the rocks, and without a time budget

56% success rate with learning, Bayesian optimization, and a time budget of **1 second**

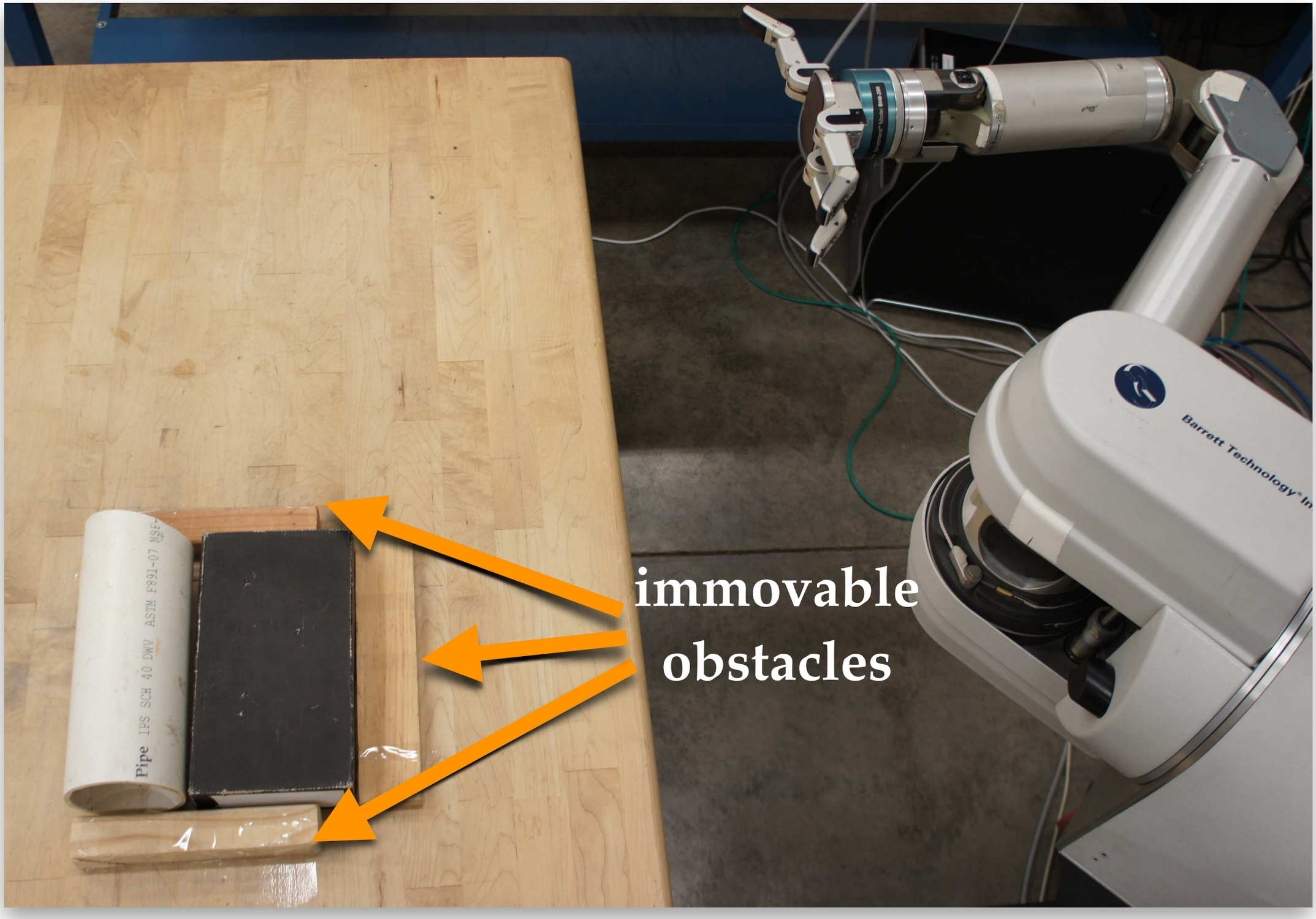
74% success rate with learning, Bayesian optimization, and a time budget of **5 seconds**

A. Boularias *et al.* (2014) in *Conference of the Association for the Advancement of Artificial Intelligence (AAAI)*

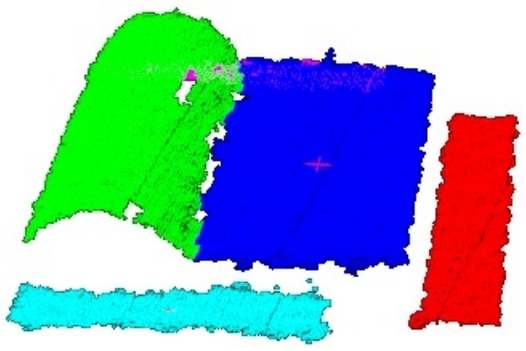
Outline

1. Overview
2. Optimal control
3. Inverse optimal control
4. Grasping
5. **Manipulation**
6. Navigation

Pushing objects is necessary for grasping in confined environments

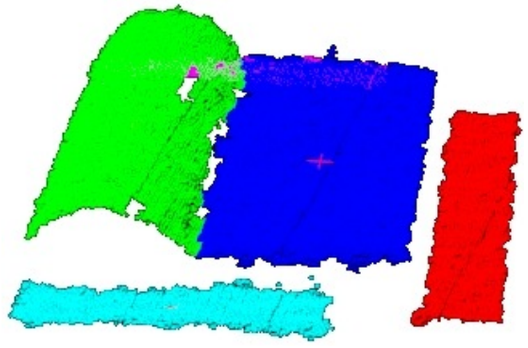


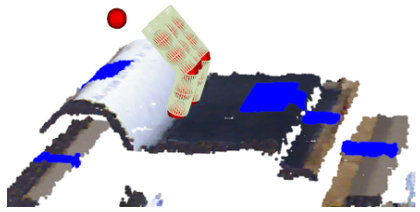
Get an image of the scene from an RGB-D sensor

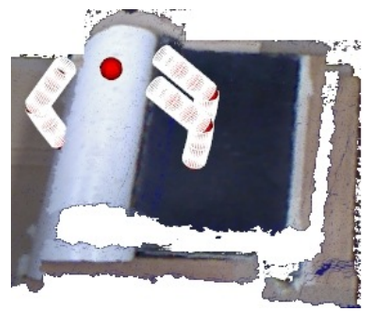


Get an image of the scene from an RGB-D sensor

Segment the scene image into objects



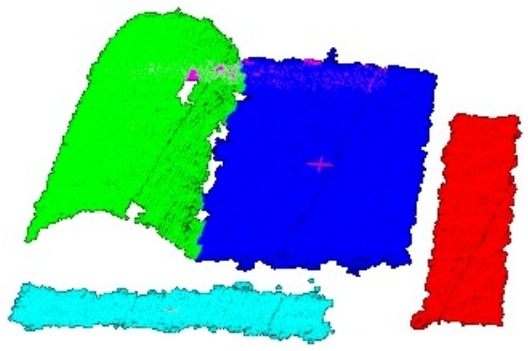


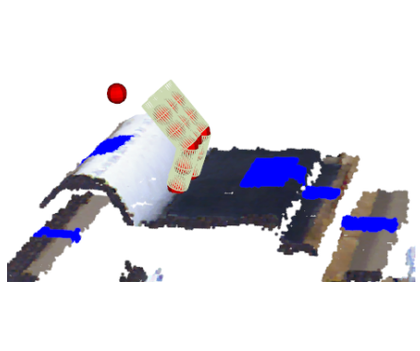


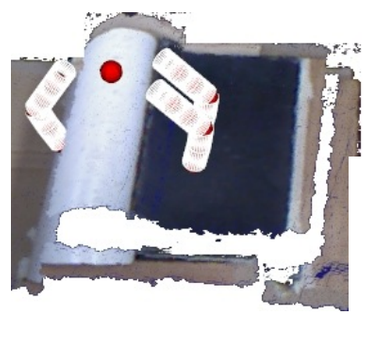
Get an image of the scene from an RGB-D sensor

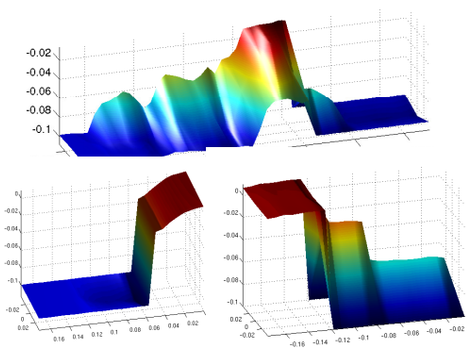
Segment the scene image into objects

Sample a number of grasping and pushing actions for each object







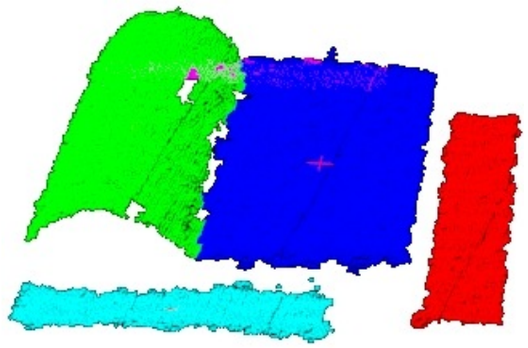


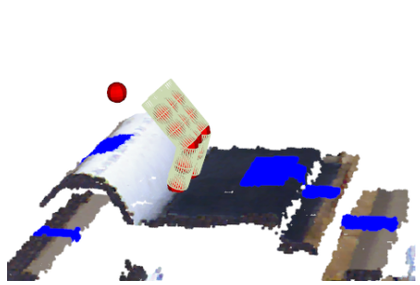
Get an image of the scene from an RGB-D sensor

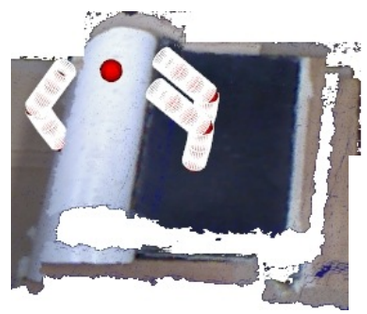
Segment the scene image into objects

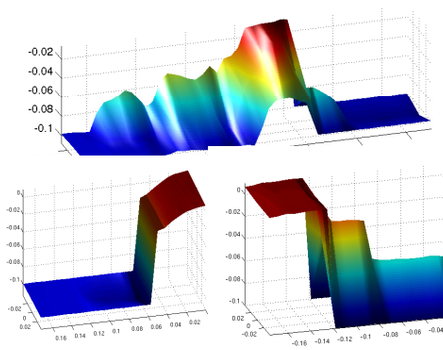
Sample a number of grasping and pushing actions for each object

Extract the features of each sampled action









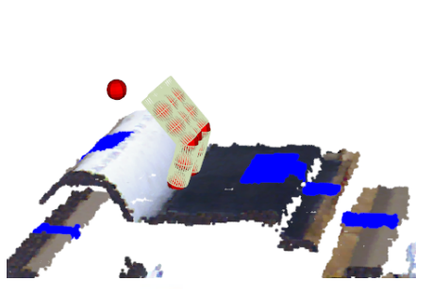
Get an image of the scene from an RGB-D sensor

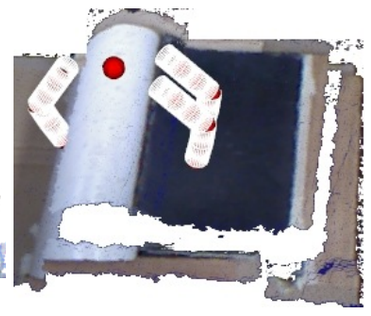
Segment the scene image into objects

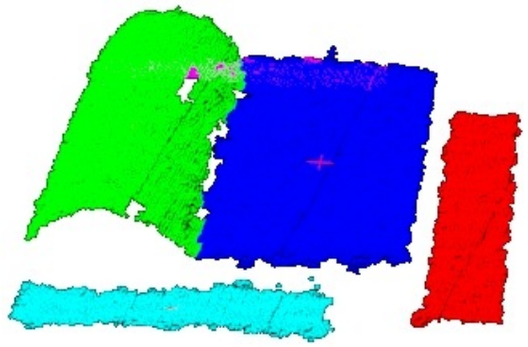
Sample a number of grasping and pushing actions for each object

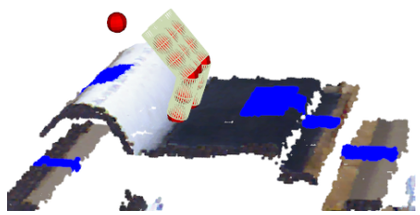
Extract the features of each sampled action

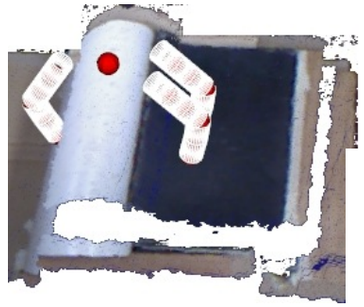
Predict the value of each sampled action using the values of the actions executed in previous states

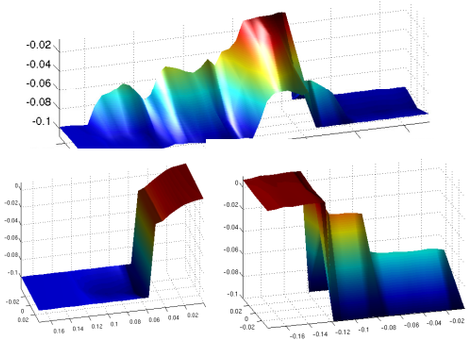












Get an image of the scene from an RGB-D sensor

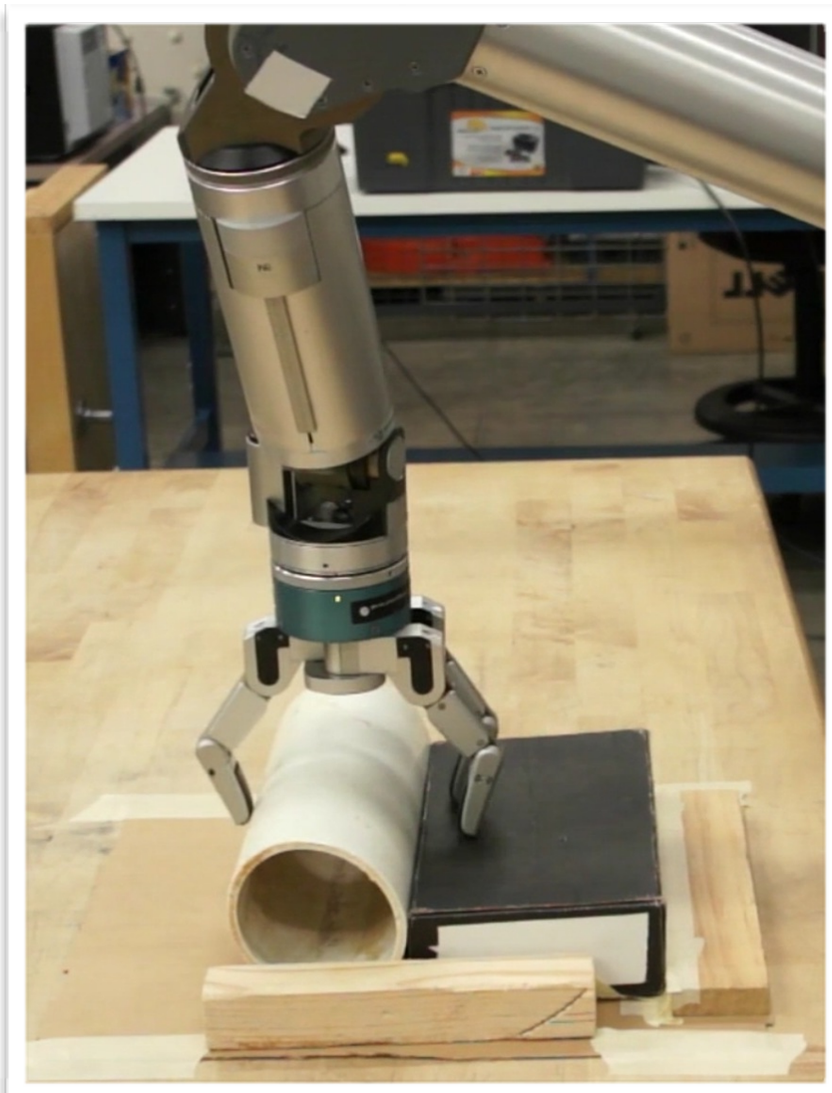
Segment the scene image into objects

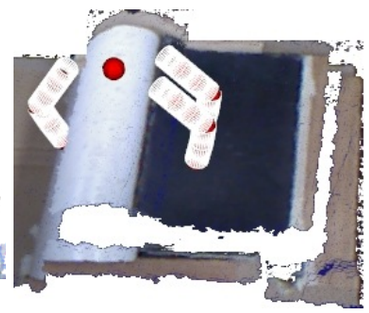
Sample a number of grasping and pushing actions for each object

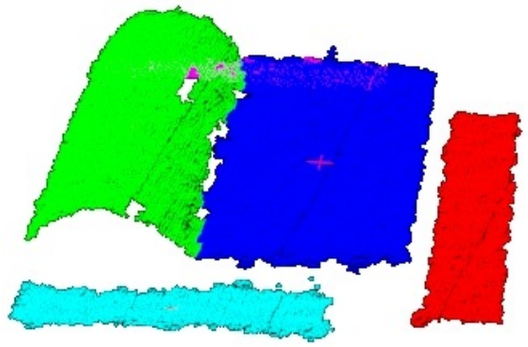
Extract the features of each sampled action

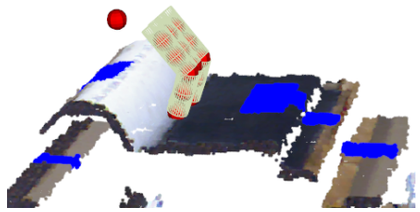
Predict the value of each sampled action using the values of the actions executed in previous states

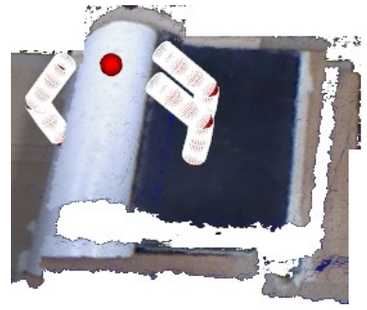
Execute the action with the highest Upper Confidence Bound (UCB), and obtain a binary reward based on the joint angles of the fingers

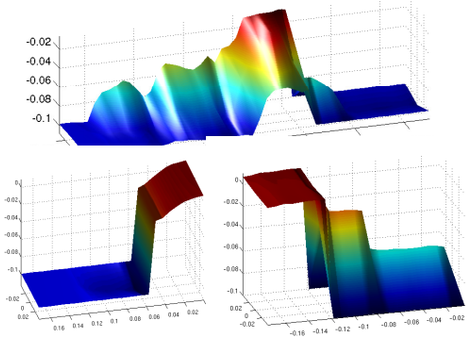












Get an image of the scene from an RGB-D sensor

Segment the scene image into objects

Re-compute the value of every previous state (scene) based on the value of the best action in the next state

Re-evaluate the actions sampled in every state

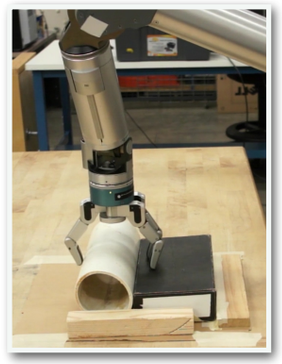
Sample a number of grasping and pushing actions for each object

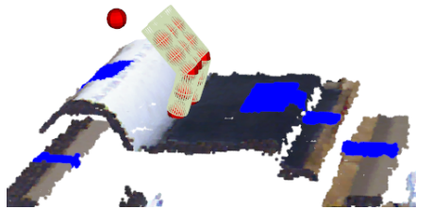
Execute the action with the highest Upper Confidence Bound (UCB), and obtain a binary reward based on the joint angles of the fingers

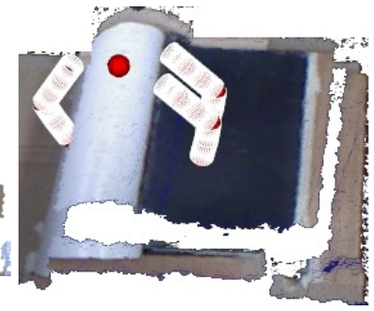
Extract the features of each sampled action

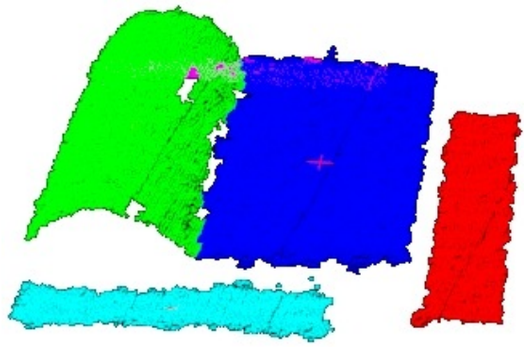
Predict the value of each sampled action using the values of the actions executed in previous states

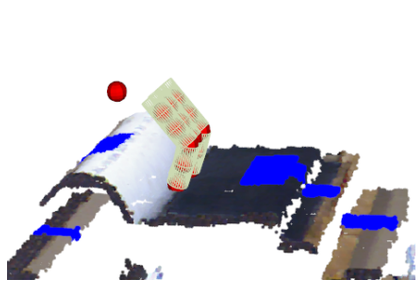
Policy Iteration (replay)

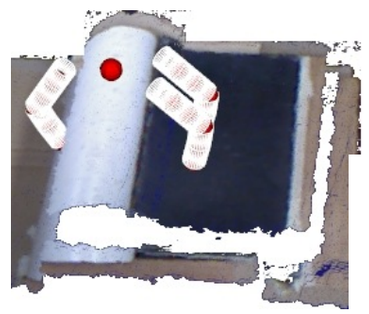


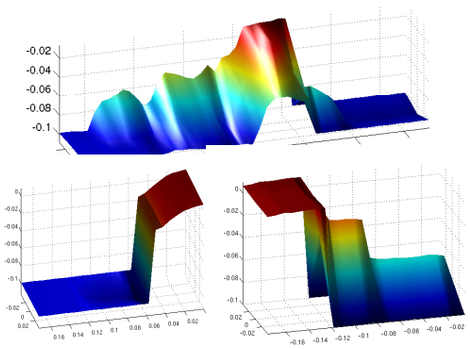












Get an image of the scene from an RGB-D sensor

Segment the scene image into objects

Sample a number of grasping and pushing actions for each object

Extract the features of each sampled action

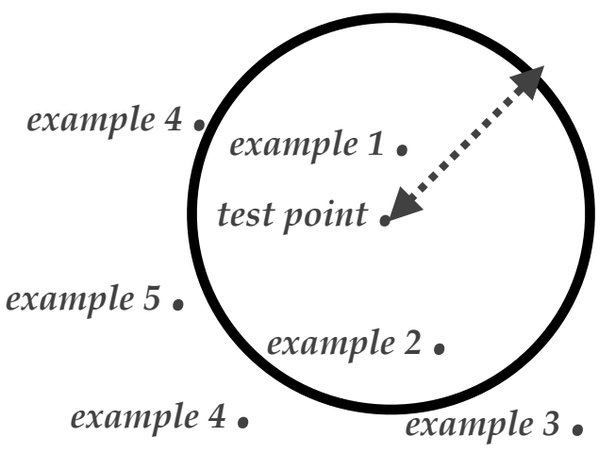
Predict the value of each sampled action using the values of the actions executed in previous states

Execute the action with the highest Upper Confidence Bound (UCB), and obtain a binary reward based on the joint angles of the fingers

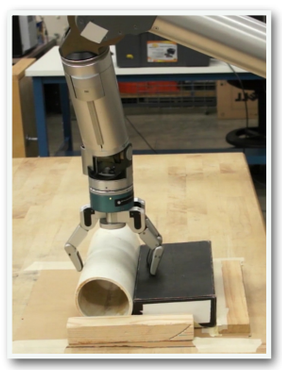
Re-compute the value of every previous state (scene) based on the value of the best action in the next state

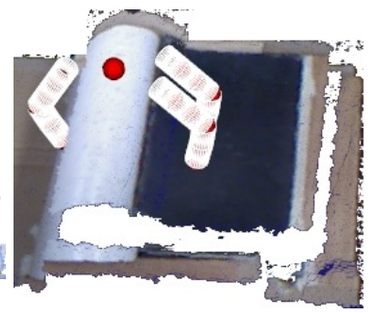
Re-evaluate the actions sampled in every state

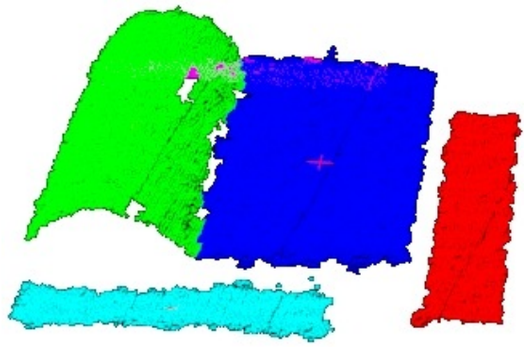
Tune the hyper-parameters (kernel bandwidths) by cross-validation

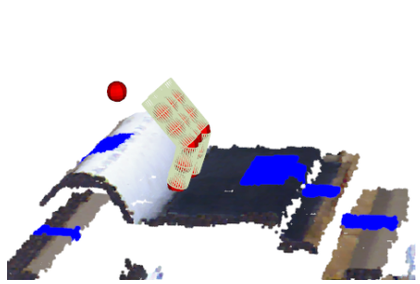


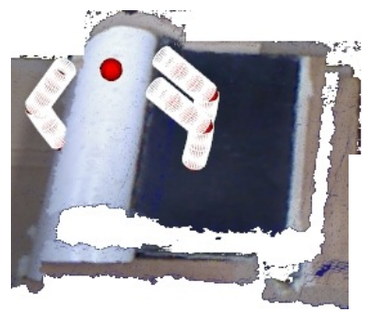
Policy Iteration (replay)

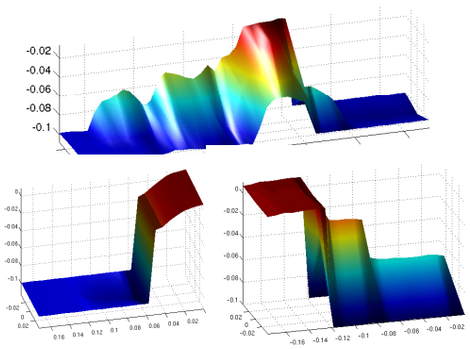












Get an image of the scene from an RGB-D sensor

Segment the scene image into objects

Sample a number of grasping and pushing actions for each object

Extract the features of each sampled action

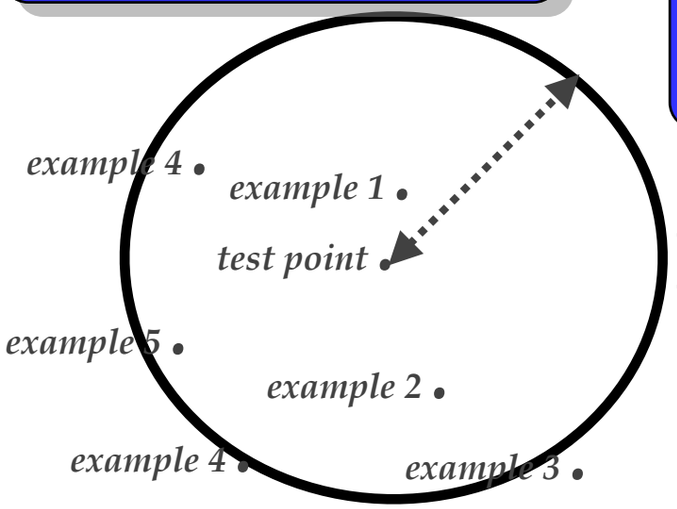
Predict the value of each sampled action using the values of the actions executed in previous states

Execute the action with the highest Upper Confidence Bound (UCB), and obtain a binary reward based on the joint angles of the fingers

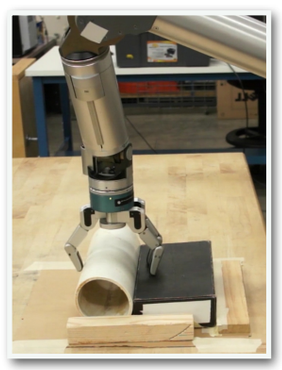
Re-compute the value of every previous state (scene) based on the value of the best action in the next state

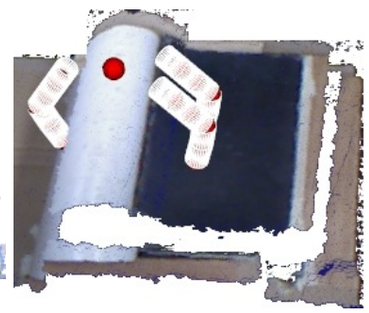
Re-evaluate the actions sampled in every state

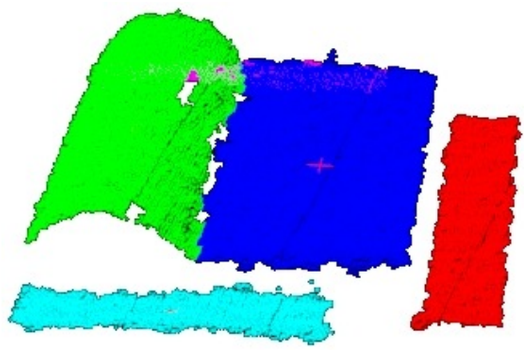
Tune the hyper-parameters (kernel bandwidths) by cross-validation

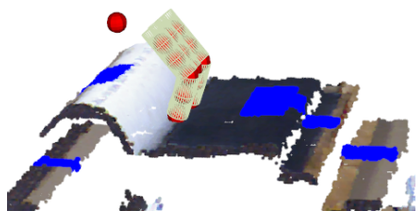


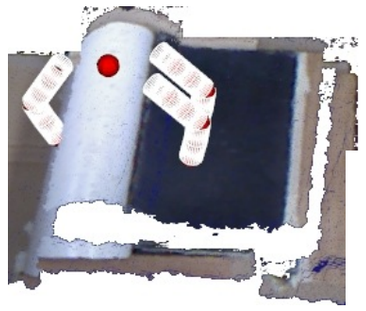
Policy Iteration (replay)

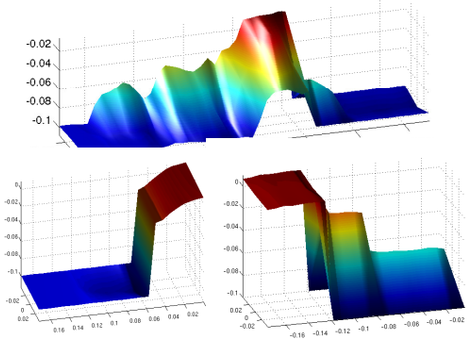












Get an image of the scene from an RGB-D sensor

Segment the scene image into objects

Sample a number of grasping and pushing actions for each object

Extract the features of each sampled action

Predict the value of each sampled action using the values of the actions executed in previous states

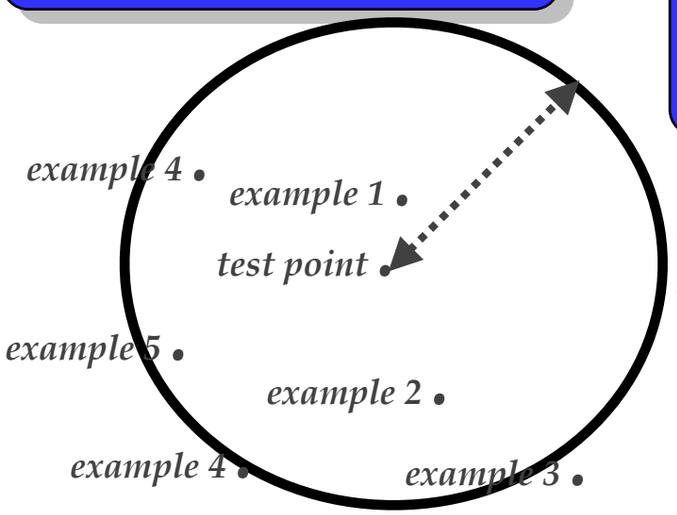
Execute the action with the highest Upper Confidence Bound (UCB), and obtain a binary reward based on the joint angles of the fingers

Re-compute the value of every previous state (scene) based on the value of the best action in the next state

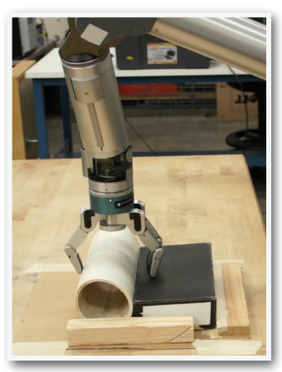
Re-evaluate the actions sampled in every state

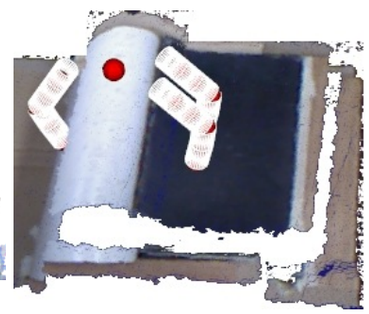
Tune the hyper-parameters (kernel bandwidths) by cross-validation

Repeat

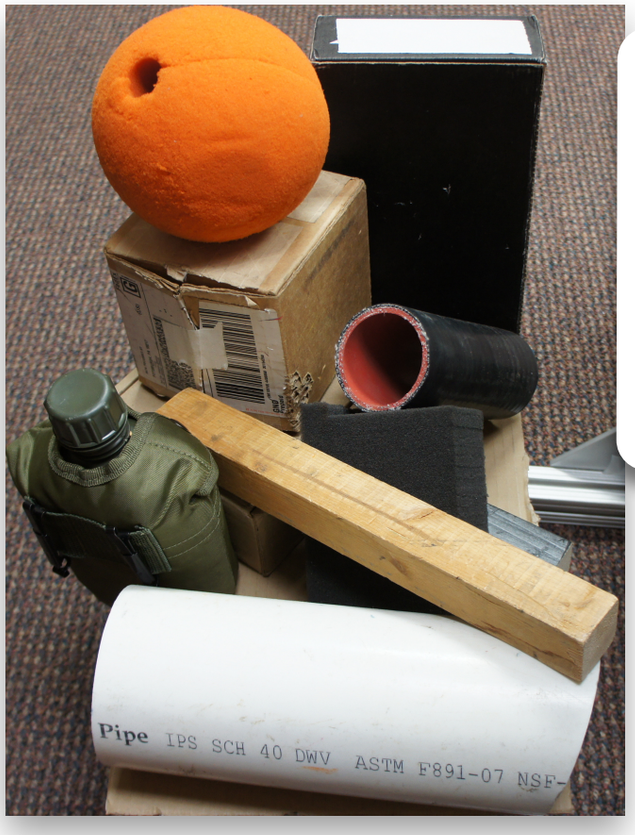


Policy Iteration (replay)





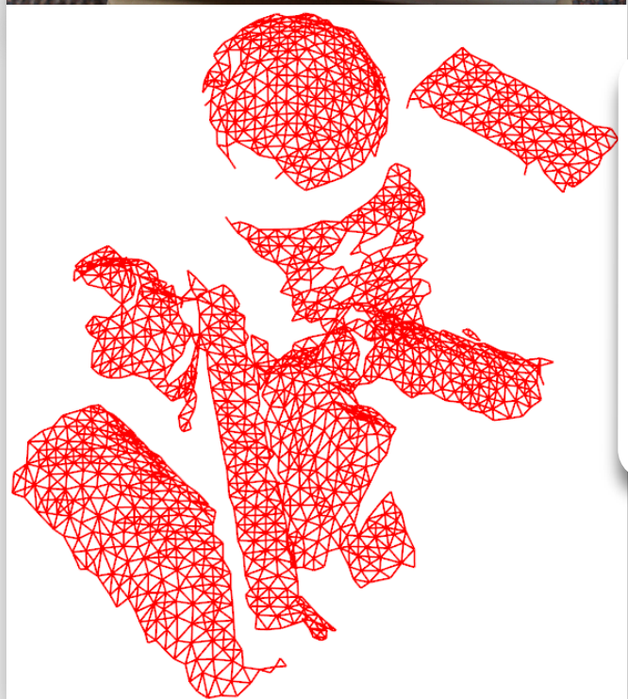
Segmenting a pile of unknown objects



1. **Detect and remove the support surface (RANSAC)**

Segmenting a pile of unknown objects

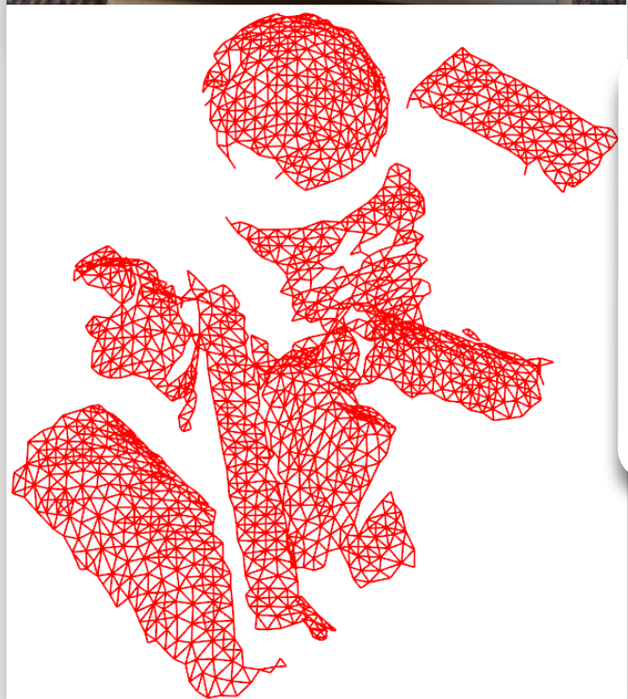
1. Detect and remove the support surface (RANSAC)



2. Cluster voxels into **supervoxels** **(k-means)**

Segmenting a pile of unknown objects

1. Detect and remove the support surface (RANSAC)



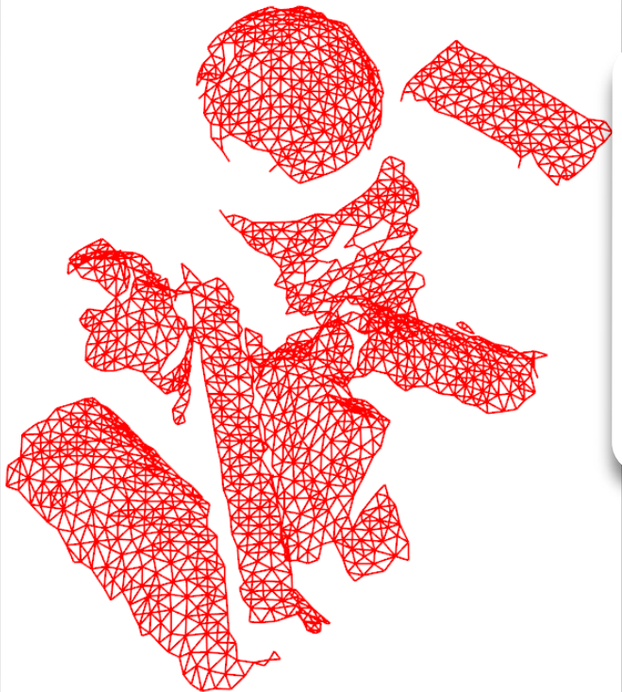
2. Cluster voxels into **supervoxels** **(k-means)**

3. Cluster supervoxels into **facets** **(mean-shift)**

Segmenting a pile of unknown objects

1. Detect and remove the support surface (RANSAC)

4. Cluster facets into **objects** (**spectral clustering**)

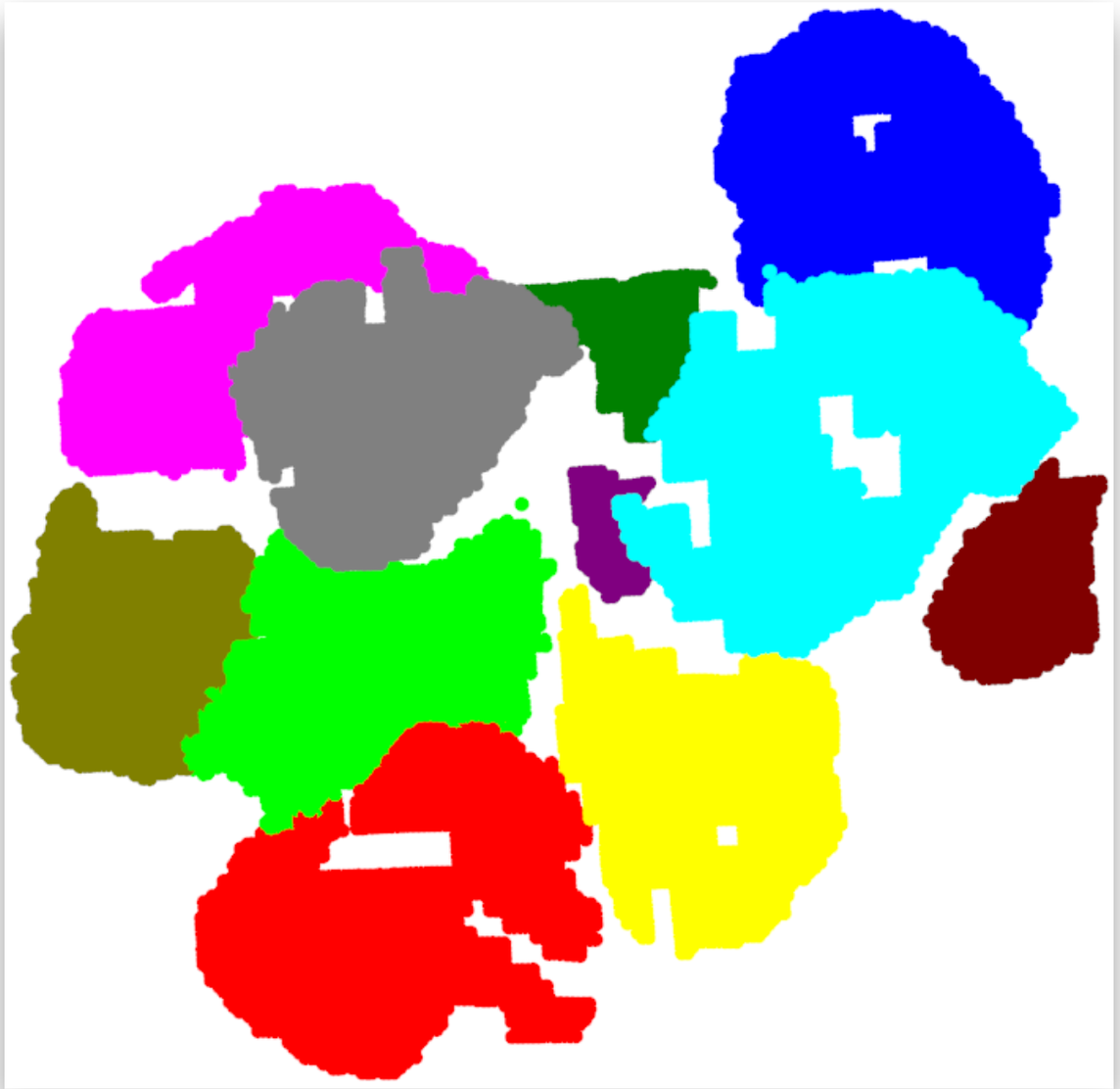


2. Cluster voxels into **supervoxels** (**k-means**)

3. Cluster supervoxels into **facets** (**mean-shift**)

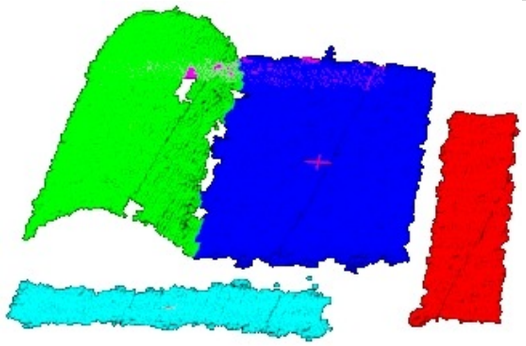
Natural Irregular Objects

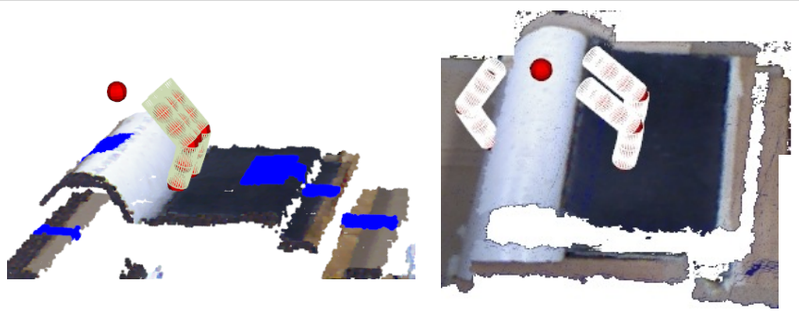
Pile of rocks

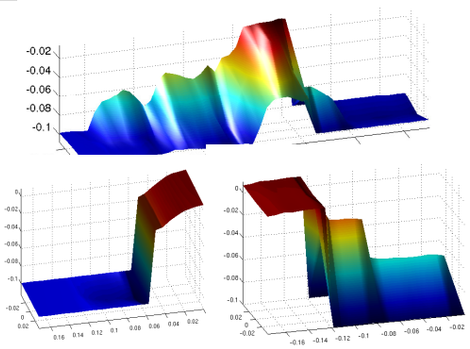


Segmented image

Extracting Features







Get an image of the scene from an RGB-D sensor

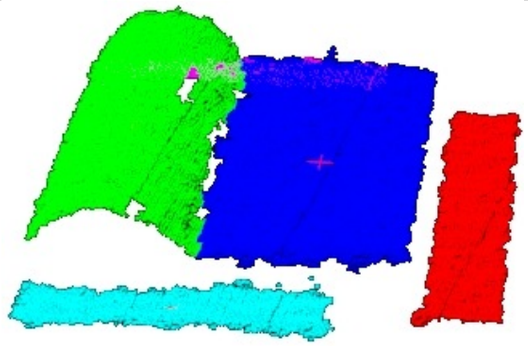
Segment the scene image into objects

Sample a number of grasping and pushing actions for each object

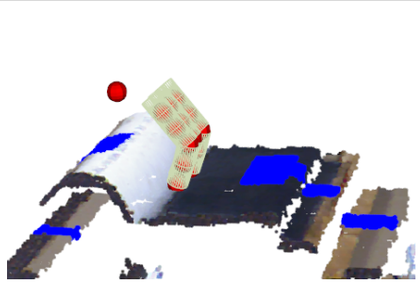
Extract the features of each sampled action

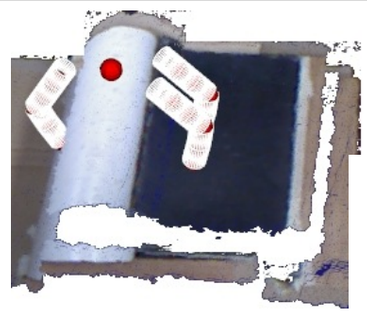
Action Evaluation

Get an image of the scene from an RGB-D sensor

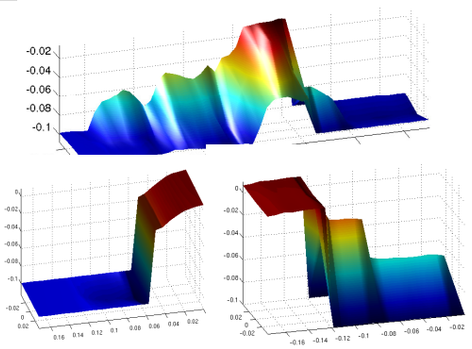


Segment the scene image into objects



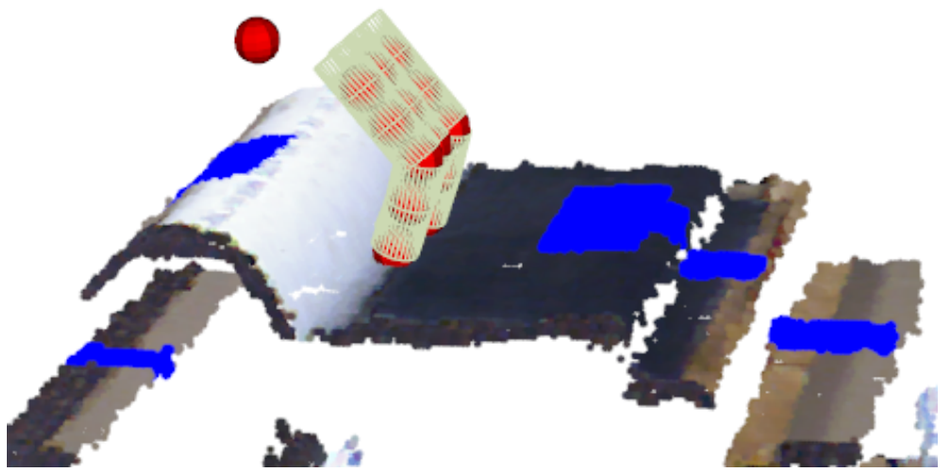


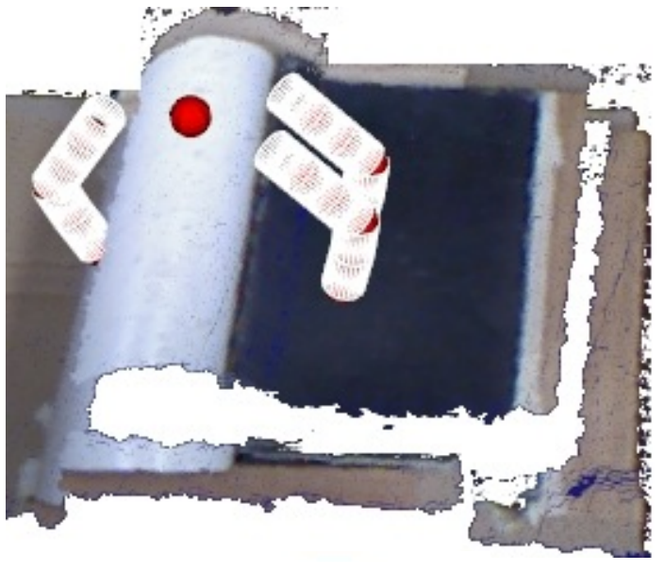
Sample a number of grasping and pushing actions for each object



Extract the features of each sampled action

Predict the value of each sampled action using the values of the actions executed in previous states





The clutter clearing task is formalized as a **Markov Decision Process**

State = 3D image of the scene

Action = Parameters of a grasp or a push

Reward = 1 for each successful grasp,

0 for anything else.

The value of action ***a*** in state ***s*** is predicted as

$$\hat{Q}_\pi(s,a) = \frac{\sum_{i=0}^{t-1} K\big((s_i,a_i),(s,a)\big)\hat{V}_\pi(s_i)}{\sum_{i=0}^{t-1} K\big((s_i,a_i),(s,a)\big)}.$$

Empirical value

Similarity measure (Kernel)

Data: state (image) and action at time i

Exploration versus Exploitation

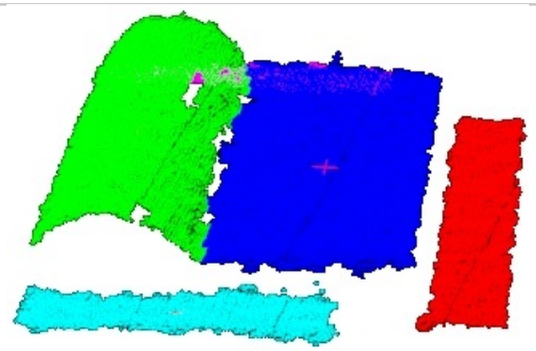
In state s_t at time t, execute action a that maximizes:

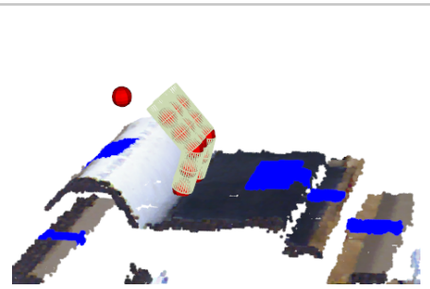
$$\hat{Q}_{\pi^*}(s_t, a) + \alpha \sqrt{\frac{2 \ln t}{\sum_{i=0}^{t-1} K\big((s_i, a_i), (s_t, a)\big)}}.$$

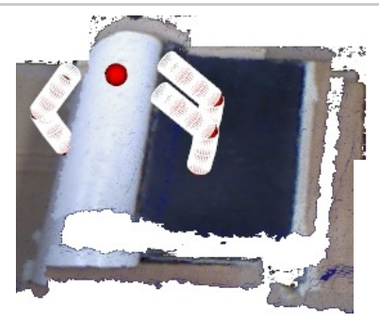
Predicted Value
(for exploitation)

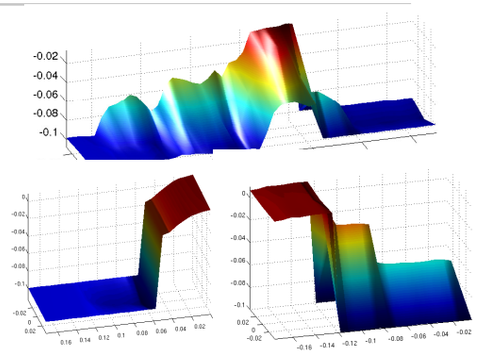
Novelty
(for exploration)

Policy Iteration









Get an image of the scene from an RGB-D sensor

Segment the scene image into objects

Sample a number of grasping and pushing actions for each object

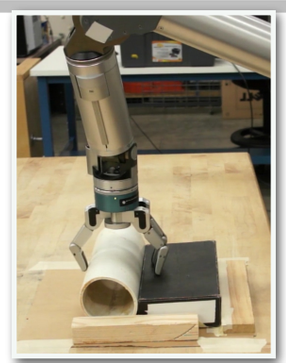
Extract the features of each sampled action

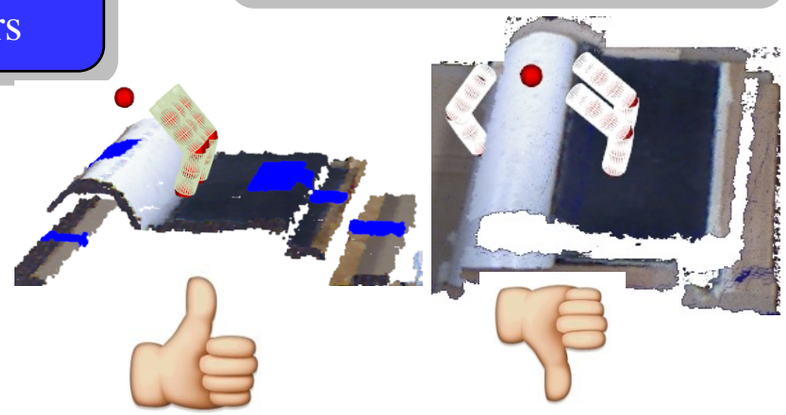
Predict the value of each sampled action using the values of the actions executed in previous states

Execute the action with the highest Upper Confidence Bound (UCB), and obtain a binary reward based on the joint angles of the fingers

Re-compute the value of every previous state (scene) based on the value of the best action in the next state

Re-evaluate the actions sampled in every state

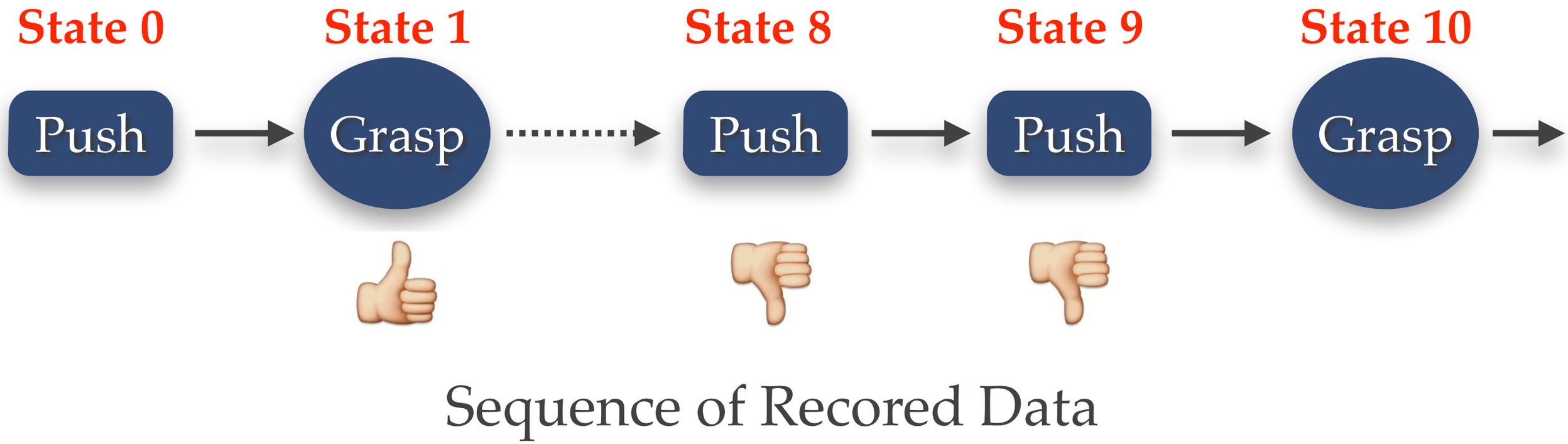




Policy Iteration (replay)

Policy Iteration (replay)

Kernel density estimation for learning the transition function between the states contained in the training data sequence

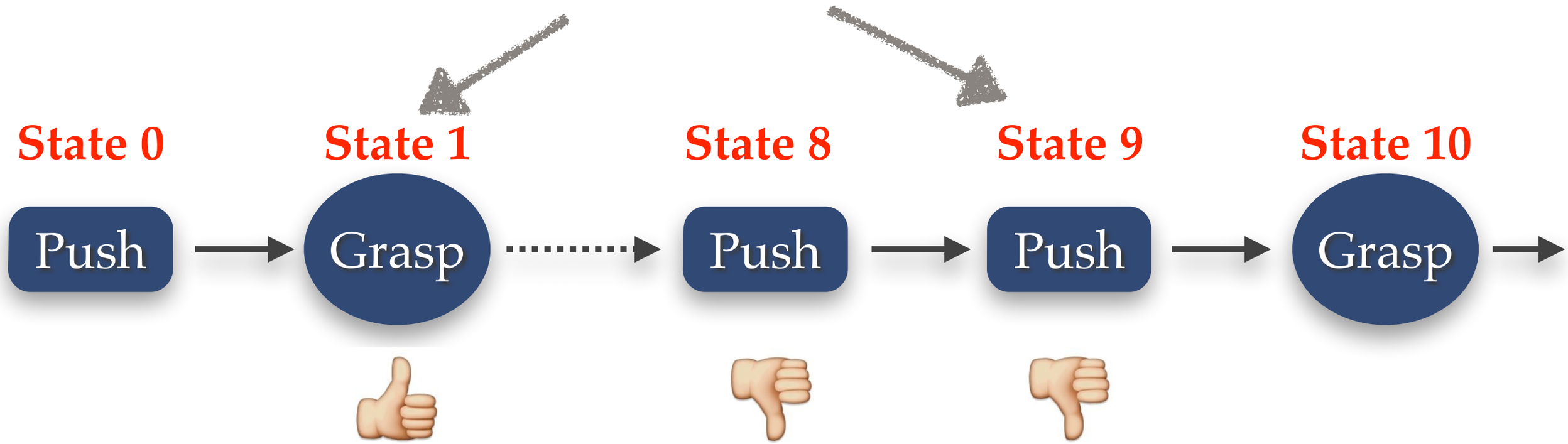


Policy Iteration (replay)

Kernel density estimation for learning the transition function between the states contained in the training data sequence

Similar states!

(State 1 ≈ State 9)



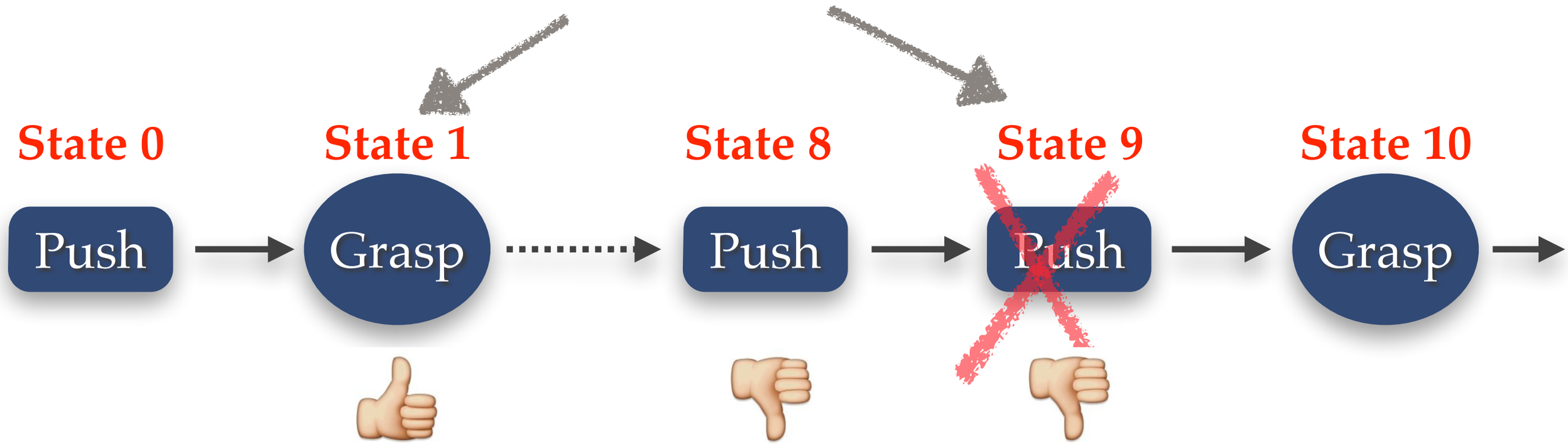
Sequence of Recored Data

Policy Iteration (replay)

Kernel density estimation for learning the transition function between the states contained in the training data sequence

Similar states!

(State 1 ≈ State 9)



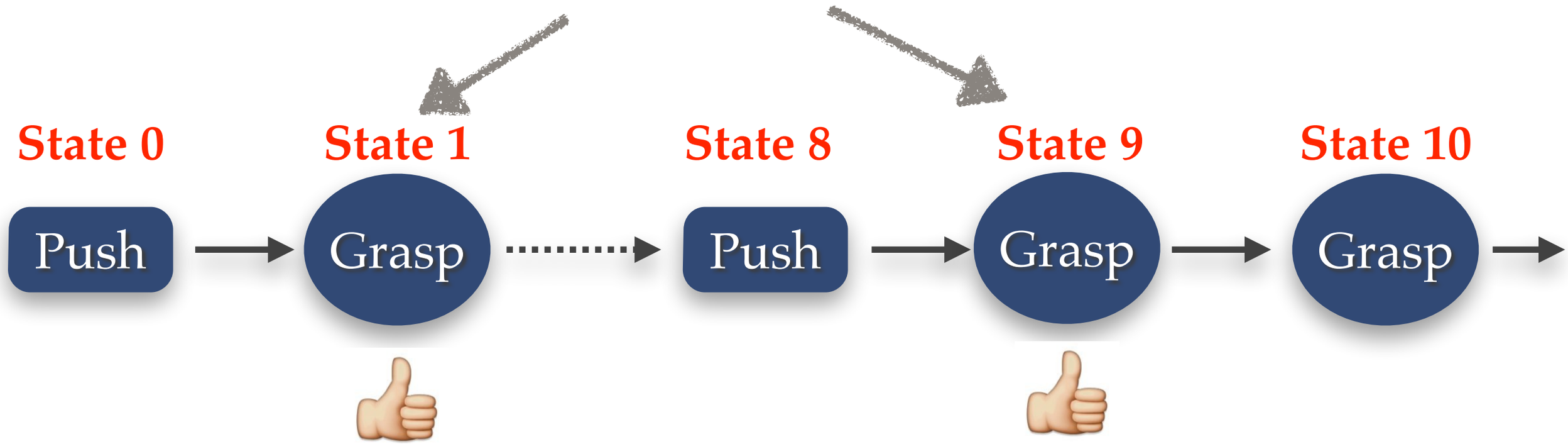
Sequence of Recored Data

Policy Iteration (replay)

Kernel density estimation for learning the transition function between the states contained in the training data sequence

Similar states!

(State 1 ≈ State 9)



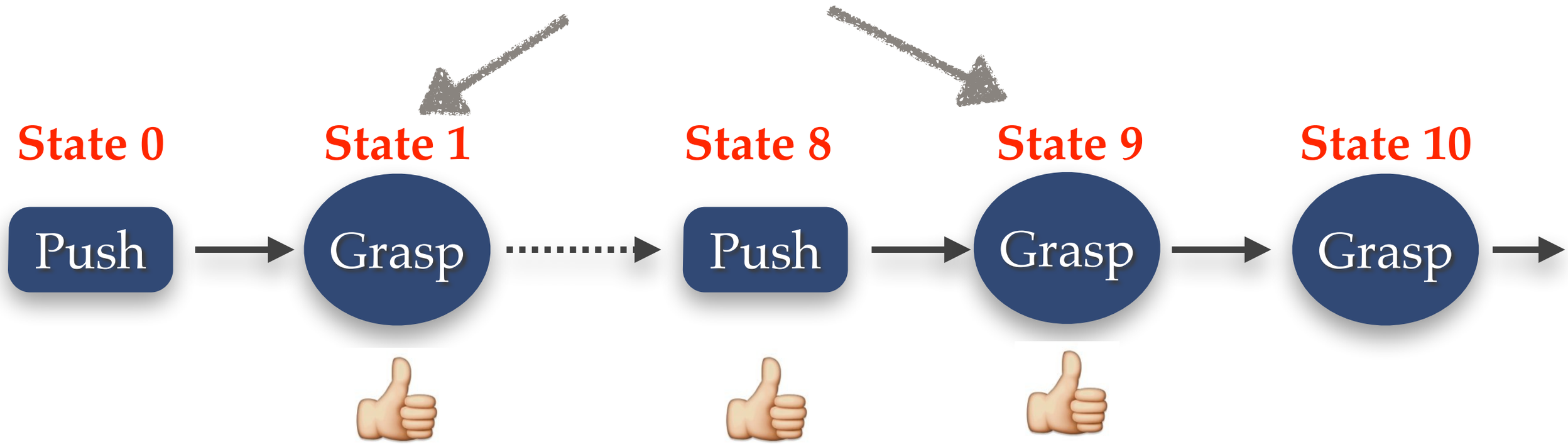
Sequence of Recored Data

Policy Iteration (replay)

Kernel density estimation for learning the transition function between the states contained in the training data sequence

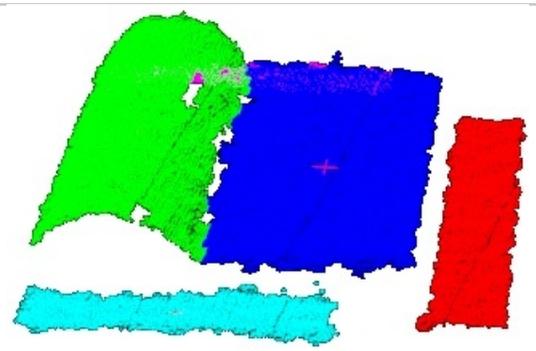
Similar states!

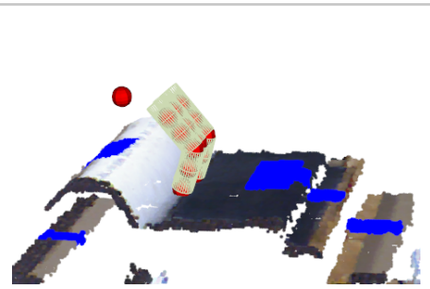
(State 1 ≈ State 9)

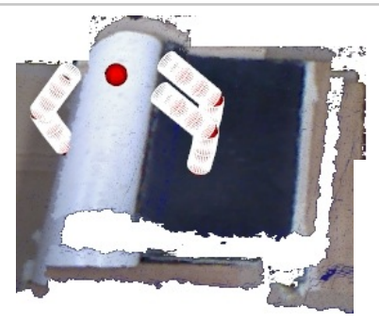


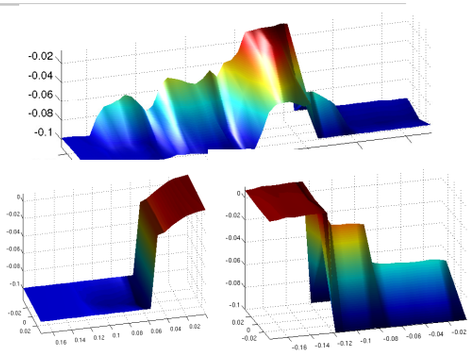
Sequence of Recored Data

Bandwidth Selection









Get an image of the scene from an RGB-D sensor

Segment the scene image into objects

Sample a number of grasping and pushing actions for each object

Extract the features of each sampled action

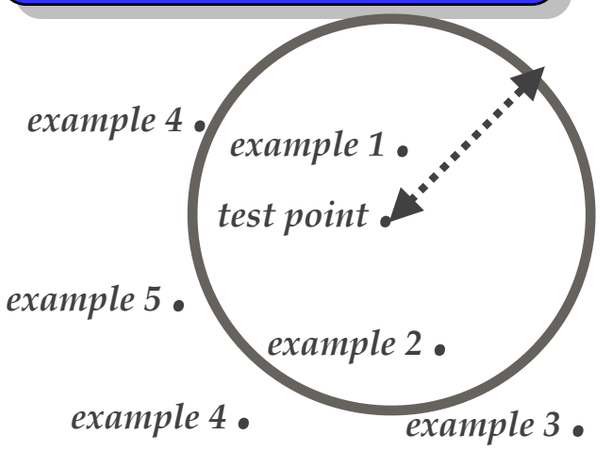
Predict the value of each sampled action using the values of the actions executed in previous states

Execute the action with the highest Upper Confidence Bound (UCB), and obtain a binary reward based on the joint angles of the fingers

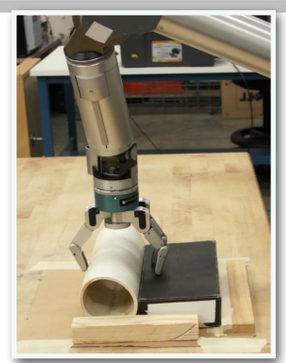
Re-compute the value of every previous state (scene) based on the value of the best action in the next state

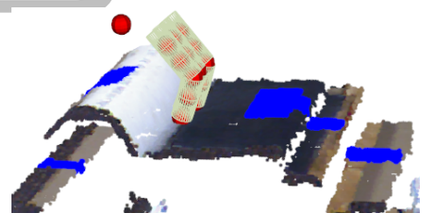
Re-evaluate the actions sampled in every state

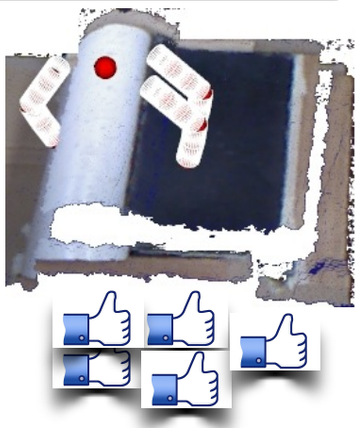
Tune the hyper-parameters (kernel bandwidths) by cross-validation



Policy Iteration (replay)

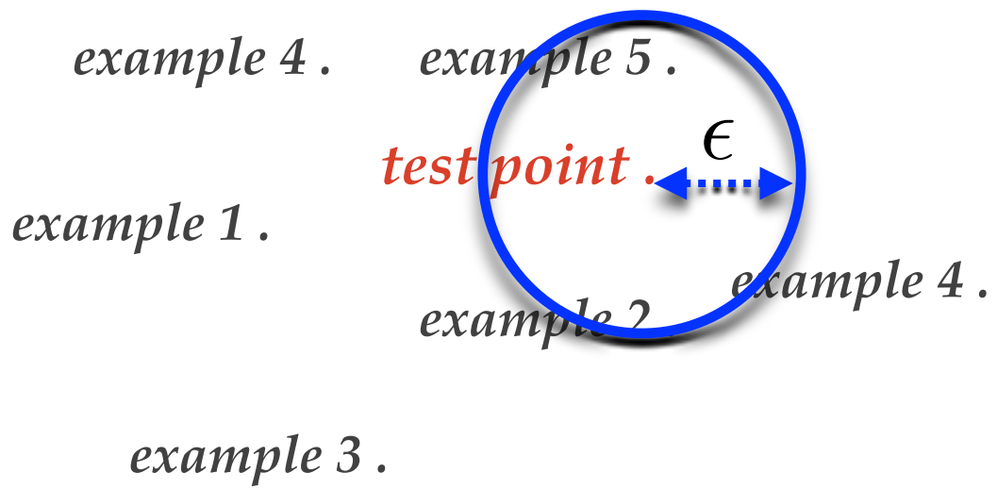






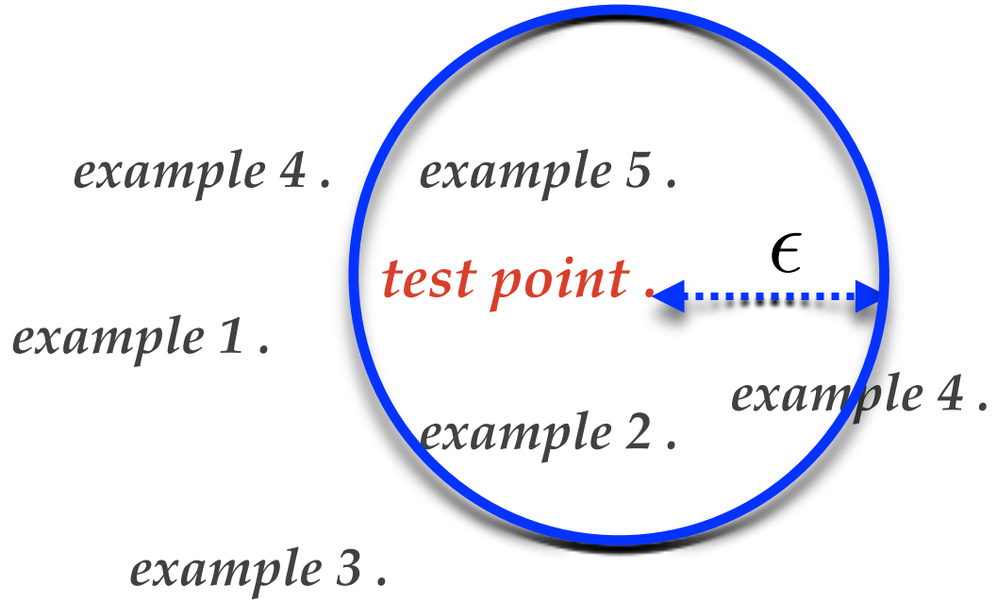
Bandwidth Selection

The kernel's threshold (range) plays a major role in the proposed system. It indicates which data points are similar.



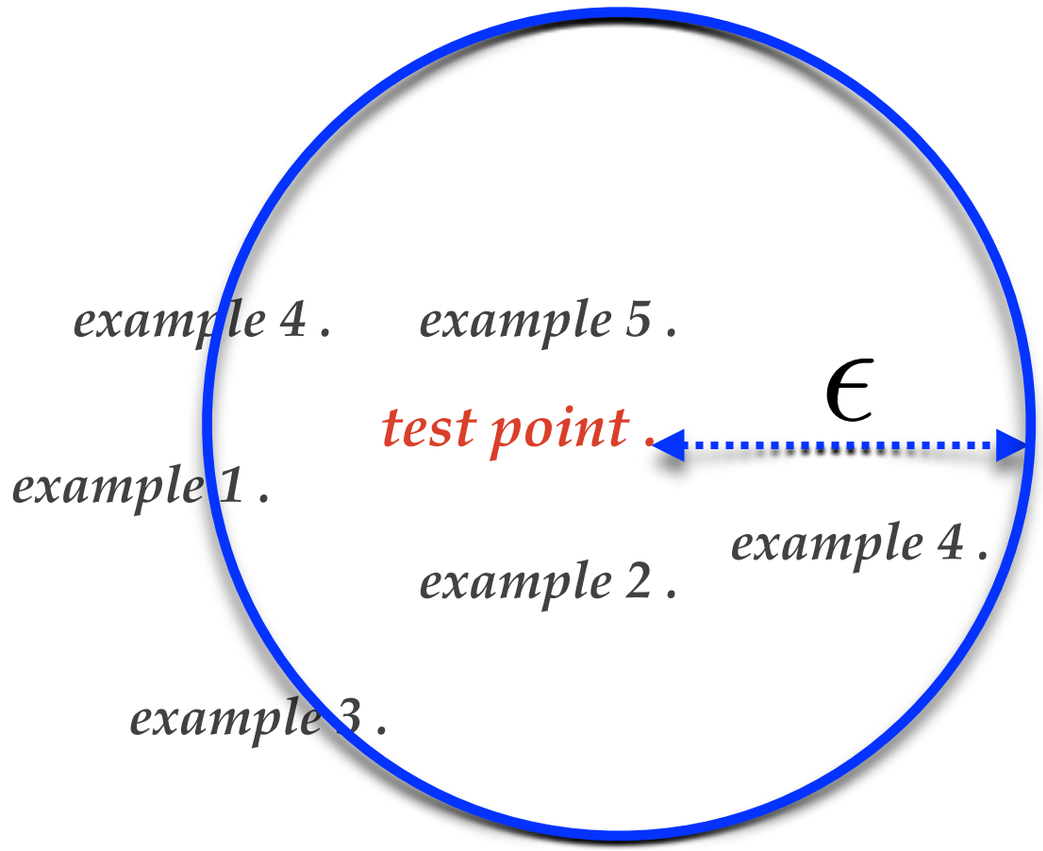
Bandwidth Selection

The kernel's threshold (range) plays a major role in the proposed system. It indicates which data points are similar.



Bandwidth Selection

The kernel's threshold (range) plays a major role in the proposed system. It indicates which data points are similar.



Bandwidth Selection

The range is automatically tuned by selecting the threshold that minimizes the *Bellman error* in the training data,

$$BE(\epsilon) = \frac{1}{t_2 - t_1} \sum_{i=t_1}^{t_2-1} \left(r_i + \gamma \hat{V}^{\epsilon}_{\hat{\pi}}(s_{i+1}) - \hat{Q}^{\epsilon}_{\hat{\pi}}(s_i, a_i) \right)^2.$$

immediate reward

Predicted value of next state

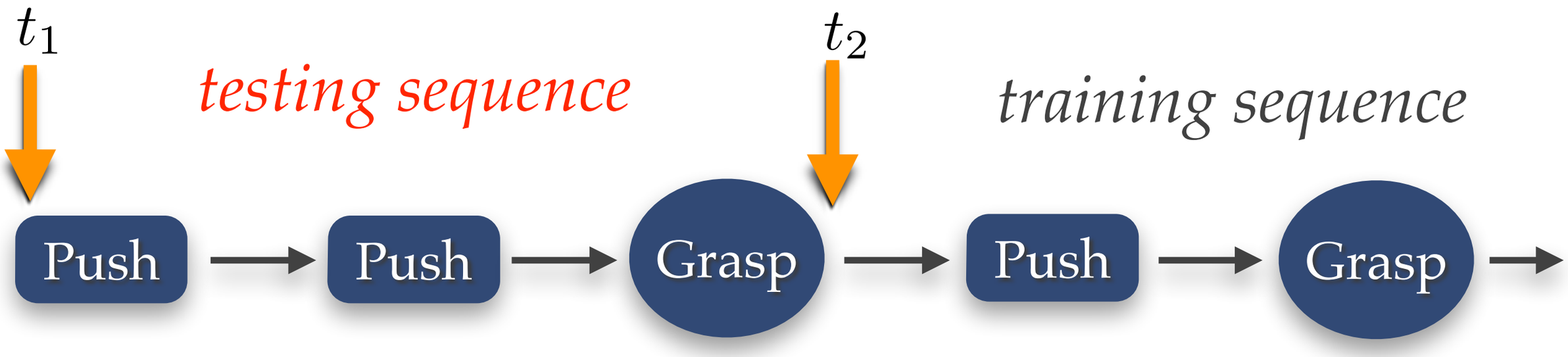
Predicted value of current state

Bandwidth Selection

The range is automatically tuned by selecting the threshold that minimizes the *Bellman error* in the training data,

$$BE(\epsilon) = \frac{1}{t_2 - t_1} \sum_{i=t_1}^{t_2-1} \left(r_i + \gamma \hat{V}^{\epsilon}_{\hat{\pi}}(s_{i+1}) - \hat{Q}^{\epsilon}_{\hat{\pi}}(s_i, a_i) \right)^2.$$

immediate reward — **Predicted value of next state** — **Predicted value of current state**

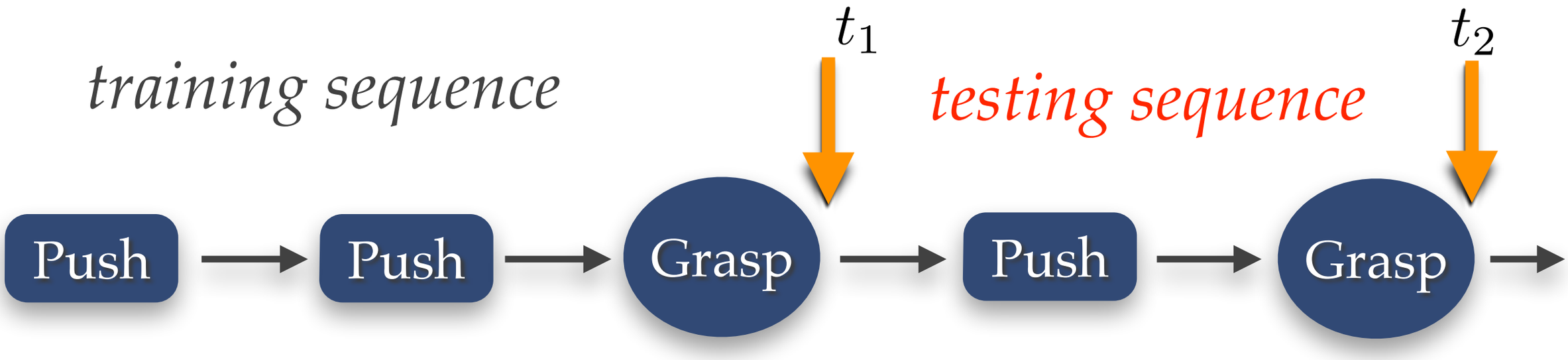


Bandwidth Selection

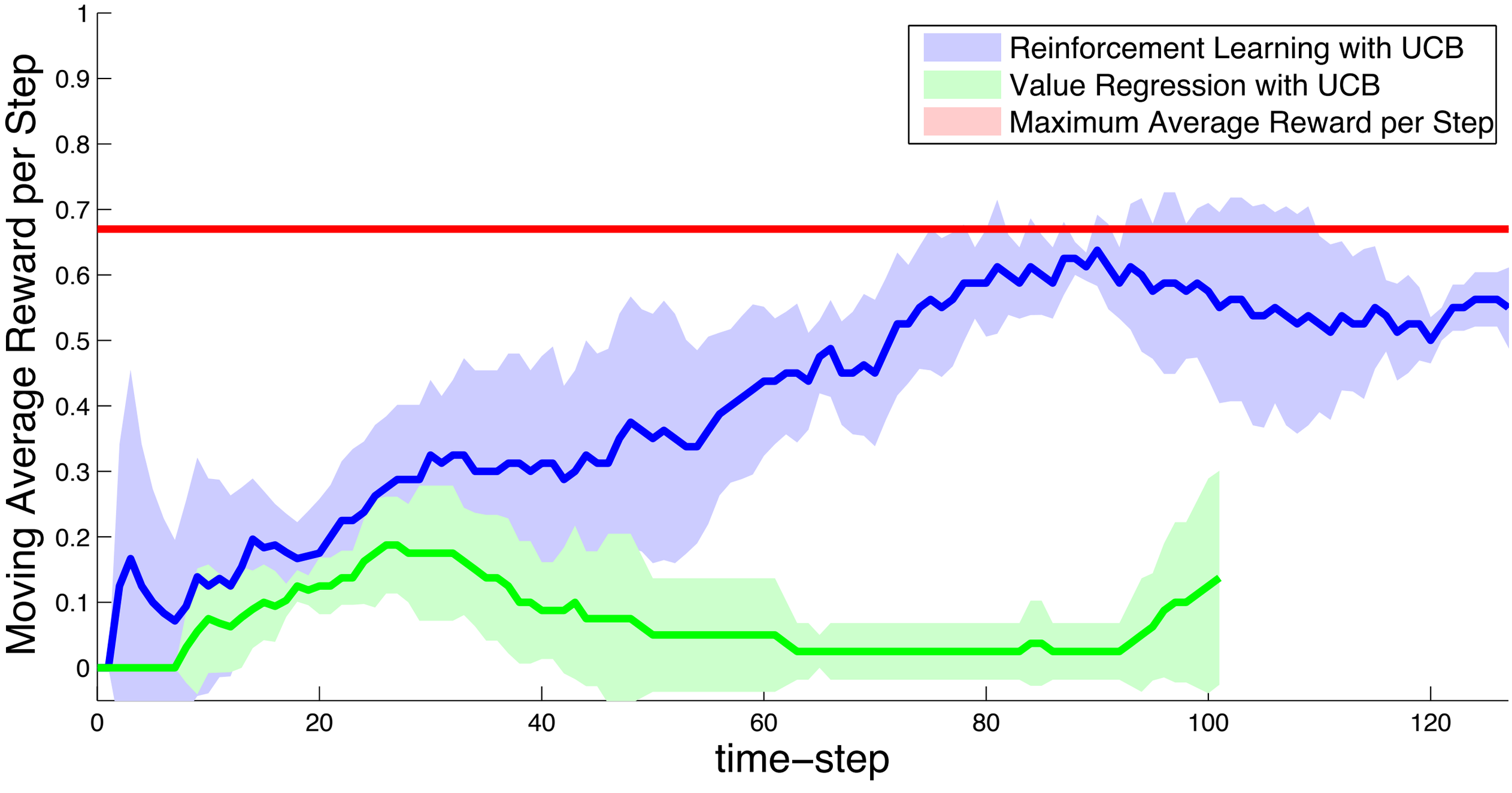
The range is automatically tuned by selecting the threshold that minimizes the *Bellman error* in the training data,

$$BE(\epsilon) = \frac{1}{t_2 - t_1} \sum_{i=t_1}^{t_2-1} \left(r_i + \gamma \hat{V}^{\epsilon}_{\hat{\pi}}(s_{i+1}) - \hat{Q}^{\epsilon}_{\hat{\pi}}(s_i, a_i) \right)^2 .$$

immediate reward — **Predicted value of next state** — **Predicted value of current state**



Learning Curve: Reinforcement Learning V.S. Regression



A. Boularias *et al.* (2015) in *Conference of the Association for the Advancement of Artificial Intelligence (AAAI)*

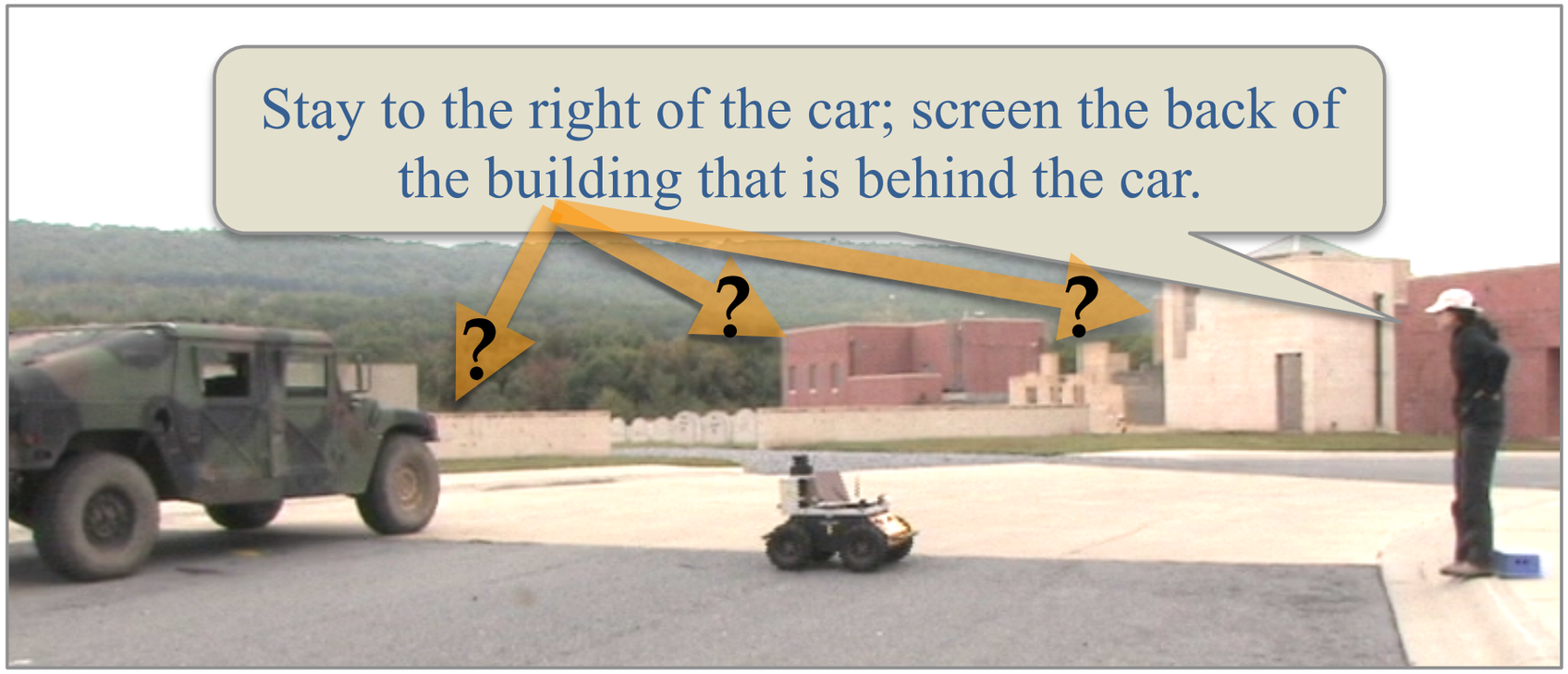
Outline

1. Overview
2. Optimal control
3. Inverse optimal control
4. Grasping
5. Manipulation
6. **Navigation**

Grounding Spatial Relations for Robot Navigation

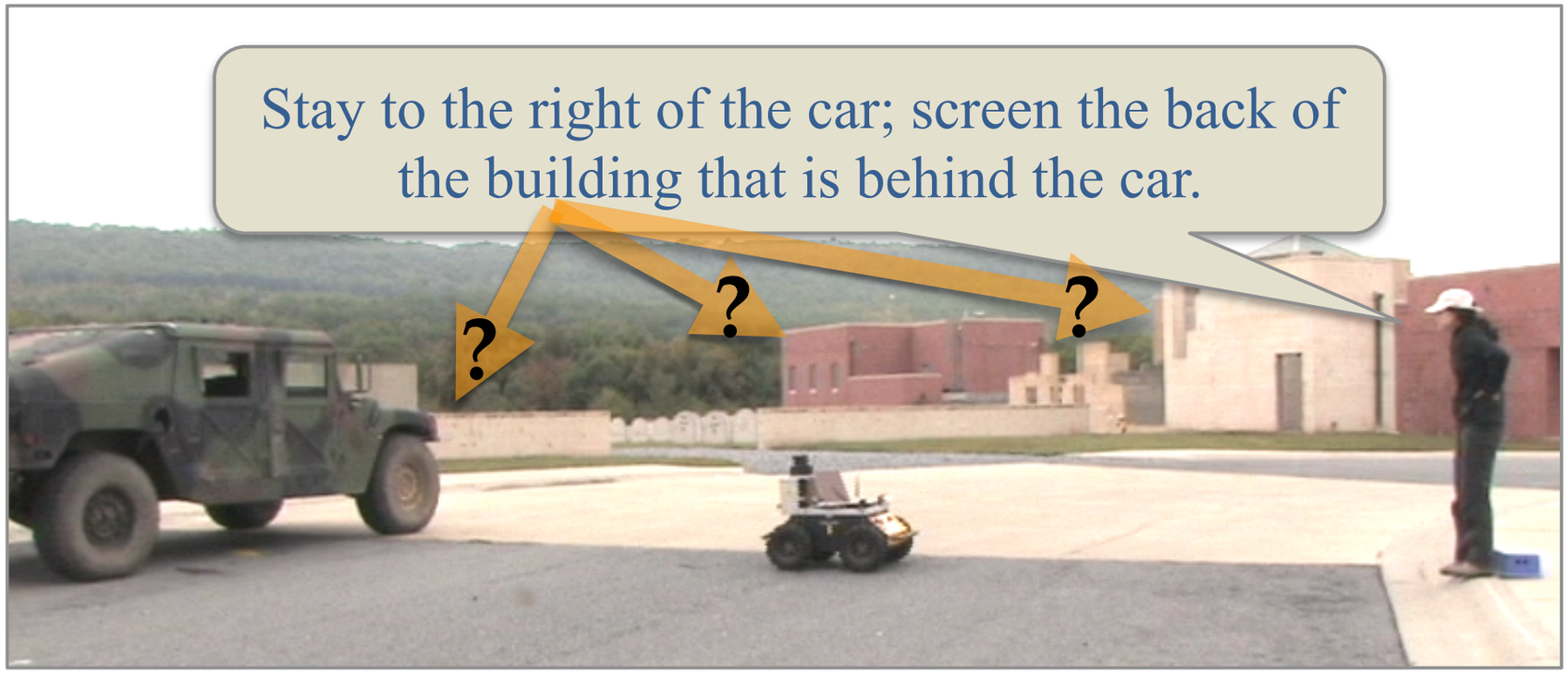
J. Oh *et al.* (2015) in *Conference of the Association for the Advancement of Artificial Intelligence (AAAI)*

A. Boularias *et al.* (2015) in *IEEE International Conference on Robotics and Automation (ICRA)*



Which building? which car??

Grounding: map each noun in the command to an object in the world

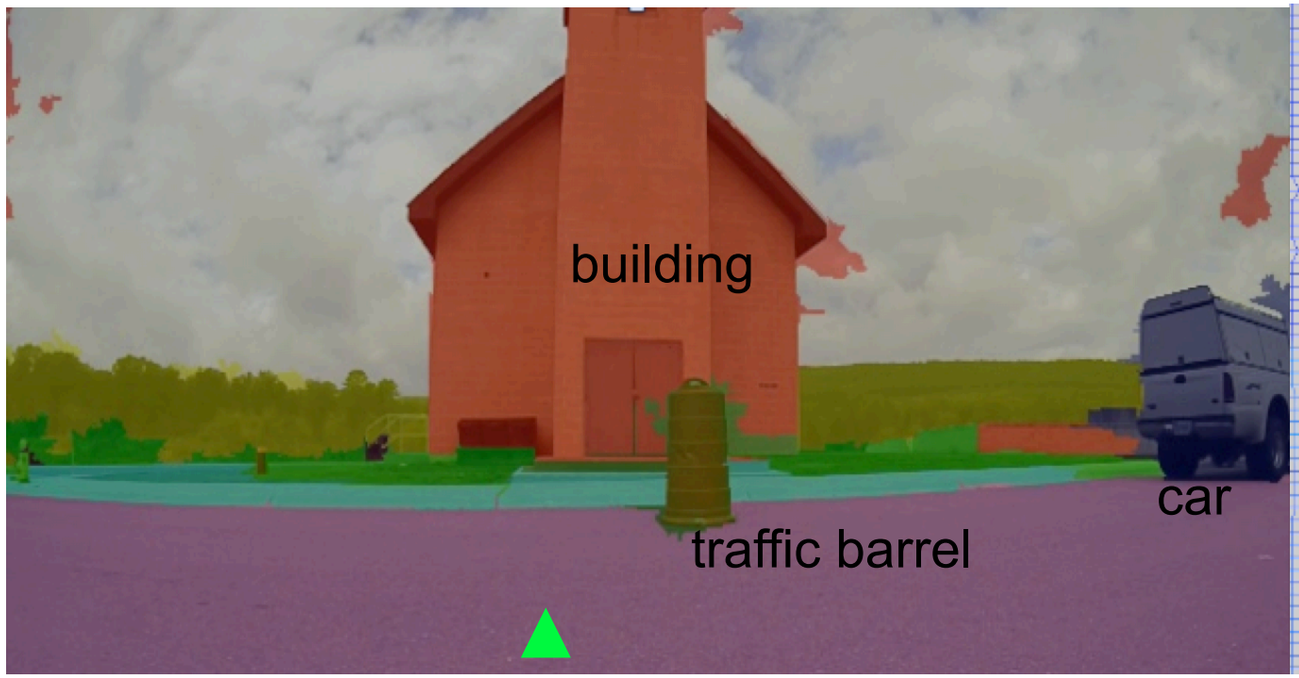


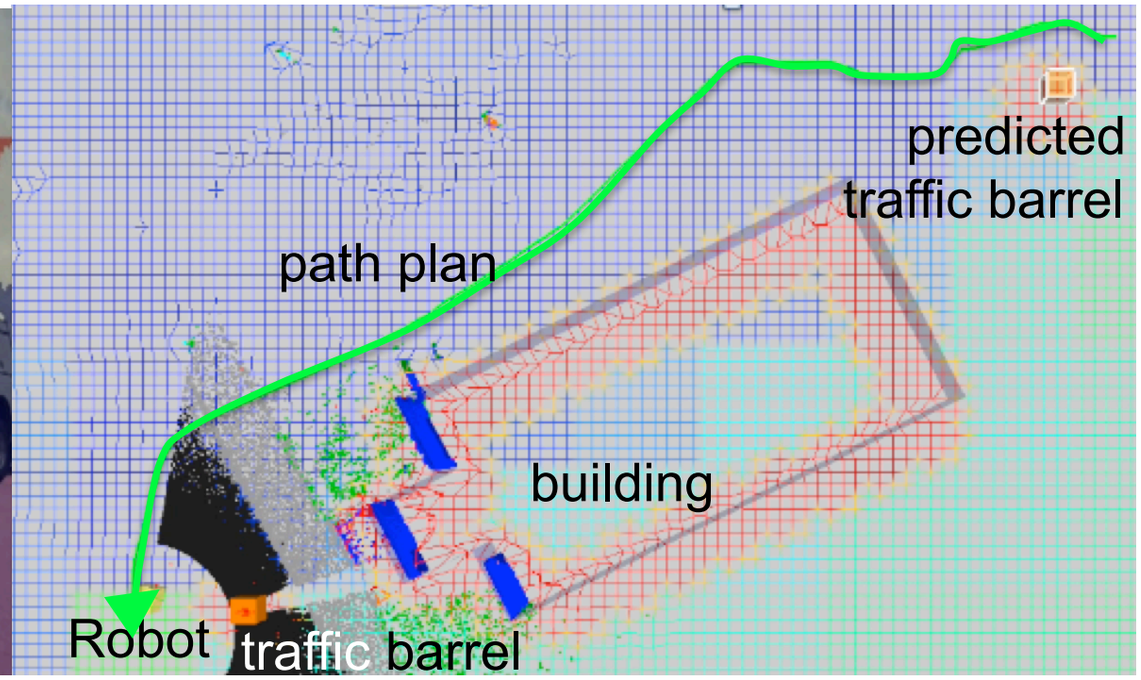
Which building? which car??

Grounding: map each noun in the command to an object in the world

Spatial concepts (such as **behind** and **near**) are learned from examples

Bayesian probabilistic model for dealing with object recognition errors





Environment as perceived by the robot **Planned path**

Result: the robot navigated to the correct goal **88%** of the time.

Thank you!