Robotic Skill Learning from Very Few Demonstrations

by

Cem Eteke

A Dissertation Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Computer Sciences and Engineering

KOÇ
UNIVERSITY

2019

Robotic Skill Learning from Very Few Demonstrations

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Cem Eteke

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____

Asst. Prof. Barış Akgün

_____

Asst. Prof. Emre Uğur

_____

Prof. Deniz Yüret

Date: _____

To my beloved family

ABSTRACT

In this thesis, a novel skill learning framework that learns rewards from very few demonstrations and uses them in a policy search setting to improve the skill is introduced. The action and perceptual data that are extracted from the demonstrations are used to learn a parameterized policy to execute the skill and a goal model to monitor the executions respectively. The skills are parameterized with Dynamical Movement Primitives (DMP) and a Hidden Markov Model (HMM) is used as the goal model. The rewards are learned from the HMM structure and its monitoring capability. The HMM is then converted to a finite horizon Markov Reward Process (MRP). A Monte Carlo approach is used to calculate the values corresponding to each hidden state of the HMM. Then, the HMM and the extracted values are merged into a Partially Observable MRP (POMRP). POMRP enabled us to obtain execution returns that are then used in policy search to improve the policy. In addition to reward learning, an adaptive exploration strategy is introduced to a black box optimization based PS method. The resulting framework is evaluated with five different policy search methods in a robotic simulation environment for two skills: Open in which robot learns to open a box and Close in which robot learns to close a box. The results show that the returns extracted from the POMRP lead to better performance compared to sparse monitoring signals, and the introduced policy search approach converges faster with higher success rates and lower variance than the rest. Finally, the efficacy of the framework is validated in a real robot setting with the introduced policy search method for three skills: Open, Close and Draw in which robot learns to pull to open a wooden drawer. We show that, in the real robot, the three skills are improved to complete success from complete failure.

ÖZETÇE

Bu tezde, çok az gösterimden ödül öğrenip, bu ödülleri politika aramasında, beceriyi geliştirmek için kullanan bir robotik beceri öğrenme sistemi tanıtılmaktadır. Gösterimlerden çıkartılan hareket ve algı verisi, beceriyi gerçekleştirmek için kullanılan poliçe ve hareketi gözlemek için kullanilan hedef modeli parametrelerinin öğrenimi için kullanılmıştır. Beceriler, Dinamik Hareket Pirimitivleri (DHP) ile parametrize edilirken, Saklı Markov Modeli (SMM) hedef modeli olarak kullanılmıştır. Öduller ise SMM'in yapısı ve gözlem kabiliyerleri vasıtası ile öğrenilmiştir. Bir sonraki adım olarak SMM, sonlu ufka sahip Markov Ödül Süreci'ne (MÖS) çevrilmiştir. Monte Carlo yöntemi ile SMM'in saklı durumlarının ödul değerleri hesaplanmıştır ve bu değerler SMM'i Kısmen Gözlenebilir MÖS'e (KGMÖS) çevirmek için kullanılmıştır. KGMÖS sayesinde robot, poliçe aramasında kullanılmak üzere dönütler elde etmiştir. Ödül öğreniminin yanı sıra, bu tezde, kara kutu en iyilemesi tabanlı bir poliçe araması metoduna adaptif keşif stratejisi uygulanmıştır. Ortaya çıkan poliçe araması yapısının performansı, robotik simülasyon ortamında beş farkli poliçe araması metodu kullanılarak iki tane beceri ile ölçülmüştür. Bu beceriler robotun bir kutu açtığı Açma ve bir kutu kapadığı Kapama becerileridir. Sonuçlar göstermektedir ki, KGMÖS'den çıkartılan ödüller, hedef modelinden elde edilen aralıklı gözlem sinyallerine göre daha iyi performans sergilemektedir. Buna ek olarak, bu tezde tanıtılan poliçe araması metodu diğer poliçe araması metodlarına göre daha hızlı ve daha az varyans ile yakınsamaktadır. Son olarak, simülasyonda gözlemlenen sonuçlar, gerek robotta üç tane beceri ile test edilmiştir. Bu beceriler Açma, Kapama ve robotun tahta bir çekmeceyi çekerek açtığı Çekme becerisidir. Robot, bütün becerileri tam başarısızlıktan tamamen başarılı olacak şekilde öğrenmiştir.

ACKNOWLEDGMENTS

I would first like to express my gratitude to my thesis advisor Dr. Barış Akgün. Thank you for believing in me, always being there as a helping hand when I got stuck and always having time for a chit chat even when we are all swamped. Most importantly, thank you for being more than a thesis supervisor.

I have always been happy to be a part of Alive Lab. I am thankful to Doğancan Kebüde for being both my academic brother and a very close friend, thank you for your robust presence both throughout my research and my life. I thank Onur for such enjoyable chats. I also thank Utku for adding different perspectives to my way of thinking about life.

I have had the privilege of knowing great people during my graduate studies at Koç University. I would like to thank Salih for being my close friend during day and my spotter during night, may he always squat deep. I am thankful to have had such enjoyable conversations with Batu, Selen, Ezgi and Merve. I would also like to thank Umuralp, Arda, Koray and Önder for being there when I needed some laughter. I am thankful to Atila, Beyza, Tuba and Nilüfer for their energetic way of conversation. I have had the privilege of having Mert Ögetürk and Luna as my flatmates, I have really enjoyed the times when we had late night discussions, binge-watched Netflix series and the countless silly moments that we have shared. I was lucky to have my close friends, Egehan, Çağatay, Tuna, Mert Kıray, Orhun, Mustafa A., Murat, Mehmet, Mustafa S., Deniz and Elibol being there for me through both good and bad times.

My biggest gratitude is to my sister Ceren, my mother Siven, my father Cenan, my brother-in-law Can and my nieces Maya and Arya. I couldn't have achieved my goals without their infinite support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

The advent of low cost and safe collaborative manipulators, and increasing automation demand in domains outside factory cages is leading to deployment of such robots in more and more diverse environments. This trend will put robots in contact with different end-users ranging from workers to elders and in various dynamic environments ranging from hospitals to our homes. It is not feasible to program robots for all the potential tasks, environments and end-users that they will encounter. As a result, Learning from Demonstration (LfD) approaches are attracting more and more interest, with the goal of enabling end-users to train their own robots. An example of such a training interaction can be seen in Fig. 1.1.

The most successful skill learning approaches rely on reinforcement learning (RL), specifically policy search (PS), with engineered reward functions. Robots have continuous and high dimensional state and action spaces which results in high sample complexities. This is usually alleviated by learning an initial skill from demonstration and using policy representations with relatively low number of parameters. The main challenges of this approach is the need for an engineered reward function and the limited patience of end-users translating to very few demonstrations. Existing methods can handle the latter challenge, given a dense reward function but the former is still an open question. Not all skills have mathematically easy to define rewards and even if they do, majority of end-users cannot be expected to define them. Motivated by this goal, our focus in this work is learning rewards from demonstrations, to be used in RL on robotic skills. We define these skills as short duration actions as opposed to complete tasks, e.g. pouring cereal vs preparing an entire breakfast. Existing reward

Figure 1.1: A kinesthetic learning from demonstration interaction.

learning methodologies require more data points than is feasible from an end-user interaction perspective. Furthermore, robots should converge with minimum number of trials during PS due to cost and safety concerns. In this work, we develop a novel reward learning approach that utilizes the perceptual information from a low amount of demonstrations and introduce an adaptive update step for a commonly used PS approach to speed up convergence.

It has been shown that end-users focus on providing successful demonstrations rather than high quality motions [Akgun et al., 2012b]. Therefore, *perceptual data* i.e robot's observation of the environment via it's sensors during the demonstration can reveal the underlying goal of the skill. Based on this observation, authors of [Akgun and Thomaz, 2016] propose to learn two models in LfD, namely an *action model* and a *goal model*. The action model represents how the skill should be performed and the goal model represents how the object of interest of the skill changes perceptually during the skill execution. They show that the goal model can be used to recognize whether an execution performed successfully or not. They then use the output of the goal model [Akgun and Thomaz, 2015] as a sparse self-supervision signal to improve and adapt the action model. As a result, the model that is used to represent the skill

(action model) can be updated depending on the goal of the skill (goal model) to make the robot adaptable both to its user and the environment. However, this line of work relies on keyframe demonstrations and a binary self-supervision signal. The former decreases the amount of applicable skills and the latter makes it unsuitable for PS methods.

Our proposed approach improves upon the aforementioned ideas by utilizing trajectory demonstrations, using learned perceptual representations instead of engineered ones and most importantly learning denser rewards, as well as enhancing a popular black-box policy search approach. For representing the action model, we use Dynamical Movement Primitives (DMP) [Ijspeert et al., 2003] from end-effector poses. For learning perceptual features to be used as *goal data*, we use a two stage approach; (1) a skill agnostic auto-encoder deep neural network [Achlioptas et al., 2017] and (2) skill specific principal component analysis (PCA). This data is used to learn a Hidden Markov Model, with its hidden-states representing the perceptual sub-goals and goals of the skill. We posit that the sub-goals contain information to update the action model more effectively during skill learning. Towards this end, we assign rewards to hidden states by using the sparse success/fail signal, converting the HMM to a Partially Observable Markov Reward Process (POMRP). In return, POMRP provides *dense reward* information to be used in RL, eliminating the need to manually provide a reward function. We use the reward information, returns obtained from the POMRP, in serveral PS methods to update the action DMP. As suggested by [Stulp and Sigaud, 2013], we treat this as a black box optimization problem and use evolutionary strategy-based PI^2 (PI^2-ES) that they introduced. We further improve PI^2-ES by adaptive exploration, introducing PI^2-ES-Cov. To summarize, we introduce a general purpose framework for skill learning with learned rewards using the combination of dense rewards extracted from a POMRP learned from demonstration and PI^2-ES-Cov. Fig. 1.2 depicts the fundamental parts of this framework.

We evaluate our approach on two skills in simulation (i) *Open*: opening a box, (i) *Close*: closing a box. We use a black box variant of PoWER [Kormushev et al.,

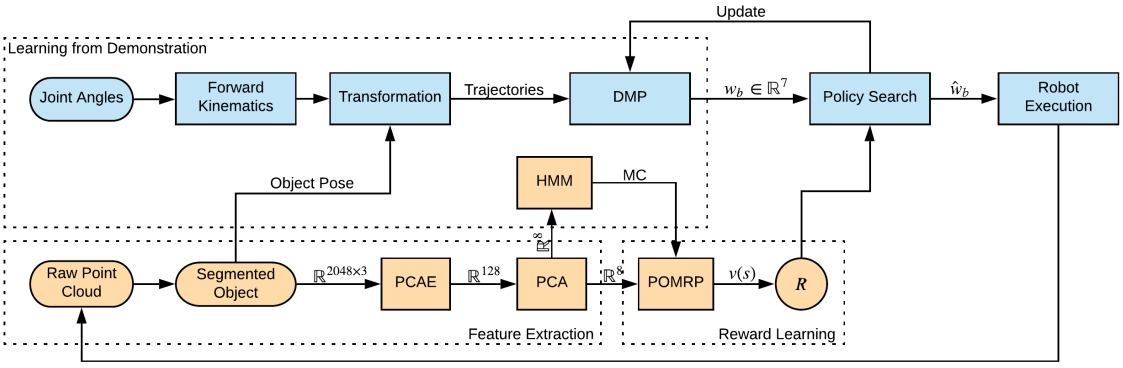


Figure 1.2: The flow chart of the introduced skill learning framework. Collected demonstrations are used to train a DMP and a Goal Model. Rewards (R) are learned from the goal model. Parameters of DMP (w_b) are updated via a policy search method using R.

2010], PI^2 [Theodorou et al., 2010], PI^2-ES and PI^2-Cov [Stulp and Sigaud, 2012] as baseline methods along with our version, PI^2-ES-Cov. We compare success rates of these methods with both sparse and dense rewards. We further validate our results on a real robot for three skills Open, Close and Draw: opening a wooden drawer. We show that using dense rewards outperforms sparse rewards for the aforementioned state of the art robotic skill learning methods. We further show that introduction of adaptive exploration to PI^2-ES leads to faster and better convergence.

1.1 Thesis Statement

Perceptual data can be used to extract sub-goals of the skill with Hidden Markov Models. Reward information can be assigned to the sub-goals to convert HMM to a POMRP. Updating the parameters of a DMP with the returns extracted from POMRP performs better than sparse reward signals and usage of adaptive covariance in policy search further improves the skill learning performance.

1.2 Contributions

Overall, this work introduces an LfD system that extracts rewards from perceptual states from very few demonstrations and use those rewards in several policy search algorithms, making the following contributions.

- Reward Learning: We present a method for learning rewards from very few demonstrations via a goal model. We utilize Partially Observable Markov Reward Process to come up with dense rewards.
- Goal Model: We show that utilizing Deep Neural Networks for perceptual feature extraction enables us to represent the goal of the skill with low-dimensional features, in turn eliminating the requirement of collecting high number of demonstrations.
- Skill Learning: The learned rewards are utilized in numerous Policy Search methods. We show that dense rewards enhance the performance of these methods. We further validate that utilization of black box optimization in Policy Search improves the performance as well. We further improve the results and enable faster convergence by introducing covariance update to the black box PS method. This results in minimal number of iterations for skill learning.

1.3 Outline

Following chapters of this thesis are organized as follows:

- Ch. 2: explains the current state of the literature regarding skill learning, reinforcement learning and reward learning in robotics.
- Ch. 3: discuss the technical details of the methods used in this work. Experienced readers can skip this chapter.
- Ch. 4: describes the novel reward learning methodology in depth.
- Ch. 5: explains the how the skill is represented, the aforementioned black box PS approach and how we utilized covariance update in the black box PS method.

- Ch. 6: explains the robotic setup used in this work both in simulation and real robot. Gets into the implementation details of the action model, goal model, perceptual feature extraction and policy search.
- Ch. 7: presents the results of the experiments discussed in Ch. 6 and discusses our observations.
- Ch. 8: discusses the overall framework and implications of the results.
- Ch. 9: summarizes the contributions in this work and discusses possible future directions.

Chapter 2

RELATED WORK

For the purposes of this work, we divide robotics skill learning and improvement in the context of LfD into three parts; (1) representation and directly learning from demonstration, (2) reinforcement learning and (3) reward learning, including inverse reinforcement learning. This section introduces the state of the literature regarding these topics. A generic review of LfD including the teacher perspective can be found in [Chernova and Thomaz, 2014].

2.1 Skill Representation and Learning

The basic idea of LfD is to come up with a skill model from user demonstrations. These demonstrations can come from several modalities such as teleoperation, motion capture, video or kinesthetic teaching [Argall et al., 2009]. This work utilizes kinesthetic teaching. An example of a kinestetic teaching interaction is depicted in Fig. 1.1.

There is considerable literature on skill representation. Skill learning in robotics requires the control of continuous high dimensional systems and majority of this literature aims to come up with parameterizations to handle this compexity. Dynamical system based approaches are widely used. For example Dynamic Movement Primitives (DMP) [Ijspeert et al., 2003] and their variants are very popular. Another common approach is probabilistic models such as the combination of Gaussian Mixture Models (GMM) with Gaussian Mixture Regression (GMR) [Calinon et al., 2007]. The ideas can also be used together as done with Stable Estimator of Dynamical Systems (SEDS) [Khansari-Zadeh and Billard, 2011]. All of these methods parameterize the skill and estimate their parameters from the demonstrated continuous motion.

Skills learned this way may not have satisfactory performance, especially when the teacher is a non-expoert. As a result, learned parameters are usually further updated with reinforcement learning. These methods do not incorporate perceptual state of the environment.

Directly learning from continuous demonstrations usually result in the failure of an action especially when the environment is dynamic or the teacher is non-expert. To overcome the problem of non-expert users and to simplify the interaction, authors of [Akgun et al., 2012a] introduce *keyframe* based LfD, teaching a robot an action model to execute the skill and a goal model to monitor the executions [Akgun and Thomaz, 2016]. Hidden Markov Models (HMM) are used to represent both of these models. They show that goal model performance is usually much better than the action model performance when the teachers are non-experts.

To overcome this, they use goal model for self-supervised re-training of the action model with the successful executions [Akgun and Thomaz, 2015]. However, using keyframes restrict the space of skills that can be learned. Furthermore, the self-supervision signal is sparse, making re-training harder. In this work, we follow similar ideas but use trajectory demonstrations, DMPs to represent the action model and learn dense rewards from the goal HMM by learning rewards for each hidden state, positing that they represent sub-goals. However, this work only looks at the perceptual final outcome of the skill but the skills may consist of perceptual sub-goals. In this work we utilize these sub-goals by assigning rewards to hidden states of the goal model in an unsupervised manner and optimize the action model to maximize these rewards. Overall process can be observed in Fig. 1.2.

There is a body of literature that segment demonstrations where these segments can be interpreted to be sub-goals. In [Konidaris and Barto, 2009], the authors introduce *skill chaining* which enables discovery of sub-goals which are less challenging to learn in continuous RL domains. Since performing skill chaining iteratively has high complexity, Maximum a Priori (MAP) change-point detection is applied to form a skill tree in [Konidaris et al., 2010] and RL is used to enhance the sub-skills. The

work in [Kuindersma et al., 2010] segments the skill and employs motion planning for execution of the sub-skills. GMMs are employed to extract *key phases* from a given continuous trajectory and GMR is used to form a trajectory in [Lee et al., 2012]. To form higher level representation from the segments, the work in [Niekum et al., 2013, Niekum et al., 2015] segment unstructured trajectories with Beta Process Auto-Regressive Hidden Markov Models (BP-AR-HMM). Most of these methods aim to segment user motions which may lead to sub-par performance as discussed above. In this work, we utilize the higher performance goal model information to learn sub-goals.

Recent studies introduced methods to employ Deep Neural Networks for LfD, e.g., [Schulman et al., 2015, Levine et al., 2016, Lillicrap et al., 2015, Gu et al., 2016]. These approaches commonly employ two different loss functions; (1) RL loss to enhance the policy and (2) supervised loss to imitate the demonstration [Nair et al., 2017, Hester et al., 2017]. Authors of [Gu et al., 2017], use an off-policy Deep Q Network that can be adapted to real robots. To make the learning process more effective they parallelize the learning to multiple real robots. Even though Deep Learning enables mapping from high dimensional states to complex motor behaviour, these approaches are impractical for realistic LfD scenarios since they require numerous demonstrations, numerous episodes and sometimes synthetic data generation [Bousmalis et al., 2017].

On the other hand, deep Neural Networks are good at extracting high level features from high dimensional raw data. Some approaches to robotic skill learning employs Deep Learning to extract perceptual states from RGB or RGB-D data. This can be achieved by either passing the state through a pre-trained network [Sermanet et al., 2016] or training the network during skill learning [Duan et al., 2017]. Since training a network from scratch during training is not viable we opted to use a pre-trained autoencoder for Point Clouds introduced by [Achlioptas et al., 2017]. This forms our high level goal data extracted from low level perceptual data. This goal data then is used to train a goal model.

2.2 Reinforcement Learning

Since learning from end-users result in sub-par skill models, reinforcement Learning (RL), especially policy search (PS) based approaches [Kober et al., 2013, Deisenroth et al., 2013], are widely used provided that a suitable reward function is available. However, learning a skill from scratch with RL is not viable due to the sample complexities resulting from the high dimensional and continuous nature of robots. Therefore, initializing RL with LfD is a favorable approach [Kormushev et al., 2010], which alleviates the issue of sample-complexity by giving the PS methods a favorable initial point.

As mentioned above, PS methods are preferred for robotic skill learning. Among the first PS approaches, REINFORCE algorithm [Williams, 1992] perturbs motor commands and uses *vanilla* gradients of the *expected return* to update the policy parameters. Vanilla gradients do not consider the geometry of the problem and as a result takes bad update-steps. Natural Actor-Critic (NAC) algorithm [Peters and Schaal, 2008] utilizes Fischer information matrix to calculate the natural gradient which gives a better direction towards the local optima.

It is preferable to perform PS by perturbing the policy parameters rather than the actions. The reasons are as follows; exploration in parameter space is safer in robotics, and depending on the action space some perturbations may not effect the motion since the controller of the robot acts as a low-pass filter or perturbations make the explored trajectory highly noisy.

Policy Learning by Weighting Exploration with the Returns (PoWER) algorithm [Kober and Peters, 2009] is one method that perturbs the parameters of the policy. PoWER calculates an update by weighting the perturbed parameters with returns and taking the average. Another common method is PI^2 [Theodorou et al., 2010]. PI^2 computes probabilities of each roll-out as normalized exponentiation of the return and just like PoWER, takes the weighted averages of the perturbed parameters. PI^2 is extended by utilizing the covariance updates of PI^2 (PI^2-Cov), introduced in [Stulp and Sigaud, 2012]. These methods require immediate rewards for each time step.

This implies that the reward function should be informative enough for each step. However, this is not always realistic when learning from goals since the return is acquired at the end of an execution, i.e the return is *black box*. In this case, it is difficult to quantify the exact steps that lead to a successful execution.

Authors of [Stulp and Sigaud, 2013] show that Black Box Optimization (BBO) based PS outperforms both PoWER and PI^2 and introduce PI^2-ES which is an Evolution Strategy (ES) based extension of PI^2. In this work we also regard the RL problem as BBO and further extend PI^2-ES to have adaptive exploration using Cross-Entropy Method based covariance matrix update of [Stulp and Sigaud, 2012] and introduce PI^2-ES-Cov. We show that when combined with dense rewards, PI^2-ES-Cov outperforms a black box variant of PoWER [Kormushev et al., 2010]. We compare the performances of PI^2, PI^2-Cov and PI^2-ES in a simulated robotic environment both for Open and Close skills. We validate our simulated robot results on a real robot for Open, Close and Draw for PI^2-ES-Cov.

Designing dense reward functions for PS approaches is not straightforward for robotic skill learning. However, some skills have relatively easy to define binary end-goals such as reaching or grasping. The authors of [Andrychowicz et al., 2017] introduced Hindsight Experience Replay (HER) which enables robots to extract *how well* they do from sparse rewards. HER achieves this by monitoring executions when the end-goal is perturbed. However, many real world tasks consists of sub-goals i.e even when the success can be measured, it does not always explain the performance of the sub-goals. By assigning rewards to each hidden state of the goal model, we are providing information about sub-goals as well. In addition, the hindsight idea cannot be applied to all the skills since extracting the context of when the failure would have been useful is not straightforward if not impossible for a large subset of skills.

2.3 Inverse Reinforcement Learning & Reward Learning

As mentioned above, one of the main challenges in RL approaches is engineering suitable reward functions which are inevitably skill specific. Furthermore, majority

of the skills with perceptual goals do not have easy to define rewards and even if they did, end-users cannot be expected to define them.

A way to overcome the problem of reward engineering in RL is Inverse reinforcement learning (IRL). IRL approaches estimate rewards from user demonstrations and use this to find the policy [Abbeel and Ng, 2004]. These ideas have been successfully applied to robotic skills such as the work done in [Finn et al., 2016]. Our idea of extracting rewards from goal data is similar to IRL. However, there are some challenges in IRL which make it difficult to apply to our LfD scenarios. The majority of IRL work in robotics aim to learn a reward (or equivalently cost) function in the robot task space whereas we are working in a perceptual space where the action-state relation is more complex. Furthermore, IRL approaches work better with expert demonstrations. As shown by [Akgun and Thomaz, 2016], humans focus on the goal of a skill rather than the quality of the motion when providing demonstrations. Therefore, we claim that extracting rewards from perceptual data of a demonstration is a favourable approach. Lastly, IRL approaches require more data than is feasible from an end-user perspective.

There exist approaches that aim to learn a reward function in a perceptual space. Authors of [Sermanet et al., 2016], extract a smooth reward function from demonstrations by extracting importance of intermediate steps and perceptual features. These rewards are then used in an RL setting. Another approach [Sermanet et al., 2017] extracts perceptual information of what type of changes in the environment is salient from multiple viewpoints. The authors use this information for both robotic skill learning with RL and human pose imitation. Both of these works require relatively large number of demonstrations. Whereas we learn rewards from very few number of demonstrations.

Chapter 3

BACKGROUND

3.1 Dynamical Movement Primitives

Dynamical Movement Primitives (DMP) [Schaal et al., 2005] uses a well-established dynamical system that generates a motion to reach the goal position g with duration τ. The dynamical system is a point-attractor, such as spring-damper system.

$$\tau\ddot{y} = K(g - y) - \tau D\dot{y} \tag{3.1}$$

The variables, y, $\dot{y}$ and $\ddot{y}$ are the position, velocity and acceleration respectively. The constants K and D are the spring coefficient and damping ratio respectively and chosen to ensure a critically-damped system i.e

$$D = 2\sqrt{K} \tag{3.2}$$

Given finite time, it is guaranteed that this system will converge to the goal position g. Nevertheless, complex motions can't be generated with this system since it is linear. Therefore, a forcing factor f that is a combination of non-linearities is introduced to modulate the system. With the addition of f, Eq. 3.1 can also be be written as a first-order dynamical system as

$$\begin{aligned} \tau\dot{z} &= K(g - y) - \tau Dz + f(x) \\ \tau\dot{y} &= z \end{aligned} \tag{3.3}$$

The forcing factor f is linear combination of B many non-linear basis functions, each parameterized by w_b.

$$f(x) = \frac{\sum_{b=1}^{B} \Psi_b(x) w_b}{\sum_{b=1}^{B} \Psi_b(x)} x(g - y_0) \tag{3.4}$$

Where, x, also referred as the *phase variable* is introduced. The initial value of the phase variable is 1 and it is gradually decreased to 0, enables the system to become autonomous.

$$\tau \dot{x} = -\alpha_x x \tag{3.5}$$

With B exponential basis functions, and initial position y_0 the basis functions are calculated as the following where σ_b and c_b are widths and centers of each basis function.

$$\Psi_b(x) = exp\Big(-\frac{1}{2\sigma_b^2}(x - c_b)^2 \Big) \tag{3.6}$$

With the fundemantals of DMP established, our main goal here is to come up with an f that forces the system to follow the given demonstration i.e y_{demo}, $\dot{y}_{demo}$ and $\ddot{y}_{demo}$. Since we know the forcing factor of the demonstration f_{target}, we can learn the parameters of f with simple linear regression.

$$f_{target} = \tau \ddot{y}_{target} - K(g - y_{target}) + \tau D \dot{y}_{target}) \tag{3.7}$$

The goal is to find parameters w_b that minimizes the following error.

$$J_b = \sum_{t=1}^{T} \Psi_b(t)(f_{target}(t) - w_b x(t)(g - y_{target_0})) \tag{3.8}$$

Solution to this weighted linear regression problem is

$$w_b = \frac{\mathbf{s}^T \mathbf{\Gamma}_b \mathbf{f}_{target}}{\mathbf{s}^T \mathbf{\Gamma}_b \mathbf{s}} \tag{3.9}$$

where

$$\mathbf{s} = \begin{bmatrix} x(1)(g - y_{target}(1)) \\ x(2)(g - y_{target}(2)) \\ \cdots \\ x(T)(g - y_{target}(T)) \end{bmatrix} \mathbf{\Gamma}_b = diag\left(\begin{bmatrix} \Psi_b(1) \\ \Psi_b(2) \\ \cdots \\ \Psi_b(T) \end{bmatrix} \right) \mathbf{f}_{target} = \begin{bmatrix} f_{target}(1) \\ f_{target}(2) \\ \cdots \\ f_{target}(T) \end{bmatrix}$$

This simple linear regression approach enables DMP to represent complex motions given enough number of basis with only one demonstration. That is why DMP is a perfect fit for the problem of learning from very few demonstrations. After coming up with DMP parameters for a skill, we improve the parameters via Policy Search as explained in Ch. 5.

3.2 Hidden Markov Models

A Hidden Markov Model (HMM) is a special type of a Markov Chain in which states are not directly observed but estimated with probability distributions. This work utilizes HMM with Gaussian Emissions. An HMM with Gaussian Emissions consist of the following components:

k : number of hidden states

S : $S = \{s_1, \ldots, s_i, \ldots, s_k\}$, the set of hidden states where s_i is the i^{th} hidden state.

P : $P = P_{ij}$ where $i, j \in S$, transition probability matrix where $P_{ij} = P(s_t = j | s_{t-1} = i)$

π : Prior probability vector

T : The set of terminal states

μ_i : The mean of the Gaussian emission probability of state i

Σ_i : The covariance of the Gaussian emission probability of state i

z_i : $\mathcal{N}(\mu_i, \Sigma_i)$ The emission of the state i

Φ : $\Phi = \{\mu_1, \Sigma_1, \ldots, \mu_k, \Sigma_k\}$, the set of emission probability parameters

λ : $\lambda = \{\Phi, \pi, P\}$, the model

Given the training data, the model parameters λ are computed with Expectation-Maximization (EM) as known as the Baum-Welch algorithm. In order to train the parameters, to sample random state sequence from an HMM and to calculate the posterior probability we utilize the forward-backward algorithm. Let $\alpha_i(t)$ be the *forward variable* of state i at time t. It is calculated as:

$$\begin{aligned} \alpha_i(1) &= \pi_i z_i(y_1), \ \forall_{1 \leq i \leq k} \\ \alpha_i(t) &= z_i(y_y) \textstyle\sum_{j=1}^{k} \alpha_{t-1}(i) P_{ij} \end{aligned} \tag{3.10}$$

With the forward variable, the likelihood of an observation sequence Y of length T is calculated as:

$$P(Y|\lambda) = \sum_{i=1}^{N} \alpha_i(T) \tag{3.11}$$

Let $\beta_i(t)$ be the *backward variable* with $\beta_i(T) = 1 \ \forall_{1 \leq i \leq k}$, then the backward variable is calculated as:

$$\beta_i(t) = \sum_{i=1}^{k} P_{ij} z_i(y_{t+1}) \beta_{t+1}(i) \tag{3.12}$$

With the forward and backward variables at hand, the posterior probability $\gamma_i(t) = P(s_t = i|Y, \lambda)$ i.e ending up at state i at time t is calculated as:

$$\gamma_i(t) = \frac{\alpha_i(t)\beta_i(t)}{P(Y|\lambda)} \tag{3.13}$$

In addition, $\xi_{ij}(t) = P(s_{t+1} = j, s_t = i|Y, \lambda)$ i.e the probability of ending up at states j and i at times $t + 1$ and t respectively is calculated for the Baum-Welch algorithm as:

$$\xi_{ij}(t) = \frac{\alpha_i(t) P_{ij} \beta_j(t+1) z_j(y_{t+1})}{P(Y|\lambda)} \tag{3.14}$$

At each iteration of the EM algorithm the following updates are performed:

$$\pi_i = \gamma_i(1) \tag{3.15}$$

$$P_{ij} = \frac{\sum_{t=1}^{T} \xi_{ij}(t)}{\sum_{t=1}^{T} \gamma_i(t)} \tag{3.16}$$

$$\mu_i = \frac{\sum_{t=1}^{T} \gamma_i(t) y_t}{\sum_{t=1}^{T-1} \gamma_i(t)} \tag{3.17}$$

$$\Sigma_i = \frac{\sum_{t=1}^{T} \gamma_i(t)(y_t - \mu_i)(y_t - \mu_i)^T}{\sum_{t=1}^{T-1} \gamma_i(t)} \tag{3.18}$$

After each iteration, forward and backward variables are recalculated to update the parameters λ until convergence i.e the likelihood does not change. After training, we use the posterior probabilities $\gamma_i(t)$ in Partially Observable Markov Reward Process to calculate expected returns as discussed in Ch. 4. Finally, we generate random state sequences from a trained HMM to be able to assign values to each hidden state of the HMM as discussed in Ch. 4.

3.3 Reinforcement Learning

Reinforcement Learning (RL) consists of an *agent*, acting in an *environment*. The agent takes *actions* depending on the *state* of the environment via its policy π. In return, the environment responds via emitting the next state of the environment and a reward. The aim of RL is to come up with a policy that maximizes the total reward i.e *return* of the agent.

3.3.1 Markov Decision Process

The agent and the environment interacts with each other at each discrete time step t. Markov Decision Process (MDP) is a framework for modeling the environment and the decision making process. An MDP consists of:

- S : Set of states
- A : Set of actions
- T : $P(s_t = s' \mid s_t = s, a = a_t)$ i.e transition probability from state s to s' given an action a.
- R : Reward obtained at state s_t after taking action a_t. Also refered as r_t for simplicity.

With the employment of MDP we can find a policy π that maximizes the expected discounted rewards where γ is the *discount factor* and $0 \leq \gamma \leq 1$:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{j=0}^{\infty} \gamma^j r_{t+j+1} \tag{3.19}$$

The discount factor determines the impact of the future rewards and introduces numerical stability. Therefore, an MDP is defined by the following set of variables (S, A, T, R, γ). When an agent is following a policy π, it needs a way to asses the *quality* of a state for taking actions. This quality is determined by the *value* of a state under the policy. A value function of a state s given π is defined as:

$$v_\pi(s) = \mathbb{E}_\pi \Big[\sum_{j=0}^{\infty} \gamma^j r_{t+j+1} \mid s_t = s \Big] \tag{3.20}$$

The function v_π is reffered as the state-value function. There exists a variant of this function which is the action-value function and it outputs the value of a state given an action:

$$q_\pi(s,a) = \mathbb{E}_\pi\left[\sum_{j=0}^{\infty} \gamma^j r_{t+j+1} \mid s_t = s, a_t = a\right] \tag{3.21}$$

Taking the maximum values across all possible policies yields to optimal values v_* and q_*. In return, the policy that follows the optimal values π_* is called the optimal policy. The solution i.e calculation of optimal values are possible through a set of equations called the Bellman Optimality Equations:

$$v_*(s) = \max_a \sum_{s',r} T(s,s',a)[r + \gamma v_*(s')] \tag{3.22}$$

$$q_*(s,a) = \sum_{s',r} T(s,s',a)[r + \gamma \max_{a'} q_*(s',a')] \tag{3.23}$$

Bellman Optimalitiy Equations shown in Eq. 3.22 and Eq. 3.23 are linear systems and their solutions yields to optimal values. These optimal values are then used to come up with the optimal policy.

3.3.2 Monte Carlo Policy Evaluation

Es explained in Sect. 3.3.1, a Markov Decisin Process contains a transition function $T(s,s',a)$ and a reward function $R(s_t,a_t)$. To solve the MDP with Bellman Optimality Equations, these functions are required. However, these functions i.e the *model* of the environment is not known for most of the time. This is where *model-free learning* comes to play. Aim of model-free learning is to learn the values of states as the agent acts in the environment.

Monte Carlo Policy Evaluation is a method to achieve model-free learning given a policy. The agent acts in the environment via the given policy until it reaches a *terminal state*. After termination i.e end of an *episode*, it sets the value of each state as the average of the returns obtained from a visited state. Algorithm below depicts Monte Carlo Policy Evaluation:

Algorithm 1 Algorithm for Monte Carlo Policy Evaluation

```
1: procedure MonteCarlo(S, π, γ)
2:     returns ← [ ]
3:     while true do
4:         τ = {s_1, r_1, s_2, r_2, ..., s_T, r_T} ← generateEpisode(π)
5:         for all t ∈ {1, 2, ..., T} do
6:             G ← getReturn(t, τ, γ)
7:             returns[s_t].append(G)
8:         for all s ∈ S do
9:             v(s) ← average(returns[s])
```

We use Monte Carlo Policy Evaluation to calculate values for each hidden states of an HMM by sampling state sequences and use those values to extract rewards as explained in Ch. 4.

3.4 Policy Search

Policy Search (PS) is set of RL algorithms that parameterizes the policy itself and optimizes the parameters θ to obtain maximum return. Policy Search (PS) methods are preferred for robotic skill learning [Deisenroth et al., 2013]. This is due to i) Exploration in parameter space is safer in robotics ii) Controller of the robot acts as a low-pass filter i.e some perturbations may not effect the motion iii) Most of the time, explored trajectory becomes very noisy. On top of that, a policy can be initialized with the demonstrations provided from the users to enable usage of RL in robotic skill learning.

In this work we use DMP parameters as policy parameters and optimize them with various of PS methods. This section gets into the detail about three of the most common PS methods in the literature which we use as baselines to compare the effects of learned rewards, black box optimization and covariance update in robotic skill learning. Note that all of the parameter explorations in these methods are just

adding random Gaussian noise to the parameters at an episode e with some covariance Σ:

$$\theta_e \sim \mathcal{N}(\theta, \Sigma) \tag{3.24}$$

3.4.1 PoWER

Policy learning by Weighting Exploration with the Returns (PoWER) is an Expectation Maximization-based (EM) policy search algorithm [Kormushev et al., 2010]. In their work, [Kormushev et al., 2010] also use this algorithm with DMPs for robotic skill learning so it is a good baseline to compare with other methods in our setting.

PoWER uses undiscounted cumulative reward as the metric to be optimized. In other words $\gamma = 1$ in Eq. 3.19. Let θ_n be the current policy parameters. PoWER updates the parameters at the current episode e by the following where $< . >$ denotes importance sampling:

$$\theta_{n+1} \leftarrow \theta_n + \frac{\left\langle (\theta_k - \theta_e) G_k \right\rangle}{\left\langle G_k \right\rangle} \tag{3.25}$$

Here G_k is the return obtained at a past episode episode k and θ_k is the explored weights at that episode. The following importance sampler determines the value of k where $ind(k)$ returns the k^{th} best episode return-wise:

$$\left\langle f(\theta_k) \right\rangle = \sum_{k=1}^{r} f(\theta_{ind(k)}) \tag{3.26}$$

Intuitively, what PoWER does is, it takes r best episodes and shifts the parameters towards the weighted average of those episodes' explored parameters.

3.4.2 PI²

The Policy Improvement with Path Integrals (PI^2) algorithm [Theodorou et al., 2010] is another very common algorithm used in PS in robotic applications, especially in skill learning. It is a path integral based approach to policy search and has its roots

from optimal control. In PI^2 episodes consists of rollouts but the parameters are not updated after a rollout, they are updated after each episode.

Unlike PoWER PI^2 utilizes per step returns for learning. Let the return obtained after each execution at time step t, episode e and rollout c be $G_{t,e,c}$ and explored parameters be $\theta_{e,c}$. PI^2 utilizes exponential weighting to the returns for a time step at each rollout to come up with probabilities:

$$P_{t,i} = \frac{e^{\frac{-1}{\lambda}G_{t,e,c}}}{\sum_i e^{\frac{-1}{\lambda}G_{t,e,c}}} \tag{3.27}$$

The probabilities calculated in Eq. 3.27 are then used for weighted averaging of the explored parameters to form the new parameters for each time step:

$$\theta^t_{e+1} = \sum_c P_{t,c}\theta_{e,c} \tag{3.28}$$

Then a temporal averaging is performed over the temporal parameters to come up with the new parameters where T is the total number of time steps:

$$\theta_{e+1} \leftarrow \frac{\sum_t (T-t)\theta^t_{e+1}}{\sum_t (T-t)} \tag{3.29}$$

Authors of [Stulp and Sigaud, 2013] introduce PI^2-ES which is a black box variant of PI^2, they utilize the total return to update the parameters instead of per-step returns and show that PI^2-ES performs better than PI^2 when DMP is used. We also compare PI^2 and PI^2-ES performance on robotic skill learning with the learned rewards. PI^2-ES is discussed in detail at Ch. 5.

3.4.3 PI^2-Cov

Notice that in Eq. 3.24 an exploration covariance Σ is utilized that determines the *exploration size* and it is the same throughout the learning process. Authors of [Stulp and Sigaud, 2012] introduce PI^2-Cov that also updates the exploration covariance matrix depending on the performance of the robot. They use the update rule used in Cross-Entropy Method but replace the probabilities with probabilities calculated in PI^2:

$$\Sigma^t_{e+1} = \sum_c P_{t,c}(\theta_{e,c} - \theta_e)(\theta_{e,c} - \theta_e)^T \tag{3.30}$$

The covariance matrices calculated in Eq. 3.30 are the covariances at each time step t, to update the exploration covariance matrix they again perform temporal averaging:

$$\Sigma_{e+1} \leftarrow \frac{\sum_t (T-t)\Sigma_{e+1}^t}{\sum_t (T-t)} \tag{3.31}$$

This enables the agent to explore parameters that leads to better returns, leading to faster convergence. In this work we combine this covariance update rule with PI^2-ES and introduce PI^2-ES-Cov. The details about PI^2-ES-Cov are discussed in Ch. 5. We compare PI^2-ES-Cov with PI^2-ES, PI^2-Cov, PI^2 and PoWER in a simulation environment with utilization of dense and sparse rewards.

3.5 Point Cloud Autoencoder

Point cloud represents a 3-D shape with a set of 3-D positions defined by x, y and z coordinates in Euclidean Space. Therefore, with N many points, the size of a single data point is $N \times 3$. In this work, we extract features from Point clouds via an autoencoder. Autoencoders are specific type of Neural Networks that are trained to reconstruct its input with encoding the input to a latent space via an *Encoder* and reconstruct the input from the latent space via a *Decoder*.

Training a conventional neural network on point clouds is not viable since point clouds are sets thus, the encoder has to be invariant of permutations in the input. To achieve that [Qi et al., 2017] introduce PointNet, a specific type of neural network that shares a Multi Layer Perceptron (MLP) between each point in the point cloud. They stack MLPs with different hidden layer sizes with non-linearities at each layer to increase model complexity to be able to represent low-level and high dimensional inputs. At the final stage of the Encoder, max pooling along the first dimension is applied to guarantee input combination invariance.

The decoder only consists of fully connected layers that at the end outputs a 1-D vector of size $N \times 3$. Like the encoder the decoder has to be permutation invariant as well. To overcome this issue authors of [Achlioptas et al., 2017], use *Earth Mover's*

Distance (EMD) to be able to measure the distance between two unordered point sets and introduce Point Cloud Autoencoder (PCAE). EMD measures how much "mass" should be transported from one distribution to other to make two of the distributions equal. Let S_1 and S_2 be two equally sized subsets, $|S_1| = |S_2|$ and $S_1, S_2 \subset R^3$. The EMD between these two subsets are defined as

$$d_{EMD}(S_1, S_2) = \min_{\phi: S_1 \to S_2} \sum_{x \in S_1} \|x - \phi(x)\|_2 \tag{3.32}$$

where ϕ is a bijection. EMD is differentiable everywhere, so when used as a loss, the gradient of EMD can be used for backpropagation in the neural network. In this work we trained PCAE with ShapeNet dataset [Chang et al., 2015] which consists of synthetic models of 57,000 objects from 55 different categories. We chose input size N to be 2048 points so we sample 2048 points from the objects surface randomly. The hyperparameters we used are same as in [Achlioptas et al., 2017]: Encoder of the PCAE consists of 5 layers of shared MLPs with hidden units of sizes 64, 128, 128, 256 and 128 where the last one is the size of the representation vector. Batch normalization is applied to the outputs of each layer in the encoder. Decoder takes input the representation vector of size 128 and consists of 3 layers of dense connections with sizes 256, 256 and $N \times 3$, without any batch normalization. Each layer of both the encoder and the decoder uses rectified linear units (ReLU) except the output layer of the decoder. L2 regularization is applied on weights with $\lambda = 10^{-3}$. Finally, PCAE is trained for 350 epochs with Adam weight update method with learning rate of 5×10^{-4}.

The representations learned by PCAE allowed us to extract skill agnostic features from the high dimensional raw point cloud data that the robot observes during the demonstration. These features are then used to train a goal model as discussed in Ch. 4.

3.6 Point Cloud Segmentation

In this work, we segment objects with solid (known) color from a Point Cloud (PC) scene captured with an RGB-D camera. This section will get into the detail about the segmentation of PC.

Since the skills teached in this work are tabletop manipulation skills we utilize PC segmentation for tabletop plane segmentation introduced by [Trevor et al., 2013]. The authors of [Trevor et al., 2013], map each point in an organized PC to an image-like structure denoted by $P(x, y)$. Aim is to demarcate P into set segments S. Each point on P has an integer label L i.e label of $P(x, y)$ is $L(x, y)$. This means that if $P(x_1, y_1) \in S_i$ and $P(x_2, y_2) \in S_i$ then $L(x_1, y_1) = L(x_2, y_2)$. The points are then compared with a comparison function C. C takes input two points $P(x_1, y_1)$ and $P(x_2, y_2)$ and outputs *true* if these two points are *similar*. The points are then labeled using C which they refer to as *Connected Component Algorithm* (CCA). There are several similarity metrics to be used in CCA introduced in [Trevor et al., 2013] which are provided below.

To detect planes on the PC, [Trevor et al., 2013] introduce *Planar Segmentation* method. This method extracts the plane using each point p's coordinates (x, y, z) and their normals n_x, n_y, n_z (p_n). In addition, they use perpendicular distance to the normal, denoted as n_d (p_{n_d}). Finally these values are used to come up with a comparison function C to be used in CCA where p_1 and p_2 are two different points taken from P:

$$C(p_1, p_2) = \begin{cases} true & p_{1_n} . p_{2_n} < \epsilon_n \;\&\&\; |p_{1_{n_d}} - p_{2_{n_d}}| < \epsilon_r \\ false & o/w \end{cases} \tag{3.33}$$

Where ϵ_n and ϵ_r are thresholds. Using this comparison function in CCA results in labeled plane segments. Segments with number of points greater than a threshold is selected as the planes. Other metrics like Euclidiean distance and color information can be used in C to segment planes according to their color values. For extracting objects, these planes are clustered to come up with complete solid-colored objects.

For more details see [Akgun, 2015]. In this work, these segments are then fed into PCAE explained in Sect. 3.5 to extract features from the object point cloud.

Chapter 4

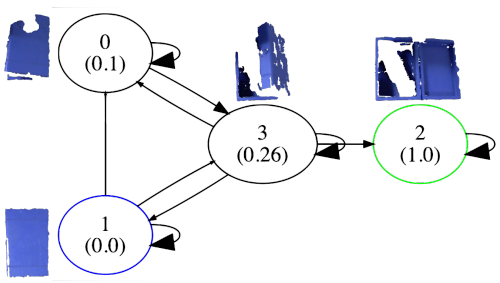
REWARD LEARNING

The main aim of this work is to develop a general-purpose reward learning approach that eliminates the need for reward engineering. Our approach uses the perceptual information obtained during user demonstrations to learn rewards. We achieve this by; (1) extracting low dimensional perceptual features from a vision sensor, (2) training an HMM, representing the goal model, with the extracted features and (3) converting this HMM to a POMRP. We would like to note that, (1) can be replaced with any feature extraction method that accommodates goal information.

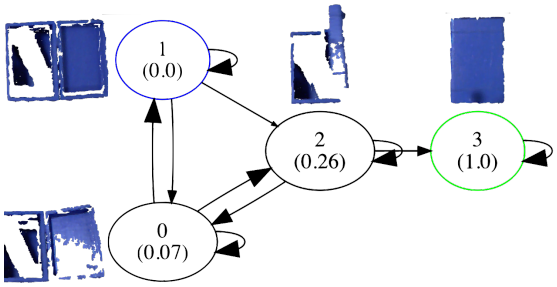
In this chapter, we detail the last step as a general framework. Other steps are discussed in Sect. 6.1. We later show the efficacy of this approach in robotic skill learning by using the learned rewards to improve a skill model learned with imperfect imitations.

As suggested in [Akgun and Thomaz, 2016], end-users focus on the goal of the skill rather than the quality of the motion to successfully realize it. This is our motivation to use the goal model to learn rewards. Therefore, similar to the authors of [Akgun and Thomaz, 2016], we train a Hidden Markov Model (HMM) with Gaussian emissions using the lower dimensional goal data obtained from demonstrations. However, our data comes from trajectories instead of keyframes. The learned HMM represents our goal model. Example goal models for the skills obtained from real world demonstrations can be seen in Fig. 4.1.

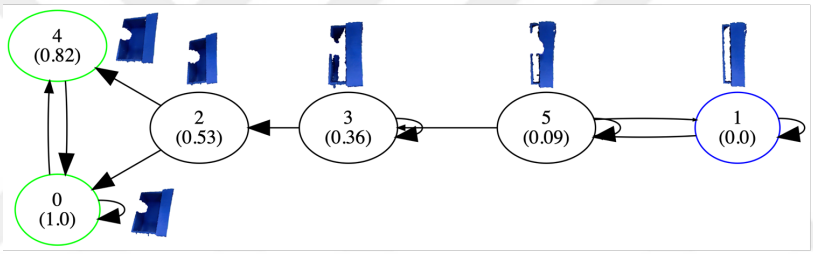
For monitoring the execution of a skill, the goal model takes the goal data of the execution as an input sequence and returns whether the execution is a *success* or a *failure*. We assume that each hidden state represents a *sub-goal*. The final HMM states of the provided demonstration sequences are set as terminal states. We



(a) Open Goal Model



(b) Close Goal Model



(c) Draw Goal Model

Figure 4.1: State transitions for the goal models, learned on the real robot, of Open (4.1a), Close (4.1b) and Draw (4.1c) skills and the representative object Point Cloud corresponding to each hidden state. Green nodes denote terminal states and blue nodes denote the initial states. Values in parentheses are the learned rewards. The size of the arrow heads denote the value of transition probabilities.

formulate the HMM with $\{k, S, P, \pi, T, \Phi\}$ as explained in Sect. 3.2 in detail.

Towards this end, we use the HMM to create a Markov Reward Process (MRP), i.e Markov Decision Process discussed in Sect. 3.3.1 without the action space. We calculate values for this MRP and finally convert the HMM to a Partially Observable Markov Reward Process (POMRP) along with the MRP values to compute the returns of a skill execution to be used in PS.

The first step is to create a Markov Reward Process (MRP) from the HMM. The HMM's probability table P describes a Markov Process (MP) on the HMM hidden

states. We assign a reward of +1 to each terminal state and a reward of 0 to all the remaining states to create a MRP. Since the skill execution must end, we treat this as a finite-horizon MRP. We use a Monte Carlo (MC) approach explained in Sect. 3.3.2 to calculate the values of each state, represented by $v(s)$. We randomly sample hidden state sequences of finite length from the HMM, using the prior probability vector π and the transition probability table P. Let $\mathbb{Q}^i = \{s_1^i, s_2^i, ..., s_t^i, ..., s_L^i\}$ be the i^{th} sampled hidden state sequence where $\forall s \in S$. The return of each state in such a sequence is calculated using Eq. 4.1, where γ is the discount factor and r_t is the reward of s_t (Eq. 4.2). The constant sequence length L is taken as the floor of the average length of the user demonstrations.

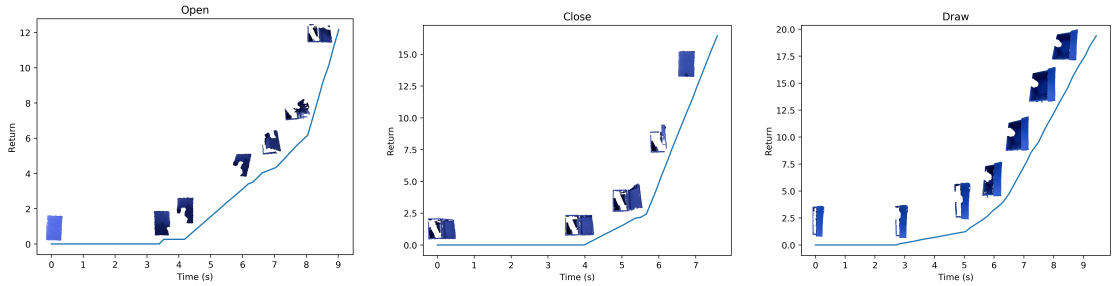
The value of each state is then approximated by the average of the returns and is calculated using Eq. 4.3 where n is the total number sampled hidden state sequences, $\mathbb{1}(\cdot)$ is the indicator function and n_s is the total number of times state s is encountered ($n_s = \sum_{i=1}^{n} \sum_{t=1}^{L} \mathbb{1}(s_t^i = s)$). Finally, the value of the terminal states are multiplied by a positive constant larger than one to emphasize the terminal rewards.

$$g^i(s_t^i) = \begin{cases} r_t^i + \gamma g^i(s_{t+1}^i), & t \neq L \\ r_L^i, & o/w \end{cases} \tag{4.1}$$

$$r_t^i = \begin{cases} +1, & s_t^i \in T \\ 0, & o/w \end{cases} \tag{4.2}$$

$$v(s) = \frac{1}{n_s} \sum_{i=1}^{n} \sum_{t=1}^{L} g^i(s_t^i) \mathbb{1}(s_t^i = s) \tag{4.3}$$

During a skill execution, the robot will not have direct access to the hidden states but gets observations in the form of goal data so we cannot directly use the MRP. As a result, we create a POMRP using the $\{k, S, P, \pi, T, \Phi\}$ of the HMM and the values from the MRP. This POMRP provides *dense* performance information about an execution, which is suitable to be used in a policy search approach. We do this by first calculating value information for each observation of an execution and then



(a) Open Returns (b) Close Returns (c) Draw Returns

Figure 4.2: Change of returns and corresponding segmented Point Clouds for Open (4.2a), Close (4.2b) and Draw (4.2c) in real life with respect to time.

summing these to get a total return. Using this as a performance metric in a policy search approach is equivalent to maximizing the average value.

Let $\mathbb{G} = \{g_1, ...g_t, ..., g_T\}, g_t \in G$ be a goal data sequence obtained in an execution. We can calculate the expected value of each g_t using the equation Eq. 4.4 where $P(s_t|\mathbb{G})$ is the posterior probability of being at hidden state s_t at time step t given the goal sequence, calculated using the forward-backward algorithm as explained in Sect. 3.2. For the purposes of our work, we refer to these as the *learned rewards* of the robotic skill. The final *return* of a skill execution is then the summation of these expected values for each perceptual observation in an execution, calculated using the Eq. 4.5.

$$h(\mathbb{G}, t) = \mathbb{E}[v(g_t)|\mathbb{G}] = \sum_{s_t \in S} v(s_t) P(s_t|\mathbb{G}) \tag{4.4}$$

$$R(\mathbb{G}) = \sum_{t=1}^{T} h(\mathbb{G}, t) \tag{4.5}$$

The output of Eq. 4.4, i.e. $h(\mathbb{G}, t)$, can be used as an immediate reward and the output of Eq. 4.5, i.e. $R(\mathbb{G})$, can be used as a black box optimization target for policy search, depending on the method. We use both immediate reward and black box optimization methods to update the skill and compare their performances in the following chapters of this work.

Chapter 5

SKILL LEARNING

Robots will face varying end-user types and most of them will not be robotics experts. As a result, their demonstrations will be sub-optimal and will often lead to failure if imitated directly. Furthermore, kinesthetic teaching alters the robot dynamics during the interaction which may also lead to failure. These are among the main motivations of using RL on top of LfD .

In this section, we explain how we parameterize the demonstrated skills and update the parameters using the learned rewards explained in Ch. 4 with RL.

5.1 Skill Representation

In this work, dynamical Movement Primitives (DMP) are used [Ijspeert et al., 2003] to parameterize the skill. In-depth discussion of DMPs are presented in Sect. 3.1. DMP consists of basis functions and each basis function has an associated weight w_b which are initially learned from demonstration and updated during policy search.

Before starting policy search a DMP is learned for each controlled dimension of the robot. In our work, we control the end-effector pose represented with a 3D vector for translation and a unit quaternion for rotation[1]. The learned DMPs are synchronized with the *phase variable*. As explained in Sect. 3.1, the initial value of the phase variable is 1 and it is gradually decreased to 0 using Eq. 3.5. This makes sure that the effect of the forcing function diminishes over time and enables the DMP to become an autonomous differential equation, guaranteeing stability if K and D are chosen appropriately.

[1]We treat the rotations as a 4D vector for learning and project them back to the unit quaternion space for skill execution.

The basis function centers and widths are the same for each controlled dimension. Therefore we treat their corresponding weights as vectors, i.e., given a basis b, its weight is represented with $w_b \in \mathbb{R}^7$.

5.2 Policy Search Approach: P^2-ES-Cov

We utilize multiple PS approaches to update the weights of the basis functions (the w_b's) with the learned rewards described in Ch. 4. Recall that we have both per-step reward and total return. The latter can be used with black-box methods. A black box evolution strategy based RL method for robotic skill learning called PI^2-ES employed in [Stulp and Sigaud, 2013] was shown to outperform two of the most common skill learning approaches on dummy tasks introduced in [Theodorou et al., 2010]: PoWER [Kormushev et al., 2010] and PI^2 [Theodorou et al., 2010]. PoWER and PI^2 discussed in Sect. 3.4 in detail. We agree with [Stulp and Sigaud, 2013] in that black box methods are better suited since we can't know for sure which time step leads to a successful execution.To show that this is the case, we utilize $h(\mathbb{G}, t)$ as a per-step reward for PI^2 and compare it with its black box variant, PI^2-ES using the total return $R(\mathbb{G})$. Our results strongly suggest that a black box approach is better. The details will be presented in Ch. 7.

In this work, we extend PI^2-ES to have a more principled adaptive exploration strategy. This section describes PI^2-ES and our extension, namely PI^2-ES-Cov.

The steps of PI^2-ES-Cov are summarized in Alg. 2 which are similar in many PS approaches including PI^2-ES. These algorithms sample a set of weights (line 5), execute the skill on the robot (line 6) and calculate the returns for each execution (line 7). These individual executions are called *roll-outs*. A pre-determined number of roll-outs is called an *episode*. The weights are updated based on the returns after each episode (lines 9-10). Depending on the exploration strategy, the sampling covariances may also be updated (line 11).

The weight update steps of PI^2-ES and PI^2-ES-Cov are the same. After an episode, the probability of each roll-out is calculated as the normalized natural exponentials

Algorithm 2 PI^2-ES-Cov

w_b: Initial weights for $b = 1 \dots B$
Σ_b: Initial exploration covariance for $b = 1 \dots B$
for all Episodes **do**
    for all Rollouts **do**
        Sample new weights $\hat{w}_b^c$ using w_b and Σ_b (Eq. 5.6)
        Execute the skill with $\hat{w}_b^c$ and get goal data $\mathbb{G}_k$
        Calculate the return $R_c = R(\mathbb{G}_c)$ (Eq. 4.5)
    Calculate the probability of a rollout P_c (Eq. 5.1)
    Update the weights (Eq. 5.2)
    Update the exploration covariances (Eq. 5.5)

of the returns, shown in Eq. 5.1, where C is the total number of rollouts per episode and $\mathbb{G}_c$ is the goal data obtained from rollout c. Then the weights are recalculated as the weighted average of the sampled weights as shown in Eq. 5.2.

$$\begin{aligned} R_c &= R(\mathbb{G}_c) \\ P_c &= \frac{e^{R_c/C}}{\sum\limits_c e^{R_c/C}} \end{aligned} \tag{5.1}$$

$$w_b = \sum_c P_c \hat{w}_b^c \tag{5.2}$$

The main difference between PI^2-ES and PI^2-ES-Cov is the exploration covariances used for the weight sampling step. PI^2-ES samples the weights based on Eq. 5.3 where Σ_b^0 represents the initial exploration covariance for basis function b. A separate DMP is learned for each demonstration and Σ_b^0 is taken as the covariance of the resulting DMP weights.

The initial w_b which is picked randomly from trained DMP weights is used to initialize the PS. The exploration step size σ_e in Eq. 5.3 is updated using Eq. 5.4 where σ_0 is initial exploration step size, E is the total number of episodes and U is episode at which σ starts decaying. The initial exploration step σ_0 is usually set to 1

for most applications.

$$\hat{w}_b \sim \mathcal{N}(w_b, \sigma_e \Sigma_b^0) \tag{5.3}$$

$$\sigma_e = \begin{cases} \sigma_0, & e \leqslant U \\ \sigma_0(1 - \frac{e-U}{E-U}), & o/w \end{cases} \tag{5.4}$$

The aforementioned exploration strategy is the same for PoWER, PI^2 and PI^2-ES. However, this exploration strategy has some drawbacks; σ_e needs to be decayed which introduces new parameters to be tuned and exploration does not adapt to the performance of the robot, leading to slower convergence.

We introduce PI^2-ES-Cov in order to alleviate these drawbacks in which we update the exploration covariance matrix (Σ_b) of each basis b during learning. The covariance matrices are initialized the same as before but are updated at the end of each episode using Eq. 5.5 which is the covariance update rule in PI^2-Cov [Stulp and Sigaud, 2012].

$$\Sigma_b = \sum_c P_c(\hat{w}_b^c - w_b)^T(\hat{w}_b^c - w_b) \tag{5.5}$$

Update rule given in Eq. 5.5 enables the robot to continue to explore the parameters which have higher impact on the return and enables us to remove the exploration step size parameter σ. With this direct update, the weight sampling step in PI^2-ES-Cov is given in Eq. 5.6.

$$\hat{w}_b^c \sim \mathcal{N}(w_b, \Sigma_b^c) \tag{5.6}$$

In the following parts of this work, we show that utilization of dense rewards outperforms sparse rewards for variety of PS methods. We later show that PI^2-ES further improve the results and PI^2-ES-Cov reduces the variance and increases convergence speed.

Chapter 6

EXPERIMENTS

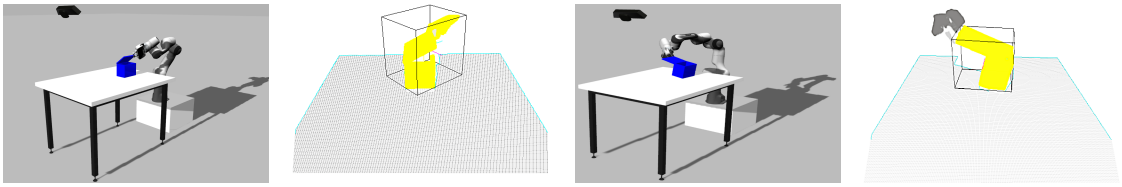
6.1 Setup

In this work, we use the Franka Emika Panda 7-DoF Robotic Arm as the manipulator and the Microsoft Kinect V1 as the depth camera, both in a robotic simulation environment (Fig. 6.1) and in the real world (Fig. 1.1). There is a table in front of the robot and the camera is faced towards the table. The underlying infrastructure is implemented using ROS.

6.2 Skills

We perform experiments on two skills in simulation, Open: Robot opens a box, Close: Robot closes a box and three skills on the real robot Open, Close and Draw: Robot pulls to open a wooden drawer. We assume there exists a solid-colored target object being manipulated and user provides the demonstrations kinesthetically, as shown in Fig. 1.1. The objects this work are a blue box for Open and Close and a blue drawer for Draw.

Number of demonstrations are 3, 2 and 2 for Open, Close and Draw respectively both for the simulation and the real robot. Screen shots of robot performing Open and Close from a third person and from robot's perspective in simulation environment can be observed in Fig. 6.1. Fig. 6.2 shows the objects and the skills being executed in the real world setting.



(a) Open (b) Open Robot's View (c) Close (d) Close Robot's View

Figure 6.1: The simulation environment and the segmented objects during skill execution. Figures 6.1b and 6.1d display the segmentation of the target objects and their bounding boxes. Negative of the images have been taken in figures 6.1b and 6.1d for visualization purposes.

6.3 Action Model

The action data is the robot's end-effector 7-D pose (3-D position, 4-D orientation as a unit quaternion) with respect to the *object frame*, collected continuously during the demonstration. This data is collected at 100 Hz and sampled uniformly to reduce the frequency to 10 Hz.

As discussed in Ch. 5, we represent the skills with DMPs and update DMP parameters with policy search using the learned rewards. In all of our experiments we set the spring coefficient K to 100 and chose damping ratio D to make the system critically damped; $D = 2\sqrt{K} = 20$. In the simulation experiments, number of DMP bases are 20 for Open and 10 for Close. In the real robot experiments, number of DMP bases are 15, 13 and 9 for Open, Close and Draw respectively. The DMP parameters used in these expemetiments are provided in Tbl. 6.1

6.4 Feature Extraction

The perceptual data is collected through the depth camera as Point Clouds. We extract the object from the scene by taking spatial clusters of similar known color and calculate its bounding boxes [Trevor et al., 2013] as explained in Sect. 3.6. The center of the bounding box is used to calculate the object frame. Fig. 6.1 shows the segmentation and the bounding boxes of the objects while the robot executing the skills in the simulation environment. Figures 4.1 and 4.2 shows different states of the

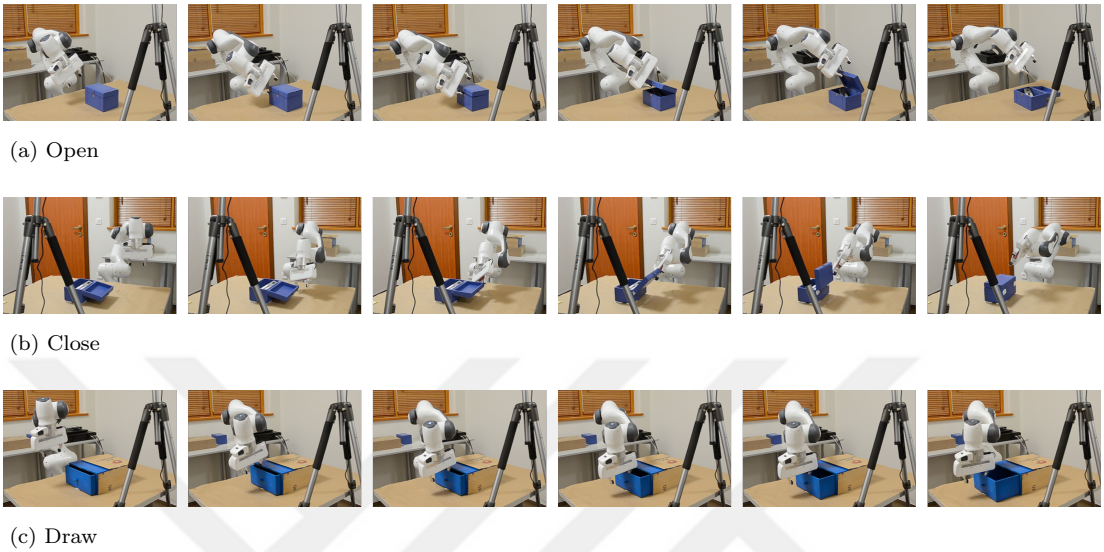
	Simulation		Real Robot		
	Open	Close	Open	Close	Draw
Number of Demonstrations	3	2	3	2	2
Number of Bases	20	10	15	13	9
Spring Coefficient (K)	100	100	100	100	100
Damping Ratio (D)	20	20	20	20	20

Table 6.1: DMP parameters used in simulation and real robot experiments.

segmented target objects.

The segmented points belonging to the object are used to extract features. The amount of data that can be collected during a LfD interaction prohibits learning in large state spaces which typically is the case with these sensors, even with segmentation. Thus, we use an unsupervised feature learning approach to get a lower dimensional state space for perceptual features. We have a two phase approach; (1) train a Point Cloud Autoencoder (PCAE) from scratch using the approach in [Achlioptas et al., 2017] to get a low dimensional skill agnostic representation and (2) apply Principle Component Analysis (PCA) to get an even lower dimensional skill specific representation. PCAE is discussed in Sect. 3.5 in depth. The output of these steps provides the goal data, containing information on the state of the object during the skill. These steps are depicted in Fig. 1.2.

The hyperparameters of the PCAE are the same as in [Achlioptas et al., 2017]. PCAE is trained on ShapeNet dataset [Chang et al., 2015] which consists of $51,300$ unique 3D models with 55 categories. PCAE takes 2048 three-dimensional points as input. We first segment the object from the organized point cloud obtained from the depth camera, using the same methodology as [Akgun and Thomaz, 2016]. Then, we sample 2048 points uniformly from the segmented object point cloud. Output of the *Encoder* of the PCAE is 128 dimensional. Later, we reduce the dimensionality to 8 with PCA. In all of our experiments, explained variance of PCA was greater than 90%.



(a) Open

(b) Close

(c) Draw

Figure 6.2: Robot executing the final policies of Open (6.2a), Close (6.2b) and Draw (6.2c) skills.

6.5 Goal Model

The goal model is an HMM with Gaussian emmissions trained with the features explained above. We train the goal HMM from multiple demonstrations using the Baum-Welch-Algorithm. Number of hidden states of the HMM is selected according to the Akaike Information Criterion. Initial values ($v(s)$) of hidden states are set to 0 and the number of the sampled hidden state sequences ($\mathbb{Q}^{\beth}$) to 100 to be used in reward learning explained in Ch. 4. In all of our experiments γ is set to 0.9. Example values of each HMM state, transition probabilities and states of HMM can be observed in Fig. 4.1.

6.6 Policy Search

We conduct simulation experiments with the Open and the Close skills for PoWER, PI^2, PI^2-Cov, PI^2-ES and PI^2-ES-Cov. Fig. 6.1 shows third-person frames in simulation. We compare the success rates of these methods for two reward types:

- Sparse: Robot receives a reward of +1 if the execution is successful, 0 otherwise.

- Dense: Robot receives the return $R(\mathbb{G})$ as explained in Ch. 4. For PI^2 and PI^2-Cov we used $f(G_{1:t})$ as per step reward acquired at time t.

For PoWER, we set the number of episodes (E) to 30 episodes and the number of importance samples to 6. For the other methods, number of episodes (E) and number of rollouts (C) are chosen as 5 and 6 respectively. For the methods that do not have adaptive covariances, we decrease the exploration size as given in Eq. 5.4 where U is set as $\frac{E}{4}$.

In addition, we perform real-world experiments with PI^2-ES-Cov with the Open, the Close and the Draw skills, see Fig. 6.2 for example executions.

Chapter 7

RESULTS

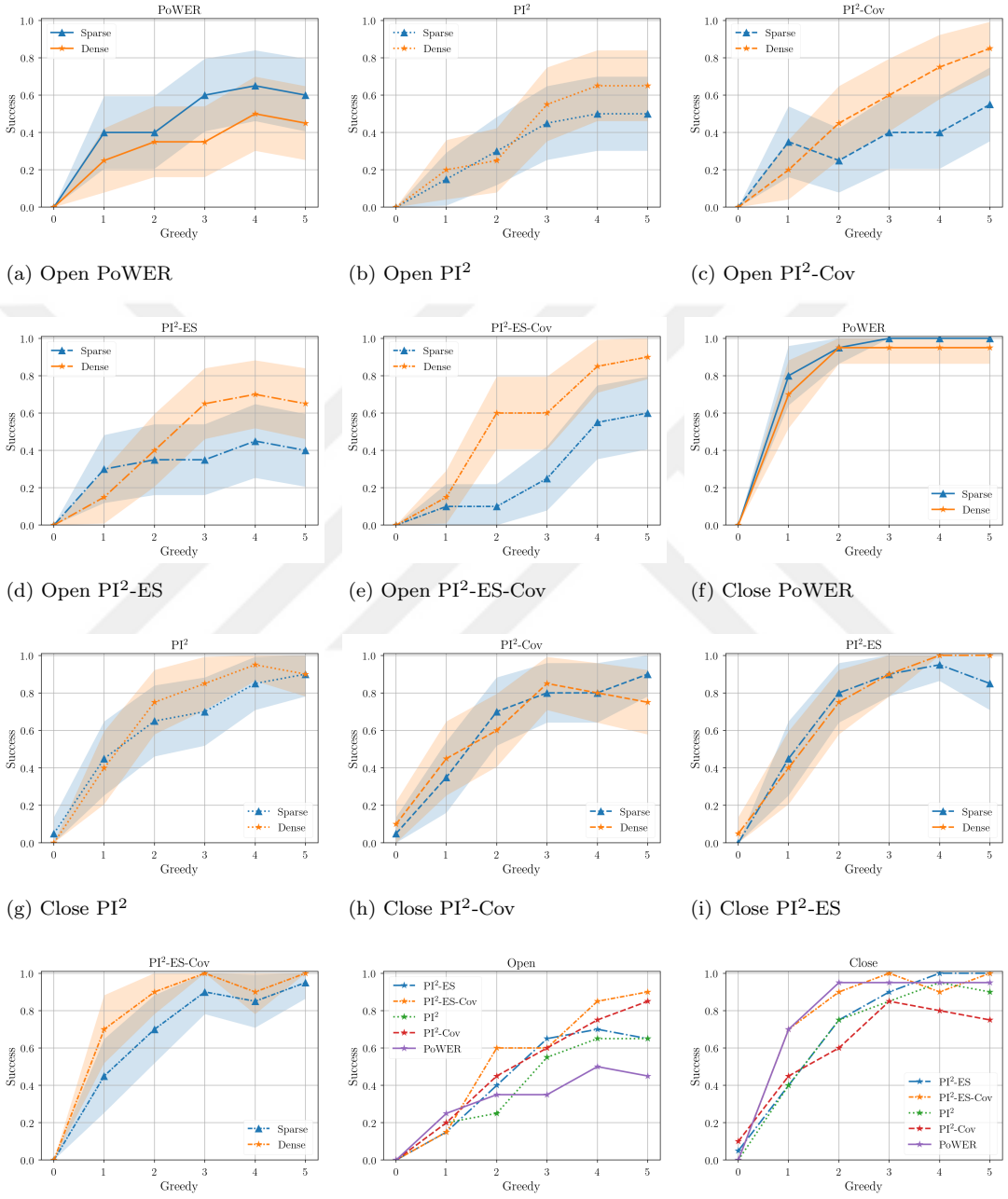
This chapter presents the results of the experiments detailed in Ch. 6. We utilize the success rates of the executions as our main performance metric. Reported success rates are from *greedy* executions, i.e steps without parameter exploration. For PoWER, greedy steps are performed after every 6 episode and for PI^2-based methods greedy steps are performed after each episode. For all PS methods, a greedy step is performed before the training starts to get initial success rates.

7.1 Simulation Results

We perform 20 simulation experiments for each combination of the reward type (sparse or dense) and the PS method, each starting from the same action model and the same goal model learned from initial demonstrations. Fig. 7.1 plots means and variances of success rates as lines and shaded regions respectively for both Open and Close. We compare the success rates of sparse and dense rewards, and the PS approaches.

7.1.1 Dense rewards are better than sparse rewards

As it can be seen in both skills, employment of dense rewards leads to better convergence and smaller variance. PoWER is an exception in both Open and Close but in Open, PoWER has very high variance and low success rate. We observe that Close is easier to learn and even with sparse rewards the robot learns the skill relatively well but it is learned faster and with higher success rates with dense rewards.



(a) Open PoWER (b) Open PI² (c) Open PI²-Cov

(d) Open PI²-ES (e) Open PI²-ES-Cov (f) Close PoWER

(g) Close PI² (h) Close PI²-Cov (i) Close PI²-ES

(j) Close PI²-ES-Cov (k) Open Mean (l) Close Mean

Figure 7.1: Simulation results. Success rates and variances of PoWER, PI², PI²-ES and PI²-ES-Cov for Open (7.1a-7.1e) and Close (7.1f-7.1j). Means success rates of PoWER, PI², PI²-ES and PI²-ES-Cov for Open (7.1k) and Close (7.1l) when dense rewards are used.

7.1.2 PI²-ES-Cov performs the best overall

To compare the methods, we plot the means of success rates when dense rewards are employed in Figs. 7.1k and 7.1l for Open and Close respectively. We observe that PI²-ES-Cov converges faster and to better success rates in both Open and Close. Furthermore, PI²-ES performs better than PI². Along with PI²-ES-Cov performing better than PI²-Cov, this suggests that a black-box approach is more favorable.

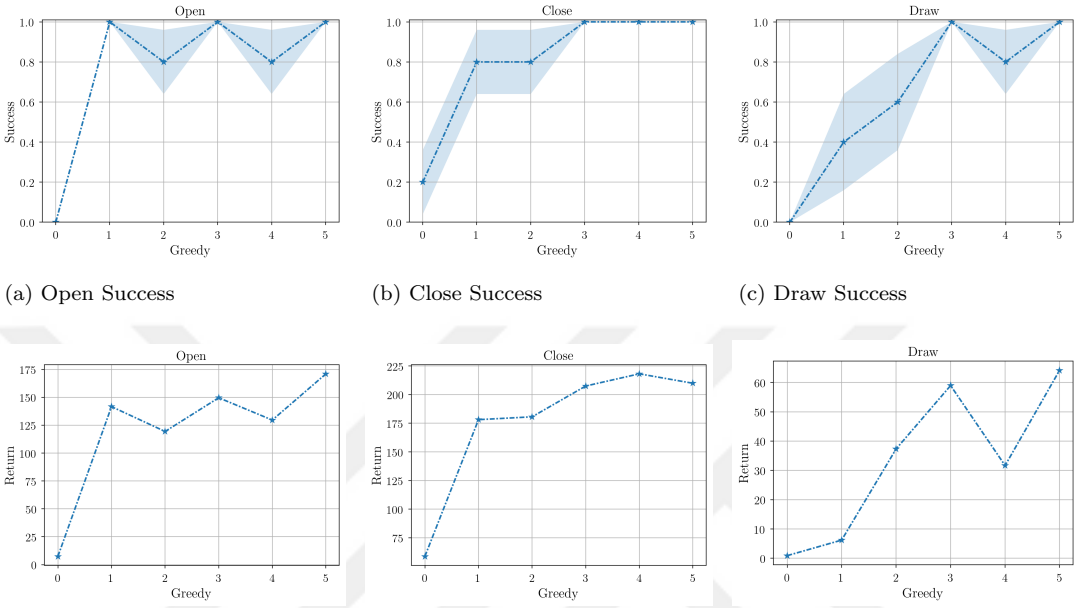
These results imply that; (1) learning with dense rewards improves the sub-performing skill models, even for different PS approaches, (2) the introduction of covariance updates to PI²-ES leads to faster convergence with higher success rates and lower variance, and (3) a black box optimization approach for skill learning may improve performance compared to approaches requiring per step rewards (PI² and PI²-Cov).

7.2 Real Robot Results

We further validate our results in a real robot setting for three skills: Open, Close, Draw using dense rewards with PI²-ES-Cov. We qualitatively look at the goal-models and present example cases for returns during executions. We then present the success rates and the returns for 5 experiments.

Fig. 4.1 shows the goal models learned from demonstration which are used to learn dense rewards. For Open and Close, the models are intuitive. They show the various states of the manipulated box being fully open and fully closed, and its lid being around half-way. For the Draw skill, there are seemingly redundant states but they are due to the heavy occlusion caused by the robot's end-effector. Fig. 4.2 shows the evolution of the total return, $R(\mathbb{G})$ (Eq. 4.5), for successful execution examples of each skill, along with corresponding segmented objects. As expected, the return increases steadily along with jumps as the object switches to a higher value states.

After the demonstrations, robot is trained 5 times from scratch using PI²-ES-Cov with learned rewards for 5 episodes and 6 rollouts. Fig. 7.2 plots the success rates and



(a) Open Success (b) Close Success (c) Draw Success

(d) Open Reward (e) Close Reward (f) Draw Reward

Figure 7.2: Real robot results for PI^2-ES-Cov. Success rates (7.2a-7.2c) and obtained returns (7.2d-7.2f)

returns for all three skills. Lines are the means and shaded regions are the variances of success rates. We did not put variances in the return plot for visualization purposes. Fig. 6.2 shows the robot executing the final policies for each skill.

The results show that the learned dense rewards along with PI^2-ES-Cov leads to successful skill learning. Initially, the success rates and the returns for all the skills are low. They increase with more episodes. The fluctuations are due to exploration (weight sampling). However, it can be clearly seen that at the end, for all of these three skills, the robot learns to successfully execute the action with very low variance.

Chapter 8

DISCUSSION

In this work, we introduce a novel framework for robot skill learning. Our framework learns rewards, negating the need for reward engineering, and improves bad action models. Our main contribution is learning rewards from very few demonstrations. Our evaluations with multiple PS approaches show that the goal model learned from low-dimensional perceptual features is a source of reward information and that dense rewards are better than sparse rewards. We further introduce an adaptive evolutionary policy search approach which leads to faster and better convergence. In this section, we discuss parts of our framework and the implications of our results.

8.1 Perceptual Features

Using a low-dimensional perceptual space enables us to learn a goal model from very few demonstrations. As explained in Sect. 6.4, we extract high level perceptual representations from raw Point Cloud data using a pre-trained Deep Neural Network, namely PCAE. We then perform PCA on top of PCAE to extract low dimensional goal data that explains the state of the object. This is important since we need to learn from low amount of data in real world robotic tasks and required data points increase with dimensionality.

8.2 Learning Rewards from the Goal Model

One of the greatest challenges in robotic skill learning with RL is engineering a reward function. In this work, we show that perceptual information obtained during demonstrations contain sub-goal and goal information of the skill which can be used to

extract dense rewards. This eliminates the need for hand engineered rewards, making robotic skill learning more viable for end-users.

As discussed throughout this work and by [Akgun et al., 2012a], users focus on the goal of the skill when providing demonstrations. Using this idea, we train an HMM as a goal model of the skill. Goal models can be used to monitor whether an execution is successful or not. This itself is a source of sparse reward information. However, we argue that this may not be informative enough to improve a skill. Fig. 4.1 shows the structure of the HMMs learned in the real-robot setting. The object states most similar to the emission means of the hidden-states are also depicted. We interpret these as the sub-goals of the skills and use them to extract dense rewards from the goal model.

To assign rewards to the sub-goals, we convert the HMM to a POMRP with a MC-based value learning method, as explained in Ch. 4. We densify the rewards by summing the expectations over the posterior probability of ending up at a sub-goal (hidden state) during an execution, which we call the return. Fig. 4.2 shows the change in return as the robot successfully executes the skill. We show that using the computed returns as dense rewards leads to better skill learning performance compared to using sparse rewards, in a simulated robotic environment on two skills with multiple PS methods. This implies that the efficacy of our reward learning approach does not depend a particular PS method.

8.3 Skill Learning with PI^2-ES-Cov

Policy Search approaches are widely used in robotic skill learning. Skills are mostly represented with a parameterized policy, which is initially learned from user demonstrations. Then this policy is improved on the real-robot. Initializing with user-demonstrations helps with the sample complexity problem by providing a good starting point. Nevertheless, it is preferable to converge to a successful skill with minimal number of trials and low variance.

We represent the skill policies with DMPs since they use relatively low number

of parameters and do not require vast amounts of data to be learned. We use PS approaches to update the DMP parameters for maximizing the learned rewards. For this, we take a black box optimization (BBO) approach as suggested by [Stulp and Sigaud, 2013] and introduce a covariance update-based adaptive exploration strategy as explained in Ch. 5. We call the resulting approach PI^2-ES-Cov.

Our results suggest that combining BBO with adaptive exploration is a favorable approach. We compare skill learning performance of PI^2-ES-Cov with PI^2, PI^2-Cov, PI^2-ES and PoWER in a simulated robotic environment for Open and Close skills. We show that using PI^2-ES-Cov leads to faster convergence, lower variance and higher success rates for both sparse and dense rewards. We also validate our approach in a real world skill learning with Open, Close and Draw skills.

Chapter 9

CONCLUSION

In this work, we developed a framework for robot skill learning suitable for end-users. Our framework can be summarized as having three phases; (1) learning from demonstration, (2) reward extraction and (3) policy search. The overall architecture of our frame work is depicted in Fig. 1.2.

In the learning from demonstration phase, the user provides demonstrations of the desired skill. We learn an action model and a goal model from these demonstrations. The action model is used to execute the skill and is learned from end-effector poses. In this work we used Dynamic Movement Primitives but any parameterized representation is applicable. The goal model is used to monitor the execution of a skill and is learned from low-dimensional perceptual features. We used Hidden Markov Models (HMM) with Gaussian emission probabilities to represent the goal model. The HMM choice has significance for the reward extraction phase.

In the reward extraction phase, we treat the hidden states of the HMM as the sub-goals and goals of the skill, and the transition structure as their partial order. We first convert the HMM to a finite horizon Markov Reward Process and calculate the values through a Monte Carlo approach. Then we combine the HMM and the values to obtain a Partially Observable Markov Hidden Process (POMRP). This POMRP can be used to calculate the expected sum of values, the total return, during an execution using the perceptual observations.

In the policy search phase, we used the total return as the metric to be maximized to improve the action model. This is equivalent to maximizing the average value. As the policy search algorithm, we introduced a black box approach with an adaptive exploration strategy and called it PI^2-ES-Cov.

9.1 Summary of Contributions

The main contribution of this framework is to learn rewards from very few demonstrations which negates the need for reward engineering, making it suitable to be used by end-users. The monitoring output of the goal model can already be used as a sparse reward signal. However, we showed that the learned dense rewards is more preferable for policy search in simulation for two skills, Open and Close, with multiple policy search approaches. We showed that PI^2-ES-Cov converges faster, with higher success rates and lower variance than several PS methods; PoWER [Kormushev et al., 2010], PI^2 [Theodorou et al., 2010], PI^2-Cov [Stulp and Sigaud, 2012] and PI^2-ES [Stulp and Sigaud, 2013]. This implies that combining BBO with adaptive exploration is beneficial for policy search. Finally we validated our findings in a real world robotic setting for three skills, Open, Close and Draw. Our framework was able to improve these skills from 0 success rate to 100% success rate with zero variance using learned dense rewards.

9.2 Future Work

The introduced robotic skill learning framework opens up interesting research directions. We argue that the skills Open and Draw are difficult to teach successfully since they require relatively precise motion during contacts and that object pose estimation is noisy. However, these skills are not dynamic, like balancing a pole. Applying the same ideas to dynamic or more complex skills would be a logical next research step. The long term vision that motivated the framework was enabling end-users to train their own robots. Even though it was out of scope of this work, since we concentrated on developing the methods, another immediate extension is to perform Human-Robot Interaction experiments to test the performance of our novel framework with non-experts.

BIBLIOGRAPHY

[Abbeel and Ng, 2004] Abbeel, P. and Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM.

[Achlioptas et al., 2017] Achlioptas, P., Diamanti, O., Mitliagkas, I., and Guibas, L. (2017). Learning representations and generative models for 3d point clouds. *arXiv preprint arXiv:1707.02392*.

[Akgun, 2015] Akgun, B. (2015). *Robots Learning Actions and Goals from Everyday People*. PhD thesis, Georgia Institute of Technology.

[Akgun et al., 2012a] Akgun, B., Cakmak, M., Jiang, K., and Thomaz, A. L. (2012a). Keyframe-based learning from demonstration. *International Journal of Social Robotics*, 4(4):343–355.

[Akgun et al., 2012b] Akgun, B., Cakmak, M., Yoo, J. W., and Thomaz, A. L. (2012b). Trajectories and keyframes for kinesthetic teaching: A human-robot interaction perspective. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, pages 391–398. ACM.

[Akgun and Thomaz, 2016] Akgun, B. and Thomaz, A. (2016). Simultaneously learning actions and goals from demonstration. *Autonomous Robots*, 40(2):211–227.

[Akgun and Thomaz, 2015] Akgun, B. and Thomaz, A. L. (2015). Self-improvement of learned action models with learned goal models. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5259–5264. IEEE.

[Andrychowicz et al., 2017] Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Pieter, A., and Zaremba, W. (2017). Hindsight experience replay. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 5048–5058. Curran Associates, Inc.

[Argall et al., 2009] Argall, B. D., Chernova, S., Veloso, M., and Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483.

[Bousmalis et al., 2017] Bousmalis, K., Irpan, A., Wohlhart, P., Bai, Y., Kelcey, M., Kalakrishnan, M., Downs, L., Ibarz, J., Pastor, P., Konolige, K., et al. (2017). Using simulation and domain adaptation to improve efficiency of deep robotic grasping. *arXiv preprint arXiv:1709.07857*.

[Calinon et al., 2007] Calinon, S., Guenter, F., and Billard, A. (2007). On learning, representing, and generalizing a task in a humanoid robot. *IEEE Transactions on Systems, Man, and Cybernetics*, 37(2):286–298.

[Chang et al., 2015] Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., et al. (2015). Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*.

[Chernova and Thomaz, 2014] Chernova, S. and Thomaz, A. L. (2014). Robot learning from human teachers. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 8(3):1–121.

[Deisenroth et al., 2013] Deisenroth, M. P., Neumann, G., Peters, J., et al. (2013). A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142.

[Duan et al., 2017] Duan, Y., Andrychowicz, M., Stadie, B., Ho, O. J., Schneider, J., Sutskever, I., Abbeel, P., and Zaremba, W. (2017). One-shot imitation learning. In *Advances in neural information processing systems*, pages 1087–1098.

[Finn et al., 2016] Finn, C., Levine, S., and Abbeel, P. (2016). Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning*, pages 49–58.

[Gu et al., 2017] Gu, S., Holly, E., Lillicrap, T., and Levine, S. (2017). Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE.

[Gu et al., 2016] Gu, S., Lillicrap, T., Sutskever, I., and Levine, S. (2016). Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838.

[Hester et al., 2017] Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Dulac-Arnold, G., et al. (2017). Learning from demonstrations for real world reinforcement learning. arxiv.

[Ijspeert et al., 2003] Ijspeert, A. J., Nakanishi, J., and Schaal, S. (2003). Learning attractor landscapes for learning motor primitives. In *Advances in neural information processing systems*, pages 1547–1554.

[Khansari-Zadeh and Billard, 2011] Khansari-Zadeh, S. M. and Billard, A. (2011). Learning stable nonlinear dynamical systems with gaussian mixture models. *IEEE Transactions on Robotics*, 27(5):943–957.

[Kober et al., 2013] Kober, J., Bagnell, J. A., and Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274.

[Kober and Peters, 2009] Kober, J. and Peters, J. R. (2009). Policy search for motor primitives in robotics. In *Advances in neural information processing systems*, pages 849–856.

[Konidaris and Barto, 2009] Konidaris, G. and Barto, A. G. (2009). Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in neural information processing systems*, pages 1015–1023.

[Konidaris et al., 2010] Konidaris, G., Kuindersma, S., Grupen, R., and Barto, A. G. (2010). Constructing skill trees for reinforcement learning agents from demonstration trajectories. In *Advances in neural information processing systems*, pages 1162–1170.

[Kormushev et al., 2010] Kormushev, P., Calinon, S., and Caldwell, D. G. (2010). Robot motor skill coordination with em-based reinforcement learning. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 3232–3237. IEEE.

[Kuindersma et al., 2010] Kuindersma, S., Konidaris, G., Grupen, R., and Barto, A. (2010). Learning from a single demonstration: Motion planning with skill segmentation. In *Proceedings of the NIPS Workshop on Learning and Planning from Batch Time Series Data*. Citeseer.

[Lee et al., 2012] Lee, S. H., Suh, I. H., Calinon, S., and Johansson, R. (2012). Learning basis skills by autonomous segmentation of humanoid motion trajectories. In *Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on*, pages 112–119. IEEE.

[Levine et al., 2016] Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373.

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

[Nair et al., 2017] Nair, A., McGrew, B., Andrychowicz, M., Zaremba, W., and Abbeel, P. (2017). Overcoming exploration in reinforcement learning with demonstrations. *arXiv preprint arXiv:1709.10089*.

[Niekum et al., 2013] Niekum, S., Chitta, S., Barto, A. G., Marthi, B., and Osentoski, S. (2013). Incremental semantically grounded learning from demonstration. In *Robotics: Science and Systems*, volume 9. Berlin, Germany.

[Niekum et al., 2015] Niekum, S., Osentoski, S., Konidaris, G., Chitta, S., Marthi, B., and Barto, A. G. (2015). Learning grounded finite-state representations from unstructured demonstrations. *The International Journal of Robotics Research*, 34(2):131–157.

[Peters and Schaal, 2008] Peters, J. and Schaal, S. (2008). Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190.

[Qi et al., 2017] Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017). Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660.

[Schaal et al., 2005] Schaal, S., Peters, J., Nakanishi, J., and Ijspeert, A. (2005). Learning movement primitives. In *Robotics research. the eleventh international symposium*, pages 561–572. Springer.

[Schulman et al., 2015] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897.

[Sermanet et al., 2017] Sermanet, P., Lynch, C., Chebotar, Y., Hsu, J., Jang, E., Schaal, S., and Levine, S. (2017). Time-contrastive networks: Self-supervised learning from video. *arXiv preprint arXiv:1704.06888*.

[Sermanet et al., 2016] Sermanet, P., Xu, K., and Levine, S. (2016). Unsupervised perceptual rewards for imitation learning. *arXiv preprint arXiv:1612.06699*.

[Stulp and Sigaud, 2012] Stulp, F. and Sigaud, O. (2012). Path integral policy improvement with covariance matrix adaptation. *arXiv preprint arXiv:1206.4621*.

[Stulp and Sigaud, 2013] Stulp, F. and Sigaud, O. (2013). Robot skill learning: From reinforcement learning to evolution strategies. *Paladyn, Journal of Behavioral Robotics*, 4(1):49–61.

[Theodorou et al., 2010] Theodorou, E., Buchli, J., and Schaal, S. (2010). A generalized path integral control approach to reinforcement learning. *journal of machine learning research*, 11(Nov):3137–3181.

[Trevor et al., 2013] Trevor, A., Gedikli, S., Rusu, R., and Christensen, H. (2013). Efficient organized point cloud segmentation with connected components. In *3rd Workshop on Semantic Perception Mapping and Exploration (SPME), Karlsruhe, Germany*.

[Williams, 1992] Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.