Robot Learning: Algorithms and Applications

Abdeslam Boularias

Carnegie Mellon University

National Robotics Engineering Center

Outline

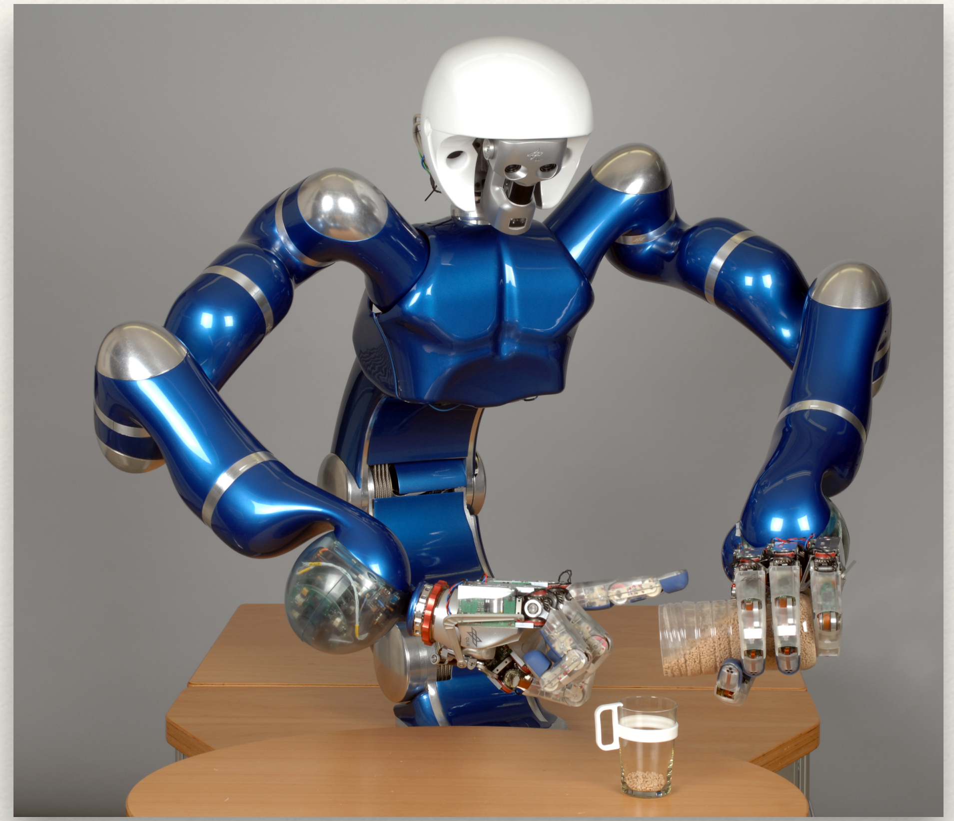
1. Overview
2. Optimal control
3. Inverse optimal control
4. Grasping
5. Manipulation
6. Navigation

Outline

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

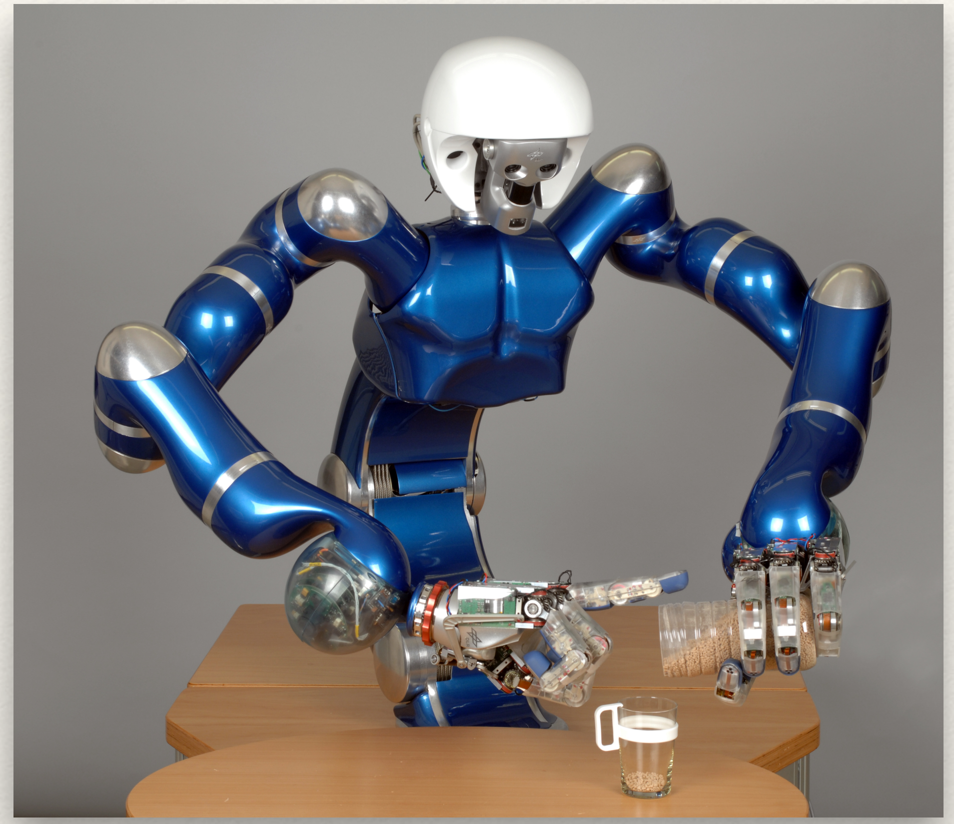
Acting in unstructured environments

DLR

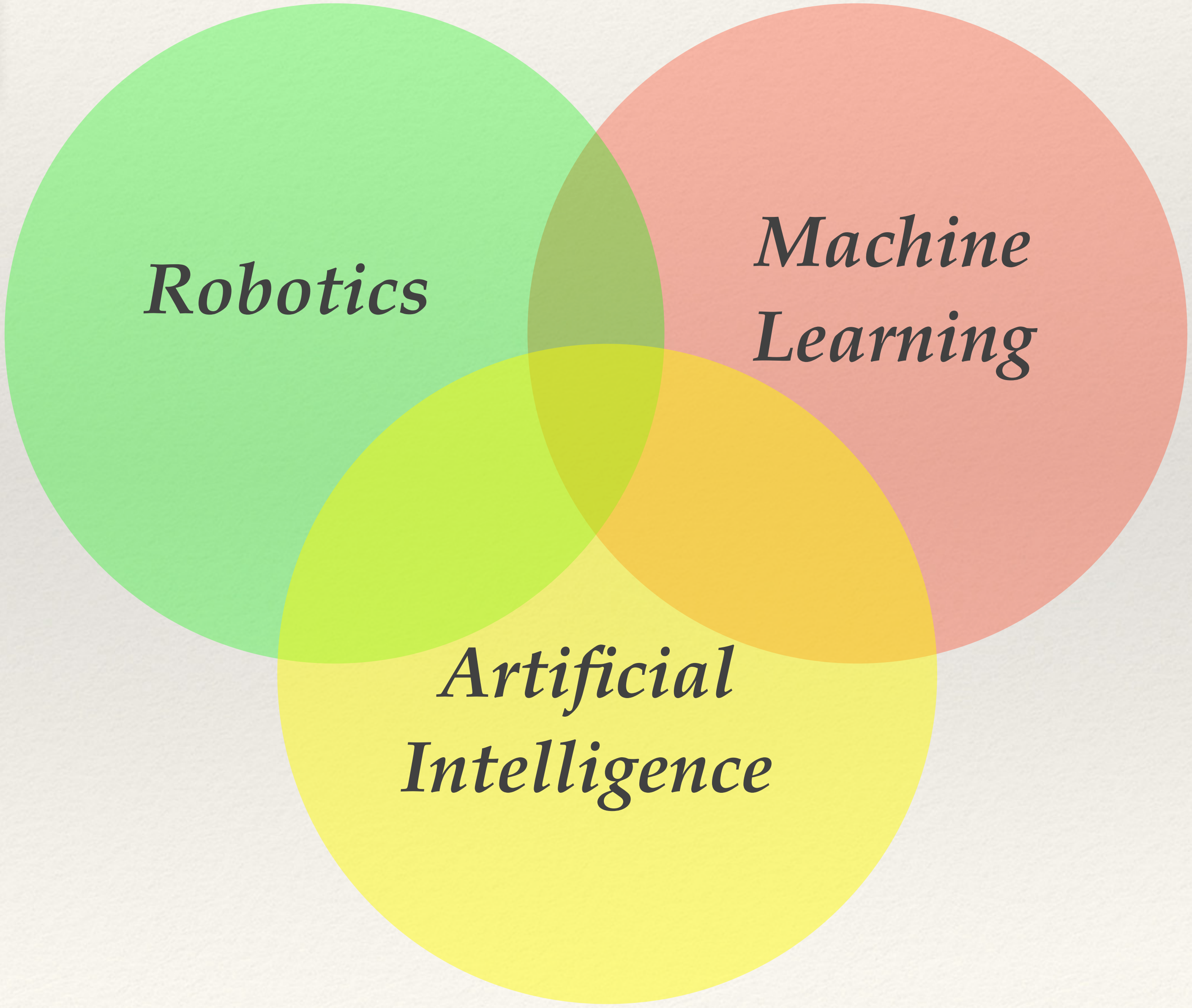


How can a robot learn to perform complex tasks from experience?

DLR

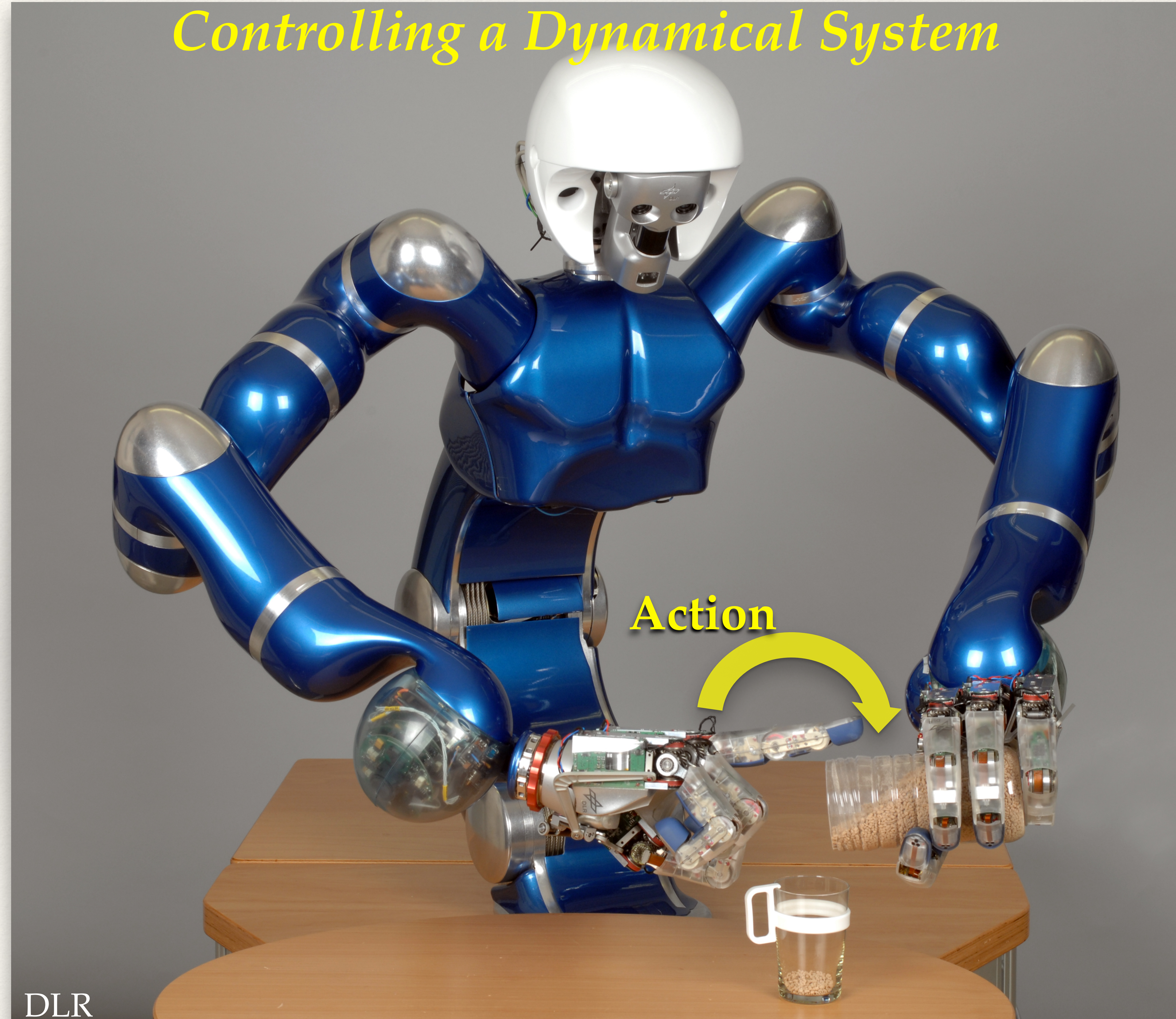


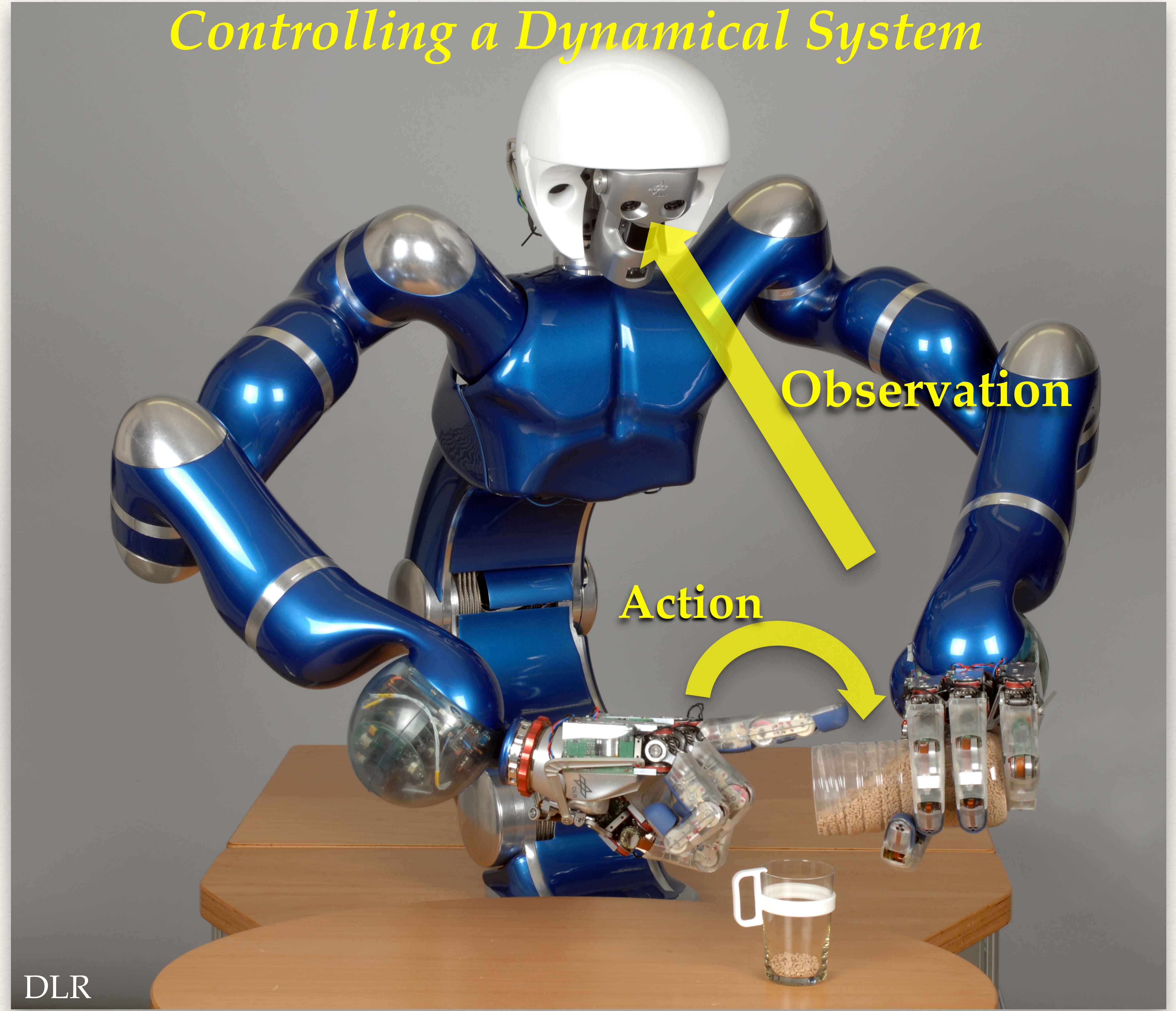
How can a robot learn to perform complex tasks from experience?



DLR

Controlling a Dynamical System

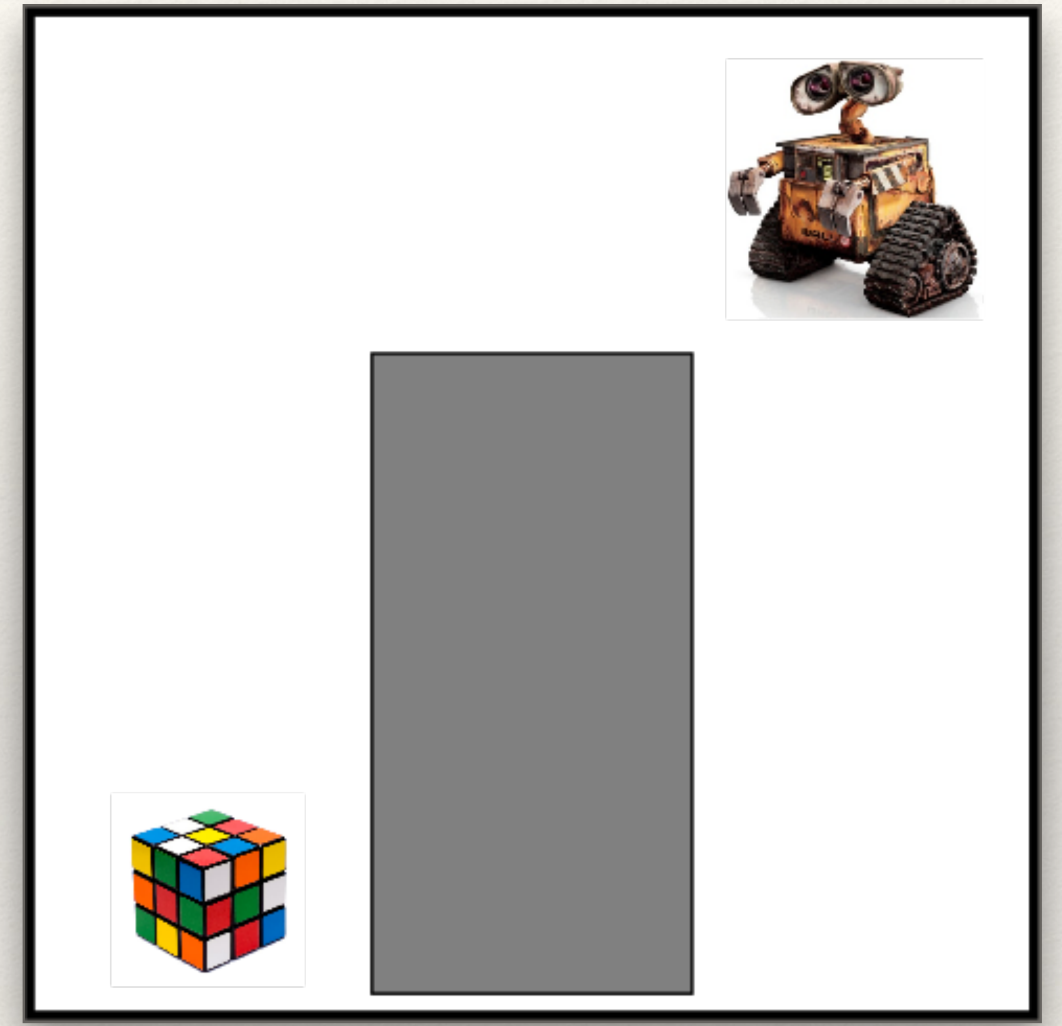



Controlling a Dynamical System
Observation
Action
DLR

Outline

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

Example

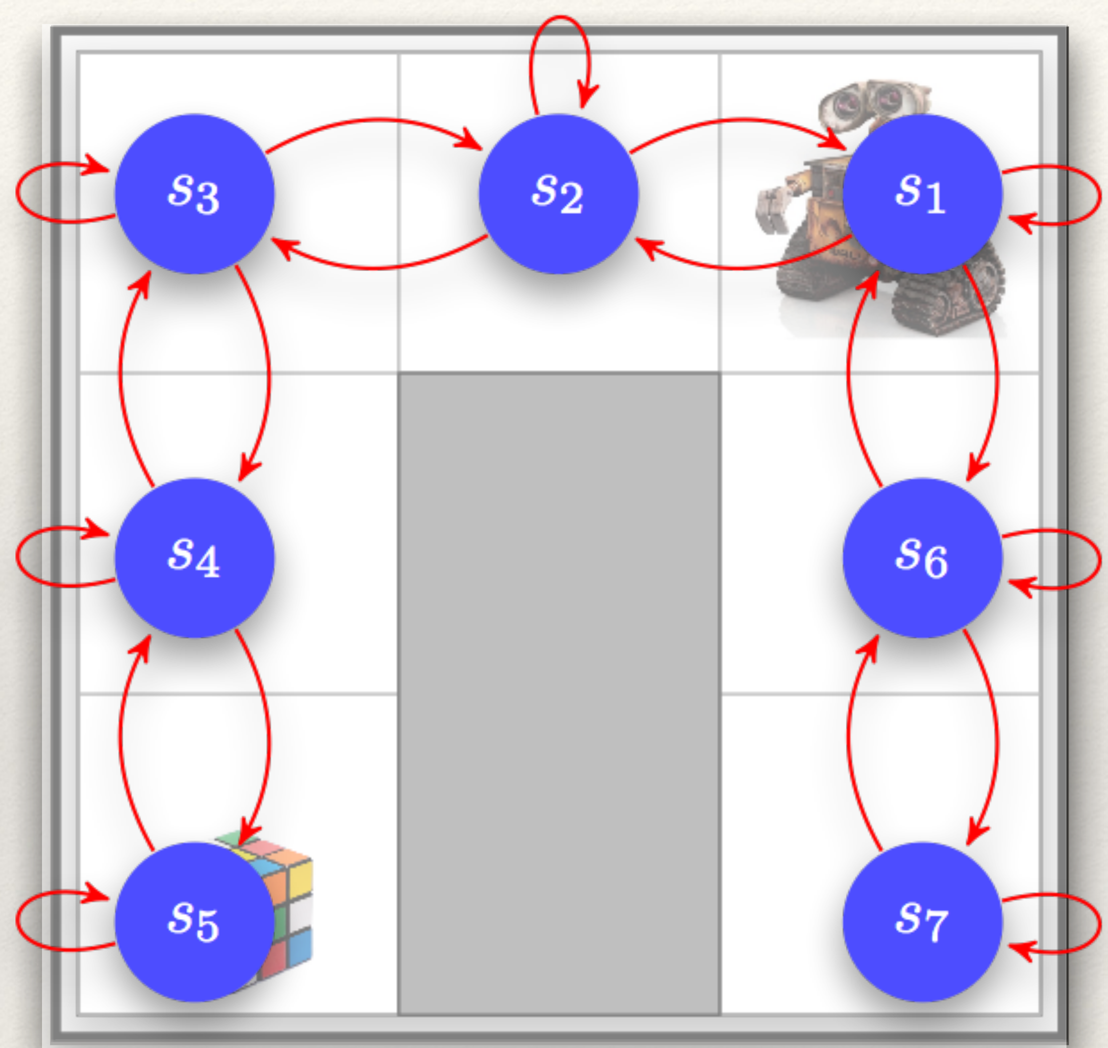


Path planning: a simple sequential decision-making problem

Example

Path planning: a simple sequential decision-making problem

States and Actions



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

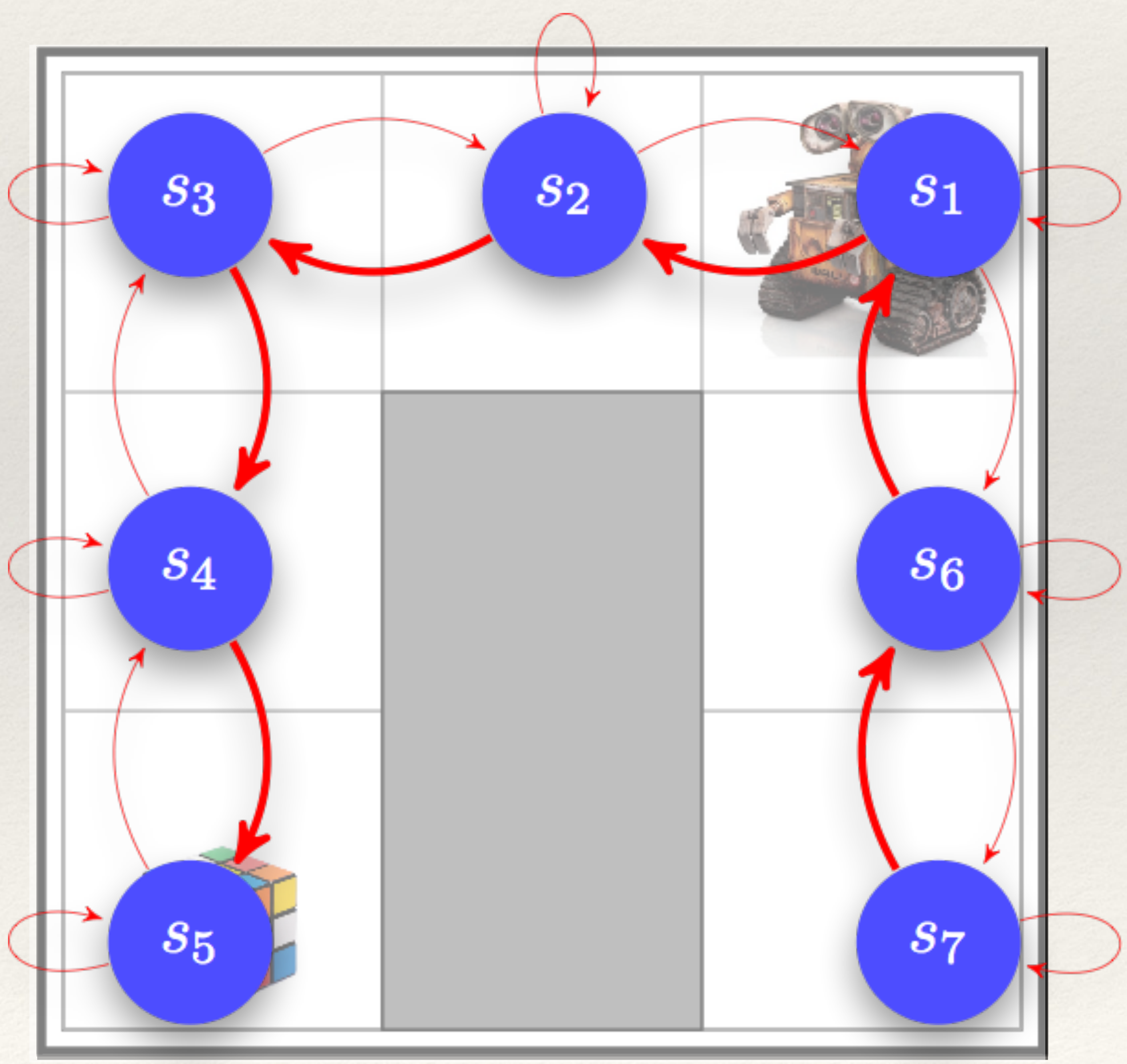
$$T(s, a, s') = Pr(\underbrace{s_{t+1}}_{\text{next state}} = s' | s_t = \underbrace{s}_{\text{current state}}, a_t = \underbrace{a}_{\text{current action}})$$

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

Policies

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

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



Value function

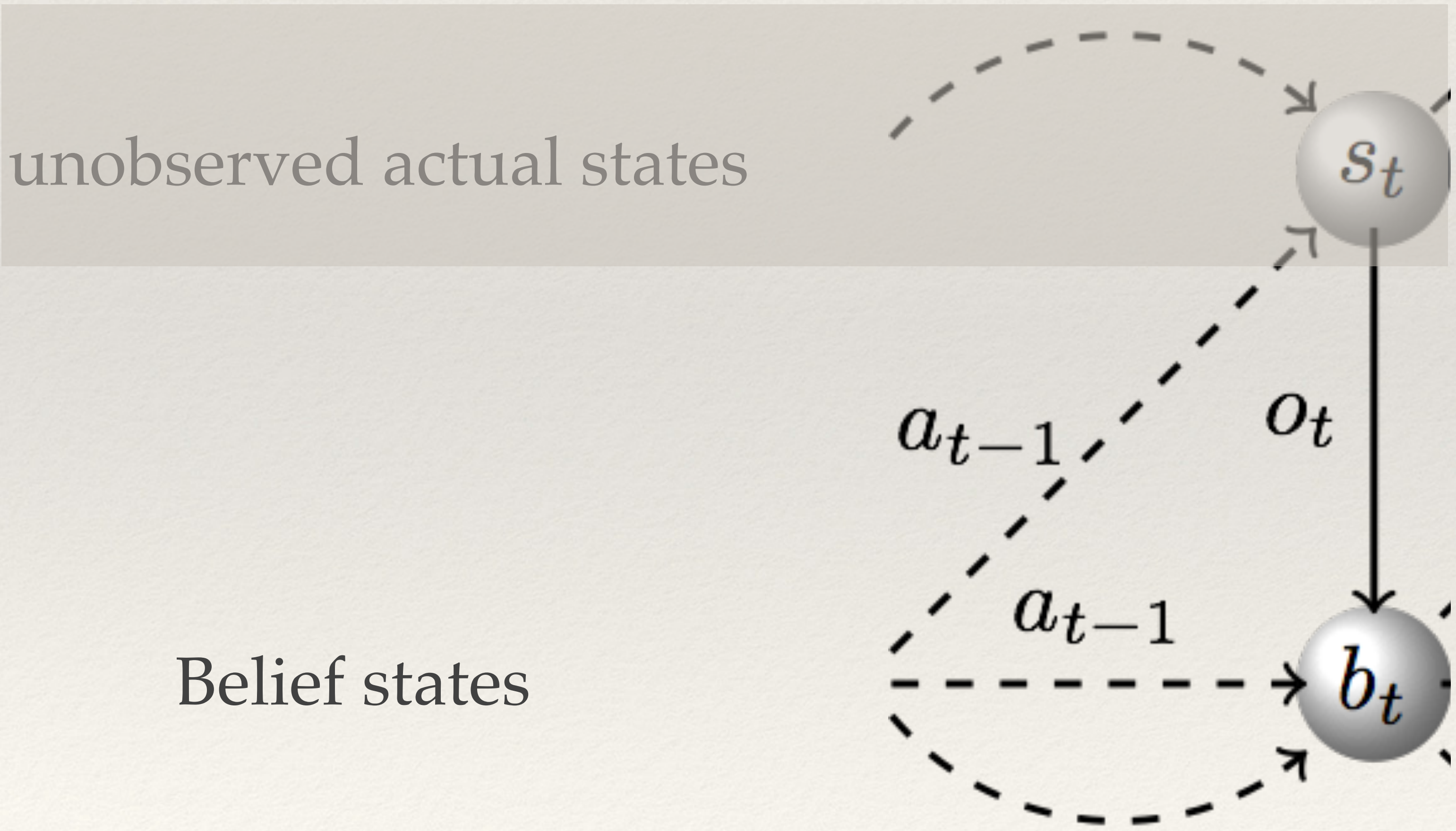
The ***value*** (or ***utility***) of a policy **π** is the sum of rewards that one expects to gain by following it.

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

Goal: finding an **optimal policy**.

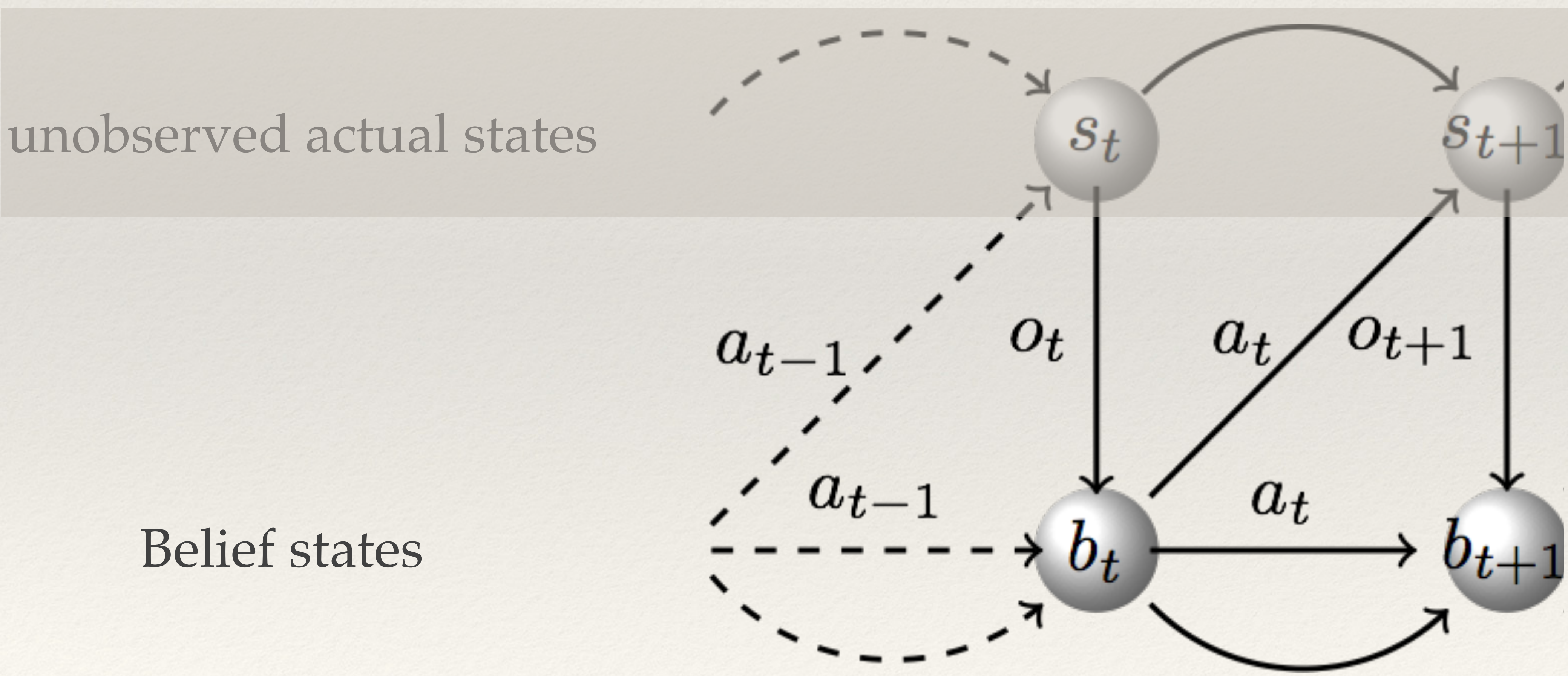
Partially Observable Markov Decision Process

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



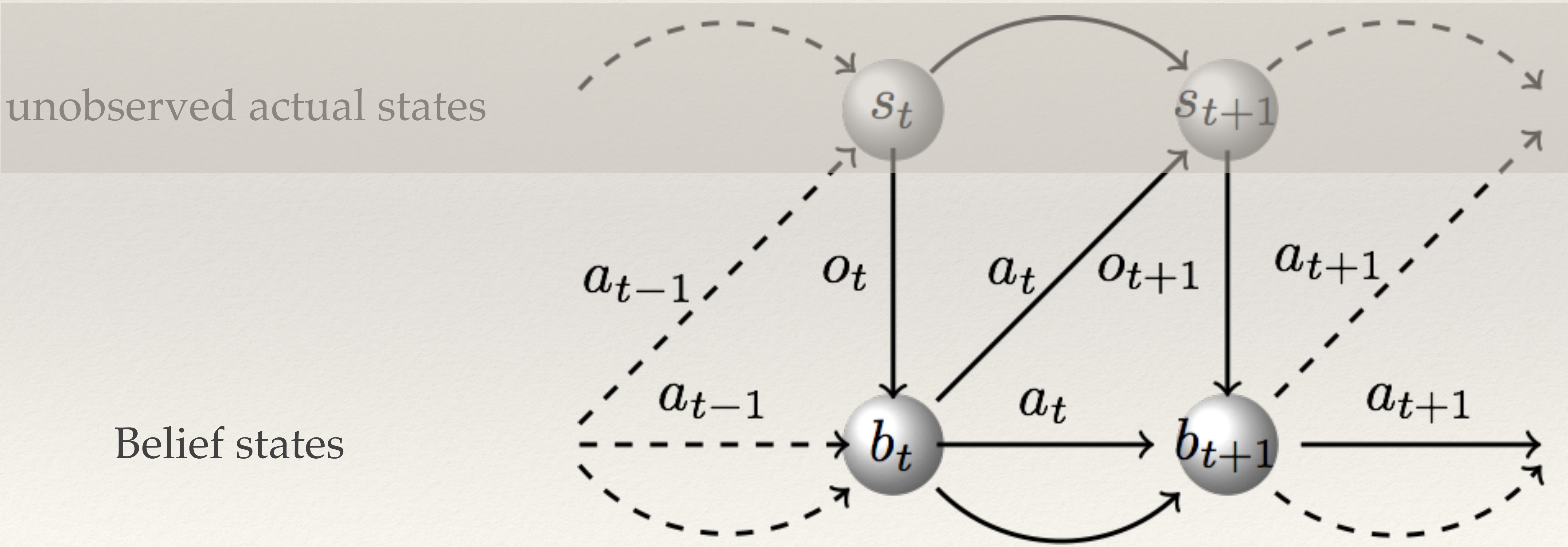
Partially Observable Markov Decision Process

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



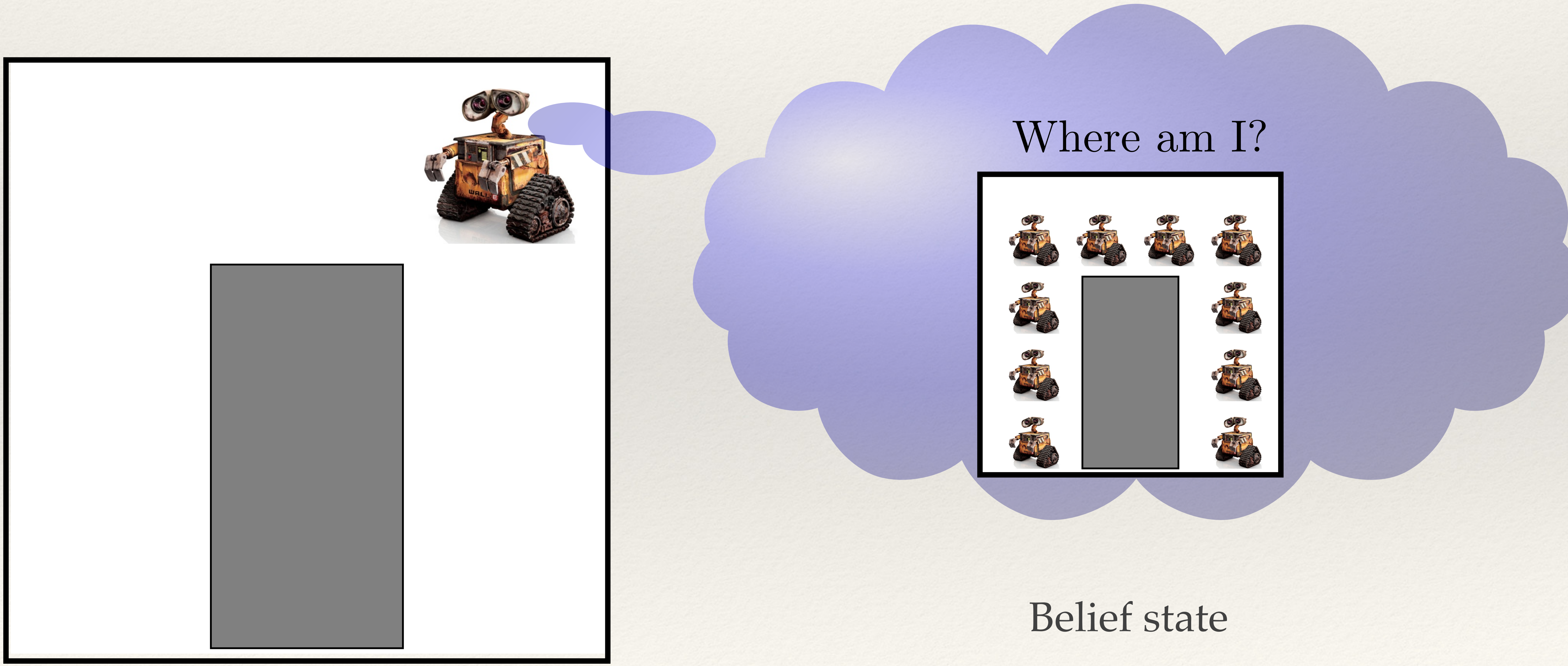
Partially Observable Markov Decision Process

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



Partially Observable Markov Decision Process (POMDP)

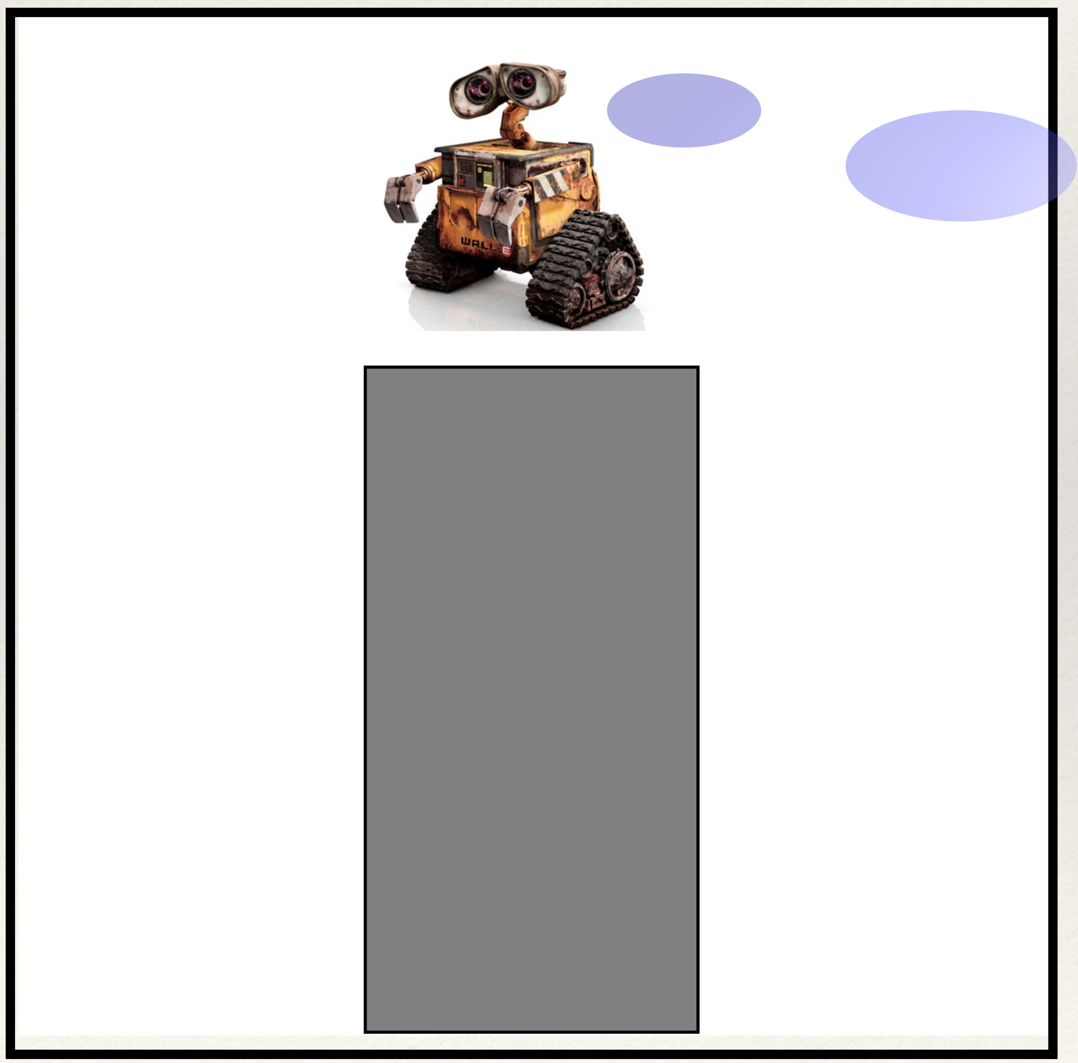
Example: the robot can sense an obstacle only after bumping into it.

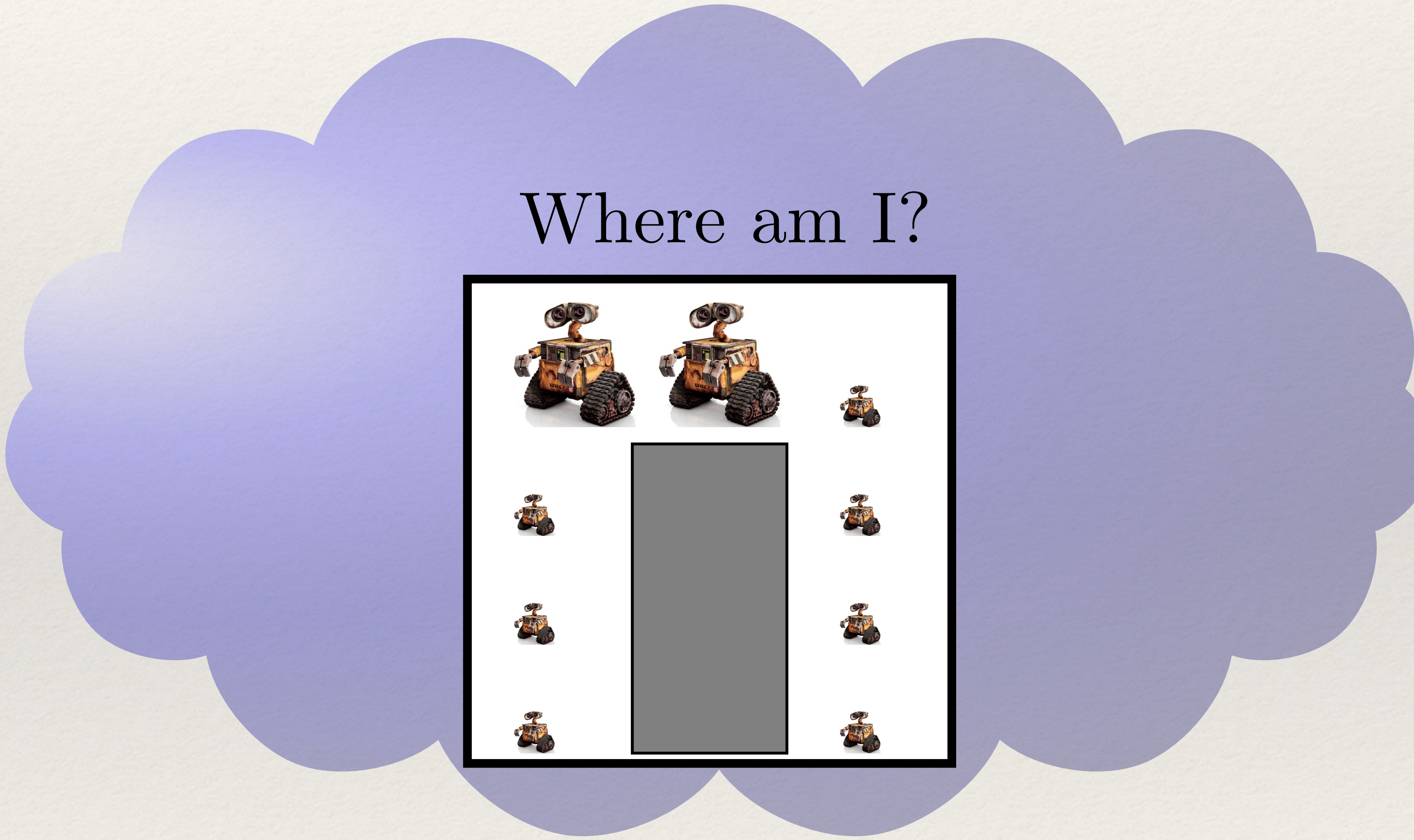


Belief state

Partially Observable Markov Decision Process (POMDP)

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

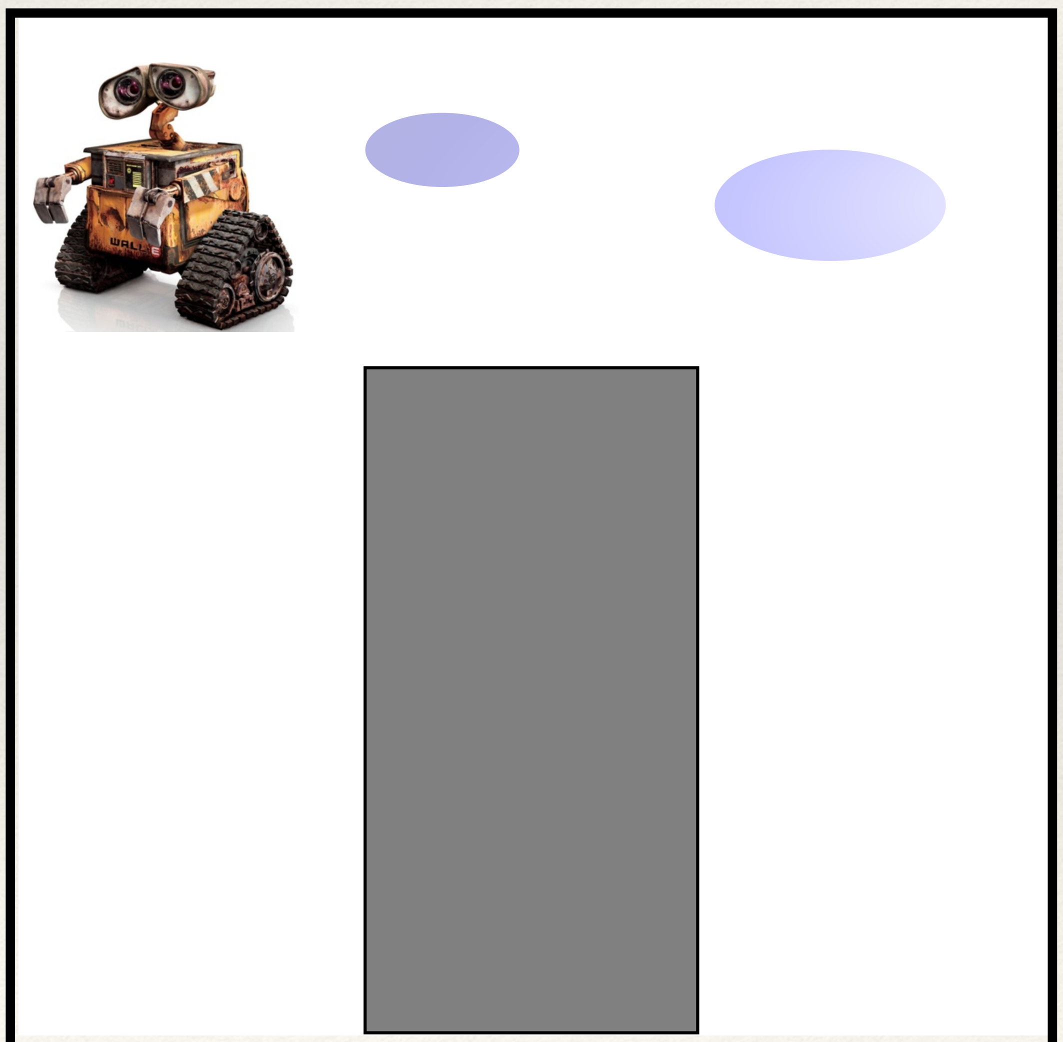


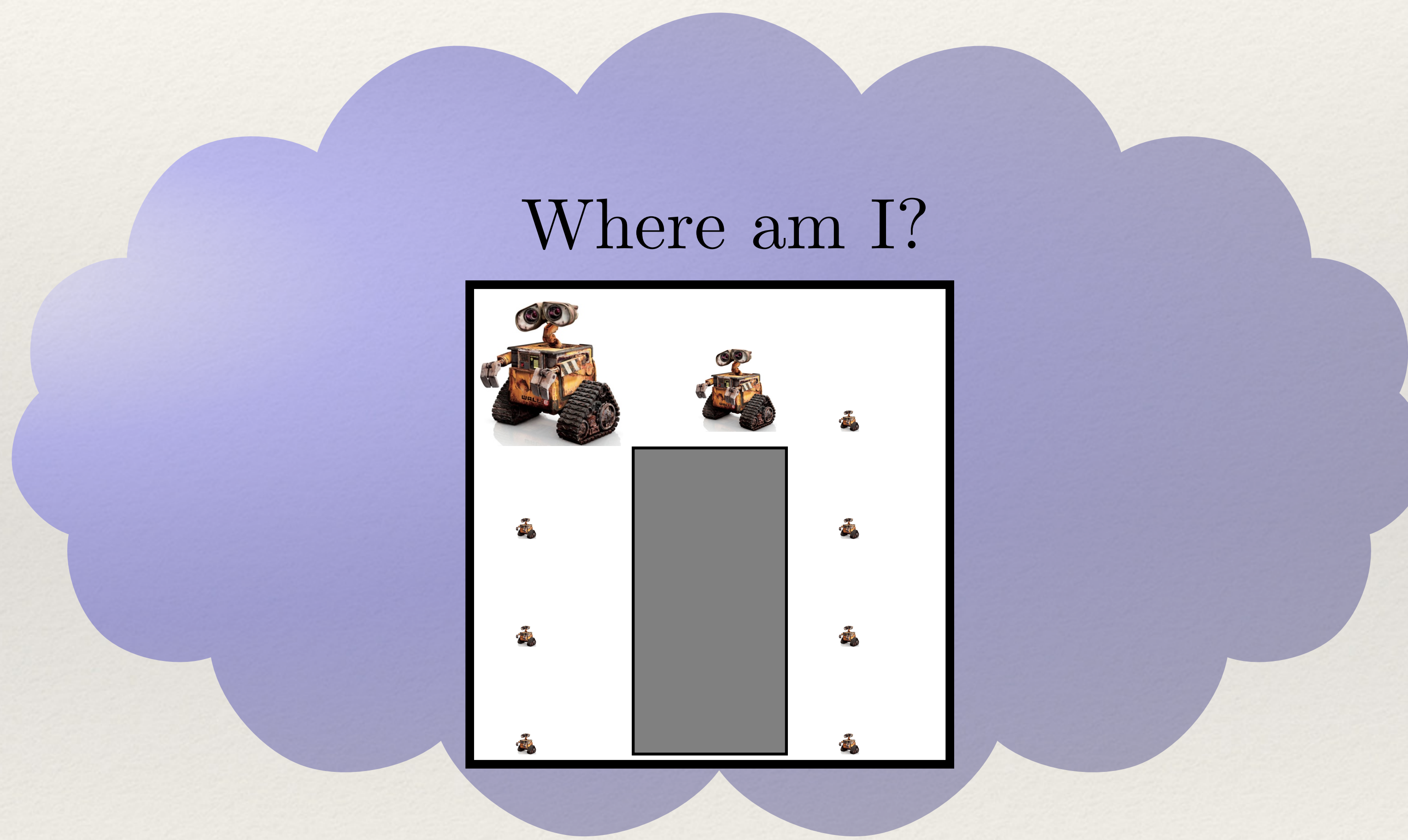


Belief state

Partially Observable Markov Decision Process (POMDP)

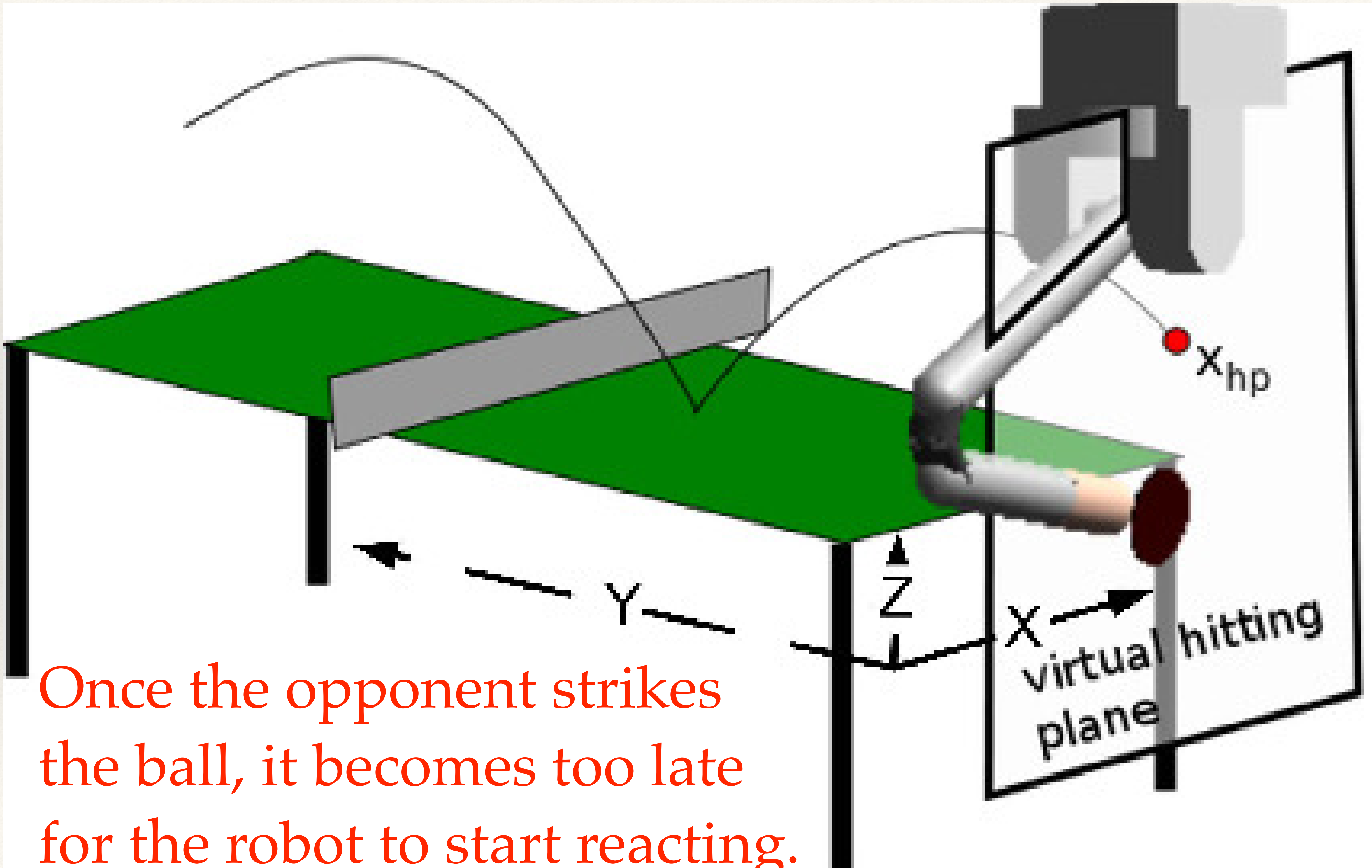
Example: the robot can sense an obstacle only after bumping into it.





Belief state

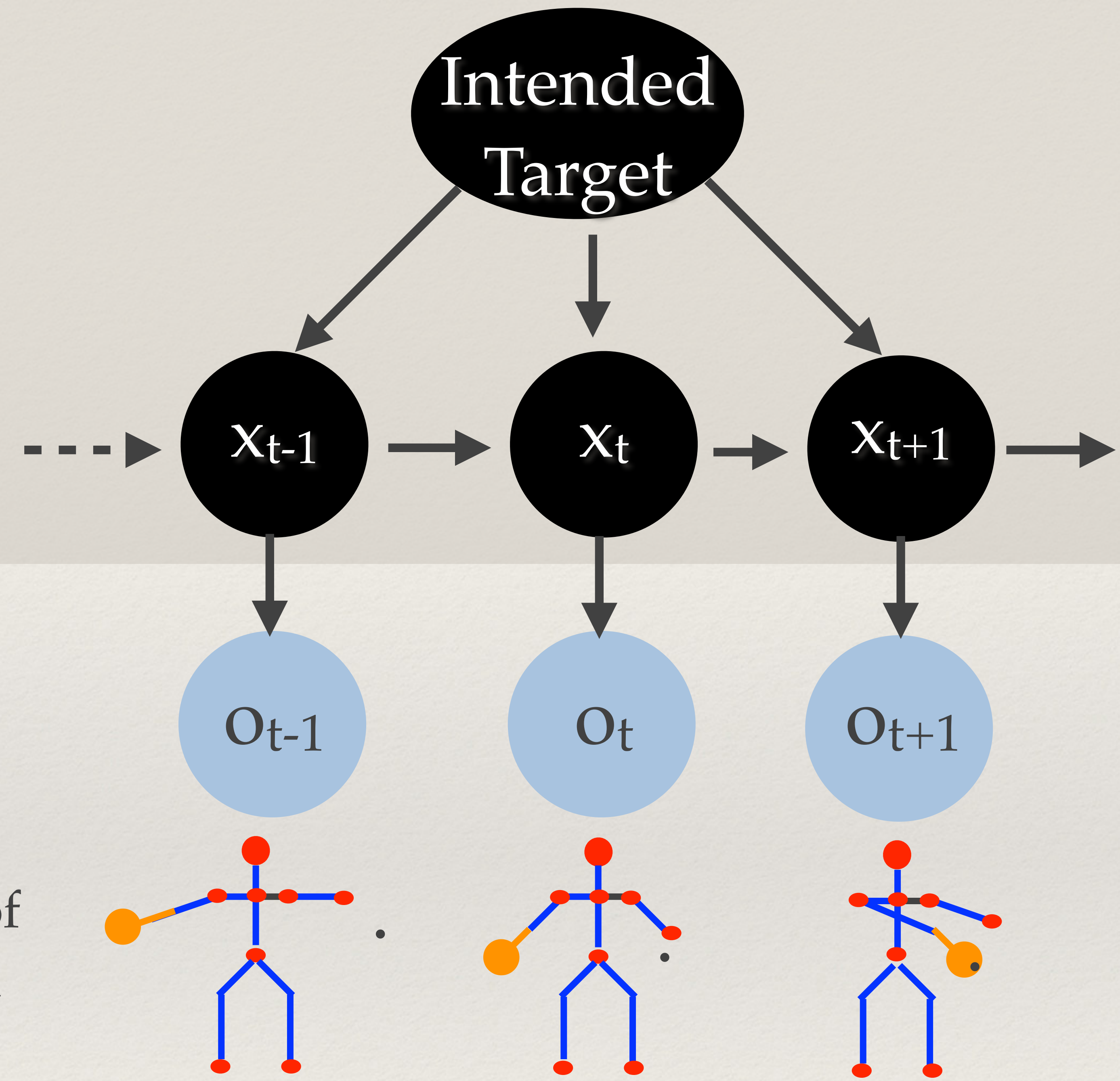
The target of the ball should be predicted in advance



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

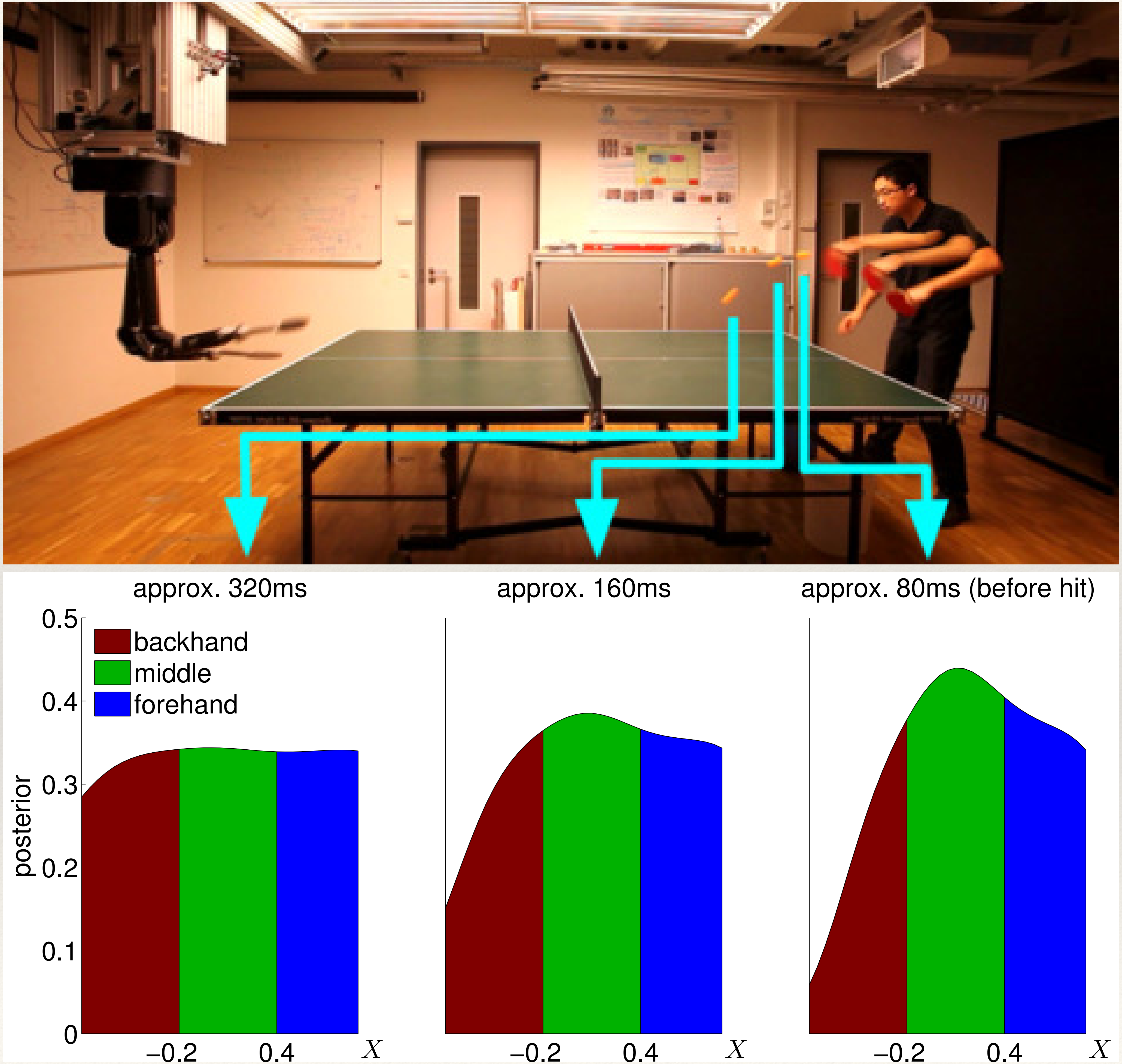


Observations

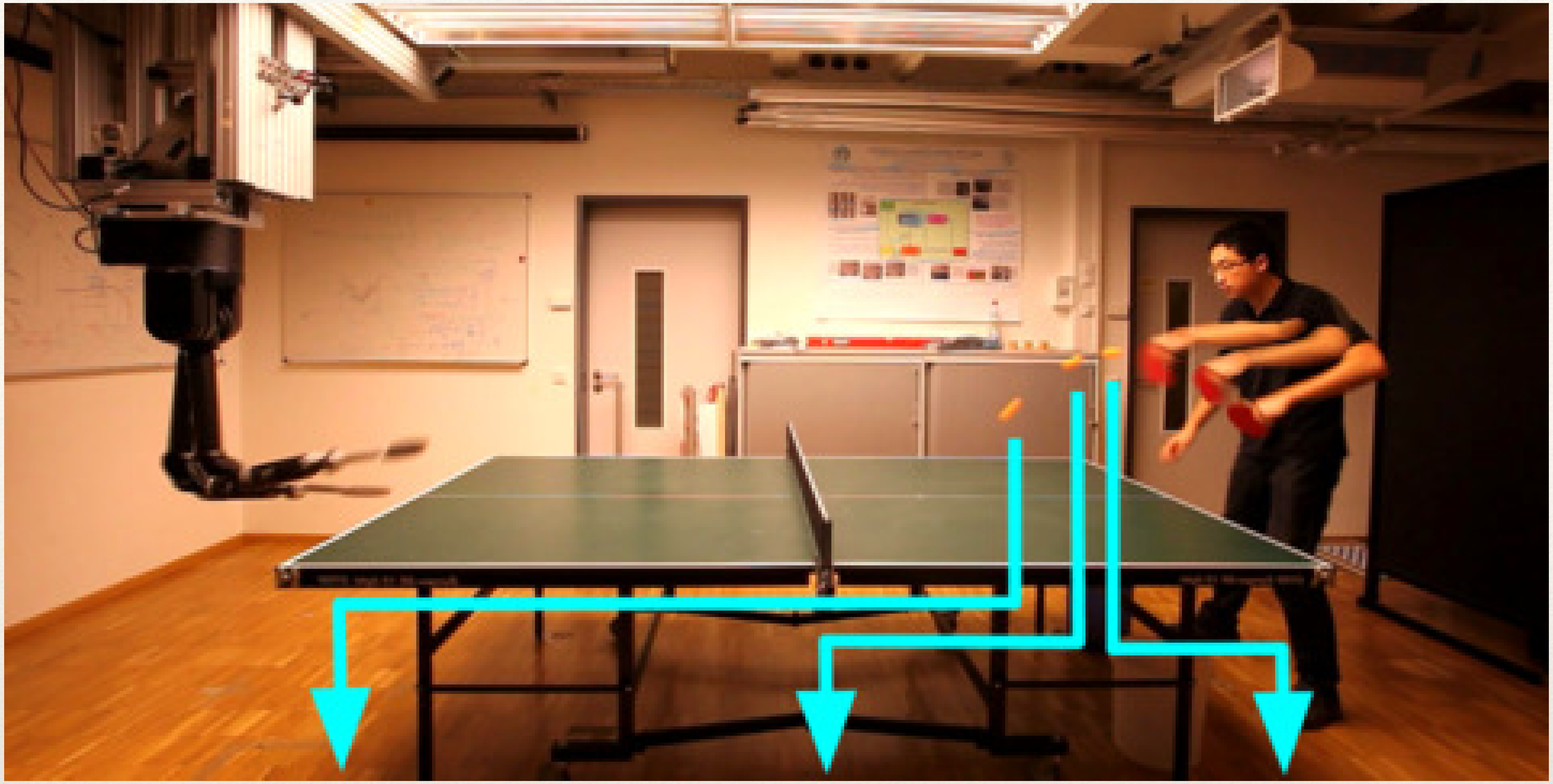
Positions of the ball, the racket and joints of the opponent, tracked using a *Kinect* camera

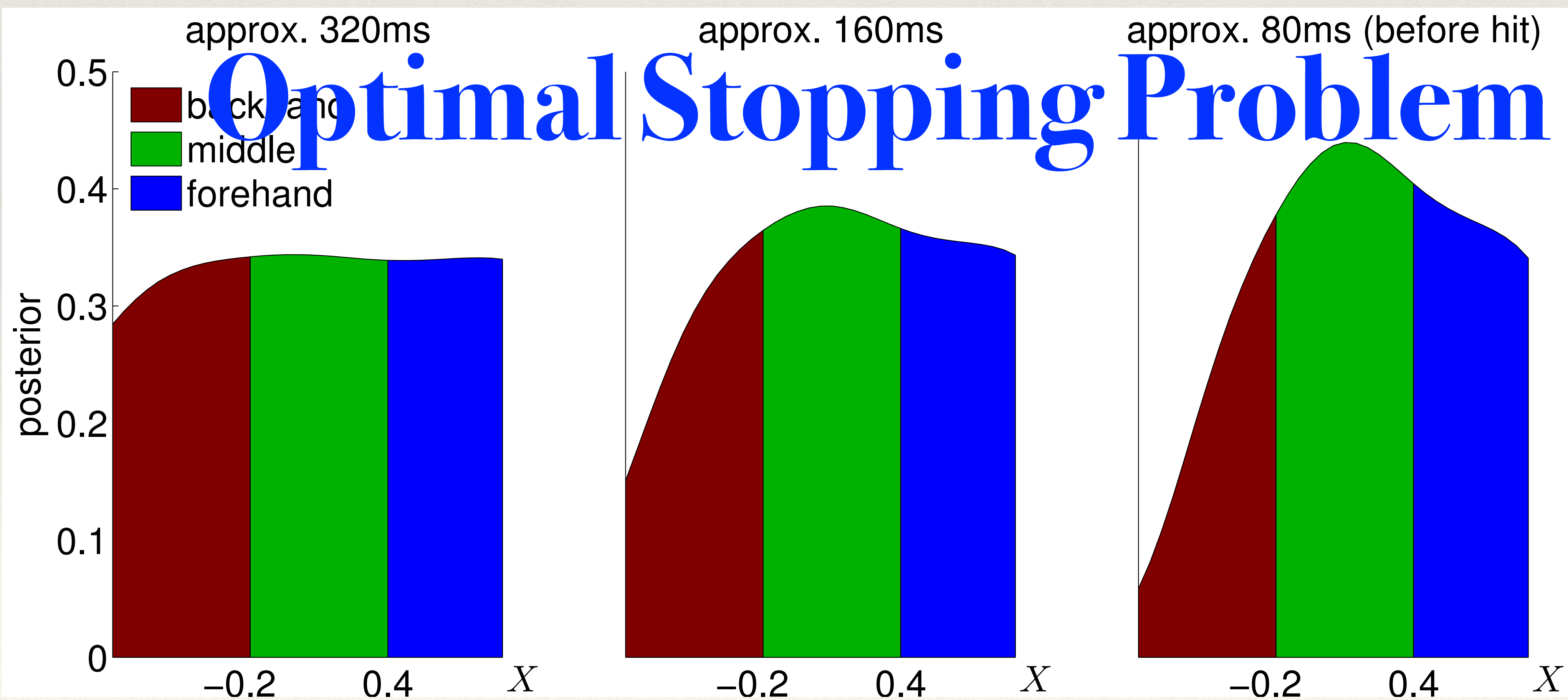
Observations o_t are generated according to a *Gaussian Process*.

From observations o_t , one can calculate a probability distribution on the intended target.


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

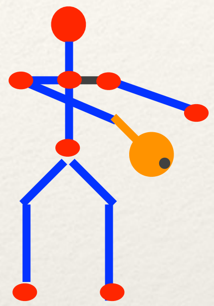
Sample current state and intention $s_t = [x_t, \text{target}]$

↓

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

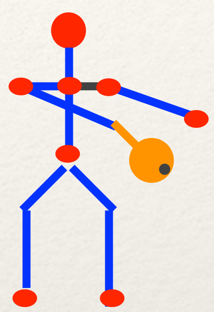
Sample current state and intention $s_t = [x_t, \text{target}]$

↓

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

↓

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



↓

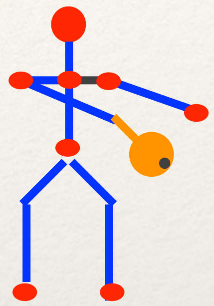
Update belief b_{t+1} provided observation o_{t+1}

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}

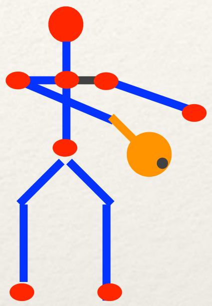
Predict the expected performance $V(b_{t+1})$

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



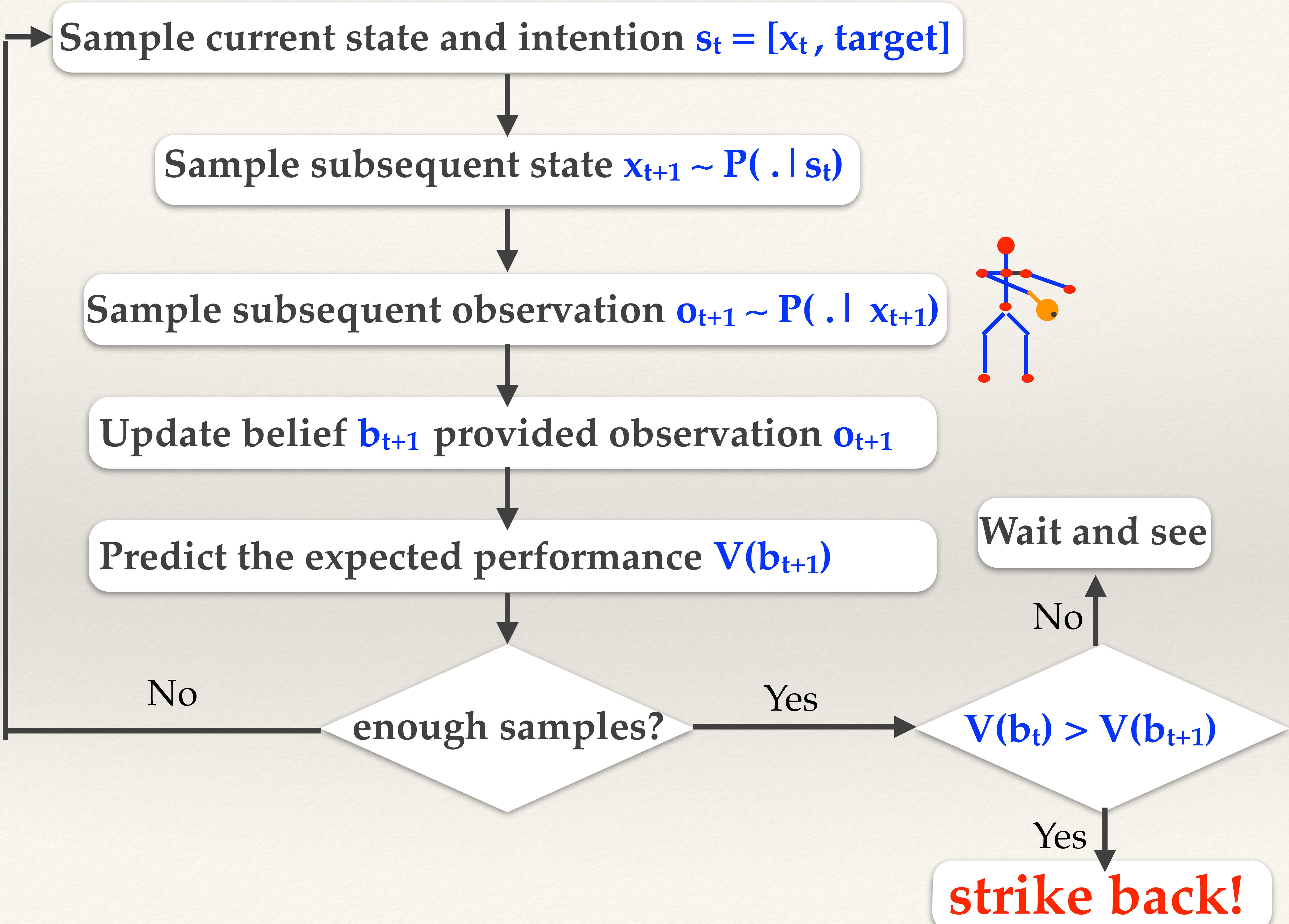
Update belief b_{t+1} provided observation o_{t+1}

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

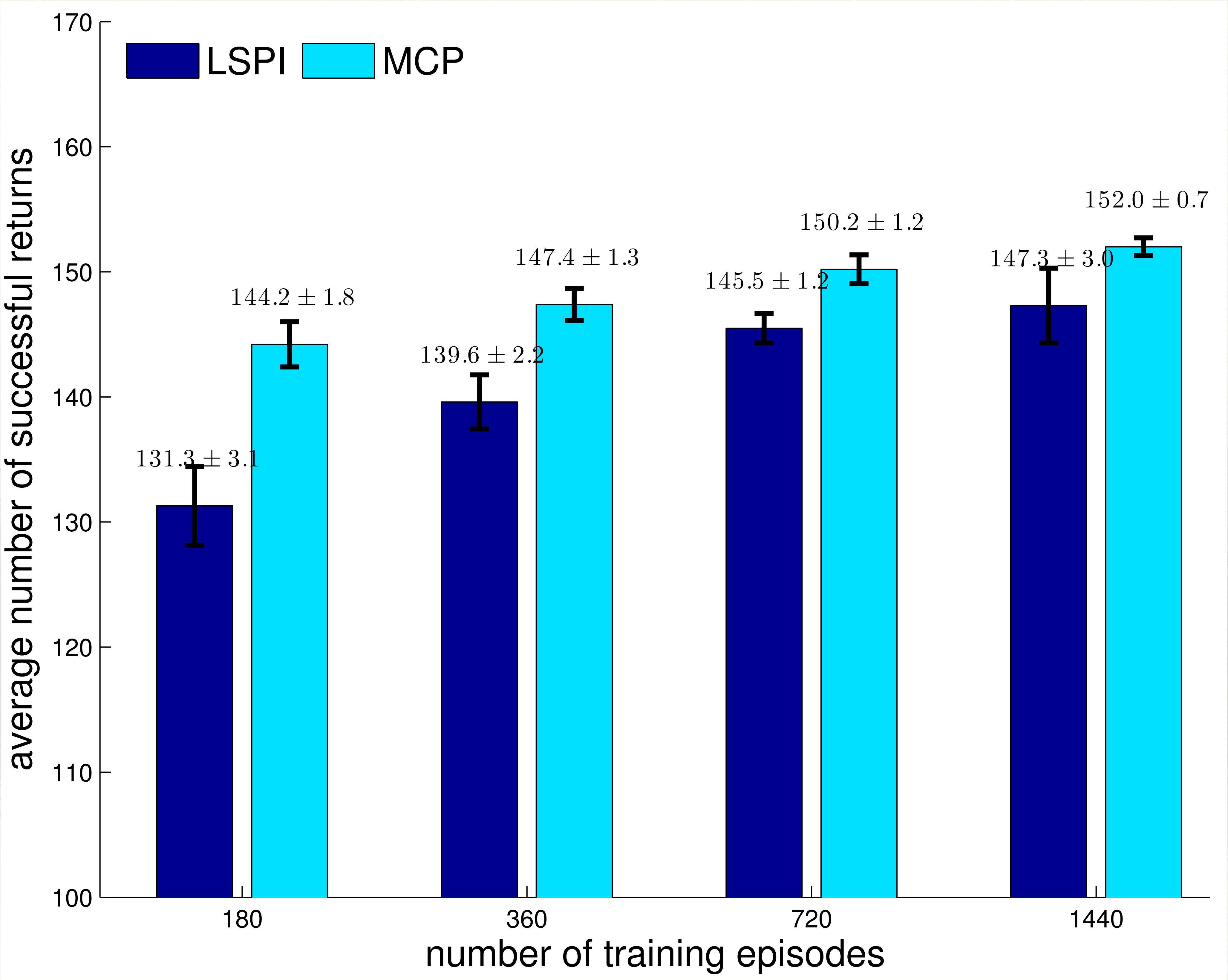
enough samples?

No

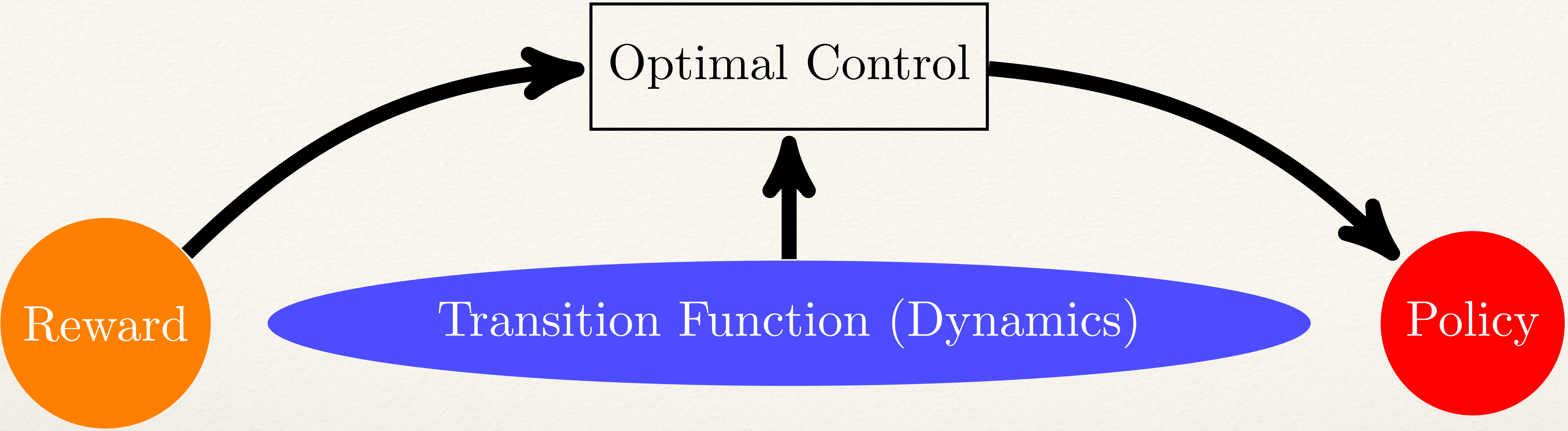
A Monte-Carlo planning algorithm



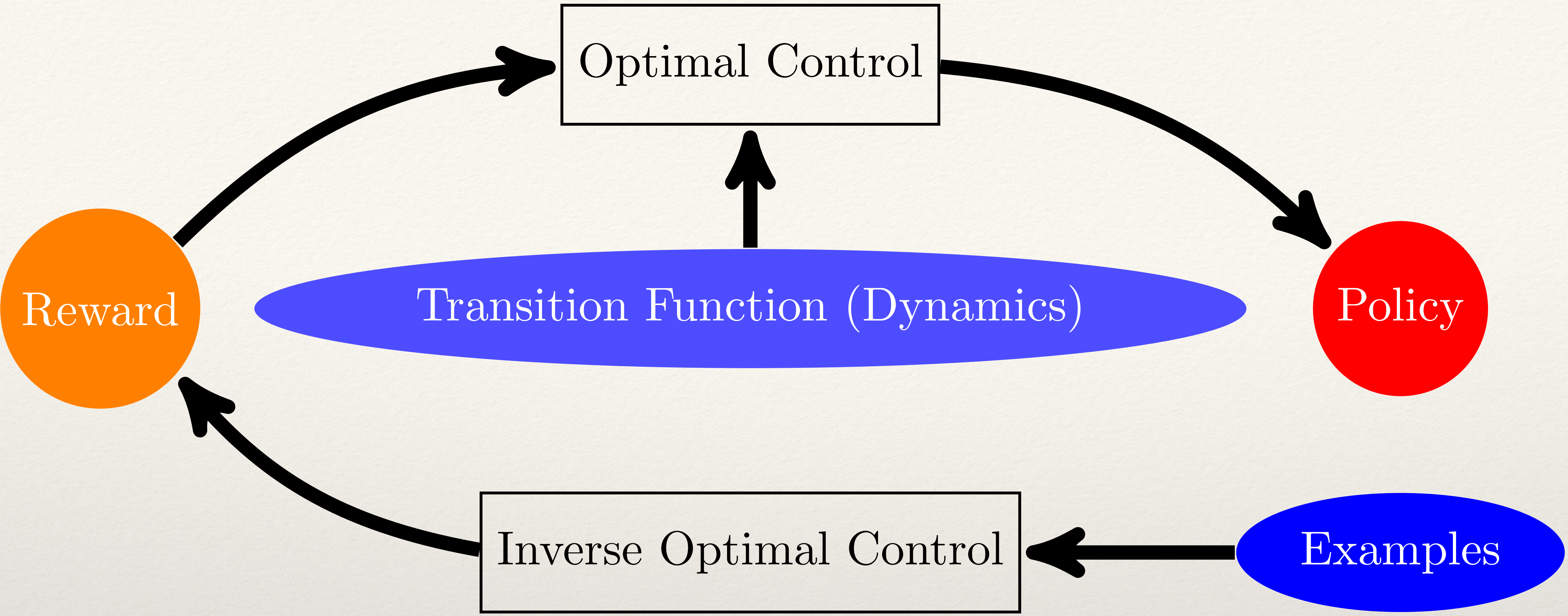
Average number of successful returns



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



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

Assumption: 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 **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 $\boldsymbol{P}$, a probability distribution on state-action trajectories 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 trajectories 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

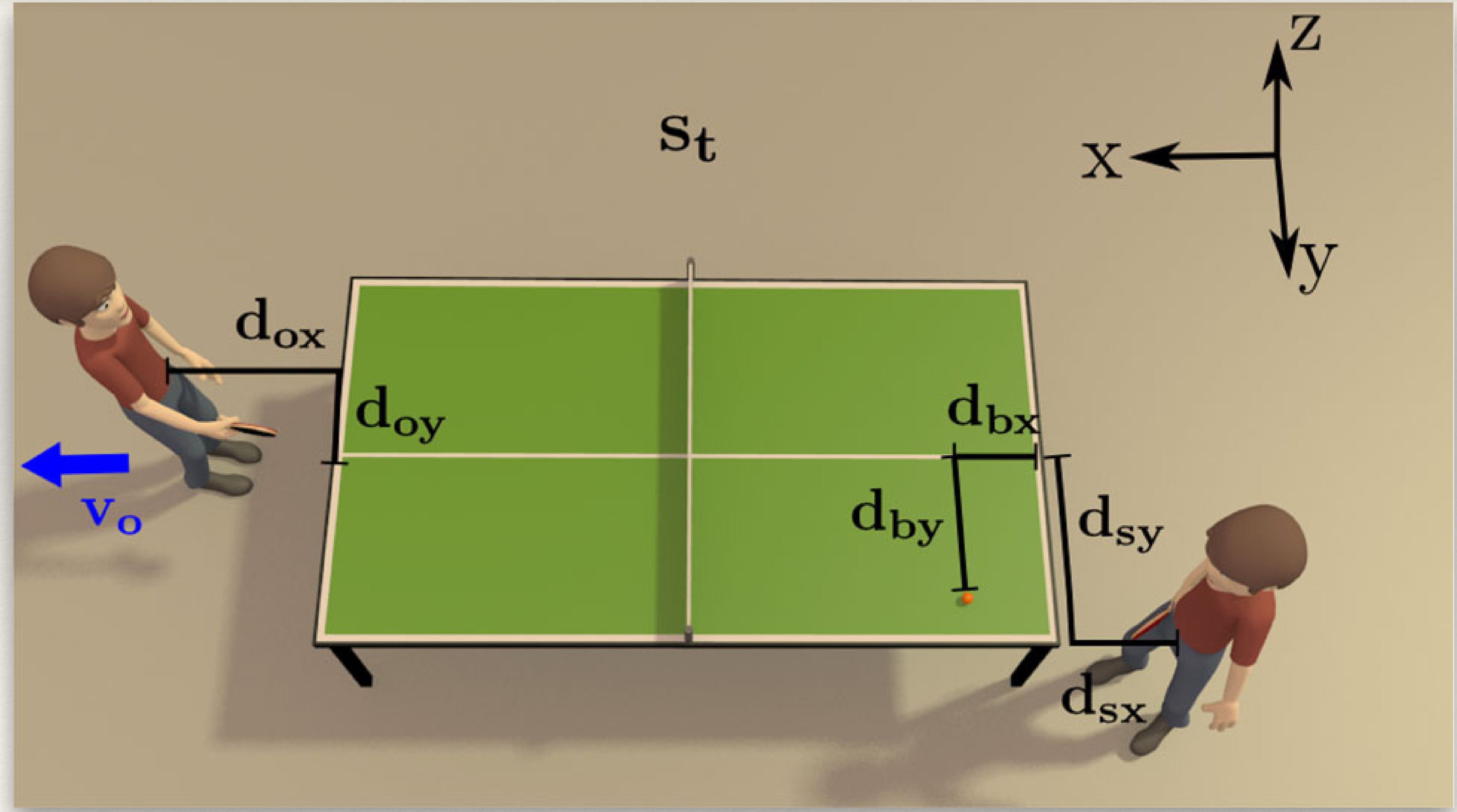
Robot Table Tennis: Learning to Imitate an Expert Player

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.

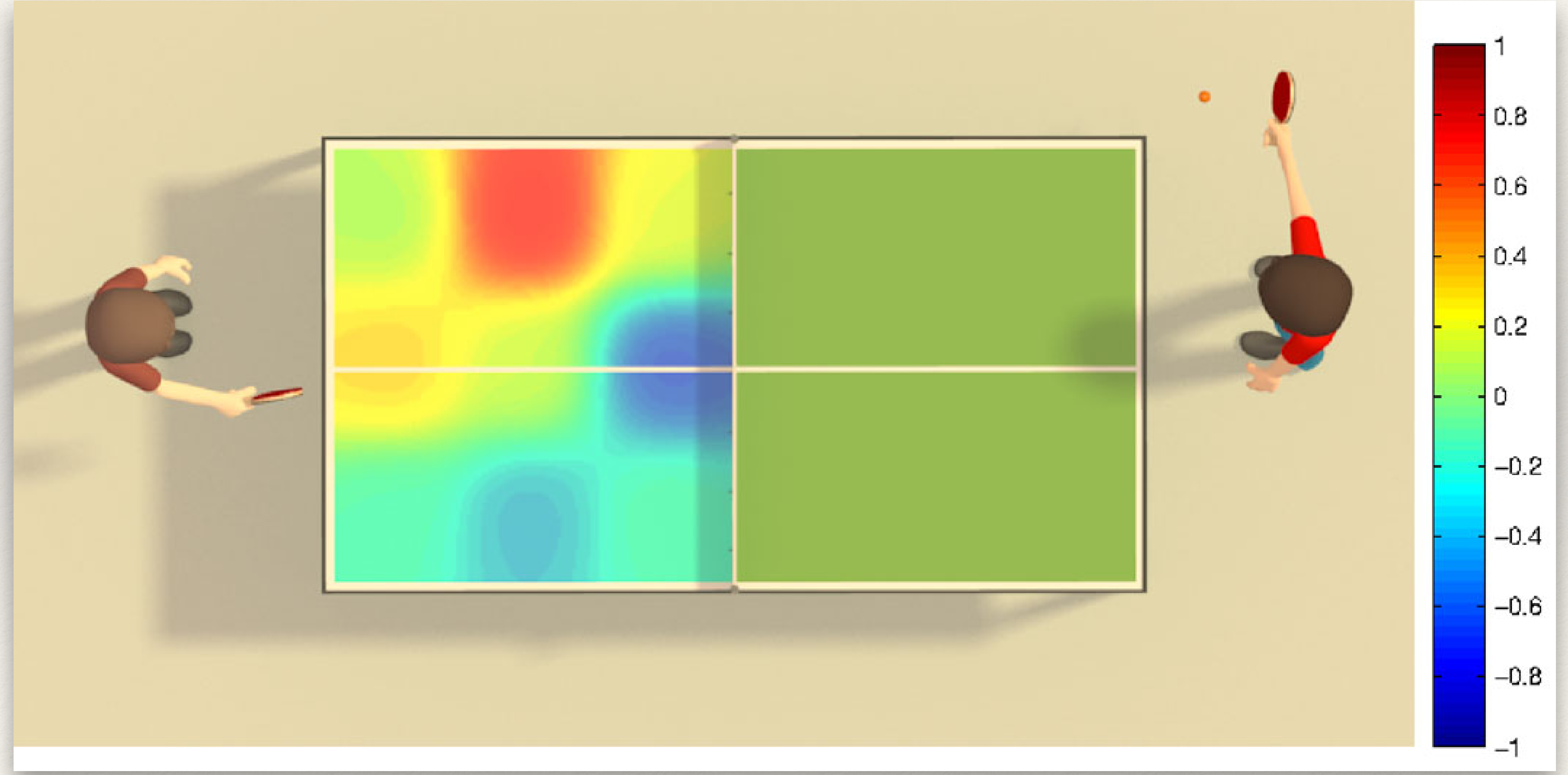
Robot Table Tennis: Learning to Imitate an Expert Player

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

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

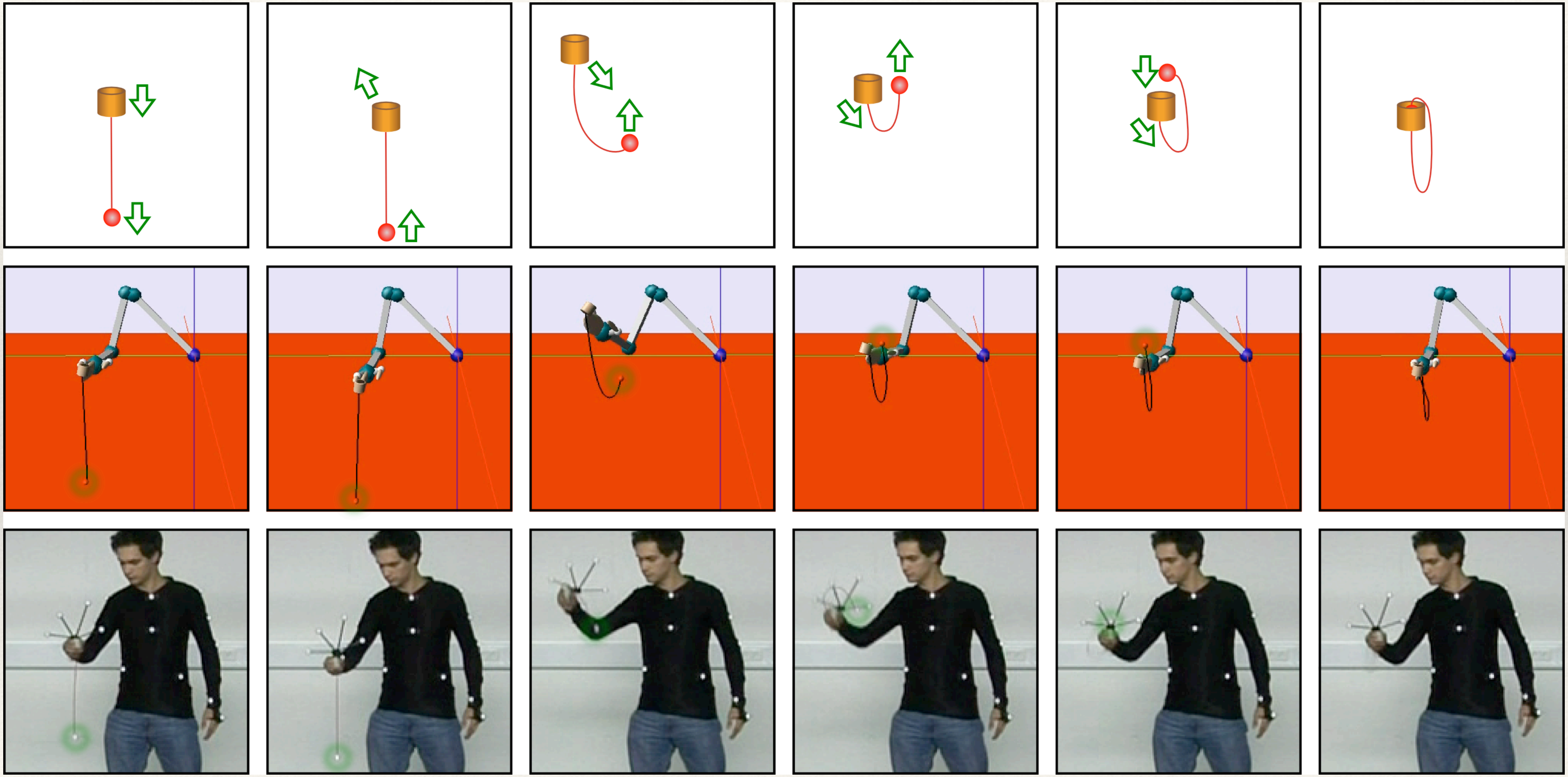
Average reward of each player is predicted from just the way she/he plays (without looking at the scores)

	horizon	Naive 1	Naive 2	Naive 3	Naive 4	Naive 5	Skilled
Average reward difference	1	1.30	0.04	1.17	0.91	0.74	0.30
with respect to the expert	2	1.20	0.07	1.22	0.87	0.72	0.33
	3	1.16	0.07	1.24	0.86	0.71	0.33
Average reward differences	2	0.91	−0.21	0.92	0.57	0.38	−0.12
directly before terminal state	3	1.12	0.04	1.23	0.89	0.76	0.24

The skilled player is the most similar to the expert in terms of predicted rewards

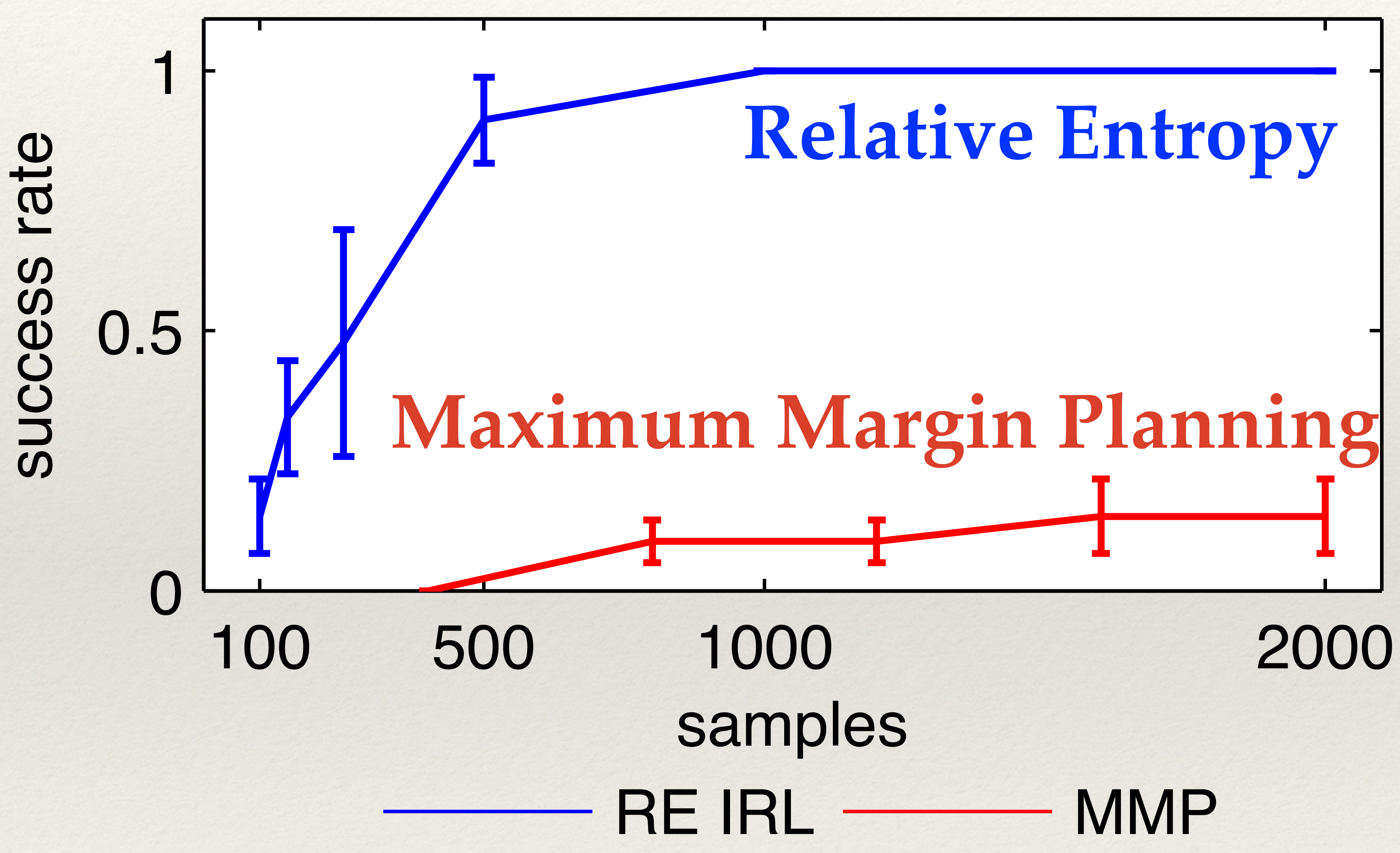
Example: Ball-in-a-Cup game (*Kendama*)

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)*.

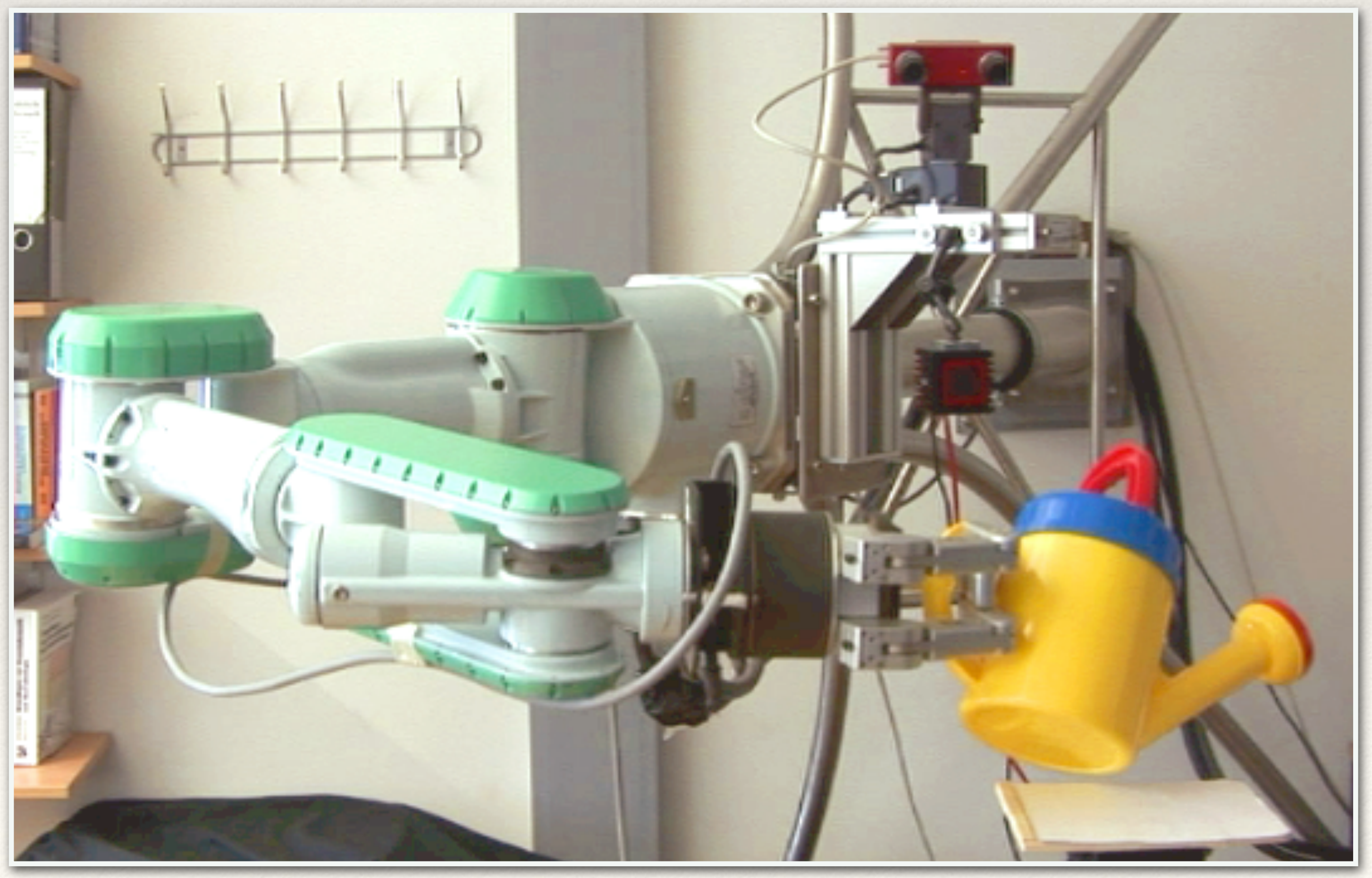
Outline

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

Purposeful Grasping

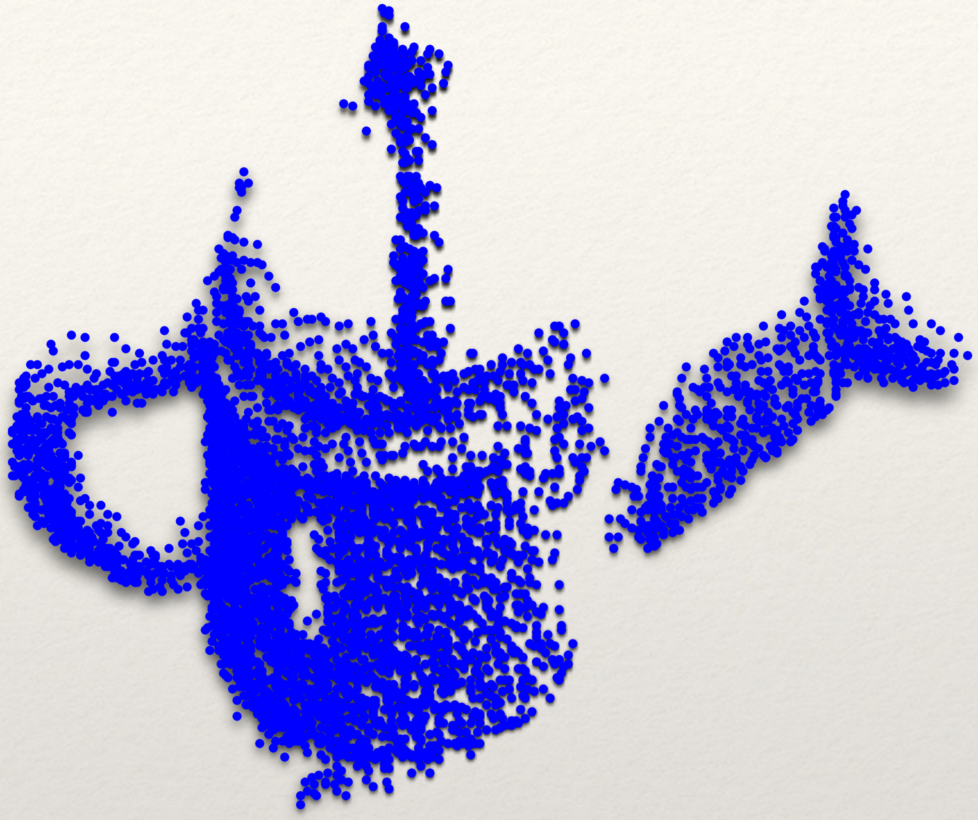
Parameters of a grasping action (position and rotation of the hand) should be **chosen depending on the intended goal**.

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



Purposeful Grasping

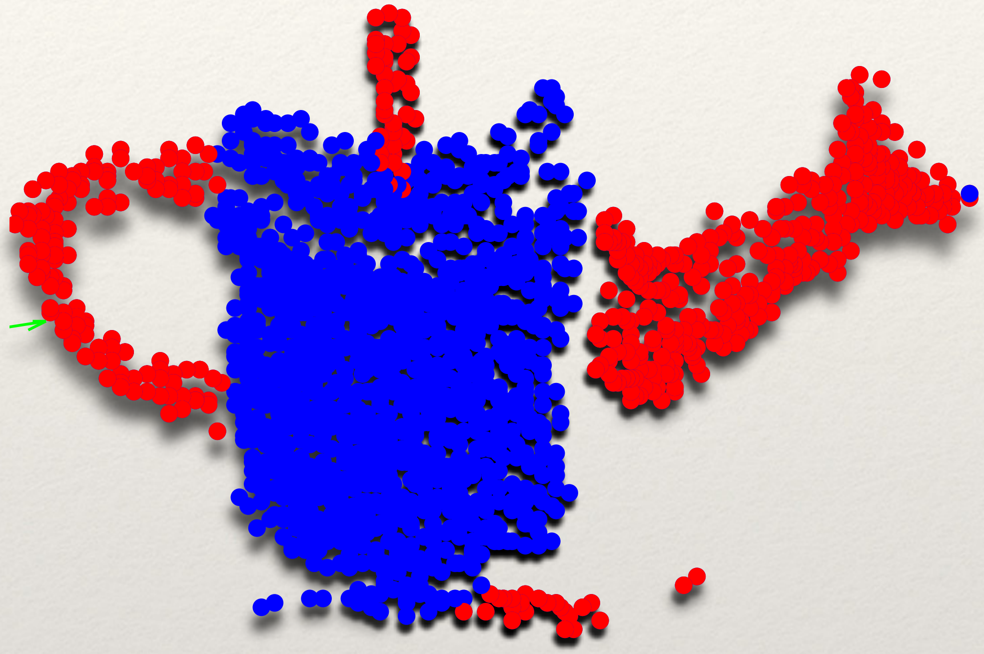
Barrett® hand



3D image of an unknown object

Purposeful Grasping

Barrett® hand

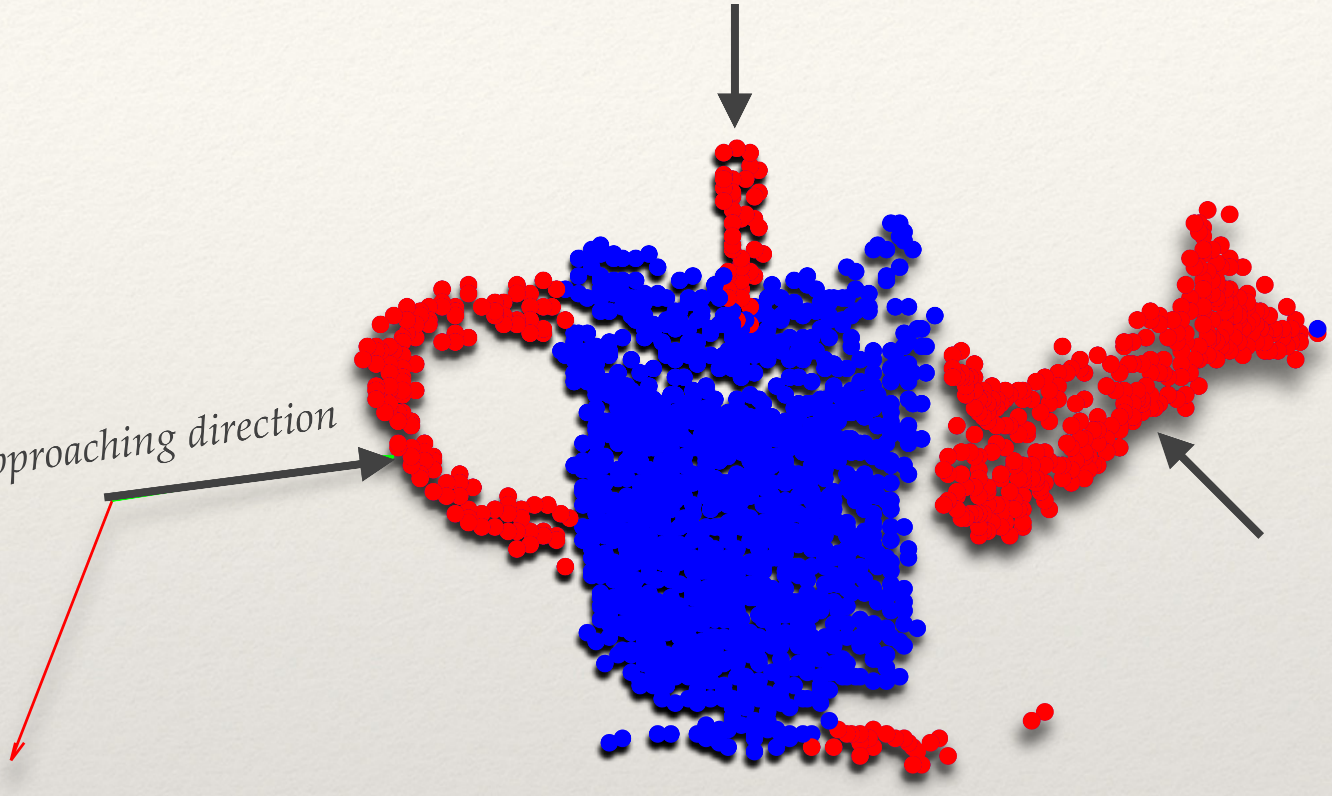


segmented object

Vision: Segment the object into parts

Purposeful Grasping

Barrett® hand



segmented object

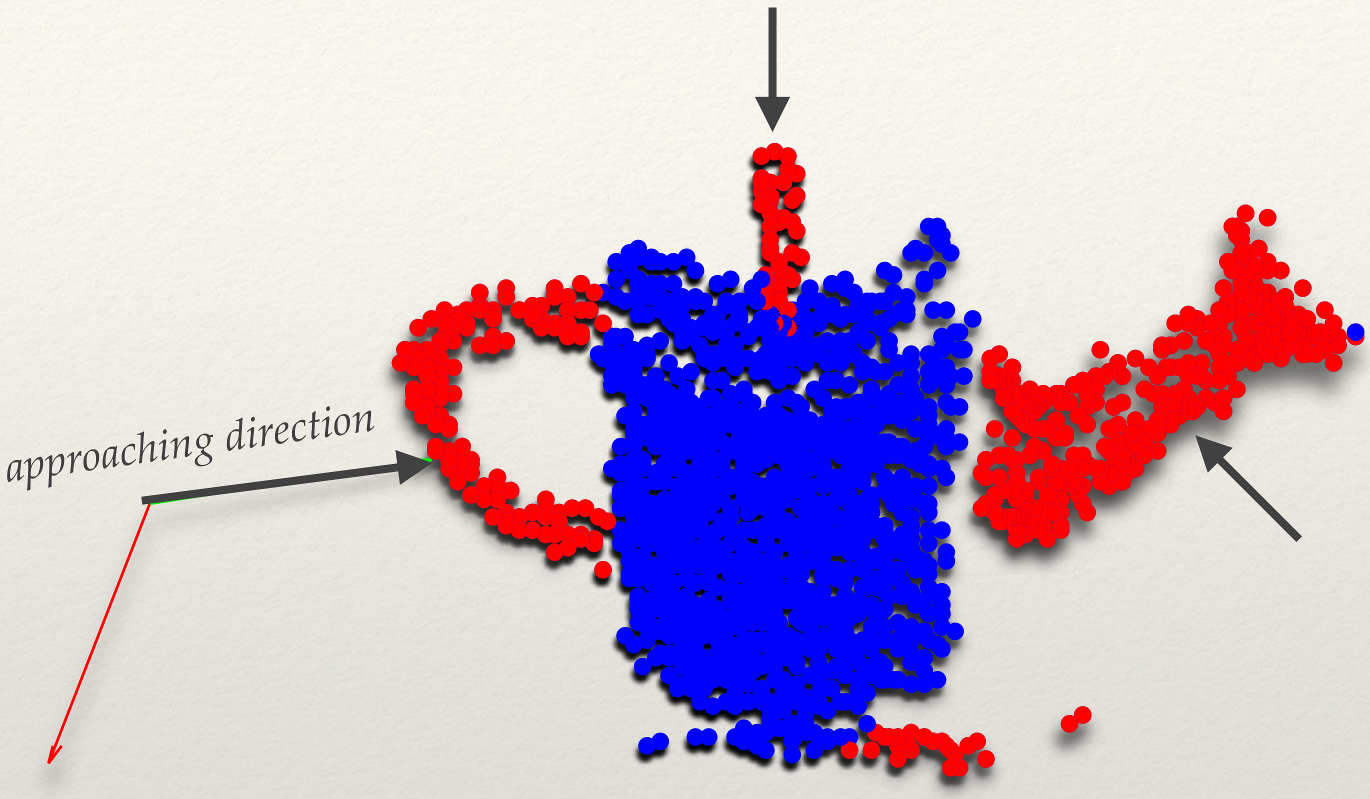
Vision: Segment the object into parts

↓

Planning: Simulate grasping actions for each part

Purposeful Grasping

Barrett® hand



segmented object

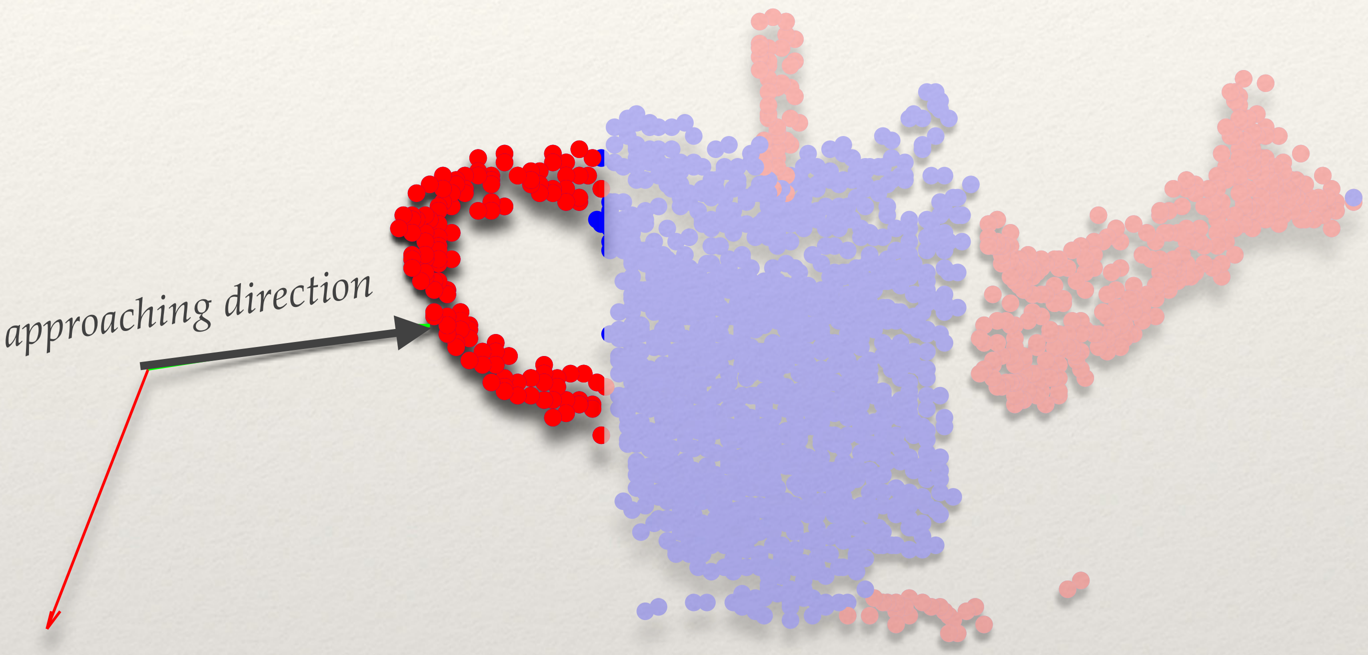
Vision: Segment the object into parts

Planning: Simulate grasping actions for each part

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

Purposeful Grasping

Barrett® hand



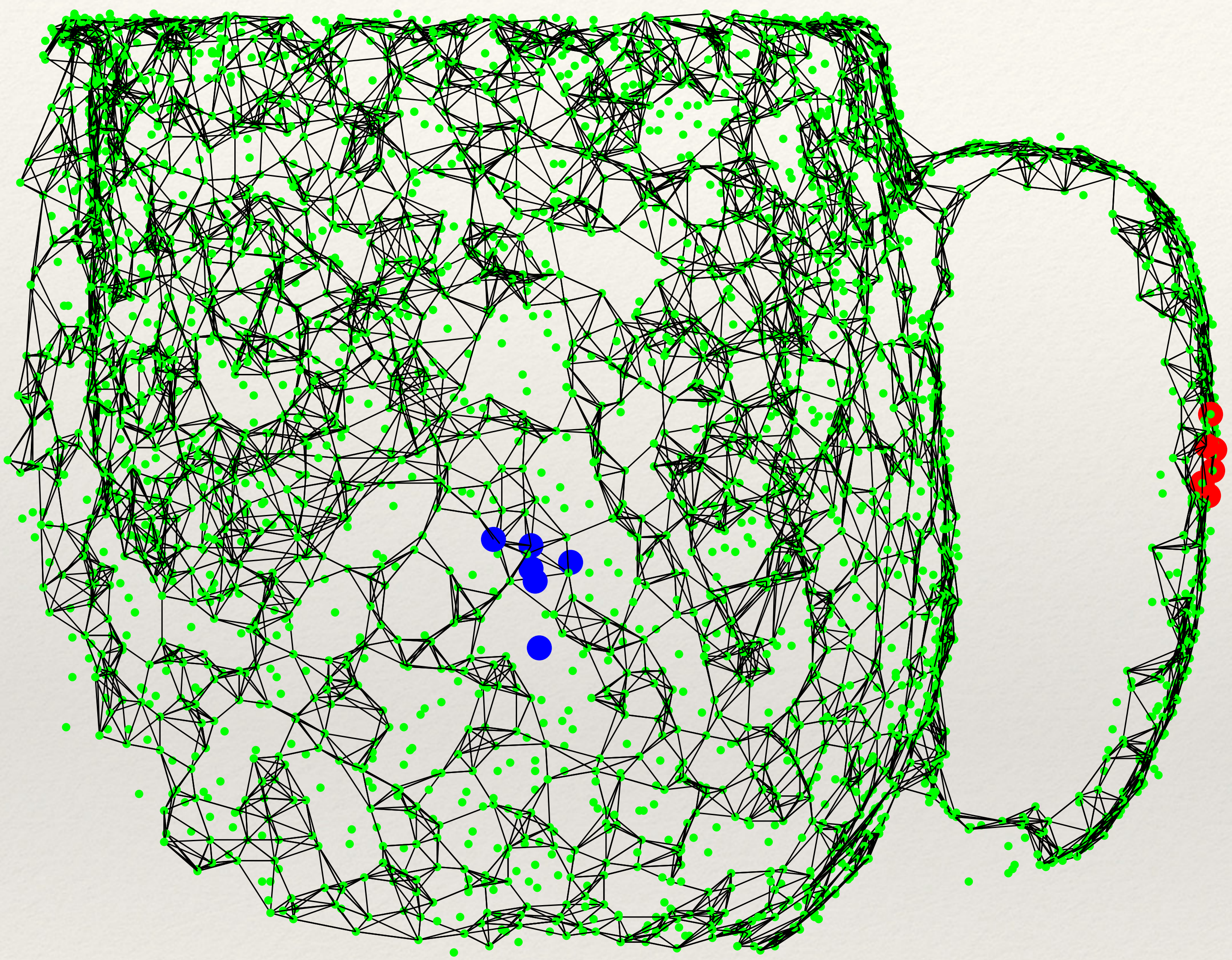
segmented object

Vision: Segment the object into parts

Planning: Simulate grasping actions for each part

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

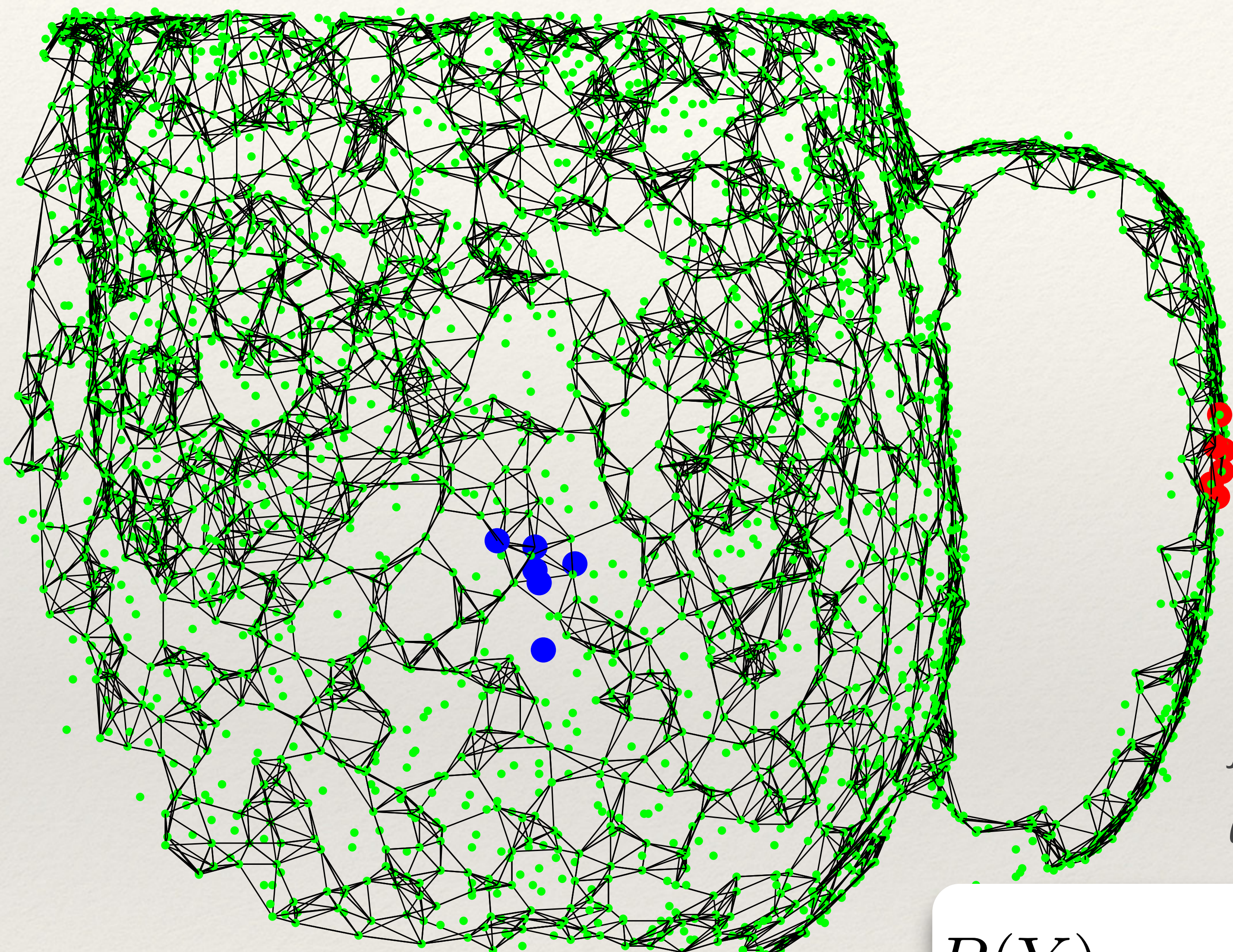
Learning Grasping Points with Associative Markov Networks



An object is represented as a k-nearest neighbor graph $(\mathcal{V}, \mathcal{E})$

Each node in the graph can be labeled as a ***success*** or ***failure*** with $y_i \in \{\mathbf{1}, \mathbf{-1}\}$

Learning Grasping Points with Associative Markov Networks



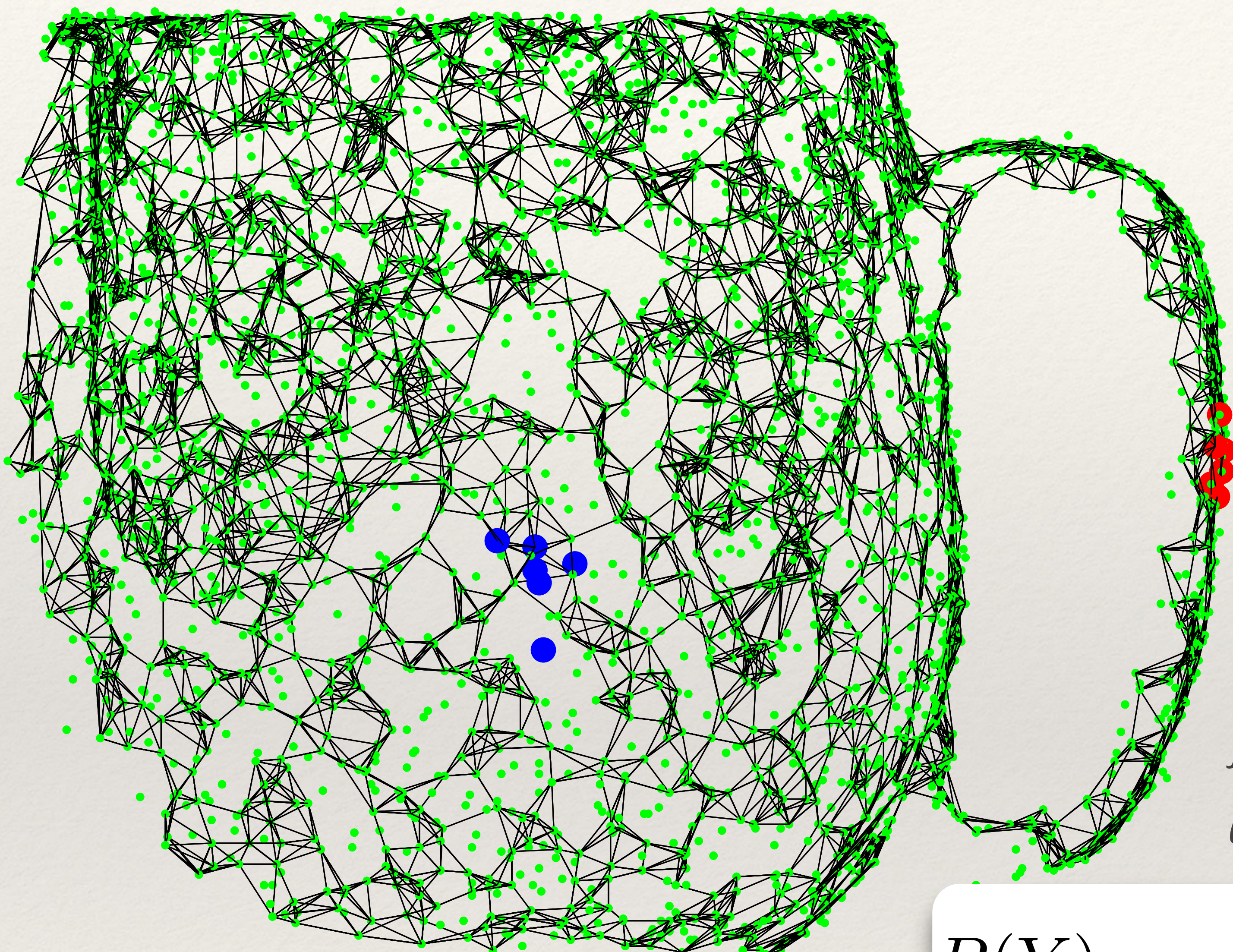
An object is represented as a k-nearest neighbor graph $(\mathcal{V}, \mathcal{E})$

Each node in the graph can be labeled as a ***success*** or ***failure*** with $y_i \in \{\mathbf{1}, -\mathbf{1}\}$

Joint distribution of the labels of all points $Y = \{y_1, y_2, \ldots, y_n\}$

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

Learning Grasping Points with Associative Markov Networks



An object is represented as a k-nearest neighbor graph $(\mathcal{V}, \mathcal{E})$

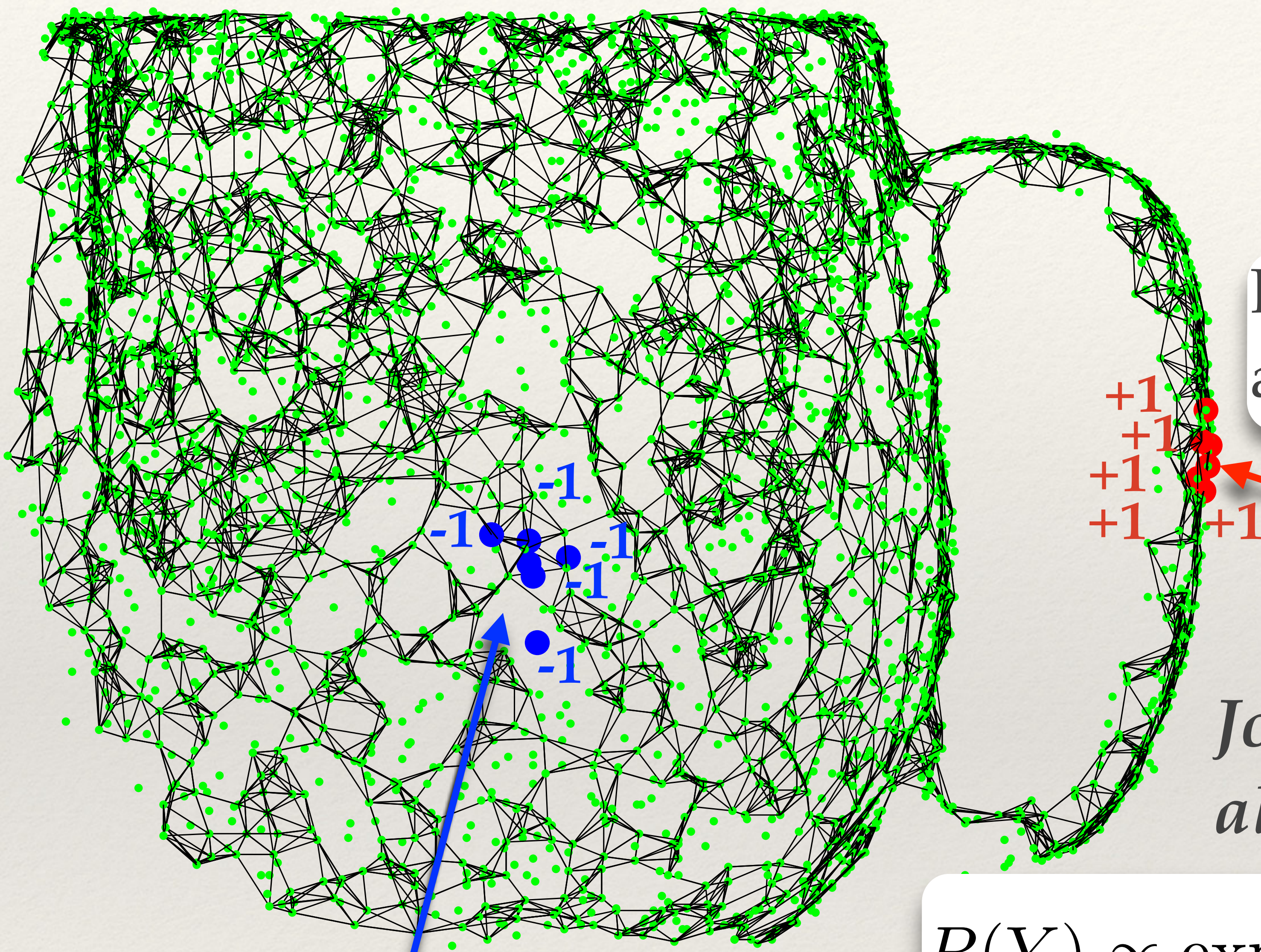
Each node in the graph can be labeled as a ***success*** or ***failure*** with $y_i \in \{\mathbf{1}, \mathbf{-1}\}$

Joint distribution of the labels of all points $Y = \{y_1, y_2, \ldots, y_n\}$

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

weight vectors

Learning Grasping Points with Associative Markov Networks



Failure examples

Success examples

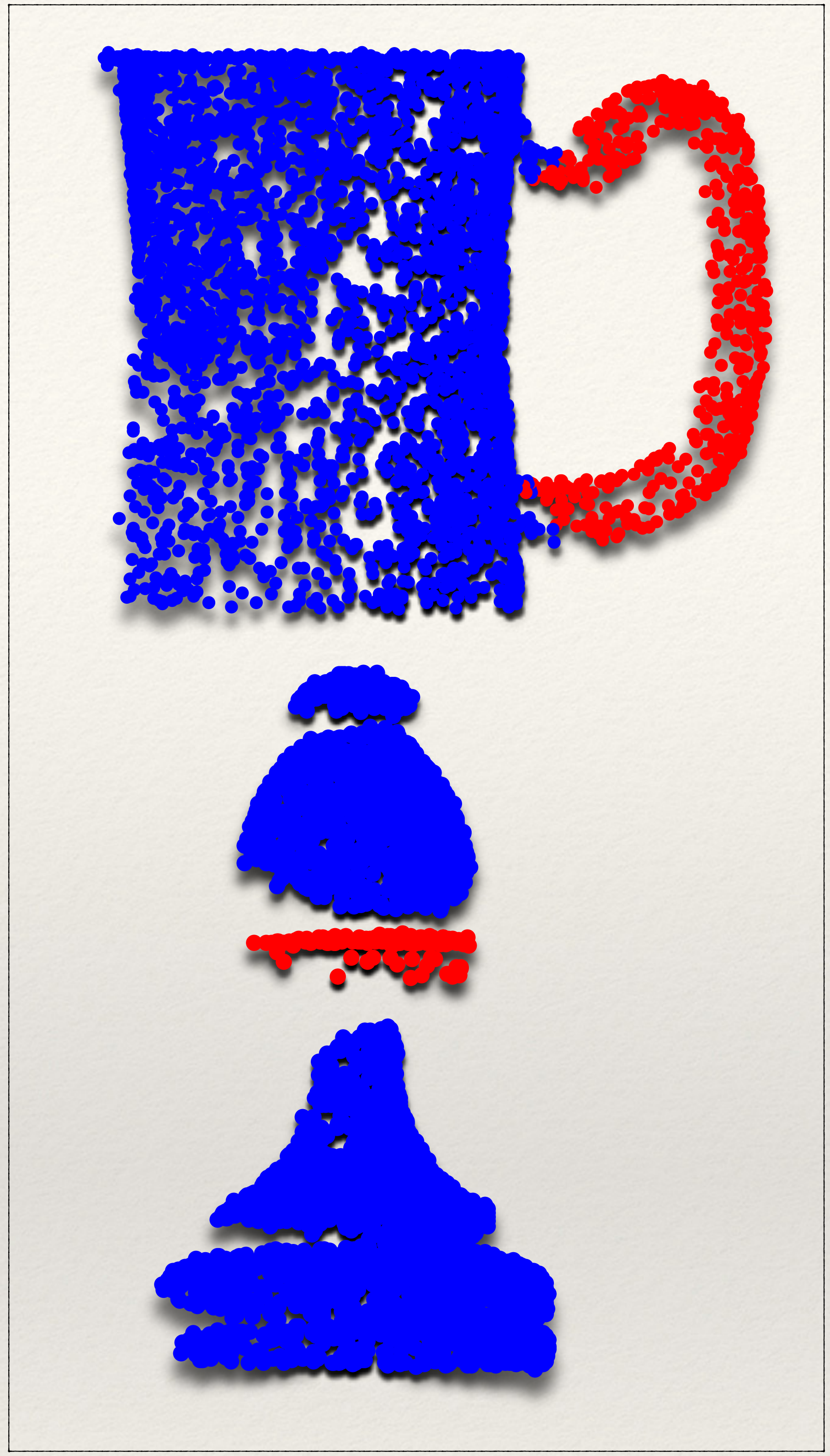
An object is represented as a k-nearest neighbor graph $(\mathcal{V}, \mathcal{E})$

Each node in the graph can be labeled as a ***success*** or ***failure*** with $y_i \in \{\mathbf{1}, -\mathbf{1}\}$

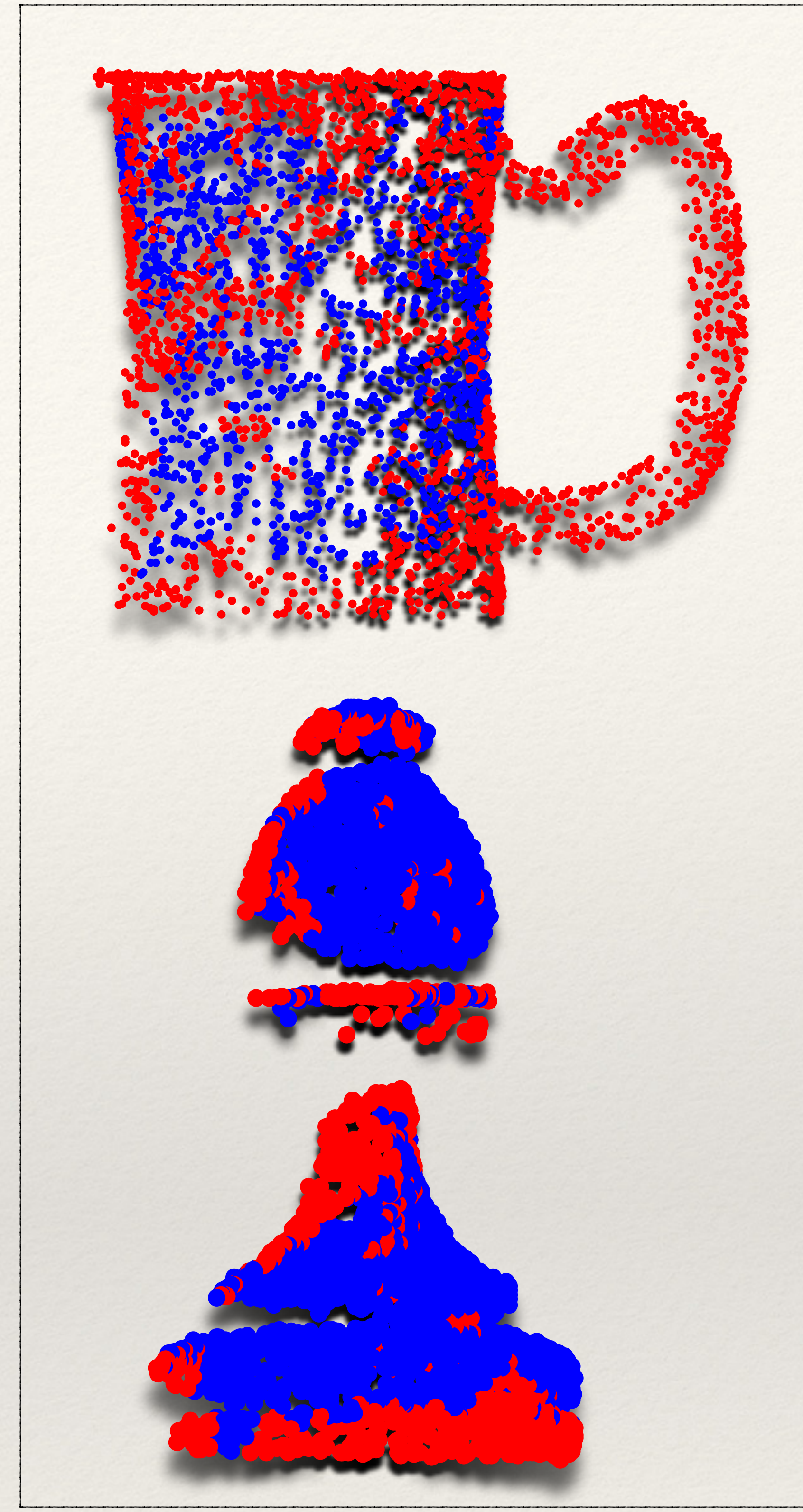
Joint distribution of the labels of all points $Y = \{y_1, y_2, \ldots, y_n\}$

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

weight vectors



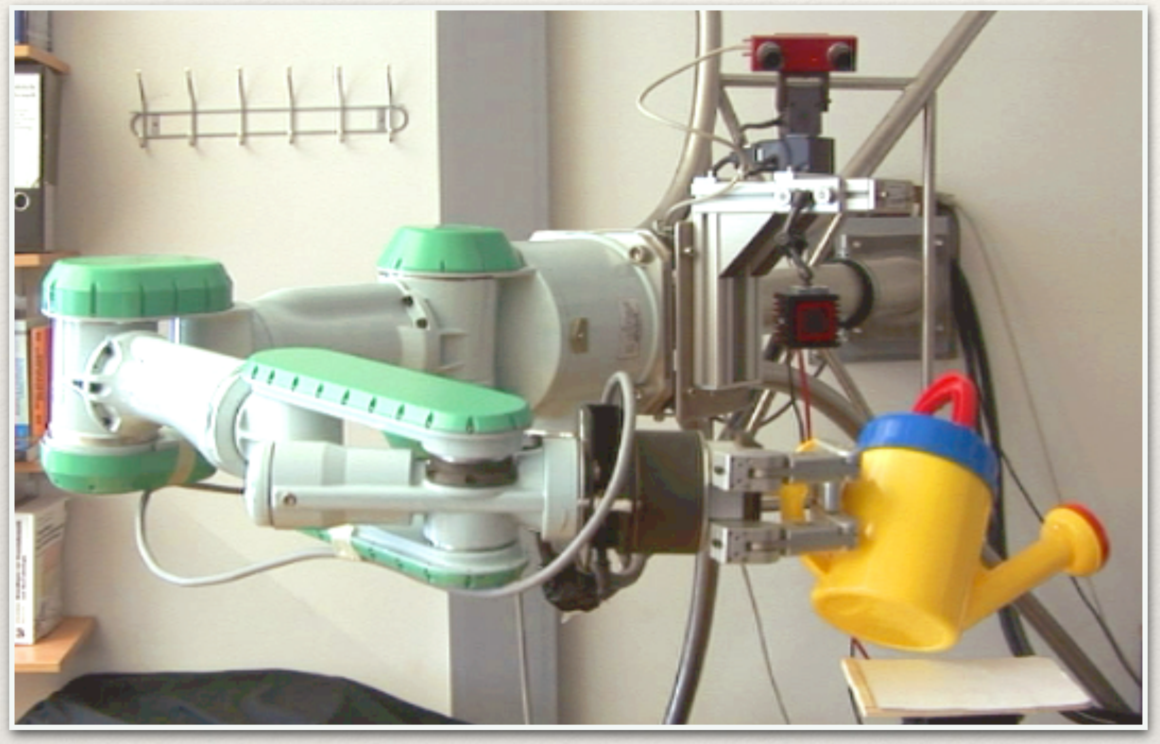
Associative Markov Networks



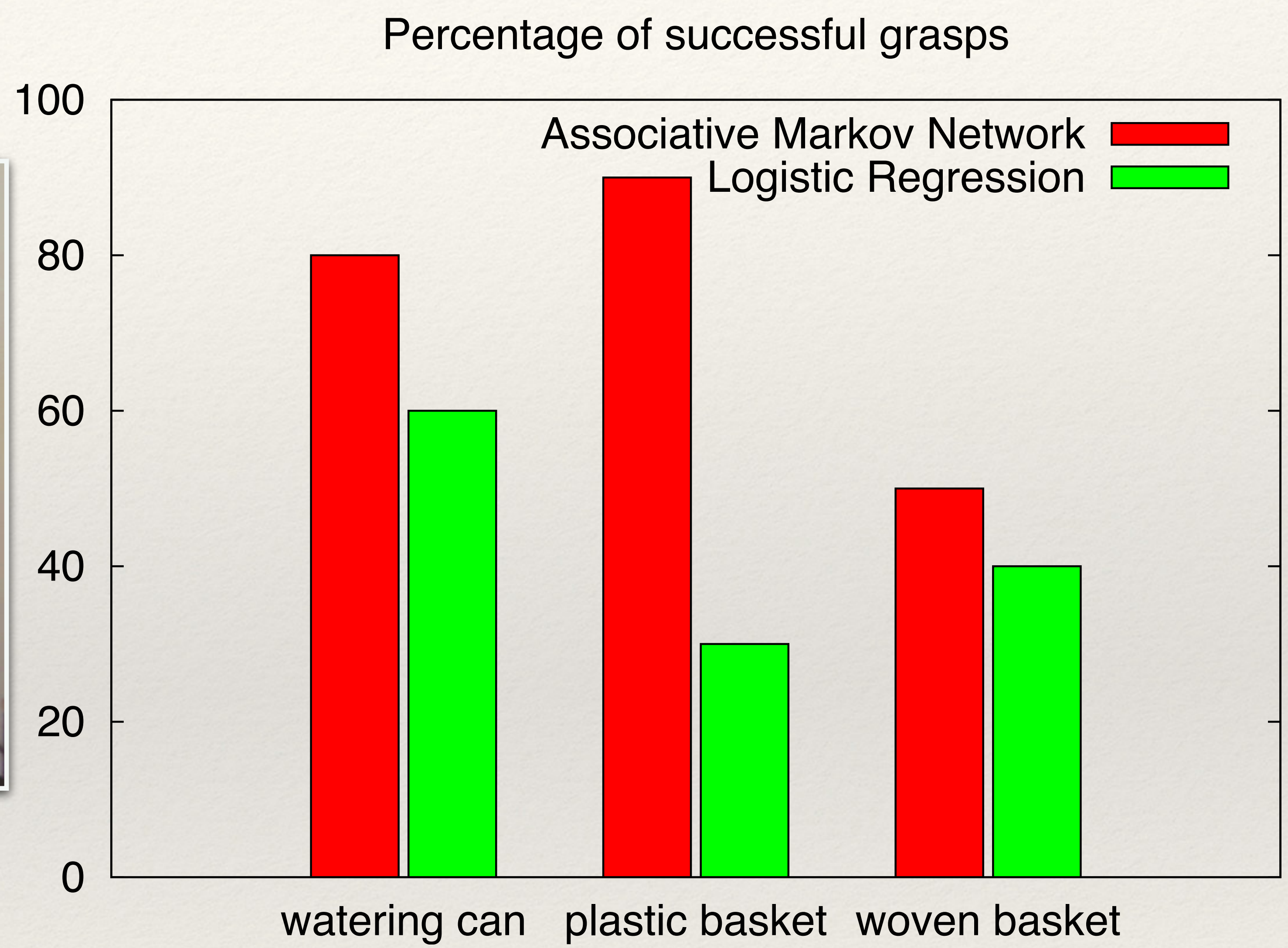
Logistic Regression

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

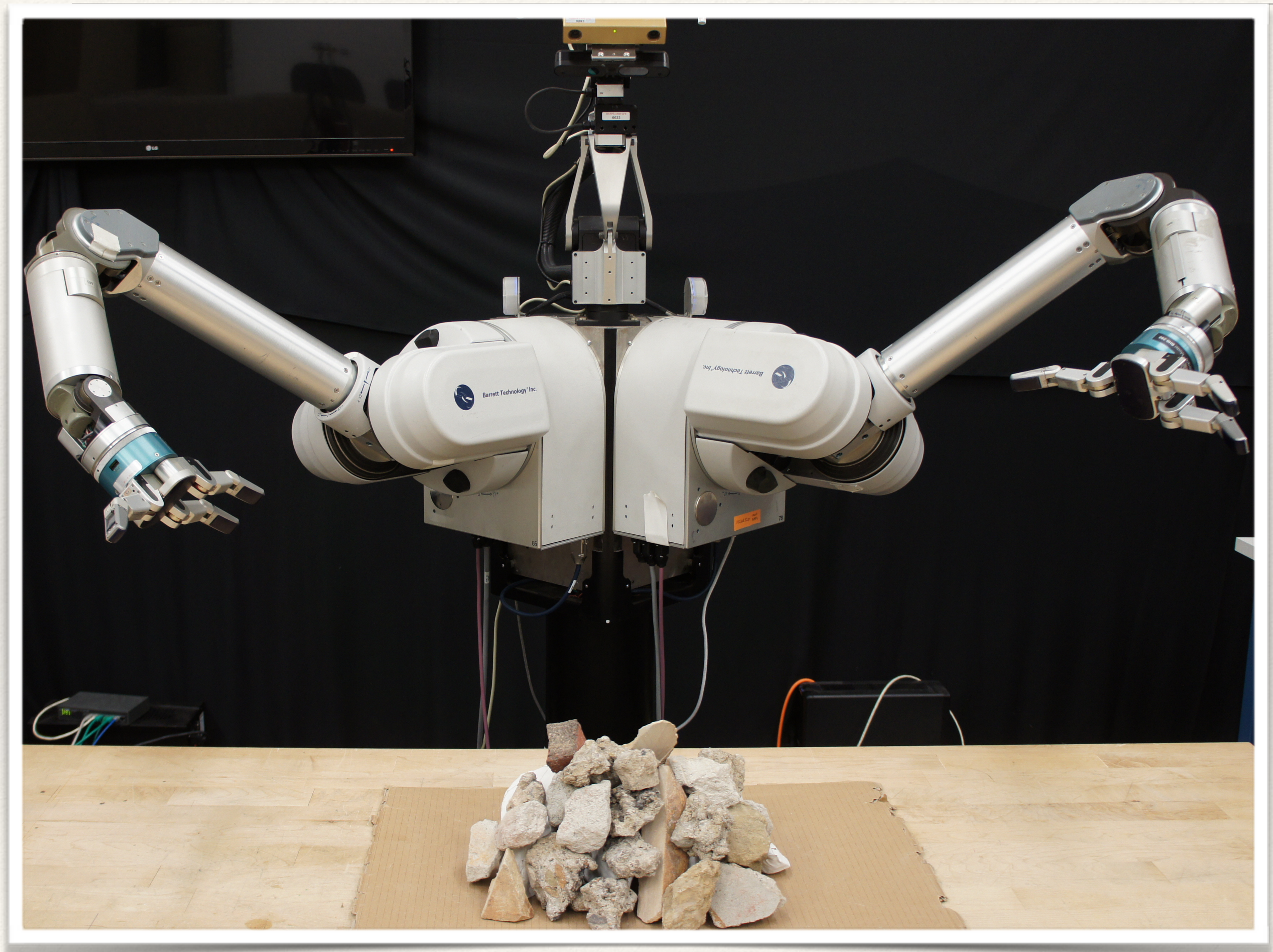
Source: AFP

© Roland Hoskins

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

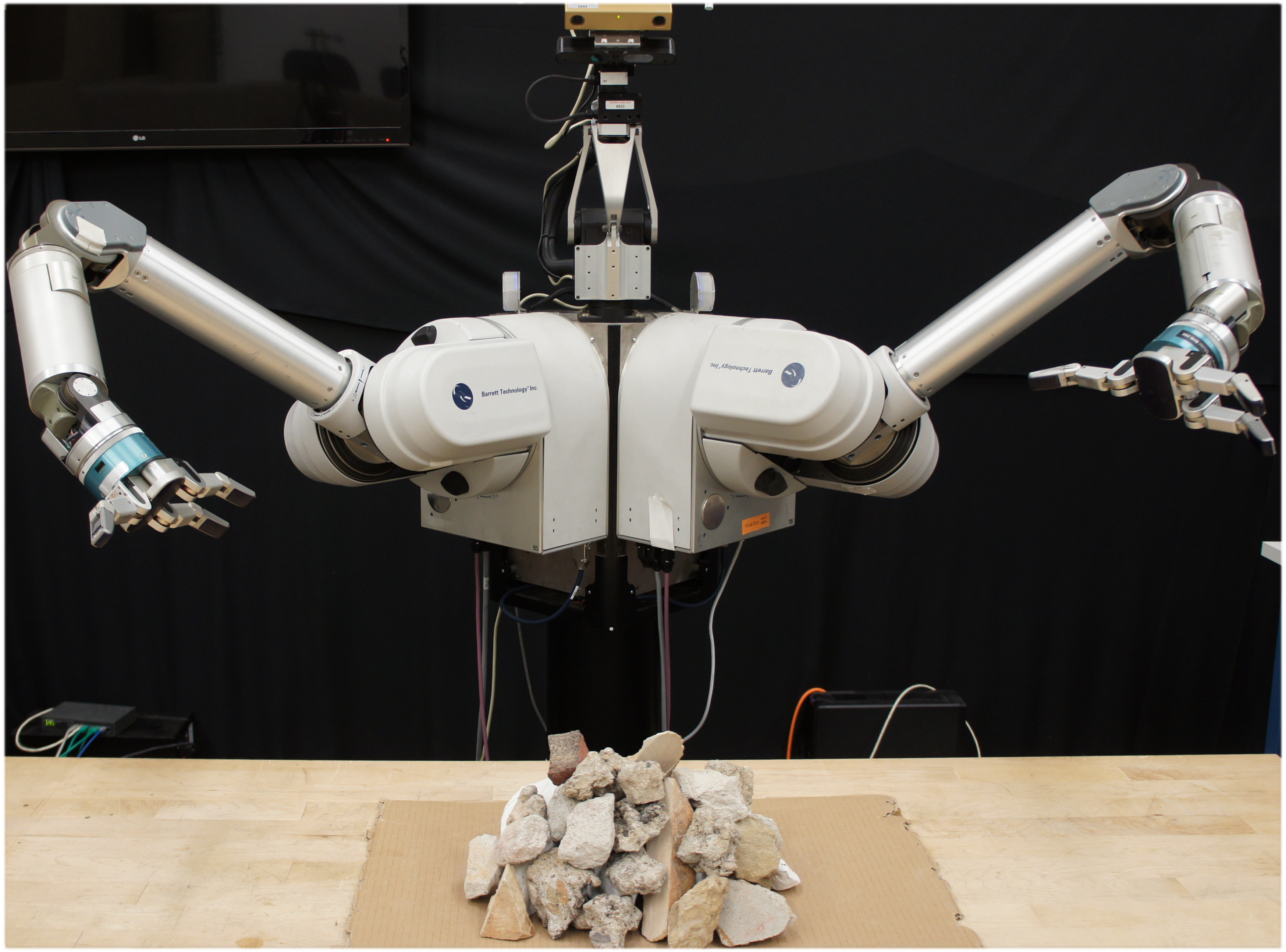
Tele-operation is tedious and requires a human expert

An Autonomous Robot for Rubble Removal



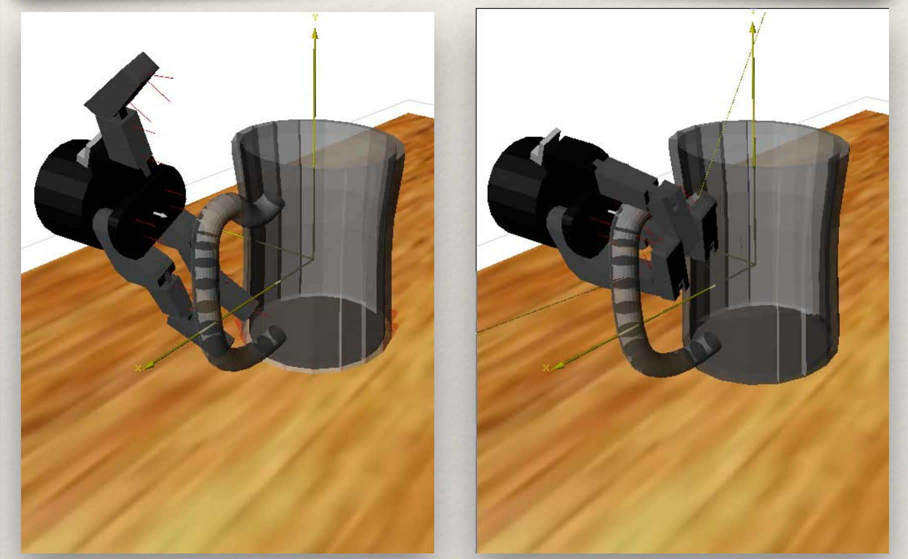
Two *Barrett*® arms and hands with a *Kinect*® camera

An Autonomous Robot for Rubble Removal



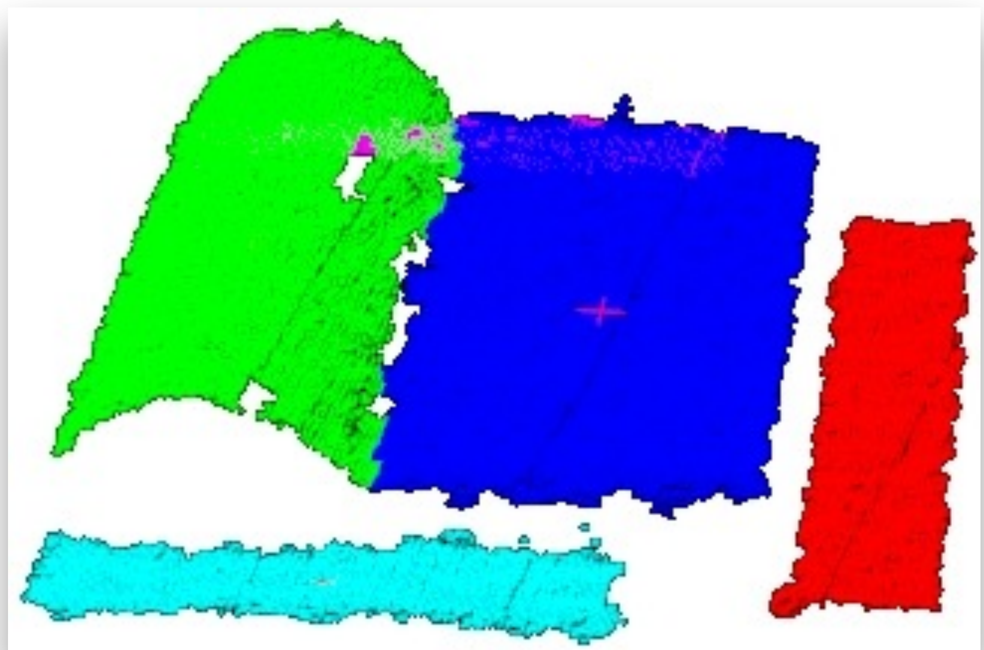
Two *Barrett*® arms and hands with a *Kinect*® camera

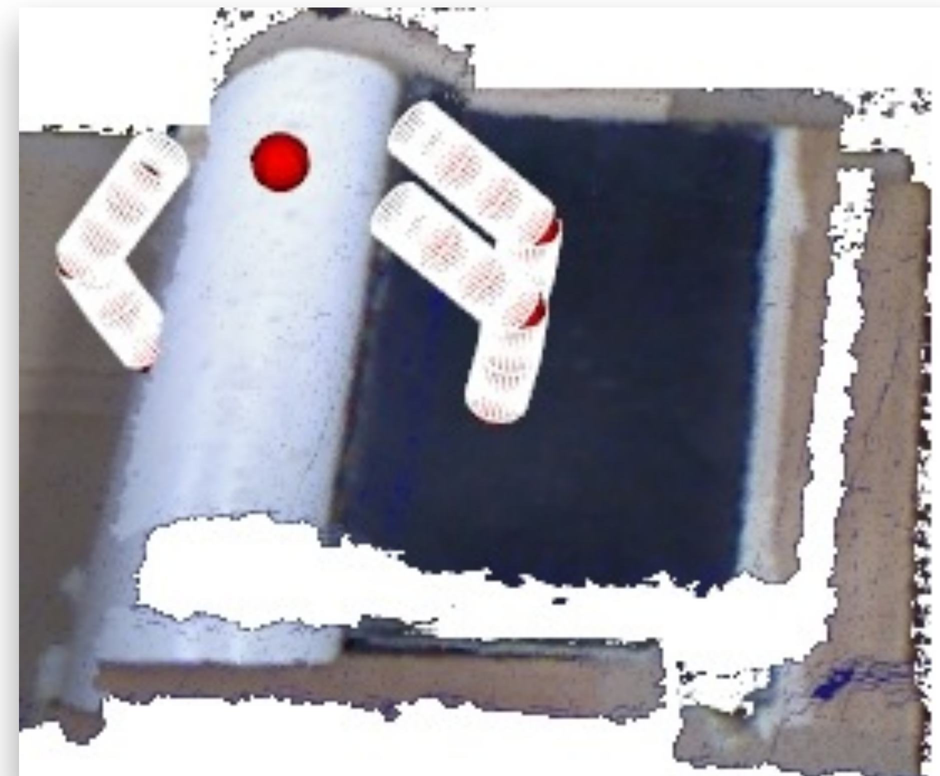
Most autonomous grasping techniques use **models** of the objects



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

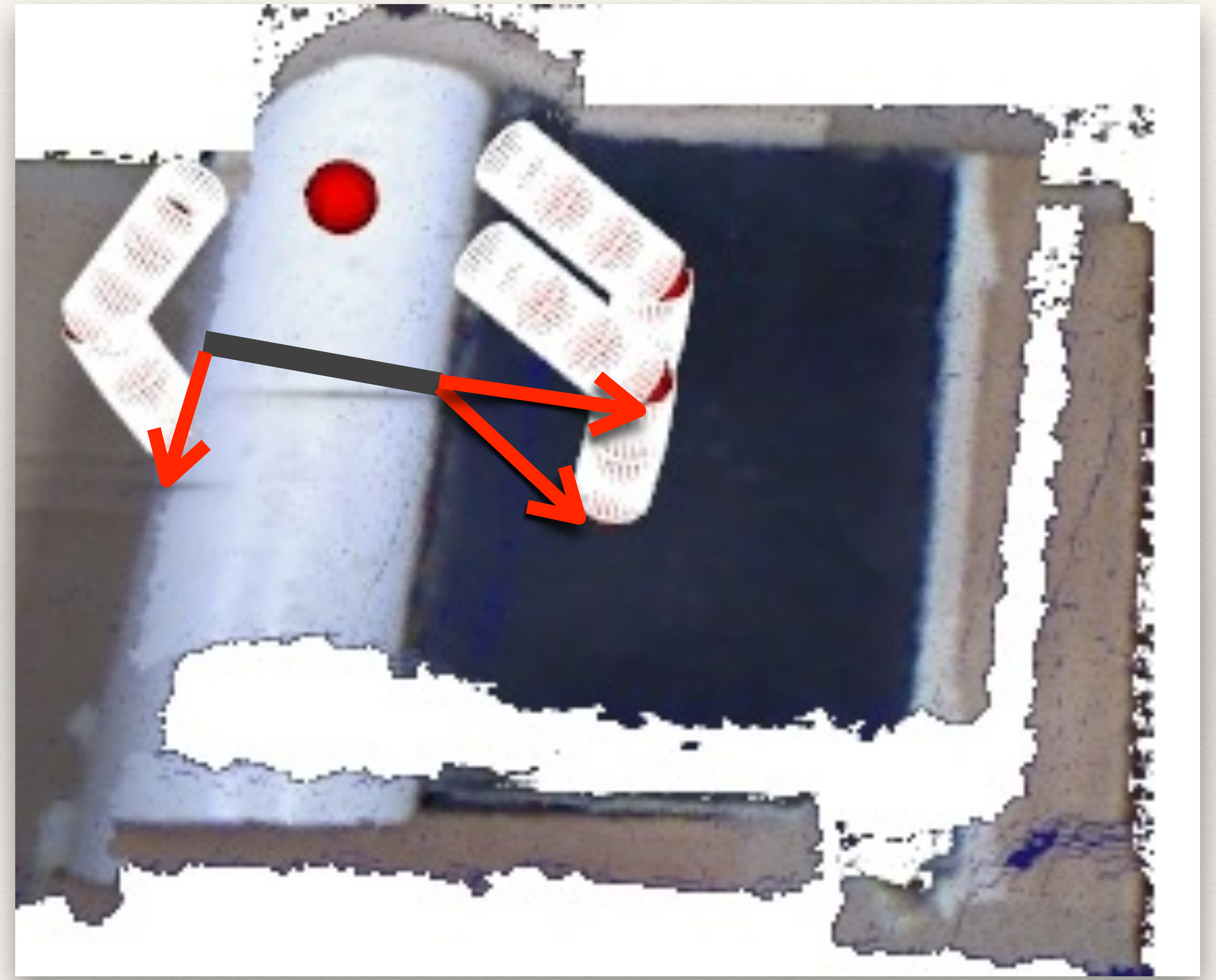
Grasping Regular Objects: A Simple Heuristic





Take a 3D image of the scene → Segment the 3D point cloud into facets by using the *mean-shift* algorithm → Simulate grasping actions for each facet by checking for collisions

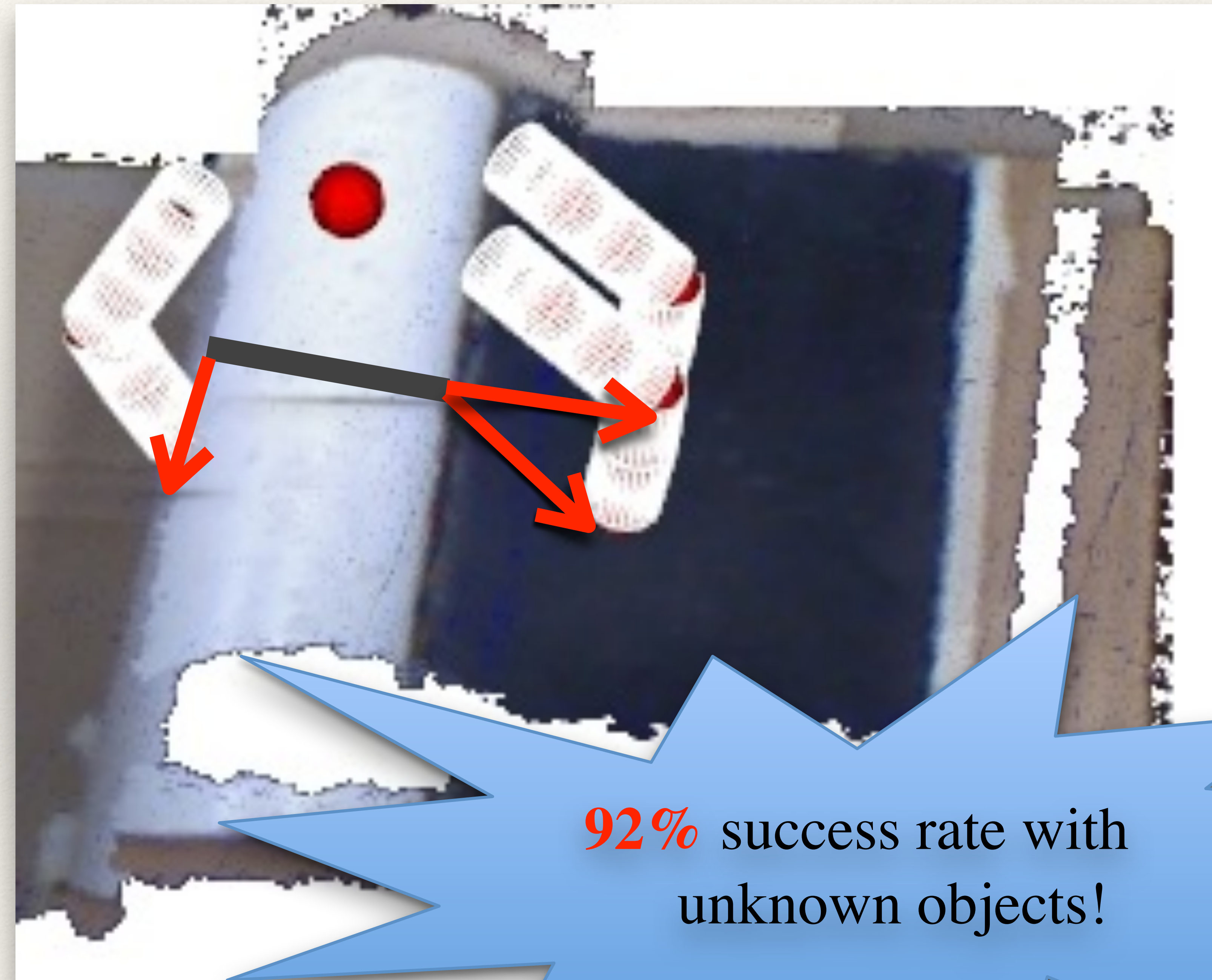
Grasping Regular Objects: A Simple Heuristic



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



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

92% success rate with unknown objects!

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

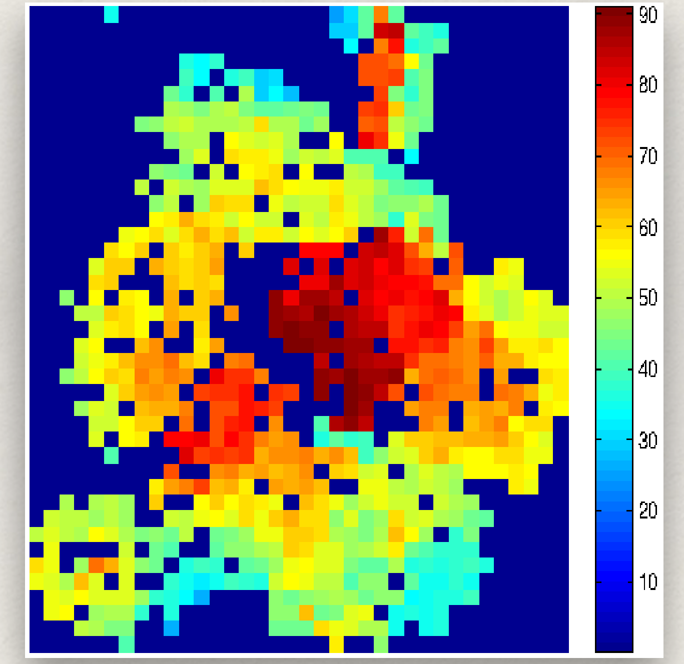
Simulation is **too slow** (0.1 second per action) for real-time requirements

Grasping Irregular Objects

Idea: Learn to predict the outcome of the simulation

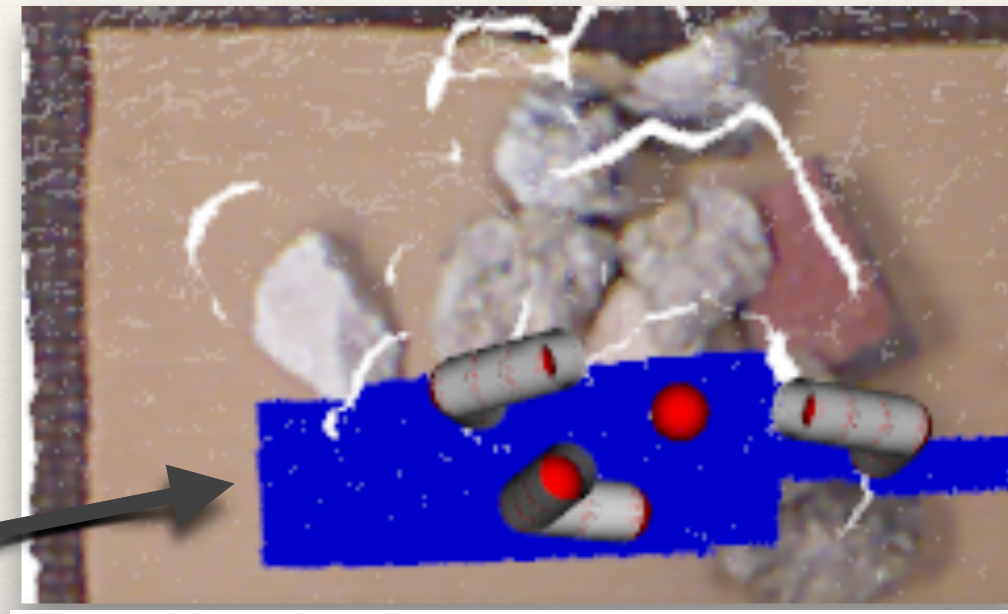
Pile of rocks

Real-time prediction

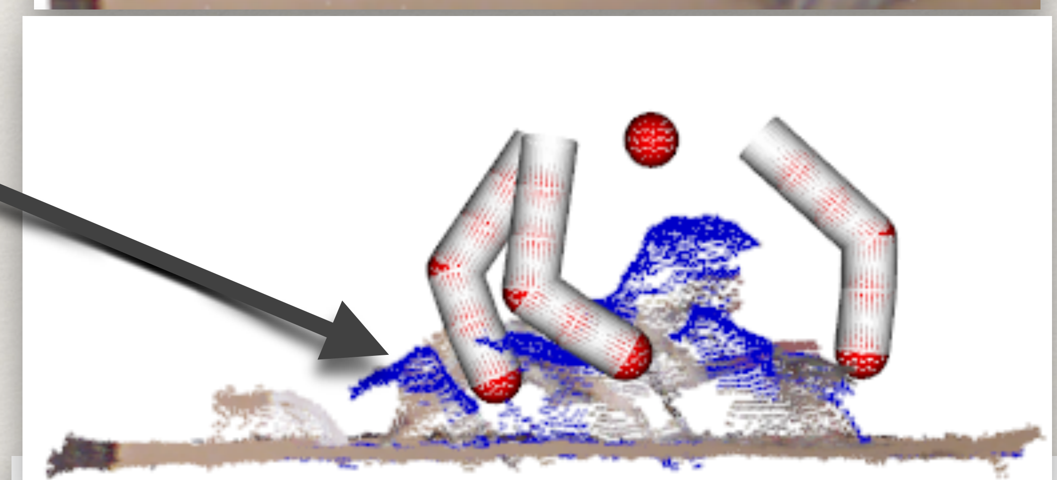


Predicted success probabilities using *k-Nearest Neighbors*

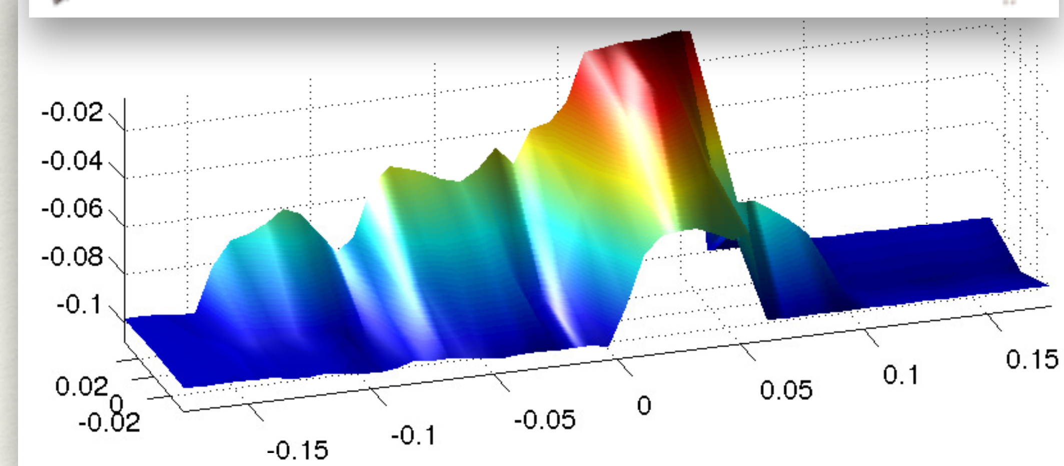
Features of Grasping Actions



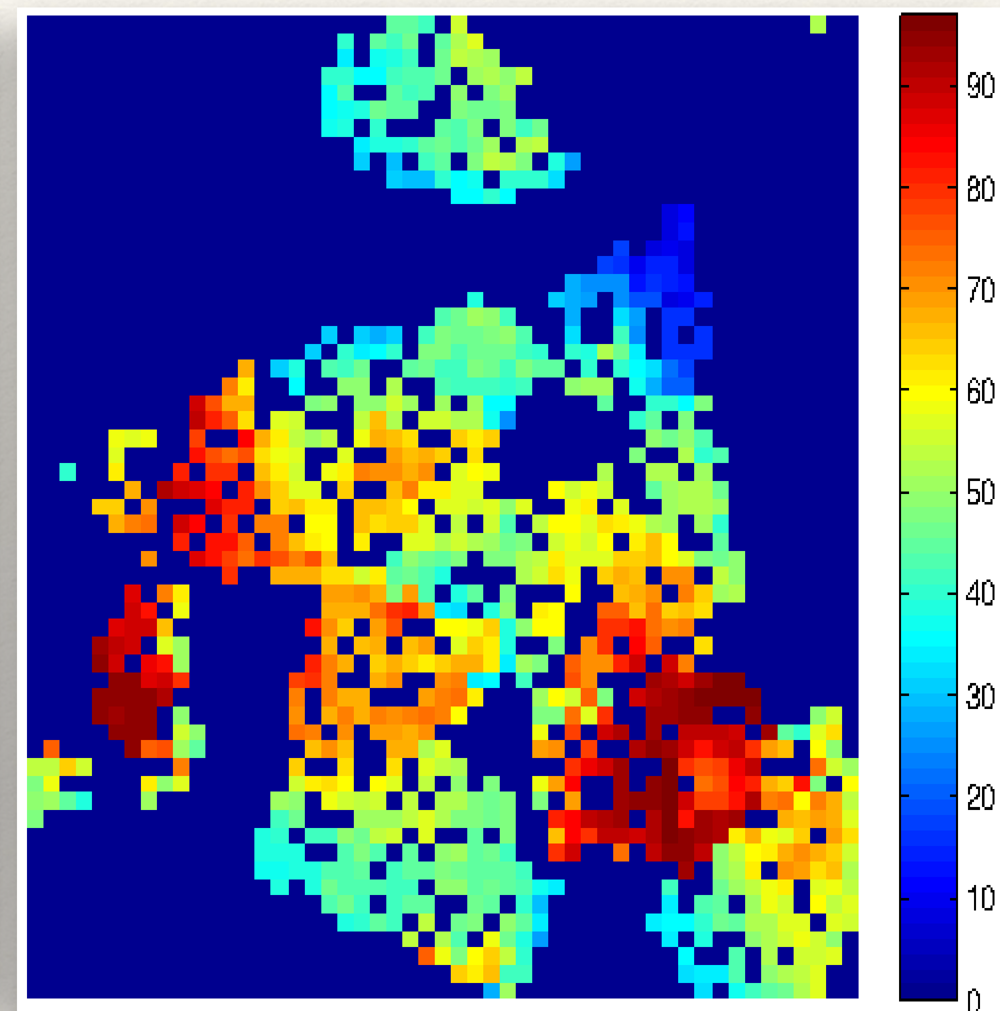
Extract all the points in the 3D cloud that may collide with the robot's hand (the **blue strip** in the figures)



Feature matrix: elevations of the points of collision

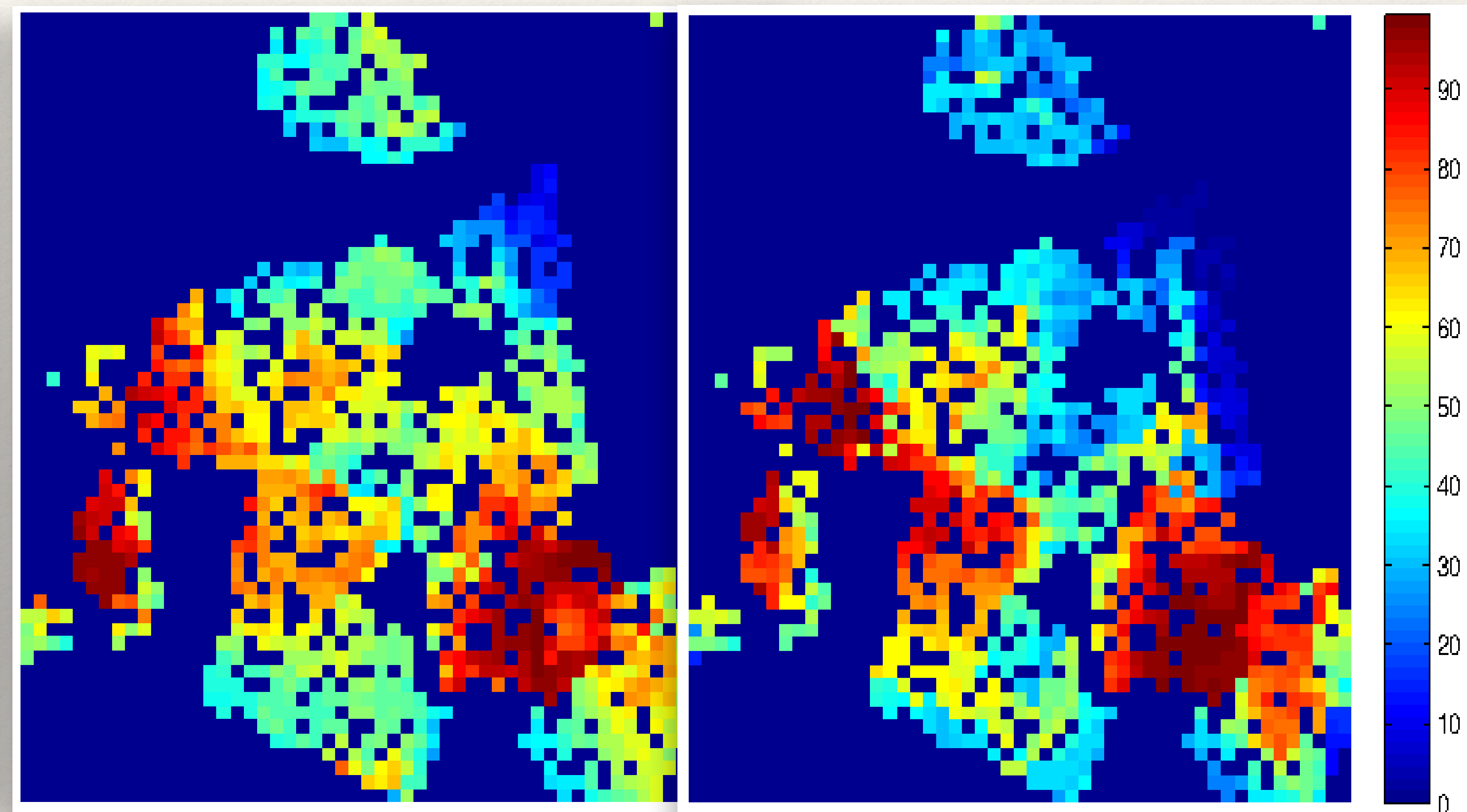


Predicting Success Probabilities of Grasping Action



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

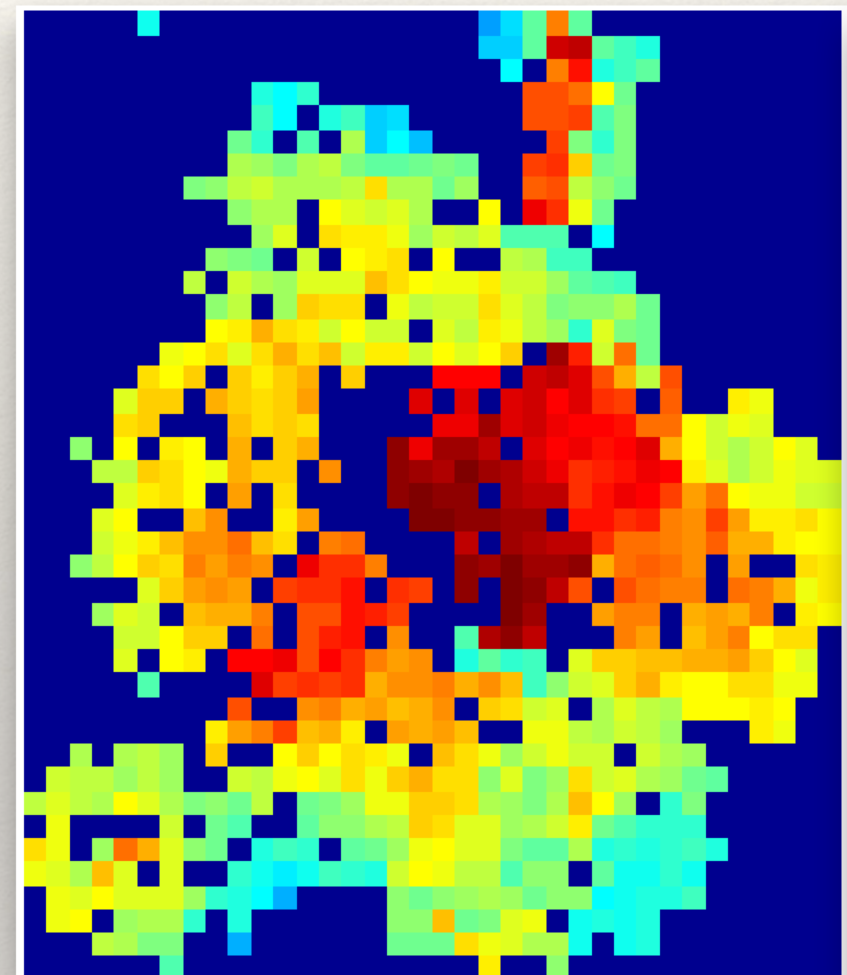
Predicting Success Probabilities of Grasping Action



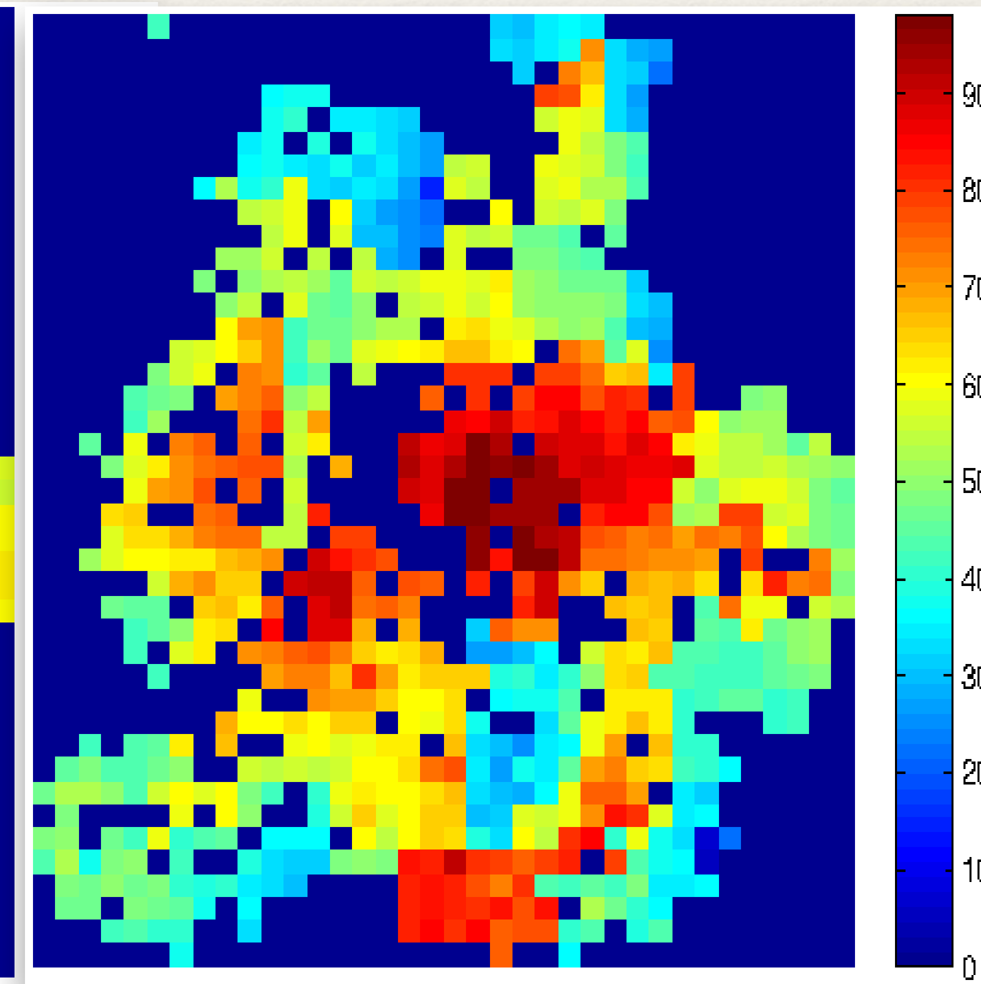
Learned probabilities, obtained in **2 seconds** with *k-NN*

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

Predicting Success Probabilities of Grasping Action

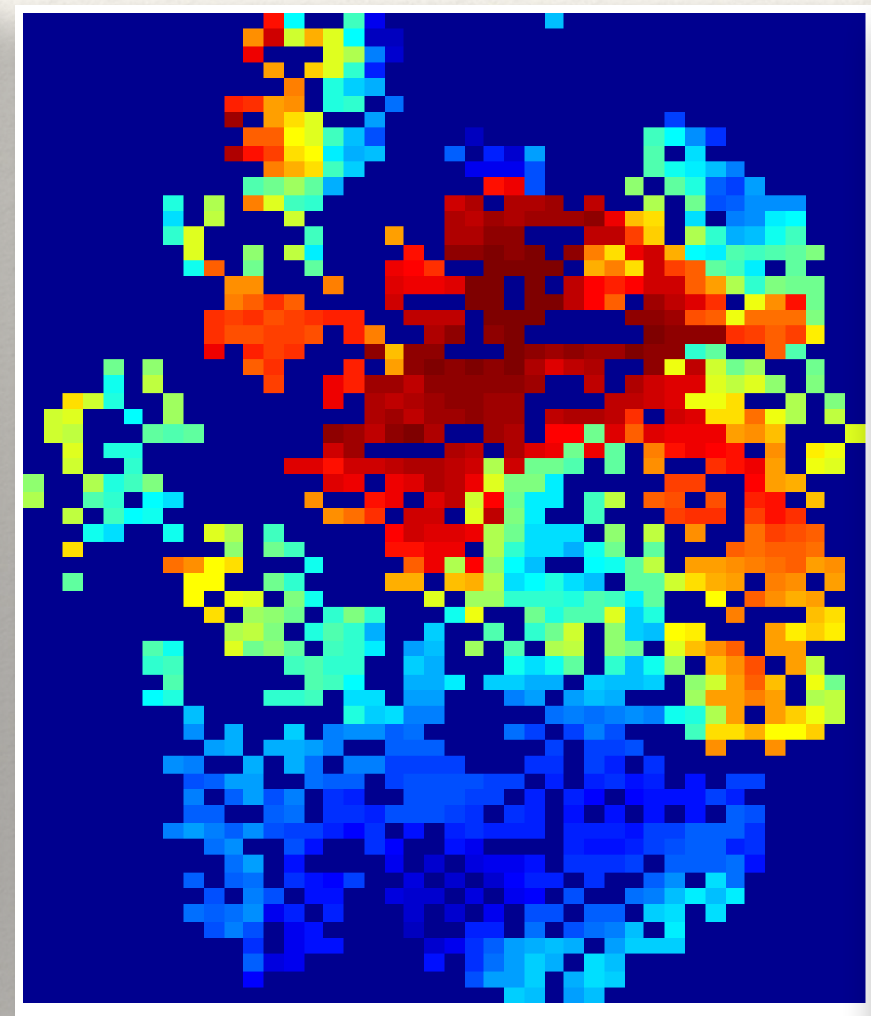


Learned probabilities, obtained in **2 seconds** with *k-NN*

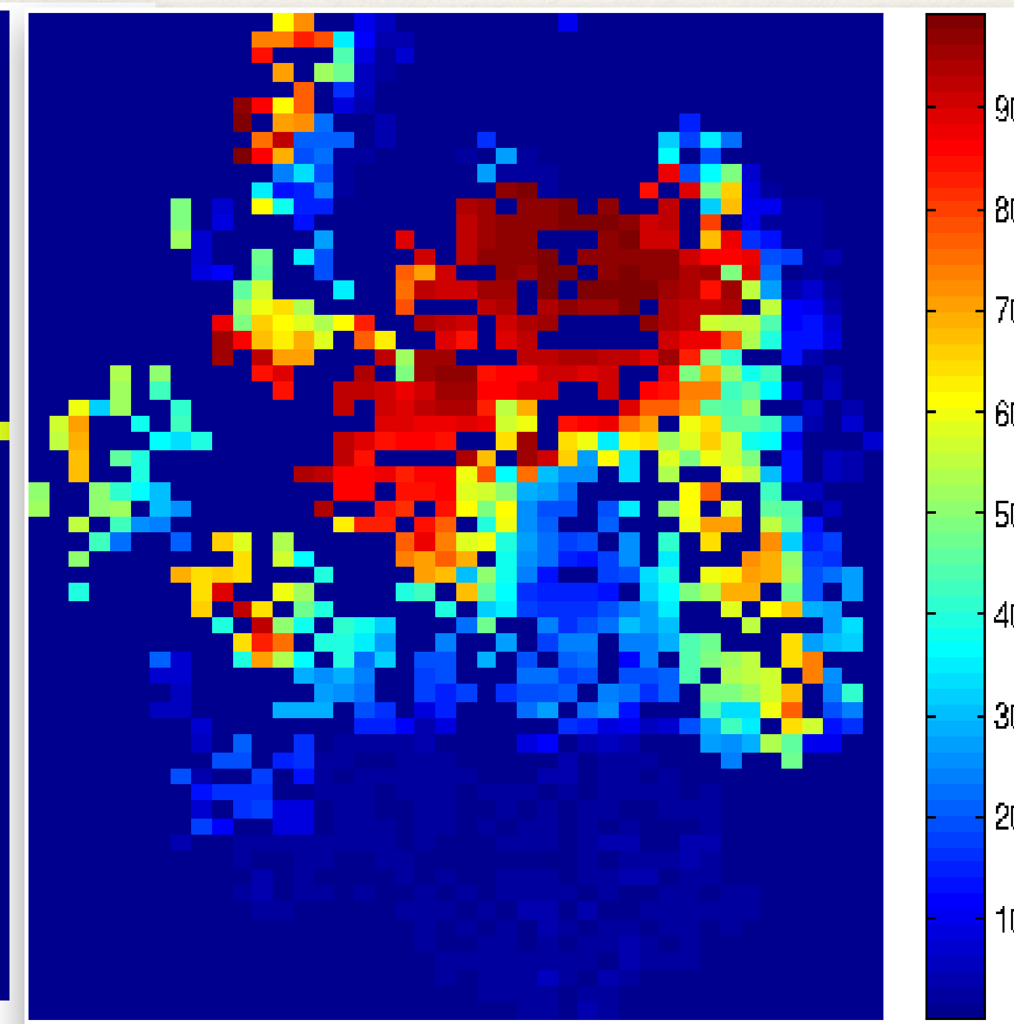


Ground truth, obtained in **200 seconds** from simulations

Predicting Success Probabilities of Grasping Action

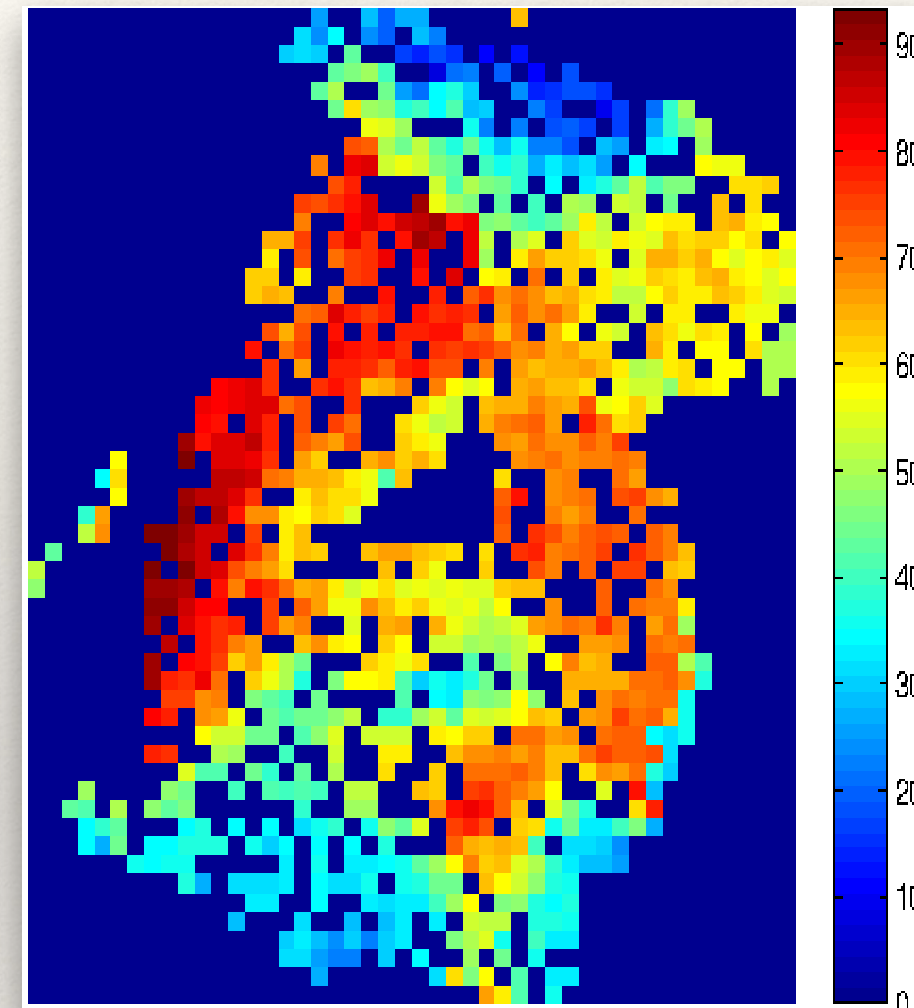


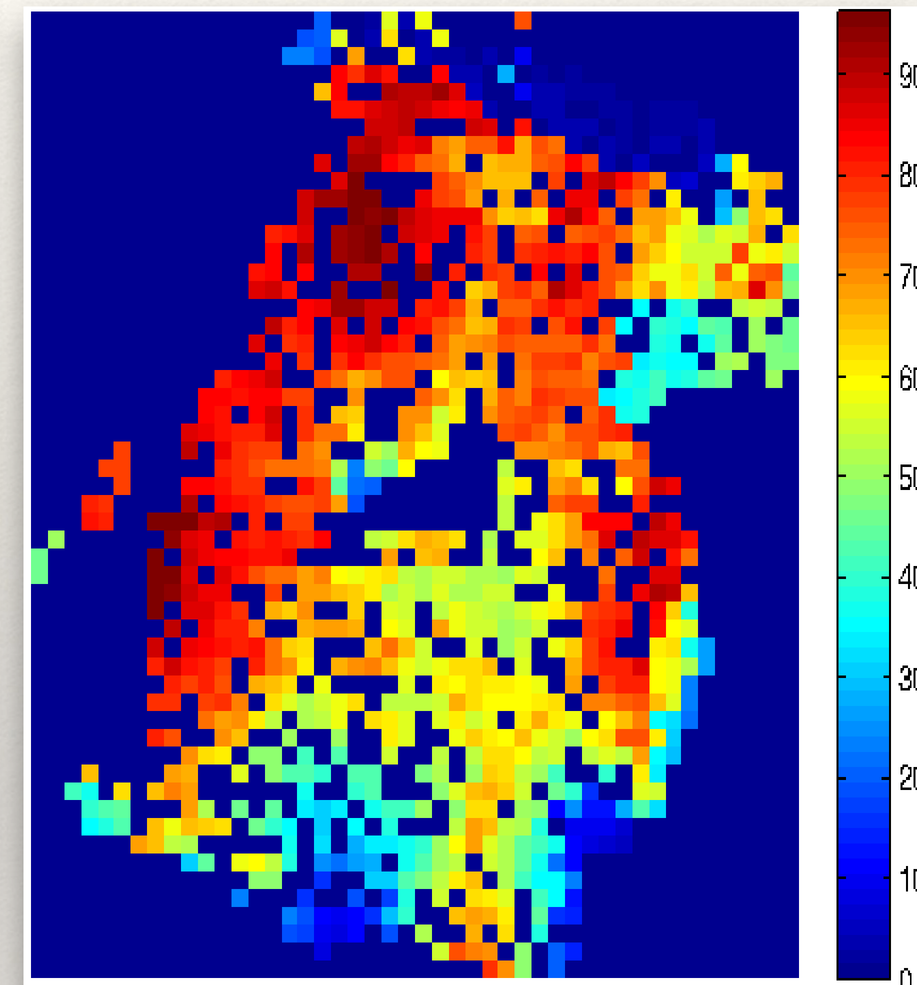
Learned probabilities, obtained in **2 seconds** with *k-NN*



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

Predicting Success Probabilities of Grasping Action

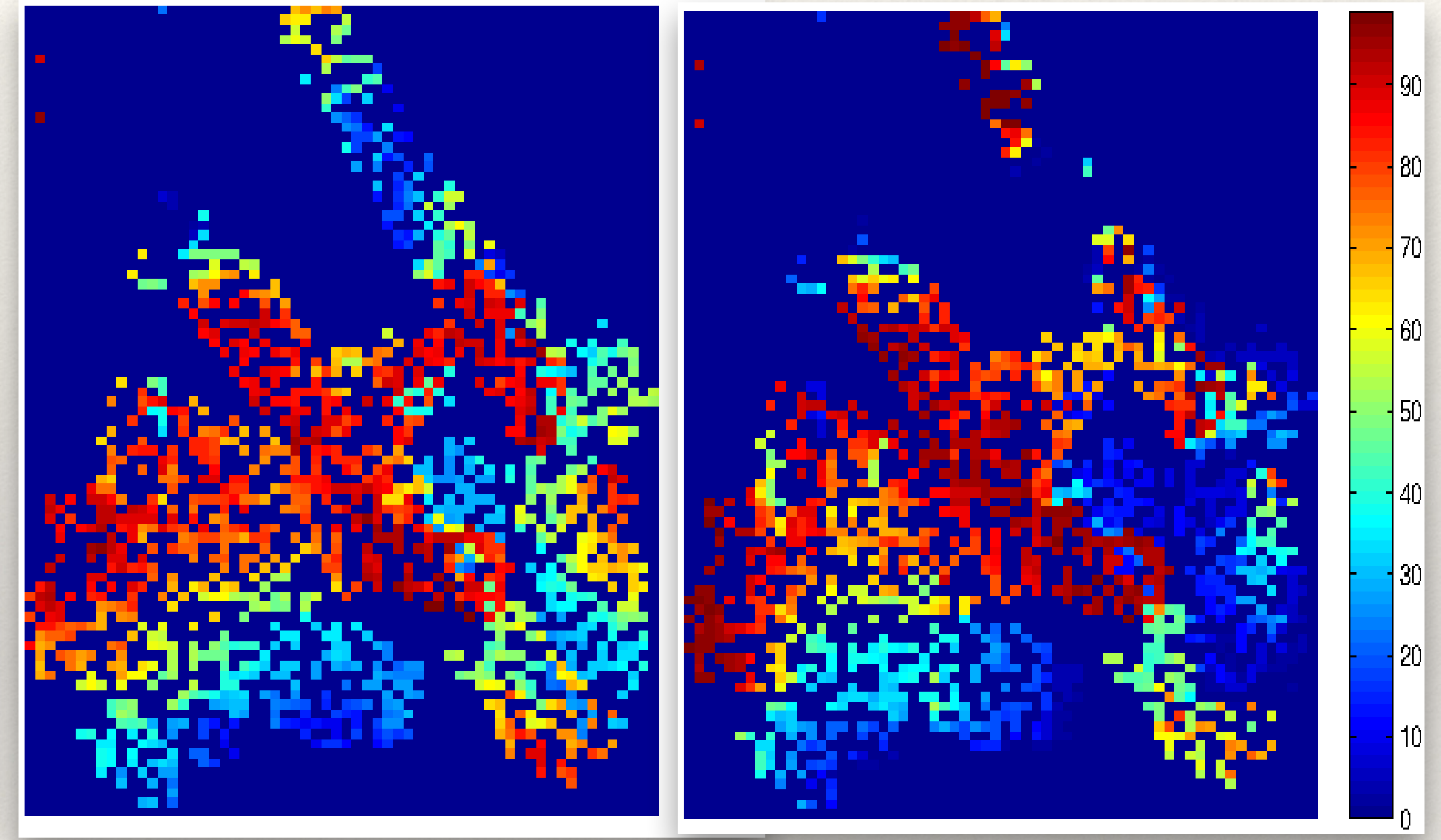




Learned probabilities, obtained in **2 seconds** with *k-NN*

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

Predicting Success Probabilities of Grasping Action



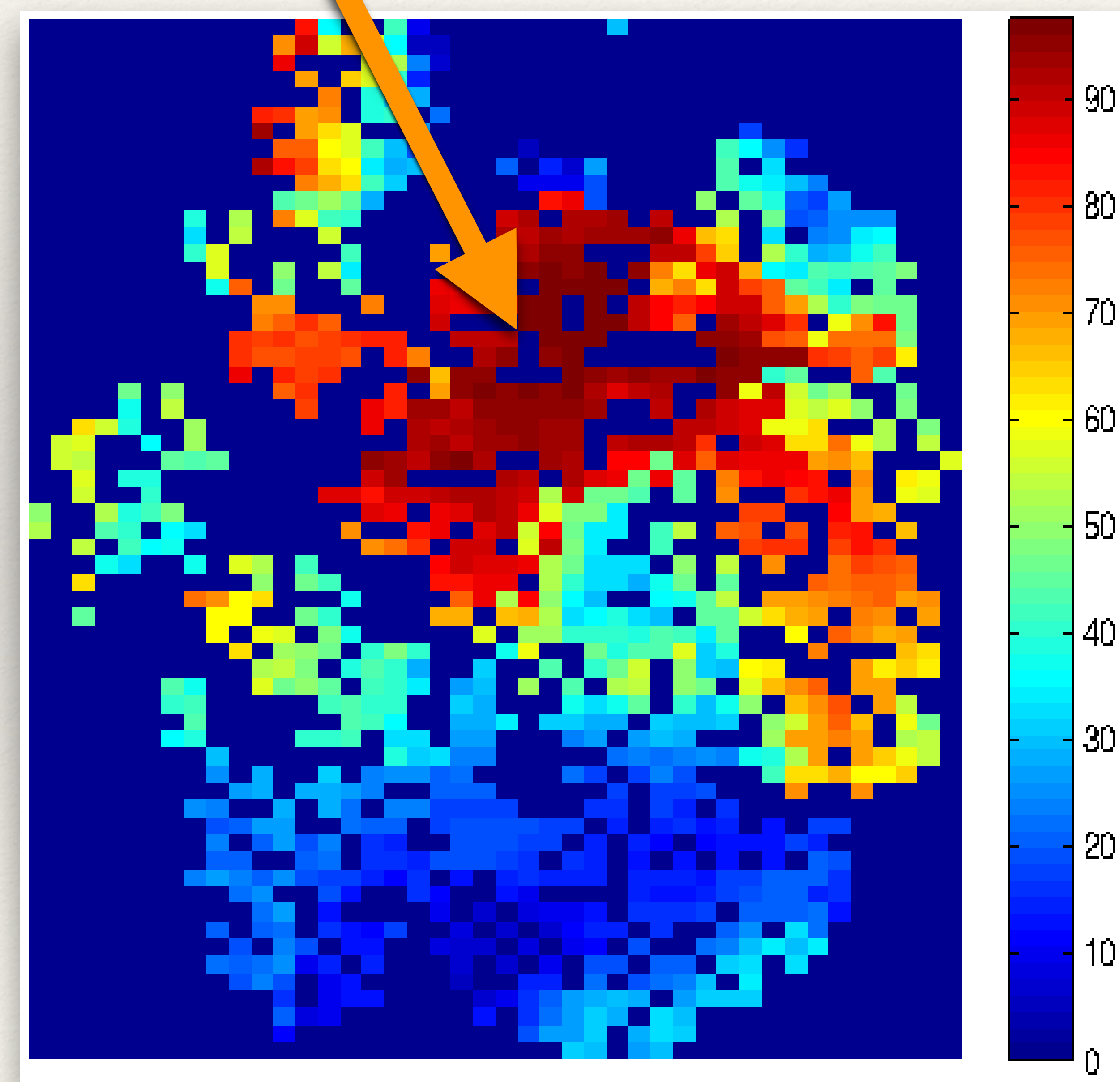
Learned probabilities, obtained in **2 seconds** with *k-NN*

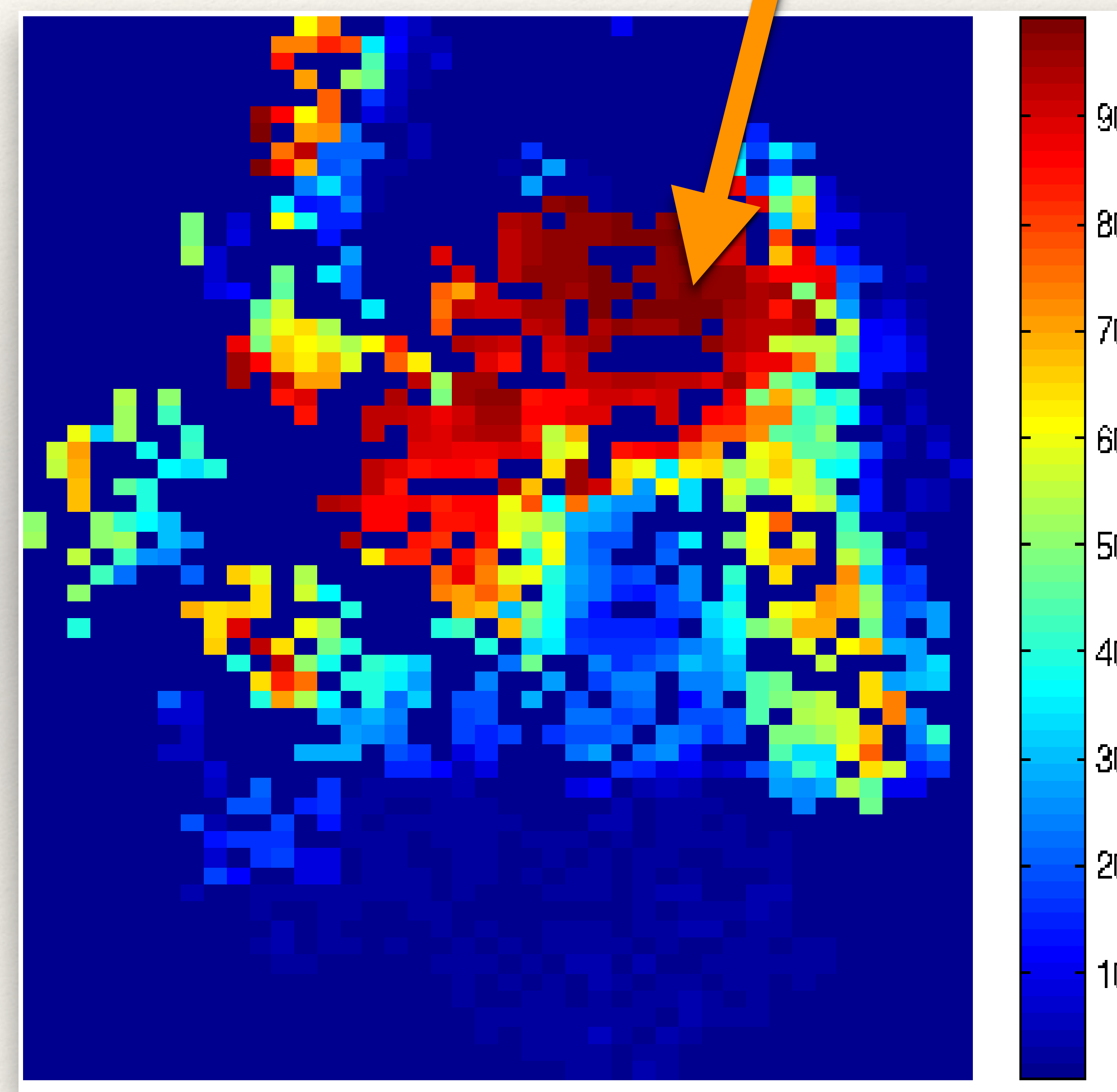
Ground truth, obtained in **200 seconds** from simulations

Best grasping point according to the learned probabilities

≠

Best grasping point according to the simulator

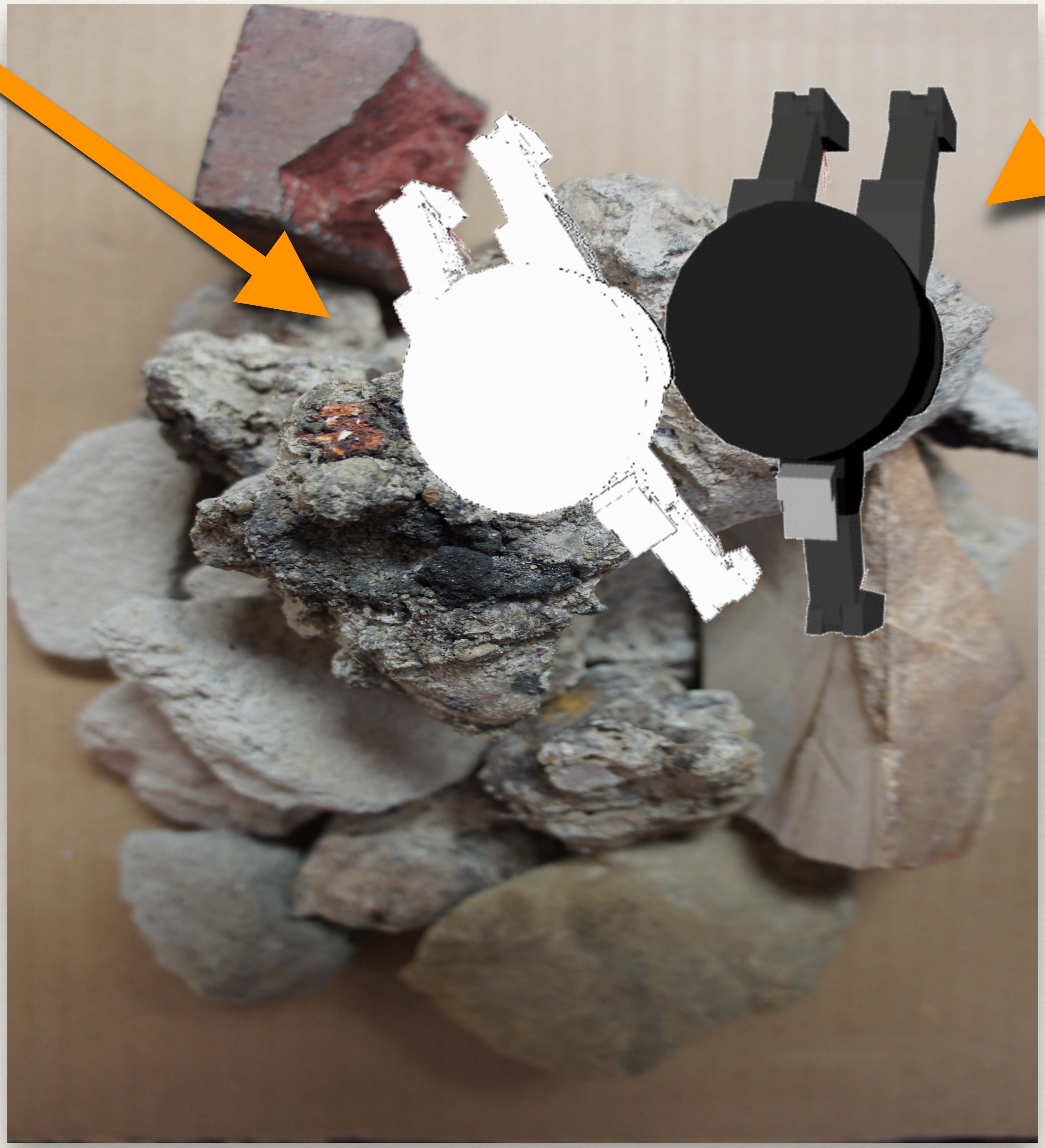




Best grasping point according to the learned probabilities

$\neq$

Best grasping point according to the simulator



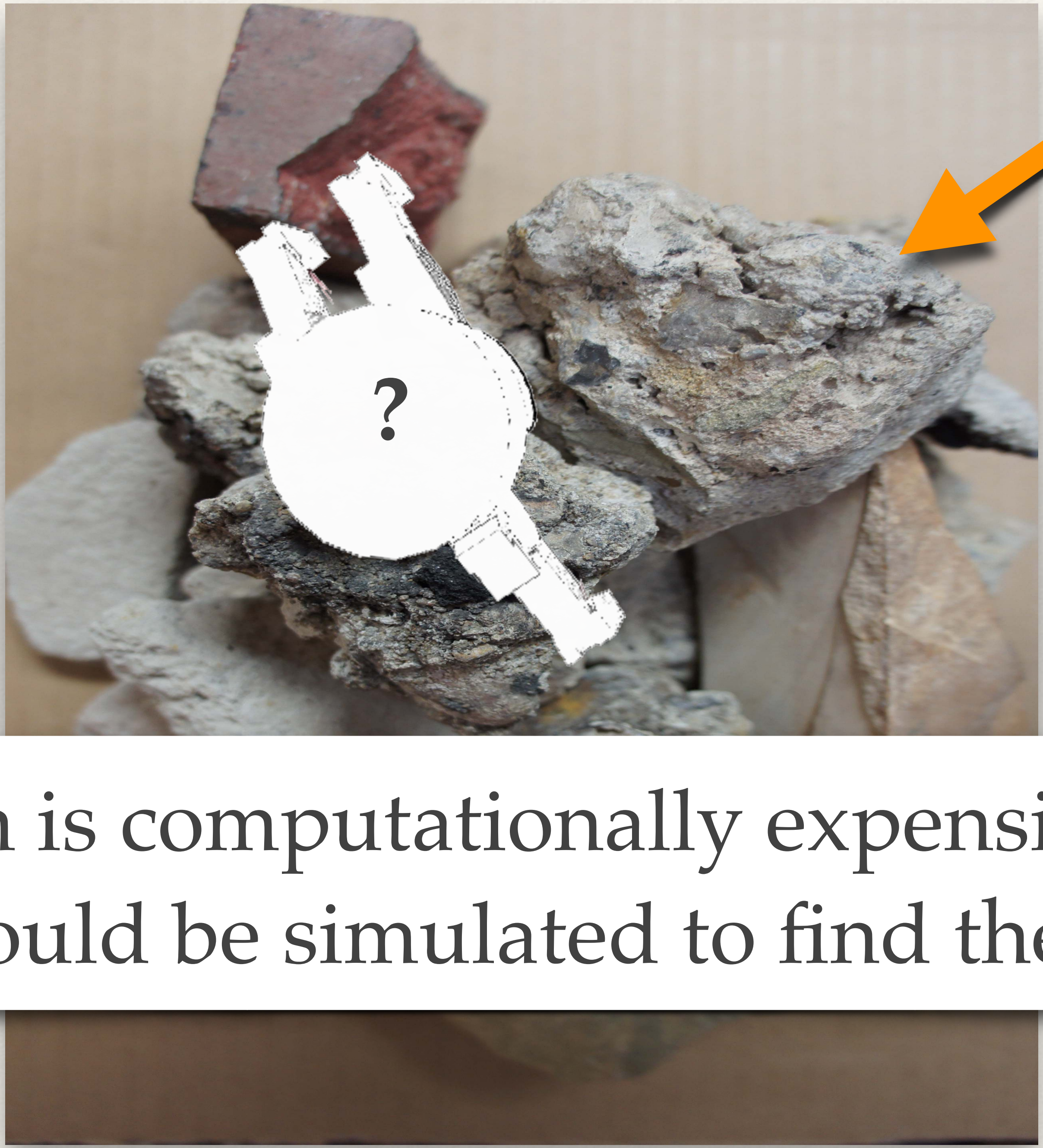
Fine-tuning in Simulation

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

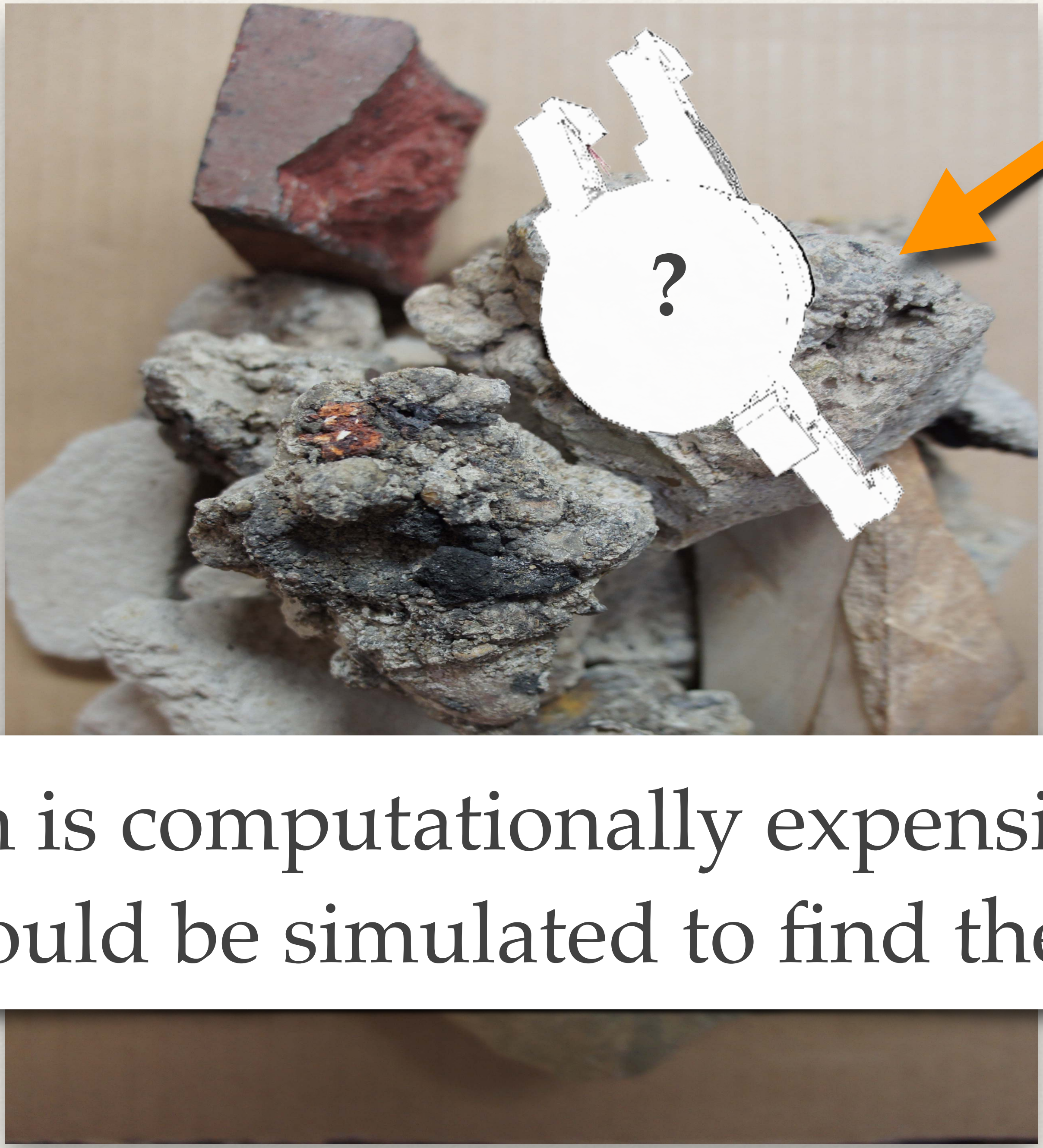
Best grasping point according to the simulator



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

Fine-tuning in Simulation

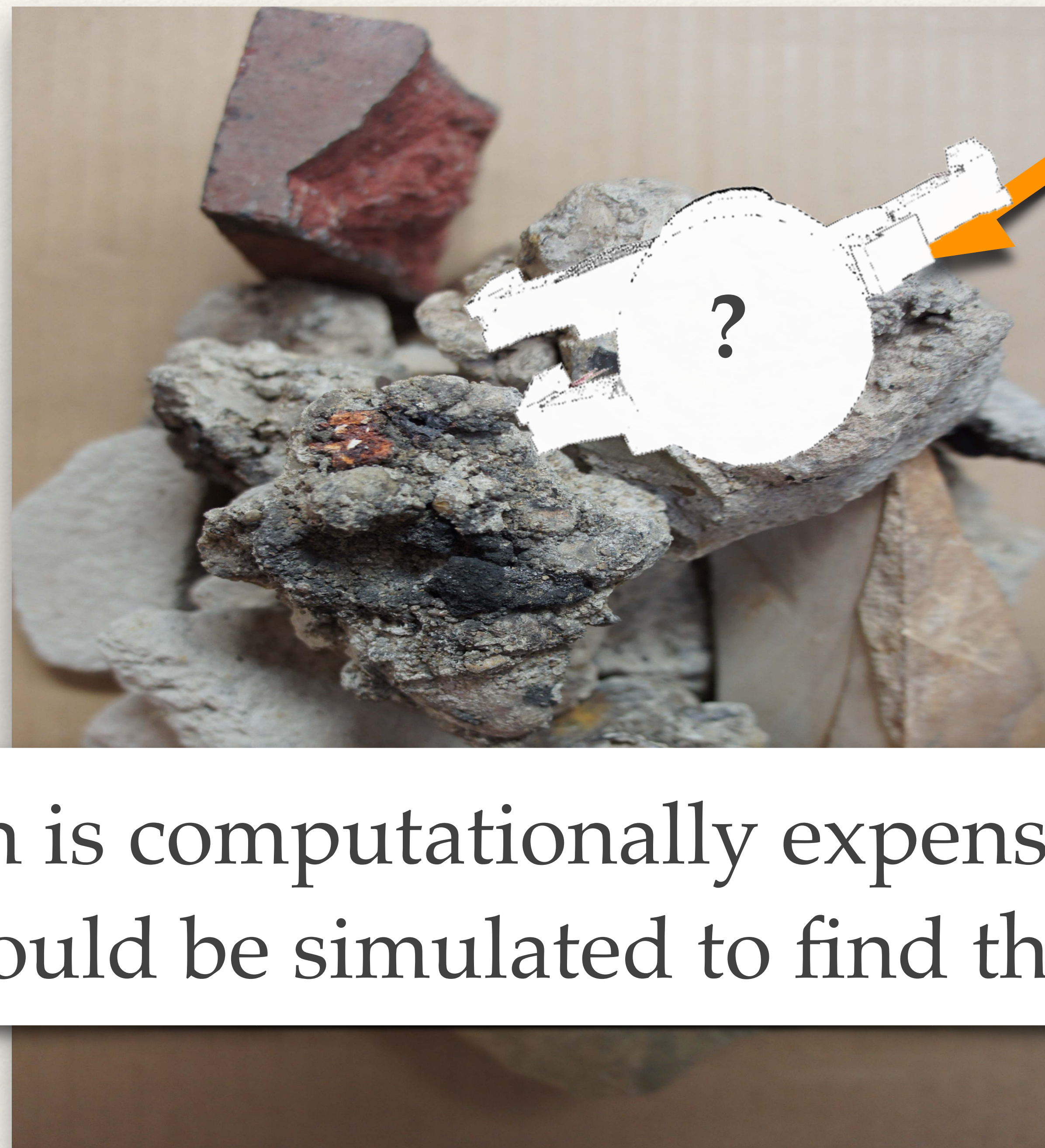
Best grasping point according to the simulator



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

Fine-tuning in Simulation

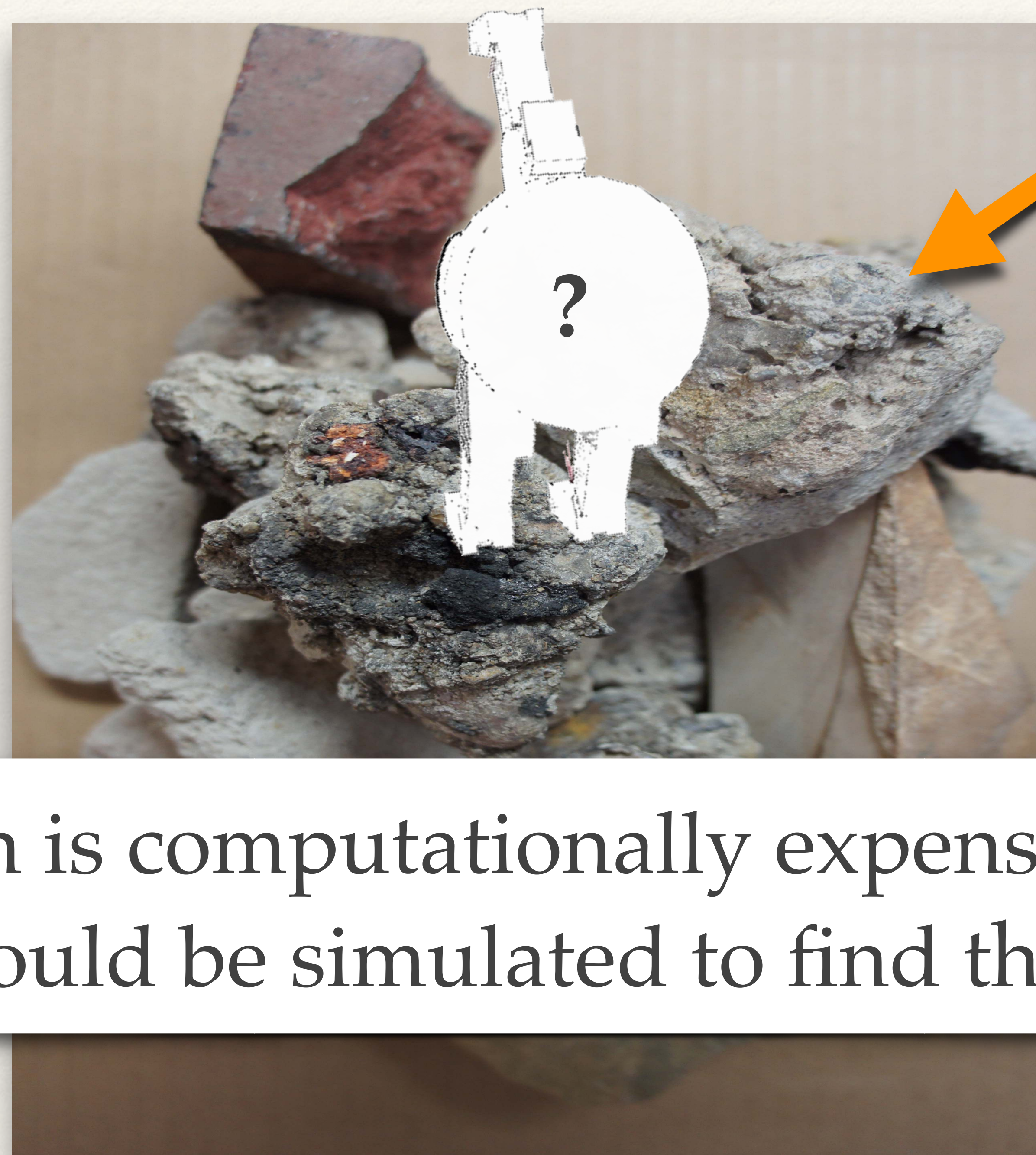
Best grasping point according to the simulator



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?

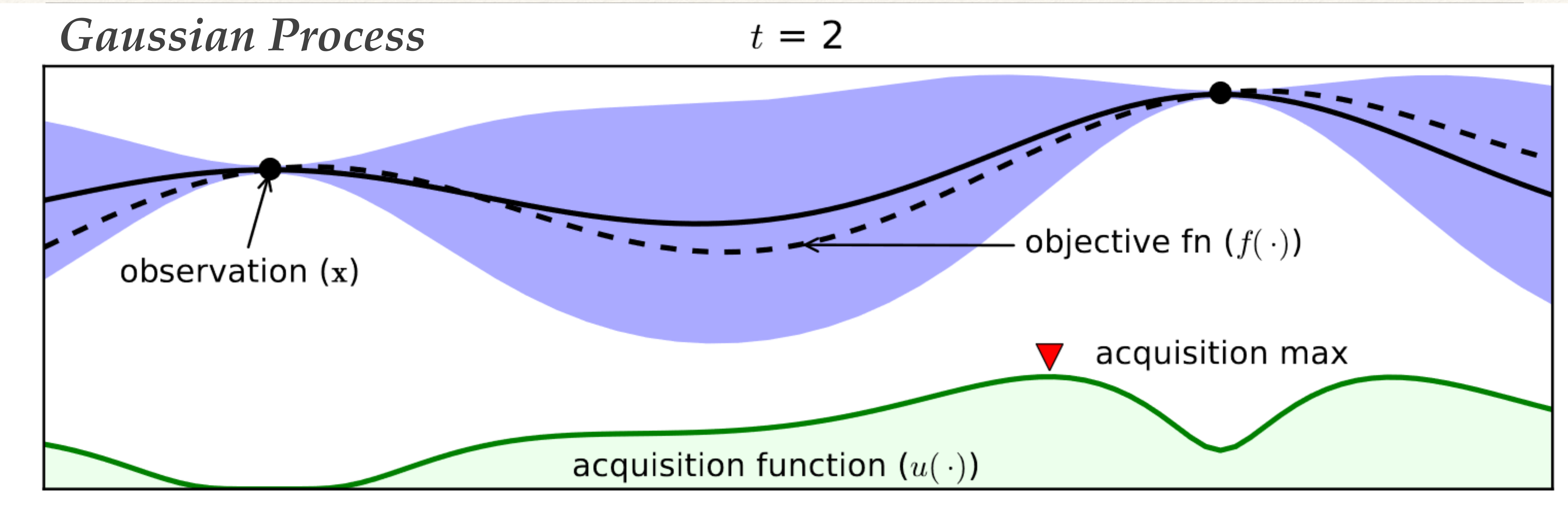
Black-box Optimization

Input: position and rotation of the robotic hand

Unknown function *f*

Output: stability of the simulated grasping action

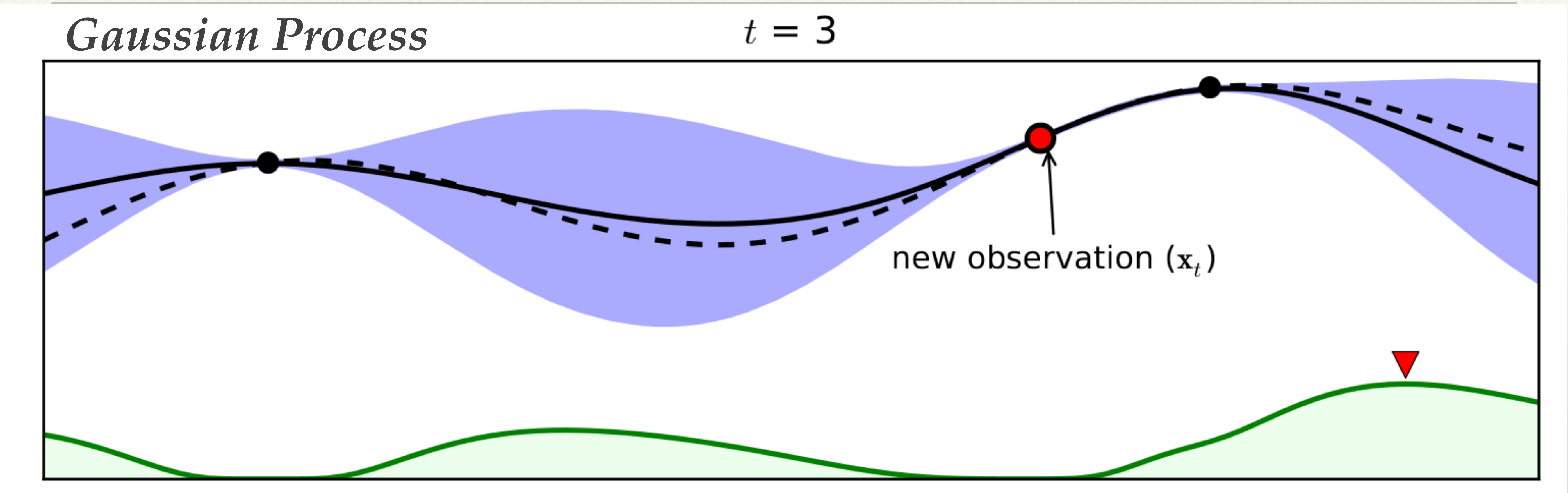
Bayesian Black-box Optimization



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.

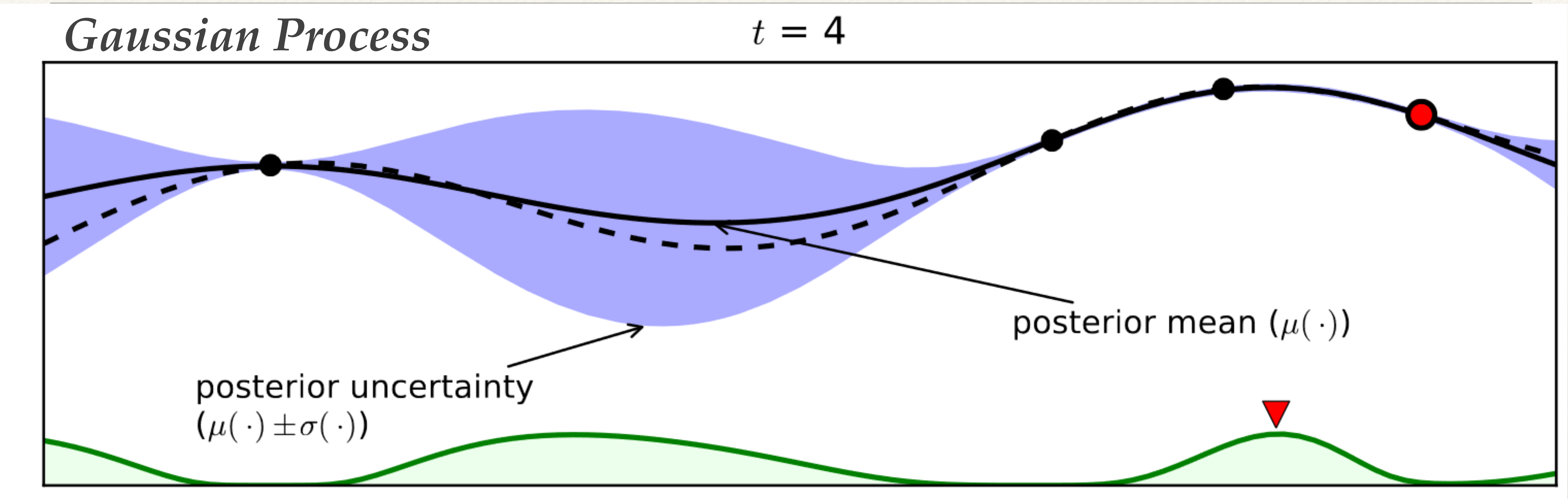
Bayesian Black-box Optimization



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.

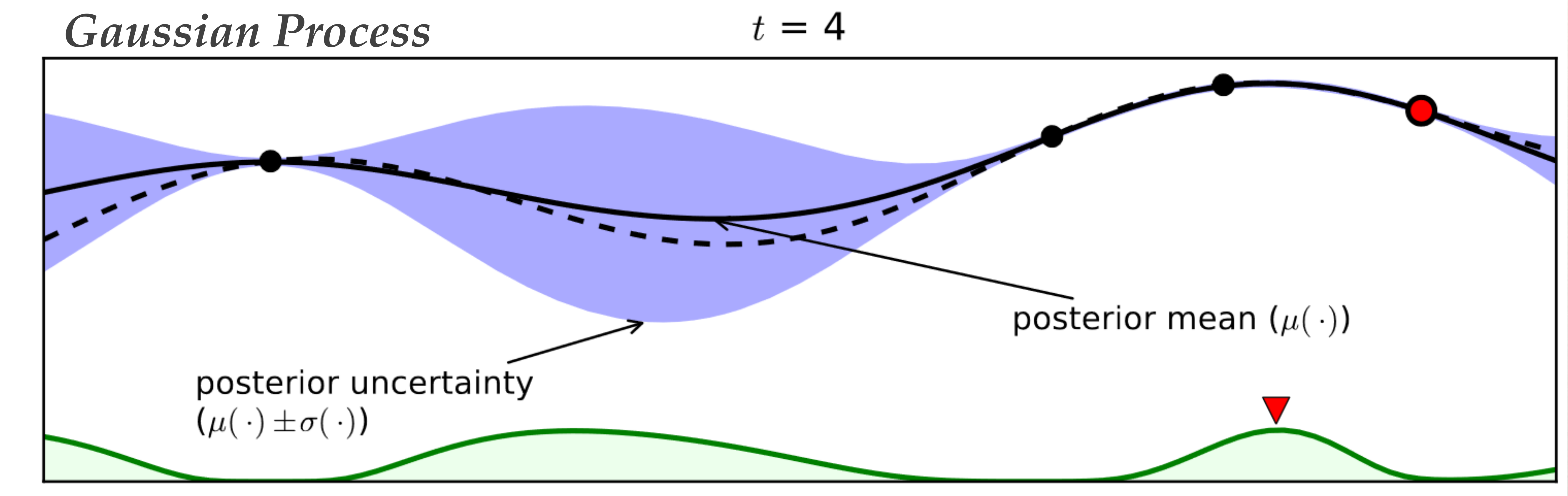
Bayesian Black-box Optimization



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.

Bayesian Black-box Optimization



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:

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

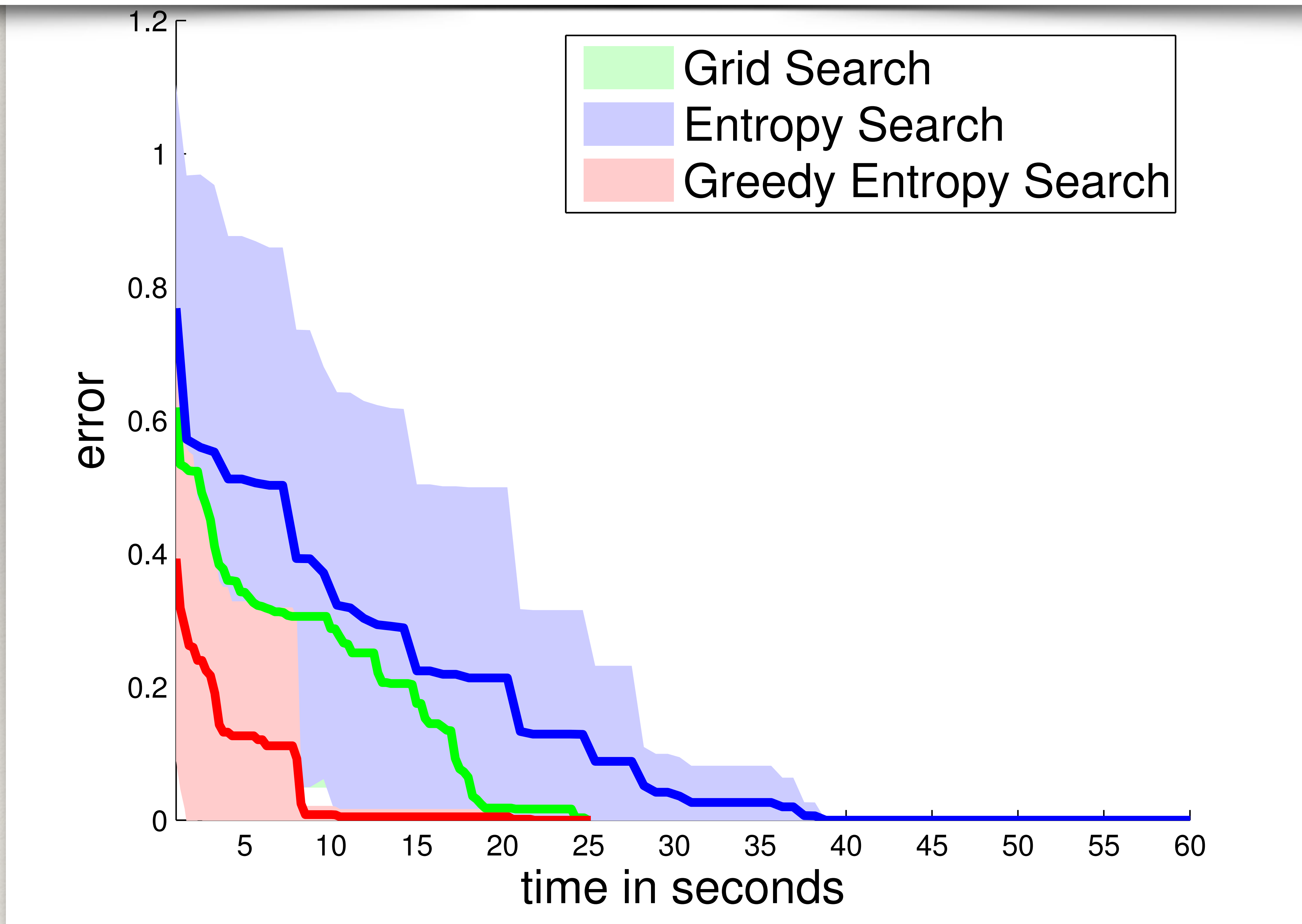
Heaviside step function

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

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

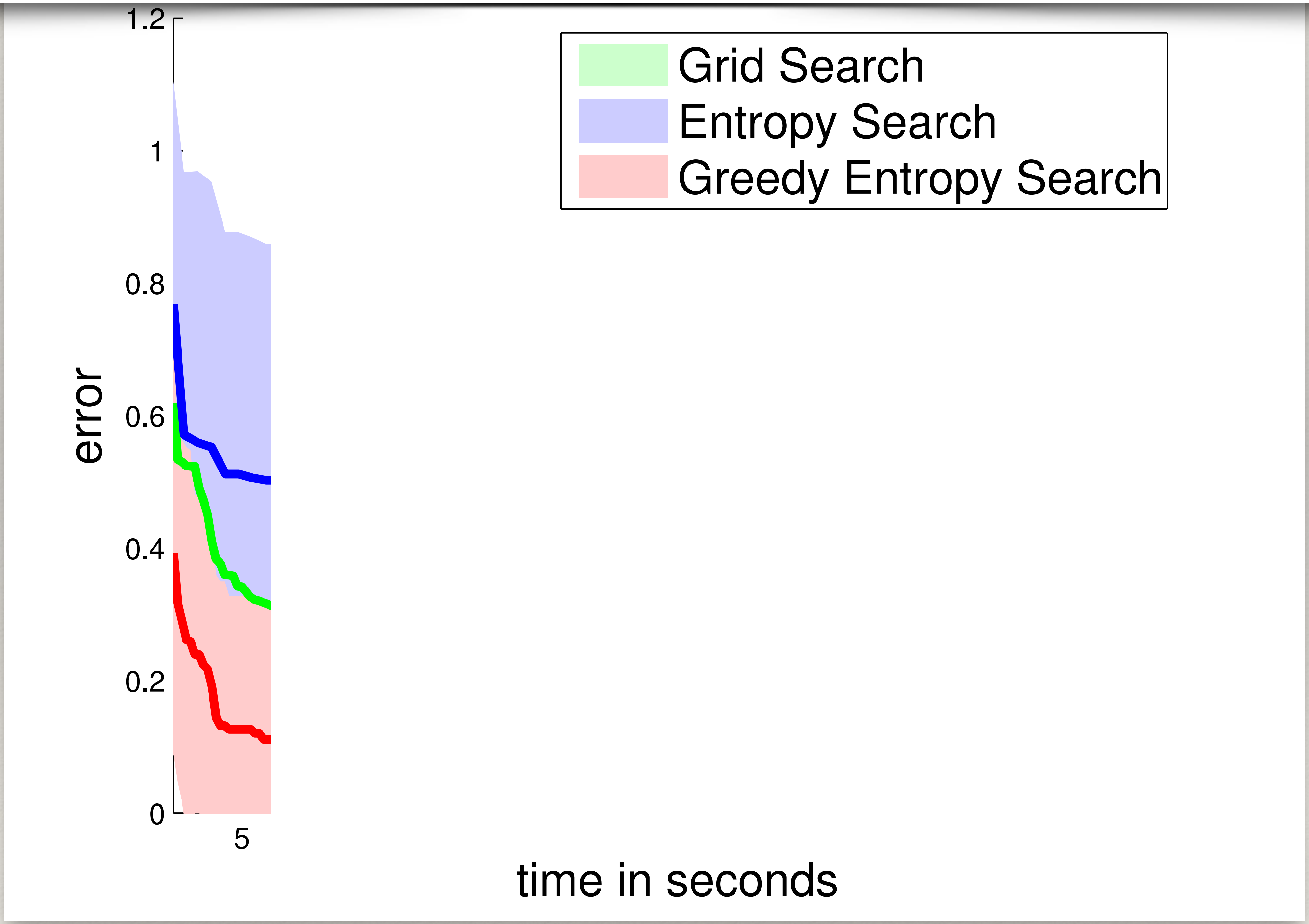
Anytime Optimization

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



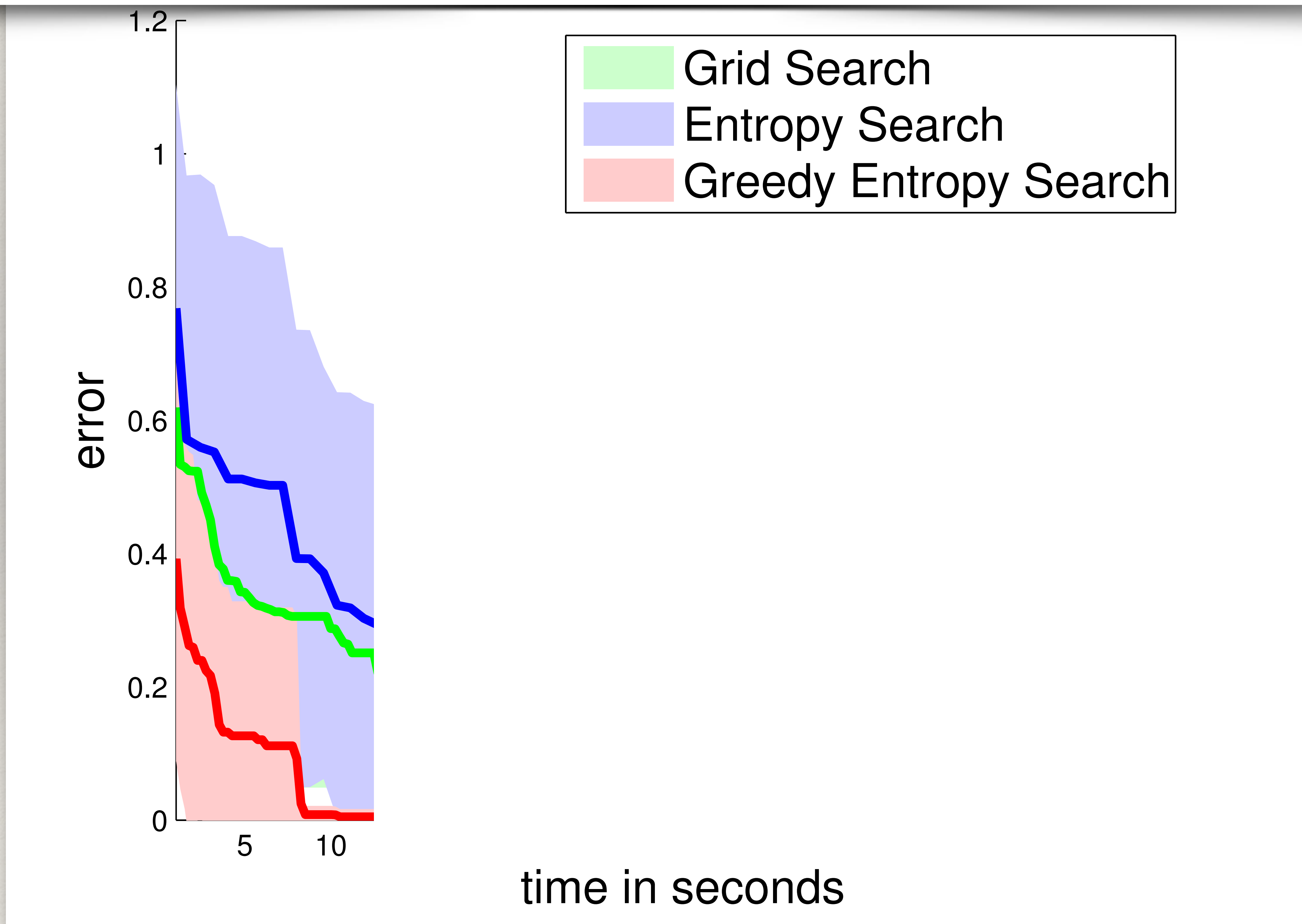
Anytime Optimization

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



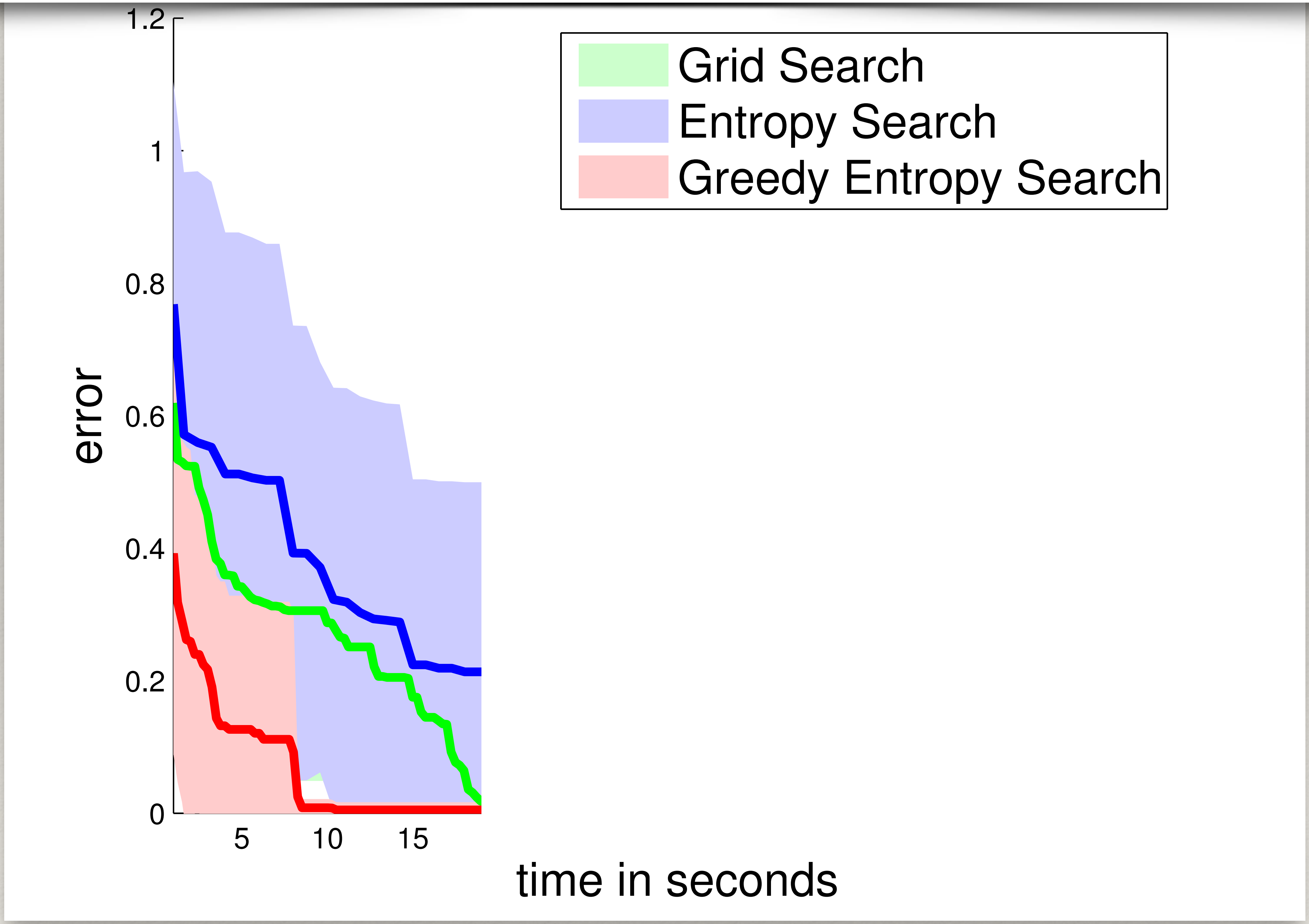
Anytime Optimization

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



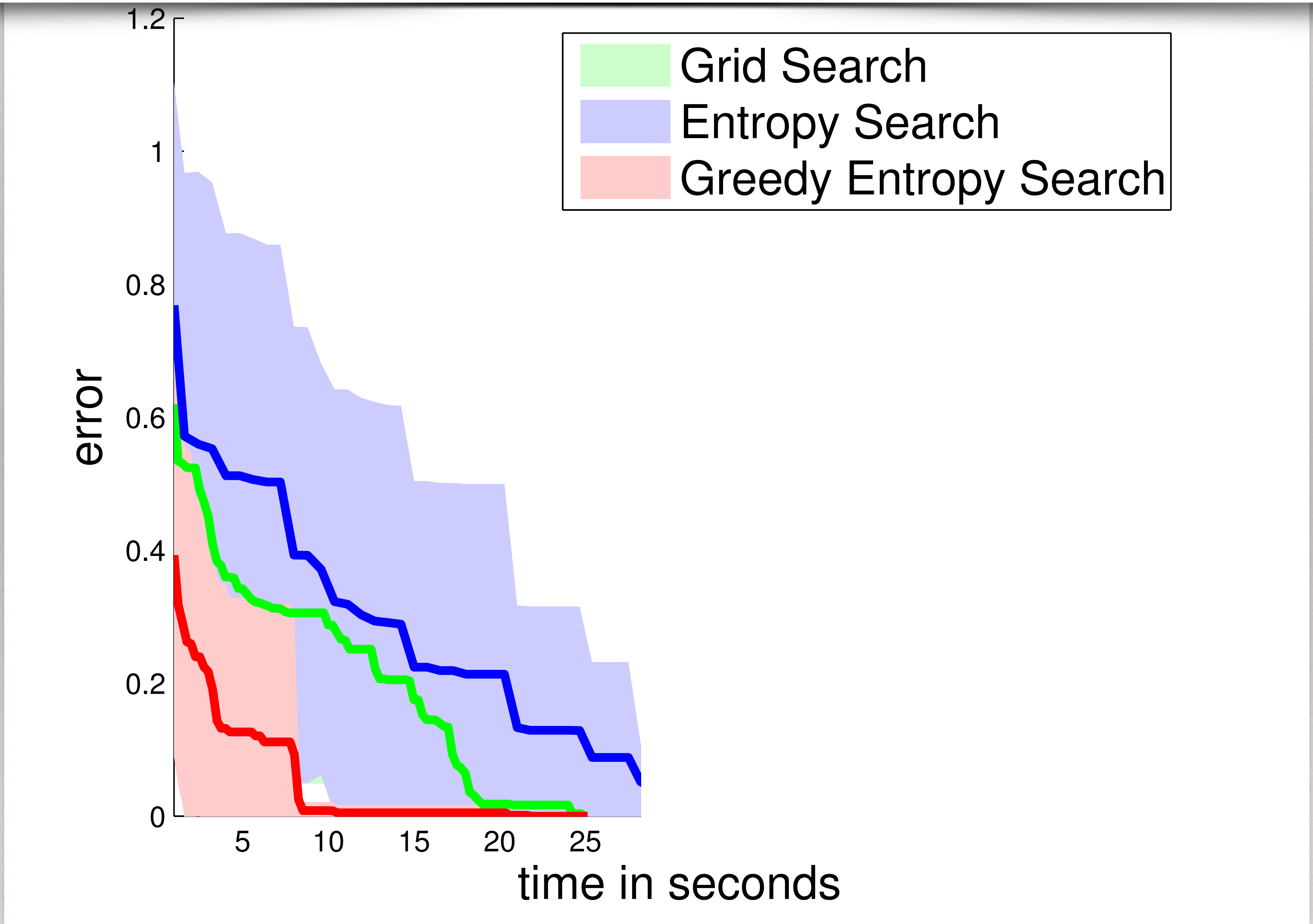
Anytime Optimization

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



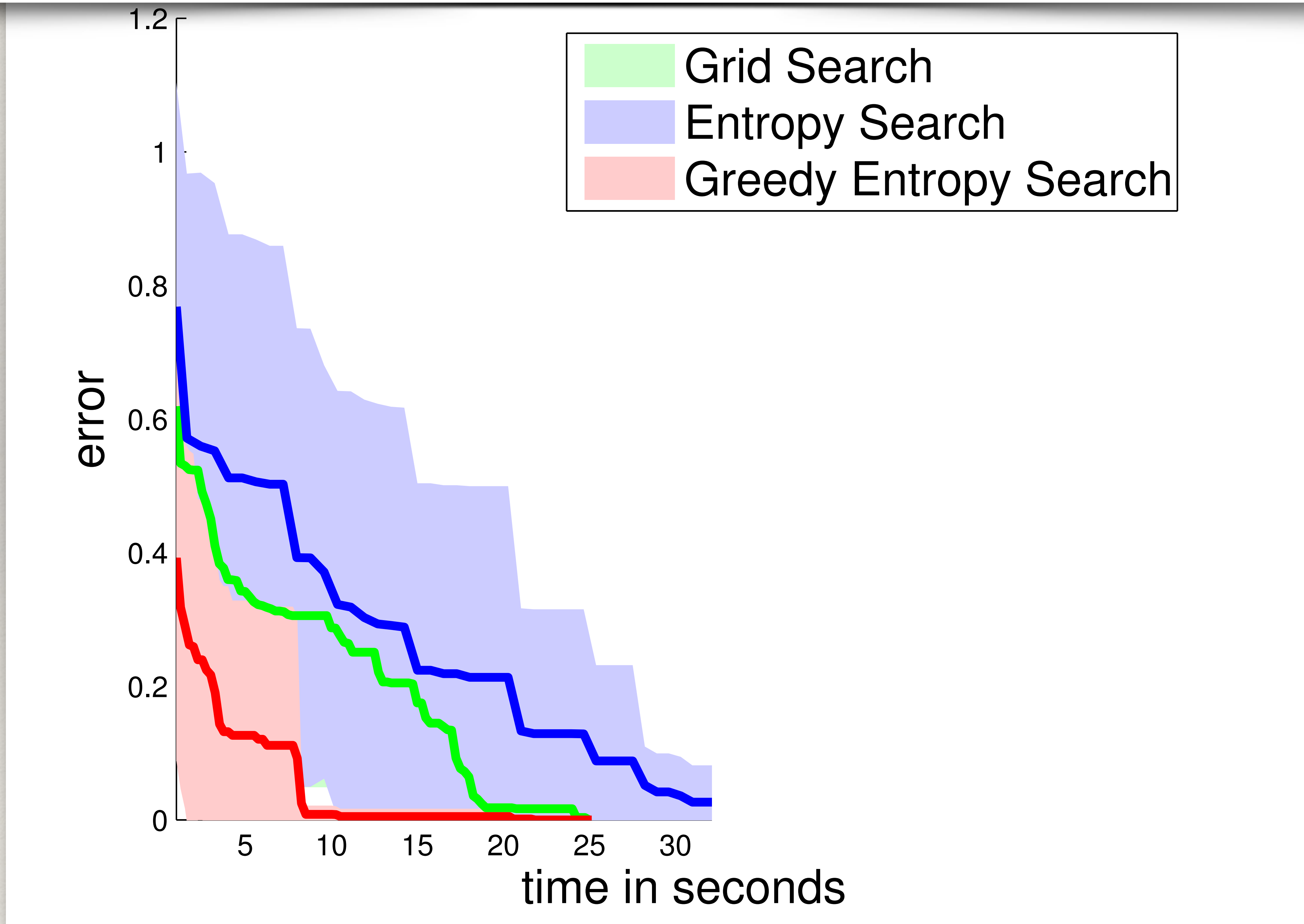
Anytime Optimization

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



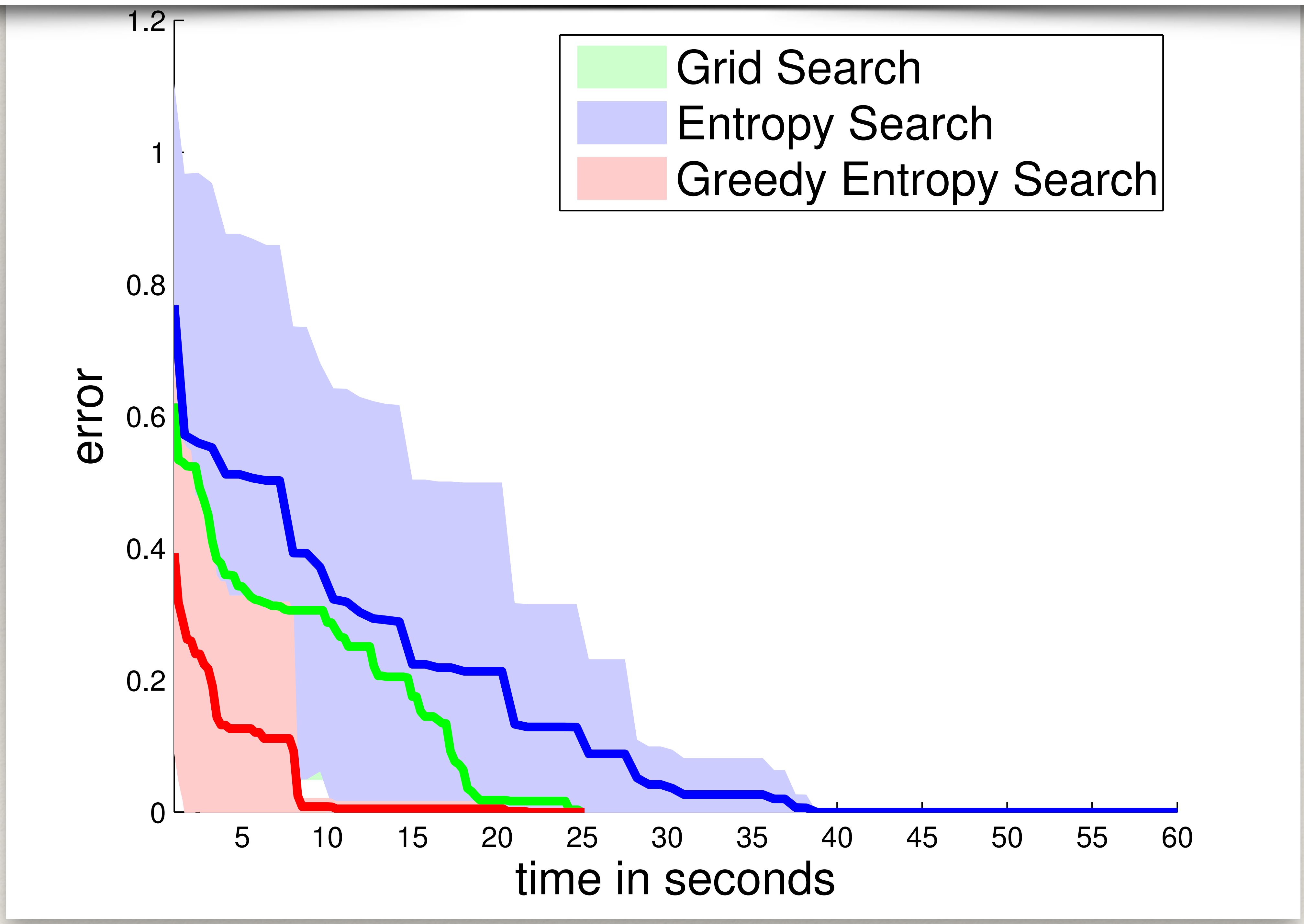
Anytime Optimization

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**

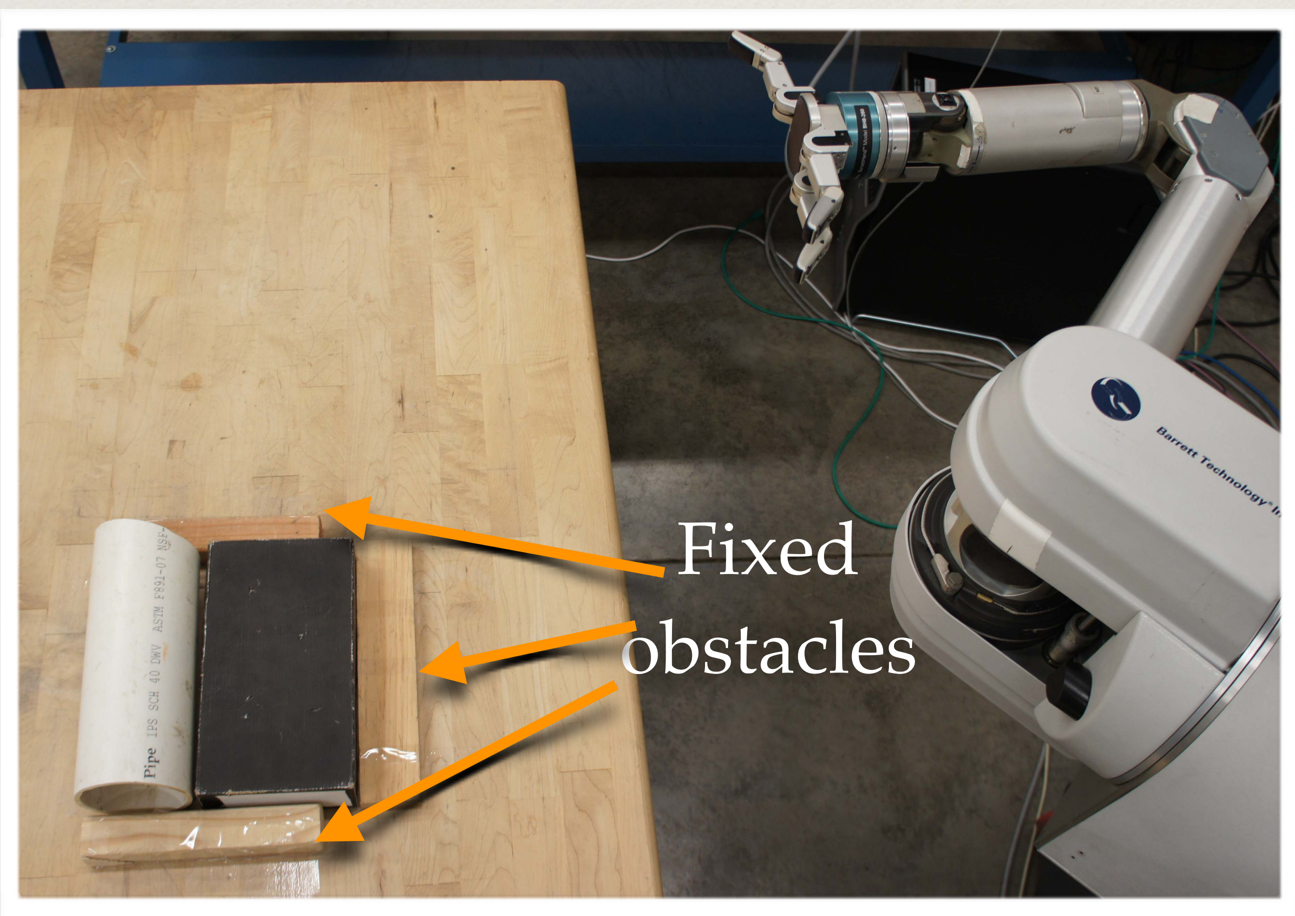
A. Boularias *et al.* (2014) in *Conference of the Association for the Advancement of Artificial Intelligence (AAAI)*

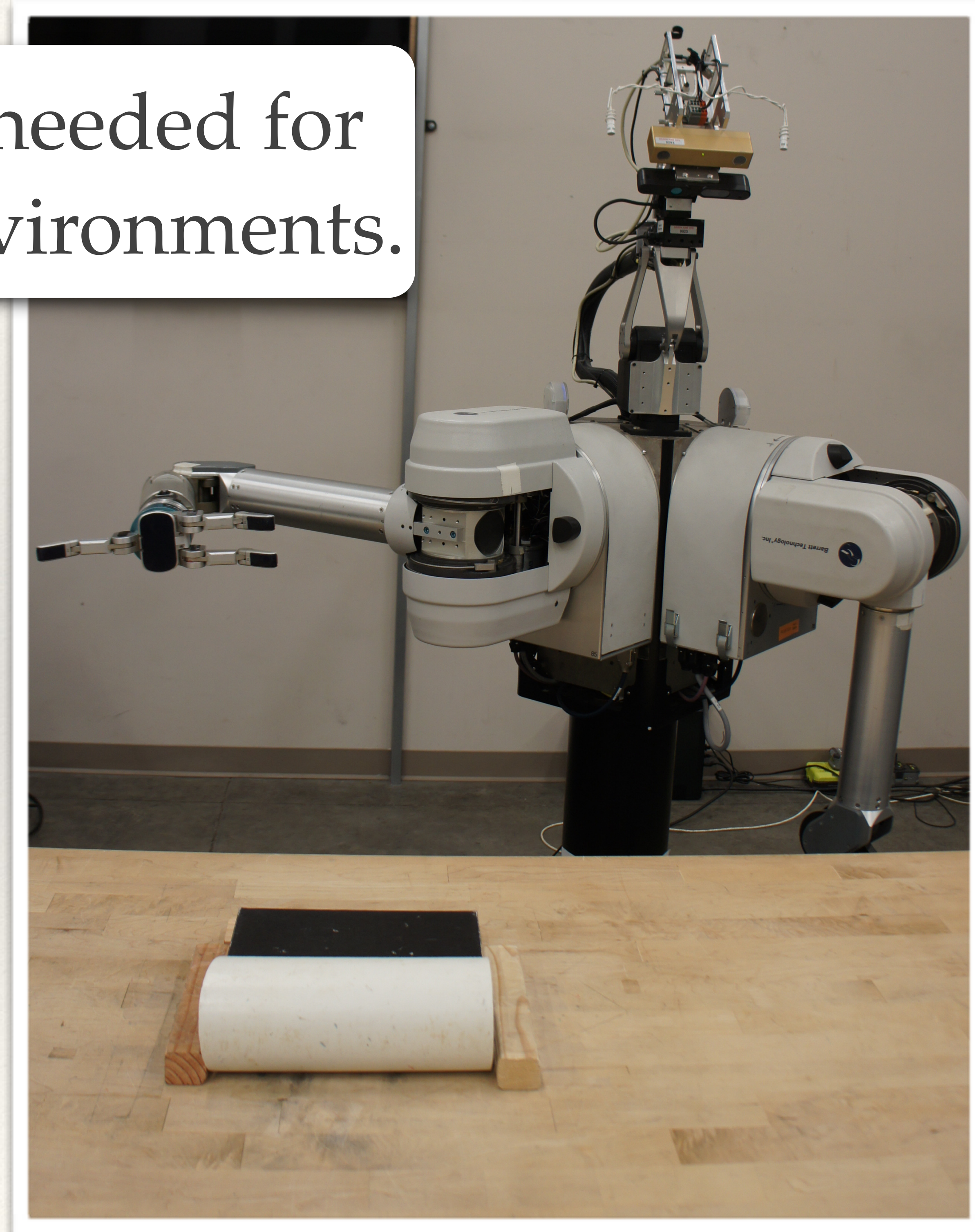
Outline

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

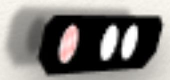
Pushing objects

Pushing and moving objects is needed for grasping objects in confined environments.





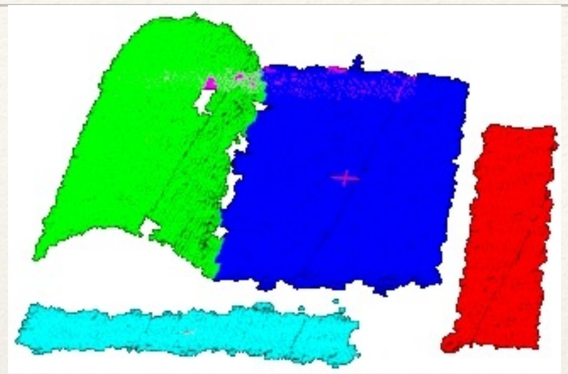
Overview of the integrated system



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

Overview of the integrated system

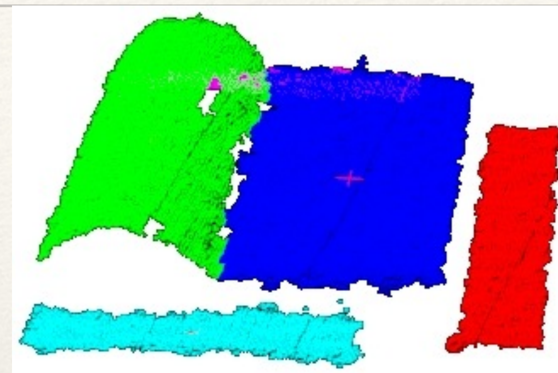
Get an image of the scene from an RGB-D sensor



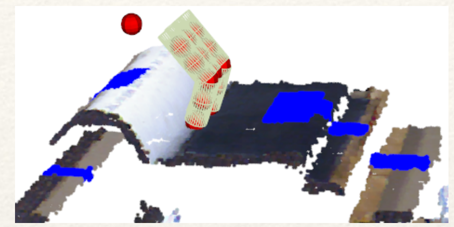
Segment the scene image into objects

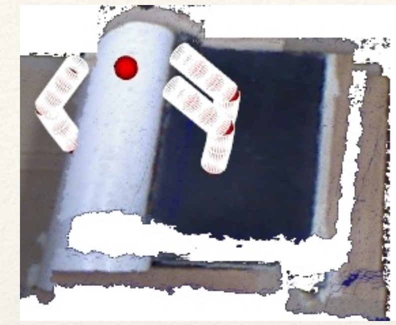
Overview of the integrated system

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



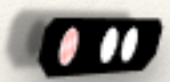
Segment the scene image into objects



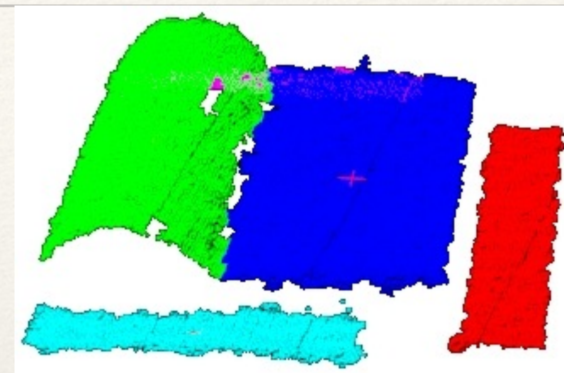


Sample a number of grasping and pushing actions for each object

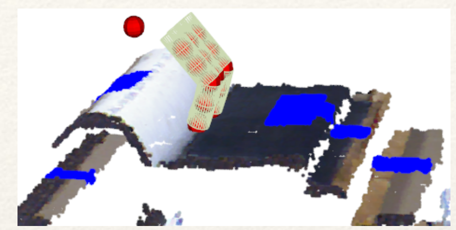
Overview of the integrated system

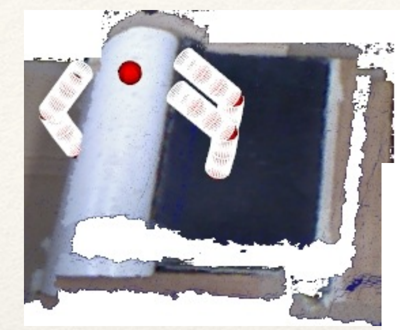


Get an image of the scene from an RGB-D sensor

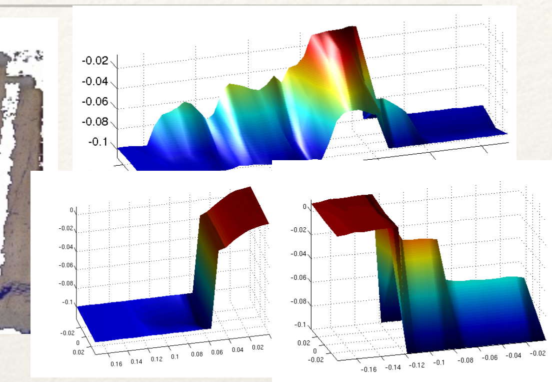


Segment the scene image into objects





Sample a number of grasping and pushing actions for each object

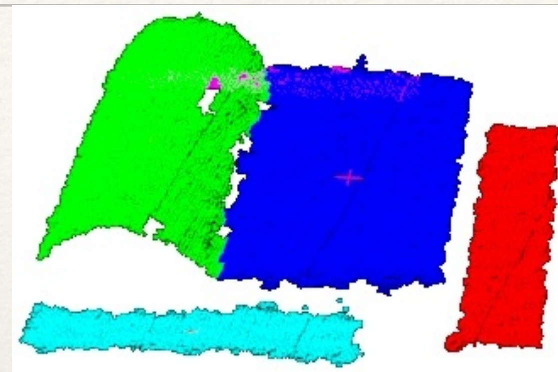


Extract the features of each sampled action

Overview of the integrated system

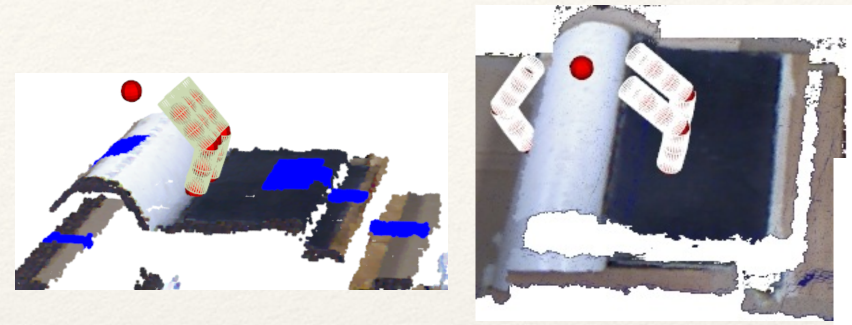
Get an image of the scene from an RGB-D sensor

→



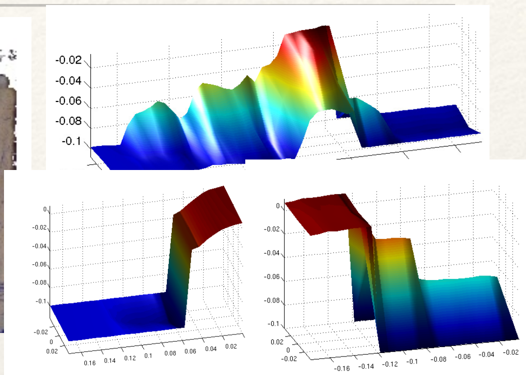
Segment the scene image into objects

→



Sample a number of grasping and pushing actions for each object

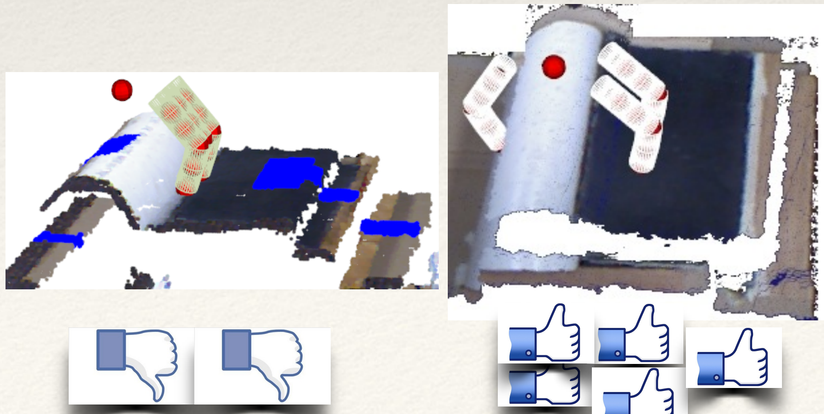
→



Extract the features of each sampled action

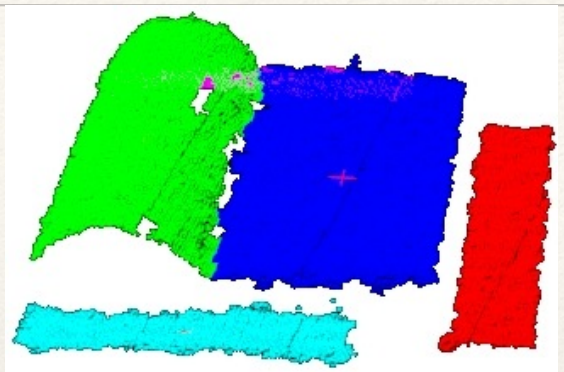
↓

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

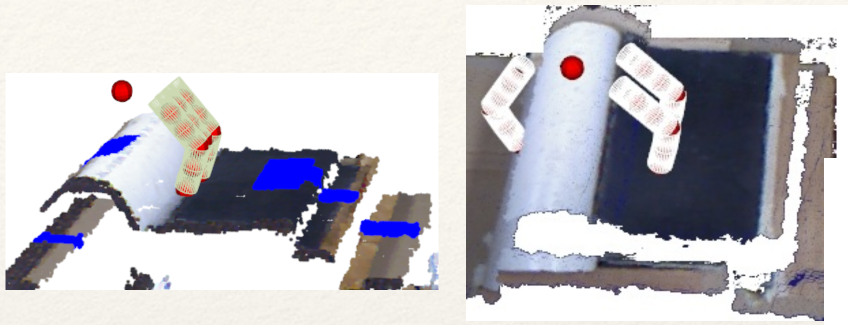


Overview of the integrated system

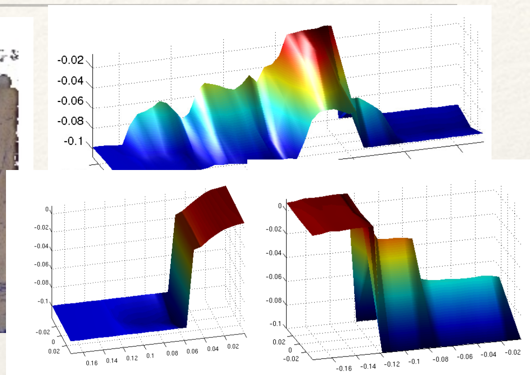
Get an image of the scene from an RGB-D sensor



Segment the scene image into objects



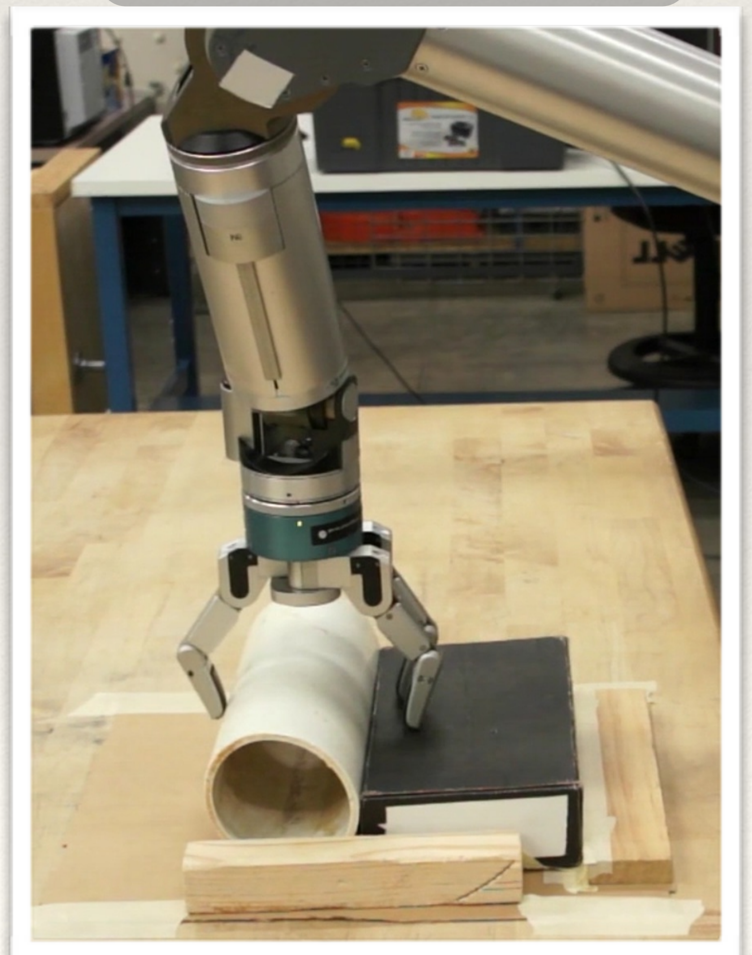
Sample a number of grasping and pushing actions for each object

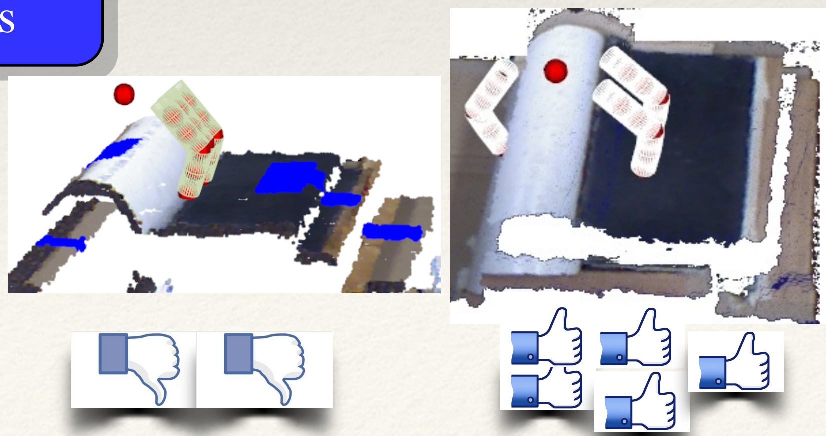


Extract the features of each sampled action

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

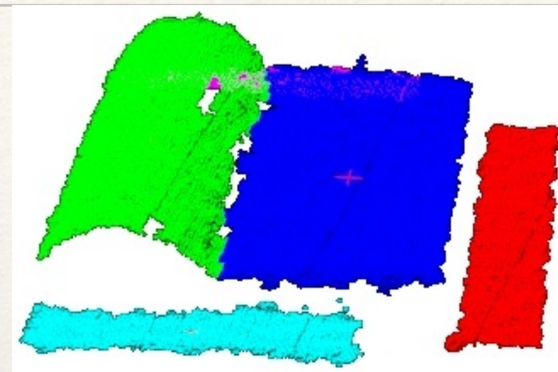
Execute the action with the highest Upper Confidence Bound (UCB), and obtain a binary reward based on the joint angles of the fingers



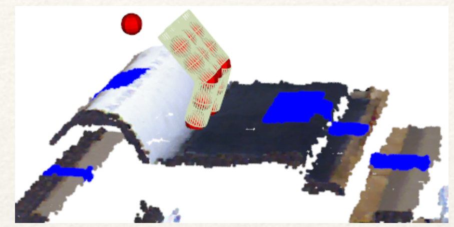


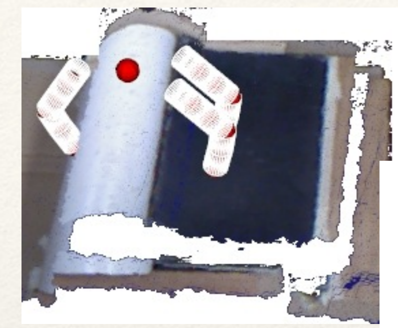
Overview of the integrated system

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

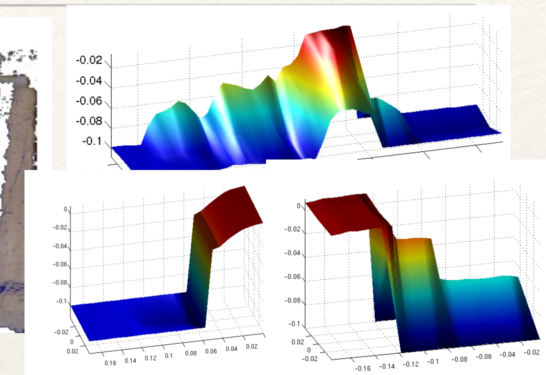


Segment the scene image into objects





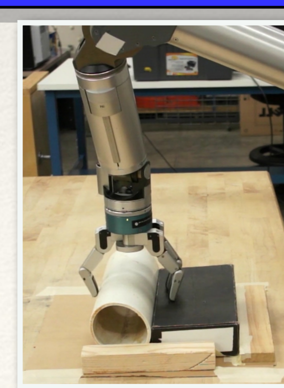
Sample a number of grasping and pushing actions for each object

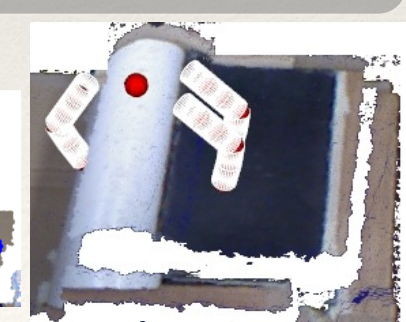


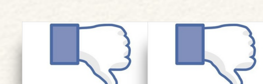
Extract the features of each sampled action

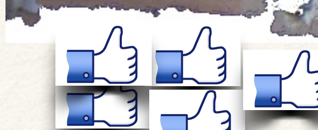
Predict the value of each sampled action using the values of the actions executed in previous states

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







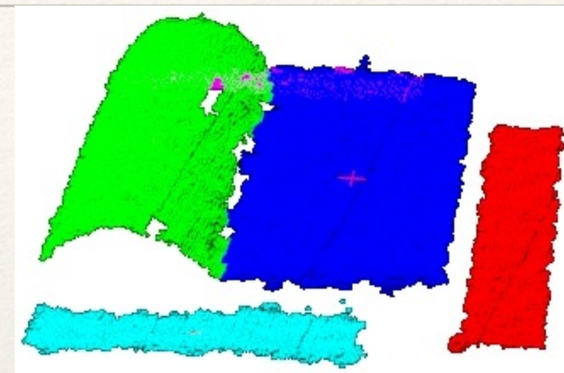


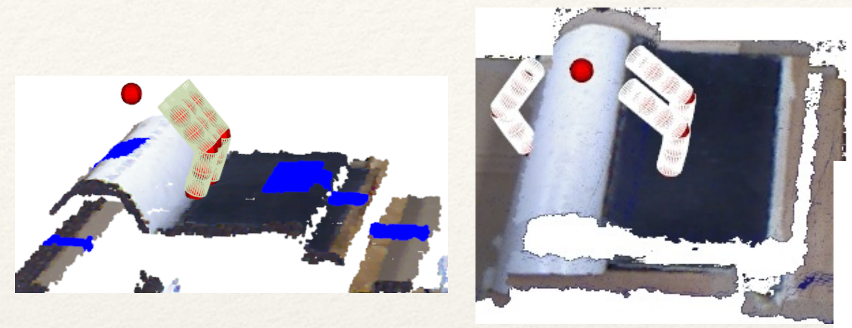
Re-evaluate the actions sampled in every state

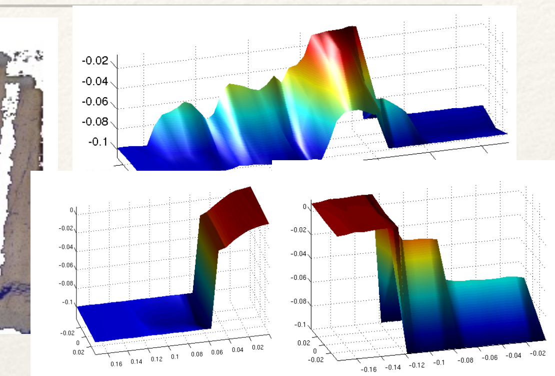
Re-compute the value of every previous state (scene) based on the value of the best action in the next state

Policy Iteration

Overview of the integrated system







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

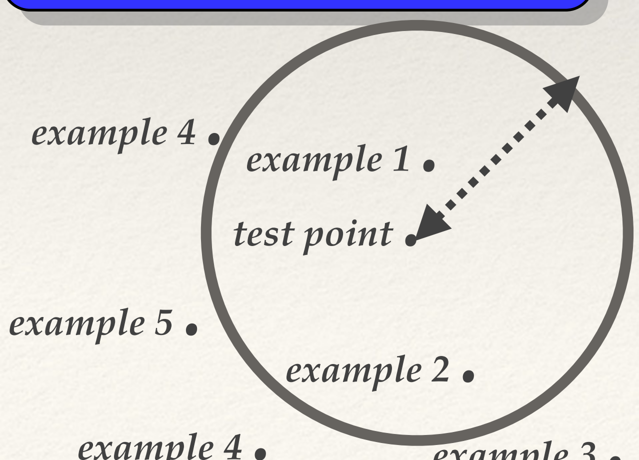
Predict the value of each sampled action using the values of the actions executed in previous states

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

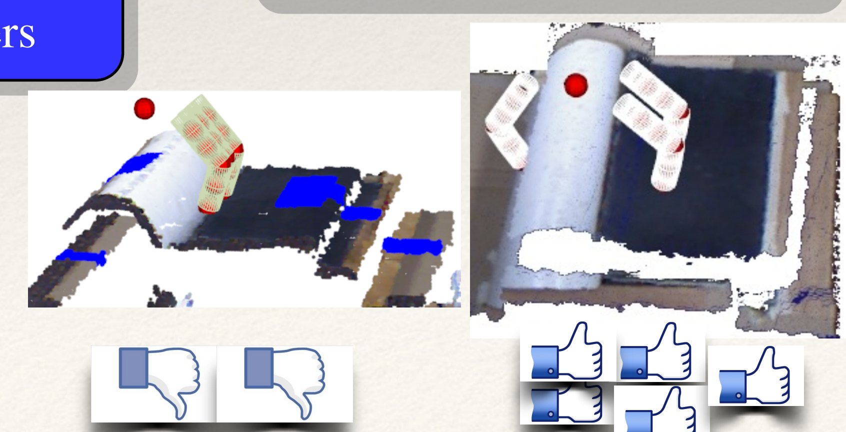
Re-evaluate the actions sampled in every state

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

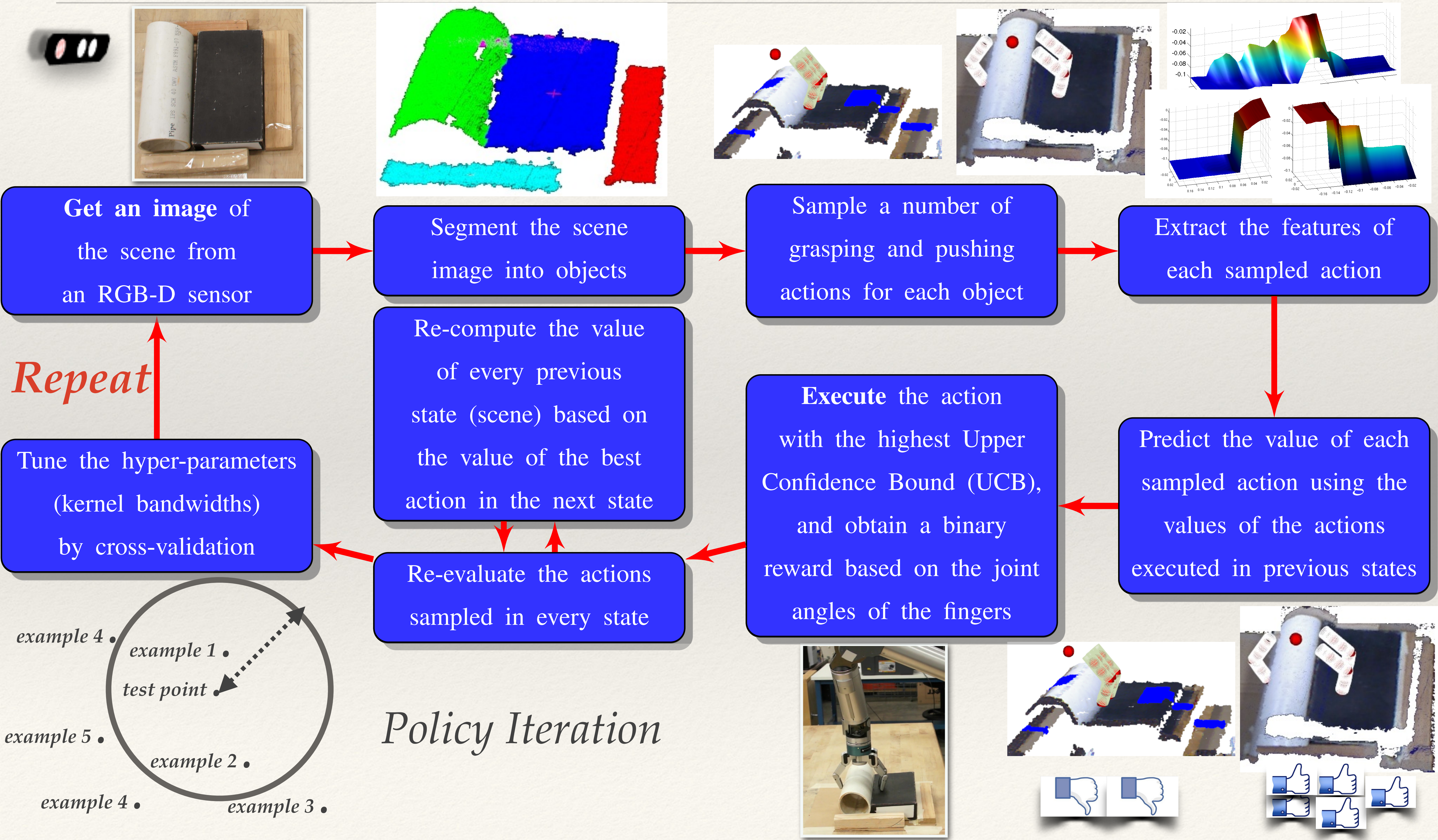
Tune the hyper-parameters (kernel bandwidths) by cross-validation



Policy Iteration



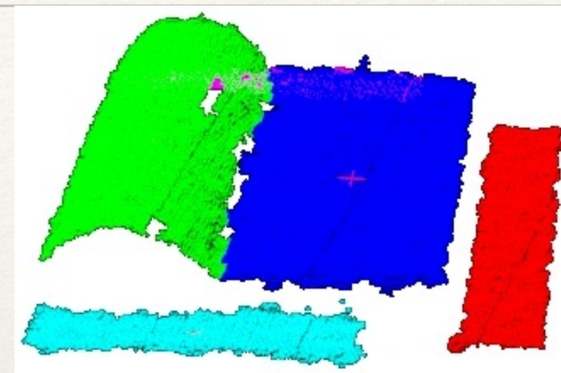
Overview of the integrated system



Segmentation

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

→

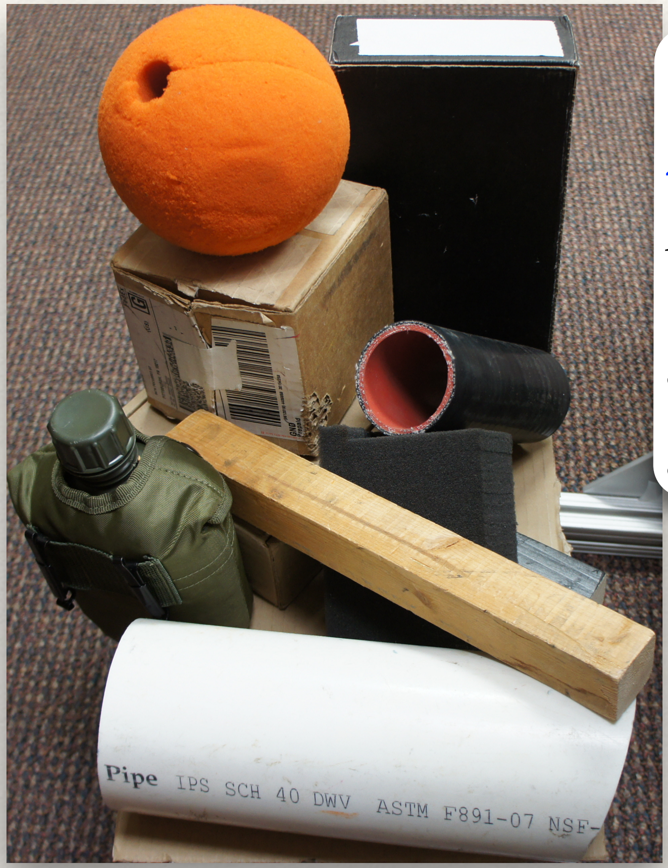


Segment the scene image into objects

The objects are unknown. We make one assumption: the shape of an object is ***overall convex***.

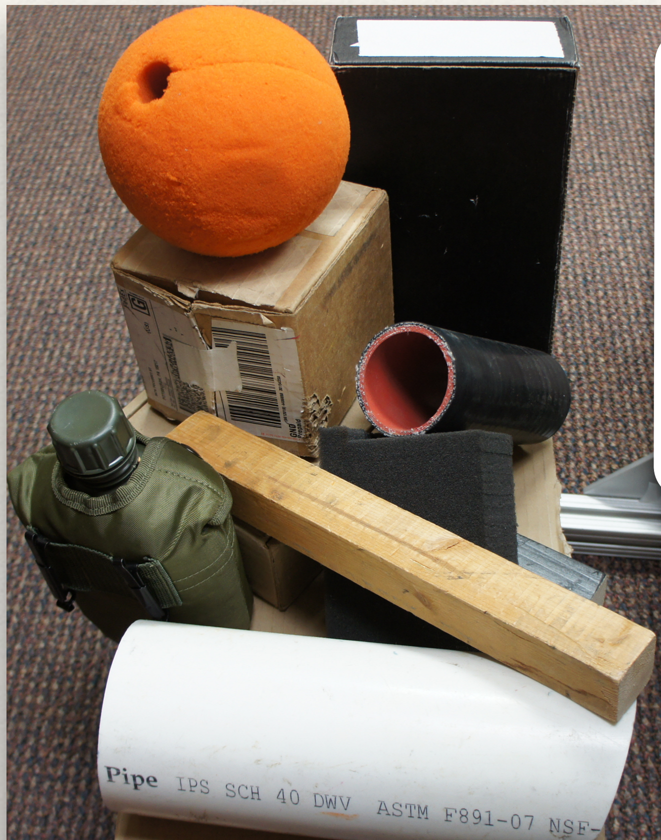
For real-time segmentation, we use a cascade of algorithms.

Segmentation



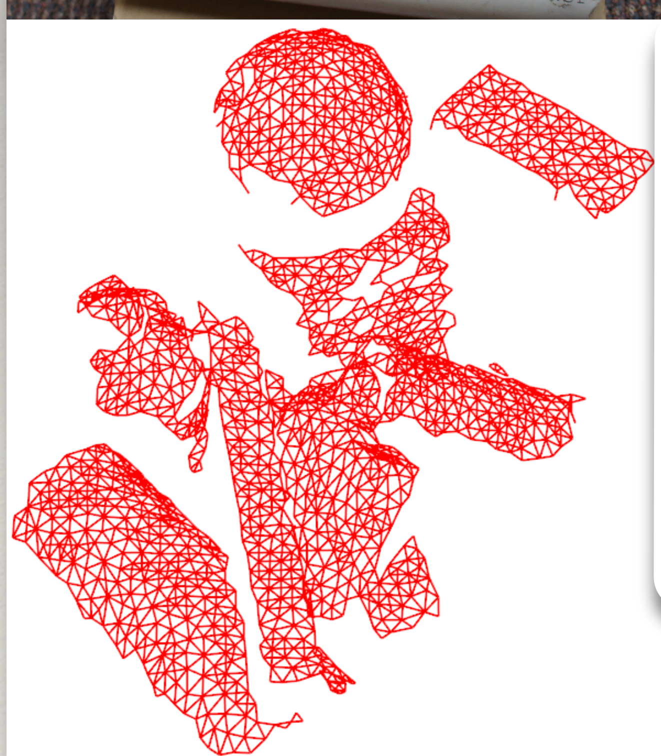
1. Detect and remove the support surface by using the *RANSAC* algorithm (Fischler and Bolles 1981).

Segmentation

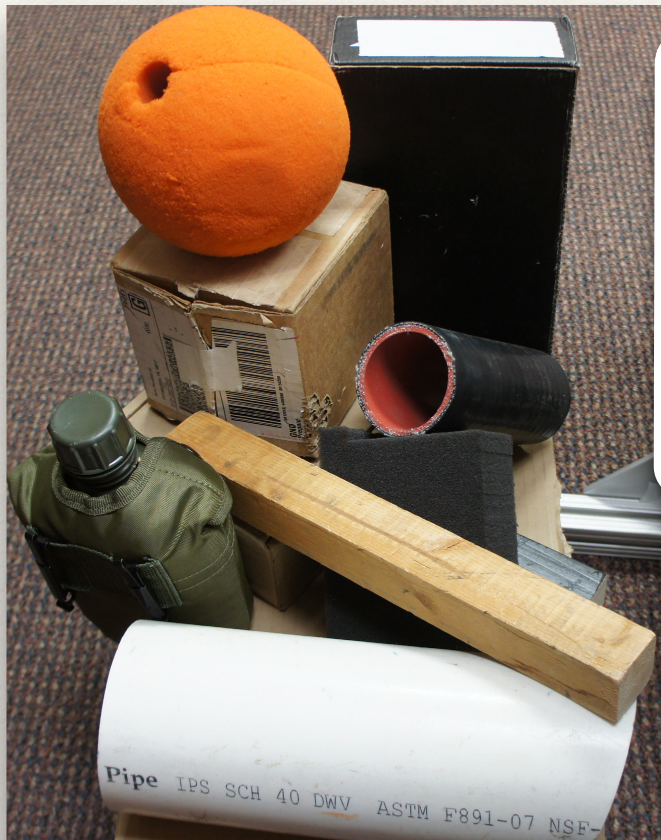


1. **Detect and remove the support surface** by using the *RANSAC* algorithm (Fischler and Bolles 1981).

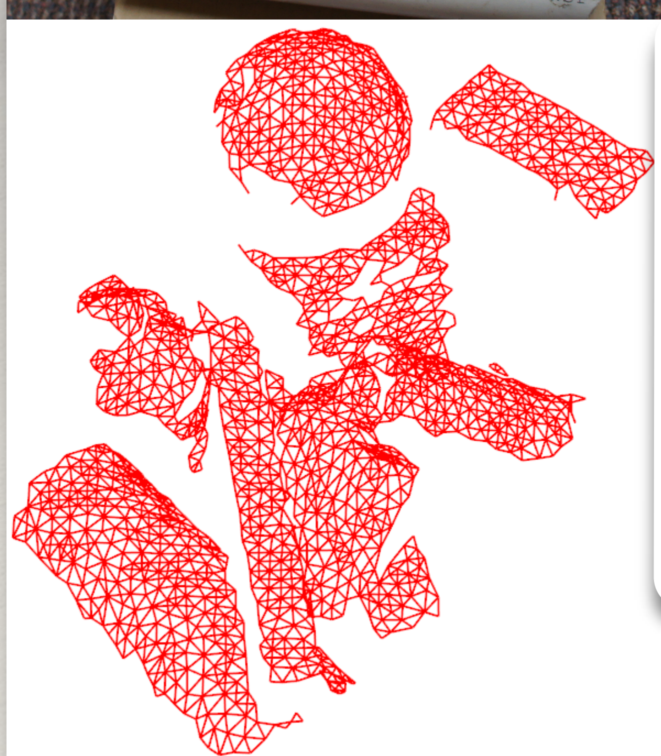
2. Cluster the voxels into ***supervoxels*** with a fast, local, *k-means* based on depth and color properties (Papon et al. 2013).



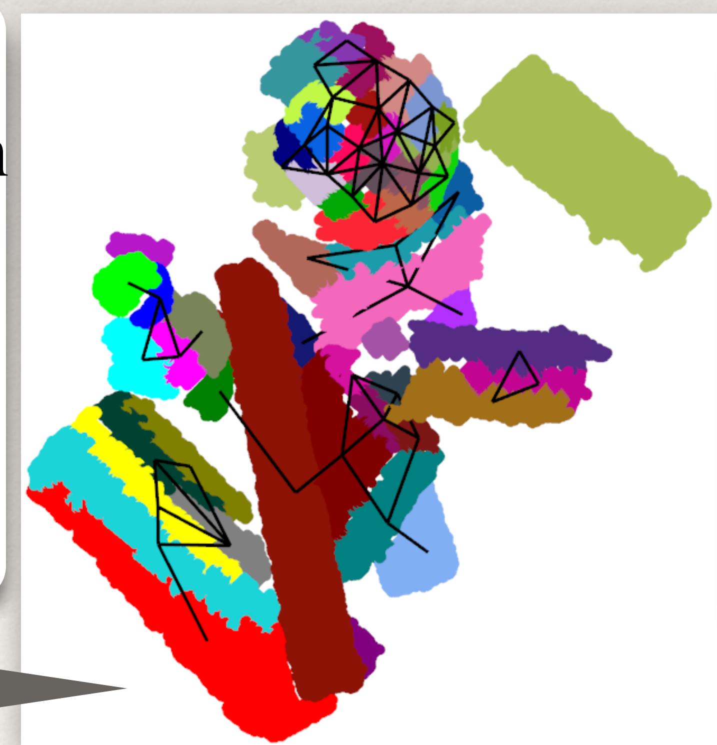
Segmentation



1. **Detect and remove the support surface** by using the *RANSAC* algorithm (Fischler and Bolles 1981).

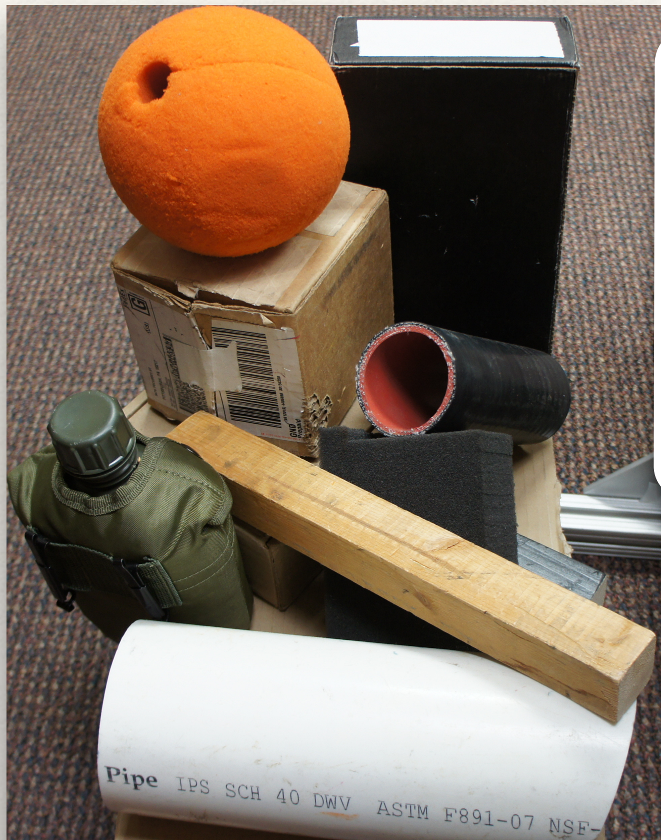


2. Cluster the voxels into ***supervoxels*** with a fast, local, *k-means* based on depth and color properties (Papon et al. 2013).



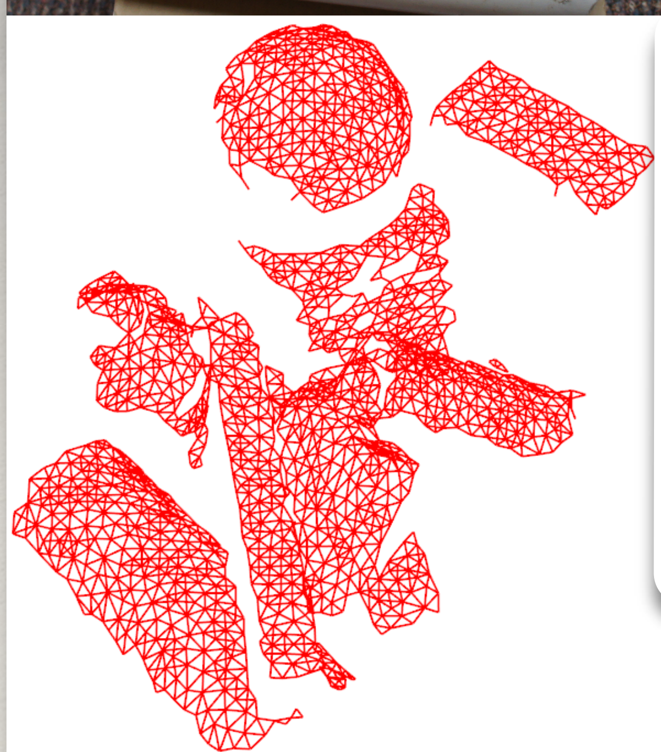
3. Cluster the supervoxels into ***facets*** (flat contiguous regions), using the *mean-shift* algorithm.

Segmentation



1. **Detect and remove the support surface** by using the *RANSAC* algorithm (Fischler and Bolles 1981).

4. Cluster the facets into **objects**, using the *spectral clustering* algorithm.



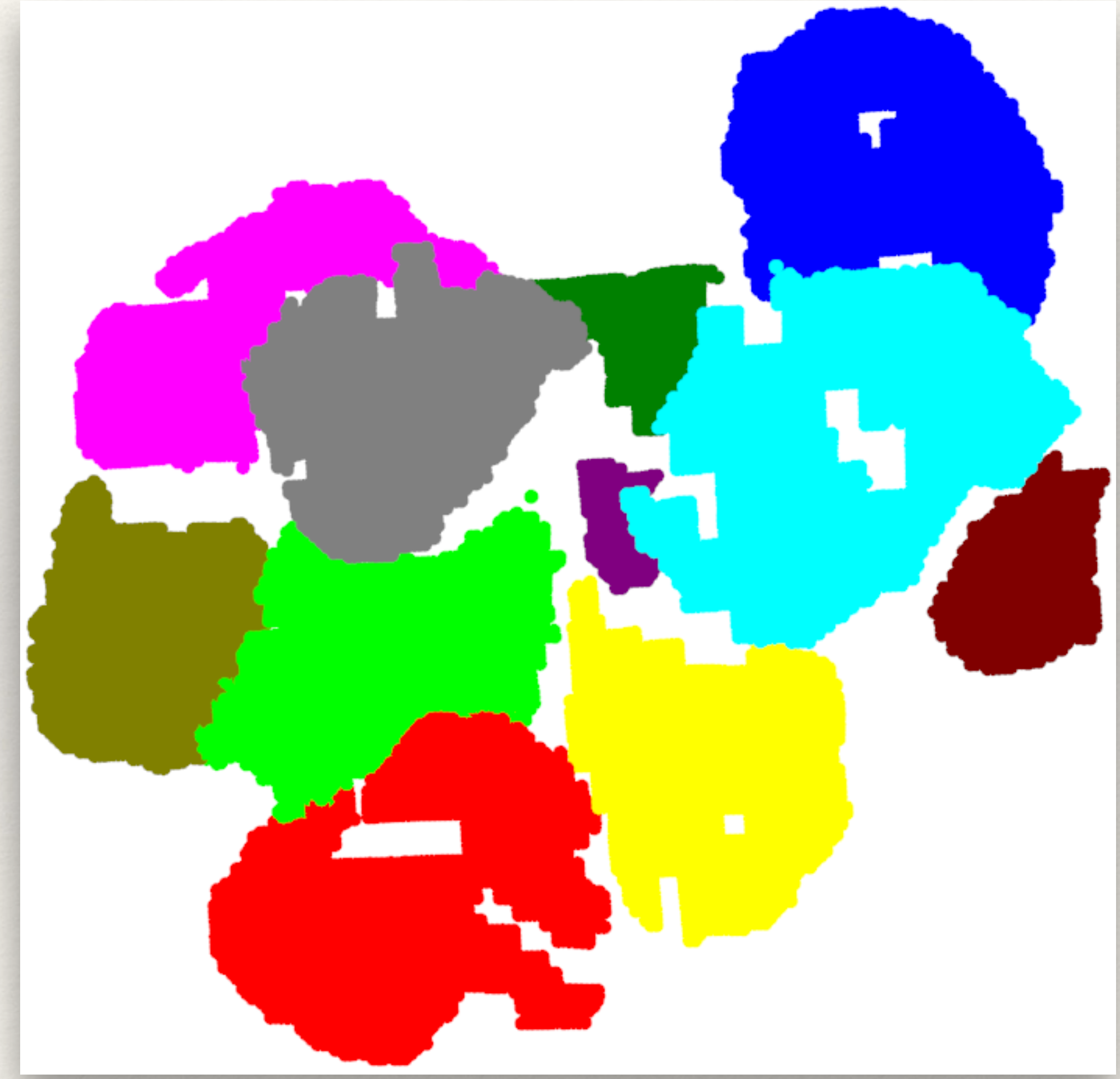
2. Cluster the voxels into ***supervoxels*** with a fast, local, *k-means* based on depth and color properties (Papon et al. 2013).

3. Cluster the supervoxels into ***facets*** (flat contiguous regions), using the *mean-shift* algorithm.

Segmentation

- The proposed approach works also with natural objects, such as rocks.

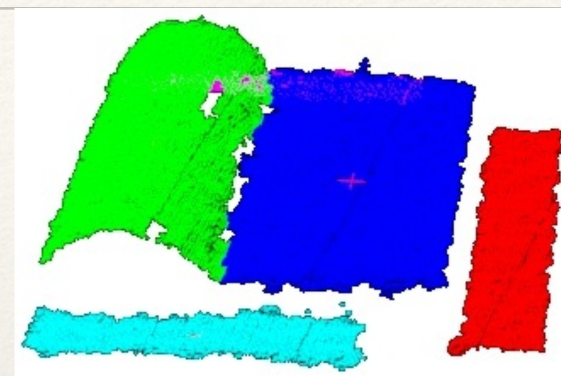
Pile of rocks



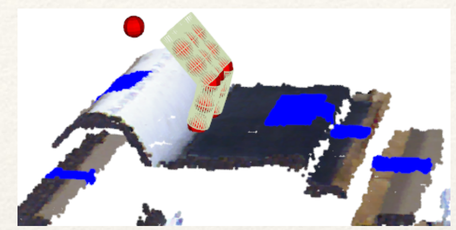
Segmented image

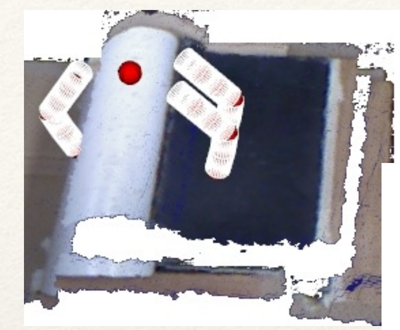
Extracting Features

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

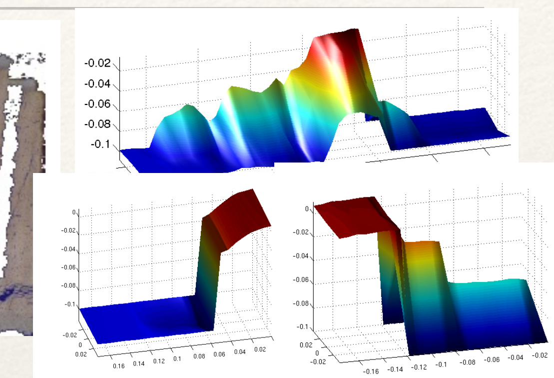


Segment the scene image into objects



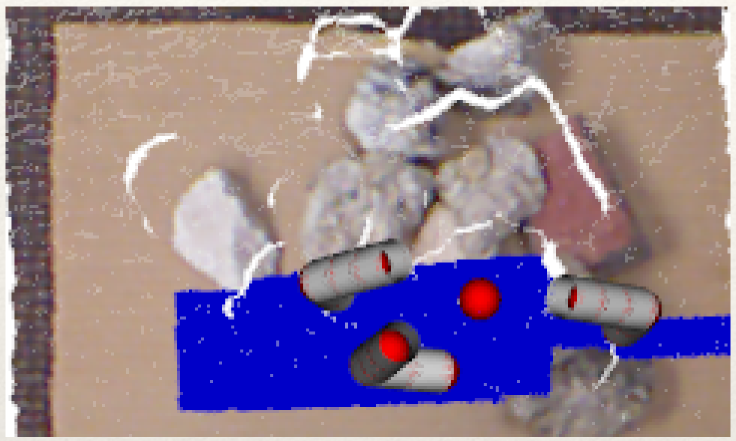


Sample a number of grasping and pushing actions for each object

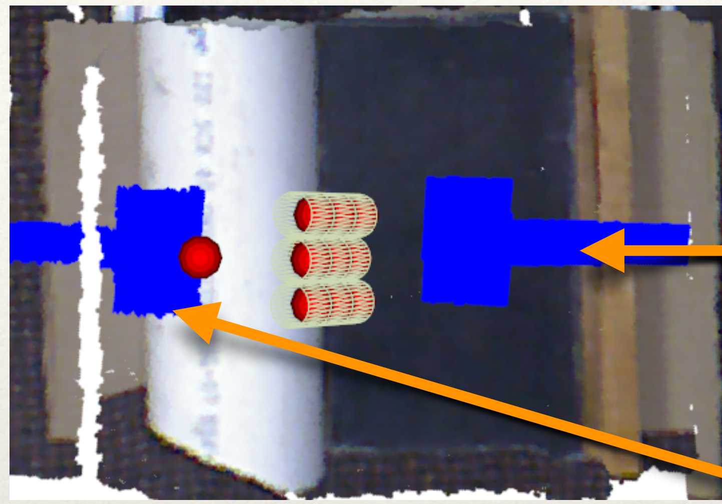


Extract the features of each sampled action

Extracting Features

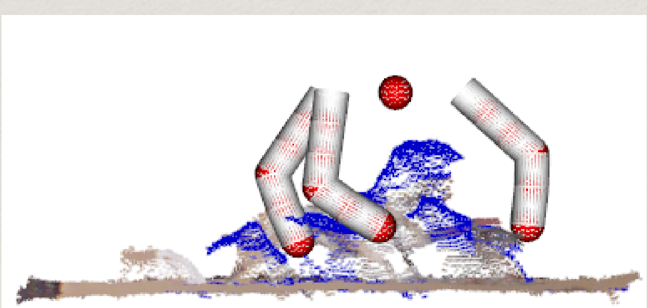


(a) Grasp action (top view)

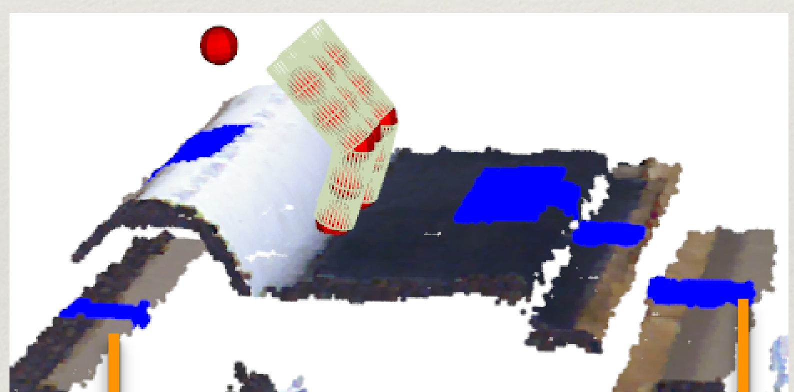


(b) Push action (top view)

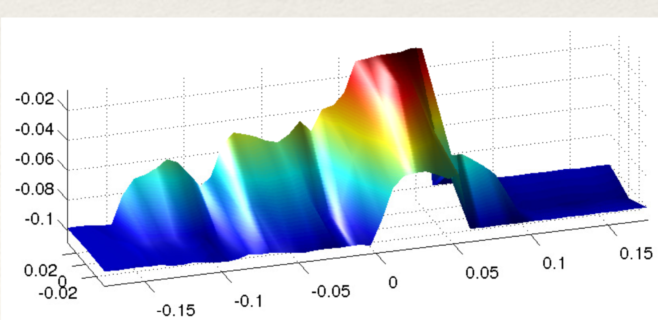
Grasping features of the pushed object's neighbors
+
Patch of the depth image in the pushing direction



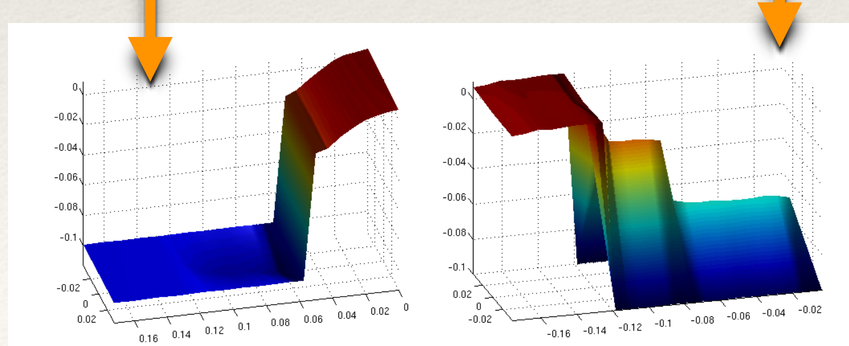
(c) Grasp action (side view)



(d) Push action (side view)



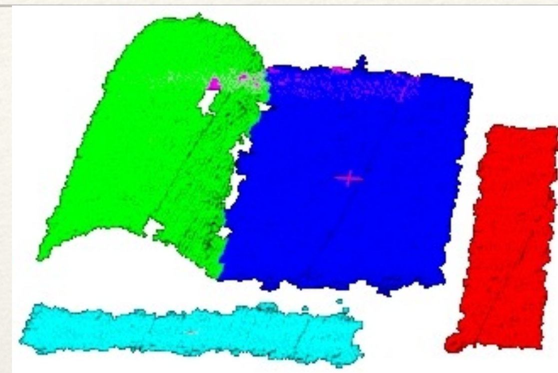
(e) Grasp features



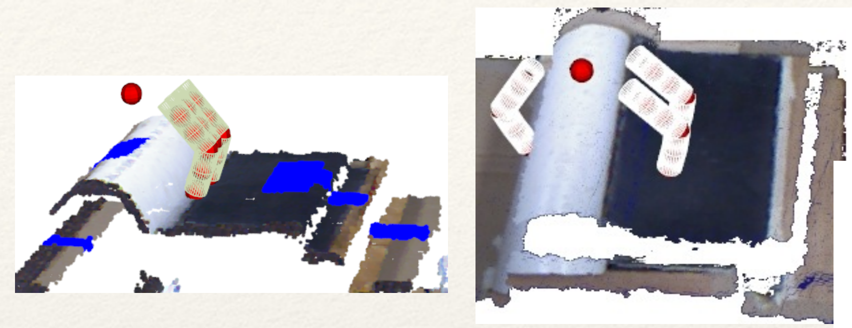
(f) Push features

Action Evaluation

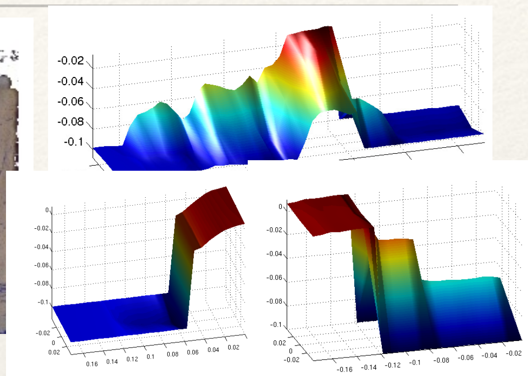
Get an image of the scene from an RGB-D sensor



Segment the scene image into objects

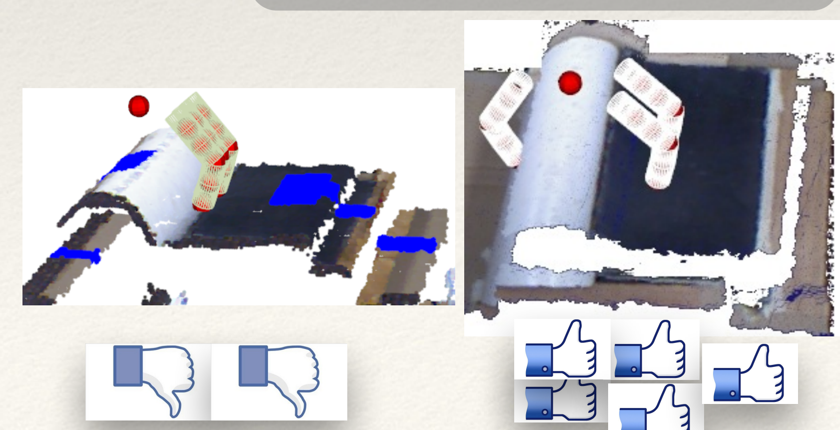


Sample a number of grasping and pushing actions for each object



Extract the features of each sampled action

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



Action Evaluation

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.

Action Evaluation

The value (expected sum of rewards) of an action ***a*** in a state ***s*** is predicted as

Empirical value

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

Current state and action

Similarity measure (Kernel)

Data: state (image) and action at time i

Action Evaluation

Grasping or Pushing

$$K\big((s_i, a_i), (s, a)\big) = \begin{cases} 1 & \text{if } \big(\text{type}(a) = \text{type}(a_j)\big) \wedge \\ & \quad \big(\|\phi(s_i, a_i) - \phi(s, a)\|_2 \leq \epsilon_{\text{type}(a)}\big), \\ 0 & \text{else} \; . \end{cases}$$

Similarity measure (kernel)

Features

Threshold

Exploration versus Exploitation

Each action should be executed sufficiently many times until a certain confidence on its value is attained

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

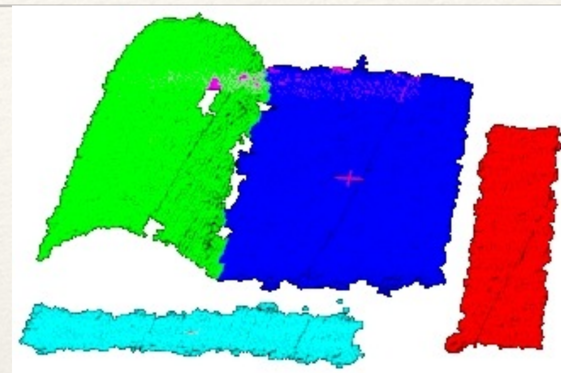
$$\hat{Q}_{\pi^*}(s_t, a) + \alpha \sqrt{\frac{2 \ln t}{\sum_{i=0}^{t-1} K\big((s_i, a_i), (s_t, a)\big)}}.$$

Predicted Value
(for exploitation)

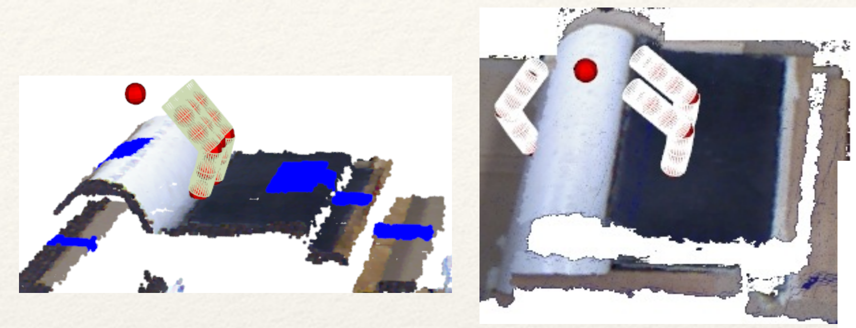
Novelty
(for exploration)

Policy Iteration

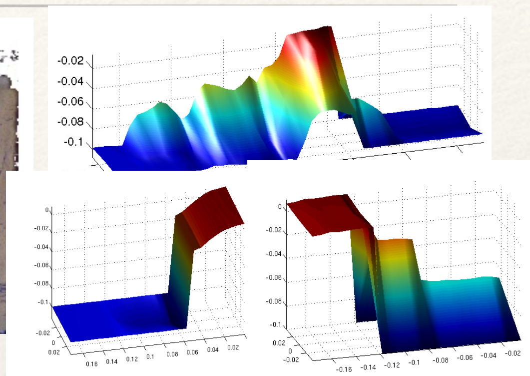
Get an image of the scene from an RGB-D sensor



Segment the scene image into objects



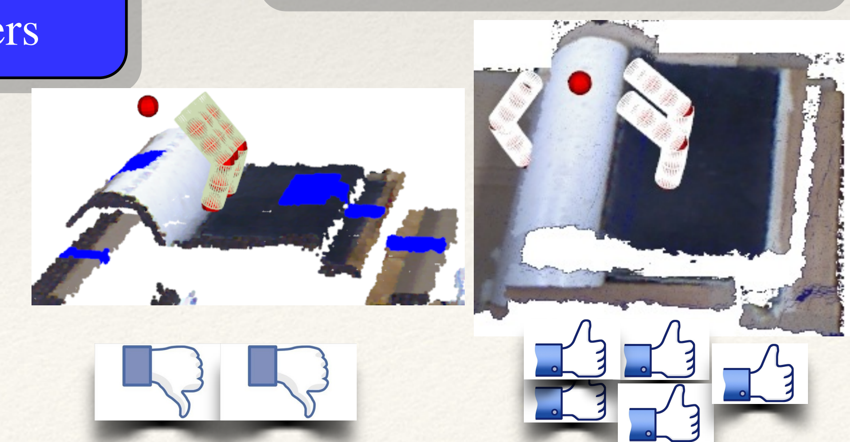
Sample a number of grasping and pushing actions for each object



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-evaluate the actions sampled in every state

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

Policy Iteration

Policy Iteration

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

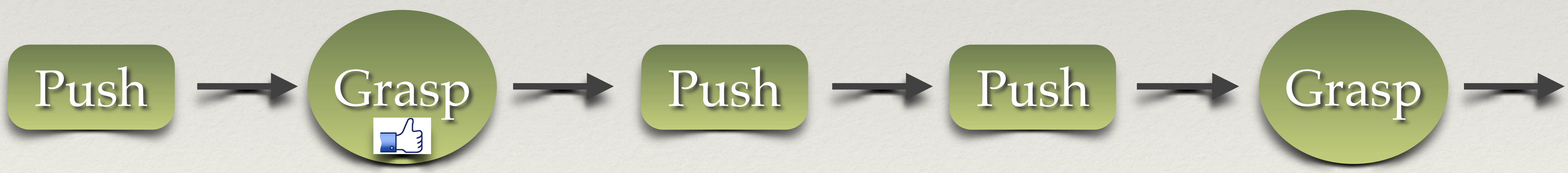
The learned transition and reward functions are used for evaluating and improving policies.

Policy Iteration

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

The learned transition and reward functions are used for evaluating and improving policies.

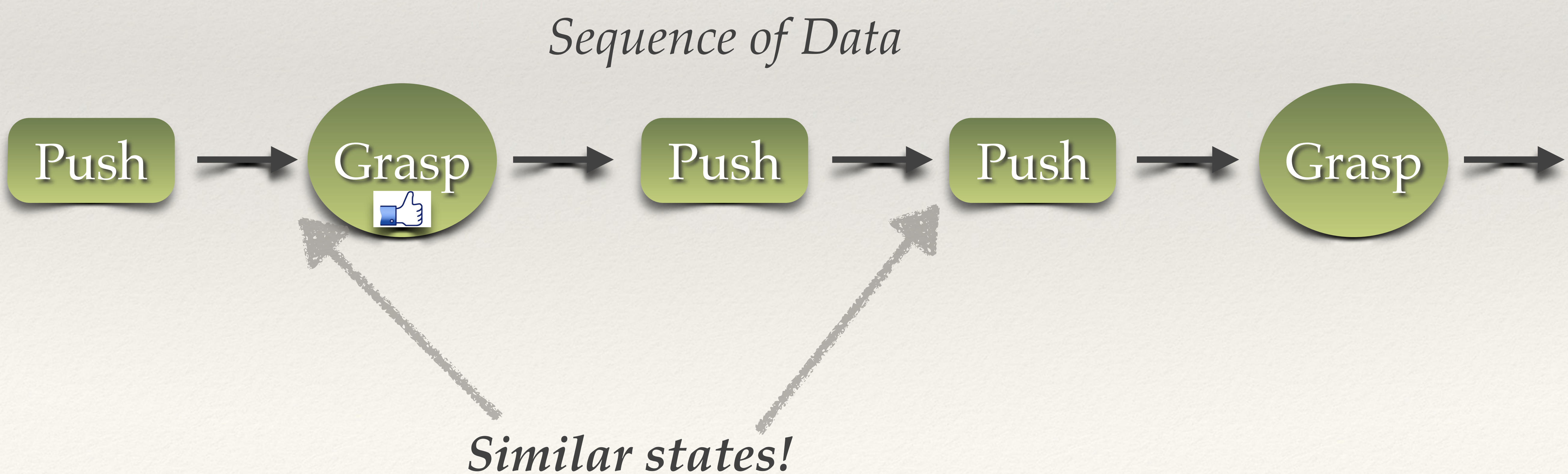
Sequence of Data



Policy Iteration

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

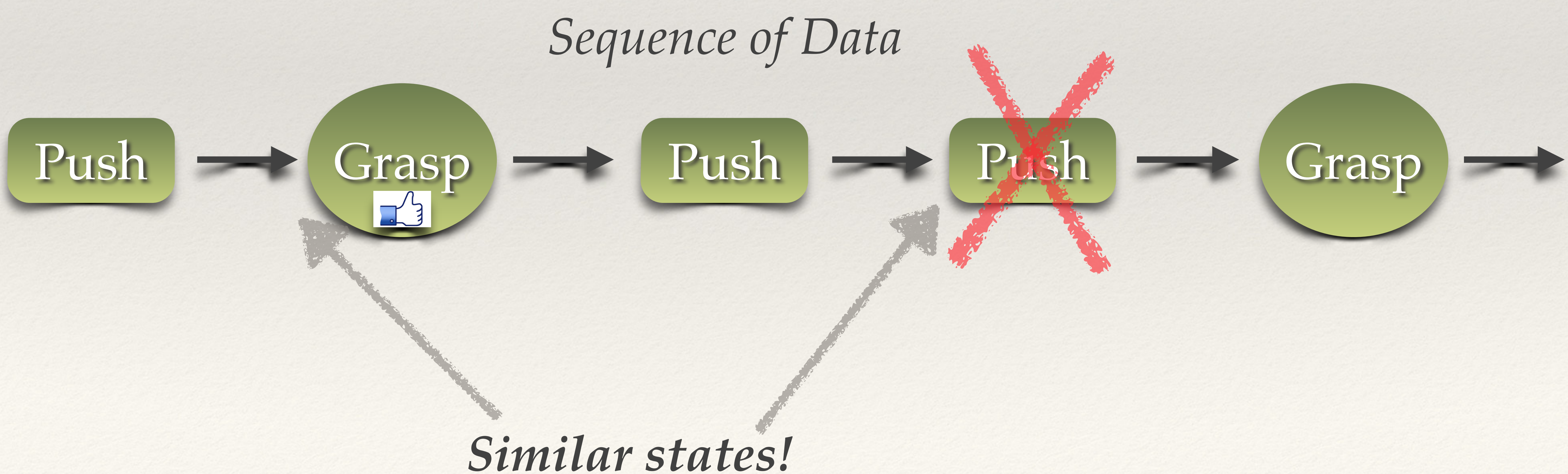
The learned transition and reward functions are used for evaluating and improving policies.



Policy Iteration

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

The learned transition and reward functions are used for evaluating and improving policies.

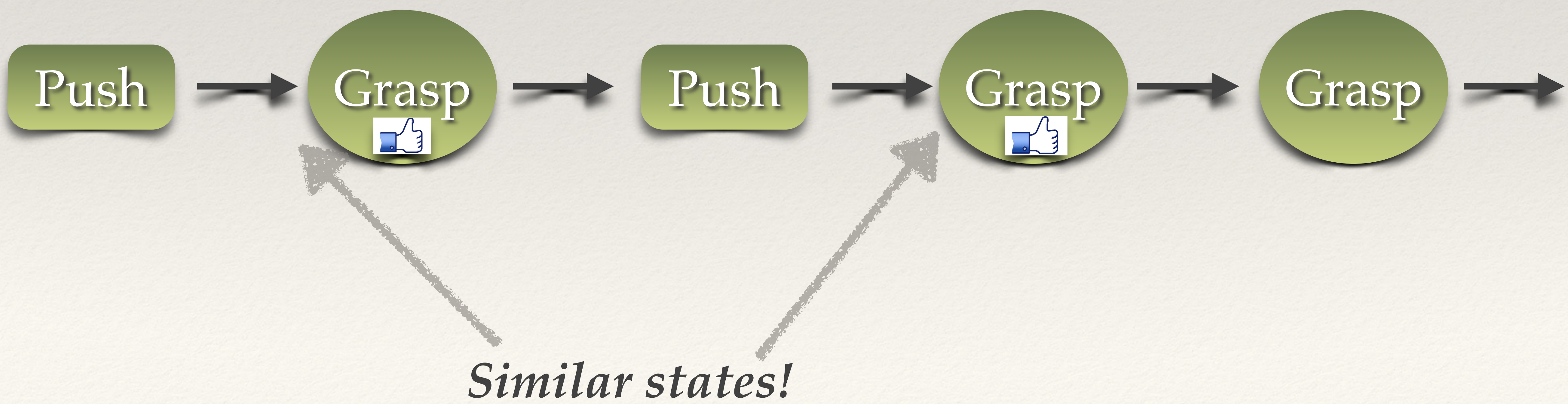


Policy Iteration

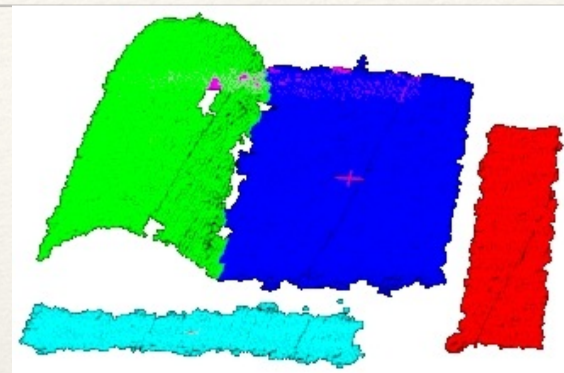
Kernel density estimation for learning the transition function between the states contained in the training data sequence

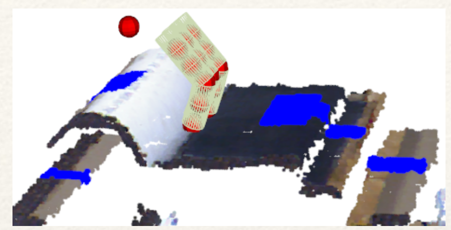
The learned transition and reward functions are used for evaluating and improving policies.

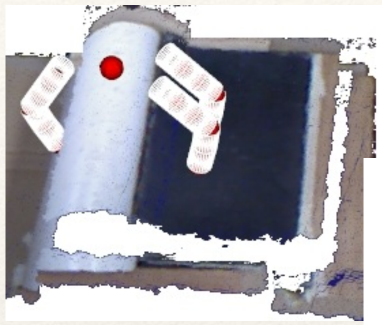
Sequence of Data

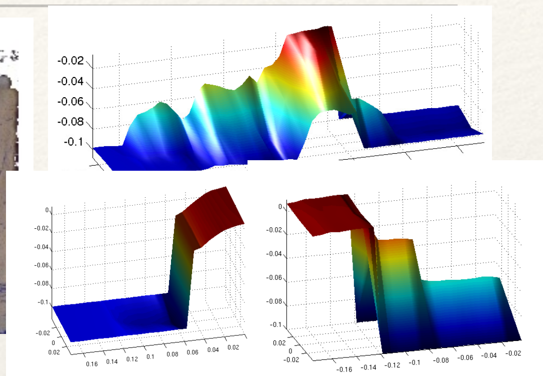


Bandwidth Selection









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

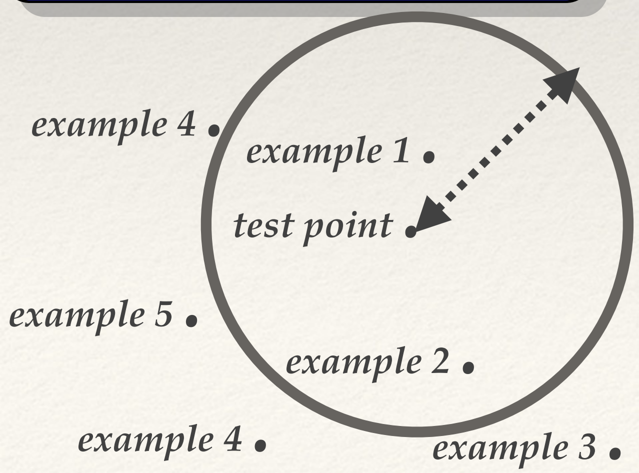
Predict the value of each sampled action using the values of the actions executed in previous states

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

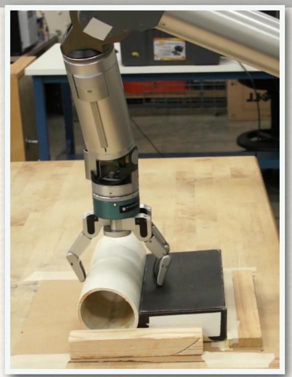
Re-evaluate the actions sampled in every state

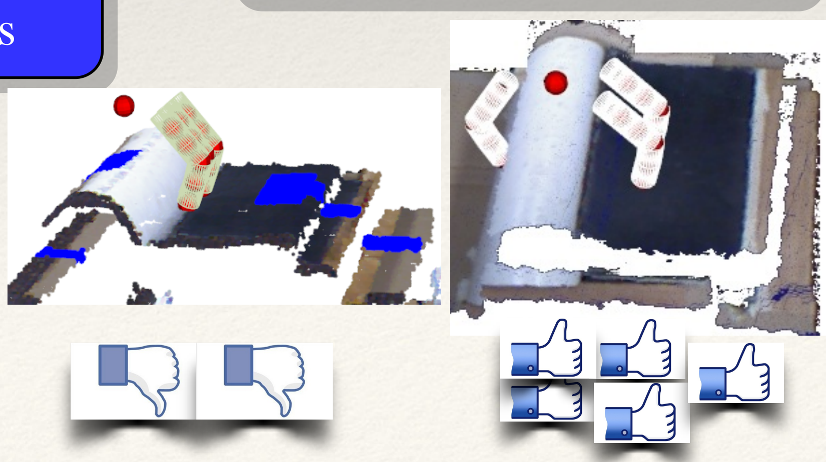
Re-compute the value of every previous state (scene) based on the value of the best action in the next state

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



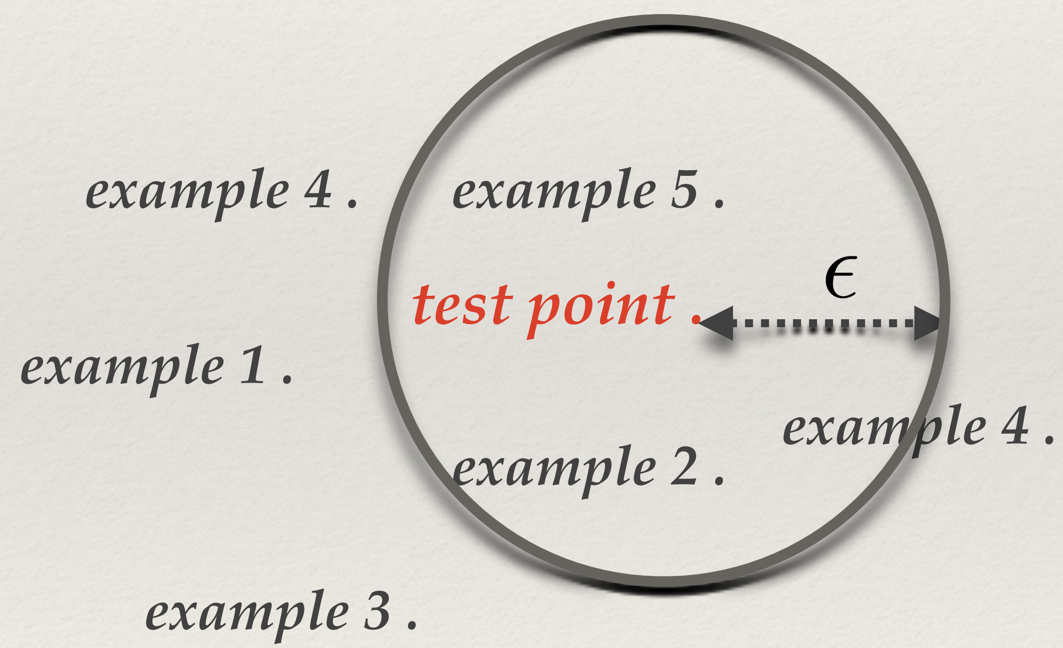
Policy Iteration





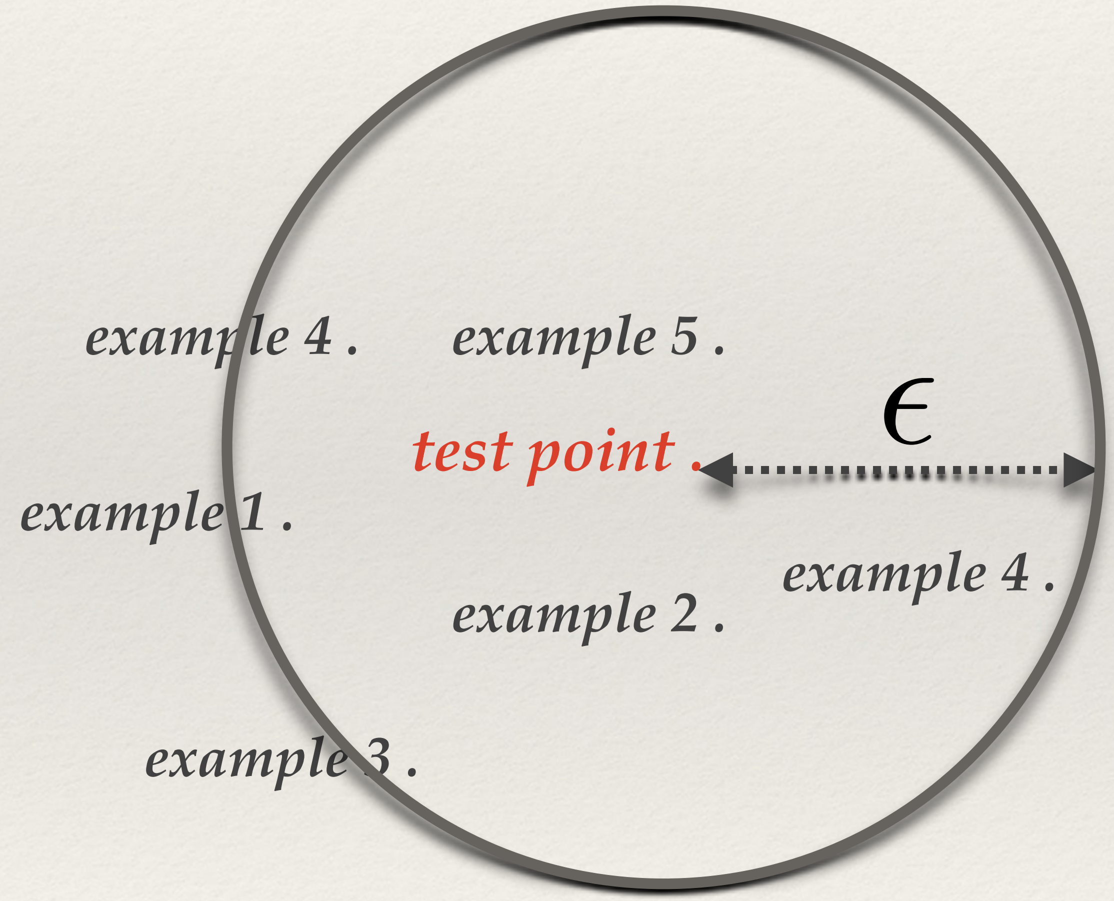
Bandwidth Selection

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,

$$BE(\epsilon) = \frac{1}{t_2 - t_1} \sum_{i=t_1}^{t_2-1} \Big(\underbrace{r_i}_{\text{immediate reward}} + \gamma \underbrace{\hat{V}_{\hat{\pi}}^{\epsilon}(s_{i+1})}_{\text{Predicted value of next state}} - \underbrace{\hat{Q}_{\hat{\pi}}^{\epsilon}(s_i, a_i)}_{\text{Predicted value of current state}} \Big)^2 .$$

Bandwidth Selection

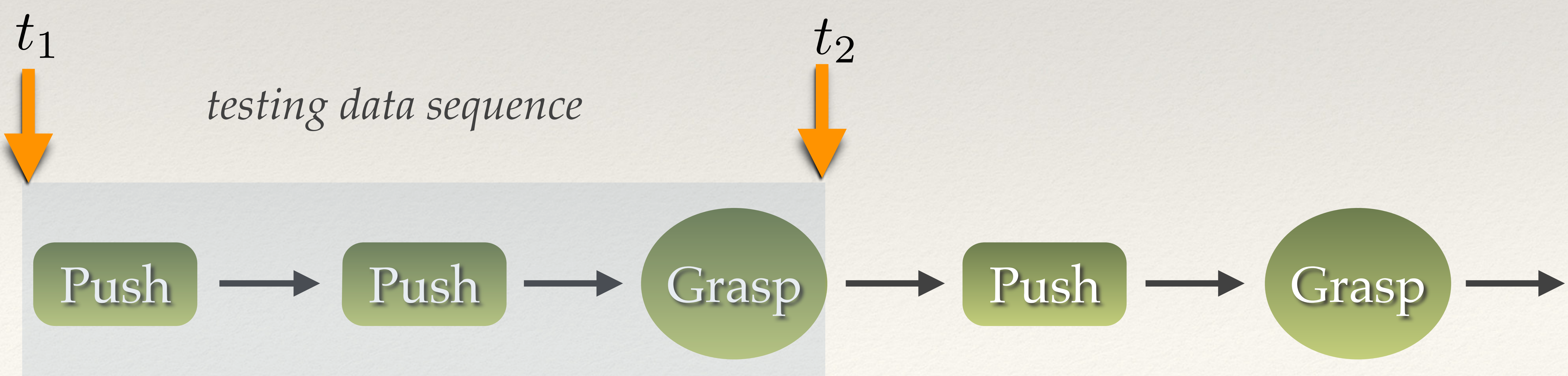
The range is automatically tuned by selecting the threshold that minimizes the *Bellman error* in the training data,

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

immediate reward

Predicted value of next state

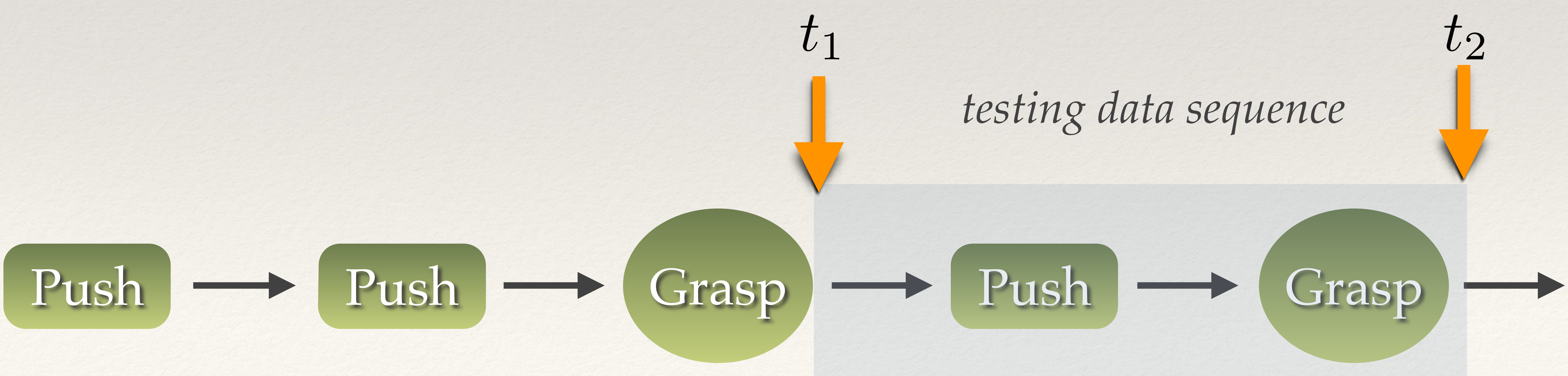
Predicted value of current state



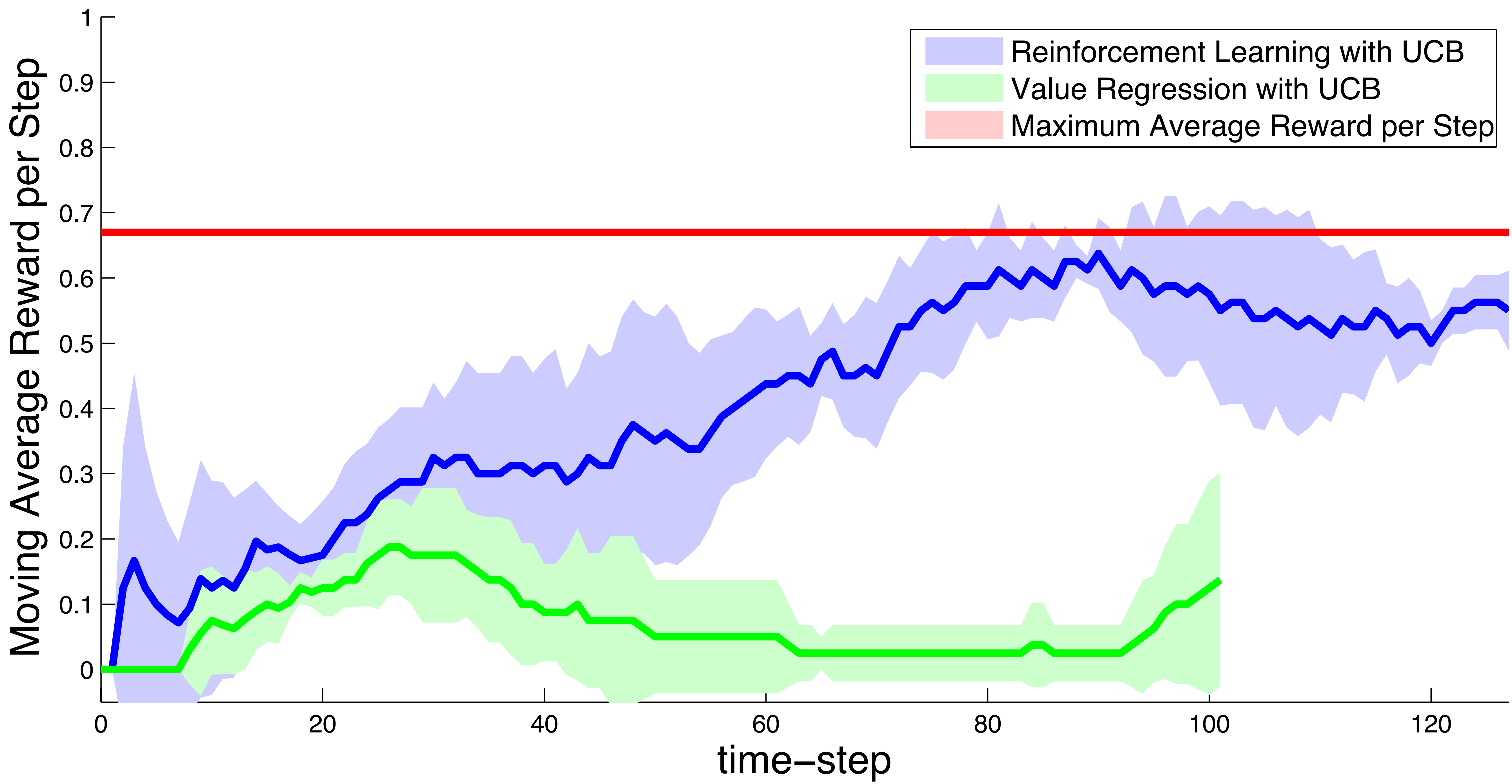
Bandwidth Selection

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

$$BE(\epsilon) = \frac{1}{t_2 - t_1} \sum_{i=t_1}^{t_2-1} \Big(\underbrace{r_i}_{\text{immediate reward}} + \gamma \underbrace{\hat{V}^{\epsilon}_{\hat{\pi}}(s_{i+1})}_{\text{Predicted value of next state}} - \underbrace{\hat{Q}^{\epsilon}_{\hat{\pi}}(s_i, a_i)}_{\text{Predicted value of current state}} \Big)^2 .$$



Learning Curve: Reinforcement Learning V.S. Regression

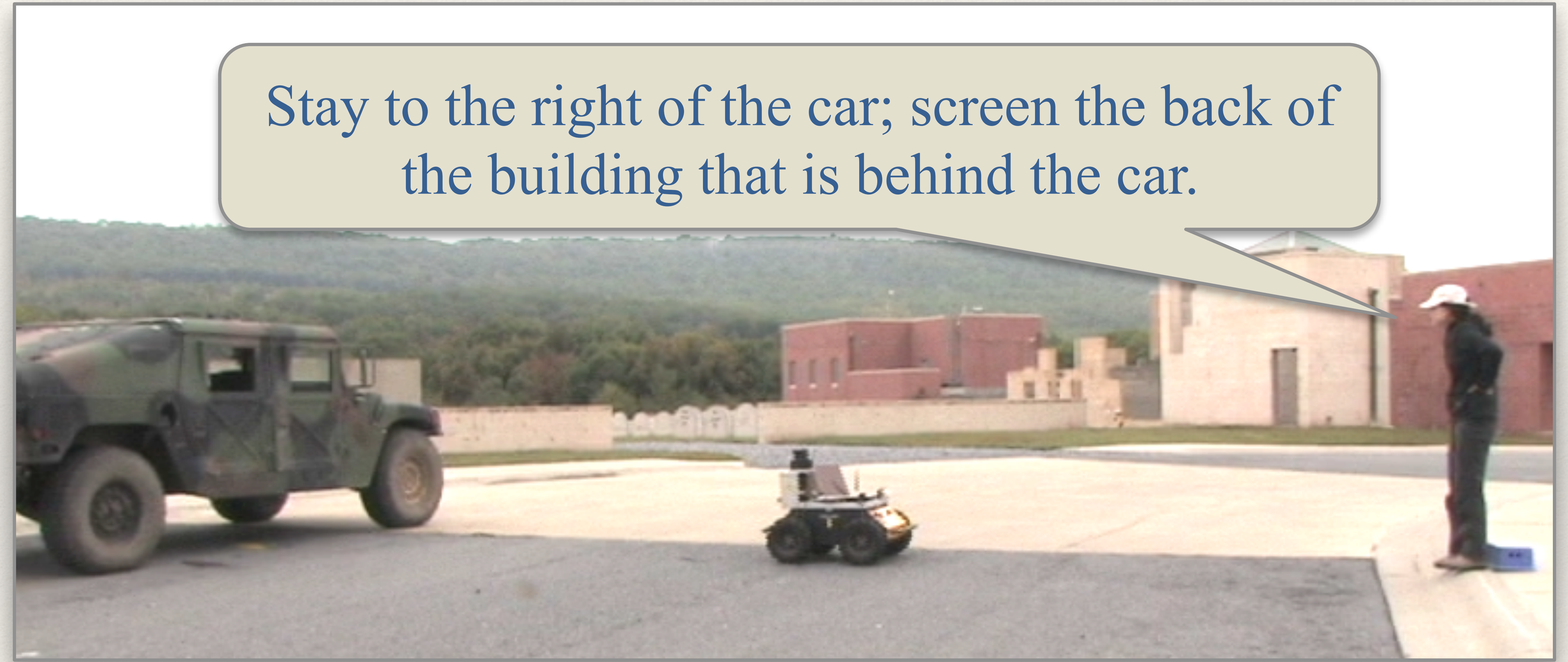


A. Boularias *et al.* (2015) in *Conference of the Association for the Advancement of Artificial Intelligence (AAAI)*

Outline

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

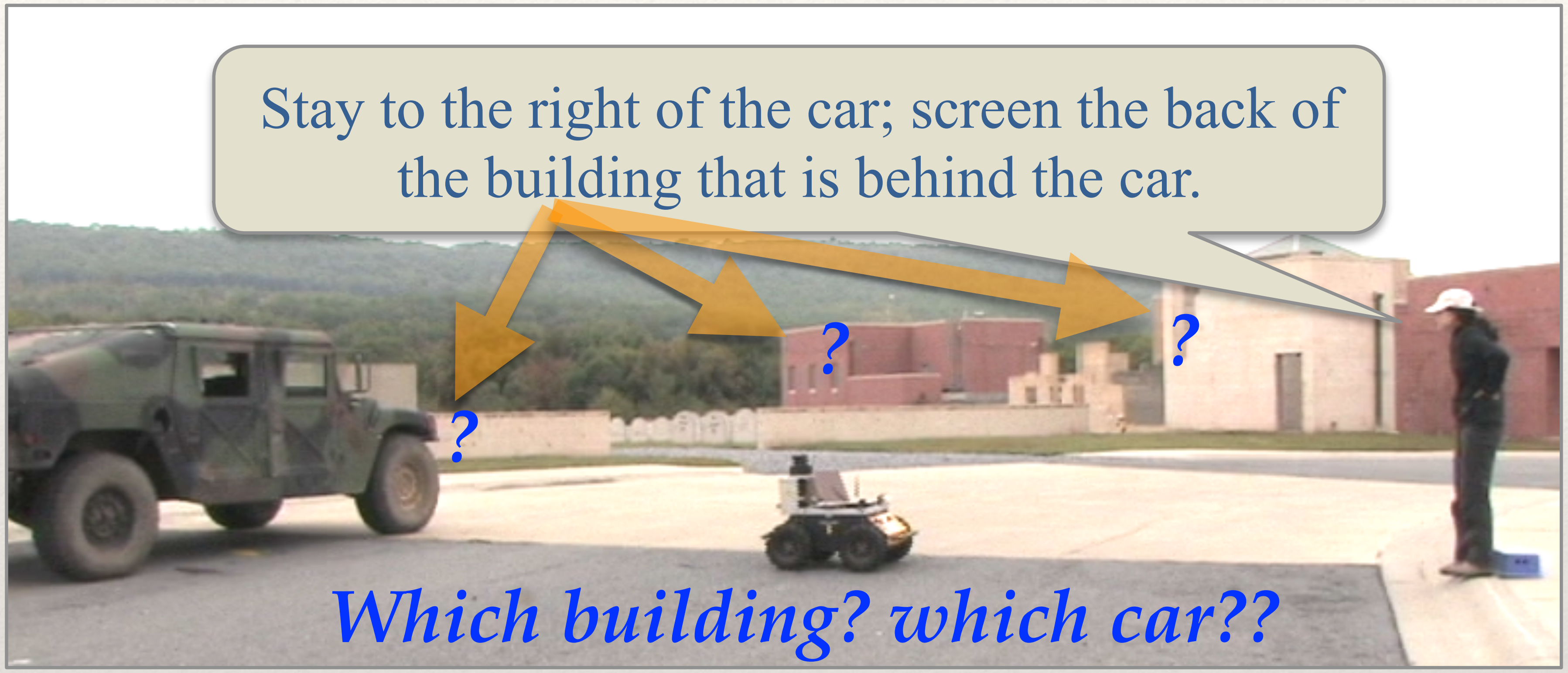
Grounding Spatial Relations for Robot Navigation



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

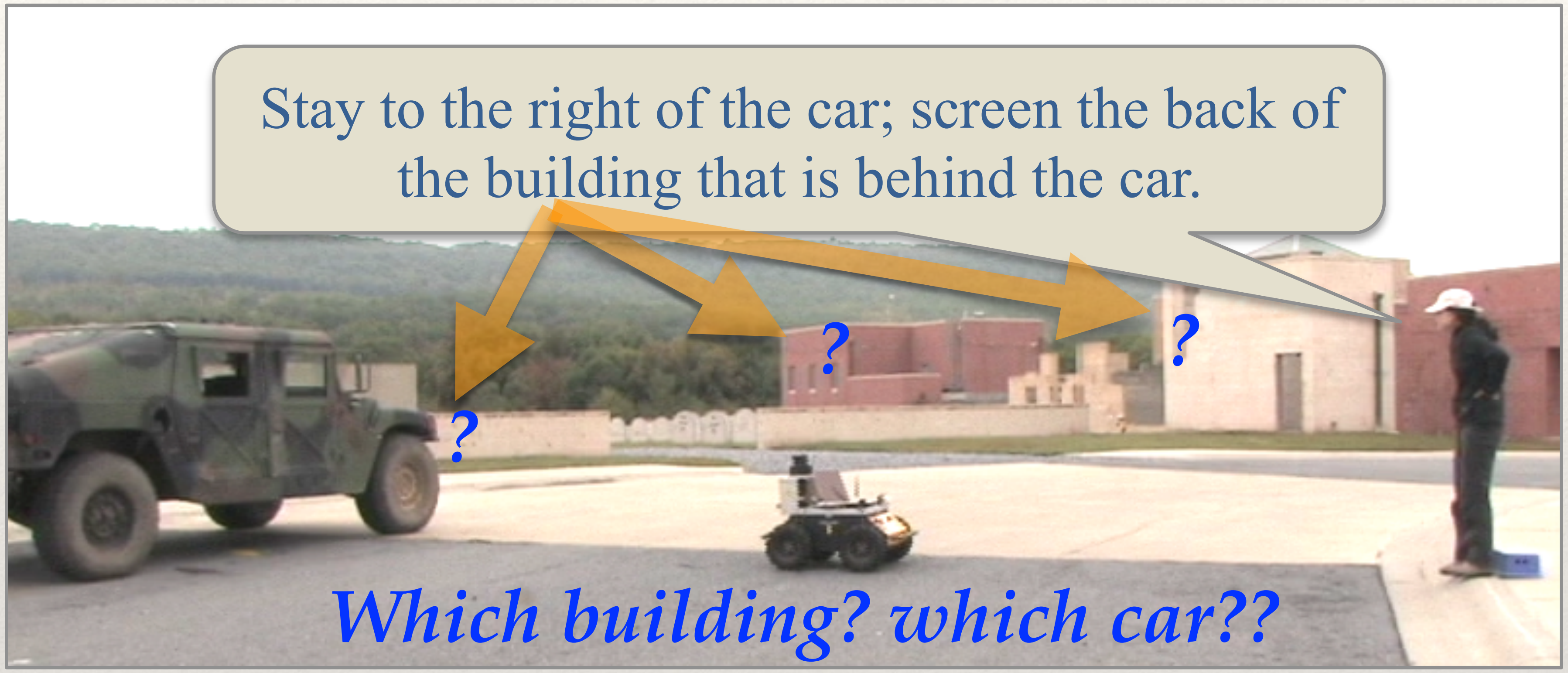
A. Boularias *et al.* (2015) in *IEEE International Conference on Robotics and Automation (ICRA)*

Grounding Spatial Relations for Robot Navigation



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

Grounding Spatial Relations for Robot Navigation

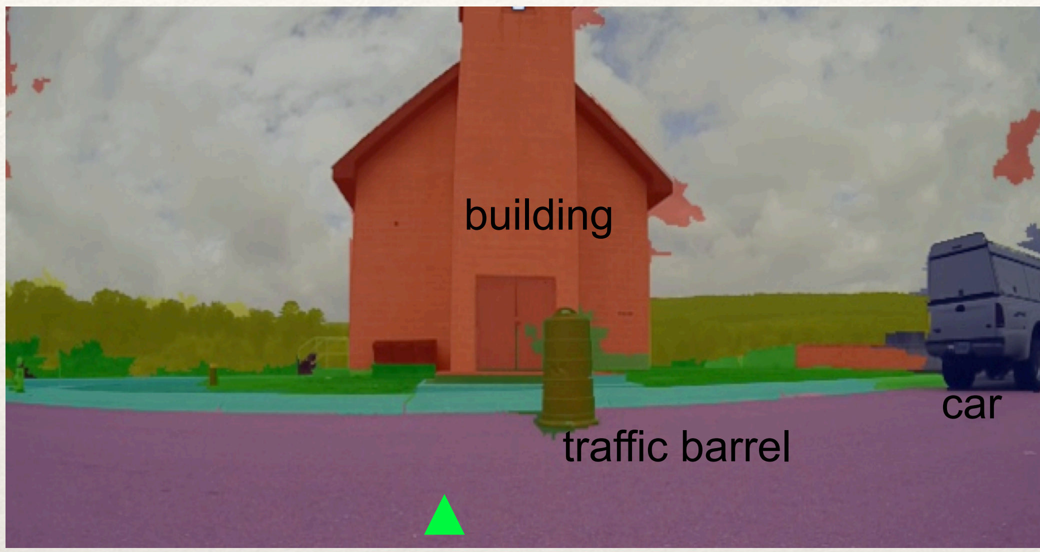


Grounding: map each noun in the command to an object in the world

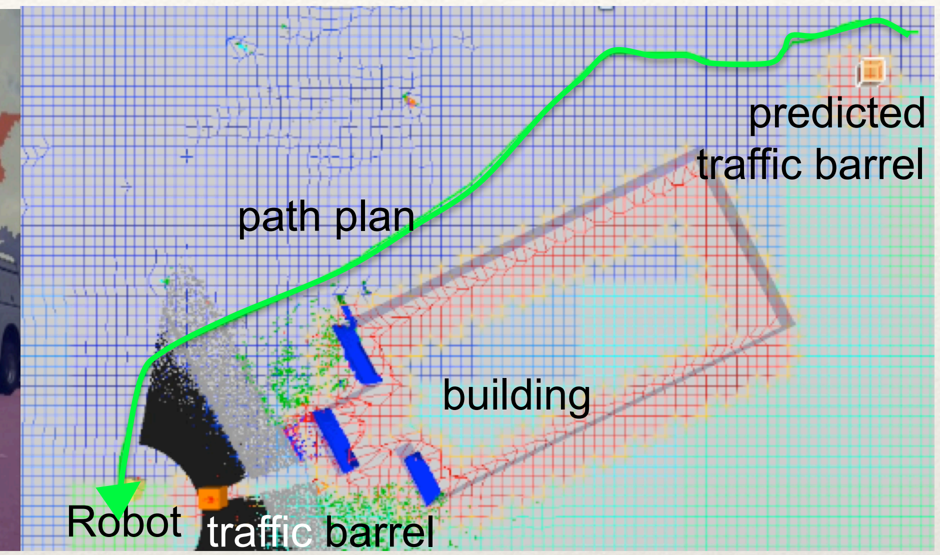
Spatial concepts (such as *behind* and *near*) are learned from examples

Bayesian probabilistic model for dealing with object recognition errors

Grounding Spatial Relations for Robot Navigation



Environment as perceived by the robot



Planned path

<u>Results</u>: the robot navigated to the correct goal 88% of the time.

Merci !