BACHELOR'S THESIS

Degree in Mathematics

Title: Policy Transfer via Modularity

Author: Ignasi Clavera Gilaberte

Advisor: Pieter Abbeel

Department: University of California Berkeley, Electrical Engineering and Computer Science

Co-Advisor: Carme Torras

Department: Institut de Robòtica i Informàtica Industrial (CSIC-UPC)

Academic year: 2016 - 2017

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

Universitat Politècnica de Catalunya
Facultat de Matemàtiques i Estadística

Bachelor's Degree Thesis

Policy Transfer via Modularity

Ignasi Clavera Gilaberte

Advisor: Pieter Abbeel[1]
Co-advisor: Carme Torras[2]

1. University of California Berkeley, Department of Electrical Engineering and Computer Science
2. Institut de Robòtica i Informàtica Industrial (CSIC-UPC)

A society that fears knowledge is a society that fears itself.
— Bernard Beckett, Genesis

Abstract

To have robots that robustly interact with the environment and achieve meaningful tasks, they must be able to interact with objects. This is why an important challenge in Robotics is object manipulation. This thesis aims to tackle the specific problem of non-prehensile manipulation, such as pushing. Which is an important function for robots to move objects and is sometimes preferred as an alternative to grasping. However, due to unknown frictional forces, pushing has been proven a difficult task for robots. We explore the use of deep reinforcement learning to train a robot to robustly push an object.

In order to deal with the sample complexity of training such a method, we train the pushing policy in simulation and then transfer this policy to the real world. The transfer from simulation to real world is also important in terms of safety. During training the robot can get damaged while exploring new policies, the joints can worn out or even the people surrounding it are at risk of being hit. In order to ease the transfer, we propose to use modularity to separate the learned policy from the raw inputs and outputs; rather than training "end-to-end," we decompose our system into modules and train only a subset of these modules in simulation. We further demonstrate that we can incorporate prior knowledge about the task into the state space and the reward function to speed up convergence. Finally, we introduce "reward guiding" to modify the reward function and further reduce the training time. We demonstrate, in both simulation and real-world experiments, that such an approach can be used to reliably push an object from many initial positions and orientations.

Acknowledgements

I am very grateful of having had the opportunity to work with Professor Pieter Abbeel. His enthusiasm, eagerness to push Artificial Intelligence to its limits and ability to do so, I sincerely admire. But, even more important than him, has been his incredible group which I am proud to be part of. I would like to specially thanks David Held, who has mentored me all this year. And Carlos, who has been an amazing friend, and without whom this experience would not has been as half as fun and intense.

I will feel forever be indebted to Professor Carme Torras, who made all of this possible. Also the CFIS, and all the people and donors behind it, who support students in his way to excellence, deserve my most sincere gratitude. During all my years in the CFIS I had the opportunity to meet extraordinary people, but there are two who made a special impact in my life: Arnau and Carla.

Finally, I would like to mention my family that have always supported my decisions no matter how many kilometers these decisions put between us. And *Anna*, for her strength and love.

Contents

1 Introduction

1.1 Motivation

Reinforcement learning (RL) is a branch of machine learning that tackles problems of sequential decision making. It considers the setting where we have an agent in an environment, who chooses actions from a set of actions, upon which it receives a reward and an observation of the modified environment. The agent learns from direct interaction with the environment, without any external supervision nor model to describe the behaviour of the world. Its objective is to maximize the reward given along the trajectory. This completely general setup makes RL useful in a lot of domains, such as robotics, operations research, game play, dialog, and so forth. Every problem where an agent takes an action, which produces a change in the world, and later on acts again upon it is a valid framework where to apply reinforcement learning.

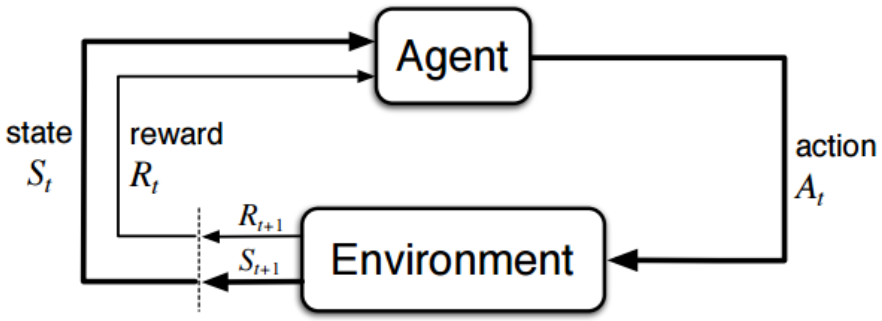

Figure 1.1: Agent-environment interaction in Reinforcement Learning.

Deep Reinforcement Learning (DRL) uses neural networks as function approximators. Those functions could be, for instance, the policy that the agent follows -its behaviour- or a parametrization of how good an action is.

The integration of reinforcement learning and neural networks dates back to the 1990s with Tesauro's work [1], among others. However, it was not until this decade with the explosion of deep learning in computer vision and speech recognition, thanks to the

increasing computational resources(Moore's law) and data, that deep reinforcement learning began to have incredible results in game playing and robotics.

The first impressive result was carried out by Mnih et al, who taught a DRL agent to play seven different 1980s Atari Games by observing the screen of the game and receiving the score as reward [2]. More impressive results have followed since then. In the field of game playing, [3] applied the same model to learn 49 different Atari games achieving superhuman performance in half of them. Also, Silver et al. [4] learned to play Go - one of the most challenging games for Artificial Intelligence (AI) due to the long term planning and huge search space - using supervised learning combined with reinforcement learning, winning the best human experts in the world.

Deep reinforcement learning has revolutionized the field of robotics as well. Levine et al. accomplished robotic manipulation using images as input [5]. They learned the vision and control together in an end-to-end fashion. Silver et al. [6], Lillicrap et al. [7] developed a policy gradient that allowed them to learn policies in continuous action space. Schulman et al. [8, 9] got incredible results in robot locomotion, a high-dimensional continuous control, using policy gradients.

1.2 Work Overview

This thesis intends to be a self-contained work, only expecting from the reader a mathematical background and some minor knowledge in Deep Learning. It contains the theory needed to understand the experimental work carried out, which was submited at the International Conference on Intelligent Robots and Systems (IROS).

Thus, this work it is divided in two parts. The first one is an explanation of all the theoretical concepts. Starting from MDPs and Bellman equations, followed by reward shaping. Finally, an introduction of policy gradient methods is presented and we explain the policy gradient method used: Trust Region Policy Optimization. In the second part, we describe our novel work in transfer from simulation to real-world "Policy Transfer via Modularity".

The work developed aims to tackle object manipulation, specifically non-prehensile manipulation, such as pushing using just simulation to learn the behaviour. This ambitious work try to solve important and hard problems in robotics and AI, object manipulation and transfer sim-to-real. Manipulating objects is inherently hard due to the unknown frictional forces and discontinuities, which results in errors in the models seldom negligible. Thus, it is important to learn from the experience (data), instead of developing a mathematical model of the dynamics of the objects. Deep reinforcement learning is able to just learn from data while trying to accomplish the task. The data learned then is meaningful for the task, and does not waste resources on trying to learn a global model.

We explore how to enhance reinforcement learning for a pushing task, to speed up the training and improve performance without modifying the true objective. Nevertheless, our key contribution is to develop a way to perform transfer from simulation to real world with comparable results between both domains. Reinforcement learning methods tend to be inefficient sampling-wise, leading to incredible results but needing thousands of trajectories. In real-world robotics this is a constraint not only

because of time constraints (real-world experiments are expensive to parallelize and the running time is slow), but also in terms of safety. Robots can be worn out and damaged while performing experiments, and the people around them can get injured. Therefore, our work is crucial for bringing robotics to industry and day-to-day life.

2
Markov Decision Processes

A Markov Decision Proces (MDP) provides a framework to reason mathematically about the planning and decision making of an agent in a stochastic environment [10]. An MDP $\mathcal{M}$ is defined as 6-tuple $(\mathcal{S}, \rho, \mathcal{A}, P(\cdot, \cdot|s, a), \gamma, R)$ where:

- $\mathcal{S}$: A set of states in the environment.
- ρ: An initial state s_0 state distribution, i.e, $s_0 \sim \rho(s)$.
- $\mathcal{A}$: A set of actions that the agent can select at each time step.
- $P(s', r|s, a)$: The transition probability distribution. Given that the agent is state s and takes action a, P gives the probability that the agent ends up in state s' and the environment emits a reward r.
- γ: It is parameter called the discount factor, $\gamma \in [0, 1]$.
- R: $R(s, a, s') : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function.

There are different formulations of an MDP in the literature. For instance, the reward function can be considered as deterministic with just dependence on the current state s or it can also depend on the action a taken. Moreover, there are settings in which there is no need to specify the initial state distribution.

An MDP is called *undiscounted* if γ is equal to 1, and *discounted* otherwise. Also the MDP is infinite if the time horizon $T = \infty$ and finite if $T < \infty$.

The transition probabilities of an MDP satisfy the *Markov Property*, the current state retain all the relevant information in order to act optimally. For example, in chess the current configuration of the figures in the board contain all the necessary information to play optimally, there is no need to know the history of actions and states that led to this configuration. More formally:

Definition (**Markov Property**). Let $(\Omega, \mathcal{F}, \mathcal{P})$ be a probability space and $\mathbf{X} = \{X_t : t \in \mathbb{N}\}$ a stochastic process where X_t is a random variable that takes values in a measurable space $(S, \mathcal{S})$. Let $\mathcal{F}_t$ denote the σ-algebra of events generated by $(X_k : s \leq t)$. The stochastic process $\mathbf{X}$ satisfies the *Markov property* if and only if the following holds: For each $k \leq t < T$, $A, x_k \in \mathcal{S}$ and $H \in \mathcal{F}_k$:

$$\mathcal{P}(X_t \in A|H, X_k = x_k) = \mathcal{P}(X_n \in A_n|X_k = x_k)$$

The setting of an MPD proceeds follows. The episode begins with a sample of an initial state s_0 from the probability distribution ρ. At each timestep $t < T, t \in \mathbb{Z}$, the agent picks an action a_t which leads him to the next state s_{t+1} and receives a reward r_t according to the probability distribution $\mathcal{P}(s_{t+1}, r_t|s_t, a_t)$. The episode is ended when a terminal state s_T is reached. The *return* of a rollout τ, $R(\tau)$, is the total sum of rewards weighted by the the discount factor: $R(\tau) = \sum_{t=0}^{T-1} r_t$.

In practice, most of the MDPs are discounted not only because it avoids infinite returns, but also because the future is often uncertain, and it might not be perfectly represented. Furthermore, from a human perspective, we tend to prefer immediate rewards rather than future ones.

As said, our main goal is that the agent's behaviour maximizes the return, i.e., the sum of rewards. The behaviour is modelled by a function named policy π that maps states to actions. There are to types of policies:

- Deterministic policy: $\pi(s) = a$
- Stochastic policy: $a \sim \pi(a|s)$

We will mostly focus on stochastic policies which give us the probability of taking certain action rather than the action per se. Therefore, we want to find π^*, the optimal policy, which correspond to

$$\arg\max_{\pi} \mathbb{E}_{\tau}[R(\tau)] \tag{2.1}$$

2.1 Value Function and Q-Value Function

The *Value function* or *state-value function* is a function of the state that provides an estimator of the goodness of your policy in a certain state s. The goodness is defined in terms of reward, the more reward your policy receives the better. Since the rewards depend on your behaviour the value function depends on the policy as well [11].

Definition. The state-value function of a policy π, V^{π}, is defined as

$$V^{\pi}(s) = \mathbb{E}_{a_t \sim \pi}[\sum_{t=0}^{T-1} \gamma^t R(s_t, a_t, s_{t+1})|s_0 = s] \tag{2.2}$$

Another important function in reinforcement learning is the *Q-value function* or *action-value function* which quantifies the goodness of taking the action a in the state s under the policy π.

Definition. The Q-value function of a policy π, V^{π}, is defined as

$$Q^{\pi}(s, a) = \mathbb{E}_{a_t \sim \pi}[\sum_{t=0}^{T-1} \gamma^t R(s_t, a_t, s_{t+1})|s_0 = s, a_0 = a] \tag{2.3}$$

These two functions can be estimated from experience, keeping an average of the estimate return for each state and action. While we are performing trajectories, the estimated value $\hat{V}^{\pi}$ and $\hat{Q}^{\pi}$ converge to their true value. This estimation methods are called *Monte Carlo Methods*. This methods will not be presented here since their are intractable when the dimensionality of the action space is big or when we are in a continuous state space. In practice in deep reinforcement learning, we use function approximation, i.e., we parametrize these functions (using a neural network), and fit them with the data sampled using supervised learning.

2.2 Bellman Equations

One important aspect of the value and Q function is their recurrent nature and their interchangeability. The *Bellman equation* for the value function is defined as

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{a_t\sim\pi}[\sum_{t=0}^{T-1}\gamma^t R(s_t,a_t,s_{t+1})|s_0=s] \\ &= \mathbb{E}_{a_t\sim\pi}[R(s_0,a_0,s_1)+\gamma\sum_{t=0}^{T-1}\gamma^t R(s_{t+1},a_{t+1},s_{t+2})|s_0=s] \\ &= \sum_a \pi(a|s)\sum_{s',r} p(s',r|s,a)(\,r+\gamma\mathbb{E}_{a_t\sim\pi}[\sum_{t=0}^{T-1}\gamma^t R(s_{t+1},a_{t+1},s_{t+2})|s_1=s'\,]) \\ &= \sum_a \pi(a|s)\sum_{s',r} p(s',r|s,a)(\,r+\gamma V^{\pi}(s')) \end{aligned} \tag{2.4}$$

This equation expresses the relationship between the current state and the next states. It can also be unrolled to k steps forward in the future. The value function V^{π} is the only solution of the Bellman Equation taking the expectation from π.

We can also define the Bellman Equation for the Q-function

$$\begin{aligned} Q^{\pi}(s,a) &= \mathbb{E}_{a_t\sim\pi}[\sum_{t=0}^{T-1}\gamma^t R(s_t,a_t,s_{t+1})|s_0=s,a_0=a] \\ &= \mathbb{E}_{a_t\sim\pi}[R(s_0,a_0,s_1)+\gamma\sum_{t=0}^{T-1}\gamma^t R(s_{t+1},a_{t+1},s_{t+2})|s_0=s,a_0=a] \\ &= \sum_{s',r} p(s',r|s,a)(\,r+\gamma\mathbb{E}_{a_t\sim\pi}[\sum_{t=0}^{T-1}\gamma^t R(s_{t+1},a_{t+1},s_{t+2})|s_1=s'\,]) \\ &= \sum_{s',r} p(s',r|s,a)(\,r+\gamma\sum_{a'}\pi(a'|s)Q^{\pi}(s',a')) \end{aligned} \tag{2.5}$$

It is interesting to see how are related the Q-value and the value function. For instance, we can express the value function in terms of the Q-function as

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{a_t \sim \pi}[\sum_{t=0}^{T-1} \gamma^t R(s_t, a_t, s_{t+1}|s_0 = s] \\ &= \sum_{a} \pi(a|s) Q^{\pi}(s, a) \end{aligned} \tag{2.6}$$

And we can also express the Q-function in terms of the value function

$$\begin{aligned} Q^{\pi}(s, a) &= \mathbb{E}_{a_t \sim \pi}[\sum_{t=0}^{T-1} \gamma^t R(s_t, a_t, s_{t+1}|s_0 = s, a_0 = a] \\ &= \sum_{s',r} p(s', r|s, a)(\, r + \gamma \mathbb{E}_{a_t \sim \pi}[\sum_{t=0}^{T-1} \gamma^t R(s_{t+1}, a_{t+1}, s_{t+2})|s_1 = s']) \\ &= \sum_{s',r} p(s', r|s, a)(\, r + \gamma V^{\pi}(s')) \\ &= \mathbb{E}[R_0 + \gamma V^{\pi}(s')] \end{aligned} \tag{2.7}$$

Interestingly, we can recover the value function from the Q-function without knowing the dynamics of the environment, but no the other way around.

2.3 Optimal Policies

The problem of reinforcement learning is to find the policy that achieves the maximum return. Therefore, the value function induce an order among the set of policies Π. We say that $pi \geq \pi'$, where $pi, pi' \in \Pi$, if and only if $V^{\pi}(s) \geq V^{\pi'}(s)\ \forall\ s \in \mathcal{S}$. One can proof that it always exists an optimal policy. We denote it as pi^* any policy in the set of optimal policies. The optimal value function is defined then as

$$V^*(s) = \max_{\pi} V^{\pi}(s) \tag{2.8}$$

$\forall\ s \in \mathcal{S}$. In the same way, we define the optimal Q-function as

$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a) \tag{2.9}$$

$\forall\ s \in \mathcal{S}$ and $a \in \mathcal{A}$. Analogously as shown in the previous section we can rewrite the optimal Q-function in terms of the optimal value function

$$Q^*(s, a) = \mathbb{E}[R_0 + \gamma V^*(s')|s_0 = s, a_0 = a] \tag{2.10}$$

Since V^* is the value function of some optimal policy π^* it must satisfy the Bellman Equations described before. Additionally, since it is the optimal we can re-write the equations so there is no reference to the optimal policy. This can be done because the

optimal value function must pick the action that leads to a maximum reward. Thus, rewriting the *Bellman optimality Equation* for the value function we obtain

$$\begin{aligned} V^*(s) &= \max_a Q^*(s,a) \\ &= \max_a \mathbb{E}_{a_t\sim\pi^*}[\sum_{t=0}^{T-1} \gamma^t R(s_t,a_t,s_{t+1})|s_0=s,a_0=a] \\ &= \max_a \mathbb{E}_{a_t\sim\pi^*}[R(s_0,a_0,s_1)+\gamma\sum_{t=0}^{T-1} \gamma^t R(s_{t+1},a_{t+1},s_{t+2})|s_0=s,a_0=a] \\ &= \max_a \mathbb{E}_{a_t\sim\pi^*}[R(s_0,a_0,s_1)+\gamma V^*(s_1)|s_0=s,a_0=a] \\ &= \max_a \sum_{s',r} p(s',r|s,a)[r+\gamma V^*(s')] \end{aligned} \tag{2.11}$$

We can perform similar operations to define the Bellman optimality Equation for the Q-function

$$\begin{aligned} Q^*(s,a) &= \mathbb{E}_{a_t\sim\pi^*}[R(s_0,a_0,s_1)\gamma \max_{a_1} Q^*(s_1,a_1)|s_0=s,a_0=a] \\ &= \sum_{s',r} p(s',r|s,a)[r\gamma \max_{a'} Q^*(s',a')] \end{aligned} \tag{2.12}$$

There is a huge variety of methods focused on solving these equations and finding the optimal policy, see [11] for a good introduction. In summary, the Q-function is used for model-free methods because you do not need to know any of the dynamics, i.e., you do not need a model and you learn just from the data, to choose the optimal action.

The work carried out in this thesis uses a different set of methods, called *policy gradients*, where the policy is approximated by a function parametrized by a parameter vector θ and the maximization of the expected return is performed with respect to those parameters. These methods are presented in Chapter 4.

3
Reward Shaping

The term *reward shaping* first appeared in [12] and consists in a modification of the reward function of the MDP that preserves the optimal policy. Modifying the reward function is vital in many environments. For instance, in setups where the reward is sparse, i.e., the agent just receives a reward if he had completed the task (or some hard sub-task), it is incredibly hard to learn since the agent does not have any feedback whether the actions taken are good or not. Therefore, a problem with sparse rewards can become intractable (it is needed too much exploration). However, a modification of the reward function leading to an observation of reward at each time-step, makes the learning easier turning those intractable problem into tractable. Reward shaping is also important in problems where the task required is complicated and the reward given not very informative. In our work we will show how reward shaping improves the learning in terms of performance and speed.

These modifications of the reward function must preserve the optimal policy. Otherwise, one can converge into a policy that does not have the desired behaviour. For example, suppose that we have an agent that wants to walk to a goal. Initially we have a reward function that is 1 if the agent reaches the goal, and 0 otherwise. We could define a reward shaping function which gives reward just if he goes near the goal (but it does not penalize if he walks away from it). Then the behaviour that the expert would learn is walk in circles approaching and walking away to the goal, because this is actually the optimal policy in this setup!

Using the same notation than in the former chapter, we are aiming at the optimal policy π^* for some MDP $\mathcal{M} = (\mathcal{S}, \rho, \mathcal{A}, P(\cdot, \cdot|s, a), \gamma, R)$. However, we want to modify the function R adding a shaping reward, such that it leads to a faster convergence of the optimal policy. Therefore, we will train our agent in a modified MDP $\mathcal{M}' = (\mathcal{S}, \rho, \mathcal{A}, P(\cdot, \cdot|s, a), \gamma, R')$, such that $R' = R + F(s, a, s')$. Where $F : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is called the shaping reward function. Since the only difference between both MDPs is the reward function, the agent and the environment will behave the same but at each transition instead of observing a reward $R(s, a, s')$ it will observe a reward $R(s, a, s') + F(s, a, s')$. This fact is important implementation wise, because in most of the cases we do not know the transition probabilities or even the set of feasible states, but we can run the same algorithm and pretend we are observing R' instead of R.

Hence, the goal of reward shaping is to find what functions F makes the optimal policy $\pi^*_{\mathcal{M}'}$ (optimal policy of $\mathcal{M}'$) also an optimal policy of $\mathcal{M}$. We will consider

the generic case where we do not know neither $P(\cdot,\cdot|s,a)$ nor $R(s,a,s')$. Although considering the reward unknown may be to restrictive, this is often the case in real world problems. The following result states the necessary and sufficient conditions of the function F.

Theorem. Let any $\mathcal{S}$, $\mathcal{A}$ and γ and a reward shaping function $F : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ be given. We say that F is a potential-based function if there exists a function $\Phi : \mathcal{S} \to \mathbb{R}$ such that $\forall\ a \in \mathcal{A}$, $s, s' \in \mathcal{S}$:

$$F(s,a,s') = \gamma\Phi(s') - \Phi(s) \tag{3.1}$$

Then, F being a potential-based function and Φ bounded are a necessary and sufficient condition to guarantee optimal consistency when learning from $\mathcal{M}'$ rather than $\mathcal{M}$ in the following sense:

- **Sufficiency**: If F is a potential based function, then every optimal policy in $\mathcal{M}'$ will be optimal policy in $\mathcal{M}$, and viceversa.
- **Necessity**: If F is not a potential-based function then exists a proper transition probabilities and reward function such that no optimal of $\mathcal{M}'$ is optimal of $\mathcal{M}$.

It is worth noticing that the sufficiency condition hold whether we know or not the transition probabilities and reward function, but the necessity condition explicitly states to those terms be unknown. Also, it might be the case that there is more than one optimal policy $\pi^*_{\mathcal{M}'}$ or $\pi^*_{\mathcal{M}'}$ but this theorem assures that *any* optimal policy $\pi^*_{\mathcal{M}'}$ will be optimal of $\mathcal{M}$.

An intuitive explanation of why it has to be a potential-based function is the following. Consider that it exists a sequence of states $s_0, s_1, \cdots, s_n, s_0$ such that the total gain of the shaping reward is positive, i.e., $\sum_{k=0}^{n-1} F(s_k, a_k, s_{k+1}) + F(s_n, a_n, s_0) > 0$ then the agent will repeatedly do this cycle instead of performing the actual task (like in the example explained before). On the other hand, if F is a potential-based function the sum of its values in any cycle is 0 and so the agent cannot exploit this trait. In the following we prove the sufficiency condition.

Proof. Let F be a potential-based function, with potential Φ. For the original MDP $\mathcal{M}$ a policy is optimal if its Q-function, $Q^*_{\mathcal{M}'}$ satisfies the Bellman Equations.

$$Q^*_{\mathcal{M}}(s,a) = \mathbb{E}_{s'}[R(s,a,s') + \gamma \max_{a' \in \mathcal{A}} Q^*_{\mathcal{M}}(s',a')]$$

If we subtract $\Phi(s)$ in both sides of the equation and add and subtract $\gamma\Phi(s')$ inside the expectation we end up with

$$Q^*_{\mathcal{M}}(s,a) - \Phi(s) = \mathbb{E}_{s'}[R(s,a,s') + \gamma\Phi(s') - \Phi(s) + \gamma \max_{a' \in \mathcal{A}}(Q^*_{\mathcal{M}}(s',a') - \Phi(s')]$$

We know define $\hat{Q}_{\mathcal{M}'}(s,a) = Q^*_{\mathcal{M}}(s,a) - \Phi(s)$ and substitute $\gamma\Phi(s') - \Phi(s)$ by $F(s,a,s')$.

$$\hat{Q}_{\mathcal{M}'}(s,a) = \mathbb{E}_{s'}[R(s,a,s') + F(s,a,s') + \gamma \max_{a' \in \mathcal{A}}(\hat{Q}_{\mathcal{M}'}(s',a')]$$

This result gives us the Bellman Equation for $\mathcal{M}'$, so $\hat{Q}_{\mathcal{M}'}(s,a)$ is the optimal Q-function of $\hat{Q}_{\mathcal{M}'}(s,a)$. Hence, $Q^*_{\mathcal{M}'}(s,a) = \hat{Q}_{\mathcal{M}'}(s,a) = Q^*_{\mathcal{M}}(s,a) - \Phi(s)$ and the optimal policy for $\mathcal{M}'$ exists. Then, we just have to prove that the actions taken by the policy following the Q-value $Q^*_{\mathcal{M}'}(s,a)$ are the same than $Q^*_{\mathcal{M}'}(s,a)$.

$$\begin{aligned}\pi^*_{\mathcal{M}'}(s) &\in \arg\max_{a \in \mathcal{A}} Q^*_{\mathcal{M}'}(s,a) \\ &= \arg\max_{a \in \mathcal{A}} Q^*_{\mathcal{M}}(s,a) - \Phi(s) \\ &= \arg\max_{a \in \mathcal{A}} Q^*_{\mathcal{M}}(s,a)\end{aligned}$$

So the exact same policy that is optimal in $\mathcal{M}'$ is optimal in $\mathcal{M}$. To show that every optimal policy in $\mathcal{M}$ is optimal in $\mathcal{M}'$ the same argument is used:

$$\begin{aligned}\pi^*_{\mathcal{M}}(s) &\in \arg\max_{a \in \mathcal{A}} Q^*_{\mathcal{M}}(s,a) \\ &= \arg\max_{a \in \mathcal{A}} Q^*_{\mathcal{M}'}(s,a) + \Phi(s) \\ &= \arg\max_{a \in \mathcal{A}} Q^*_{\mathcal{M}'}(s,a)\end{aligned}$$

Now we prove the necessity of the Theorem. In this proof we will consider the case where $|\mathcal{A}| = 2$, the general proof follows the same fashion. First we will state a Lemma which shows that the reward shaping function cannot depend of the action.

Lemma: If there exists $s, s' \in \mathcal{S}$ and $a, a' \in \mathcal{A}$ such that $F(s,a,s') \neq F(s,a',s')$ then there exists a set of transitions probabilities $P(\cdot|s,a)$ and a reward function R such that no optimal policy in $\mathcal{M}'$ is an optimal policy of $\mathcal{M}$.

Proof We assume without loss of generality that $F(s,a,s') > F(s,a',s')$ and we define $\Delta = F(s,a,s') - F(s,a',s')$. Then we construct the MDP as:

- $P(s'|s,a) = P(s'|s,a') = 1$
- $R(s,a,s') = 0$ and $R(s,a',s') = \frac{\Delta}{2}$

The obviously the optimal policy in state s is choosing action a', $a' = \arg\max \pi^*_{\mathcal{M}}(s)$.

However, if we consider the MDP given by $\mathcal{M}'$ we have that $R'(s,a,s') = F(s,a,s')$ and $R'(s,a',s') = \frac{\Delta}{2} + F(s,a',s') = -\frac{\Delta}{2} + F(s,a',s)$. Since $\Delta > 0$ we have that $R(s,a,s') > R(s,a',s')$. Therefore the optimal policy in $\mathcal{M}'$ has $a = \arg\max \pi^*_{\mathcal{M}'}(s)$. Thus, $\pi^*_{\mathcal{M}}(s) \neq \pi^*_{\mathcal{M}'}(s)$

Proof We assume that F is not a potential-based function and by the former Lemma we can assume that $F(s,a,s') = F(s,s')$. Then we consider the following MDP

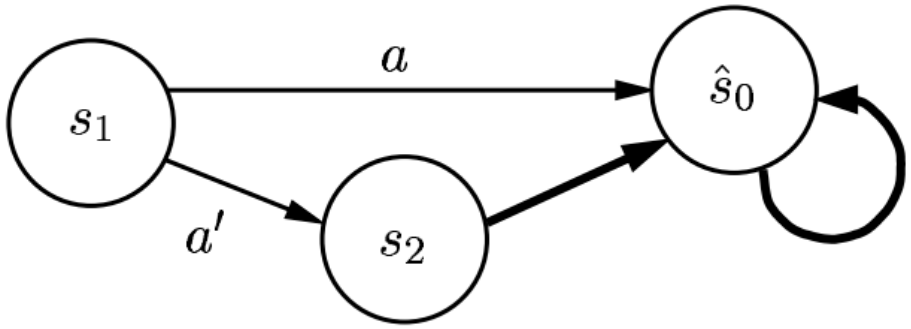

Figure 3.1: MDP. All the edges have probability 1.

Since if we shift all the rewards by a constant offset the MDP remains invariant we can consider without loss of generality that $F(\hat{s}_0, \hat{s}_0) = 0$. Let us define $\Phi(s) = -F(s, \hat{s}_0)$. Since F is not a potential-based function then $\gamma\Phi(s_2) - \Phi(s_1)! = F(s_1, s_2)$. We consider the MDP M the one given by 3.1.In the MDP of the Figure 3.1 we consider that all the edges have 0 reward except the edge $(s_1, a, \hat{s}_0)$ that has a reward of $\Delta/2$, where $\Delta = F(s_1, s_2) + \gamma F(s_2, \hat{s}_0) - F(s_1, \hat{s}_0)$. Then

Where he have considered that $V^*_M(\hat{s}_0) = V^*_{M'}(\hat{s}_0) = 0$. Then the optimal policy in this the MDP given by M and the one given by M′ differ because

$$\pi^*_M(s_1) = \begin{cases} a & \text{if} \quad \Delta > 0 \\ a' & \text{Otherwise} \end{cases}$$

$$\pi^*_{M'}(s_1) = \begin{cases} a' & \text{if} \quad \Delta > 0 \\ a & \text{Otherwise} \end{cases}$$

It is also proven that for every policy π, $Q^\pi_{M'}(s, a) = Q^\pi_M(s, a) - \Phi(s)$ and $V^\pi_{M'}(s) = V^\pi_M(s) - \Phi(s)$. This is interesting because it means that near-optimal policies are preserved, i.e., if $\|V^\pi_{M'}(s) - V^*_{M'}(s)\| < \epsilon \iff \|V^\pi_M(s) - V^*_M(s)\| < \epsilon$. Which makes the learned policy under reward shaping robust. Another important remark is the invariance of the potential-based functions in the reinforcement learning paradigm. Interestingly, if one has a potential-based function as unique reward then all policies are optimal in this MDP. Finally, the sufficiency result is proven for the discounted case, $\gamma < 1$. For the undiscounted case one needs to assume that it exists an absorbing state or final state s_T such that there is no more rewards after this state is visited and that all the

policies end up visiting the absorbing state (this condition is often referred as all policies are *proper*). Then the proof follows exactly the one presented using that the optimal policy is invariant with respect adding constants to the reward function and defining $\Phi = \Phi(s) - \Phi(s_T)$.

4
Policy gradients

In this chapter we will present the methods currently used reinforcement learning for determining the optimal policy [13, **?**]. The policy, which determines the probability of each action, is parametrized by a set of parameters θ. Our objective then is to maximize the expected return with respect those parameters:

$$\max_{\theta} J(\theta) = \mathbb{E}_{\tau}[\, R(\tau)]$$

Where τ is a trajectory and $\pi = \pi(a|s;\theta)$, which for ease of notation we will denote as $\pi_\theta(a|s)$.

So the main goal of policy optimization is to find the parameters $\theta \in \Theta \subseteq \mathcal{R}^m$ such that the return $J(\theta)$ (we make the abuse of notation $J(\theta) = J(\pi_\theta)$) is maximal. Policy gradient methods compute the gradient of the expected performance and take a small steep in the steepest direction. These methods have special importance in robotics, since we require that any change on the policy to be smooth. A significant step could jeopardize the integrity of the robot and the safety of the people around it. These methods also have strong theoretical guarantees, which other methods such as greedy value function based methods do not posses. Finally, one of the fundamental traits that has lead policy gradient methods to its current popularity is that they can be used in model-free reinforcement learning, i.e, they do not need any underlying model of the dynamics as we will see.

Policy gradient methods follow the update rule:

$$\theta_{n+1} = \Pi_\Theta(\theta_n + \alpha_n \hat{\nabla}_\theta J|_{\theta=\theta_n}) \tag{4.1}$$

where $\alpha_n \in \mathbb{R}^+$ denotes the learning rate and Π_Θ is the projection to the feasible region Θ . An important property of these methods is that if the gradient is unbiased, and if the learning rate satisfies

$$\sum_{n=0}^{\infty} \alpha_n = \infty \qquad \text{and} \qquad \sum_{n=0}^{\infty} \alpha_n^2 < \infty \tag{4.2}$$

then the algorithm is guaranteed to converge almost surely to at least a local minimum. Nevertheless, in practise it is used a fixed the step size to a sufficiently small value. If $\hat{\nabla}_\theta J$ is an unbiased estimator it is referred as being of the *Robbins-Monro* type. Then it has a asymptotic convergence rate of $n^{-\frac{1}{2}}$. However, the main difficulty of these methods is to compute a good estimator of the gradient. There traditional approaches to compute the gradient of $J(\theta)$ can be divided into *finite differences* and *vanilla policy gradients*.

4.1 Finite Differences

This method estimates the gradient of the expect reward $J(\theta)$ by simply using finite differences. This "naive" approach perturbs each component of θ while holding the other components to its reference value. However, since the output is stochastic if the perturbance is too small the estimation of the gradient could be very noisy. Therefore, there is a trade-off between bias and variance.

The differences can be one-sided or central. Central difference gives a more accurate gradient but there is the necessity of estimating the function $2m$ times, which can be a burden in high-dimensional systems or when the simulation is very costly. On the other hand, one-side differences just have the need to evaluate $m+1$ times our function. The ith component of the central difference estimator is given by

$$\frac{\hat{J}(\theta + \delta_i e_i) - \hat{J}(\theta - \delta_i e_i)}{2\delta_i} \tag{4.3}$$

where δ_i is the perturbation applied to the unit vector in the ith direction, e_i.

The advantage of this approach is that it does not require any knowledge about the of the policy or its differentiability.

This method is not applied in deep reinforcement learning setups due to the rewards functions are parametrized with neural networks, which have a huge number of parameters. Also, in order to estimate the expected reward we have to perform roll-outs of the agent and can be costly, especially in real-world applications. The convergence rate of this method highly depends on the uncertainties. In real systems the uncertainties tend to be high, which results in convergence rates between $O(n^{-\frac{1}{4}})$ and $(O(n^{-\frac{2}{5}})$.

4.2 Vanilla Policy Gradient

In the following, and as before, we will denote as τ the trajectory of the agent, that is a sequence of states and actions $\tau = \{s_0, a_0, s_1, a_1, \cdots, s_T\}$. The trajectory is generated by a roll-out, $\tau \sim p(\tau|\theta)$. Let $R(\tau)$ be the return of the trajectory τ, and $\mathbb{T}$ the set of all trajectories. Therefore, the total expected reward of our policy is

$$J(\theta) = \mathbb{E}_\tau[\mathbb{R}(\tau)] \ = \int_{\mathbb{T}} p(\tau|\theta) R(\tau) d\tau \tag{4.4}$$

And the gradient can be written as

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_\tau[\mathbb{R}(\tau)] \ = \nabla_\theta \int_{\mathbb{T}} p(\tau|\theta) R(\tau) d\tau = \int_{\mathbb{T}} \nabla_\theta p(\tau|\theta) R(\tau) d\tau = \int_{\mathbb{T}} \frac{p(\tau|\theta)}{p(\tau|\theta)} \nabla_\theta p(\tau|\theta) R(\tau) d\tau =$$

$$\int_{\mathbb{T}} p(\tau|\theta) \nabla_\theta \log(\ p(\tau|\theta))\ R(\tau) d\tau = \mathbb{E}_\tau[\ \nabla_\theta \log(\ p(\tau|\theta))\ R(\tau)] \tag{4.5}$$

Thus we can obtain the LHS of the equation above by making use of the Central Limit Theorem in the RHS. This trick is known as the likelihood ratio trick. More importantly, the $p(\tau|\theta)$ can be written as

$$p(\tau|\theta) = \rho(s_0) \Pi_{t=0}^{T-1} p(s_{t+1}|a_t, s_t) \pi_\theta(a_t|s_t) \tag{4.6}$$

then

$$\nabla_\theta \log(p(\tau|\theta)) = \sum_{t=0}^{T-1} \nabla_\theta \log(\pi_\theta(a_t|s_t)) \tag{4.7}$$

All the derivatives with respect the dynamics of the environment drop out as well as $\rho(s_0)$ (the initial state distribution). Which incredibly results in being able to maximize our reward without needing any knowledge of how the system behaves. Intuitively, what the gradient is just increasing the log-likelihood of the trajectories which led to higher rewards.

The gradient estimator is defined by

$$g_{RF} = \mathbb{E}_\tau[\ (\sum_{t=0}^{T-1} \nabla_\theta \log(\pi_\theta(a_t|s_t)))(\sum_{t'=0}^{T-1} \gamma^{t'} r_{t'})] \tag{4.8}$$

where $R(\tau) = \sum_{l=0}^{l=T-1} \gamma^l r_l$ and γ is the discount factor, is known as REINFORCE. This method is guaranteed to converge to the true gradient estimator in $O(n^{-\frac{1}{2}})$, where n is the number of samples. Also, just a single roll-out makes the gradient estimation unbiased, which reduces the number of roll-outs needed for performing a good update step.

The REINFORCE estimator, despite its fast convergence, has a high variance which is very troublesome in practice. Modifications in this formula are made to reduce the variance.

The policy gradient theorem (PGT) is based upon the fact that future actions does not affect past rewards, which can be formalized as:

$$\mathbb{E}_{s_{t+1:l}}[\,\nabla_\theta \log(\,\pi_\theta(a_l|s_l))\,r_{t'}\,] = 0 \tag{4.9}$$

Then we can write $\nabla_\theta J$ as

$$g_{PGT} = \mathbb{E}_\tau[\sum_{t=0}^{T-1}(\nabla_\theta \log(\pi_\theta(a_t|s_t)))(\sum_{t'=t}^{t'=T-1}\gamma^{t'-t}r_{t'})] \tag{4.10}$$

This gradient estimator is known as the policy gradient theorem. Reordering the summations we can write the G(PO)MDP estimator

$$g_{GMDP} = \mathbb{E}_\tau[\sum_{t'=0}^{T-1}(\sum_{t=0}^{t=t'}\nabla_\theta \log(\pi_\theta(a_t|s_t)))\,(\gamma^{t'}r_{t'})] \tag{4.11}$$

We will work with the former, as it is more convenient numerically implementation-wise.

To further reduce the variance of the gradient estimator we will use a *baseline* $b(s_t)$. A baseline is a term subtracted to the returns which leave the estimator unbiased and have the function to reduce its variance. Therefore, we will write the gradient estimator g_PGT as

$$g_{PGT} = \mathbb{E}_\tau[\sum_{t=0}^{T-1}(\nabla_\theta \log(\pi_\theta(a_t|s_t)))(\sum_{t'=t}^{t'=T-1}\gamma^{t'-t}r_{t'} - b(s_t))] \tag{4.12}$$

This equality is for any arbitrary function $b(s_t)$ that just depend on the state s_t. One advantage of using the policy gradient estimator in front of G(PO)MDP and is that the baseline in the former can be dependent on the state while the latter it is just time dependent. Now we prove that adding a baseline leaves the estimator unbiased.

Proof

$$\begin{aligned}
g_{PGT} &= \mathbb{E}_\tau[\,(\nabla_\theta \log(\pi_\theta(a_t|s_t)))b(s_t))] \\
&\quad \mathbb{E}_{s_{0:t},a_{0:(t-1)}}[\,\mathbb{E}_{s_{(t+1):T},a_{t:(T-1)}}[\,(\nabla_\theta \log(\pi_\theta(a_t|s_t)))b(s_t)]\,] \\
&\quad \mathbb{E}_{s_{0:t},a_{0:(t-1)}}[\,b(s_t)\mathbb{E}_{s_{(t+1):T},a_{t:(T-1)}}[\,(\nabla_\theta \log(\pi_\theta(a_t|s_t)))]\,] \\
&\quad\quad \mathbb{E}_{s_{0:t},a_{0:(t-1)}}[\,b(s_t)\mathbb{E}_{a_t}[\,(\nabla_\theta \log(\pi_\theta(a_t|s_t)))]\,] \\
&\quad\quad \mathbb{E}_{s_{0:t},a_{0:(t-1)}}[\,b(s_t)\int_{\mathcal{A}}\nabla_\theta \log(\pi_\theta(a_t|s_t))\pi_\theta(a_t|s_t)] \\
&\quad\quad\quad \mathbb{E}_{s_{0:t},a_{0:(t-1)}}[\,b(s_t)\int_{\mathcal{A}}\nabla_\theta \pi_\theta(a_t|s_t)] \\
&\quad\quad\quad \mathbb{E}_{s_{0:t},a_{0:(t-1)}}[\,b(s_t)\nabla_\theta\int_{\mathcal{A}}\pi_\theta(a_t|s_t)] \\
&\quad\quad\quad\quad \mathbb{E}_{s_{0:t},a_{0:(t-1)}}[\,b(s_t)\nabla_\theta(1)] \\
&\quad\quad\quad\quad\quad \mathbb{E}_{s_{0:t},a_{0:(t-1)}}[\,b(s_t)\cdot 0] = 0
\end{aligned}$$

Which proves that adding a baseline which just depends on the state does not modify the estimate of the gradient.

4.3 Optimal Baselines

Once presented the concept of baseline for variance reduction, it is interesting to see which are the baselines that lead to a minimum variance. An optimal baseline is the one that minimizes the variance of each component of the gradient g_j $\sigma_j^2 = Var(g_j)$ without biasing it.

Here we derive the optimal baseline for G(PO)MDP, recall that the baseline in the G(PO)MDP is time dependent. Therefore, we want $b(s)$ that satisfies

$$\min_{\{b_k^j\}_{k=0}^T} \sigma_j^2 \text{ s.t.} \mathbb{E}_\tau[\, g_j^2] \,= \nabla_{\theta_j} J \tag{4.13}$$

Where b_k^j denotes the baseline for the jth component of the gradient and for the kth time step. We can express the variance as $\sigma_j^2 = \mathbb{E}[\, g_j^2] \, - \mathbb{E}[\, g_j^2]^{\,2}$, and since $\mathbb{E}[\, g_j]^{\,2}$ does not depend on the choice of the baseline the minimization problem can be expressed as

$$\min_{\{b_k^j\}_{k=0}^T} \mathbb{E}_\tau[\, g_j^2] \,\geq \mathbb{E}_\tau[\, \min_{\{b_k^j\}_{k=0}^T} g_j^2] \tag{4.14}$$

Where the inequality is due to the Jensen Inequality. Deriving with respect b_k^j and forcing the derivative equal to 0, we end up with:

$$b_k^j = \frac{\mathbb{E}_\tau[\, (\sum_{t=0}^{T-1} \nabla_{\theta_j} \log(\pi_\theta(a_t|s_t)))^2 \gamma^t r_t]}{\mathbb{E}_\tau[\, (\sum_{t=0}^{T-1} \nabla_{\theta_j} \log(\pi_\theta(a_t|s_t)))^2]} \tag{4.15}$$

For computing the baselines the expectations are approximated with data from the roll-outs. Even thought, G(PO)MDP and PGT are numerically equivalent people in practice use PGT. This estimate of the algorithm allows the baseline to be state dependent. The drawback is that there is not an analytic solution for the optimal state dependent baseline. Conversely, it is used a near-optimal baseline, the Value function $V^\pi(s_t)$,

$$V^\pi(s) = \mathbb{E}_\tau[\, \sum_{t'=t}^{T} r_{t'} | s_t = s, a_{t'} \sim \pi(\cdot|s_t)]$$

So we will choose a baseline that approximates the Value function in practice, which usually is either a linear combination of features or a neural network. Using the Value function as baseline is a very intuitive and natural choice: suppose we perform a roll-out of the trajectory and collect a noisy estimate

$$\hat{g} = \sum_{t=0}^{T-1} (\nabla_\theta \log(\pi_\theta(a_t|s_t)))(\sum_{t'=t}^{t'=T-1} \gamma^{t'-t} r_{t'}) \tag{4.16}$$

which we will use to update our policy with the rule $\theta \leftarrow \theta - \alpha\hat{g}$. If we were using PGT without a baseline then this update rule increases the probability of the action proportionally to $(\sum_{t'=t}^{t'=T-1} \gamma^{t'-t} r_{t'}$, which are all the rewards received following the action. However, this update will increase the probability of all actions, which makes the update rule noisy. Using the Value function as baseline we get an estimate of how much better the action was than expected. Therefore, we increase the log-likelihood of the actions that lead to better results than we expect and *decrease* the log-likelihood of those that lead to worse sum of rewards. The difference $\sum_{t'=t}^{t'=T-1} \gamma^{t'-t} r_{t'} - V^\pi(s_t)$ is called *Advantage* and is an estimate of the goodness of the action a_t with respect the other ones at state s_t.

It is common to consider that the true MDP that you want to solve is the undiscounted MDP, i.e., $\gamma = 1$. However, the undiscounted case leads to gradient estimates with excessive variance in long episodes (T big). Then γ is introduced as variance reduction parameter. Even though, doing so has the cost of biasing your gradient estimate – trade-off bias-variance – the bias introduced is not very harmful. Intuitively, it considers that the current action a_t has no effect in distant rewards. This assumption is completely valid in episodes were there is no delay on the rewards (it can be harmful in sparse settings where you get the rewards just if you complete a task). By adding the terms $1, \gamma, \gamma^2, \cdots$ we have an effective time horizon of $\frac{1-\gamma^{T+1}}{1-\gamma}$.

Algorithm 1 'Vanilla' Policy gradient

Initialize policy parameters θ and baseline b_0
for i=0,1,$\cdots$
  Perform a set of roll-outs with N trajectories
  for $t = 0, \ldots, T$
    Compute the return $R(s_t) = \sum_{t'=t}^{T} \gamma^{t'-t} r_t$
    Compute the advantage $\hat{A}(s_t) = R(s_t) - b(s_t)$
  Fit the baseline minimizing: $\frac{1}{N} \sum_{n=0}^{N} \sum_{t=0}^{T} R(s_t) - b(s_t)^2$
  Compute the gradient estimate $\hat{g} = \frac{1}{N} \sum_{n=0}^{N} \sum_{t=0}^{T-1} (\nabla_\theta \log(\pi_\theta(a_t|s_t)))\hat{A}$
  $\theta \leftarrow \theta + \alpha\hat{g}$

4.4 Natural Policy Gradient

The standard gradient rule presented in the previous section is nonconvariant, does not have consistent dimensions. The update rule $\theta_j + \alpha \partial J \partial \theta_j$ is dimensionally inconsistent, since $\partial J \partial \theta_j$ has units of $\frac{1}{\theta_j}$ and all θ_j does not need to have the same dimensions [14].

We need to compute the steepest ascent direction, $d\theta$, which maximizes $J(\theta)$ under the constraint that $d\theta^2$ is small, so the linear approximation is still valid. The norm is defined in a differential manifold that has a first fundamental form $G(\theta)$, $d\theta^2 = d\theta^T G(\theta) d\theta$. Then the steepest descent direction in this manifold is given by $G(\theta)^{-1}\nabla J(\theta)$. The method presented in the former section considers that the manifold of the parameters is $\mathbb{R}^m$, and therefore the first fundamental form is the identity matrix. However, this is not a sensible choice and it is better to use the coordinates of the manifold that θ parametrize.

The natural gradient uses the Fisher information matrix (FIM) as first fundamental form. The distribution $\pi_t heta(\cdot|s)$ defines a manifold for each state s that has coordinates θ ($\pi_t heta(a|s)$ is a point on this manifold). Then the Fisher information matrix of the distribution $\pi_t heta(\cdot|s)$ is written as

$$F_s(\theta)_{ij} = \mathbb{E}_{\pi_\theta(\cdot|s)}[\frac{\partial log(\pi_\theta(a|s))}{\partial \theta_i}\frac{\partial log(\pi_\theta(a|s))}{\partial \theta_j}] \tag{4.17}$$

Under certain regularity conditions the FIM can also be computed by

$$F_s(\theta)_{ij} = \mathbb{E}_{\pi_\theta(\cdot|s)}[\frac{\partial^2 log(\pi_\theta(a|s))}{\partial \theta^2}] \tag{4.18}$$

However, this last form is not usually used in practice due to the excessive cost of computing second derivatives. The FIM defines the same distance between two points regardless the choice of coordinates used. This invariance on the metric is defined up to a scale. Obviously, if our manifold is parametrized by the same parametrization times a scalar λ the fundamental form will be λ^2 times the original, but the surface will remain the same. This property does not hold if we use $G = I$ since the manifold changes with the choice of coordinates θ.

Lets now assume that every policy π_θ is ergodic, meaning that it has a well defined stationary distribution over the states $\mu^\pi(s)$. Then we can define our metric in the manifold as the expectation of the FIM over all the states given by this distribution Âą

$$G(\theta) = \mathbb{E}_{\mu^\pi(s)}[F_s(\theta)] \tag{4.19}$$

The metric given by Equation 4.18 defines a distance on the probability manifold for each state s and $F(\theta)$ is the average of all these distances. It is worth noticing that even thought $F_s(\theta)$ does not depend on the transitions probability of the MDP, $F(\theta)$ does. Since $\mu^\pi(s)$ is defined by

$$\mu_0^\pi(s) = \rho(s)$$

$$\mu_t^\pi(s) = \int_{s_t \in \mathcal{S}} \int_{a_t \in \mathcal{A}} p(s|a_t, s_t)\pi_\theta(a_t|s_t)\mu_{t-1}^\pi(s) da_t ds_t$$

$$\mu^\pi(s) = \sum_{t=0}^{T} \tag{4.20}$$

where $\rho(s)$ is the initial state probability distribution of the MDP with time horizon T. The steepest descent direction of the natural policy gradient is

$$g(\theta) = F(\theta)^{-1}\nabla J(\theta) \tag{4.21}$$

The natural gradient is the key ingredient for the results that policy gradient methods have develop in the recent years. In particular, the next method we explain (the one used in our work) is based upon the natural gradient.

5
Trust Region Policy Optimization

Trust Region Policy Optimization (TRPO) is an iterative algorithm that guarantees monotonic improvement of the policy over iterations [15, 8]. This algorithm is derived from the policy gradients explained in the former section applying a hard constraint on the KL-divergence between the current policy and the new one. The algorithm is suitable for policies parametrized with a huge number of parameters (such as neural networks). Our experiments are carried out using TRPO, due to it has proven impressive results in many complex environments such as walking humanoid and Atari. In the following we describe the derivation of the algorithm.

We define the advantage function $A^{\pi}(s, a)$ as

$$A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s) \tag{5.1}$$

The advantage function tell us the goodness of an action compared with the mean of all possible ones in that specific state. The identity given by [16] let us express the total return of another policy $\tilde{\pi}$ in terms of the policy π and its advantage function.

$$J(\tilde{\pi}) = J(\pi) + \mathbb{E}_{\tilde{\pi}}[\sum_{t=0}^{T} \gamma^t A^{\pi}(s_t, a_t)] \tag{5.2}$$

Proof First we note that

$$A^{\pi}(s, a) = \mathbb{E}_{s' \sim P(s'|s,a)}[R + \gamma V^{\pi}(s') - V^{\pi}(s)]$$

Hence, we can rewrite the expected sum of advantage functions as

$$\begin{aligned}\mathbb{E}_{\tilde{\pi}}[\sum_{t=0}^{T}\gamma^t A^{\pi}(s_t,a_t)] &= \mathbb{E}_{\tilde{\pi}}[\sum_{t=0}^{\infty}\gamma^t(R_t+\gamma V^{\pi}(s_{t+1})-V^{\pi}(s_t))]\\ &= \mathbb{E}_{\tilde{\pi}}[-V^{\pi}(s_0)+\sum_{t=0}^{\infty}\gamma^t R_t]\\ &= -\mathbb{E}_{s_0}[V^{\pi}(s_0)]+\mathbb{E}_{\tilde{\pi}}[\sum_{t=0}^{\infty}\gamma^t R_t]\\ &= -J(\tilde{\pi})+J(\pi)\end{aligned}$$

We define $\mu^{\pi}(s)$ as the unnormalized discounted visitation frequency

$$\rho_{\pi}(s)=\sum_{t=0}^{\infty}\gamma^t P(s=s_t) \tag{5.3}$$

where $s_0 \sim \rho_0$ and the actions are sampled according to π . Then rewriting the former equation

$$J(\tilde{\pi})=J(\pi)+\sum_{s}\rho_{\tilde{\pi}}\sum_{a}\tilde{\pi}(a|s)A^{\pi}(s,a) \tag{5.4}$$

Hence, an update of the policy π to $\tilde{\pi}$ which has non-negative advantage function almost everywhere in every state s will improve the expected total return. This makes the classical result of deterministic policy optimization $\tilde{\pi} = arg\max_a A^{\pi}(s,a)$ a valid algorithm that converge to the optimal policy when there is no positive advantage function in an state where the visitation frequency is positive. However, this equation is intractable due to the complex dependency on the visiting frequency.A linear approximation is used instead

$$L_{\pi}(\tilde{\pi})=J(\pi)+\sum_{s}\rho_{\pi}\sum_{a}\tilde{\pi}(a|s)A^{\pi}(s,a) \tag{5.5}$$

Instead of using the visitation frequency of the updated policy $\tilde{\pi}$ we are using π. Since 5.5 is a linear approximation of $J(\tilde{\pi})$ (when the policy is a differentiable function of a parameter vector θ) if a small step improves L it will also improve J. However, we do not know which is the maximum step-size where this approximation is valid and an improvement on L leads to an improvement on J.

[16] proved that if we set as updated policy as the mixture

$$\tilde{\pi}=(1-\alpha)\pi+\alpha\pi' \tag{5.6}$$

being $\pi' = arg\min_{\pi'} L_{\pi}(\pi')$, then the following inequality is satisfied

$$J(\tilde{\pi}) \geq L_{\pi}(\tilde{\pi})-\frac{2\epsilon\gamma}{(1-\gamma(1-\alpha))(1-\gamma)}\alpha^2 \tag{5.7}$$

ϵ is defined as $\epsilon = \max_s |\mathbb{E}_{\pi'}[A^{\pi}(s,a)]$. Note that this bound is only valid when the updated policy is a mixture, which is not desirable in many cases since it is restrictive and intractable (such as when computing the KL-divergence). It would be convenient to have a similar updated scheme where we could use a more general class of stochastic policies.

Theorem Let $\alpha = D_{TV}^{max}(\tilde{\pi}, \pi)$ then Equation 5.7 holds.

D_{TV} is the total variation distance that is defined as $D_{TV}(p,q) = \frac{1}{2}\sum_i |q_i - p_i|$ for discrete distributions and $D_{TV}(p,q) = \frac{1}{2}\int |q(x) - p(x)|dx$ for continuous ones. We define $D_{TV}^{max}(\pi, \tilde{\pi}) = \max_s D_{TV}(\pi(\cdot|s), \tilde{\pi}(\cdot|s)$

Interestingly we can use the KL-divergence (more tractable) instead of the total variation distance to constrain the distance from the two distributions. It is known that $D_{KL}(p,q) \geq D_{TV}(p,q)^2$. Then if $D_{KL}^{max}(\pi, \tilde{\pi}) = max_s D_{KL}(\pi, \tilde{\pi})$ we have the following inequality

$$J(\tilde{\pi}) \geq L_{\pi}(\tilde{\pi}) - \frac{2\epsilon\gamma}{(1-\gamma)^2} D_{KL}^{max}(\pi, \tilde{\pi}) \tag{5.8}$$

This is a slightly softer constrain than the one presented in 5.7. It follows immediately from the fact that $\alpha, \gamma \in [0,1]$. This bound is weaker when $\alpha \approx 1$.

Maximizing iteratively the term $L_{\pi}(\tilde{\pi}) - CD_{KL}^{max}(\pi, \tilde{\pi})$, where $C = \frac{2\epsilon\gamma}{(1-\gamma)^2}$ gives an algorithm that improves monotonically the total expected return $J(\pi)$. This is easily proven. Let's denote $M_i(\pi_{i+1}) = L_{\pi}(\pi_{i+1}) - CD_{KL}^{max}(\pi_i, \pi_{i+1})$. Then,

$$\begin{aligned} J(\pi_{i+1}) &\geq M_i(\pi_{i+1}) \\ J(\pi_i) &= M_i(\pi_i) \\ J(\pi_{i+1}) - J(\pi) &\geq M_i(\pi_{i+1}) - M_i(\pi_i) \end{aligned}$$

So maximizing M_i induces the maximization of the true objective function J.

However, this algorithm is not practical since it implies the computation of the exact advantage functions. Also, the steps performed while maximizing M are often too small leading to slow convergence. We can take bigger steps if we instead impose a constraint on the KL-divergence between the policies, hence a trust region constraint:

$$\begin{aligned} &\max_{\tilde{\theta}} L_{\pi_\theta}(\pi_{\tilde{\theta}}) \\ &\text{subject to: } D_{KL}^{max}(\pi_\theta, \pi_{\tilde{\theta}}) \leq \delta \end{aligned} \tag{5.9}$$

It is implied that the policies π are parametrized by the vector θ. This maximization problem is sounded given the theory, but it is very hard in practice since it bounds every point in the state space. We modify it using an heuristic approximation on the KL-constraint

$$D_{KL}^{\rho}(\theta, \tilde{\theta}) = \mathbb{E}_{s\sim\rho}[D_{KL}(\pi_\theta(\cdot|s), \pi_{\tilde{\theta}}(\cdot|s))] \tag{5.10}$$

Then Trust Region Policy Optimization solves the following optimization problem

$$\begin{aligned} &\max_{\tilde{\theta}} L_{\pi_\theta}(\pi_{\tilde{\theta}}) \\ &\text{subject to: } D_{KL}^{\rho_\theta}(\theta, \tilde{\theta}) \leq \delta \end{aligned} \tag{5.11}$$

5.1 Practical Algorithm

We desire to solve the optimization problem presented in equation 5.11. Expanding its terms we obtain

$$\begin{aligned} &\max_{\tilde{\theta}} \sum_s \rho_\theta(s) \sum_a \pi_{\tilde{\theta}}(s|a) A_\theta(s,a) \\ &\text{subject to: } D_{KL}^{\rho_\theta}(\theta, \tilde{\theta}) \leq \delta \end{aligned} \tag{5.12}$$

In order to solve it, we replace the term $\sum_s \rho(s)$ by an expectation – where the samples are taken from $\rho(s)$. Also, the advantage function is replaced by the Q-values given by Q_θ, which differs from the original function just by a constant. Finally, the term $\sum_a \pi_\theta(s|a)$ is modified using an importance sampling estimator, where we sample the actions from a distribution $q(a|s)$ instead of $\pi_\theta(s|a)$ to compute the expectation.

The result is we actually solve the optimization problem given by

$$\begin{aligned} &max_{\tilde{\theta}} \mathbb{E}_{s\sim\rho_\theta, a\sim q}[\frac{\pi_{\tilde{\theta}}(s|a)}{q(s|a)} Q_\theta(s,a)] \\ &\text{subject to: } \mathbb{E}_{s\sim\rho_\theta}[D_{KL}(\theta, \tilde{\theta})] \leq \delta \end{aligned} \tag{5.13}$$

The expectation is computed averaging the samples and the Q-values with an empirical estimate. Usually we parametrize the Q-function as a neural network and fit it with the Monte-Carlo estimates. And, in order to solve the constraint term of the optimization problem, we use the conjugate gradient algorithm and later on a line search.

We use as a gradient the natural gradient, so it is needed to solve the linear system $Fg = \nabla L$, where F is the Fisher Information Matrix. In large scale problems (such as neural networks, that have millions of parameters) it is intractable to invert this matrix. Thus, thanks to a cheap computation of the hessian-vector product (see [8] Appendix C.1) it is used conjugate gradients to solve the linear system.

Once the gradient direction is computed, we need to ensure that the KL-divergence constraint is satisfied, i.e, we need β such as $D_{KL}^{\rho_\theta}(\theta, \theta + \beta g) \leq \delta$. We approximate the KL-divergence with its quadratically form, $D_{KL}^{\rho_\theta}(\theta, \theta + \beta g) \approx \frac{1}{2}\beta^2 g^T F g$ and force this term equal to δ. Hence, we end up obtaining $\beta = \sqrt{\frac{2\delta}{g^T F g}}$

Since we have used a quadratic approximation of the KL-divergence to compute β it is not guaranteed that the original constraint is fully satisfied, and also, perhaps, the step given by δ is too big and results in a worsened expected total return. Therefore, a line-search is applied in order to not only ensure the constraint is fulfilled, but also ensure policy improvement. Starting from the parameter β obtained we shrink it exponentially until the policy has improved and the constraint is satisfied.

6
Policy Transfer via Modularity

6.1 Introduction

Non-prehensile manipulation, such as pushing, can be used by robots to move or to rearrange objects. Compared to grasping, pushing may be easier or even necessary in certain cases. For example, pushing is especially important for moving objects that are too big or too heavy to grasp [17]. Pushing can also be used to move multiple objects at once, such as when clearing space on a cluttered table, in which the robot can sweep multiple objects out of the way in a single stroke [18]. Additionally, if a robot needs to grasp an object, it may need to first push the object to a more ideal position or orientation before grasping [19, 20, 21].

However, pushing is a challenging task for robots due to frictional forces that are difficult to model [22, 23, 24, 25]. Many of the assumptions that are commonly made for analyzing pushing motions have been shown to not always hold in practice. For example, the frictional forces involved can be non-uniform, time-varying, anisotropic, and deviating in other ways from the assumptions that are usually made in pushing models [26].

In this work, we explore the use of reinforcement learning for robot pushing tasks. Reinforcement learning has shown to be a powerful technique for environments with unknown dynamics or for tasks with complex dynamics that are difficult to explicitly optimize over. Reinforcement learning has been used for tasks such as learning to dress, swing a bat, perform locomotion, stand up from a sitting position, drive a car, and many other tasks [27, 28, 8, 9, 29, 30, 31].

However, reinforcement learning methods, especially those using a neural network to represent the policy, can often be difficult to use in the real world due to the large number of samples required for learning. Some approaches have been to reduce the dimensionality of the exploration space and explore more significant directions [32]. This problem is especially important for deep policy gradient methods; such methods have demonstrated impressive results in simulation, but their use in the real world is limited by their large sample complexity [33, 8, 9, 30]. Based on the impressive results that these methods have demonstrated in simulation, we are motivated to explore whether such methods can also be made to work in the real world.

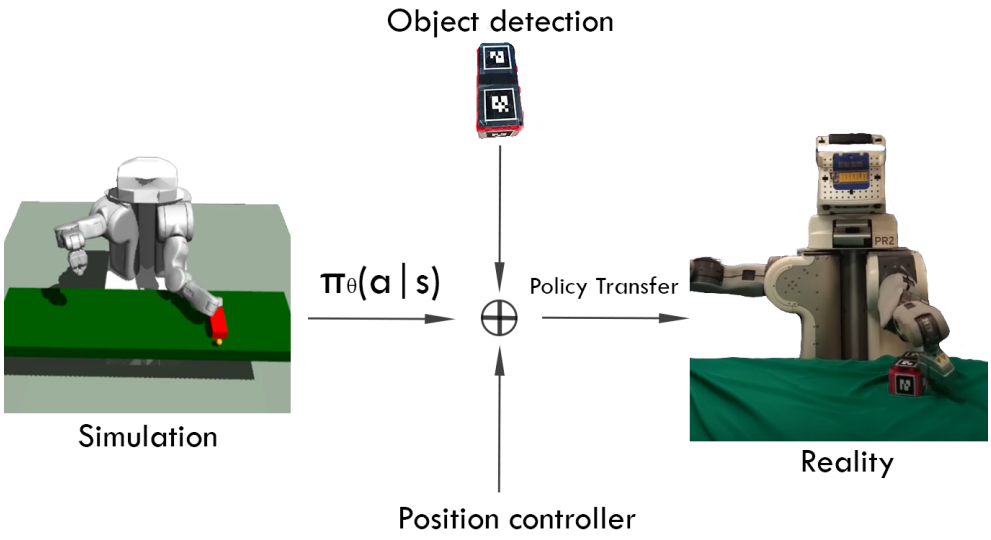

Figure 6.1: After training a policy to perform a pushing task in the simulator, we transfer the policy directly to the real world. We decompose the system into the modules of a pose estimator, a policy, and a position controller. This decomposition leads to a robust transfer of the policy from simulation to the real world.

We propose an approach for dealing with the sampling complexity by training a policy in simulation, where many training samples can be quickly generated. We then transfer the trained policy into the real world, without further training. However, such an approach is challenging due to mismatch between the simulation and the real world: images in simulation differ from images in the real world, and the system dynamics in simulation can differ from the dynamics in the real world.

We show that deep reinforcement learning methods can be trained in simulation and then transferred to the real world by using the principle of modularity. In contrast to most deep reinforcement learning approaches which they train the system end-to-end (e.g. from pixels to torques) [8, 30, 34], we break the problem into multiple separate pieces. Our system has a module that maps from image inputs to object pose, from object pose to target joint positions, and from target joint positions to motor torques. As shown in Figure 6.1, we train only the middle module in simulation and then transfer this module to the real-world. The surrounding modules are designed to enable ease of transfer, and are different in simulation and in real world.

Further, we show that we can incorporate our *prior knowledge* about the task into the system through modifications to the state space and to the reward function. This is again in contrast to the traditional deep reinforcement learning approach in which minimal prior knowledge of the robot task is incorporated into the learning procedure [8, 9, 29, 30].

Finally, we formulate a new approach for varying the reward function over

iterations of the optimization with asymptotic convergence to the original MDP, which we refer to as "reward guiding". We show that our approach leads to faster convergence (compared to other approaches such as reward shaping) without modifying the optimal policy asymptotically. Our simulation and real-world experiments illustrate the success of our approach for robot pushing tasks.

6.2 Related Work

Non-prehensile Manipulation. Pushing, a form of non-prehensile manipulation, has been studied in robotics for many years. An early analysis showed that one can compute the direction of rotation of the pushed object, based on the direction of the friction cone and the pushing force compared to the location of the center of friction [19]. Others have modeled frictional forces using the notion of a limit surface [22, 23, 24, 25].

However, all these approaches make a number of assumptions about the friction that recent experiments have shown do not always hold in practice [26]. For example, a recent study has shown that, for certain materials, the friction of an object can vary greatly over the object surface. The friction can also vary over time (it becomes lower as the object is repeatedly pushed and the surface is smoothed) and it can vary based on the object speed and the direction of pushing [26]. This study showed that most of the models that are normally used for estimating friction, such as the principle of maximum-power inequality [22] and the ellipsoidal approximation of a limit surface [24], do not always hold in practice, resulting in pushing motions that differ from those predicted by these models.

Learning a Dynamics Model. Some previous efforts have been made to deal with the uncertainties in friction by learning how to push objects [35, 36, 37, 38, 39, 34]. These approaches typically involve first learning a dynamics model for how an object will respond when pushed, and then choosing actions based on this dynamics model. However, as discussed above, the dynamics model for pushing can itself be fairly complex, sometimes involving a non-uniform friction distribution over the object surface [26]. Thus, learning an accurate dynamics model is itself a challenge, and once the dynamics model is learned, finding an optimal policy for pushing can be similarly complex. In practice, a number of approximations are usually made to the dynamics model to make it easier to learn and to optimize over (such as assuming linear-Gaussian dynamics as a function of the state [34]), but these approximations might lead to a reduced task accuracy [40]. In contrast, we explore whether we can directly learn a policy for pushing using reinforcement learning.

Policy Transfer. Due to the long training times required to learn complex policies using reinforcement learning algorithms, we explore whether we can train such policies in simulation and then transfer the trained policies to the real world. Most of previous work for policy transfer involves policies that do not involve object interaction [41, 42, 43, 44]. We explore whether we can transfer policies that use non-prehensile manipulation (i.e. pushing).

Furthermore, many transfer learning techniques require further training in the real-world to adapt the policy to the real-world dynamics [45]. However, such an approach does not work well when training complex policies represented by neural networks trained using policy gradient reinforcement learning algorithms, which require many samples for the policy to converge. We explore whether we can train policies in simulation and transfer them to the real-world directly, without further time-consuming fine-tuning required.

6.3 Methods

6.3.1 Overview

Our approach to object pushing aims at training a policy in simulation and transferring it to the real-world. The task that the robot has to perform consists of pushing a block placed on a table, starting from any initial position and orientation, to a certain goal position (for this work, we are not concerned with the final orientation of the block).

In order to train the policy quickly in simulation, we explore how we can incorporate prior knowledge into the learning algorithm by modifying the input state and the reward function. This is in contrast to many current approaches for deep reinforcement learning which attempt to train the system end-to-end, from pixels to torques, without any prior knowledge of the task. Additionally, we explore using the idea of modularity to enable the policy to easily transfer from simulation to the real-world. These ideas will be discussed in further detail below.

6.3.2 Modularity

Rather than learn a policy end-to-end, from pixels to torques, we use the idea of modularity to decompose the policy into multiple pieces, as shown in Figure 6.2. Each piece can be designed separately and then composed together to obtain the entire system. We show that decomposing the policy in this manner allows us to easily transfer a policy trained in simulation to the real world.

Image to Object Pose

The input to our system is an image containing the object to be pushed. In order to push this object correctly, we need to obtain the object position. To achieve this, we use AR tags, which can be easily identified and used to determine the object pose. An alternative approach would be to train an object pose detector, using one of various image algorithms for such tasks [46, 47, 48, 49]. When we train the policy in simulation, we directly use the ground-truth object position, obtained via our simulator.

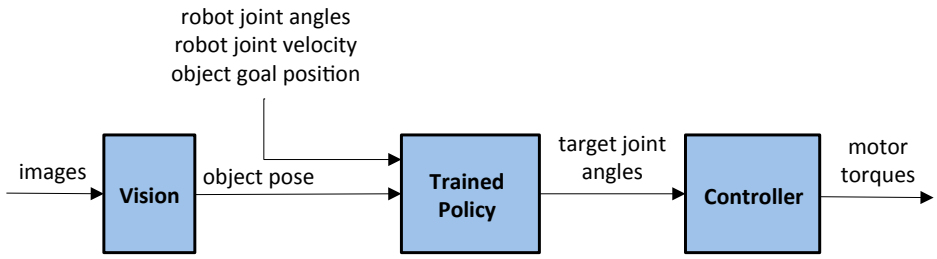

Figure 6.2: Our system for robot pushing. The center module ("Trained Policy") is trained in simulation and transferred to the real world. The other modules are designed to create the desired inputs and outputs for ease of transfer.

Separating the pose estimation from the rest of the system is crucial for transferring policies from simulation to the real world. The images rendered in simulation have a very different appearance than the images from the real world. Although some efforts have been made to increase the appearance variation of the simulated images [50], such an approach is still somewhat fragile, as differences between the simulated and real-world images can still lead to unexpected behavior.

Object Pose to Robot Joint Angles

Next, we learn a policy that maps from the object pose to the target robot position. We learn this policy entirely in simulation. The policy is trained using a reinforcement learning algorithm (TRPO with GAE) [8, 9], which allows the robot to learn a fairly complex policy for object pushing. However, because this policy is separated from the raw image inputs and the raw torque outputs by the surrounding modules (see Figure 6.2), the policy easily transfers from simulation to the real-world without any fine-tuning required.

Robot Joint Angles to Robot Torque

Rather than outputting the raw torques, the policy that we train in simulation outputs the target robot joint angles. We then use a PD controller to map from the target position to the robot torques. A similar approach has been used for reaching, leaning, balancing, and other tasks that do not involve object interaction [41, 42, 43, 44]. We demonstrate that such an approach can also be used for tasks that involve object manipulation, specifically non-prehensile manipulation (pushing).

6.3.3 Input State

A common approach for deep reinforcement learning is to train a policy using the minimal information necessary to perform the task. For example, the minimal information required for the robot pushing task is the robot joint

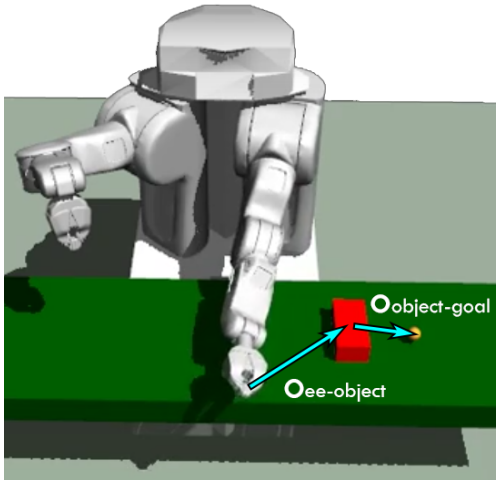

Figure 6.3: We add additional inputs to our observation based on the vector from the end-effector to the object, $o_{ee-object}$, and the vector from the object to the goal, $o_{object-goal}$.

angles and velocities; the object position, orientation, and velocity; and the goal position (i.e. the location to which the robot must push the object). Using only this information and sufficient training time, the robot is able to learn a policy to successively achieve the task.

However, for many robotics tasks, the algorithm designer might have prior knowledge about how the task should be performed. For example, in the case of object pushing, it is clear that the robot must first move its arm towards the object, and then it must move the object towards the goal. We thus simplify the task faced by the learning algorithm by adding additional features to our state-space: the three-dimensional vector from the robot end-effector to the object, and the three-dimensional vector from the object to the goal position, for a total of six additional features (See Figure 6.3). We show that adding these additional features significantly speeds up the time required to learn an optimal policy.

6.3.4 Reward Function

The goal of the robot is to push the object to the goal location. Thus, the minimal reward function necessary to complete this task is the negative distance from the object to the goal. However, the robot will have a difficult time learning a policy under this reward function, since most random actions that the robot takes will not have any affect on this reward (since most random actions will not cause the robot to interact with the object, and hence will not affect the distance between the object and the goal).

Thus, in order to decrease the training time, we add additional terms to the reward function to guide the policy in a direction to maximize the objective. For

example, we add a term to the reward corresponding to the negative distance from the robot end-effector to the object, to encourage the robot to move its end-effector near to the object. We also add a term based on the angle between the end-effector, the object, and the goal, to encourage the robot to move its end-effector such that these three points lie on a straight line; specifically, we add $\cos(o_{ee-object}, o_{object-goal})$, where $o_{ee-object}$ is the vector from the robot end-effector to the object and $o_{object-goal}$ is the vector from the object to the goal.

6.3.5 Reward Guiding

Modifying the reward function by adding additional terms can help to decrease the time to convergence, but it has the downside of modifying the learned policy. It has been shown that, in order for the optimal policy to be unchanged, then the reward function can only be modified in the following way:

$$r(s, a, s') = r_0(s, a, s') + \gamma\phi(s') - \phi(s) \tag{6.1}$$

where $r(s, a, s')$ is the new reward function when transitioning from state s to state s' after taking action a, $r_0(s, a, s')$ is the original reward function, γ is the discount factor, and $\phi(s)$ is a real-valued function over states [12].

Although such a modification has been shown to not affect the optimal policy, unfortunately it also does not always speed up the training time as much as one would like. The reason for this is simple: the effect of modifying the reward function in this way is equivalent to simply subtracting a baseline [51]. We can see this if we accumulate the reward function to compute the return. The original, unmodified return is given by

$$R_0 = \sum_{k=0}^{\infty} \gamma^k r_{0,k} \tag{6.2}$$

where $r_{0,k}$ is the unmodified reward r_0 received on the kth timestep. Then the modified return is given by [51]

$$\begin{aligned} R' &= \sum_{k=0}^{\infty} \gamma^k (r_{0,k} + \gamma\phi(s_{k+1}) - \phi(s_k)) && (6.3)\\ &= \sum_{k=0}^{\infty} \gamma^i r_{0,k} + \sum_{k=0}^{\infty} \gamma^{k+1}\phi(s_{k+1}) - \sum_{k=0}^{\infty} \gamma^k \phi(s_k)) && (6.4)\\ &= R_0 + \sum_{i=1}^{\infty} \gamma^k \phi(s_k) - \left(\phi(s_0) + \sum_{k=1}^{\infty} \gamma^k \phi(s_k)\right) && (6.5)\\ &= R_0 - \phi(s_0) && (6.6)\\ & && (6.7) \end{aligned}$$

where s_k is the state encountered at the kth timestep. Thus the return is only changed by subtracting a baseline which is a function of the initial state s_0. It has been shown that subtracting a baseline, especially an optimal baseline, can help

reduce the variance of the estimator [52, 28]. However, this is unfortunately a somewhat restricted way to modify the reward and we show that this can lead to slower convergence than might be possible.

Instead, we set the reward-shaping discount factor from Equation 6.1 to $\gamma = \gamma(i)$, in which the discount factor can vary over the iterations of the optimization. Let γ_M be the discount factor from the MDP. We then increase the reward-shaping discount factor over later iterations of the optimization as:

$$\gamma(i) = \gamma_M(1 - \exp(-i/\tau)) \tag{6.8}$$

where $\gamma(0) = 0$ and $\gamma(i) \to \gamma_M$ as $i \to \infty$. Thus, at convergence, $\gamma(i) \to \gamma_M$ and our time-varying reward shaping method approaches the original reward shaping formulation from Eq. 6.1, and thus the optimal policy at convergence is unchanged (the proof from the original reward shaping formulation [12] applies at convergence). However, by varying $\gamma(i)$ we are able to achieve a more flexible form of reward shaping that leads to faster convergence. We name this approach “reward guiding".

In this formulation, the discount factor $\gamma(i)$ can be viewed as a variance reduction parameter. A similar perspective of the discount factor is taken in previous work [9], in which the discount factor of the MDP is viewed as a variance reduction parameter compared to the undiscounted MDP.

6.4 Implementation Details

Our neural network maps from the input state to target robot joint positions, which are defined as changes in angle from the current joint positions. The network is defined by 3 hidden layers followed by the output layer. Each hidden layer has 64 nodes, with tanh non-linearities between each hidden layer. The network is implemented in Theano [53]. For the reinforcement learning algorithm, we use TRPO [8] and Generalized Advantage Estimation (GAE) [9] with a linear baseline using the implementation from rllab [33]. We use the default parameters from TRPO in rllab for our optimization, and we train the policy for 15,000 iterations with a batch size of 50,000 timesteps and 150 timesteps per episode. We use a discount factor of γ = 0.95, a KL-divergence constraint of δ_{KL} = 0.01, and a λ = 0.98 from GAE. We train the policy in simulation using the Mujoco [54] simulator.

We use a position controller to map the target joint angles output by the network to motor torques. For ensuring good transfer from simulation to reality, we found that it is important that the controller gains are set low enough such that the controller does not overshoot. In simulation, we allow enough time between actions to allow the controller reach the target position with an error smaller than 0.01 radians.

On the real PR2, we detect the position of the object using the Robot Operating System (ROS) package `ar_track_alvar`. For the position controller, we use the integrated position controller of the PR2, from the ROS package `pr2_controller_manager`.

6.5 Results

The experiments were designed to evaluate the impact of our contributions, both of how to learn the policy learning in simulation and how to transfer the policy from simulation to reality. In particular we seek to answer the following questions:

- Can we incorporate prior task knowledge into the observation space or reward function to increase the sampling efficiency or improve task performance?
- Can reward guiding improve the learning time without asymptotically modifying the MDP?
- How does a learning-based approach to non-prehensile manipulation compare to that of a hard-coded baseline?
- Can modularity be used to transfer a policy from simulation to the real-world while maintaining a similar level of performance?

We analyze our method on simulation in Sections 6.5.1, 6.5.2, and 6.5.3. And We show the results on real world in Section 6.5.4. In addition to the below experiments, videos of our results are also available online[1].

6.5.1 Simulation Training

Prior knowledge in the state space

We analyze the effect of modifying the state space on the training time. In a typical deep reinforcement learning setup, the system only receives the minimal state space as input. In this case, that would include the robot joint angles and velocities; the object position, orientation, and velocity; and the goal position to which the robot must push the object. We call this minimal observation o_{base}.

However, we show that we can augment the state space to improve the convergence rate. We define $o_{ee-object}$ as the 3-dimensional vector pointing from the robot end-effector to the object, and we define $o_{object-goal}$ as the 3-dimensional vector pointing from the object to the goal. Figure 6.4 shows that adding these terms to the state space reduces the training time needed for the robot to learn to perform the task.

Prior knowledge in the reward function

Next, we analyze the effect of modifying the reward function has on training time. We explore the contribution of each of the following reward terms:

(i) $r_{obj,goal} = -d(obj, goal)$

[1] https://goo.gl/fehPWw

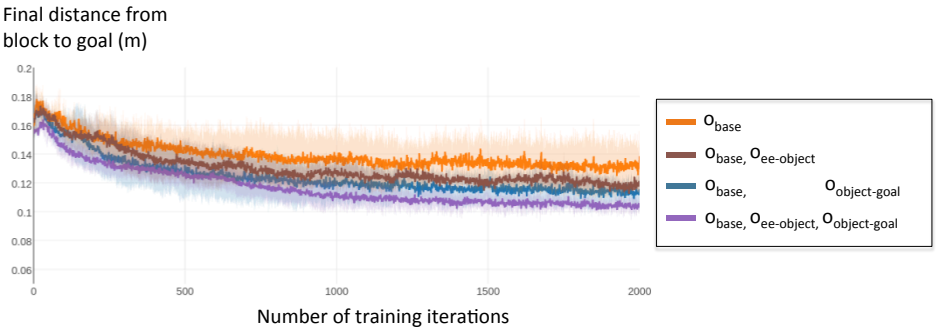

Figure 6.4: We use prior knowledge about the task to modify the state space. Our results shown that adding prior knowledge can lead to faster convergence compared to just using the original state space.

(ii) $r_{angle} = \cos(o_{ee-object}, o_{object-goal})$

(iii) $r_{ee,obj} = -d(ee, obj)$

The first term, $-d(obj, goal)$, measures the negative distance between the object and the goal (the position to which the object must be pushed). This term must be included in order to encourage the robot to push the object towards the goal.

The second term, $\cos(o_{ee-object}, o_{object-goal})$, encourages the robot's end-effector to approach the object from the appropriate angle. The term $o_{ee-object}$ is the vector from the robot end-effector to the object, and $o_{object-goal}$ is the vector from the object to the goal. By measuring the cosine between these vectors, we encourage the end-effector, the object, and the goal to lie along a straight line. However, the weight on this term is relatively small because this term is only intended to guide the robot end-effector to the approximately correct position. For accurate pushing, the robot may occasionally need to move its end-effector to a different location that deviates from this line.

The third term, $-d(ee, obj)$, measures the negative distance from the robot end-effector to the object. This term will encourage the robot end-effector to move near the object. If the robot is not given this term, then it will initially perform actions that do not interact with the object. The robot would then not see any change to the distance between the block and the goal, and the robot would not receive any feedback that allows it to discover whether its actions are useful. By adding this term, the robot's initial actions are guided towards the object. However, the weight on this term is again small because this is not the primary objective; the robot has to move the end-effector around, and occasionally away from the block, in order to re-position and push the block in different directions.

We explore the effect of each of these terms by investigating the following reward functions:

(i) $r_1 = r_{obj,goal}$

(ii) $r_2 = r_{obj,goal} + w_1 r_{angle}$

(iii) $r_3 = r_{obj,goal} + w_2 r_{ee,obj}$

(iv) $r_4 = r_{obj,goal} + w_1 r_{angle} + w_2 r_{ee,obj}$

where w_1 and w_2 are weighting factors on each of the reward terms.

The results are shown in Figure 6.5. In the plot, we have named r_1 = baseline, r_2 = baseline + angle, r_3 = baseline + ee-distance, and r_4 = baseline + angle + ee-distance. We can see that, if only the minimum reward function is specified, as in r_1, then the robot makes very little progress in learning, since as mentioned, the robot's initial random actions will not usually affect the distance from the block to the goal. Adding in only the angle term (as in r_2) also does not have much effect. If we encourage the robot to move its end-effector towards the object (as in r_3), then we see a significant improvement in the training time. Combining all three terms leads to the best convergence; once the robot is encouraged to move towards the block, encouraging the robot to move and place its end-effector at the correct angle further helps guide the robot towards the correct pushing behavior.

Figure 6.5: We use prior knowledge about the task to modify the reward function. Our results show that adding prior knowledge can lead to significantly faster convergence compared to just using the original reward function.

6.5.2 Reward Guiding

Finally, we analyze the increase in sample efficiency of using our proposed annealing reward shaping, which we refer to as "reward guiding," instead of the conventional reward shaping method from Ng, *et. al.* [12].

For guiding we use the reward $r_{obj,goal}$, and $\phi(s) = w_1 r_{angle} + w_2 r_{ee,obj}$ as potential function. This results in a faster learning (as show in section 6.5.1), and at the same time we optimize the original objective function asymptotically. Then our reward can be written as:

$$r_{guiding} = r_{obj,goal}(s') + \gamma(i)[w_1 r_{angle} + w_2 r_{ee,obj}](s') - [w_1 r_{angle} + w_2 r_{ee,obj}](s)$$

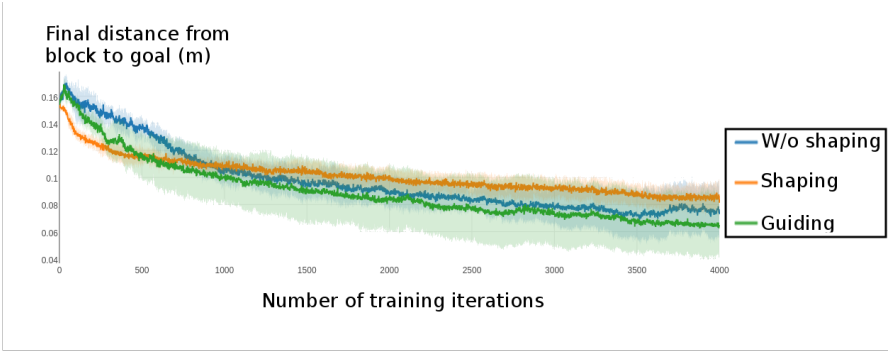

Figure 6.6: We measure the performance over iterations of reward guiding, shaping, [12] and learning without shaping (i.e. just adding extra reward terms). Our results show that reward guiding can lead to faster convergence compared to just using reward shaping or learning without shaping.

Figure 6.6 shows the comparison between using reward guiding, reward shaping, and modifying the reward without proper shaping by adding additional reward terms (as in r_4 from the previous section).

The results show that reward guiding has the best performance. Reward guiding accomplishes the same final distance from the block to the goal as shaping after 40% of the total number iterations. Ng, *et. al.* [12] proved that reward shaping does not affect the optimal policy. Similarly, reward guiding is guaranteed to have the same final optimal policy, but simply adding extra reward terms (as in r_4) would not.

6.5.3 Baseline Comparison

In this section, we show the need for learning-based methods for non-prehensile manipulation, especially in domains where the robot needs to push objects from arbitrary initial positions and orientations. To demonstrate the need for learning in such scenarios, we compared our performance to that of a non-learning based ("hard-coded") baseline procedure. In our "hard-coded baseline", at the beginning of each episode, a gripper is placed on the opposite side of the object from the goal. The gripper then pushes the object along the vector that goes from the center of the object to the goal until the distance from the object to the goal stops decreasing.

We compare the performance of our learning-based approach to that of the baseline, testing both methods in simulation (the real-world experiments of our method are desribed in Section 6.5.4). In these experiments, the goal is fixed, and the object is placed in every point of a grid of size 0.4 m $\times$ 0.25 m, with a resolution of 0.01 m, and the object orientation is sampled from a random uniform distribution on $[-\pi, \pi]$.

Figure 6.7 shows the final distance between the object and the goal from each starting position. In it we can see that our method is robust to varying initial

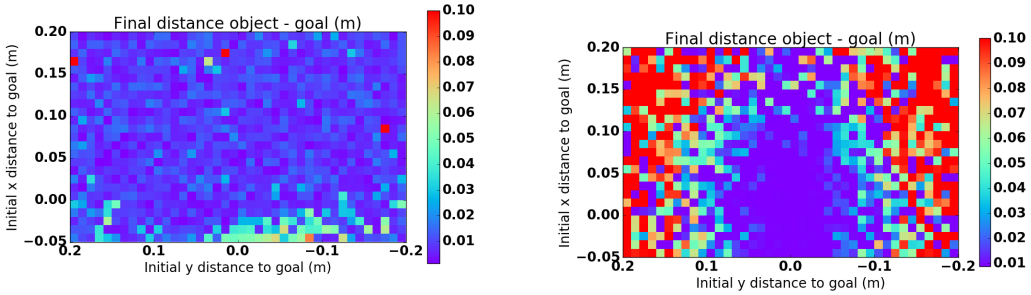

Figure 6.7: Heatmap of the final distance between the object and the goal. The axes denote the relative initial position of the object with respect the goal. The final distance is thresholded to a maximum value of 0.1 m. Red indicates that the final distance from the object to the goal is higher than 0.1m, whereas purple indicates a 0 distance. **Left:** Our method. **Right:** Baseline. (Best viewed in color).

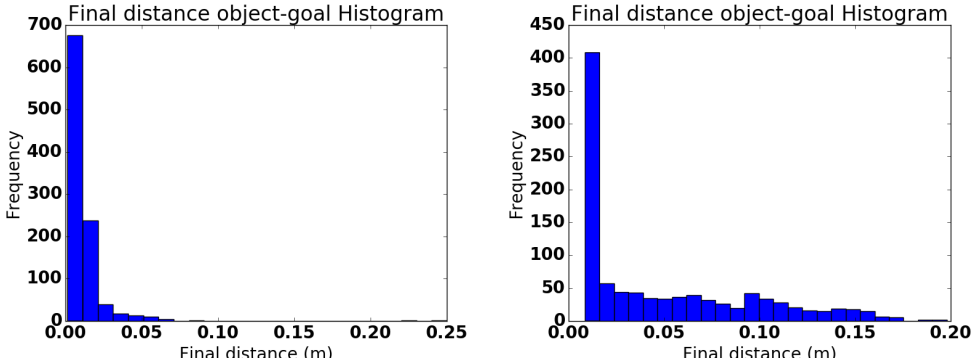

Figure 6.8: Histogram of the final distance between the object and the goal. Thresholded to a maximum value of 0.25 m. **Left:** Our method. **Right:** Baseline.

positions. On the other hand, the hard-coded baseline has much worse accuracy for some initial object positions. A histogram of the final distances between the block and the goal can be found in Figure 6.8. The poor performance displayed by the baseline is due to the variability in the environment, specifically in the orientation of the object relative to the goal position.

Figure 6.9 shows the final distance between the block and the goal as a function of the initial orientation of the object. The orientation is defined as the angle between longest side of the block and the x-axis.

The baseline fails to push the object for some initial orientations since the gripper slides off the side of the block instead of pushing it closer to goal. In contrast to our learning-based approach, the baseline is unable to recover from such situations. The sharp decrease in the final distance around 1.1 radians is due to the orientation at which the gripper starts pushing against the corner or

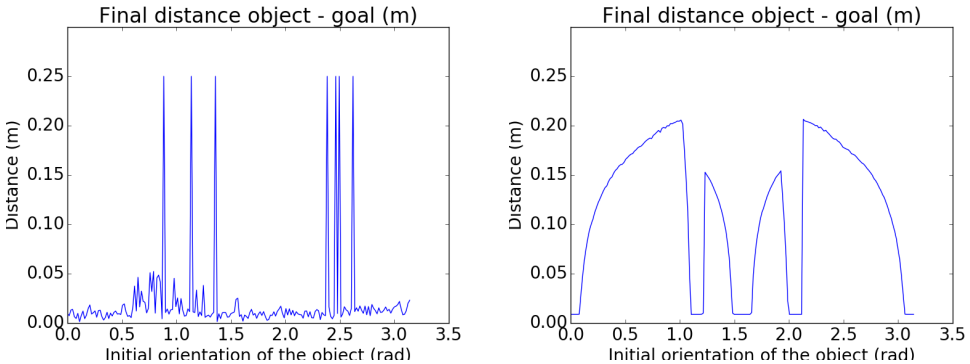

Figure 6.9: Final distance between the object and the goal with respect the orientation of the object in the plane of the table. **Left:** Our method. **Right:** Baseline.

against the shortest side of the block. To view some examples of the baseline failure cases, see the online video[2].

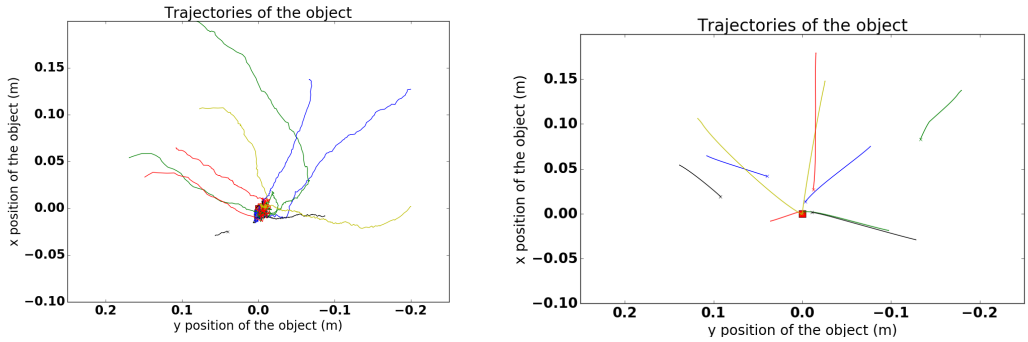

Figure 6.10: Trajectories of the center of the object while being pushed. The axes denote the relative position of the object with respect to the goal. The red square identifies the goal, which is located at the origin. **Left:** Our method. **Right:** Baseline. (Best viewed in color).

In contrast, our learned policy is robust to changes of orientation, as seen in Figure 6.9 (left). The learned policy is able to perform complex pushing trajectories, correcting the trajectory and recovering the object if it has pushed the block too far. A visualization of the pushing trajectories produced by our method, compared to those of the baseline, are shown in Figure 6.10.

[2]https://goo.gl/fehPWw

6.5.4 Real-World Pushing

In this section we show how modularity leads to a good performance when transferring the policy from simulation to real world.

The experiments with the real PR2 were implemented with the same set up as described in the former section: the goal is placed in a fixed position and the object is randomly sampled on the table within a 1000 cm^2 area. Each episode is terminated after 300 timesteps or when the distance from the object to the goal is less than 2 cm (whichever happens first). We sampled 20 initial object positions to test the performance of our approach.

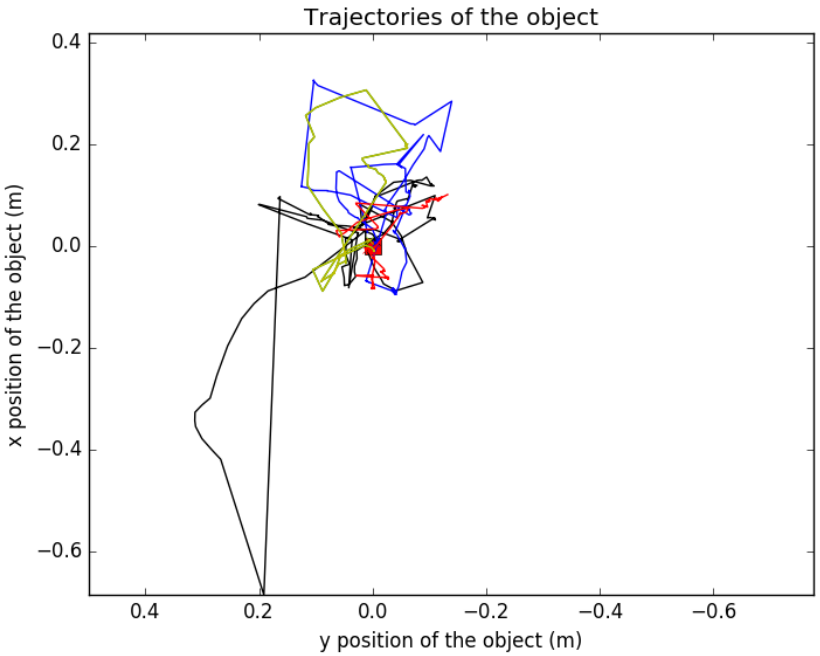

Figure 6.11: Trajectories of the center of the object while being pushed in the real world. The axes denote the relative initial position of the object with respect the goal (located at the origin). The red square identifies the goal. (Best viewed in color).

The block in simulation and real-world are similar in size; but the mass distribution is different, the simulated block is solid, whereas the real one is hollow. The friction between the block and the table and between the block and the gripper are also different between simulation and the real world. This leads to different dynamics of the object and different resulting block trajectories, as shown in Figure 6.11. Nevertheless, our method is capable of performing the task (defined as at the final time step being within 2cm of the goal) in 70% of the episodes, as shown in Figure 6.12. Figure 6.13 shows the final distance of the object for each of the 20 sampled initial positions, demonstrating that our method succeeds from varying initial block positions.

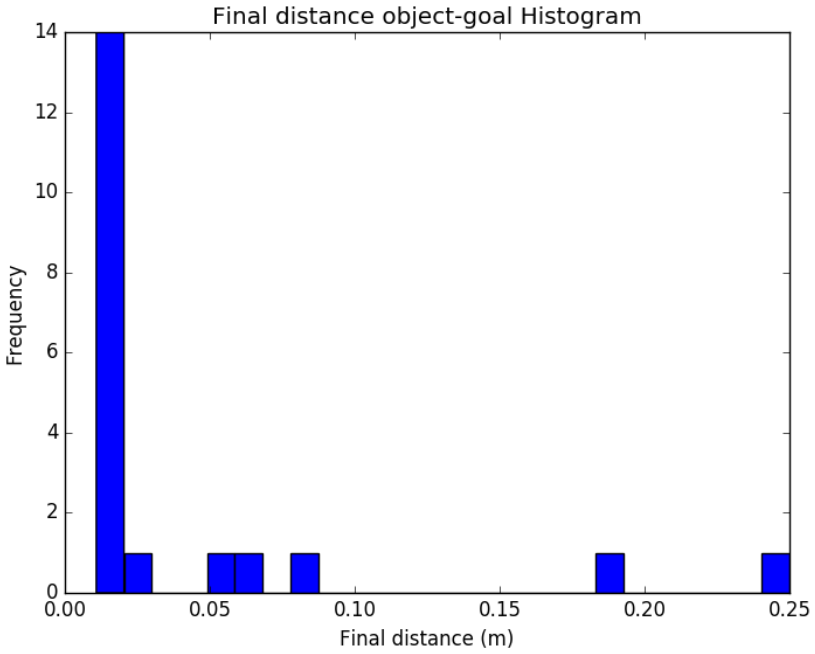

Figure 6.12: Histogram of the final distance between the object and the goal. Thresholded to a maximum value of 0.25 m.

The failing cases, in which the block does not reach within 2 cm of the goal, were due to a poor estimation of the object position or orientation. The AR tags were detected from a single kinect camera attached to the robot's head. In some cases the robot arm occludes the AR tags, which led to an incorrect estimate of the object position. We mitigate this problem by estimating the position of the object using the Mujoco simulator whenever the AR tags are not visible. Moreover, the small differences between controllers lead to different motions of the arm making it hard to achieve fine grain precision. This can be fully viewed in the more erratic trajectory of the real world object in Figure 6.11.

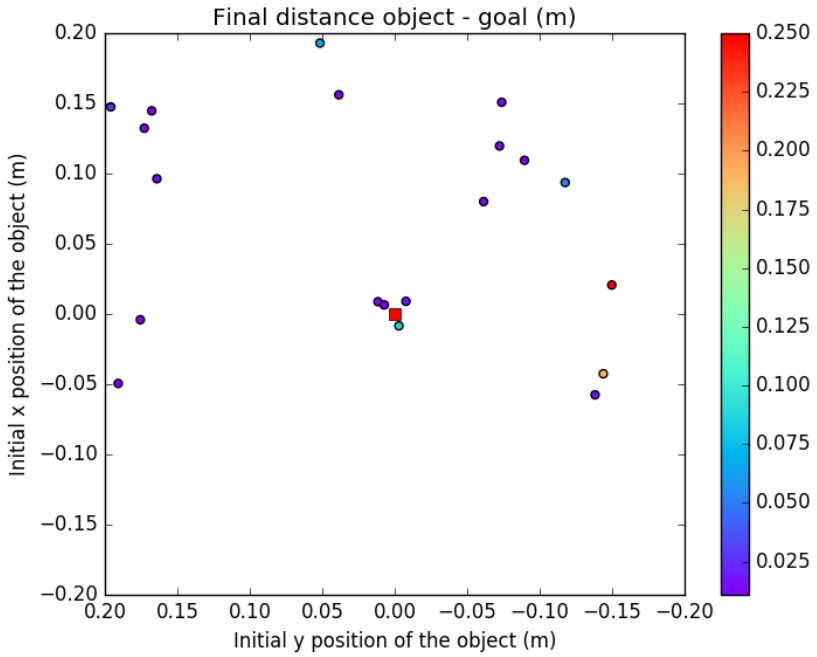

Figure 6.13: Scatter plot of the final distance between the object and the goal. The axis denote the relative initial position of the object with respect the goal. Thresholded to a maximum value of 0.1 m. Red indicates that the final distance from the object to the goal is higher than 0.1m, whereas purple indicates a 0 distance. (Best viewed in color).

7 Conclusions

7.1 Evaluation of Results

Transferring from simulation to real-world in non-prehensile manipulation with comparable performance has been achieved. Decoupling the policy from the visual inputs and torque outputs we alleviate the mismatch problem between simulation and real-world, thus making the policy more robust. This work has been extended by showing how to incorporate prior knowledge in the MDP by modifying of the observation space and the reward function.

Furthermore, we present the concept of "reward guiding" which does not modify the asymptotic optimal policy and leads to a faster learning than reward shaping [12] or adding a linear combination of other reward terms.

7.2 Future Work

This work aims to shed new light into the problem of transfer from a simulated environment to the real world. A problem that, when fully solved in all their aspects, will revolutionize the field of Robotics and Artificial Intelligence. There is still much work to do in this problem, and since we submitted the paper some new exciting advances have been published.

The main difficulty to tackle in sim-to-real is the changing dynamics between both environments. The friction coefficients, masses, inertias, and so on may differ from one environment to the other. So the question is, how do we make policies robust to such changes? There have been some works which have contributed to this question [55, 56, 57, 41]. The work presented here deals with it by imposing a quasi-stationary environment and using a controller that does not overshoot. However, in dynamic tasks this approach becomes infeasible. An immediate extension of our work would be to use a recurrent neural network as a policy, and train it with auxiliary tasks to predict the new dynamics. Then the recurrent neural network could adapt to those changes and act accordingly. Using a recurrent neural network may end up with a policy robust against occlusions, as well.

Further limitations of our work come from the use of AR tags for the detection of the objects. This could be solved using an object detection algorithm. However, in more complex tasks that involve cluttered scenes, fine grain position or an unknown number of objects, it would be desirable to train the policy using the entire image as input. Some recent work by Tobin et al. [58] deals with the differences between real images and simulated ones by using a large number of simulated scenes with different colors, scenes, camera positions, etc... Hence, treating the real image just as some random image from simulation. This opens the door to the use of "cheap" data to solve transfer and enhance generalization.

Bibliography

[1] G. Tesauro, "Temporal difference learning and td-gammon," *Commun. ACM*, vol. 38, no. 3, pp. 58–68, Mar. 1995. [Online]. Available: http://doi.acm.org/10.1145/203330.203343

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," in *NIPS Deep Learning Workshop*, 2013.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 02 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14236

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.

[5] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1334–1373, Jan. 2016. [Online]. Available: http://dl.acm.org/citation.cfm?id=2946645.2946684

[6] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, T. Jebara and E. P. Xing, Eds. JMLR Workshop and Conference Proceedings, 2014, pp. 387–395. [Online]. Available: http://jmlr.org/proceedings/papers/v32/silver14.pdf

[7] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015. [Online]. Available: http://arxiv.org/abs/1509.02971

[8] J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz, "Trust region policy optimization." in *ICML*, 2015, pp. 1889–1897.

[9] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[10] A. Y. Ng, "Shaping and policy search in reinforcement learning," Ph.D. dissertation, 2003, aAI3105322.

[11] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[12] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *ICML*, vol. 99, 1999, pp. 278–287.

[13] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural Networks*, vol. 21, no. 4, pp. 682–697, May 2008.

[14] S. Kakade, "A natural policy gradient," in *Advances in Neural Information Processing Systems 14 (NIPS 2001)*, T. G. Dietterich, S. Becker, and Z. Ghahramani, Eds. MIT Press, 2001, pp. 1531–1538. [Online]. Available: http://books.nips.cc/papers/files/nips14/CN11.pdf

[15] J. Schulman, "Optimizing expectations: From deep reinforcement learning to stochastic computation graphs," Ph.D. dissertation, 2017.

[16] S. Kakade and J. Langford, "Approximately optimal approximate reinforcement learning," in *Proceedings of the Nineteenth International Conference on Machine Learning (ICML 2002)*, C. Sammut and A. Hoffman, Eds. San Francisco, CA, USA: Morgan Kauffman, 2002, pp. 267–274. [Online]. Available: http://ttic.uchicago.edu/~sham/papers/rl/aoarl.pdf

[17] M. Dogar and S. Srinivasa, "A framework for push-grasping in clutter," *Robotics: Science and systems VII*, vol. 1, 2011.

[18] A. Cosgun, T. Hermans, V. Emeli, and M. Stilman, "Push planning for object placement on cluttered table surfaces," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011, pp. 4627–4632.

[19] M. T. Mason, "Mechanics and planning of manipulator pushing operations," *The International Journal of Robotics Research*, vol. 5, no. 3, pp. 53–71, 1986.

[20] G. Lee, T. Lozano-Pérez, and L. P. Kaelbling, "Hierarchical planning for multi-contact non-prehensile manipulation," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 264–271.

[21] M. R. Dogar and S. S. Srinivasa, "Push-grasping with dexterous hands: Mechanics and a method," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE, 2010, pp. 2123–2130.

[22] S. Goyal, A. Ruina, and J. Papadopoulos, "Planar sliding with dry friction part 1. limit surface and moment function," *Wear*, vol. 143, no. 2, pp. 307–330, 1991.

[23] R. D. Howe and M. R. Cutkosky, "Practical force-motion models for sliding manipulation," *The International Journal of Robotics Research*, vol. 15, no. 6, pp. 557–572, 1996.

[24] S. H. Lee and M. Cutkosky, "Fixture planning with friction," *Journal of Engineering for Industry*, vol. 113, no. 3, pp. 320–327, 1991.

[25] K. M. Lynch, H. Maekawa, and K. Tanie, "Manipulation and active sensing by pushing using tactile feedback." in *IROS*, 1992, pp. 416–421.

[26] K.-T. Yu, M. Bauza, N. Fazeli, and A. Rodriguez, "More than a million ways to be pushed. a high-fidelity experimental dataset of planar pushing," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 2016, pp. 30–37.

[27] R. Tedrake, T. Zhang, and H. Seung, "Learning to walk in 20 min," in *Proceedings of the Yale workshop on adaptive and learning systems*, 2005, pp. 10–22.

[28] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural networks*, vol. 21, no. 4, pp. 682–697, 2008.

[29] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[30] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, 2016.

[31] A. Colome, A. Planells, and C. Torras, "A friction-model-based framework for reinforcement learning of robotic tasks in non-rigid environments," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 5649–5654.

[32] A. Colome and C. Torras, "Dimensionality reduction and motion coordination in learning trajectories with dynamic movement primitives," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept 2014, pp. 1414–1420.

[33] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 2016.

[34] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 39, pp. 1–40, 2016.

[35] M. Salganicoff, G. Metta, A. Oddera, and G. Sandini, *A vision-based learning method for pushing manipulation*. University of Pennsylvania, 1993.

[36] T. Meriçli, M. Veloso, and H. L. Akın, "Push-manipulation of complex passive mobile objects using experimentally acquired motion models," *Autonomous Robots*, vol. 38, no. 3, pp. 317–329, 2015.

[37] M. Lau, J. Mitani, and T. Igarashi, "Automatic learning of pushing strategy for delivery of irregular-shaped objects," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 3733–3738.

[38] S. Walker and J. K. Salisbury, "Pushing using learned manipulation maps," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*. IEEE, 2008, pp. 3808–3813.

[39] J. Zhou, R. Paolini, J. A. Bagnell, and M. T. Mason, "A convex polynomial force-motion model for planar sliding: Identification and application," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 372–377.

[40] Y. Chebotar, M. Kalakrishnan, A. Yahya, A. Li, S. Schaal, and S. Levine, "Path integral guided policy search," 2017 (In press).

[41] A. A. Rusu, M. Vecerik, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-real robot learning from pixels with progressive nets," *arXiv preprint arXiv:1610.04286*, 2016.

[42] P. Christiano, Z. Shah, I. Mordatch, J. Schneider, T. Blackwell, J. Tobin, P. Abbeel, and W. Zaremba, "Transfer from simulation to real world through learning deep inverse dynamics model," *arXiv preprint arXiv:1610.03518*, 2016.

[43] I. Mordatch, N. Mishra, C. Eppner, and P. Abbeel, "Combining model-based policy search with online model learning for control of physical humanoids," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 242–248.

[44] M. Cutler and J. P. How, "Autonomous drifting using simulation-aided reinforcement learning," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 5442–5448.

[45] S. Barrett, M. E. Taylor, and P. Stone, "Transfer learning for reinforcement learning on a physical robot," in *Ninth International Conference on Autonomous Agents and Multiagent Systems-Adaptive Learning Agents Workshop (AAMAS-ALA)*, 2010.

[46] S. Gupta, P. A. Arbeláez, R. B. Girshick, and J. Malik, "Aligning 3D models to RGB-D images of cluttered scenes," in *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[47] A. Krull, F. Michel, E. Brachmann, S. Gumhold, S. Ihrke, e. D. Rother, Carsten", I. Reid, H. Saito, and M.-H. Yang, *6-DOF Model Based Tracking via Object Coordinate Regression*. Cham: Springer International Publishing, 2015, pp. 384–399.

[48] E. Brachmann, A. Krull, F. Michel, S. Gumhold, J. Shotton, and C. Rother, *Learning 6D Object Pose Estimation Using 3D Object Coordinates*. Cham: Springer International Publishing, 2014, pp. 536–551.

[49] E. Brachmann, F. Michel, A. Krull, M. Y. Yang, S. Gumhold, and C. Rother, "Uncertainty-driven 6d pose estimation of objects and scenes from a single rgb image," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 3364–3372.

[50] F. Sadeghi and S. Levine, "$(CAD)^2$RL: Real singel-image flight without a singel real image," *arXiv preprint arXiv:1611.04201*, 2016.

[51] J. Asmuth, M. L. Littman, and R. Zinkov, "Potential-based shaping in model-based reinforcement learning." in *AAAI*, 2008, pp. 604–609.

[52] E. Greensmith, P. L. Bartlett, and J. Baxter, "Variance reduction techniques for gradient estimates in reinforcement learning," *Journal of Machine Learning Research*, vol. 5, no. Nov, pp. 1471–1530, 2004.

[53] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016. [Online]. Available: http://arxiv.org/abs/1605.02688

[54] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control." in *IROS*. IEEE, 2012, pp. 5026–5033.

[55] P. Christiano, Z. Shah, I. Mordatch, J. Schneider, T. Blackwell, J. Tobin, P. Abbeel, and W. Zaremba, "Transfer from simulation to real world through learning deep inverse dynamics model," *arXiv preprint arXiv:1610.03518*, 2016.

[56] E. Tzeng, C. Devin, J. Hoffman, C. Finn, X. Peng, S. Levine, K. Saenko, and T. Darrell, "Towards adapting deep visuomotor representations from simulated to real environments," *CoRR*, vol. abs/1511.07111, 2015. [Online]. Available: http://arxiv.org/abs/1511.07111

[57] W. Yu, C. K. Liu, and G. Turk, "Preparing for the unknown: Learning a universal policy with online system identification," *CoRR*, vol. abs/1702.02453, 2017. [Online]. Available: http://arxiv.org/abs/1702.02453

[58] J. Tobin, R. Fong, R. Alex, J. Schneider, W. Z. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," *arXiv preprint arXiv:1703.06907*, 2017.