Robot Learning: Algorithms and Applications

Abdeslam Boularias

Carnegie Mellon University
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
How can a robot learn to perform complex tasks from experience?
How can a robot learn to perform complex tasks from experience?
Controlling a Dynamical System
Controlling a Dynamical System
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

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

- \( R \): reward (or cost) function, \( R(s, a) \in \mathbb{R} \)
A policy is a function $\pi$ that maps each state to an action, $\pi : \mathcal{S} \rightarrow \mathcal{A}$. 
The value (or utility) of a policy $\pi$ 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.

![Diagram of Partially Observable Markov Decision Process](image)
Partially Observable Markov Decision Process (POMDP)

*Example:* the robot can sense an obstacle only after bumping into it.
Partially Observable Markov Decision Process (POMDP)

Example: the robot can sense an obstacle only after bumping into it.
Partially Observable Markov Decision Process (POMDP)

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

Intended Target

\[ x_{t-1} \rightarrow x_t \rightarrow x_{t+1} \]

Observations

\[ o_{t-1} \rightarrow o_t \rightarrow o_{t+1} \]

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.
Figure 4: Bar plots show the distribution of the target (X coordinate) at approximately 320ms, 160ms, and 80ms before the player hits the ball. The prediction became increasingly confident as the player finishes the hitting movement, and the robot later chose the default hitting movement accordingly. Figure adapted from Wang et al. (2013).

Assuming that the dynamics of the human player’s racket is driven by the intended target $g$, we can apply the IDDM to predict the target $g$ given a time series of observations $z_1:t$ that are generated from corresponding latent states $x_1:t$. While exact inference of the intention $g$ and states $x_t$ is not tractable, Wang et al. (2013) presented an efficient online inference algorithm to update the belief $p(g, x_t | z_1:t)$, i.e., the posterior probability of the intention $g$ and the latent states $x_t$ once a new observation $z_t$ is obtained.

Figure 4 shows that the predictive uncertainties decrease as the human player finishes the hitting movement.

To summarize, the IDDM provides an estimate of the transition model $T$ and of the measurement model $\phi$ in the considered POMDP with Gaussian processes, which are used for updating the belief.
Figure 4: Bar plots show the distribution of the target (X coordinate) at approximately 320ms, 160ms, and 80ms before the player hits the ball. The prediction became increasingly confident as the player finishes the hitting movement, and the robot later chose the default hitting movement accordingly. Figure adapted from Wang et al. (2013).

Assuming that the dynamics of the human player's racket is driven by the intended target $g$, we can apply the IDDM to predict the target $g$ given a time series of observations $z_{1:t}$ that are generated from corresponding latent states $x_{1:t}$. While exact inference of the intention $g$ and states $x_t$ is not tractable, Wang et al. (2013) presented an efficient online inference algorithm to update the belief $p((g, x_t | z_{1:t}))$, i.e., the posterior probability of the intention $g$ and the latent states $x_t$ once a new observation $z_t$ is obtained.

Figure 4 shows that the predictive uncertainties decrease as the human player finishes the hitting movement. To summarize, the IDDM provides an estimate of the transition model $T$ and of the measurement model $\Sigma$ in the considered POMDP with Gaussian processes, which are used for updating the belief.
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 $x_{t+1} \sim P(\cdot | s_t)$
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 $x_{t+1} \sim P(\cdot | s_t)$

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

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 $x_{t+1} \sim P(\cdot | s_t)$

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

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

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

No

enough samples?
A Monte-Carlo planning algorithm

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

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

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

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

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

enough samples?

Wait and see

No

Yes

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

strike back!
Figure 8: Performance of the LSPI and MCP methods evaluated in ten repetitions on the test data with 207 valid trials, based on sampled episodes on the training data. The numbers on the X-axis showed the number of sampled episodes on the training data.

3.3. Discussion

We conducted a case study with a simplified human-robot table tennis scenario. We assume that the recruited non-professional players do not deliberately mislead the opponent by changing their intended target during the execution of stroke. More generally, IDDM assumes that the intention is time-invariant and the main driving factor when learning the dynamics model. For example, the speed and spin of the table tennis ball, which also affect the dynamics of strokes, are not explicitly considered in the model. This assumption does not necessarily limit the capability of IDDM for intention inference (Wang et al., 2013) and planning. Moreover, this assumption can be further relaxed by taking into account other driving factors as exogenous variables in the dynamics model, leading to Hierarchical Gaussian Process Dynamics Models (H-GPDMs), discussed by (Wang, 2013). Note that, however, the method is not applicable for adversarial planning, as the assumption of time-invariant intention is violated in adversarial scenarios. Incorporation of game-theoretic perspectives in the framework of H-GPDMs is a future direction of this work.

Despite the simplification, the case study clearly showed the importance of combining anticipation and planning and the feasibility of presented methods. We discuss implications of the experimental results in (Wang, 2015) in Artificial Intelligence Journal.

Average number of successful returns

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
Assumption: The reward is a linear function of state-action features

\[ R \overset{\text{def}}{=} w^T \phi \]

- reward
- unknown weights
- features (e.g. velocity, energy, distance from goal, ..)
Inverse Optimal Control

Assumption: The reward is a linear function of state-action features

\[ R \stackrel{\text{def}}{=} w^T \phi \]

Value of policy \( \pi \)

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

Assumption: The reward is a linear function of state-action features

\[ R \overset{\text{def}}{=} w^T \phi \]

Value of policy \( \pi \)

\[ V(\pi) = \sum_{t=0}^{H} E_{s_t} [R(s_t, \pi(s_t))] \]

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

(expected features under \( \pi \))

(e.g. expected energy, distance, etc.)
Inverse Optimal Control: Problem Statement

Given an expert’s policy $\pi^*$, find reward weights $w$ such that:

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

Value of the expert’s policy \hspace{2cm} Value of an arbitrary policy

In other terms, expert’s policy $\pi^*$ has the highest possible value.
Inverse problems are generally ill-posed

Given an expert’s policy $\pi^*$, 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
Relative Entropy Inverse Optimal Control

Find $P$, a probability distribution on state-action trajectories $\tau$

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

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

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 $\tau$.

Solve

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

Subject to:

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

$\text{convex}$

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

Expected features under $P$

Reference distribution

Expected features in the expert’s demonstration
Relative Entropy Inverse Optimal Control

Solution

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

Reference distribution

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.
State $s_t$: position of the ball + positions of the players at time $t$
The average reward differences in the evaluations indicate the edge in the x-direction (direction toward the player). Attractively high negative reward for playing the ball close to y-direction (i.e., along the width of the table) and a relative reward for playing the ball close to the edge in the x-direction. The function yielded by the RE algorithm assigned a little negative reward function for table preferences for Algorithm 3 (MM). The reward of the table had only a small positive influence in the reward function of the RE algorithm (4), respectively.

The weights of all other features for Algorithm 3 (MM) and Algorithm 4 (RE), respectively.

Fig. 7. Resulting parameter values for the individual features.

Red indicates good location for bouncing the ball.

Differences in the average reward between expert and naive player for each feature separately using the entropys algorithm (4) and RE (4).
Average reward of each player is predicted from just the way she/he plays (without looking at the scores)

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

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

Vision: Segment the object into parts
Purposeful Grasping

Vision: Segment the object into parts

Planning: Simulate grasping actions for each part
Purposeful Grasping

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

**Vision:** Segment the object into parts

**Planning:** Simulate grasping actions for each part

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

*Barrett® hand*

Segmented object
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 \(y_i \in \{1, -1\}\).
Learning Grasping Points with Associative Markov Networks

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

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

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

\[
P(Y) \propto \exp \left( \sum_{i \in V} y_i w_{node}^T \phi_i + \sum_{(i,j) \in E} w_{edge}^T \phi_{ij} \right)
\]
Learning Grasping Points with Associative Markov Networks

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

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

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

\[
P(Y) \propto \exp \left( \sum_{i \in V} y_i w_{\text{node}}^T \phi_i + \sum_{(i,j) \in E \text{ such that } y_i = y_j} w_{\text{edge}}^T \phi_{ij} \right)
\]

weight vectors
Learning Grasping Points with Associative Markov Networks

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

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

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

\[
P(Y) \propto \exp \left( \sum_{i \in V} y_i w^T_{\text{node}} \phi_i + \sum_{(i,j) \in E} w^T_{\text{edge}} \phi_{ij} \right)
\]
Associative Markov Networks

Logistic Regression

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

Setup

Percentage of successful grasps

Associative Markov Network
Logistic Regression

- watering can
- plastic basket
- woven basket

A. Boularias et al. (2011) in IEEE International Conference on Intelligent Robots and Systems (IROS)
An Autonomous Robot for Rubble Removal

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)
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

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

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

\textit{Unknown function} \( f \)

Output: stability of the simulated grasping action
Bayesian Black-box Optimization

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.
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

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 $x$:

$$P_{max}(x) \triangleq P(x = \arg \max_{\tilde{x} \in \mathbb{R}^n} f(\tilde{x})) = \int_{f: \mathbb{R}^n \to \mathbb{R}} p(f) \prod_{\tilde{x} \in \mathbb{R}^n - \{x\}} \Theta(f(x) - f(\tilde{x})) \, df$$

Heaviside step function
Greedy Entropy Search

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

$$P_{\text{max}}(x) \triangleq P(x = \arg \max_{\tilde{x} \in \mathbb{R}^n} f(\tilde{x})) = \int_{f : \mathbb{R}^n \rightarrow \mathbb{R}} p(f) \prod_{\tilde{x} \in \mathbb{R}^n - \{x\}} \Theta(f(x) - f(\tilde{x})) \, df$$

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

$$-P_{\text{max}}(x) \log (P_{\text{max}}(x))$$
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**)

![Graph showing the comparison of Grid Search, Entropy Search, and Greedy Entropy Search methods over time. The x-axis represents time in seconds, and the y-axis represents error. The lines indicate the reduction in error over time for each method.](image-url)
Anytime Optimization

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

Grid Search
Entropy Search
Greedy Entropy Search
Anytime Optimization

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

![Graph showing error over time for different optimization methods: Grid Search, Entropy Search, Greedy Entropy Search. Error decreases over time with different slopes for each method.](graph.png)
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
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

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

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

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

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

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

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

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

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
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-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

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
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).

---

*Image of various objects with a highlight box.*
Segmentation

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

2. Cluster the voxels into superv Voxels with a fast, local, \textit{k-means} based on depth and color properties (Papon et al. 2013).
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.
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.

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

Grasping features of the pushed object’s neighbors +
Patch of the depth image in the pushing direction
Action Evaluation

1. Get an image of the scene from an RGB-D sensor
2. Segment the scene image into objects
3. Sample a number of grasping and pushing actions for each object
4. Extract the features of each sampled action
5. 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.
The value (expected sum of rewards) of an action $a$ in a state $s$ is predicted as

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

**Similarity measure (Kernel)**

**Current state and action**

**Data: state (image) and action at time $i$**

**Empirical value**
Action Evaluation

\[ K((s_i, a_i), (s, a)) = \begin{cases} 
1 & \text{if } (\text{type}(a) = \text{type}(a_j)) \land \\
 & (\| \phi(s_i, a_i) - \phi(s, a) \|_2 \leq \epsilon_{\text{type}(a)}), \\
0 & \text{else}. 
\end{cases} \]
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:

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

1. Get an image of the scene from an RGB-D sensor.
2. Segment the scene image into objects.
3. Sample a number of grasping and pushing actions for each object.
4. Extract the features of each sampled action.
5. Predict the value of each sampled action using the values of the actions executed in previous states.
6. Execute the action with the highest Upper Confidence Bound (UCB), and obtain a binary reward based on the joint angles of the fingers.
7. Re-compute the value of every previous state (scene) based on the value of the best action in the next state.
8. Re-evaluate the actions sampled in every 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.
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

```
Push  Grasp  Push  Push  Push  Grasp
```

*Similar states!*
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

Sequence: Push → Grasp → Push → Grasp → Grasp

**Similar states!**
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

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

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
Bandwidth Selection

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

diagram: A circle with examples labeled as follows:
- example 1
- example 2
- test point
- example 3
- example 4
- example 5

The test point is within the range indicated by the threshold, suggesting similarity to the surrounding examples.
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,

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

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

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

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

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

- \( r_i \): Immediate reward
- \( \gamma \hat{V}_\pi^\epsilon(s_{i+1}) \): Predicted value of next state
- \( \hat{Q}_\pi^\epsilon(s_i, a_i) \): Predicted value of current state

\( t_1 \) and \( t_2 \) define the testing data sequence.
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
Stay to the right of the car; screen the back of the building that is behind the car.

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

A. Boultarias et al. (2015) in IEEE International Conference on Robotics and Automation (ICRA)
Grounding Spatial Relations for Robot Navigation

Stay to the right of the car; screen the back of the building that is behind the car.

Which building? which car??

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


Grounding Spatial Relations for Robot Navigation

Stay to the right of the car; screen the back of the building that is behind the car.

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
Grounding Spatial Relations for Robot Navigation

Results: the robot navigated to the correct goal 88% of the time.
Merci !