Learning to Manipulate Unknown Objects in Clutter by Reinforcement

Abdeslam Boularias and **J. Andrew Bagnell** and **Anthony Stentz**
The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213 USA
{abdeslam, dbagnell, tony}@andrew.cmu.edu

Abstract

We present a fully autonomous robotic system for grasping objects in dense clutter. The objects are unknown and have arbitrary shapes. Therefore, we cannot rely on prior models. Instead, the robot learns online, from scratch, to manipulate the objects by *trial and error*. Grasping objects in clutter is significantly harder than grasping isolated objects, because the robot needs to push and move objects around in order to create sufficient space for the fingers. These *pre-grasping* actions do not have an immediate utility, and may result in unnecessary delays. The utility of a pre-grasping action can be measured only by looking at the complete chain of consecutive actions and effects. This is a sequential decision-making problem that can be cast in the reinforcement learning framework. We solve this problem by learning the stochastic transitions between the observed states, using nonparametric density estimation. The learned transition function is used only for re-calculating the values of the executed actions in the observed states, with different policies. Values of new state-actions are obtained by regressing the values of the executed actions. The state of the system at a given time is a depth (3D) image of the scene. We use *spectral clustering* for detecting the different objects in the image. The performance of our system is assessed on a robot with real-world objects.

1 Introduction

We propose an autonomous robotic system for rubble removal. The robot can execute two types of actions: grasping and pushing. The robot gets a reward for every successful grasp. Pushing actions are used only to free space for future grasping actions. Pushing actions and failed grasping actions are not rewarded. The robot chooses a sequence of actions that maximizes the expected sum of discounted rewards over a given period of time. An depth image of the cluttered scene is the only sensory input available for decision-making.

This seemingly simple task is one of the most challenging problems in autonomous robotics (Amor et al. 2013; Bohg et al. 2013). First, the image needs to be segmented into separate objects in order to predict, at least at some high level, the effects of the pushing actions. Objects that are typically found in rubble, such as rocks and debris, are unknown and have irregular shapes, they are also usually occluded. This makes object detection particularly difficult. Image segmentation is a core problem in computer vision, and the literature dedicated to it is abundant. Our contribution to solving this problem will be an integration of well-known techniques. Namely, we use *k-means* for local clustering of voxels into supervoxels (Papon et al. 2013), *mean-shift* for fast detection of homogeneous regions of supervoxels (Comaniciu and Meer 2002), and *spectral clustering* for regrouping the regions into objects (von Luxburg 2007). This hierarchical approach is justified by the real-time requirements. Low-level clustering reduces the input size of top-level techniques, which are computationally more complex.

Second, the robot needs to learn grasp affordances from the executed actions and their features (Detry et al. 2011). In our case, the grasp affordances are the expected immediate rewards, which can also be interpreted as probabilities of succeeding the grasps. Despite the fact that a large body of work was devoted to manipulation learning, e.g. (Saxena, Driemeyer, and Ng 2008; Boularias, Kroemer, and Peters 2011), this problem remains largely unsolved, and rather poorly understood. Besides, most of these methods either assume that the objects are isolated, or rely on an accurate segmentation for extracting the objects and computing their features. In this work, we take an agnostic approach that does not depend on segmentation for grasping. The features of a grasp are simply all the 3D points of the scene that may collide with the robotic hand during the action. We use kernel density estimation for learning the expected reward function. The only parameter is the kernel bandwidth, which is automatically learned online by cross-validation.

Third, pushing actions have long-term effects. The robot should learn to predict the expected sum of future rewards that result from a pushing action, given a policy. This is an extremely difficult problem because of the unknown mechanical properties of the objects, as well as the complexity of physical simulations. A physics-based simulation was used by Dogar et al. (2012) for predicting the effects of pushing actions, but the authors considered only flat, well-separated, objects on a smooth surface. A nonparametric approach was used by Merili, Veloso, and Akin (2014) for learning the outcome of pushing large objects (furniture). Also, Scholz et al. (2014) used a Markov Decision Process (MDP) for modeling interactions between objects. However, only simulation results were reported in that work.

Copyright © 2015, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

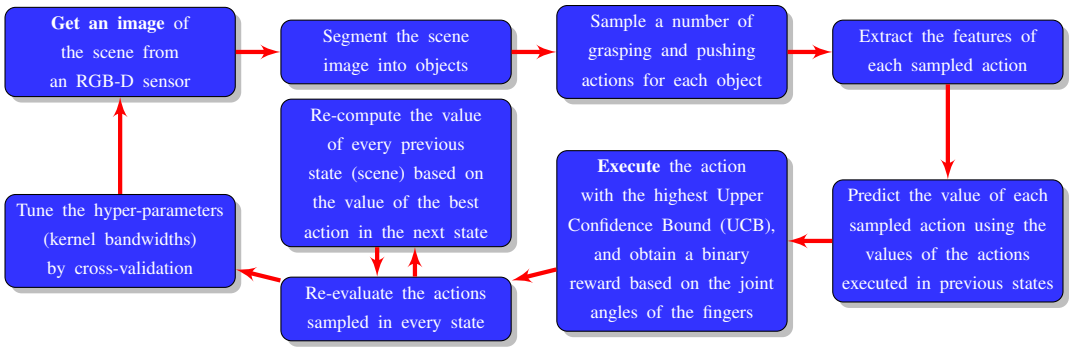


Figure 1: Overview of the integrated system. The inner loop corresponds to policy iteration (evaluation and improvement).

In our application, pushing actions are used to facilitate the grasping of nearby objects. Consequently, features of a pushing action are simply the grasping features of the pushed object's neighbors, in addition to a patch of the depth image in the pushing direction. These features are used in a regression for predicting the expected future rewards of a given pushing action. They are also used for learning a stochastic transition function in the discrete state space that corresponds to the list of collected observations (images).

Last, the robot needs to keep a balance between exploring new actions and exploiting the learned skills. We use the Upper Confidence Bound (UCB) method to solve this problem (Auer, Cesa-Bianchi, and Fischer 2002). We present here a modified UCB algorithm for dealing with continuous states and actions.

The integrated system is demonstrated on the task of clearing a small confined clutter. The robot starts with no prior knowledge of the objects or on how to manipulate them. After a few dozens of trials and errors, and using only depth images, the robot figures out how to push the objects based on their surroundings, and also how to grasp each object. For transparency, unedited videos of all the experiments have been uploaded to http://goo.gl/ze1Sqq .

2 System Overview

Figure 1 shows the work-flow of our autonomous system. First, an RGB-D image of the clutter scene is obtained from a time-of-flight camera. The image, which describes the state, is segmented into objects after removing the background and the support surface (Section 3). Various actions for grasping and pushing each object are sampled. Features of each action in the current state are extracted and inserted in a space-partitioning tree to speed-up range search inquiries (Section 4). The value of every action is predicted by a kernel regression (Section 5). The UCB technique is used for choosing an action (Section 6). A reward of 1 is obtained in case an object was successfully lifted. All the actions sampled in previous scenes are iteratively re-evaluated, by selecting in each scene the action with the highest value (Section 5). Finally, the kernel bandwidths are tuned in a leave-one-sequence-out cross-validation (Section 7).

3 Segmentation

Scene segmentation refers here to the process of dividing an image into spatially contiguous regions corresponding to different objects. The most successful algorithms for segmenting images of unknown objects still require initial seeds of points labeled by a human user (Rother, Kolmogorov, and Blake 2004). This type of methods is unsuitable for autonomous robots. Unknown objects cannot be detected without prior models, examples, or assumptions on their shapes. We make one assumption in this work: the surface of an object is overall convex. Consequently, articulated objects with more complex shapes will be divided into different convex parts. However, over-segmentation affects only pushing actions in certain situations. We enforce the convexity assumption as a soft constraint in the spectral clustering algorithm (Ng, Jordan, and Weiss 2001) by assigning smaller weights to edges that connect concave surfaces.

We start by detecting and removing the support surface using the RANSAC algorithm (Fischler and Bolles 1981). The complexity of spectral clustering is cubic in the number of nodes, thus, it is important to reduce the number of voxels without distorting the shapes of objects. We use the publicly available implementation of the Voxel Cloud Connectivity Segmentation (VCCS) (Papon et al. 2013) to pre-process the image. VCCS clusters the voxels into *supervoxels* with a fast, local, k-means based on depth and color properties. Graphs of supervoxels are shown in Figures 2(b) and 3(b).

The next step consists in extracting *facets*, which are mostly flat contiguous regions, using the Mean-Shift algorithm (Comaniciu and Meer 2002). The mean surface normal of each facet is iteratively estimated by averaging the normal vectors of adjacent supervoxels. At each iteration, the normal vector v_i of a supervoxel i is updated by $v_i = \frac{1}{\eta}\sum_{j\in N(i)} \exp\big(-\alpha cos^{-1}(v_i^t.v_j)\big)v_j$, where $\eta = \sum_{j\in N(i)} \exp\big(-\alpha cos^{-1}(v_i^t.v_j)\big)$, j is a supervoxel neighboring i and v_j is the normal vector of j. After many iterations, adjacent supervoxels that have nearly similar normals are clustered in the same facet. Figures 2(c) and 3(c) show detected facets after applying Mean-Shift (with $\alpha = 10$).

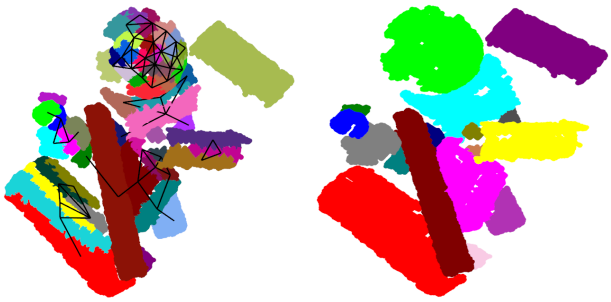
In the final step, facets are regrouped into objects using spectral clustering (Ng, Jordan, and Weiss 2001). The graph of facets is obtained from the graph of supervoxels; a facet is adjacent to another one if it contains a su-

(a) natural objects (rocks) (b) graph of supervoxels

(c) graph of detected facets (d) detected objects

Figure 2: Segmenting a clutter of unknown natural objects

(a) artificial objects (b) graph of supervoxels



(c) graph of detected facets (d) detected objects

Figure 3: Segmenting a clutter of unknown regular objects

pervoxel that is adjacent to another supervoxel in the other facet. The graphs of facets are typically small, as illustrated in Figures 2(c) and 3(c). An edge (i, j) is weighted with $w_{i,j} = \max\{v_i^t.(c_i - c_j), v_j^t.(c_j - c_i), 0\}$, where c_i and c_j are the centers of adjacent facets i and j respectively, v_i and v_j are their respective surface normals. $w_{i,j}$ is nonzero when the shape formed by facets i and j is convex. We compute the eigenvectors of the normalized Laplacian matrix of the graph of facets. We retain only the eigenvectors with eigenvalues lower than a threshold ϵ. Finally, the objects are obtained by clustering the facets according to their coordinates in the retained eigenvectors, using the k-means algorithm. Figures 2(d) and 3(d) show examples of objects detected using this method (with $\epsilon = 0.01$).

4 States, Actions and Features

The state of the objects at a given time is a depth image. We do not consider in this work latent variables, such as occluded parts or mass distributions, as this will increase the decision-making complexity. Actions are always defined within a specific context, or state. In the following, we use the term action to refer to a state-action. We consider two categories of actions, grasping and pushing.

Grasping actions

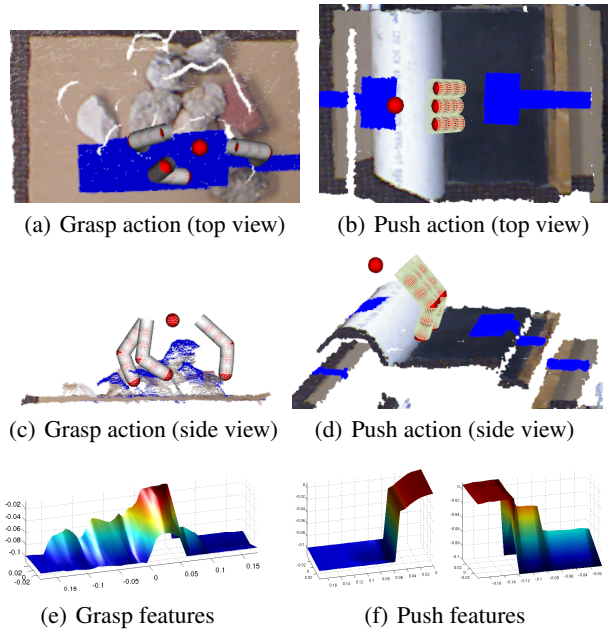
Grasping is performed with two fingers and an opposite thumb (Figure 4(a)). A grasping action is parameterized by: the approaching direction of the hand (the surface normal of the palm), the rotation angle of the wrist, the initial distance between the tips of the two fingers and the thumb (the hand's initial opening), and the contact point, which corresponds to the projection of the palm's center on the surface of objects along the approaching direction. We sample a number of grasping parameters in each image. An image and a sampled parameter vector specify one grasping action. To reduce the grasping space's dimension, we limit the contact points to the centers of objects, obtained from segmentation.

The success probability (or expected reward) of a sampled grasping action is given as a function of its contextual geometric features. These features are obtained by projecting a 3D model of the robotic hand onto the point cloud, and retaining all the points that may collide with the hand when it is fully open (blue strips in Figures 4(a),4(c)). We call this set a *collision surface*. The collision surface is translated into the hand's coordinate system, and discretized as a fixed-size grid. The obtained elevation matrix (Figure 4(e)) is the feature vector used for learning and generalization.

Pushing actions

Pushing actions are defined by the same type of parameters as the grasping actions, except that the three fingers are aligned together (Figures 4(b),4(d)). Instead of closing the fingers, the hand is moved horizontally along the detected support surface, for a fixed short distance in the opposite direction of the fingers. The pushing direction is calculated from the approaching direction and the wrist rotation. A number of pushing actions are sampled in each image.

Pushing features indicate how an object would move, and how the move could affect the grasping of the pushed object or of its neighbors. In principle, a sequence of pushing actions could affect the graspability of objects far from the first pushed one. One should then include features of all the objects in the scene. However, this is unnecessary in clutter clearing where pushing an object in the right direction is often sufficient to free space for grasping an adjacent object.



(a) Grasp action (top view) (b) Push action (top view)

(c) Grasp action (side view) (d) Push action (side view)

(e) Grasp features (f) Push features

Figure 4: Examples of grasping and pushing actions in clutters, and their corresponding contextual features. The features of grasping actions have been presented in our previous work (Boularias, Bagnell, and Stentz 2014).

We use two feature vectors for pushing. The first vector contains features that predict the motion of the pushed object, they correspond to the same part of the surface used in the grasping features, but only the second half of the surface in the pushing direction is considered here (blue strip on the left in Figure 4(b)). The second vector is given by the grasping features of the nearest object behind the pushed one. Here, we consider the second half of the collision surface in the opposite direction of the pushing action (blue strip on the right in Figure 4(b)). If segmentation did not detect any object behind the pushed one, then a default value is used for the second vector of features. The surface between the two objects is ignored, because it would mostly be a free space if the push succeeds. The first vector predicts whether a pushed object would move or not, and how it would move. For instance, the pipe in Figure 4(b) has pushing features that indicate a cylindrical surface and an obstacle-free front, which means that the object would most likely roll, while the box would be blocked by the nearby wood stick, captured in the staircase-like pushing features. The second vector predicts if a push would help grasping a nearby object. In our example, the second vector of features related to pushing the pipe forward indicates a shape (box) that can later be grasped without being obstructed. The two vectors combined are the feature vector of a push action (Figure 4(f)).

5 Learning

We start by formulating the clutter clearing task as a Markov Decision Process (MDP). We define the state at a given time as an RGB-D image of the scene, and denote the state space by $\mathcal{S}$. The action space, denoted by $\mathcal{A}$, contains parameter vectors of grasping and pushing actions. The type of an action $a \in \mathcal{A}$ is given by $\text{type}(a) \in \{\text{grasping}, \text{pushing}\}$. T denotes the stochastic transition function, defined as $T(s, a, s') = p(s_{t+1} = s' | s_t = s, a_t = a)$ wherein s_t and a_t are respectively the state and action at time-step t. $R(s, a) \in \{0, 1\}$ is the reward obtained from executing action a in state s. $R(s, a) = 1$ iff a is a successful grasping action in s. A policy is a function π that returns an action given a state. The state-action value of policy π is defined as $Q_\pi(s, a) = R(s, a) + \gamma \mathbb{E}_{s' \sim T(s,a,.)} V_\pi(s')$, wherein the state value function V_π is defined as $V_\pi(s) = Q_\pi(s, \pi(s))$ and γ is a discount factor (γ is set to 0.5 in our experiments). Given data set $\mathcal{D}_t = \{(s_i, a_i, r_i, s_{i+1}) | i \in [0, t[\}$ of observed states, executed actions, and received rewards up to current time t, we search for a policy with maximum value.

Unlike the pushing actions, grasping is mostly chosen for its immediate reward. Although, the order of the grasps also can be important. To take into account long-term effects of grasping, one should consider additional features, which may slow down the learning process. Therefore, we consider grasping as a terminal action that restarts the episode. After executing a grasp in state s_i, next state s_{i+1} is a fictitious, final, state s_F defined as: $\forall a$, $T(s_F, a, s_F) = 1$ and $T(s_F, a, s) = 0, \forall s \neq s_F$. The terminal state is used simply to mark the end of a sequence of pushes. The robot continues the task without any interruption, the next state following a grasping action is seen as the initial state of a new episode (or task) rather than as being caused by the grasping action.

Learning the transition function is difficult, especially in high-dimensional state spaces, such as images. Nevertheless, this problem can be partially relieved if the states are limited to those in the finite data set $\mathcal{D}_t$. In fact, images of the same clutter are not very different, because several actions are needed before removing all the objects, and each image contains mostly the same objects, in different positions. Based on this observation, we propose to learn a transition function that maps each state s_i in $\mathcal{D}_t$ and action $a \in \mathcal{A}$ to a discrete probability distribution on the states in $\mathcal{D}_t$. Note that a could be an arbitrary hypothesized action, and it is not necessarily a_i, the action that was actually executed at time-step i. The transition function predicts how similar the state at $i + 1$ would be to the state at $j + 1$, had the robot chosen at i an action that is similar to the one that was executed at j, wherein both i and j are past time-steps. These predictions are used for improving past policies, in a kernel-based reinforcement learning algorithm (Ormoneit and Sen 2002).

In the following, we first show how a value function $\hat{V}_\pi$ is generalized to new states and actions. We then show how we compute value functions $\hat{V}_\pi$, and find the best policy π^*.

Given values $\hat{V}_\pi(s_i)$, $i < t$, of policy π, we estimate value $Q_\pi(s, a)$ in new state s and action a using a local regression,

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

K is a kernel that is chosen in our experiments as

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

$\phi(s,a)$ are features of action a in state s (Figure 4). Threshold $\epsilon_{\text{type}(a)}$ depends on the type of a (grasping or pushing), it is set by cross-validation (Section 7). $\hat{Q}_\pi(s,a)$ is set to 0 when $\sum_{i=0}^{t-1} K\big((s_i,a_i),(s,a)\big) = 0$. To speed up range search queries, data set $\mathcal{D}_t$ is saved in a *cover tree* (Beygelzimer, Kakade, and Langford 2006), and updated online.

We consider the set $\mathcal{U}_t = \{(s_i, a_i^k), i \in [0,t[\}$, wherein s_i is the observed state (RGB-D image) at the previous time-step i and $\{a_i^k\}$ are all the actions that were sampled in state s_i, including the executed one. The reward vector R_π of a policy π in $\mathcal{U}_t$ is defined as $R_\pi(s_i) = R(s_i, \pi(s_i))$. Similarly, the transition matrix T_π of policy π is defined as $T_\pi(s_i,s_j) = T(s_i,\pi(s_i),s_j)$ for $i,j \in [0,t[$. The Nadaraya-Watson method provides consistent estimates of R_π and T_π: $\hat{R}_\pi = \text{diag}\big(K_\pi \mathbf{1}\big)^{-1} K_\pi \hat{R}$ and $\hat{T}_\pi = \text{diag}\big(K_\pi \hat{T}\mathbf{1}\big)^{-1} K_\pi \hat{T}$, where $K_\pi(s_i,s_j) = K\big((s_i,\pi(s_i)),(s_j,\pi(s_j))\big)$, $\mathbf{1}$ is a vector of ones, and $\hat{R}(s_i) = r_i$ for $i \in [0,t[$. $\hat{R}(s_i)$ is the reward obtained at time-step i. The sample transition matrix $\hat{T}$ is defined as follows. If a_i was a push action, then $\hat{T}(s_i,s_{i+1}) = 1$ and $\hat{T}(s_i,s_j) = 0$ for $j \neq i+1$. If a_i was a grasp, then $\hat{T}(s_i,s_F) = 1$ and $\hat{T}(s_i,s_j) = 0$ for $j \neq F$.

The state-action space $\mathcal{U}_t$, with the learned models $\hat{R}_\pi$ and $\hat{T}_\pi$, define a finite, discrete, MDP. Therefore, the value function of a policy π defined on $\mathcal{U}_t$ is obtained by simply solving the Bellman equation, i.e. $\hat{V}_\pi = \big(I - \gamma \hat{T}_\pi\big)^{-1} \hat{R}_\pi$, I is the identity matrix. Moreover, it is well-known that standard policy iteration can be used in a finite MDP to find the best policy π^* for the provided model (Bertsekas and Tsitsiklis 1996). Computed value function $\hat{V}_{\pi^*}$ of the best policy π^* is generalized to new states and actions using Equation 1.

6 Exploration versus Exploitation

If the robot always executes the actions that maximize $\hat{Q}_{\pi^*}(s,a)$, then it will learn only about those actions. To find the best policy, each action should be executed sufficiently many times, until a certain confidence on its value is attained. We solve this problem by using the Upper Confidence Bound (UCB) technique (Auer, Cesa-Bianchi, and Fischer 2002), a multi-armed bandit method with a logarithmic regret. To deal with continuous states and actions, we propose a heuristic that uses kernel K for computing a confidence on the value of a state-action. At a time-step t, we sample a number of push and grasp actions, and execute the one that maximizes

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

α is a constant, set to 0.1 in all our experiments. If $\sum_{i=0}^{t-1} K\big((s_i,a_i),(s_t,a)\big) = 0$, then (s_t,a) is considered as completely new. The executed action is randomly selected from the set of the completely new actions, whenever such actions are encountered. This happens, for example, when an object unlike the previous ones appears, when objects appear in novel orientations or contexts, when $\epsilon_{\text{type}(a)}$ drops low, or when the scene is segmented differently.

7 Bandwidth Learning

The two ranges ϵ_{grasp} and ϵ_{push} of the kernel K play a major role in our system. The kernel function defines which state-actions are similar, and is used for both learning and maintaining the exploration-exploitation balance. ϵ_{grasp} and ϵ_{push} are hyper-parameters that cannot be manually tuned or fixed in advance because their optimal values depend on the types of objects, which are unknown to the system, and on the data set $\mathcal{D}_t$, which increases at each time-step.

Bandwidth learning is a typical *model selection* problem (Farahmand and Szepesvári 2011). We propose a solution based on leave-one-sequence-out cross-validation. Given a data set $\mathcal{D}_t = \{(s_i,a_i,r_i,s_{i+1}) | i \in [0,t[\}$ at time t, and a time interval $[t_1,t_2[$ where $0 \leq t_1 < t_2 \leq t$, we estimate the *Bellman error* of a given $\epsilon = (\epsilon_{\text{grasp}}, \epsilon_{\text{push}})$ as

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

where $\hat{V}_\pi^\epsilon$, $\hat{Q}_\pi^\epsilon$ and $\hat{\pi}$ are respectively the state and state-action value functions and the empirical policy (executed actions), obtained by using a kernel with bandwidths $(\epsilon_{\text{grasp}}, \epsilon_{\text{push}})$ and the data in $\mathcal{D}_t$ collected in the intervals $[0,t_1[$ and $[t_1,t[$. $\hat{V}_\pi^\epsilon$ is obtained by solving the Bellman equation using the learned model, while $\hat{Q}_\pi^\epsilon$ is obtained from Equation 1. Finally, the overall average Bellman error is the average of $\hat{BE}(\epsilon)$ in different intervals $[t_1,t_2[$.

At each time-step t, we calculate ξ_{grasp} (resp. ξ_{push}), the Euclidean distance between the most distant grasping (resp. pushing) feature vectors in $\mathcal{D}_t$. We set ϵ_{push} to 0 and search for $\epsilon^*_{\text{grasp}} \in \{\xi_{\text{grasp}}/2^n\}$, $n = 0,\ldots,10$, that has the lowest average Bellman error. The best threshold is further tuned by performing a grid-search in the interval $[\epsilon^*_{\text{grasp}}, 2\epsilon^*_{\text{grasp}}]$. ϵ^*_{push} is obtained using a similar approach, with ϵ_{grasp} set to $\epsilon^*_{\text{grasp}}$.

8 Experiments

We performed extensive tests of the presented system using a WAM robotic arm equipped with a Barrett hand and a time-of-flight camera (Kinect). Figure 5 shows the robot and objects used in the experiments. The CHOMP algorithm (Ratliff et al. 2009) is used to generate arm trajectories, and a library of compliant hand motions with force-feedback is used to execute the grasping and pushing actions (Kazemi et al. 2012). The robot should remove a box and a heavy pipe from the table. However, the objects are too close to each other, and none of them can be grasped unless one of them is pushed away. To make the task even more challenging, the objects are surrounded by fixed obstacles, so that they can move only when pushed in a particular direction. To reduce the learning time, we limit the actions to the box and the pipe by ignoring the actions on detected objects that have a height smaller than a predefined threshold (3cm).

The system has no prior knowledge of any of the objects present on the scene, or on how to grasp or push objects. Based solely on depth images, and readings of the hand joint angles that indicate successes and failures of executed grasps, the robot learned to remove all the objects in less

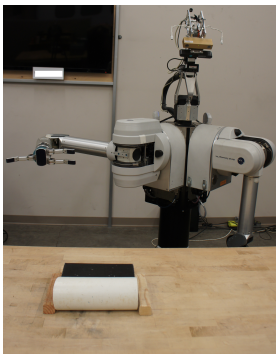


Figure 5: The robotic setup used in the experiments. The robot's task is to remove all the objects.

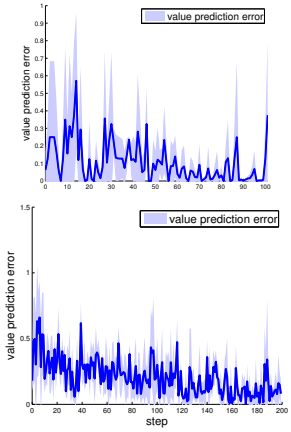


Figure 6: Prediction errors.

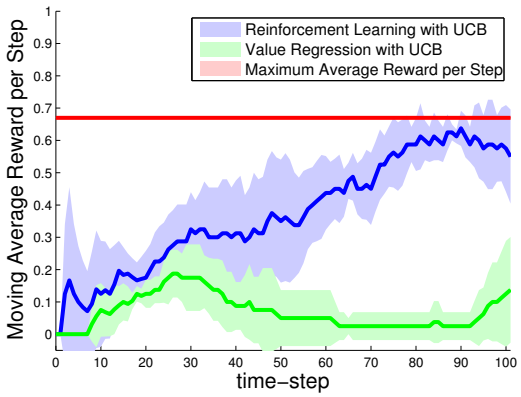


Figure 7: Average reward per time-step and standard deviation in the task illustrated in Figure 5. A reward of 1 is given for each successful grasp. Average at time-step t is computed using the rewards received between $t-20$ and t.

than a hundred time-steps. Figure 7 shows the average reward per time-step as a function of time. The averages are computed from four consecutive repetitions of the same experiment. An important factor that made the learning particularly difficult was the fact that the robot was learning both pushing and grasping at the same time. Often, the robot successfully pushes the objects, but then fails to grasp. Although the robot tries several useless push actions between a useful push and a good grasp, it always manages to discover the relation between these actions. This is done in the policy improvement by realizing that an object could have been immediately grasped in the state that followed the useful push.

Using one CPU, the average time per decision was $0.89s$. This includes perception and learning times, but excludes the time needed to plan trajectories and to execute the action.

Before starting the experiments, we thought that the optimal strategy would be to roll the pipe forward, then to grasp the pipe and the box. But it turned out that this action succeeds with only a small probability. Surprisingly, the robot figured out a different trick for achieving the same result. The learned strategy consists in pushing the pipe forward by pushing the box behind it. This shows an advantage of reinforcement learning over handcrafted techniques.

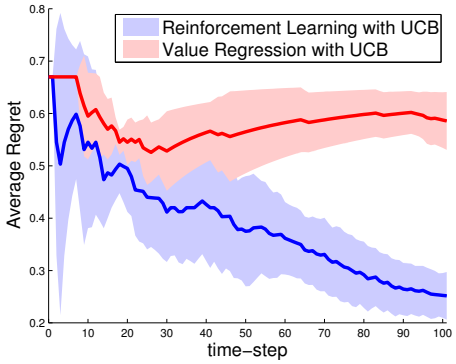


Figure 8: The average regret at time-step t is the number of grasps that the robot could have succeeded so far minus the actual number of successful grasps, divided by t.

We compared the presented approach to a value regression technique that uses directly the empirical values $\hat{V}_{\hat{\pi}}(s_i)$ in Equation 1, instead of learning the transition function and performing the policy evaluation and improvement. The rest of the approach is kept identical. Figure 7 shows that with this method, the performance falls down after an initial period of extensive exploration. We noticed that ϵ^*_{push} drops drastically due to the high variance of empirical $\hat{V}_{\hat{\pi}}(s_i)$ for push actions. Consequently, the robot goes into a long phase of exploring only the pushing actions (Section 7).

We also report prediction errors, which are calculated as the absolute difference between the predicted value of an executed action and its empirical value. The empirical values are the discounted sums of the received rewards in a given trial, while the predicted values are expectations, which explains why the errors are high in certain times. Figure 6 shows that the prediction errors decreases as more actions are executed. The top figure corresponds to the robot experiments. The bottom figure corresponds to a simulation using the pile of rocks in Figure 2(a), where we simulated a number of random grasping actions, and labeled them based on the final reached distance between the palm and the objects.

9 Conclusion

We showed how a robot can learn, by trial and error, to grasp unknown objects in clutter. By reasoning about future states and actions, the robot pushes objects to grasp them more easily. The presented system integrates vision techniques for object detection with reinforcement learning for control and adaptation. To accelerate the online learning process, metric learning should be considered in a future work as an alternative to the plain Euclidean distance used in this work.

Acknowledgment

This work was conducted through participation in the Robotics Consortium sponsored by the U.S Army Research Laboratory under the Collaborative Technology Alliance Program, Cooperative Agreement W911NF-10-2-0016.

References

Amor, H. B.; Saxena, A.; Hudson, N.; and Peters, J., eds. 2013. *Special Issue on Autonomous Grasping and Manipulation*. Springer: Autonomous Robots.

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning* 47(2-3):235–256.

Bertsekas, D. P., and Tsitsiklis, J. N. 1996. *Neuro-Dynamic Programming*. Athena Scientific, 1st edition.

Beygelzimer, A.; Kakade, S.; and Langford, J. 2006. Cover Trees for Nearest Neighbor. In *Proceedings of the 23rd International Conference on Machine Learning*, 97–104.

Bohg, J.; Morales, A.; Asfour, T.; and Kragic, D. 2013. Data-Driven Grasp Synthesis - A Survey. *IEEE Transactions on Robotics* 289–309.

Boularias, A.; Bagnell, J. A.; and Stentz, A. 2014. Efficient Optimization for Autonomous Robotic Manipulation of Natural Objects. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2520–2526.

Boularias, A.; Kroemer, O.; and Peters, J. 2011. Learning Robot Grasping from 3-D Images with Markov Random Fields. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1548–1553.

Comaniciu, D., and Meer, P. 2002. Mean Shift: A Robust Approach toward Feature Space Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24:603–619.

Detry, R.; Kraft, D.; Kroemer, O.; Peters, J.; Krüger, N.; and Piater, J. 2011. Learning Grasp Affordance Densities. *Journal of Behavioral Robotics* 2(1):1–17.

Dogar, M.; Hsiao, K.; Ciocarlie, M.; and Srinivasa, S. 2012. Physics-Based Grasp Planning Through Clutter. In *Robotics: Science and Systems VIII*.

Farahmand, A. M., and Szepesvári, C. 2011. Model Selection in Reinforcement Learning. *Machine Learning* 85(3):299–332.

Fischler, M. A., and Bolles, R. C. 1981. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM* 24(6):381–395.

Kazemi, M.; Valois, J.-S.; Bagnell, J. A. D.; and Pollard, N. 2012. Robust Object Grasping using Force Compliant Motion Primitives. In *Robotics: Science and Systems*, 177–184.

Merili, T.; Veloso, M.; and Akin, H. 2014. Push-manipulation of Complex Passive Mobile Objects Using Experimentally Acquired Motion Models. *Autonomous Robots* 1–13.

Ng, A. Y.; Jordan, M. I.; and Weiss, Y. 2001. On Spectral Clustering: Analysis and an Algorithm. In *Advances in Neural Information Processing Systems*, 849–856.

Ormoneit, D., and Sen, S. 2002. Kernel-Based Reinforcement Learning. *Machine Learning* 49(2-3):161–178.

Papon, J.; Abramov, A.; Schoeler, M.; and Worgotter, F. 2013. Voxel Cloud Connectivity Segmentation - Supervoxels for Point Clouds. In *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR, 2027–2034.

Ratliff, N.; Zucker, M.; Bagnell, J. A. D.; and Srinivasa, S. 2009. CHOMP: Gradient Optimization Techniques for Efficient Motion Planning. In *IEEE International Conference on Robotics and Automation*, 489–494.

Rother, C.; Kolmogorov, V.; and Blake, A. 2004. GrabCut -Interactive Foreground Extraction using Iterated Graph Cuts. *ACM Transactions on Graphics (SIGGRAPH)* 23(3).

Saxena, A.; Driemeyer, J.; and Ng, A. Y. 2008. Robotic Grasping of Novel Objects using Vision. *The International Journal of Robotics Research* 27:157–173.

Scholz, J.; Levihn, M.; Isbell, C. L.; and Wingate, D. 2014. A Physics-Based Model Prior for Object-Oriented MDPs. In *Proceedings of the 31st International Conference on Machine Learning (ICML)*, 1089–1097.

von Luxburg, U. 2007. A Tutorial on Spectral Clustering. *Statistics and Computing* 17(4):395–416.