Learning a Strategy for Whole-Arm Grasping

by

Ariel Anders

B.S. Computer Engineering
University of California, Santa Cruz 2012

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of degree of
Master of Science in Electrical Engineering and Computer Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 2014

Author .....
Department of Electrical Engineering and Computer Science
May 21, 2014

Certified by .....
Leslie Kaelbling
Professor
Thesis Supervisor

Certified by .....
Tomás Lozano-Pérez
Professor
Thesis Supervisor

Accepted by .....
Leslie Kolodziejski
Chairman, Department Committee on Graduate Students

Learning a Strategy for Whole-Arm Grasping

by

Ariel Anders

Submitted to the Department of Electrical Engineering and Computer Science
on May 21, 2014, in Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

Traditionally, robot grasping has been approached in two separate phases: first, finding contact positions that yield optimal grasps and, then, moving the robot hand to these positions. This approach works well when the object's location is known exactly and the robot's control is perfect. However, in the presence of uncertainty, this approach often leads to failure, usually because the robot's gripper contacts the object and causes the object to move away from the grasp. To obtain reliable grasping in the presence of uncertainty, the robot needs to anticipate the possible motions of the object during grasping. Our approach is to compute a policy that specifies the robot's motions over a range of joint states of the object and gripper, taking into account the expected motion of the object when pushed by the gripper. We use methods from continuous-state reinforcement-learning to solve for these policies. We test our approach on the problem of whole-arm grasping for a PR2, where one or both arms, as well as the torso can all serve to create contacts.

Thesis Supervisor: Leslie Kaelbling
Title: Professor

Thesis Supervisor: Tomás Lozano-Pérez
Title: Professor

Acknowledgments

To my family, advisors, and friends.

Contents

1 Introduction **11**

1.1 Problem Motivation 11

1.2 Contributions 12

1.3 Thesis Outline 12

2 Background **13**

2.1 Grasping 13

2.1.1 Force and Form Closure 14

2.1.2 Synthesizing and Executing Grasps 15

2.2 Markov Decision Processes 16

2.3 Model-Free Reinforcement Learning 18

2.3.1 SARSA 19

2.3.2 Linear Gradient-Descent SARSA 22

2.4 Related Work 25

3 Approach **29**

3.1 Grasping Problem Formulation 30

3.2 Grasping Process as an MDP 30

3.3 Reinforcement Learning in Simulation 32

3.3.1 Physics Simulator 32

3.3.2 Learning Algorithms 35

3.4 Execution on Real Robot 36

3.4.1 Execution on Real Robot Overview 36

3.4.2 Execution on Real Robot Implementation Details 37
3.4.3 PR2 Robot 38

4 Experiments **41**
4.1 Simulation 41
4.1.1 State Representation 41
4.1.2 Simulation Experiment 43
4.1.3 Simulation Results 44
4.1.4 Terminating State Locations 45
4.1.5 Parameter's Effect on Performance 49
4.1.6 Simulation Results Overview 55
4.2 Real Robot 56
4.2.1 Dual-Arm Grasping Results 56
4.2.2 Single-Arm Grasping Results 59
4.2.3 Real Robot Results Overview 62

5 Conclusion and Future Work **65**
5.1 Future Work 67

List of Figures

2-1 Frictional Point Contact [15] 14
2-2 Pseudocode for Discrete SARSA [3] 19
2-3 Pseudocode for Discrete SARSA(λ) [3] 22
2-4 Pseudocode for Linear Gradient SARSA [3] 23

3-1 Grasping Problem Formulation 29
3-2 Convex Polygon Construction of the Robot's Shoulder-flex Link . . . 33
3-3 Simulated Robot with a Small Object 33
3-4 Start State for Dual-Arm Grasping with a Large Box 37
3-5 Final Grasp State for Dual-Arm Grasping with a Large Box 37
3-6 Real Robot with a Small Object 38
3-7 Robot in RVIZ with Information about the Object Location from AR Tags 39

4-1 Time Performance of RBF versus Tile Coding 42
4-2 Simulation Results 45
4-3 Dual-Arm Terminating State Results 47
4-4 Single-Arm Terminating State Results 48
4-5 Time Performance vs. Number of Learning Episodes 50
4-6 Dual-Arm Grasps Performance vs. Number of Learning Episodes . . 52
4-7 Single-Arm Grasps Average Reward Performance vs. Number of Learning Episodes 53
4-8 Single-Arm Grasps Percent Successful Performance vs. Number of Learning Episodes 54

4-9 Terminal State for Trial 2 of Dual-Arm Grasping Results Success with Simulator and with PR2 57
4-10 Terminal State for Trial 5 of Dual-Arm Grasping Results 58
4-11 Terminal State for Trial 6 of Dual-Arm Grasping Results Failure with Simulator and with PR2 59
4-12 Terminal State for Trial 1 of Single-Arm Grasping Results Failure in Simulator, but Success with PR2 60
4-13 Terminal State for Trial 2 of Single-Arm Grasping Results Success with Simulator, but Failure with PR2 61
4-14 Terminal State for Trial 3 of Single-Arm Grasping Results Success with Simulator and PR2 61
4-15 Failure State for Trial 7 of Single-Arm Grasping Results Failure in Simulator and with PR2 62

5-1 Barrett Hand [16] 68

List of Tables

4.1 Tile Discretization Types 49
4.2 Dual-Arm Grasping Results 57
4.3 Single-Arm Grasping Results 60

Chapter 1

Introduction

1.1 Problem Motivation

This thesis presents a reinforcement learning approach to whole-arm robotic grasping. The motivation for the project is two-fold. First, we want to view grasping as a process where the movement of the object is exploited, rather than avoided. Second, we want to explore grasping large objects using the arms and torso of the robot.

Traditionally, robot grasping has been approached in two separate phases: first, finding contact positions that yield optimal grasps and, then, moving the robot hand to these positions. This approach works well when the object's location is known exactly and the robot's control is perfect.

However, this approach often fails in the presence of uncertainty, usually because the robot's gripper contacts the object in an incorrect position, which causes the object to move away from the grasp.This thesis investigates an approach that considers the process of bringing the object into a grasp and not just moving to the ideal final contacts.

Second, we want to develop an algorithm for planning grasps on large objects without special features like handles; these grasps should exploit unconventional contact points along the robot's arm and torso. Although there has been extensive research on grasping with hands, research on single and dual whole-arm grasping is quite limited.

1.2 Contributions

We present a framework to model the interactions between the robot and the object to be grasped. This is used to plan robust grasping procedures that incorporate the object dynamics in the grasp process. Our approach departs from conventional grasping procedures by treating the problem of finding contact locations and the grasp approach as a completely integrated process.

This was done by modeling the grasping problem as a Markov decision process and extending existing metrics on a final grasp, such as force closure, to reward functions that provide an objective for a whole policy. We generated policies using reinforcement learning. These policies exploit object dynamics; for example, the robot successfully grasps objects by pushing them into its torso to create contacts for a stable grasp. We developed policies that work for single and dual whole-arm manipulation and tested these policies in simulation and on a real robot. In developing these policies, we investigated different representations for the state space and performed empirical comparisons to show their relative advantages and disadvantages.

1.3 Thesis Outline

The remainder of the thesis is organized as follows. In Chapter 2 we introduce background material on grasping, Markov decision processes, and reinforcement learning and then we discuss related work. In chapter 3 we describe the detailed problem formulation and define the system design in simulation and on a real robot. In Chapter 4 we explain our experimental design and present our results. Finally, we conclude with an outline of future work in Chapter 5.

Chapter 2

Background

In this chapter we review a small amount of the most relevant background material in grasping, Markov decision processes, and reinforcement learning. In Section 2.1.1 we define the metrics for analyzing a grasp and in Section 2.1.2 we discuss traditional robotic grasping methods for synthesizing and executing grasps. Then, in Section 2.2 we introduce Markov decision processes and traditional methods for solving them. In Section 2.3 we introduce reinforcement learning and explain how it can be used to solve Markov decision processes. Finally, we discuss previous work that is most closely related to ours in Section 2.4

2.1 Grasping

In this section, we will discuss robotic grasping. We will discuss how to analyze a grasp in Subsection 2.1.1. This will give an overview of existing metrics for determining the quality of a grasp and how to compute these metrics. Then, in Subsection 2.1.2 we will discuss how these quality metrics are used to implement grasps using a real robot.

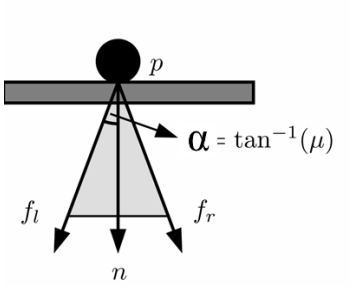


Figure 2-1: Frictional Point Contact [15]

2.1.1 Force and Form Closure

There has been a great deal of research on robotic grasping. In particular, geometric approaches have been developed to analyze the properties of a grasp. In this section we will outline two grasp properties, however, we recommend the survey by Bicchi and Kumar [4] for a thorough introduction to grasping.

Informally, a grasp has the *force closure* property if it can counteract any possible applied external force. A grasp has the *form closure* property if it is capable of preventing any motion of the grasped object. The analysis to determine whether a grasp has the force or form closure property only considers point contacts. A contact exerts a force $\mathbf{f}$ and torque τ on an object. These can be combined into a wrench:

$$\mathbf{w} = [\mathbf{f}, \tau]^T \in \mathbf{R^k}, \tag{2.1}$$

where $k = 3$ for planar systems and $k = 6$ for 3-dimensional systems.

Point contacts are modeled by either a frictionless point contact, frictional point contact, or soft contact. In this thesis, we modeled our contacts as frictional point contacts. Due to Coulomb friction, the force applied by the frictional point contact forms a friction cone about the inward pointing normal $\mathbf{n}$. All forces exerted by the point contact are within an angle α of the normal, where $\alpha = tan^{-1}(\mu)$ and μ is the friction coefficient. An example of a frictional point contact is shown in Figure 2-1.

It is only necessary to compute the wrenches for the outermost forces of the friction cone, $\mathbf{f_l}, \mathbf{f_r}$, for determining if a grasp has the force closure property. This is because the interior forces of the friction cone are included in the positive linear span of $\mathbf{f_l}, \mathbf{f_r}$. Thus the wrenches from the point contacts are $\mathbf{w_l} = [\mathbf{f_l}, \tau_\mathbf{l}]^T$ and $\mathbf{w_r} = [\mathbf{f_r}, \tau_\mathbf{r}]^T$.

A set of wrenches, $\mathbf{W}$, is said to achieve force closure when the positive linear span of $\mathbf{W}$ is equal to R^k, where the positive linear span is computed as:

$$positive\ linear\ span(\mathbf{W}) = \left\{ \sum_{i=1}^{n} \alpha \mathbf{w_i} : \alpha \geq 0 \right\}. \tag{2.2}$$

We can compute whether a grasp has the force closure property from the following theorem.

Theorem 1. A set of vectors $\mathbf{w_i}$ positively spans the entire space $\mathbf{R^k}$ if and only if the origin is in the interior of the convex hull of the set of vectors $\mathbf{w_i}$ [14] .

The second grasp quality metric we review is form closure. A grasp with the form closure property has the capability to prevent any motion of the grasped object. Ferrari and Canny showed that a frictionless force closure grasp is also a form closure grasp [9]. Thus, a simple way to compute if a grasp achieves the form closure metric is to execute the force closure test with the friction coefficient $\alpha = 0$.

2.1.2 Synthesizing and Executing Grasps

Given a criterion, such as force or form closure, the traditional strategy is to search for a set of finger placements on the object that satisfies the criterion, and then to move the robot to those placements. *Grasp synthesis* is the task of finding grasping configurations for the robot and a target object; this has been studied extensively and a variety of methods have been developed from off-line computation with grasping databases to on-line planning with sampling-based approaches. Given a grasping configuration, a motion planner is then used to find a path to move the robot to those contact positions. This, in and of itself, is a hard problem because it is necessary to a path that is feasible in the sense that there is a reachable collision-free inverse-

kinematic solution to move the robot to the grasping pose. A survey of grasp synthesis techniques and algorithms is given by Shimoga [19].

2.2 Markov Decision Processes

In order to plan to take advantage of the dynamics of grasping, we need to model the interaction between the robot and the object. However, we cannot perfectly model the dynamics because we do not know the exact properties of the world, such as the center of mass of the object, coefficient of friction of the object and robot, and many other unknown variables. Instead, we only have estimates of the state properties, such as the location of the robot and object. To account for our inaccurate estimate of the world state and imperfect model of the dynamics, we treat the results of the robot's actions as being stochastic. A Markov decision process is a formal model of this type of process.

An MDP is a framework for modeling decision making in situations where outcomes are stochastic and partly under control of an agent [23]. An MDP is constructed by describing the states, actions, transitions and reward functions of a dynamic system. The solution to an MDP is a *policy* that determines the action the agent should take for every state of the world.

More formally, an MDP is a 4-tuple $\langle S, A, T, R \rangle$, where:

- S is a finite set of *states* of the world.
- A is a finite set of *actions*.
- T is the *state-transition function*: $T(s, a, s')$ describes the probability of ending in state s', given the agent starts in state s and takes action a.
- R is the *reward function*: $R(s, a)$ gives the observed reward for taking action a in state s.

The solution to an MDP is a policy $\pi : S \rightarrow A$, which is a mapping that specifies for each state the action to be taken. An agent following the policy from an initial

start state will observe a sequence of rewards $(r_1, r_2, r_3, ...)$. Rather than evaluating the policy on its sum of observed rewards, it's typical to use the sum of discounted rewards, which means rewards in the future are less important than rewards received immediately. The sum of its discounted, called the return, is computed as:

$$R = r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \ldots, \tag{2.3}$$

where γ is a parameter, $0 \leq \gamma \leq 1$, called the *discount rate*.

The goal is to find a policy that maximizes the expected discounted sum of rewards over some horizon of interest. In our case, we focus on an infinite horizon with discounted rewards. For each policy, π, we can define a value function, $V_\pi(s)$, which is the expected discounted sum of the rewards earned by following policy π from state s. Similarly, we can define a Q-value function, $Q_\pi(s, a)$, which is the expected discounted sum of rewards for starting in s, taking action a, and then continuing to act using policy π.

A policy is optimal for state s if its value at that state is greater than or equal to the value for any other policy at that state. For an MDP, it has been shown that there exists at least one policy that is optimal for all states. Optimal policies, π^*, and corresponding value functions, V^*, can be computed using dynamic programming techniques called value iteration and policy iteration.

Value Iteration

Value iteration is a method to solve for an optimal policy, π^*, and corresponding value function, V^*. Given an arbitrary initial value function V_0, we can compute a sequence of value functions V_i by using the following update:

$$V_{i+1}(s) = max_a \left\{ R(s, a) + \gamma \sum_{s'} T(s, a, s') V_i(s') \right\}. \tag{2.4}$$

V_i converges to the optimal value function V^*. Then, the optimal policy π^* can be derived by V^* :

$$\pi^*(s) = argmax_a \; V^*(s'|s,a). \tag{2.5}$$

In order to achieve the optimal value function V^*, there needs to be an infinite number of iterations; however, in practice, value iteration is set to terminate when the value function changes by only a small amount. It also usually happens that policy π derived from V_i is optimal long before V_i is close to V^* [3].

Each iteration of value iteration can be performed in $O(|A||S|^2)$ steps [6]. Because of this, value iteration can quickly become intractable with large state and action spaces.

2.3 Model-Free Reinforcement Learning

Reinforcement learning is an area of machine learning in which the goal is to learn a policy to maximize a numerical reward signal [3]. The learner is not given explicit actions to take, but must discover them on its own. One characteristic of many reinforcement learning problems is that actions may not yield immediate reward, but may still be important for a delayed reward. Furthermore, in reinforcement learning, the learner must balance between taking exploration actions to explore the space and exploitation actions to accumulate reward from actions it has already found to be effective.

Reinforcement learning can be used to find a solution of an MDP and other dynamic systems where the transition and reward functions are not explicitly given ahead of time. In our case, we are interested in reinforcement learning because we are using a continuous space MDP and cannot use traditional methods for solving MDPs.

Model-free reinforcement learning has been applied in robotics for solving continuous MDPs in which the goal is to learn an approximate value or Q-value function. This differs from model-based reinforcement learning, which attempts to learn transition and reward models and then use value iteration on those models to determine

a policy. The work in this thesis is model-free and uses a simulator to describe the transition function. The following sections briefly describe the algorithms used in this thesis. Barto and Sutton give a thorough analysis of these reinforcement learning algorithms [3].

2.3.1 SARSA

SARSA (State-Action-Reward-State-Action) is a reinforcement learning algorithm used for learning the Q-value function of an MDP. SARSA is particularly appropriate when combined with function approximation (which we will discuss later) and/or when the domain is not strictly Markov. SARSA is an 'on-policy' learning algorithm: an agent will interact with the environment and update its Q-value function based on the actions it takes and the reward it receives.

Discrete one-step SARSA Algorithm

Initialize $Q(s, a)$ arbitrarily

Repeat for each episode:

Initialize s

$a = \pi(s)$, where π is derived from Q

Repeat for each step of episode:

Take action a, observe r, s'

$a' = \pi(s')$

$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$ // update rule

$s \leftarrow s'; a \leftarrow a'$

Until s is terminal

Figure 2-2: Pseudocode for Discrete SARSA [3]

Figure 2-2 shows the the algorithm for a simple discrete one-step SARSA. First, the Q-value function is initialized arbitrarily. Then, a number of learning episodes are run. During each learning episode, the agent starts from an initial start state

and then takes an action based on the policy derived from the Q-value function. It observes a reward and then updates the Q-value function by an amount adjusted by the learning rate α. The update rule is shown in Figure 2-2.

In this subsection, we will discuss the learning rate parameter α and how the policy is derived from the Q-value function for the SARSA algorithm. Next, we will describe how to extend the discrete one-step SARSA to include eligibility traces to update more states in the update rule. Finally, we will explain how SARSA can be used with continuous spaces using function approximation.

Adaptive Step Size

The step size parameter, α, for the gradient descent update can directly effect the performance of SARSA. If α is too large, the policy may never converge on an effective policy; alternatively, if α is too small, the the algorithm converges too slowly. To counteract these effects, we applied an adaptive step size technique developed by Dabney and Barto [7].

Exploration versus Exploitation in SARSA

The policy derived from the learned Q-value function is deterministic, in which for every state there is only one action returned from $\pi(s)$. This is found by using a *greedy policy:*

$$\pi(s) = argmax_a \; Q(s, a) \tag{2.6}$$

During the on-line learning process, the policy derived from the Q-value function is not deterministic. In SARSA, there is a trade-off between exploration and exploitation. When selecting the next action, should the agent take actions that provided good rewards in the past or should it take random actions that could eventually return even better reward? Our implementation takes both exploration and

exploitation actions by using an *ϵ-greedy policy*:

$$\pi(s) = \begin{cases} \textit{random action} & \textit{with probability } \epsilon \\ \textit{argmax}_a\ Q(s,a) & \textit{with probability } 1-\epsilon \end{cases} \tag{2.7}$$

We use a converging *ϵ-greedy policy* by setting $\epsilon = \frac{1}{t}$. By using a converging *ϵ-greedy policy*, the agent starts exploring the space by taking random actions and eventually takes more exploitation actions as $p(1 - \frac{1}{t}) \to 0$.

Convergence Properties

SARSA converges to the optimal Q-value function with probability 1 under the following assumptions: all the state-action pairs are visited an infinite number of times and the policy converges in the limit to the greedy policy (this can be done by using the converging *ϵ-greedy policy* described above) [3].

Discrete SARSA(λ)

Discrete one-step SARSA credits only the last action in a sequence of actions that lead to a terminating state; however, it can be modified to credit many actions of the sequence with the use of *eligibility traces*. By using eligibility traces we can greatly improve the efficiency of learning a Q-value function.

An eligibility trace records the previous state and action pairs visited in a sequence of actions. The update step of the discrete-SARSA algorithm is modified to give some of the eligible state-action pairs credit or blame for the error. The degree previous state-action pairs are updated is associated with a parameter $\lambda \in [0, 1]$. If $\lambda = 1$, then the credit given falls off by γ per step; conversely, if $\lambda = 0$ then SARSA(λ) is equivalent to one-step SARSA. Figure 2-3 shows the algorithm for SARSA(λ), the modified version of discrete one-step SARSA that uses eligibility traces.

Discrete-SARSA(λ)

Initialize $Q(s, a)$ arbitrarily and $e(s, a) = 0$ for all s, a

Repeat for each episode:

Initialize s

$a = \pi(s)$, where π is derived from Q

Repeat for each step of episode:

Take action a, observe r, s'

$a' = \pi(s')$

$\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$

$e(s, a) \leftarrow e(s, a) + 1$

for all s, a:

$Q(s, a) \leftarrow Q(s, a) + \alpha\delta e(s, a)$

$e(s, a) \leftarrow \gamma\lambda e(s, a)$

$s \leftarrow s'; a \leftarrow a'$

Until s is terminal

Figure 2-3: Pseudocode for Discrete SARSA(λ) [3]

2.3.2 Linear Gradient-Descent SARSA

Although SARSA can be used in a discrete state space (where the Q-value function is represented with a table), an adaption called Linear Gradient SARSA allows it to work for continuous spaces using features of the state space and function approximation.

Function approximation is a technique to represent a function when it is computationally intractable to represent exactly (e.g.: in tabular form) [13]. We're interested in *parametric* function approximation methods which use a finite set of arguments. In our case, the finite set of arguments are features of the state space, which are functions $\phi_i(s)$ that compute some useful property of a state. Then, we can represent

the value of a state as a linear combination of the features:

$$Q(s,a) = \theta^{\mathbf{T}}\phi_{\mathbf{s}} = \sum_{i=1}^{n} \theta(i)\phi_s(i), \tag{2.8}$$

where $\phi_{\mathbf{s}}$ is a column vector containing the features of a state s and θ is a column vector of the same length.

θ contains the weights that are adjusted while implementing Linear Gradient-Descent SARSA to make $Q(s,a)$ as close to $Q^{\pi}(s,a)$. Figure 2-4 shows the pseudocode for Linear Gradient-Descent SARSA.

Linear Gradient-Descent SARSA(λ)

Initialize $Q(s,a)$ arbitrarily and $e(s,a) = 0$ for all s, a

Repeat for each episode:

Initialize s

$a = \pi(s)$, where π is derived from Q

Repeat for each step of episode:

Take action a, observe r, s'

$a' = \pi(s')$

$\delta = r + \gamma Q(s', a') - Q(s,a)$

$\mathbf{e_t} = \gamma\lambda\mathbf{e_t} + \phi_{\mathbf{s}} Q(s,a)$

$\theta \leftarrow \theta + \alpha\delta_t \mathbf{e_t}$

$s \leftarrow s'; a \leftarrow a'$

Until s is terminal

Figure 2-4: Pseudocode for Linear Gradient SARSA [3]

State-Space Features

In order to use the Gradient-Descent SARSA the features of the state space must be represented in a single column vector $\phi_{\mathbf{s}}$, which is shorthand notation for the following

column vector:

$$\phi_{\mathbf{s}} = \phi(s) = [\phi_1(s), \phi_2(s), \phi_3(s) \dots \phi_n(s)]^T. \tag{2.9}$$

The features are functions that compute some useful property of a state. Typically, features are computed by applying basis functions on some numerical properties of a state. In this research, we investigated two types of basis functions: tile coding, and radial basis functions.

Tile coding exhaustively partitions the input. We used a grid-based tiling in which the input space was partitioned into uniform tiles. Every state will lie in one particular tile and each tile corresponds to one element of the feature vector $\phi_{\mathbf{s}}$; hence, tile coding is binary (0-1) valued. Sutton and Barto do a more thorough analysis of tile codings in [3].

Radial Basis Functions generalize tile coding into continuous-valued features. We can center an RBF feature i at some state c_i of the state space. Then, for any particular state we can compute the value for that feature:

$$\phi_s(i) = exp\left(-\frac{||s - c_i||^2}{2\sigma_i^2}\right). \tag{2.10}$$

Now, each feature outputs a value in the interval [0,1].

There are trade-offs between using tile codings versus radial basis functions. The value for each tile coding feature can be computed in constant time and only requires one bit of memory because it is binary valued. However, tile codings may require a larger number of features to appropriately approximate the state space. Additionally, since tile coding features are binary, they can form discontinuous, non-differentiable function approximations.

Alternatively, using RBF features to approximate the value function can produce a smooth and differentiable function approximation. Since RBF features have more descriptive power, there might not need to be as many features as in tile coding. However, computing the value for each feature requires evaluating an exponential function and requires more memory to store.

2.4 Related Work

There is a lot of literature in grasping using a small number of point contacts (typically fingers of a robot hand) some of which we talked about in Section 2.1.1. In this section, we will focus on literature more relevant to whole-arm grasping and enveloping grasps. An enveloping grasp is formed by wrapping the fingers[arms] and the palm[torso] around an object, this is in contrast to a pinching grasp, where the object is restrained by the fingertips at certain contact points [12].

First, we discuss an approach for planar enveloping grasps. Then, we investigate more recent literature in whole-arm grasping. And finally, we discuss an approach for grasping that uses reinforcement learning.

Enveloping Grasps

In 1988, Trinkle et al. [22] developed a planner and simulator for two-dimensional frictionless, enveloping grasps. In this case, the robot hand was composed of a palm and two hinged fingers and the object was a rigid body convex polygon.
Grasping was divided into three phases: pre-liftoff, lifting, and grasp-adjustment. In the pre-lift off phase the hand is moved directly over the object such that the center of the gripper is in line with the center of the object. During the lifting stage, the hand deliberately moves the object to direct it towards the palm. Once a force-closure grasp is achieved the grasp-adjustment phase begins. This simulates different movements of the fingers to find position increment/decrement actions that will lead to a form closure grasp. This work is related to our approach because the planner intentionally pushes the object into a grasp.

Whole-Arm Grasping Approaches

Hsaio and Lozano-Pérez [11] developed a system for enveloping whole-arm grasps and tested it in simulation. Their approach used a human to demonstrate grasping by using a teleoperated simulated robot. The demonstrations were completed on a

set of template grasps with objects modeled by three primitives: boxes, cylinders, and spheres. To grasp a new target object, the system selected the template grasp of a template object that was most similar to the target object. They applied a transformation to map the contact points from the template object to the target object. A collision-free trajectory was found with the mapped contacts to create a grasp trajectory. The grasp trajectories were then executed in simulation.

There has been some previous work that, like ours, focusses on whole-arm grasps. Seo et al. [18] developed an approach for planar whole-arm grasping. This work used the traditional grasping approach of separating the problem into separate phases: first, finding contact positions that yield optimal grasps, and, then, moving the robot to these positions. The algorithm developed by Seo et al. could handle complex non-convex shapes.

reinforcement learning and Grasping

Stulp et al. [21] demonstrated a model-free reinforcement learning approach to grasping under uncertainty and verified it in simulation. This approach uses a popular policy search algorithm called Policy Improvement with Path Integrals (PI^2). PI^2 takes an initial policy encoded as a Dynamic Motion Primitive (DMP) and optimizes it using reinforcement learning. The initial policies were generated using the open-source planner GraspIt!.

GraspIt! generates a pre-shape pose and grasp pose from an initial robot configuration and object. Then, a minimum-jerk trajectory is formed that moves the end effector from the initial pose to the pre-shape pose and then to the final grasp pose. This trajectory is encoded as a DMP and optimized using PI^2 with the constraint that the final grasp pose is kept constant. In this method, the robot learned to push the object forward during the pre-shape pose before moving to the grasp pose. This made farther objects easier to grasp and pushed closer objects into the hand. They also achieved positive results when introducing noise into the simulator. This result is similar to ours in which the robot learns to exploit object dynamics to achieve a

better a grasp. Although both approaches use reinforcement learning, they are not interchangeable. Their approach still treats the grasping problem in two separate phases: finding optimal grasp contacts and then moving the robot to the optimal contacts. Alternatively, our approach never specifies the contact configurations, only the criteria for an end grasp configuration.

Chapter 3

Approach

In this chapter, we discuss our approach for whole-arm robotic grasping using reinforcement learning. In Section 3.1 we discuss the grasping problem formulation and in Section 3.2 we formalize the grasping problem using an MDP. Then, in section 3.3 we discuss how we use reinforcement learning for whole-arm grasping in simulation. Finally, in Section 3.4 we discuss the execution of whole-arm grasping on a real robot. The robot used in our experiments is the PR2 Robot developed by Willow Garage [10].

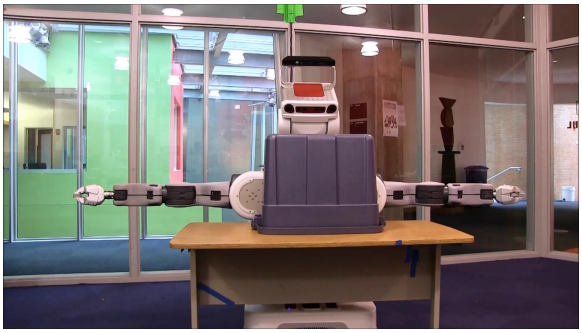


Figure 3-1: Grasping Problem Formulation

3.1 Grasping Problem Formulation

Our problem domain consists of a PR2, which has two arms, and a target object to be grasped. The PR2 has the target object in front of it and can sense its location. The objective is to grasp the object stably enough to lift it off the table by lifting the torso. Possible failure modes are knocking the object out of reach or not grasping stably so the object slips out when the torso lifts. Figure 3-1 illustrates our problem setting.

For simplicity, we are modeling the problem as planar, however, this framework could be extended for three dimensions. We approximate the problem using a cross-section through the arms of the robot and allow the PR2 to only move its arms in the plane. There are single and dual-arm versions of the problem; in the dual-arm case, to reduce dimensionality of the problem for now, we assume the arm movements are "mirrored".

The object to be grasped is assumed to rest stably on a table in front on the robot and to have a constant cross-section at the height of the arms. The object has (x, y, θ) degrees of freedom on the surface and has varying starting locations. The object is a simple circle or convex polygon object defined by its edges relative to its origin; however this can be modified to include more complex objects.

3.2 Grasping Process as an MDP

The overall approach begins with modeling this problem as a continuous state MDP. The state space is that of the object position and arm configuration and the actions are incremental motions of the arm. The transition function is implemented using a simulator or a real robot and the reward function is based on the force closure analysis of a grasp.

The following defines the 4-tuple $\langle S, A, T, R \rangle$ for our MDP model:

States S

- The underlying state space is described by six joint angles: $\theta_{\mathbf{arm,i}}$, three for each arm and the location of the object (x_o, y_o, θ_o), where θ_o is the rotation of the object about the z-axis.

  The state space is represented using tile coding or RBF functions as described in Section 2.3

- A state is a *final state*, $s \in S_f$, if it is a *goal* state or a $failure$ state. A *goal* state is a state such that the contacts by the links and torso on the object create a force-closure grasp. A $failure$ state occurs if the distance of the object relative to the origin is greater than the sum of the link lengths of one of the robot's arms. A $failure$ state can also occur if the opposing arms of the robot contact each other; this is to ensure the robot does not unnecessarily collide into itself and cause any damage.

Actions A

At every time step, each link can increase or decrease its joint angle by a discrete value $\Delta\theta$ or do nothing.

Transitions T

The transition function is handled by the simulator or robot. Given an action a from the set of actions A, the simulator or robot will execute the action and return the resultant state.

Reward Function R

Final states can receive two types of rewards: *goal* states receive a reward of 1.0 and *failure* states receive a reward of -100.0. Non-terminating states receive a score of -0.5 for one contact and -0.25 for two or more contacts. All other non-terminating states receive a reward of -1.0. Note that only final states receive

a reward greater than 0.0 or less than -1.0.

$$R_f(s) = \begin{cases} 10.0 & force\ closure \\ -0.25 & two\ or\ more\ contacts \\ -0.50 & one\ contact \\ -1 & nonterminating\ with\ no\ contacts \\ -100 & failure\ state \end{cases} \tag{3.1}$$

Modifying the reward function can greatly affect the performance of the learned policy. We went through different iterations of reward functions and found this reward function to be the most successful. First, the failure states have a particularly large negative punishment. This motivates the learned policy to avoid failure states. Second, each non-terminating state has negative reward. This encourages the solved policy to converge quickly. Lastly, we have reward shaping in terms of contacts: non-terminating states with contact locations have less penalty than that of non-terminating states without contacts.

The solution to an MDP is a policy π that maps states to actions. To find a solution for our MDP we use reinforcement learning to find a Q-Value function as described in Section 2.3.

3.3 Reinforcement Learning in Simulation

We implemented the reinforcement learning algorithm, Linear Gradient SARSA, in simulation because it requires a high number of training examples. Because of this, it is unrealistic to execute the learning on a real robot. Thus, we used Box2D to simulate our domain.

3.3.1 Physics Simulator

Box2D [5] is an open source 2D Physics Engine for simulation of the motion of rigid bodies. Some of its features include collision detection, top-down friction, and revolute

joints with joint limits, motors, and friction. We are using the pybox2D software which is a python wrapper for Box2D [2].

Each link of the robot is described by a set of convex polygons. These convex polygons were created with the open source software Physics Body Editor [17]. Figure 3-2 shows the convex polygons that make up the first link of the robot's right arm. The objects used in our software consist of simple convex polygons or circle objects described by their edges and vertices or radii, respectively. However; the method for importing the robot could be used for importing more interesting objects for grasping.

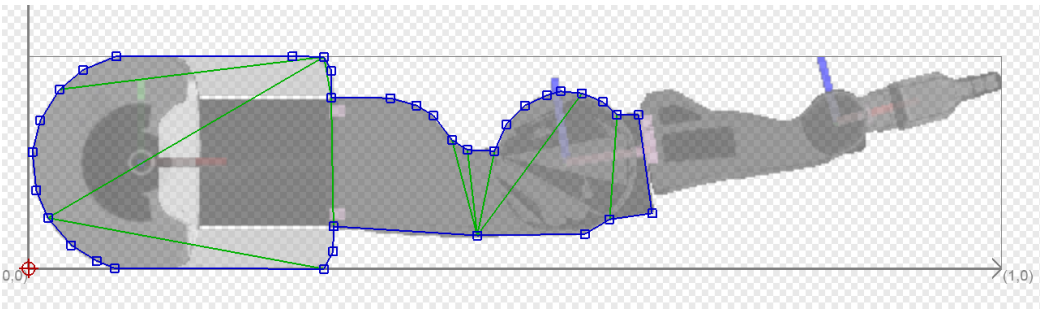


Figure 3-2: Convex Polygon Construction of the Robot's Shoulder-flex Link

The Box2D simulator allows the input of custom coefficients of friction. The friction coefficient between the table and object was 0.5 and the friction coefficient between the object and the robot was 0.3.

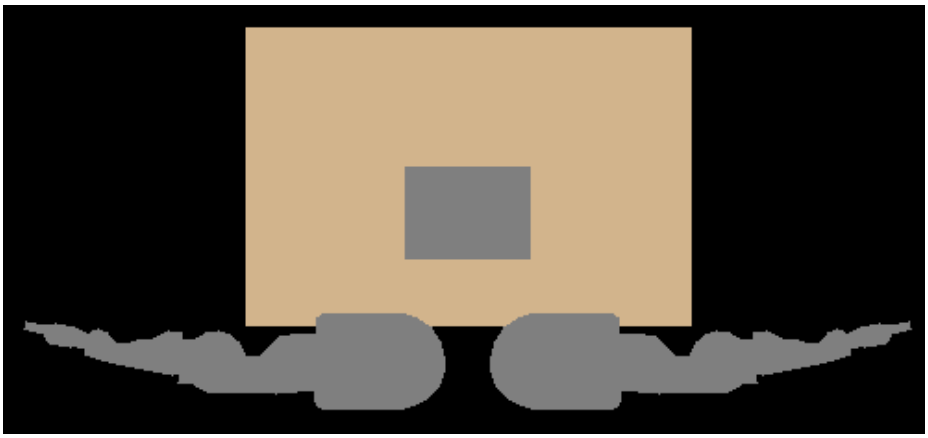


Figure 3-3: Simulated Robot with a Small Object

The following defines the 4-tuple $\langle S, A, T, R \rangle$ implemented by the Box2D simulator for the MDP:

States S

The state space is described by the joint angles and the location of the object (x_o, y_o, θ_o). Each of these can be retrieved using the native API functions for Box2D.

Actions A

At every time step, each link can increase or decrease its joint angle by a discrete value of 0.1 radians or do nothing.

Transitions T

The transition function is handled by commanding an action to the simulator. The simulator adds the discrete action offset to the current joint angles to obtain the *desired joint angles*. The simulated robot moves to the desired joint angles by using a custom PD controller we built for the simulated robot. Once each joint of the robot is within $\pm$ 0.033 radians of the desired joint angles the action terminates; naturally, this leads to some stochastic behavior.

Reward Function R

Goal state We can retrieve the object contact information using the native API functions for Box2D. These contacts include contact force and position information, which is used to compute the frictional point contact wrenches discussed in Section 2.1.1. Then, we can use ConvexHull, a method from SciPy [8] to compute the convex hull of our contact position wrenches. The simulator computes a force closure test by determining if the origin is in the interior of the convex hull. If the object is in force closure, then the simulator returns a reward of 0.

Failure state If the location of the object is further away than length of the robot's arm, then the grasp is deemed a failure and the simulator returns

a reward of -100. Additionally, we can retrieve contact information of the robot links to determine if there is a self collision between both arms; if this is the case then the simulator returns a reward of -100 as well.

Nonterminating state with object contacts If the object is not in force closure, but has two or more contact locations or one contact location, then the simulator returns a reward of -0.25 or -0.5, respectively.

Nonterminating state Lastly, if there are no contacts and the object is not too far away then the simulator returns the typical non-terminating state reward state of -1.

Figure 3-3 is a picture of the simulated robot with a small box as the target object. The robot is currently in the initial start state configuration in which the joint angles are all zero. The object is a large box placed at (0,0.4) where the coordinate frame is oriented between the two arms.

3.3.2 Learning Algorithms

We used Linear Gradient-Descent SARSA to learn a Q-Value function for our MDP. We tested our implementation with the state space represented with tile coding and radial basis functions, where the state values were the joint angles (3 or 6 values) and object location (3 values). In the single-arm manipulation trials we only modeled active arm's joint angles leading to a total of six state parameters; however, in the dual-arm case we had nine.

The learning algorithm is executed using a set of initial starting positions for the object location. In our experiments we tested using a single initial position versus using multiple locations. Additionally, we tested the performance of the learning algorithm using testing locations that differed from the initial starting positions.

There are two main parameters in learning: grid discretization size and number of learning episodes. In our experiments, we tested using different grid discretization amounts and number of episodes. We tested with the number of episodes ranging from one to ten. For the dual-arm case, the number of tiles ranged from 500 to

1953125; and for the single-arm case the number of tiles ranged from 64 to 531441. The results from these experiments can be found in Chapter 4.

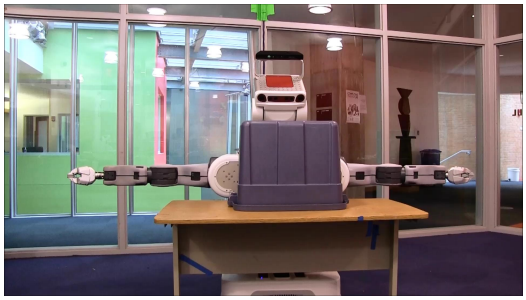
3.4 Execution on Real Robot

First, we will give an overview on how to execute whole-arm grasping using a robot. Then, we provide particular details on how we implemented the actions on the PR2.

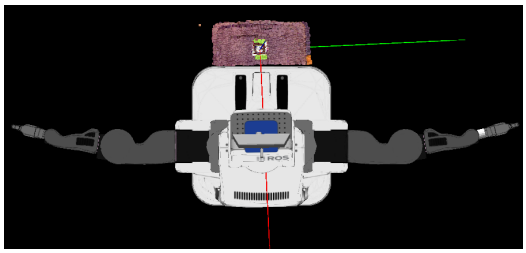
3.4.1 Execution on Real Robot Overview

To run this approach on a real robot, the implementation uses both the simulator and the actual robot. Figure 3-4 shows an overview of the start state for dual-arm grasping of a large box. Figure 3-4a shows an image of the actual problem setup with the PR2 robot and a grasping object. Prior to execution, the object is measured and imported into the Box2D simulator. Then, the robot obtains the object's location using AR tags. Figure 3-4b shows the position of the object visualized in RVIZ, a robot visualization tool available from the Robot Operating System (ROS) [1]. The location is then imported into the simulator, which is shown by Figure 3-4c.

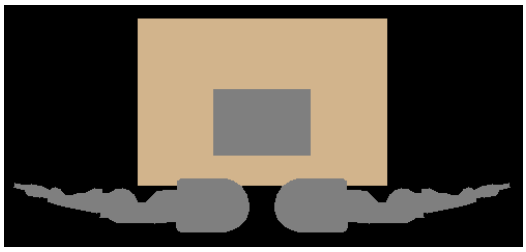
The reinforcement learning algorithm is executed using the simulator with the imported object location. Once a policy has been found, the execution is completed on the robot. First, the robot finds the object location using the AR Tags. The object location and the robot's current joint states are imported into the simulator. The simulator is used to compute the features of the state and determine the next action from the Q-value function as described in Section 2.3. The robot executes this action and repeats this process until the simulator detects a terminating state (failure or goal state). Figure 3-5 shows the final goal state of a dual-arm grasping procedure.



(a) Actual Start State

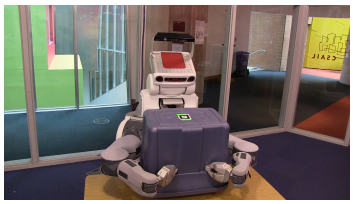


(b) RVIZ Representation

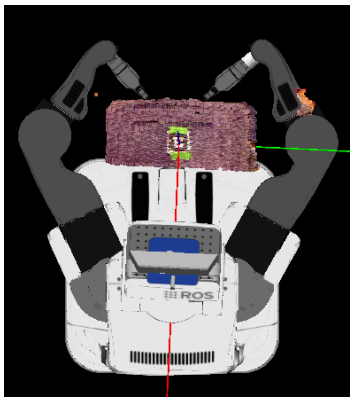


(c) 2D Simulator Representation

Figure 3-4: Start State for Dual-Arm Grasping with a Large Box



(a) Actual Grasp State



(b) RVIZ Representation

(c) 2D Sim. Representation

Figure 3-5: Final Grasp State for Dual-Arm Grasping with a Large Box

3.4.2 Execution on Real Robot Implementation Details

The PR2 has two arms, each with seven degrees of freedom. In our experiments, we fix four of the arm joints, leaving three revolute joints per arm: shoulder pan, elbow flex, and wrist flex. Additionally, we move the torso height so that the arms are placed approximately at the middle height of the objects used in our experiments.

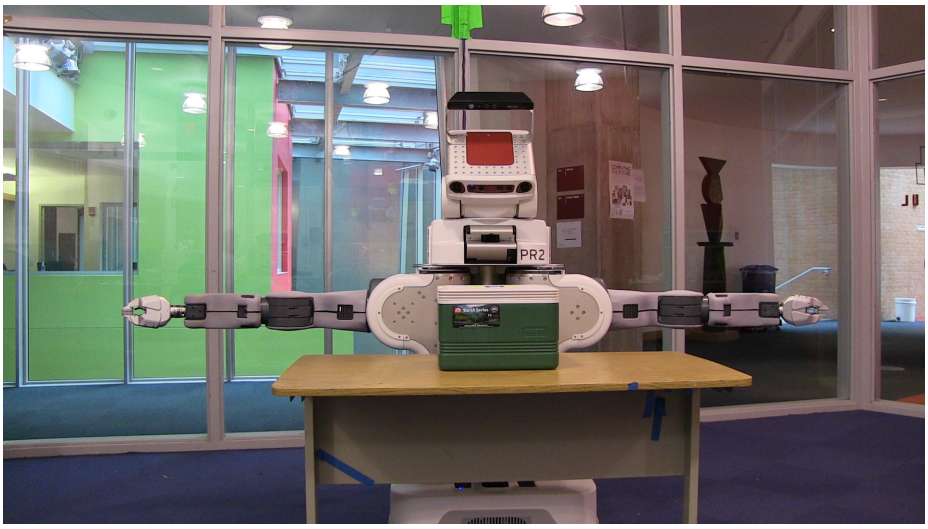


Figure 3-6: Real Robot with a Small Object

Figure 3-6 shows a picture of the PR2 robot with a small box, the target object; this figure shows the actual world state represented in the simulator shown in Figure 3-3.

3.4.3 PR2 Robot

In order to execute a policy we need to use the robot within the same framework of the MDP. The following defines how the 4-tuple $\langle S, A, T, R \rangle$ of the MDP is instantiated on the robot:

States S

Joint Angles The joint angles of the robot are published in real time on the ROS /l_arm_controller/state and /r_arm_controller/state topics. When the *get_state*() function is called, the most recently published joint angles are returned.

Representing and Locating the Object The objects used in our experiments are simple convex polygons or circle objects. The size of the object is measured and imported into our Box2D simulator. Each object has an

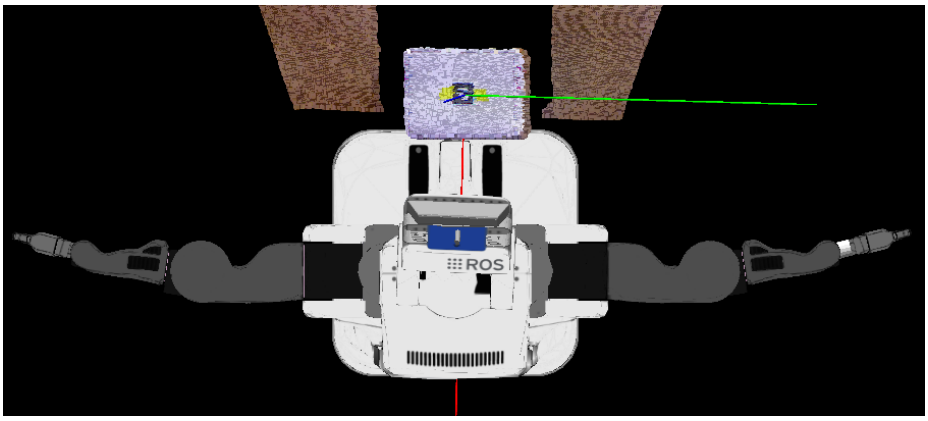


Figure 3-7: Robot in RVIZ with Information about the Object Location from AR Tags

AR tag placed at its center. This is used with the ROS ar_pose_marker package to locate the object with respect to the Torso link (the coordinate frame for the Box2D simulator). Figure 3-7 shows the data received from the AR tags in RVIZ, a robot visualization tool available from ROS.

Actions A

The ROS JointTrajectoryAction package allows us to command joint-positions for the PR2 robot. We are using this package to hold four of the joints fixed and send the joint angles that correspond to our 3-degree of freedom model. For each joint an action consists of three possible values (increment, decrement, or do nothing) by a fixed step size of 0.1 radians. The action is executed using the JointTrajectoryAction package and takes approximately 1 second to complete.

Transitions T

The transition function is handled by having the robot execute the corresponding action and returns its state after the action is complete. The execution of the action on the real robot is stochastic and the effect of the action differs from the 2D model.

Reward Function R

The reward depends on the position of contact locations between the object

and the robot. Currently, we are using the Box2D simulator to compute where the contact locations are by importing the current joint angles and the object location from the AR tags. Then the reward is simply the reward computed from the simulator.

Chapter 4

Experiments

In this chapter we discuss our experiments and their results. In Section 4.1 we discuss experiments we conducted in simulation to analyze the grasping capability, behavior, and performance of the policies we derived using reinforcement learning. Then, in Section 4.2 we explore preliminary results of our approach executed on a real robot.

4.1 Simulation

This section describes our simulator experiments and results. More specifically, in Subsection 4.1.1 we look at the time performance of representing the state with a radial basis function versus tile coding. In Subsection 4.1.2 we discuss the experiments we conducted in simulation to analyze the grasping capability and grasping behavior of the policies we derived using reinforcement learning. In Subsection 4.1.5 we explore the effect of varying learning parameters on performance.

4.1.1 State Representation

In Section 2.3 we described different basis functions to represent the state space. The two functions we investigated were radial basis functions (RBF) and tile coding. We decided to choose one basis function to represent the state space in our experiments. We made this decision based on their time performance. To test the time performance,

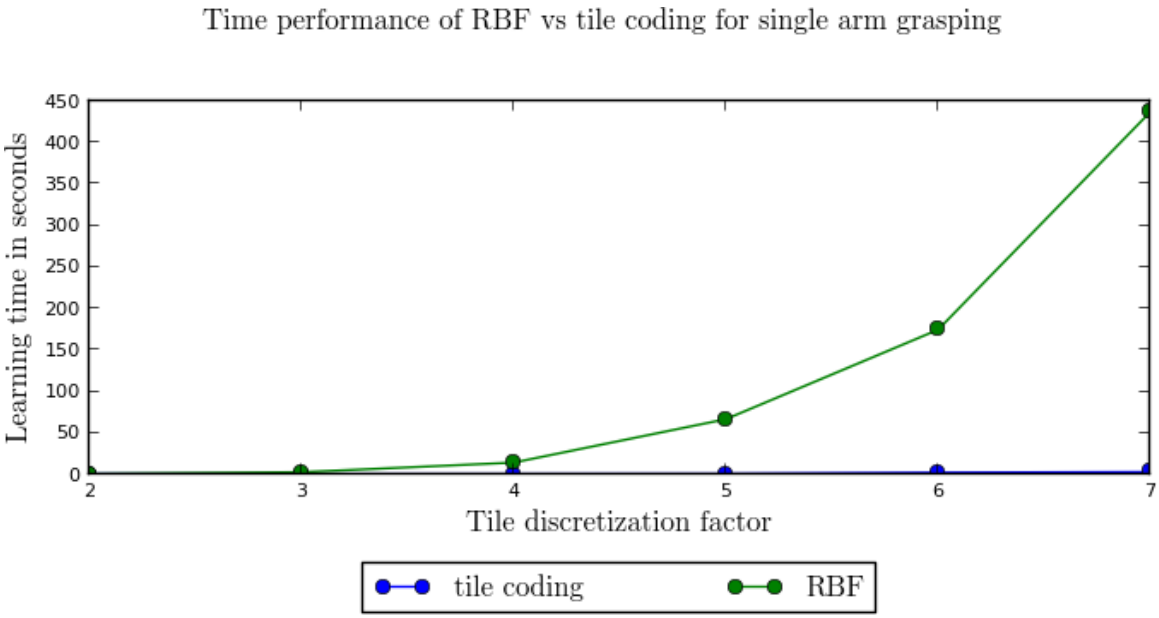


Figure 4-1: Time Performance of RBF versus Tile Coding

we discretized the state space using a uniform grid for the tile coding. Then, we used the same discretization to place RBF features at each point of the grid. Thus, the number of features are the same. Figure 4-1 shows the time for executing one episode of SARSA for a single-arm grasp.

We evaluated the time performance at different tile discretization factors which are described in detail in Subsection 4.1.5. The learning time for RBF increases extensively with the tile discretization factor, whereas tile coding appears constant. There is, in fact, an increase in time for tile coding that is not shown in this graph because tile coding is significantly faster than RBF. This is not surprising given tile features are binary valued and are evaluated in constant time.

Since RBF features took such a long time to compute, we used tile coding to represent the state space in our simulations and experiments. In the results, detailed below, we will see that tile coding had good performance even though it is possibly less descriptive than RBF.

4.1.2 Simulation Experiment

We tested our implementation on a variety of different objects located in different starting locations relative to the robot. We tested different initial locations for single and dual-arm manipulation. To give an overview of our quantitative results, we will discuss two experiments: dual-arm grasping with a large box and single-arm grasping with a small box.

One of the important properties of function approximation methods is its ability to generalize. Essentially, this means we should be able to use our reinforcement learning algorithm with a limited set of training locations for the obstacle, and then use the derived policy for starting locations outside of the training set. To test our implementation and its ability to generalize we have created three test cases that vary across training data. Each test contains a testing region, which is a $0.254m \times 0.254m$ box and contains 100 evenly distributed testing points. The three test cases are:

Base Policy The robot executes a simple policy that increments each joint at every time step.

Single Training Case The learning algorithm trains with the origin of the testing region as the sole training point.

Multiple Training Case The learning algorithm trains with a set of testing points distributed about the region.

Evaluation

Each policy is evaluated with 100 test points in simulation as described above. While executing a policy, the robot accumulates a reward at every state it encounters. The overall score for a policy is the accumulated discounted reward as described in Section 2.2. To evaluate a policy for a specific test point we look at two metrics: a boolean value of whether or not the policy terminated in a successful goal state and the accumulated discounted reward. A state is a goal state if the object is in a force closure grasp. A state is a failure state if it falls into one of the following categories:

Failure Modes

1. **Time out:** The robot took 35 steps and did not arrive at a terminating state.

2. **Links touch:** The right and left arms of the robot have collided. (This mode is included as a failure to ensure the robot does not collide into itself and cause any damage)

3. **Object too far:** The object is farther away than the length of the robot's arm. This occurs when the robot pushes the object away from the grasp and the object is no longer within reach.

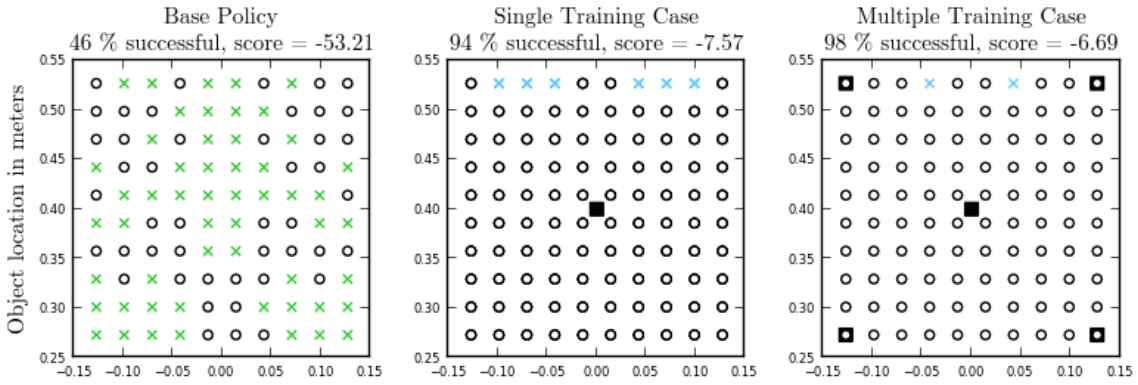
4.1.3 Simulation Results

Figure 4-2 shows the results from the three test cases for a dual-arm and single-arm grasping task. Each graph shows the 100 evenly distributed points within the testing region; the axes are with respect to the torso-link frame. The black squares indicate the initial positions of the training locations. The white circles indicate successful policies starting from that initial position and the 'x' markers indicate policies starting from that state led to a failure state.

In both cases, we see the base policy performs much worse than the learned policies. In the dual-arm case, the base policy only has green colored failures. This means it failed because the links touched. The learned policies only failed due to the policy taking more than 35 steps. None of the policies had the third failure mode of moving the object too far away.

For the single-arm case, there were two types of failures: time out and the object moving too far away. The base policy had the most failures due to moving the object too far away, followed by the single training case. The multiple training case showed the best results with the highest percent of successful grasps and the highest average accumulated reward.

(a) Dual-Arm Simulation Results



(b) Single-Arm Simulation Results

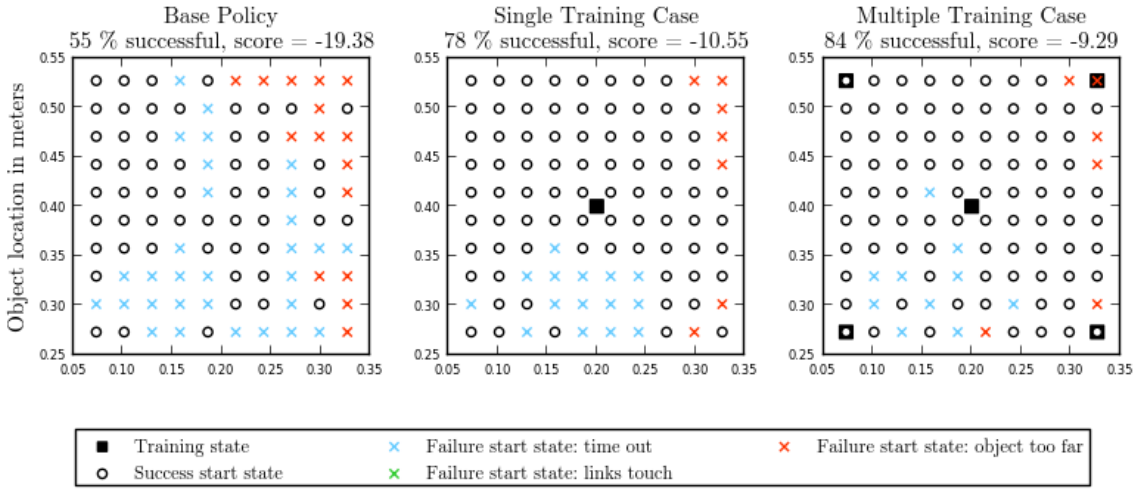


Figure 4-2: Simulation Results

4.1.4 Terminating State Locations

To characterize the derived grasping policies, we looked at the terminating state of a grasp. The following analysis builds upon our previous simulation results. In the previous graphs, we placed the object origin at each specific test point and then executed the policy until it reached a terminating state. We then labeled the initial starting points with their terminating mode. Now, we are including the final position

of the object when the robot completed its execution.

Figures 4-3 and 4-4 show the terminating state simulation results. If the grasp was successful, then the goal state is shown with a white diamond. If the grasp was unsuccessful, the failure state is indicated with a cross and a color indicating its failure mode. The interesting result from the graphs is that most of the successful grasps terminated in similar positions.

The dual-arm case, shown in Figure 4-3, has a little more variation from the single and multiple training cases; however in general all of the grasps are within a relatively small area compared to the testing region box. This shows an interesting result that in order to grasp an object, the robot intentionally moved it to its torso where it could use its torso as another contact for the object and achieve force closure.

In the single-arm case, shown in Figure 4-4, the terminating successful states seem to have an even tighter grouping than in the dual-arm case. Additionally, most of the terminating success states end outside of the testing region which means the object had to move a significant amount in order to achieve a force closure grasp. Once again, the multiple training case showed the best results and had very few failures due to moving the object too far away.

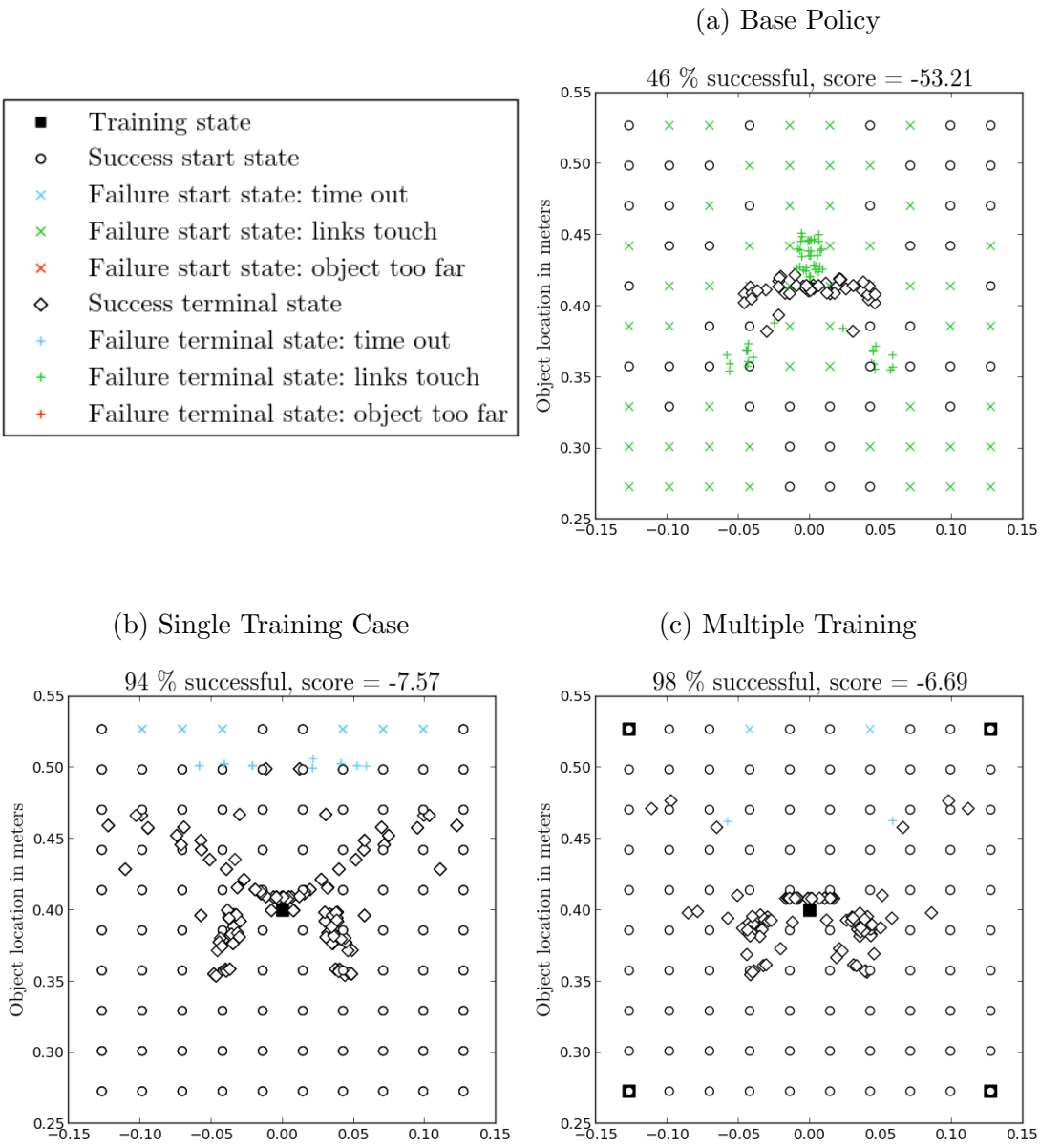
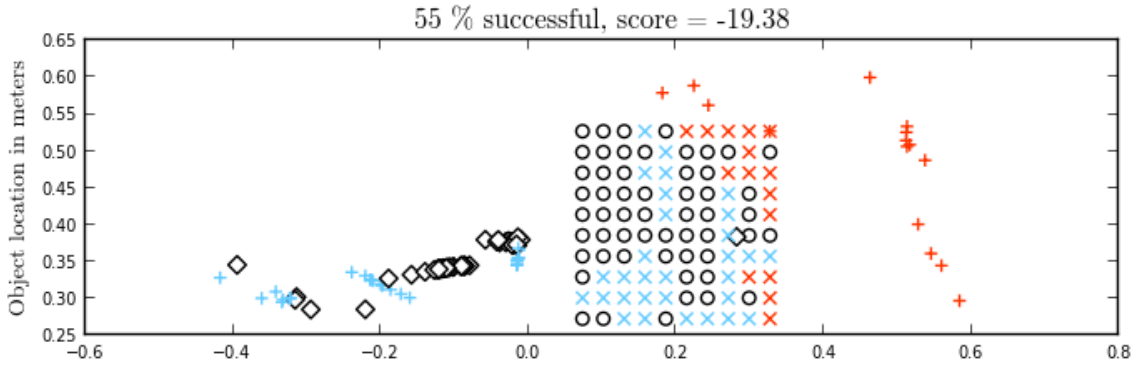
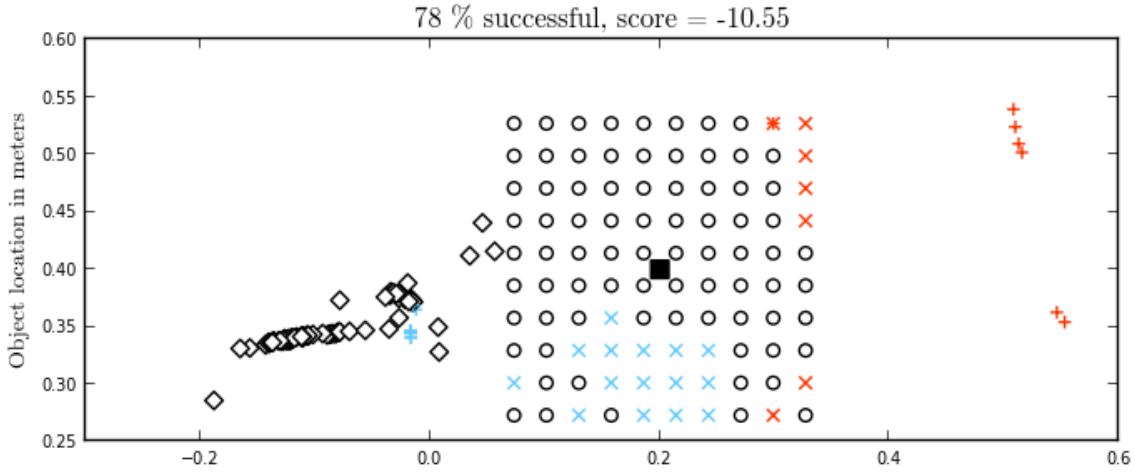


Figure 4-3: Dual-Arm Terminating State Results

(a) Base Policy



(b) Single Training Case



(c) Multiple Training

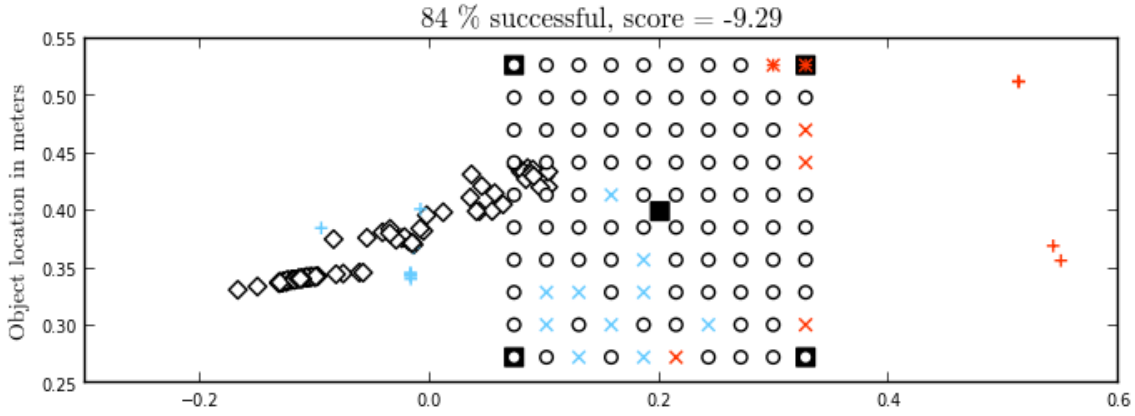


Figure 4-4: Single-Arm Terminating State Results

4.1.5 Parameter's Effect on Performance

There are two major parameters in the learning algorithm: number of learning episodes and tile discretization factor. A learning episode is one execution of the SARSA algorithm from an initial start state to a terminal state as described in Section 2.3. For the single-arm manipulation we tested five types of tile discretization factors and for the dual-arm case we tested four types of discretization factors. Table 4.1 shows the total number of tiles for each discretization factor type. In our performance plots, we describe our plots using tile discretization names listed in the table rather than the total number of tiles. For example, we will describe a line as "2 tiles" rather than 64 tiles in a single-arm manipulation graph.

Manipulation Type	2 Tiles	3 Tiles	4 Tiles	5 Tiles	6 Tiles
Single-Arm	64	729	4096	15625	46656
Dual-Arm	512	19683	262144	1953125	n/a

Table 4.1: Tile Discretization Types

Time Performance

First, we review the effect tile discretization and the number of learning episodes has on the learning time. Figure 4-5 shows the performance for both single and dual-arm manipulation. In general, the multiple training cases have a larger learning time than the single training cases. This is a natural result because the multiple training cases do learning from five more start states. As expected, increasing the number of learning episodes tends to increase the total time. Additionally, increasing the number of tiles also increases the total learning time. The time performance results are encouraging because they are all under two seconds.

Grasp Performance

In this section we will review the effect of tile discretization and the number of learning episodes on grasp performance. Figure 4-6 shows the performance for dual-arm

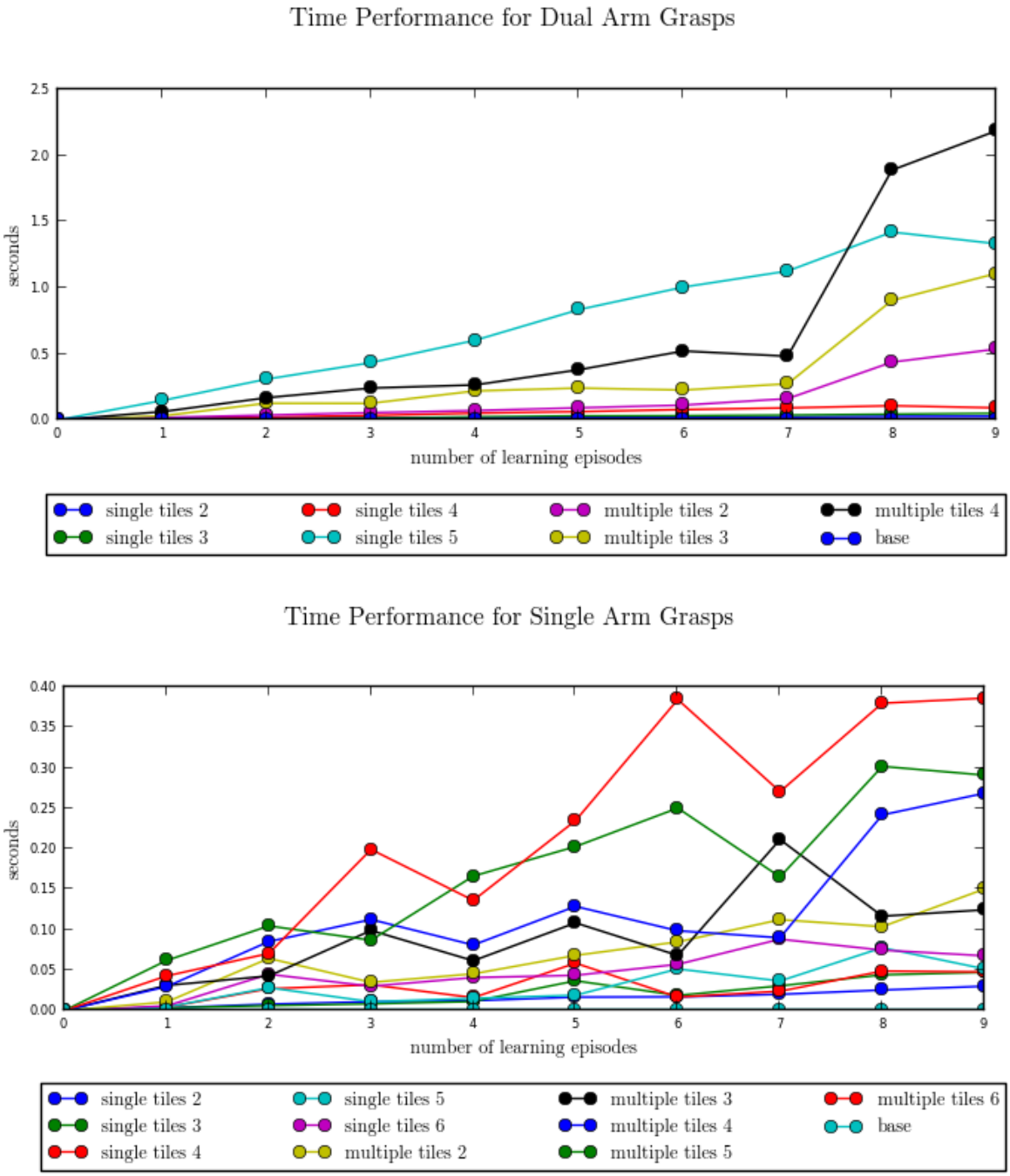


Figure 4-5: Time Performance vs. Number of Learning Episodes

grasping. The top graph shows the percent of successful grasps and the bottom graph shows the average discounted cumulative reward. The average discounted cumulative reward plot is slightly smoother than the percent successful plot; however, they both show the same behavior. The 2 tiles training cases never perform better than the base policy. The training cases with more than 2 tiles perform better than the base policy and tend to converge after one episode.

Figure 4-7 and 4-8 show the performance for single-arm manipulation. These results are much more noisy than the dual-arm case. Additionally, the percent graph differs from the average reward graph in a significant way. In the percent graph four trials have a lower percent than the base policy; however, the base policy scores the lowest in the average reward graph. Similarly to the dual-arm case, most of the trials peak after the first learning episode.

The most curious aspect of these results is the fact that most of the learning happened in the first episode. This is caused by the action selection during the reinforcement learning. The set of actions are ordered with the first action being the base policy, which increments every joint. During the learning algorithm, all ties are handeled by selecting actions in order to the predefined action set. This leads to selecting the base policy action quite often. Essentially, this means our learning algorithm builds upon the base policy, which already works over 45% of the time for both experiments.

For the dual-arm case, the problem is not too difficult. The robot learned to use the base policy for most of its actions, but when it was located in a particular tile, it took the action that incremented only the wrist joints. By doing so, the robot learned to not touch its links together. Once this failure mode was avoided, the robot was able to grasp these large objects fairly simply because they had no rotation.

The single-arm case was a much more difficult problem and did not show the same results. The single-arm case shows much more noise, but tends to peak or reach near a peak after one episode as well. For example, if we consider 'single tiles 3' in the single-arm performance graphs we notice they tend to oscilate around 75% successful with a reward of -11. Additionally, increasing the number of episodes does not have

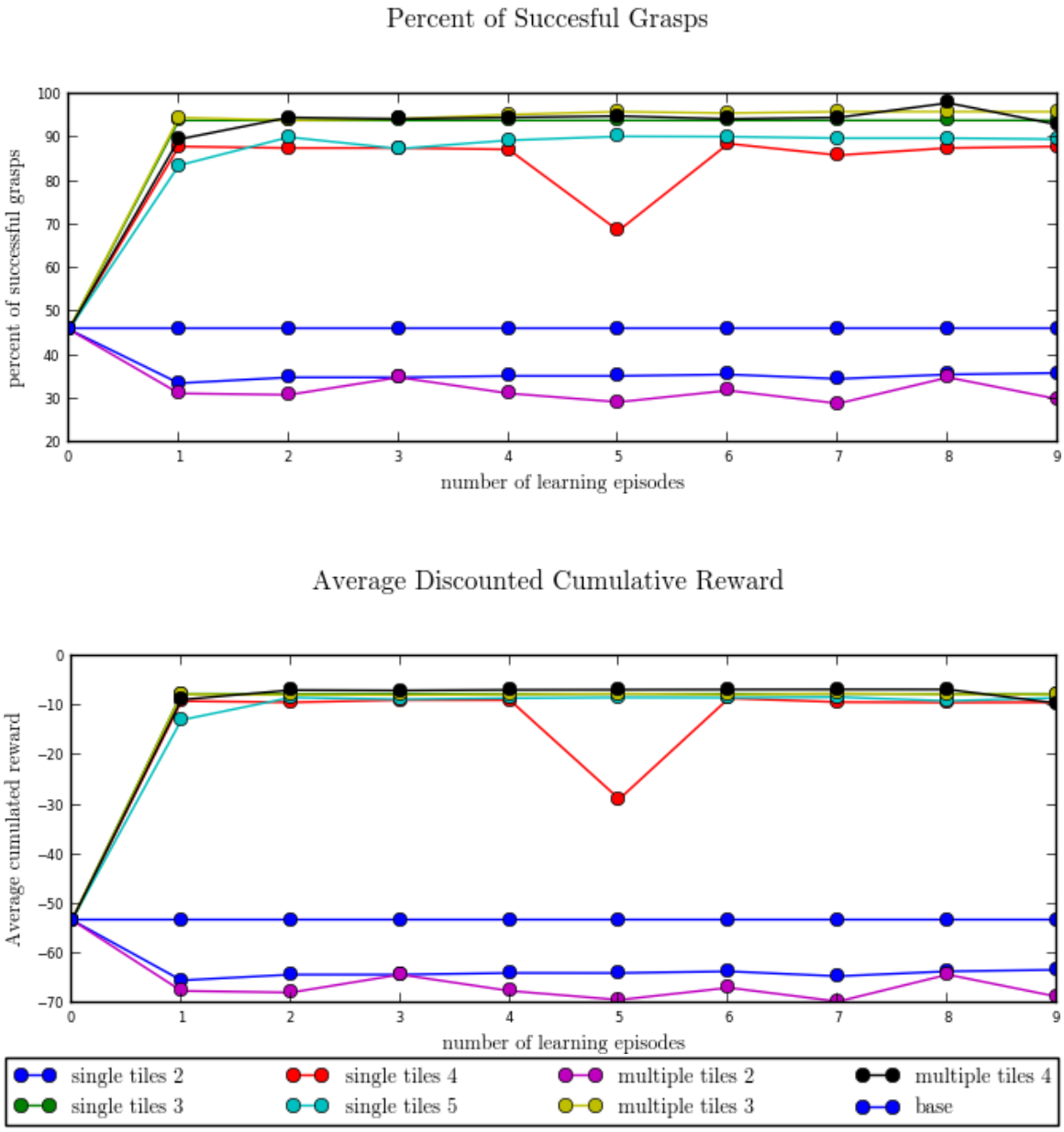
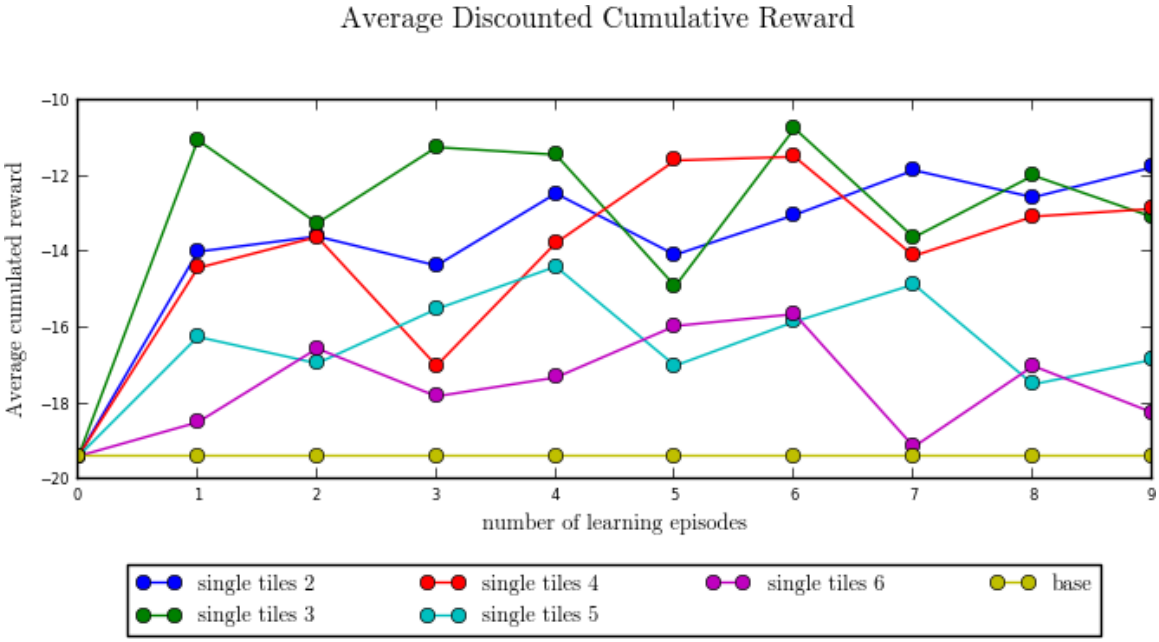


Figure 4-6: Dual-Arm Grasps Performance vs. Number of Learning Episodes

(a) Single Training Case



(b) Multiple Training Case

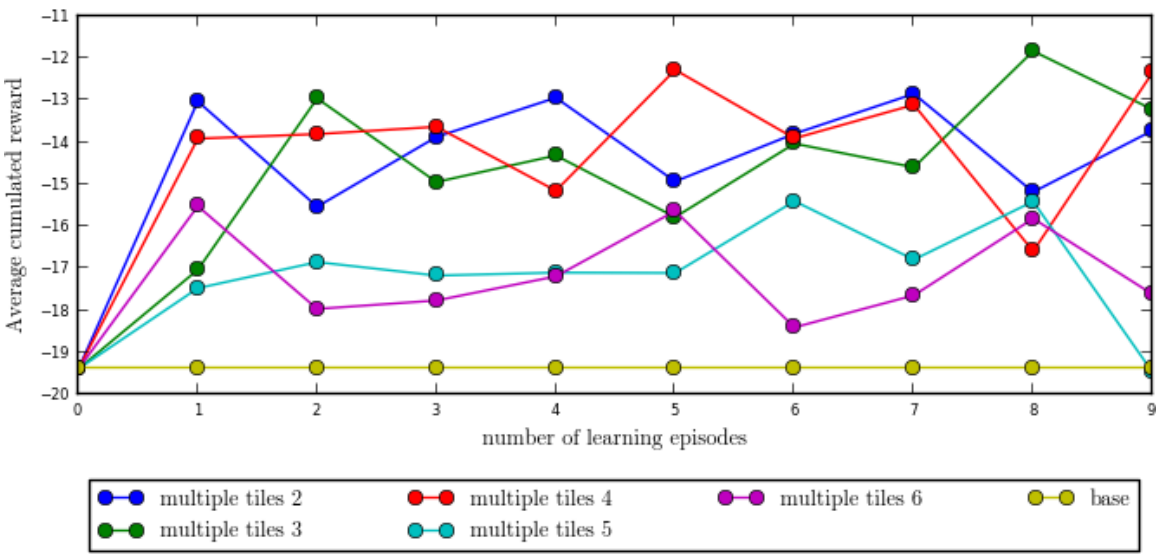
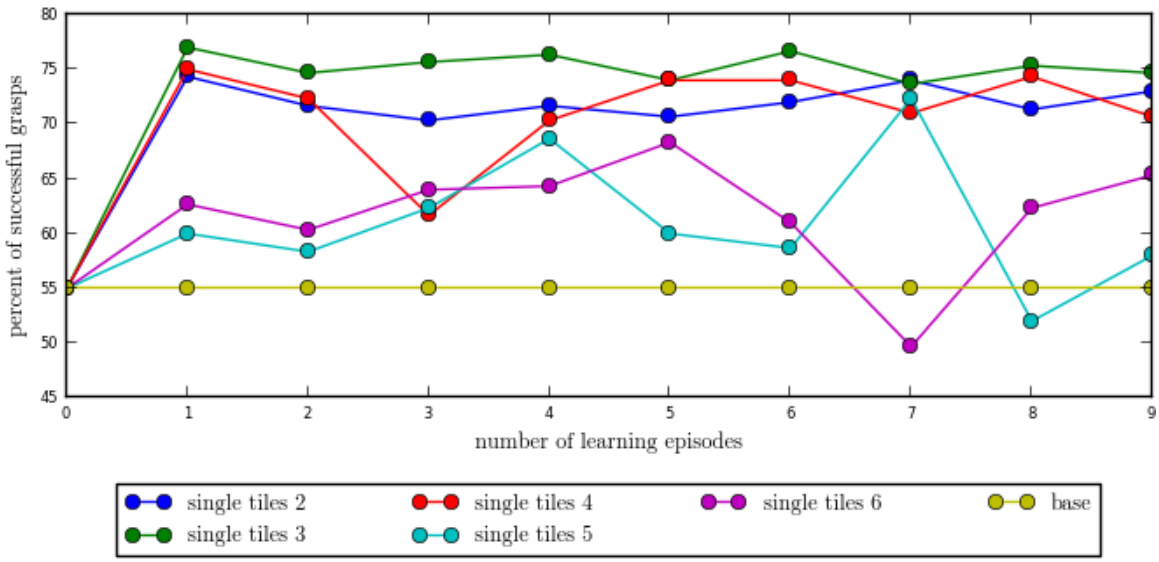


Figure 4-7: Single-Arm Grasps Average Reward Performance vs. Number of Learning Episodes

(a) Single Training Case



(b) Multiple Training Case

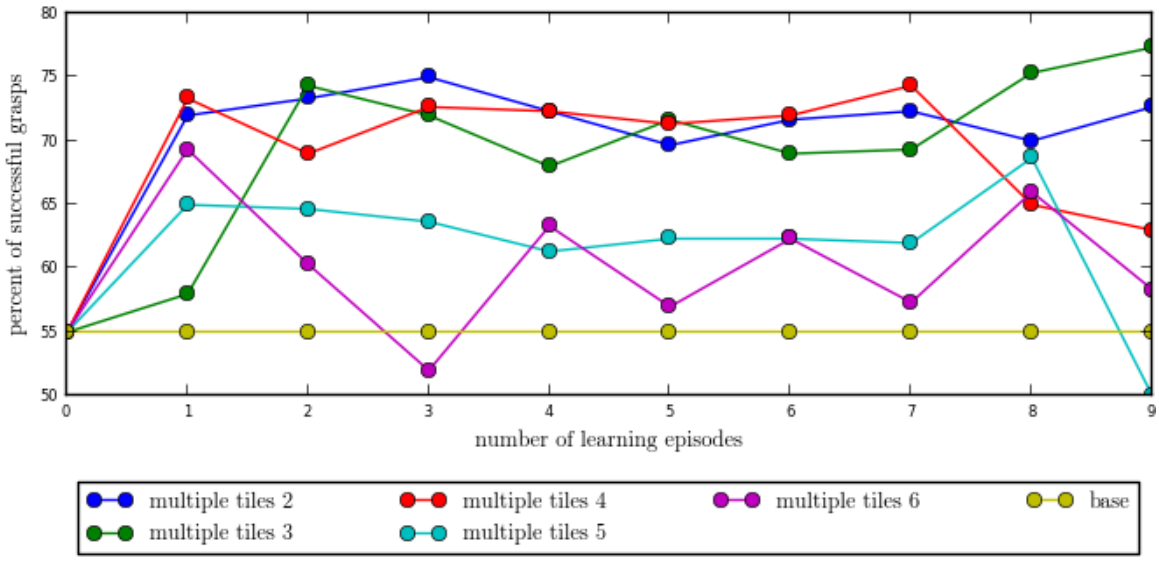


Figure 4-8: Single-Arm Grasps Percent Successful Performance vs. Number of Learning Episodes

a dramatic effect on performance.

The goal of these experiments were to analyze our algorithm's ability to generalize. The derived policies for the dual-arm case showed positive generalization results and could consistently grasp objects outside of the training data. However, the single-arm case did not show as good of results. We believe this could be caused by two main problems. First, the tile coding may not be discretized enough to accurately represent the state. Second, the reinforcement learning algorithm does not explore much of the space.

4.1.6 Simulation Results Overview

We tested our implementation in simulation for single and dual-arm grasping using objects located within a $0.254m \times 0.254m$ testing region. The testing region was used to test our implementation's ability to generalize, the ability to perform well on states unvisited in learning.

Overall, these experiments show two main results. First, using policies derived from reinforcement learning performed better than using a basic grasping strategy of incrementing each joint. Also, using multiple training cases performed better than a single initial training position.

Second, the grasps incorporated object dynamics. In both the single and dual-arm grasping experiments we saw successful terminal states occurred in groupings away from the initial state positions. This implies the robot had to move the object to a position where it could achieve a force closure grasp.

In addition to successfully achieving grasps, our implementation had positive time performance results. We were able to derive successful policies that were solved in seconds. This fast learning time is due to the fact that most of the policies took relatively few learning episodes to reach top performance. This indicates that the problem is relatively easy and simple to solve. Therefore, we can conclude that this approach could be reasonable to extend to handle more uncertainty by adding more

stochasticity in the simulator and a more complex domain with varying orientations and non-convex objects.

4.2 Real Robot

We executed our approach on the PR2 as outlined in Section 3.4. The object is imported into the simulator and a policy is derived from reinforcement learning. To execute the policy the robot imports its current joint angles and the observed object location into the simulator. The simulator uses the policy to return the next action and computes the reward of the current state. The trial ends when the simulator rewards the imported state as a terminal state or if 35 steps have been taken. After the trial ends we elevate the PR2's torso. We evaluate a grasp as successful if the object is lifted off the table when the torso is elevated. In our results, we also include whether the simulator detected the final state as successful or not.

Overall, we had mixed results for single and dual-arm manipulation. Table 4.3 and Table 4.2 shows quantitative results of testing our grasping approach on the PR2. Each table shows the results for 10 test grasps. We tested the dual-arm grasps with a large box and the single-arm grasps with a small box. We give an in-depth analysis on some of the trials indicated with an asterisk.

4.2.1 Dual-Arm Grasping Results

Table 4.2 shows the results of 10 trials of dual-arm grasping with a large box. The robot successfully grasped the object nine out of ten times. The single failure case occurred in Trial 6 when the object was rotated $\frac{\pi}{2}$ radians. However, Trial 7 had a different initial location with the same rotation and was successful. In some trials the PR2 successfully grasped the object, but the simulator resulted in a failed state. To describe some of the results, we will give an in depth analysis of Trials 2,5, and 6.

Trial	Object Location	rotation	Success (reality)	Success (simulation)
1	(0.35, 5.5)	0	Yes	No
2*	(0.11, 4.57)	0	Yes	Yes
3	(0.13, 4.44)	0	Yes	No
4	(0.82, 5.38)	0	Yes	No
5*	(-0.79, 5.06)	0	Yes; links touched	Yes
6*	(0.49, 4.90)	$\frac{\pi}{2}$	No	No
7	(-0.19, 4.18)	$\frac{\pi}{2}$	Yes	No
8	(0.31, 4.40)	$\frac{\pi}{4}$	Yes	No
9	(1.10, 5.55)	$\frac{\pi}{4}$	Yes	Yes
10	(0.96, 5.10)	$\frac{\pi}{4}$	Yes	Yes
	Total		9	4

Table 4.2: Dual-Arm Grasping Results

(a) PR2 Terminal State

(b) Simulator Terminal State

Figure 4-9: Terminal State for Trial 2 of Dual-Arm Grasping Results
Success with Simulator and with PR2

Trial 2 was successful in reality and simulation. Figure 4-9 shows the successful terminal states in simulation and with the real robot. Although this was a positive

result, the simulated robot does not have the contact with the right end effector.

(a) PR2 Terminal State

(b) Simulator Terminal State

Figure 4-10: Terminal State for Trial 5 of Dual-Arm Grasping Results

In Trial 5 the PR2's arms touched while executing the policy. The end result was a successful grasp in reality and in simulation. However, the simulator should have returned a failed state because of the arms touching failure mode. Figure 4-10b and Figure 4-10a show images of the terminal state in simulation and on the PR2. The large box has a very different orientation in the simulator than in the real world. This offset in orientation made it possible for the links to not touch in the simulator. Although the misrepresentation of the object did not result in a failed grasp, it did have the undesired effect of allowing the PR2 links to collide.

(a) PR2 Terminal State (b) Side View of PR2 (c) Simulator Terminal State

Figure 4-11: Terminal State for Trial 6 of Dual-Arm Grasping Results Failure with Simulator and with PR2

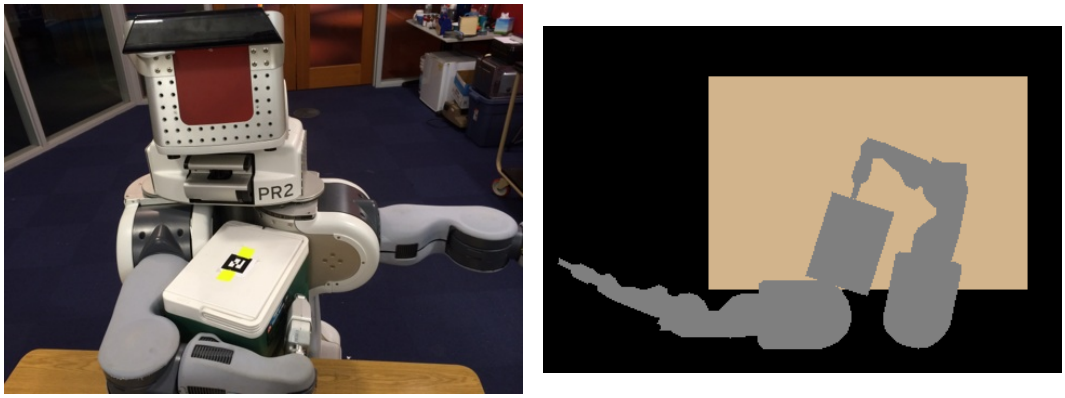
Trial 6 failed in reality and simulation. Figure 4-11 shows the failure terminal states in simulation and with the real robot. This trial used a initial starting location with an object rotated $\frac{\pi}{2}$ radians from all of the training locations. It's possible that training with rotated initial object locations could have resulted in a successful grasp.

4.2.2 Single-Arm Grasping Results

Table 4.3 shows the results of 10 trials of single-arm grasping with a small box. The robot grasped the object six out of ten times. Half of the failures occurred when the object was rotated $\frac{\pi}{4}$ radians. Surprisingly, the successful grasps in reality were not always consistent with the successful grasps in simulation. To understand this result, we will discuss a few of the trials.

Trial	Object Location	rotation	Success (reality)	Success (simulation)
1*	(0.188288,4.39037)	0	Yes	No
2*	(1.17102,3.86922)	0	No	Yes
3*	(0.288461,3.7046)	0	Yes	Yes
4*	(0.244427,4.44992)	0	Yes	No
5	(0.498114,3.69268)	0	No	Yes
6	(1.43259,4.24403)	0	Yes	No
7*	(0.956065,4.49392)	$\frac{\pi}{2}$	No	No
8	(-0.474323,4.10908)	$\frac{\pi}{2}$	No	No
9	((1.32666,4.63726)	$\frac{\pi}{4}$	Yes	No
10	(0.921646,4.91485)	$\frac{\pi}{4}$	Yes	No
	Total		6/10	3/10

Table 4.3: Single-Arm Grasping Results



(a) PR2 Terminal State

(b) Simulator Terminal State

Figure 4-12: Terminal State for Trial 1 of Single-Arm Grasping Results
Failure in Simulator, but Success with PR2

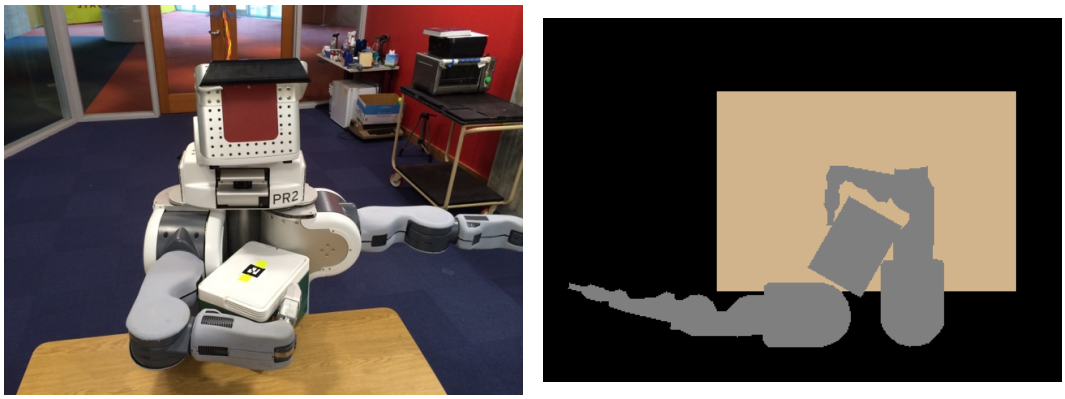
Trial 1 was successful in reality, but failed in simulation. Figure 4-12 shows the terminal states of the actual robot and the simulator. In this figure, we can see the simulator does not accurately model the actual behavior of the object. The observed

joint angles and object location are imported into the simulator, and the simulator moves to recreate this state. If the object location, joint angles, and robot model are inaccurate then it is not always possible to represent the state, as shown in this figure where the robot's arm is clearly misrepresented.

(a) PR2 Terminal State (b) Side View of PR2 (c) Simulator Terminal State

Figure 4-13: Terminal State for Trial 2 of Single-Arm Grasping Results
Success with Simulator, but Failure with PR2

Trial 2 failed in reality, but was successful in simulation. An image of Trial 2 is shown in Figure 4-13. In this case the robot did, in fact, have good contact positions, but was not applying enough force to hold the object when the torso was elevated.



(a) PR2 Terminal State (b) Simulator Terminal State

Figure 4-14: Terminal State for Trial 3 of Single-Arm Grasping Results
Success with Simulator and PR2

Trial 3 was successful in both the simulator and the PR2. Figure 4-14 shows the successful terminating state in simulation and reality. Although this is a positive result, we do not see the extra contact the robot is making with its left arm on the box.

Figure 4-15: Failure State for Trial 7 of Single-Arm Grasping Results
Failure in Simulator and with PR2

Trial 7 was a failure in reality and simulation. Figure 4-15 shows the failure on the actual robot. In this case, the object started fairly far away and was pushed away from a grasp.

4.2.3 Real Robot Results Overview

The results of our experiments on the real robot were mixed. The dual-arm grasping was successful in nine out of ten trials and the single-arm grasping was successful

in six out of ten. Most of the grasping errors occurred when the object was rotated substantially from the training cases. We did not use rotated objects for training states in our learning algorithm and this result suggests we might want to do so. Although some grasps were successful in reality, many returned failures in simulation. Additionally, there were many inconsistencies of the simulated robot and the actual world state. This result indicates that we need a more principled approach in handling the uncertainty of the observed world state and dynamics. Furthermore, for the single-arm experiments we saw a grasp fail in reality although it had relatively good contacts. This indicates that we might require a method to apply force at the contacts and not just use position control.

Chapter 5

Conclusion and Future Work

In this thesis we presented a novel approach for robotic whole-arm grasping using reinforcement learning. Our main focus in this project was treating grasping as a process. Traditional grasping approaches separate grasping into distinct phases: finding contacts for an optimal grasp and moving the robot to the optimal contacts positions. We wanted to develop a method that considers the process of bringing the object into a grasp and not just moving to the ideal final contacts.

By constructing the problem as a continuous space Markov Decision Process (MDP) we were able to incorporate the object dynamics in the planning process. This resulted in policies that exploit object dynamics, where the robot deliberately moved the object into its torso to achieve a successful grasp. This was a substantial departure from traditional grasping techniques where the object moving is typically an undesired result.

In our approach the final grasp position is not given beforehand, only the *criteria* for the desired goal state is given. This leads to a more flexible algorithm because the criteria are consistent with a wide variety of final grasp configurations. This is important because manipulation is a stochastic process and we cant generally guarantee that a specific target state can be achieved. However, by constructing a policy for our MDP we can guide the object towards one of the acceptable final states.

Since we used a continuous MDP and a simulator rather than an explicit transition model we could not apply optimal solutions for solving MDPs, such as value itera-

tion. Instead, we used value-based function-approximation approaches for model-free reinforcement learning. This approach does not scale well to high-dimensional spaces but has proven effective for planar manipulation problems.

We tested this approach extensively in simulation and executed a proof of concept on a real robot. The implementation on the robot had promising results and completed single and dual arm grasps on different testing objects. Some of the failures when executing with a real robot are caused by our inability to accurately determine the world state or simulate the dynamics.

The MDP framework we used in this thesis assumes the world state is completely observable. However, in reality there is noise in the robot's observations leading to undesired behavior while executing a policy. Additionally, we also have the problem of our simulator not being an accurate reflection of the underlying physics. When the simulator was used to learn a Q-value function and the derived policy was executed in simulation, we had positive results; however, execution on the robot resulted in failures because the transitions were not predicted correctly. This may be addressable by introducing appropriate noise into the transition model or may require recasting the problem as partially observable.

Although the approach suffered from having an inaccurate transition model and treating the world as completely observable, it did show a promising result for treating grasping as a process. Not only did the policies take object dynamics into account, they exploited them to move the object into a location the robot could grasp the object. This was shown in the terminating state plots in Section 4.1.4. The successful grasp locations were all near the torso where the robot could use the torso to apply an extra contact and achieve force closure. Overall, we believe this approach has promise and is a starting base for investigating robotic grasping as a single, cohesive process.

5.1 Future Work

Our investigation into grasping as a process has shown promising results; however there is room for development. First, we will discuss direct extensions to the approach and then we will discuss future research areas for investigation.

Current Approach

There is room for development and experimentation with our current approach. Currently, we are using a uniform grid for our tile coding to represent the state space; however, we could look into multiple and non-uniform tile codings. Using multiple grids could potentially generalize the state space better. Additionally, we could use a higher discretization in parts of the state space where the dynamics are more interesting. For example, the arm typically does not come into contact when the shoulder joint angle is $\in (0, \pi/6)$ so the discretization for those angles should be larger than the rest of the space.

When executing our approach on a real robot, we found inaccuracies in the observed state and errors in the simulated transition model. A potential way to handle this would be to introduce noise into the simulator's transition model. Currently there is some stochasticity in the transition model; however, it would be interesting to test the effects of adding more stochasticity into the transition model.

Another area we are interested in looking into is a more thorough analysis of our approach. We would like to test using different starting orientations and non-convex objects. Furthermore, it would be interesting to compare with another whole-arm grasping approach, such as the one developed by Seo et al. [18].

Future Directions

There are three principal directions for future work.

First, we would like to tackle the problem of inaccuracy in the transition model and error in the observed state. Currently, we are handling this uncertainty by treating the robots actions as stochastic. Next we will attempt to introduce stochasticity

Figure 5-1: Barrett Hand [16]

directly into the transition model. However, it may be that the state uncertainty is too significant for the MDP approach and we will need to cast the problem as a Partially Observable Markov Decision Process (POMDP). But, solving continuous state POMDPs represents a substantial challenge, although some promising approaches exist [20].

Second, we would like to tackle the problem of dimensionality, for example, for grasping objects in full three dimensions. It may be that in this setting we will need to pursue Policy Search methods, a reinforcement learning approach that scales with higher dimensionality better than value-function based approaches.

The third area would be tackling different robot models. We would like to test this approach on a multi-fingered gripper like the Barrett Hand shown in figure 5-1. Additionally, we are interested in seeing the performance of whole-arm grasping of interesting non-convex objects.

Bibliography

[1] Robot operating system (ROS). `http://www.ros.org`.

[2] pybox2d 2.3b0. `https://code.google.com/p/pybox2d/`, 2013.

[3] Andrew G Barto and Richard S Sutton. *Reinforcement learning: An introduction*. MIT press, 1998.

[4] Antonio Bicchi and Vijay Kumar. Robotic grasping and contact: A review. In *Robotics and Automation, IEEE International Conference on*, pages 348–353. Citeseer, 2000.

[5] Erin Catto. Box2d v2.3.1. `http://box2d.org`, 2014.

[6] Anne Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203–224, 1992.

[7] William Dabney and Andrew G Barto. Adaptive step-size for online temporal difference learning. In *AAAI*, 2012.

[8] SciPy developers. SciPy 0.13.3. `http://www.scipy.org`, February 2014.

[9] Carlo Ferrari and John Canny. Planning optimal grasps. In *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, pages 2290–2295. IEEE, 1992.

[10] Willow Garage. Overview — willow garage. `http://www.willowgarage.com/pages/pr2/overview`, May 2014.

[11] Kaijen Hsiao and Tomas Lozano-Perez. Imitation learning of whole-body grasps. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 5657–5662. IEEE, 2006.

[12] Moslem Kazemi, Jean-Sebastien Valois, J Andrew Bagnell, and Nancy Pollard. Robust object grasping using force compliant motion primitives. 2012.

[13] Jens Kober and Jan Peters. Reinforcement learning in robotics: A survey. In *Reinforcement Learning*, pages 579–610. Springer, 2012.

[14] Matthew T Mason. *Mechanics of robotic manipulation*. MIT press, 2001.

[15] Jean Ponce and Bernard Faverjon. On computing three-finger force-closure grasps of polygonal objects. *Robotics and Automation, IEEE Transactions on*, 11(6):868–881, 1995.

[16] Robotics Business Review. Barrett technology leads with technology and transforms an industry. `http://www.roboticsbusinessreview.com/article/barrett-technology-leads-with-technology-and-transforms-an-industry/Barrett_Technology`, May 2014.

[17] Aurelien Ribon. Physics body editor. `http://www.aurelienribon.com/blog/projects/physics-body-editor/`, 2011.

[18] Jungwon Seo and Vijay Kumar. Spatial, bimanual, whole-arm grasping. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 257–264. IEEE, 2012.

[19] Karun B. Shimoga. Robot grasp synthesis algorithms: A survey. *The International Journal of Robotics Research*, 15(3):230–266, 1996.

[20] Adhiraj Somani, Nan Ye, David Hsu, and Wee Sun Lee. Despot: Online pomdp planning with regularization. In *Advances in Neural Information Processing Systems*, pages 1772–1780, 2013.

[21] Freek Stulp, Evangelos Theodorou, Jonas Buchli, and Stefan Schaal. Learning to grasp under uncertainty. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5703–5708. IEEE, 2011.

[22] Jeffrey C Trinkle, Jacob M Abel, and Richard P Paul. An investigation of frictionless enveloping grasping in the plane. *The International journal of robotics research*, 7(3):33–51, 1988.

[23] Chelsea C White III and Douglas J White. Markov decision processes. *European Journal of Operational Research*, 39(1):1–16, 1989.