Keyframe Demonstration Seeded and Bayesian Optimized Policy Search

by

Onur Berk Töre

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 Science and Engineering

KOÇ ÜNİVERSİTESİ

September 1, 2022

Keyframe Demonstration Seeded and Bayesian Optimized Policy Search

Koç University

Graduate School of Sciences and Engineering

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

Onur Berk Töre

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:

_____

Assist. Prof. Barış Akgün (Advisor)

_____

Assoc. Prof. Emre Uğur

_____

Assoc. Prof. Nazım Kemal Üre

Date: _____

To my family, friends and professors

ABSTRACT

Keyframe Demonstration Seeded and Bayesian Optimized Policy Search
Onur Berk Töre
Master of Science in Computer Science and Engineering
September 1, 2022

Reinforcement learning (RL) is a promising approach to endow robots with skills. However, RL requires many trials to get satisfactory results. Learning from Demonstration (LfD) seeded RL alleviates this problem by learning an initial skill from human demonstrations. Nevertheless, this approach still requires robots to perform a non-trivial amount of trials. In this thesis, we develop an approach to further reduce these for manipulation skills with perceptual goals. Our main contributions are (1) an algorithm to focus the exploration by using the learned relationship between action and perception and a (2) Black-Box RL Policy Search (PS) method that improves upon the popular Policy Improvement with Path Integral (PI^2) algorithm, called the Bayesian Optimized PI^2 (BO-PI^2), that uses reward predictive UCB-type exploration.

Our underlying LfD framework utilizes a Dynamic Bayesian Network (DBN) learned from keyframe demonstrations to jointly model the action (end-effector pose) and the goal (object-specific perceptual data) of the skill. The action part is used to generate robot trajectories, and the goal part is used to monitor the success of trajectory executions and to create a Partially Observable Markov Reward Model in order to learn rewards.

BO-PI^2 is used to improve the action part of the DBN with trial-and-error using the learned returns. The novelty of BO-PI^2 comes from its exploration strategy. The coupling between the action and the goal is used to pick the part of the model to focus on to reduce the effort, in a sense to solve the credit attribution problem. After picking the part to focus on, BO-PI^2 samples trajectories from the action model to get rollouts, which is typical of PS approaches. In addition, BO-PI^2 uses a Gaussian Process (GP) to learn local returns from these rollouts, which is improved with each executed trajectory. The next samples are selected by utilizing an Upper Confidence Bound (UCB) approach, using the predicted return and uncertainty of

the possible candidate points. This is in contrast to random sampling, used in most PS approaches. BO-PI^2 also utilizes a skill success based termination criteria, using the goal model to monitor success autonomously.

We evaluate BO-PI^2 with expert and non-expert keyframe demonstrations for three skills. In the expert case, the models are perturbed so that the initial skill execution starts from a failure condition. In the non-expert case, we pick skill models that fail to begin with. We test our approach against the current state-of-the-art PI^2-ES-Cov algorithm using three metrics: (1) skill success rate, (2) total accumulated reward, and (3) number of trials. In both the expert case and the non-expert case, on average, our approach performed better than the baseline on all three metrics. Our results show that utilization of keyframes allows us to focus on failed sub-goals rather than the entire trajectory, and combined with reward predictive exploration strategies, are beneficial to improve RL performance and reduce the number of trials to endow robot arms with real-life manipulation skills.

ÖZETÇE

Anahtar Nokta Gösterimlerinden Desteklenerek Başlatılmış ve Bayessel Optimize Edilmiş Politika Öğrenimi

Onur Berk Töre

Bilgisayar Bilimleri ve Mühendisliği, Yüksek Lisans

1 Eylül 2022

Pekiştirmeli öğrenme (PÖ), robotlara beceriler kazandırmak için gelecek vaadeden bir yaklaşımdır. Bununla birlikte, PÖ kullanarak iyi sonuçlar elde etmek için birçok deneme yapmak gerekir. Gösterimlerden Öğrenme (GÖ) destekli PÖ, insan gösterimlerinden başlangıç becerisi öğrenerek bu sorunun etkisini azaltır. Ne yazık ki, GÖ destekli PÖ hala daha robotların beceri öğrenmesi için fazla denebilecek miktarda deneme gerektirir. Bu çalışmada, bu deneme sayısını daha da azaltmak için bir yaklaşım geliştirdik. Başlıca katkılarımız, (1) keşfe odaklanan bir algoritma ve (2) Yol İntegralleriyle Politika İyileştirme yöntemini geliştiren kapalı kutu politika bulma yöntemi olan Bayessel Optimize Edilmiş Politika Öğrenimi methodur (BO-PI^2).

GÖ yapımız, becerinin hareketini (robot kol pozu) ve hedefini (nesneye özgü algısal veriler) birlikte modellemek için anahtar nokta gösterimlerini ve bunlardan öğrenilen bir Dinamik Bayessel Ağ'dan (DBA) yararlanır. Bu ağın hareket bölümü robot hareketini modellemek için, hedef bölümü ise hareketin başarısını izlemek ve Kısmen Gözlenebilir Markov Ödül Modeli oluşturarak ödül sinyali oluşturmaya yarar.

BO-PI^2, öğrenilen ödül sinyali sayesinde DBA'nın hareket bölümünü geliştirir. BO-PI^2'nin yeniliği, keşif stratejisinden gelir. Hareket ve hedef arasındaki bağlantı deneme miktarını azaltmak için yeteneğin odaklanılacak kısmını seçmek için kullanılır. Bu yaklaşım, bir anlamda kredi atama sorununu çözmeye benzer. Odaklanacak kısım seçildikten sonra, BO-PI^2, PÖ yaklaşımlarının tipik olarak kullandığı denemeleri üretmek için hareket modelini kullanır. BO-PI^2, bu işlem sırasında yürütülen hareketler ile gelişen kümülatif ödülleri öğrenmek için Gaussal Süreç modeli (GS) kullanır. BO-PI^2 bir sonraki hareket noktalarını Üst Güven Sınırı (ÜSB) algoritmasını kullanarak seçer. Bu algoritma adayların tahmin edilen ödül ve belirsi-

zliğine dayanır. Bu, çoğu PÖ yaklaşımında kullanılan rastgele hareket yaratmaktan farklıdır. BO-PI^2 ayrıca yeteneği öğrendiğinde erken durmak için hedef modeline dayanan bir sonlandırma kriterinden faydalanır.

Bu çalışmada BO-PI^2'yi 3 beceri için uzman ve uzman olmayanlar alınmış anahtar nokta gösterimleriyle test ettik. Uzmanlar tarafından verilmiş gösterimlerden öğrenilen hareket modelleri başarılı oldukları için, elle yapılan müdahalelerle başarısız hale getirdik. Uzman olmayan kişilerden alınmış gösterimlerde ise başlangıçta başarısız olan haraket modellerini seçtik. Yaklaşımımızı şu zamana kadar ki en başarılı sonuçları elde etmiş PI^2-ES-Cov algoritmasına karşı üç metrikde karşılaştırdık, bunlar: (1) beceri başarı oranı, (2) toplam birikmiş ödül ve (3) deneme sayısı. Hem uzman, hem de uzman olmayan durumlarda, yaklaşımımız ortalama olarak her üç metrikte de PI^2-ES-Cov dan daha iyi performans gösterdi. Sonuçlarımız, anahtar nokta gösterimlerinin bütün gezingeden ziyade başarısız olan kısımlara odaklanmamıza izin verdiğini ve bunun ödül tahminine dayalı keşif stratejileriyle birleştirildiğinde, PÖ performansını iyileştirdiğini ve robot kollarını gerçek hayatta kullanılacak yeteneklerle donatmak için deneme sayısını azaltmaya faydalı olduğunu gösteriyor.

ACKNOWLEDGMENTS

I am grateful to have Barış Akgün as my advisor. His supervision through my journey was an ideal mix of guidance and freedom. I admire his vast knowledge of numerous topics and intellectual capabilities. His traits allowed me to enhance my understanding of subjects that make this work possible and help me think and question different perspectives. The skills I learned by being his student will affect every aspect of my life. Before I started, I would not imagine a master's degree could have such effects on one's life.

I was lucky to have such a great labmate, Farzin. We worked throughout our degrees, overcame challenges, and helped each other on the way. I am always impressed by his hard-working capabilities and focus, even on the small details.

I am grateful to Göksu, Buğra, Fiona, Gökçan, and Merve for the cheerful time we spent together and for their support. The relationships we built turned my master into a much more joyful and memorable experience. I believe our friendship will extend this short amount of time.

Last but not least, to my family, your caring and support make all of this work possible. To my sister, whom I believe I can relate to most, thanks for being there whenever I needed even if I could not keep in touch. I am forever indebted to them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

BBO	Black Box Optimization
BO-PI^2	Bayesian Optimized PI^2
CEM	Cross-Entropy Method
CMA-ES	Covariance Matrix Adaptation Evaluation Strategy
DBN	Dynamic Bayesian Network
GP	Gaussian Process
HMM	Hidden Markov Model
IRL	Inverse RL
LfD	Learning from Demonstration
MP	Markov Process
MRP	Markov Reward Process
PCA	Principal Component Analysis
PCAE	Point Cloud Auto Encoder
PI^2	Policy Improvement with Path Integrals
PI^2-CMA	PI^2 with Covariance Matrix Adaptation
POMRP	Partially Observable Markov Reward Process
PS	Policy Search
RL	Reinforcement Learning
UCB	Upper Confidence Bound

Chapter 1

INTRODUCTION

Robotic manipulators are finding their way to more applications and fields on top of their classical caged factory use cases. Some of these new applications and fields include collaborative assembly, product testing, quality control, food preparation, logistics, warehouses, and construction. This increase in applications brings about a robotic programming challenge. Robots are typically programmed by experts, which is costly. This is exacerbated if the robots need to be re-programmed after their deployment. There are multiple paradigms to program robots, or as we call it endow robots with skills. The main ones, not necessarily mutually exclusive, include (1) direct programming, (2) learning from demonstration (LfD, also known as imitation learning) and (3) reinforcement learning (RL).

Direct programming has been successful for highly controlled environments (e.g. factory floors) and relatively simple tasks (e.g. vacuum cleaning), however, most applications require a lot of expert effort (programming itself, setting up suitable work cells within factories etc.). Recent robotic systems involving collaborative manipulators are shipped with better programming interfaces to alleviate this challenge, yet they still require relatively controlled environments and some amount of expertise.

LfD aims to limit the engineering effort and make it easier to deploy robots. However, LfD suffers from the distribution shift problem, especially if perceptual inputs are used, where the state distribution during deployment is not the same distribution observed during training, a prevalent problem in dynamic and uncertain environments. One way to deal with this is to collect a lot of data, which is only feasible for very high-value applications (e.g. autonomous driving). Another challenge of LfD is to get good demonstrations from the end-users. Non-roboticist

provide noisy data which limits the success of the learned models.

In RL, the requirement is to formulate the problem (reward, state and action spaces) and provide a means to perform trials (offline RL does not have the last requirement but then it requires a lot of data). The robot can then learn the skill with trial-error and even improve itself during deployment. However, formulation requires expertise and not all skills have easy to define rewards. Furthermore, high sample complexity of RL leads to long and potentially unsafe trials. Using simulators help with the latter issues but results in the simulation to real transfer challenge.

LfD seeded RL alleviates the sample complexity problem by learning an initial skill from human demonstrations. The idea is that an initial model helps start RL from a favorable point so that the exploration effort is reduced. Nevertheless, this approach still requires robots to perform a non-trivial amount of trials. On top of this, the challenges of getting good demonstrations from users and reward engineering remain.

In this thesis, we are interested in endowing robotic manipulators with skills that have perceptual (sub-)goals. We introduce a LfD seeded RL approach to tackle the aforementioned challenges within this context. Our developed approach:

- Works with keyframe demonstrations since they were found to be easier to use by non-roboticist users [3].

- Learns a probabilistic skill model that jointly represents the end-effector motion and coarse perceptual changes of the object of interest with this motion. This model is learned from keyframe demonstrations.

- Uses the perceptual part of the skill model to learn rewards using an existing approach to remove the need for specifying rewards.

- Leverages the model structure, the relationship between action and perception, to focus the exploration to reduce number of trials.

- Performs policy search with Upper Confidence Bound (UCB) based exploration to update skill model parameters to further reduce the number of trials.

We use a Dynamic Bayesian Network (DBN) as our probabilistic skill model. The DBN in our LfD framework is very similar to a Hidden Markov Model (HMM) but it has two hidden states to represent the action and the perceptual (sub-)goals. The action part models the end effector motion and the goal part models the perceptual states that the object of interest goes through during user demonstrations. The transition model of our DBN is composed of two tensors, depicting the probability of the next action/goal state with the current action and goal state. The emission space of the action part is the end-effector pose with respect to the object of interest. The goal emissions are extracted from the segmented object point cloud data in two steps. The first step is to pass the object points through a pre-trained point cloud auto-encoder to get skill agnostic features. The second step is to perform PCA on the demonstrations to get lower dimensional skill specific features. For executing the skill, we use the most likely state-paths and the action emission models to generate a trajectory. We adapt the reward learning of Cem et al. to be used with the perceptual part of our DBN to learn rewards [5], which can be interpreted as an Inverse RL (IRL) method.

The RL part follows the LfD and the IRL. We update the emission parameters of the action part, which can be interpreted as our policy. The DBN model provides a relationship between the robot motion and the skill sub-goals. We use this to select action state pairs to concentrate the RL effort on by figuring out the part of the motion that does not reach the desired sub-goals. Afterwards we use a novel Black-Box RL Policy Search (PS) method to update the parameters. This PS method improves upon the popular Policy Improvement with Path Integrals (PI^2) algorithm, called the Bayesian Optimized PI^2 (BO-PI^2). BO-PI^2 models the expected return with Gaussian Processes (GP) to perform Upper Confidence Bound (UCB) based exploration. BO-PI^2 is wrapped within an algorithm to pick the action state to concentrate on, within the previously selected pair. Focusing on a single state drastically reduces the data requirements of the GP. We perform each part of the approach until the skill is learned or maximum number of trials is reached.

We test our approach in two real-world setups, both involving three skills. It is compared against the current state-of-the-art PI^2-ES-Cov algorithm using three

metrics: (1) skill success rate, (2) total accumulated reward, and (3) number of trials in both setups. Both BO-PI^2 and PI^2-ES-Cov utilize the aforementioned state selection algorithms. The first evaluation setup includes expert demonstrations and manual perturbations to learned models to make the initial action model start from failure with a good learned reward model. In this setup, BO-PI^2 outperforms PI^2-ES-Cov for almost all the skills. On average, BO-PI^2 (1) reaches 95% skill success vs 40%, (2) accumulates 2.5 more returns, and (3) terminates faster with 4.5 number of episodes vs 5.7. The second evaluation setup includes an HRI study to collect data from non-roboticist users. We pick the cases where the initial skill is not successful to perform the evaluations. In this setup, BO-PI^2 is either better than PI^2-ES-Cov or they perform about the same. On average, BO-PI^2 (1) reaches 86.67% skill success vs 63.33%, (2) accumulates 1.5 more returns, and (3) terminates faster with 3.7 number of episodes vs 4.3.

These results show that our keyframe demonstration seeded policy search approach can be utilized to learn real-life manipulation skills with perceptual sub-goals. Within this context, our contributions are as follows:

- Algorithmic methods to concentrate the RL exploration effort using the action and goal relationship.

- The BO-PI^2 algorithm that combines black-box policy search and UCB-type reward predictive exploration.

1.1 Thesis Statement

The connection between end-effector motion and perception, learned from keyframe demonstrations, can be used to focus the exploration effort in Reinforcement Learning. Reward weighted policy search approaches can further use the benefits of this relationship and reward predictive exploration techniques to improve skill performance.

1.2 Thesis Organization

This thesis includes six chapters: the first chapter, the problem, and the thesis statement. The second chapter describes skill demonstration methods and reviews the policy search and reward predictive exploration techniques. Chapter 3 explain the underlying approach used to learn the model of user demonstrations. The next chapter describes BO-PI^2, a novel black-box policy search approach used to learn complex robotic policies. Chapter 5 shows the result of the experiments we did to test our approach and compares it with the previous state of the art. Chapter 6 briefly mentions a few critical points and suggests future directions. Finally, Chapter 7 concludes our overall framework.

Chapter 2

LITERATURE REVIEW

There are many existing work relating to learning from demonstration and reinforcement learning for robots. We focus on the ones that utilize keyframes, relevant policy search methods and perform reward predictive exploration in this chapter.

2.1 Skill Demonstration Methods

We focus on two types of demonstrations; (1) trajectories where the data from teacher (demonstrator) are sampled with high frequency and (2) keyframes where the data is only collected for points that teacher demarcates [6]. In a sense, keyframe demonstrations end up as sparse (along the time dimension) trajectories.

Majority of the learning from demonstration approaches in the literature utilize trajectory demonstrations. Furthermore, these methods are mainly evaluated with teachers that are familiar with robots and/or the underlying learning approach. Such teachers typically provide good quality demonstrations that lead to successful skill models. Unfortunately, humans who do not have the familiarity or the expertise, which we call naive-users, do not always give good quality demonstration and as such, the aforementioned approaches fail. Many of these failures originate from the difficulties naive users face during teaching [3].

Difficulties can originate from the robot. High-DoF robots are harder to give demonstrations due to increased complexity. Furthermore, demonstrations given with such robots have more noise than their simpler counterparts. The demonstration noise decreases the quality of the learned model. Collecting more demonstrations, i.e., increasing the demonstration time, may solve the noise issue. However, naive users tend to lose focus and motivation after several demonstrations. As an alternative, keyframe demonstrations allow naive users to provide higher quality and

lower noise demonstrations since the noisy portions are not collected. However, the sparsity of the keyframe demonstrations and the lack of velocity information make it harder for the algorithms to learn complex skills.

Keyframes can be interpreted as the consecutive sub-goals of the demonstrated skill. Concentrating only on the sub-goals rather than entire trajectory increases the generalization capacity of the methods and decreases the amount of needed data. This properties allow them to be utilized in broad range of areas.

Luo et al. [29] used improve the accuracy of the keyframe based SLAM approaches. Their approach utilizes non-keyframe information to further refine the MAP based SLAM. Proposed method, both benefits the sparse information matrix due to keyframes and knowledge from the non-keyframe information. The authors tested the approach indoor environment and showed they significantly reduced the number of keyframes and landmarks without compromising the accuracy.

Previous works utilized keyframes for visual purporses as well. Nerurkar et al. [28], used extracted keyframes to create semantically meaningful information from the videos. Authors used the insights gathered from the ground truth keyframe data to develop and automatic extraction framework. They showed that created method shows promising results.

Keyframes can be used to improve skill execution in LfD. The authors of [30] uses keyframes with conceptual constraints to repair the learned skills. The method tested with pick-and-pour and placements task to show that Concept Constrained Learning from Demonstration algorithm can improve low-quality demonstration data. Their approach showed only single demonstration is enough to improve skill performance.

In kinesthetic teaching keyframes can be modeled with Hidden Markov Models (HMM) [4]. Hidden states of the HMM represent the true sub-goals, and emissions are the keyframes. Akgun et al. [7] use two HMMs to model the changes in the object of interest and end-effector movement. They called these two models action (models the end-effector pose during the demonstration) and goal (models the perceptual representation of the object of interest). They use the action HMM to generate end-effector trajectories for skill execution and the goal HMM to monitor this execution.

Their approach using sub-goals with perception information decreases the amount of needed goal data due to the sparsity of keyframes. In contrary, dense trajectories would need more data to be able to distinguish sub-goals.

The execution monitoring of the goal HMM can be utilized for skill improvement as well. However, this gives a sparse signal for reinforcement learning, making it challenging the learn. Cem et al., converts the goal HMM to a Partially Observable Markov Reward Process (POMRP) to provide denser rewards [5].

These HMM-based approaches do not utilize the connection between the action and goal model to improve the skill learning approach further. In this thesis, we learn a single probabilistic model to jointly represent the action and goal. We then utilize this connection to focus the reinforcement learning effort on the failed sub-goals to reduce the amount of exploration.

2.2 Policy Search

Policy search (PS) is a common reinforcement learning method in robotics which the aim is to find a policy that maximizes the reward outcome. In recent years, model-free black-box optimization (BBO) PS methods have outperformed the traditional policy search approaches [12, 13]. These methods does not learn the model of the robot dynamics thus named model-free, and have three properties of the BBO methods regarding the policy space and objective function; (1) There are no presumptions concerning the policy search space and objective function, (2) The only property of the objective function is to generate one scalar reward, and (3) no other assumptions are made, such as its differentiability or continuity.

One of the first BBO approach, Cross-Entropy Method (CEM) [14], iteratively perturbs policy parameters to generate better policies. During the policy update, CEM sorts the parameters with respect to generated trajectory cost and uses reward-weighted averaging to select the following policy using the top M parameters. Reward-weighted averaging is especially beneficial when the reward signal is noisy. Averaging top M parameters decreases the effect of outliers.

Matrix Adaptation Evaluation Strategy (CMA-ES) [15] improves CEM by stor-

ing step size and covariance matrices. Furthermore, CMA-ES allows probabilities to be any scalar reward. This changes resulted in a great improvement on convergence speed. However, CMA-ES requires hand-tuned parameters which requires additional effort for adaptation.

Derived from stochastic optimal control, PI^2 [17] samples from a Gaussian Distribution and use probability weighted averaging just like CMA and CMA-ES. These similarities are used in Path Integral Policy Improvement with Covariance Matrix Adaptation (PI^2-CMA) [16] approach which have the structure of the PI^2 and covariance matrix adaptation of CEM. Authors get rid of the hand-tuned parameter of the CMA-ES and use full covariance matrix adaptation which resulted with increased learning speed [27].

PI^2-CMA requires the reward signal to be dense and available for each time step. Unfortunately, this is not possible for all tasks and in many skill executions it is difficult to quantify the exact moment that leads to skill success. On the other hand, CMA-ES [33], comply with the BBO properties and have less strict requirements. As an alternative, PI^2-ES [32], a special case of CMA-ES, combines the PI^2 with BBO and removes the strict requirements.

Finally the PI^2-ES-Cov [5] approach combines the adaptive exploration strategy of the PI^2-CMA and the black-box nature of the PI^2-ES. The authors tested PI^2-ES-Cov against the PI^2, PI^2-CMA, PI^2-ES on convergence speed and skill success and showed that black-box version of the algorithm outperforms all counterparts.

The algorithm we propose is BBO-type PS approach, and is similar to the mentioned methods due to the complex and noisy perceptual reward function. We combine reward-weighted averaging and adaptive exploration strategy of PI^2-ES-Cov with a reward-predictive strategy to guide the exploration.

2.3 Reward Predictive Exploration Strategies

Models of the expected reward can be used as a promising data-efficient approach to learning new policies [13]. These approaches refine the reward function with each new sample and choose the following parameter depending on the reward model.

Bayesian Optimization (BO) is a popular choice among these approaches. Bayesian optimization [18] searches for maximum objective function value using previous samples. These methods are particularly well-fitted when sampling for a objective function that is costly to evaluate, such as in robotics.

Bayesian Optimization techniques can drastically reduce to find a good-enough policy. Cully et al. [20] showed that, GP can be used to guide the exploration effort of the robot parameters to recover the speed of the damaged robot. Utilization of the Bayesian Optimization allowed robot to learn walking just under 2 minutes.

Scalability problems of the BO restrict the dimensionality of the policy. Due to this, previous work is only possible with well-chosen policy structures with small policy spaces. Furthermore, these approaches do not consider the complex, noisy reward functions, where small changes in parameter spaces can have drastic effects.

As an alternative, methods that combine the reward-weighted averaging and exploration strategies of BBO with the reward model of the BO provide efficient and safe policy updates. One such approach Black-DROPS [34] uses Gaussian Process (GP) for both robot dynamics, and reward model. While this approach is data efficient, it is not scalable for learning complex and high-dimensional dynamics (e.g., when perception is involved). It also has the negative sides of the dynamic modeling approaches which are increase in required data due to incorporation of the dynamics model of the robot and error accumulation with each time step. These properties makes this method unsuitable for our needs.

Our idea, using reward modeling in episodic black box policy search, can be seen as using the best parts of episodic BBO and BO approaches, to increase the data efficiency while maintaining the benefits of reward-weighted averaging and elasticity of the black-box algorithms. Our method does not utilize a model of the robot and, as a result, requires less data and is free from error accumulation. This is made possible by concentrating the exploration with the action-perception relation. We categorize our overall approach as semi-BBO because of keyframe constraints in our search space.

Chapter 3

LEARNING FROM DEMONSTRATION FRAMEWORK

This thesis develops a Learning from Demonstration seeded Policy Search approach which utilizes keyframe demonstrations, as mentioned in Chapter 1. We are interested in robotic manipulation skills with perceptual (sub-)goals where dynamics is not the main component. In this chapter, we describe the learning from demonstration framework.

Previous work [7] has shown that naive-users tend to focus on successfully demonstrating the goals of the skill while paying less attention to the exact robot movements. There were two main ideas that came out of this observation; (1) using keyframe demonstrations to allow the users to highlight the sub-goals and (2) learning both the robot movements (action) and the perceptual sub-goals.

It was shown that the learned action models can be used to execute the skill, goal models can be used monitor the execution and the monitoring output can be used to improve the action model. These approaches do not make any skill-specific assumptions, including rewards. However, they assume that there is a single object of interest and that there is a way to segment out these objects from perceptual input. We follow the same assumptions.

In this thesis, we build upon these ideas. Our main aim is to reduce the number of trials to get successful skill models. We do this by learning a model to jointly represent actions and goals, using this to focus the high-level exploration effort and reward predictive exploration to focus the low-level exploration effort.

The motivation behind learning a joint model is that this information gives us the expected/desired sub-goals and their sequences as we execute the trajectory. Any deviations from this expectation can then be used to focus the RL effort. The demonstrations include information about the relationship between the sub-goals and the actions; the state of the object of interest based on the robot arm pose.

For example, in a box closing skill, the lid of the box should be closed when the robot end-effector is under the lid and it should be half-closed when the end-effector moves up (see the Demonstrations part at the top of Fig. 3.1) etc. If we know this relationship, then we can detect when the robot motion does not lead to desired sub-goal, e.g. if the arm moved too much upwards causing the lid to drop and stay open. However, in previous work, the relationship between the actions and the sub-goals are not explicitly learned.

Towards this end, we use a joint model to represent the relationship between the robot action and perceptual goals which is learned from keyframes. Our probabilistic model is a Markovian Dynamic Bayesian Network (DBN) which resembles Hidden Markov Models (HMM). It has two set of hidden states, representing the action and the goal. These hidden states have their own prior probabilities and emission models, independent of the other. However, they are tied with the transition dynamics; the next action/goal hidden state is dependent on both of the current action and goal states. The Fig. 3.1 shows the two HMM idea utilized by the previous work and the Fig. 3.2 shows the DBN version.

In this section, we first describe our data and then move on to describe the structure, training and usage of the DBN.

3.1 Action and Goal Data

In our approach, human teachers provide keyframe data during kinesthetic demonstrations. A snapshot of an example interaction can be seen in Fig. 3.3. We use the robot joint encoders to collect action data and a depth camera to collect perceptual data which are then furher processed as described next. Relevant information is collected whenever the teacher provides a keyframe.

Our action observations are the pose of the end-effector with respect to the object of interest in 7-D (3-D position and 4-D unit quaternion to represent orientation). We calculate this by using the robot's forward kinematics, the transformation between the camera and the robot base and object pose in the camera frame, which is obtained from the point cloud data.

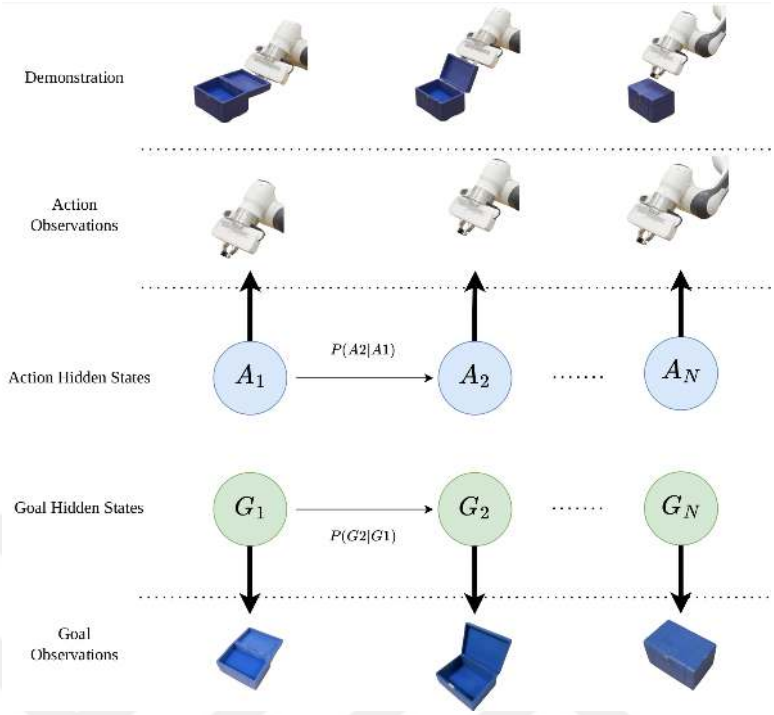

Figure 3.1: Two Hidden Markov Models are used to learn the action and goal separately. Both action and goal observations are multivariate Gaussian distributions.

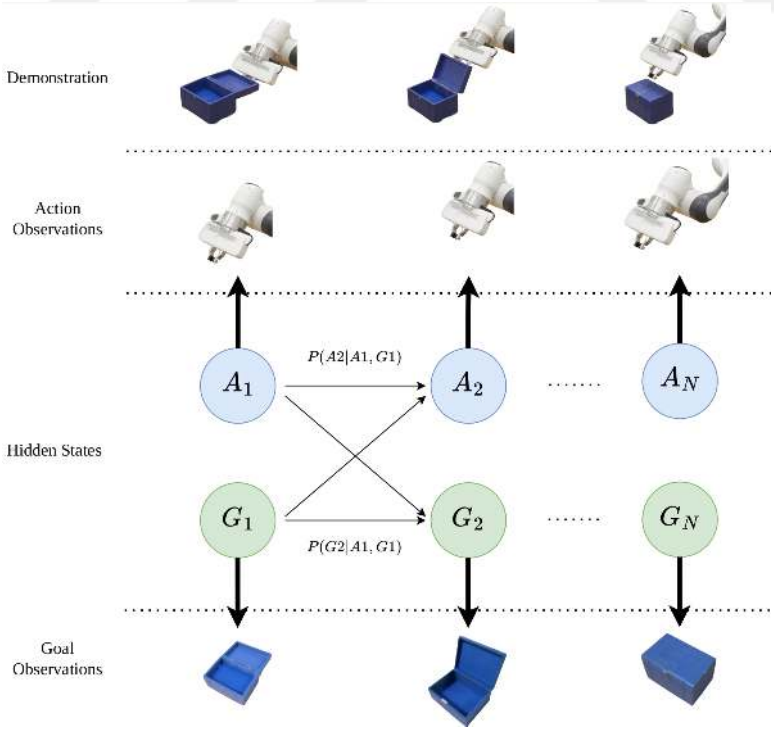

Figure 3.2: One DBN model is used to learn the action and goal, along the connection between each other. Both action and goal observations are multivariate Gaussian distributions.

Figure 3.3: User kinesthetically teaches *close the box* skill to Franka Emika Panda arm. Microsoft Kinect V1 records Point Cloud data of the object of interest.

Our goal observations are extracted object features. Recorded Point Cloud information incorporates redundant information such as the background, table and user (Fig. 3.3). In order to extract the object of interest from the raw Point Cloud data, we use the technique presented in [8]. This approach utilizes Euclidean clustering using the similarity of colors and surface normals.

We utilize two-step approach to extract object-specific features due to problems related to learning large state spaces with low amount of data. First, we pass the segmented object point cloud data through a Point Cloud Auto Encoder [9] (PCAE) trained with the ShapeNet dataset [10] to generate 128-D embeddings[1]. Our network specifications are same with the [5]. This gives us a skill-independent object representation. Second, we apply principal component analysis (PCA) on all the feature vectors of the demonstrated keyframes to further reduce the observation dimension to 8. This gives us a skill-specific representation. Our perception pipeline can be seen in 3.4.

As mentioned, we calculate the object pose with respect to the camera frame using the point cloud. After segmenting the object, we fit a 3-D rotated bounding box to its points. We take the pose of the object as the pose of this bounding box.

[1]In order to create a constant size input to PCAE, we uniformly sample 2048 points from the segmented object Point Cloud.

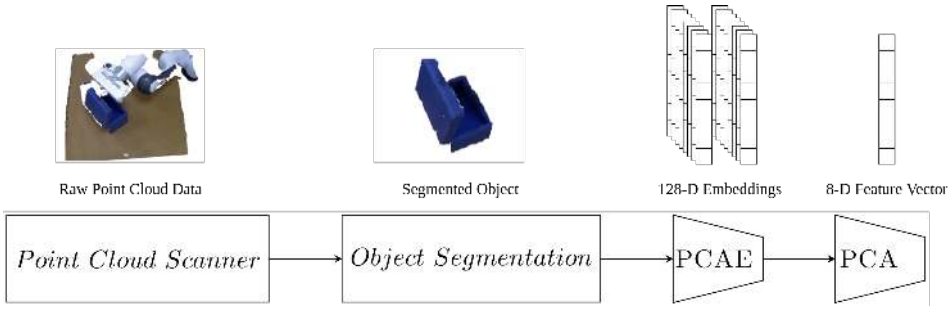

Figure 3.4: Perception pipeline

To summarize, our action observations are the end-effector pose with respect to the object and our goal observations are the skill specific object features extracted from the point cloud data.

3.2 Dynamic Bayesian Network

Previous works utilize separate HMMs to learn pose (action model) and perceptual information (goal model) [7]. The sequential nature of the HMMs makes them a natural candidate for representing keyframe data.

A HMM is an extension of a Markov Chain. A Markov chain is used to represent a sequence of possible discrete events. These possible events can be anything (Integers, Names, Tags) as long as the current event only depends on the previous event in the sequence (Markov Assumption). Hidden Markov Models extends this idea by making the events hidden, meaning that they can not be directly accessed but can only be observed with a noisy process. To illustrate, an HMM can be used to convert speech into text using an encoder. In such problem, hidden states represent the words, and observations are vocal utterances. Elements of an HMM can be learned directly from observed sequences given the number of hidden states. After learning, HMM can be queried for the likelihood of an observed sequence and, most likely hidden state path of this sequence.

In this work, hidden states are interpreted as the sub-goals to accomplish, and observations are multivariate Gaussian distributions learned from the keyframes. Learning two separate HMMs loses the connection between the action and the goal

parts by creating two independent models and makes it impossible to utilize their dependency.

Our novel approach, Dynamic Bayesian Network, includes the connection between the action and goal models. This new architecture demonstrated in Fig. 3.2, which is very similar to the HMM [4] architecture in Fig. 3.1. The connection between the two parts changes the transitions of DBN. The new transitions use previous hidden states from action and goal parts rather than just action or goal. DBN requires 3D transition tensors to represent such transitions compared to HMM's 2D transition matrix. This minimal change between models makes it possible to use an EM-based parameter-learning using the Baum-Welch algorithm.

Our DBN has similar parameters to that of a HMM but for each action and goal state. The difference is due to the transition probabilities which capture the relationship between the action states and goal states. We use the following notation to describe our DBN:

$l^{a/g}$: Number of action/goal hidden states

m : The number of keyframes, different for each demonstrations

A : the set of action hidden states, $\{a_1, ..., a_{l^a}\}$

G : the set of goal hidden states, $\{g_1, ..., g_{l^g}\}$

$X_t^{a/g}$: action/goal state at time $t, t \in 1, \ldots, m$

$O_t^{a/g}$: action/goal observation at time $t, t \in 1, \ldots, m$

$\pi^{a/g}$: The action/goal prior probability vector

$\pi_i^a = \Pr(X_1^a = a_i),\ \pi_i^g = \Pr(X_1^g = g_i)$

$\zeta^{a/g}$: The set of terminal action/goal hidden states

T^g : The goal state transition probability tensor $(l^G \times l^A \times l^G)$

$T_{ijk}^g = \Pr\left(X_{t+1}^g = g_k \middle| X_t^a = a_i, X_t^g = g_j\right)$

T^a : The action state transition probability tensor $(l^A \times l^A \times l^G)$

$T_{ijk}^a = \Pr\left(X_{t+1}^a = a_k \middle| X_t^a = a_i, X_t^g = g_j\right)$

$\Theta^{a/g}$: The mean and covariance of the action/goal emissions $\{\mu_i^{a/g}, \Sigma_i^{a/g}\}\ |_{i=1}^{l^{a/g}}$

3.2.1 Learning

We learn the prior probabilities, transition probability vectors, emission model parameters and the terminal states (the states which are the end of a demonstration) of the DBN from demonstrations. We derive the EM update rules of the transition tensors in Appendix A. The remaining parameters are calculated using the same rules as a HMM.

We start the EM-updates by first initializing the DBN model. We initialize the emission probabilities, multivariate Gaussian distributions, by learning Gaussian Mixture Models [11], which itself initialized ten times for best fit. The number of action hidden states, l^a, is selected as the maximum number of keyframes seen during the demonstrations, and the number of goal hidden states l^g is selected experimentally depending on the skill type.

3.2.2 Trajectory Generation

We generate a keyframe trajectory to use in our episodic RL approach. Our method is similar to the technique authors used with HMMs [7]. This approach finds the most likely hidden state path between the Start and Terminal states of the DBN and uses a linear trajectory between the mean of the emissions to generate a continuous trajectory. DBN version of the algorithm generates both action and goal hidden states for the path rather than only action. Trajectory generation is detailed in Alg. 1

Our approach finds the most likely path between each prior state in π^a, π^g to each terminal state in ζ_a, ζ_g (lines 1-11). Given transition tensors T^a, T^g, start and end states, *CalcPath* function calculates the path between states. We take the negative logarithm of the values in transition tensors and sum them along the trajectory, which is equal to multiplying probabilities. Then the most likely path among the possible trajectories is selected (line 12). This trajectory and its corresponding hidden goal states are selected as the trajectory to be used in episodic RL (lines 13-20). Action emission means are then used to generate the end-effector trajectory from action means. We used linear interpolation between keyframes to create a continuous trajectory. This trajectory is transformed into robot base during the execution.

Algorithm 1 Trajectory Generation

$\mathbf{\Phi} = \varnothing$
for all $(s_i^a, s_i^g) \in l^a \times l^g$ **do**
  if $\pi^a(s_i^a) \neq 0$ **and** $\pi^g(s_i^g) \neq 0$ **then**
    for all $(z_i^a, z_i^g) \in l^a \times l^g$ **do**
      if $\zeta^a(z^a) \neq 0$ **and** $\zeta^g(z_i^g) \neq 0$ **then**
        ϕ = CalcPath(s_i^a, s_i^g, z_i^a, z_i^g, T^a, T^g)
        $\mathbf{\Phi} = \mathbf{\Phi} \cup \phi$
      end if
    end for
  end if
end for
$\Gamma = \underset{\phi \in \mathbf{\Phi}}{arg\ max}(loglik(\phi))$
R $\leftarrow \varnothing$
goalPath $\leftarrow \varnothing$
for all $X_i^a \in \Gamma$ **do**
  R $\leftarrow \mu_i^a$
end for
for all $X_i^g \in \Gamma$ **do**
  goalPath $\leftarrow X_i^g$
end for
$path$ = CartesianPath(R, t)
return $path$, goalPath

Chapter 4

REINFORCEMENT LEARNING

This section explains the steps required to adapt our LfD framework into Reinforcement Learning problem. We will first explain the reward signal we used during the experiments. Then reveal our approach, BO-PI^2, and its novelties over PI^2 based approaches. Throughout that we introduce our algorithms for focusing on the failed sub-goals of the trajectory and then go over the novel sampling approach BO-PI^2 utilizes.

4.1 POMRP

We use a modified version of the three-step approach introduced in [5] to generate dense reward signals using the Partially Observable Markov Reward Process (POMRP). First, rewards are assigned to goal terminal states of the DBN to generate Markov Reward Process (MRP). Then, rewards of the terminal states are used in Monte Carlo sampling to calculate the value of each hidden state. Finally, values are multiplied with posterior probability to calculate the instant reward.

DBN's hidden states and transition tensors can be used to generate a Markov Process (MP). In order to extend the MP into (Markov Reward Process) we assign +1 reward to the goal terminal hidden states and 0 to the rest of the goal hidden states of the MP (Eqn. 4.1).

Second, we use Monte Carlo sampling to update values of hidden states ($v(s)$). We use random sampling to generate hidden state paths from DBN. Demonstration with the maximum length during training selected as hidden state path length. We terminate the current sampling process if we encounter a terminal state. In order to calculate the return of sampled goal hidden states, we use Eqn. 4.2 where γ is the discount factor and r_t is the reward of g_t (Eqn. 4.1).

We use Eqn. 4.3 to calculate the value of each state where n is the number of sampled goal hidden state path, n_g is the number of times state g is sampled and $\mathbb{1}$ is the indicator function. ($n_g = \sum_{i=1}^{n}\sum_{t=1}^{L} \mathrm{R}^i(\mathrm{g}_t^i)\mathbb{1}(\mathrm{g}_t^i{=}\mathrm{g})$). Finally, we multiply the value of goal terminal hidden states with a constant to weight the terminal rewards.

$$r_t^i = \begin{cases} +1, & g_t^i \epsilon \zeta^g \\ 0, & o/w \end{cases} \tag{4.1}$$

$$R^i(g_t^i) = \begin{cases} r_t^i + \gamma R^i(g_{t+1}^i) & t \neq L \\ r_L^i & o/w \end{cases} \tag{4.2}$$

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

Finally, we sum the observation's expected value along the trajectory to calculate total reward of the trajectory. Let $\mathbb{Z} = \{z_1, ..., z_t, ..., z_T\}$ be a goal observations collected during the skill execution. The expected value of observations are calculated by multiplying its value with the state posterior (Eqn. 4.4). Return value of the skill execution is the summation of these expected values using the Eqn. 4.5

$$h(\mathbb{Z}, t) = E[v(R_t)|\mathbb{Z}] = \sum_{g_t \in G_t} v(g_t) P(g_t|\mathbb{Z}) \tag{4.4}$$

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

Our approach only uses goal states to calculate rewards from the goal data, but it is also possible to use action information and generate Goal-Action POMRP to include the action during reward calculation.

4.2 State and Episode Selection: Focusing the High-Level Exploration

Previous PI^2 based approaches execute the entire trajectory in a single rollout. Our approach naturally extends DBN into RL domain by breaking the trajectory into

more manageable sub-goals to solve the credit attribution problem. In this section, we will explain additional steps used to adapt our approach to keyframes.

We execute the initial trajectory generated by the DBN to find failed sub-goals. During the execution, we collect action and goal data at keyframe points. Then we use the Viterbi algorithm to find the most likely hidden states corresponding to the observed keyframes. Our approach compares the expected hidden states generated from the trajectory and observed hidden states calculated from the real execution to find the failed sub-goals using Alg. 2. This approach allows us to focus on the failed sub-goals of the trajectory necessary for skill success and skip unrelated parts.

We loop through goal hidden states (line 2) and compare expected and observed hidden states. If the two are the same, we check the next sub-goal (lines 3-4). Otherwise, we consider the current sub-goal as failed. In such cases, we append current and previous action hidden states to the improvement list (lines 6-9). The reason for adding the previous keyframe is that the current sub-goal depends on the motion between the previous and the current action state. Note that the DBN's connection between the action and goal part allows us to use this algorithm.

As mentioned above, the motion between two hidden states is vital in achieving the sub-goals. Thus we take pairs of consecutive action states from the output of Alg. 2 as candidates for updates and generate rollouts between the start and end states of the pair. We sample from the emission models of these states (see Sect. 4.3) and generate a linear trajectory between them to execute for rollouts.

We update the emission model of a single action hidden state among the pair. We choose this approach because failure might depend on only one hidden state, and updating only one halves the parameter space. Decreasing the parameter space leads to better sample efficiency for the reward-predictive modelling that comes later. This method requires us to choose the hidden state to update among the pair. We start updating the end-state first. If the reward obtained between consecutive episodes is less than a threshold after a pre-determined number of episodes and start-state is not updated in previous sub-goals we switch the state to update using Alg. 3. We perform this until the sub-goal is satisfied or the maximum number of episodes is reached. This is done in the Line 4 of the BO-PI^2 algorithm that will be

Algorithm 2 Sub-Goal Assessment

Input: Expected and Observed Action/Goal Hidden States

Output: Failed Sub-Goals

```
SubGoals = ∅
for i = 0 to #Observations do
    if ObsGoalHS[i] = ExpectedGoalHS[i] then
        continue
    else
        if ExpectedGoalHS[i − 1] ∉ SubGoals then
            SubGoals ← ObsActionHS[i − 1]
        end if
        SubGoals ← ObsActionHS[i]
    end if
end for
return SubGoals
```

introduced in the next section.

One drawback of the Alg. 3 is that it is still possible to make unnecessary updates in the first episode (i.e., Unnecessarily updating the goal state while failure depends on the start of the trajectory.) and use a model with inappropriate updates. After flipping the update vector, we use the initial model for the other keyframe to solve this problem.

Once the hidden action states that result in a failed sub-goal are identified, it is time to update the action emission parameters.

Algorithm 3 Update Candidate Selection

Input: $returndDiff$, $returnThreshold$, $startUpdated$

Output: $updateVector$

1: **if** ($currEpisode == 0$ or $startUpdated$) **then**
2: $updateVector = [False, True]$
3: **else if** ($returnDiff < returnThreshold$) **then**
4: $updateVector$ = FlipVector($updateVector$)
5: **end if**
6: **return** $updateVector$

4.3 Return Prediction: Focusing the Low-Level Exploration

After selecting the action state to update, we perform Episodic RL. Our approach, BO-PI^2, utilizes Bayesian Optimization to model the expected-return and use UCB type exploration to improve upon the random sampling used in PI^2-ES-Cov.

BO-PI^2 utilizes GP [1] to model the return function and refine its prediction with each new execution. We use GP because its shown notable results in the previous works [20, 21, 34] and gives us both the prediction and its uncertainty. Gaussian Process can be interpreted as a set of random variables which can be used to define distributions over functions. Form of the GP can be seen below:

$$GP(x, D, k) = \mathcal{N}(\mu(x), \sigma^2(x)) \tag{4.6}$$

Where x is the query point, D is the dataset including previous observations, and k is the kernel function. The output of the Gaussian Process is Gaussian distribution with mean μ and standard deviation σ. The kernel function is key part of the GP and models the effect of observed data over the emission space. In our work we used Gaussian Kernel, due to it's simplicity. The kernel function can be seen below, the parameter l is the kernel length scale and can be used to adjust correlation.

$$k(x_i, x_j) = exp\left(- \frac{||x_i, x_j||^2}{2l^2} \right) \tag{4.7}$$

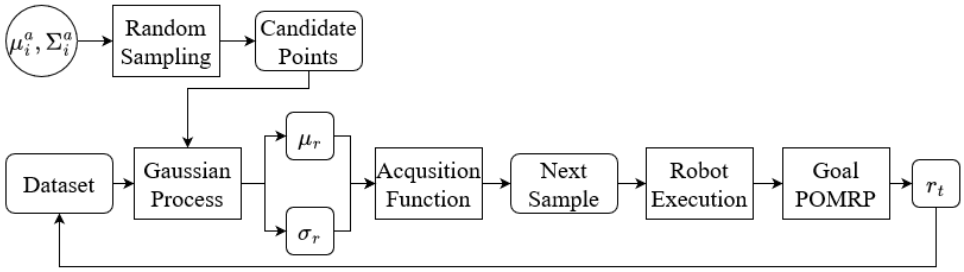

Figure 4.1: Sampling pipeline

Our GP takes 3D position information as input and outputs the expected return obtained from the learned POMRP and its variance. We exclude the 4D orientation information to work in a lower dimensional space due to the trade-off between prediction performance and data requirements.

The pipeline for GP-based sampling is depicted in Fig. 4.1. We randomly sample N candidate points during each rollout using the action hidden state emissions. GP trained with the data gathered through the previous executions is used the predict reward and uncertainty of the candidate points. Then BO-PI^2 utilizes the acquisition function to compare and select the next executed point. We use UCB [23] strategy as can be seen in Eqn. 4.8, to select the next sample depending on the predicted reward and the uncertainty of the candidate points. We utilize UCB because of its simplicity and success in previous studies [23, 20]. UCB utilizes α constant to handle the trade-off between exploration and exploitation. The maximum is taken based on the N sampled points.

$$a_{t+1} = \underset{x}{arg\ max}(\mu_t(a) + \alpha\sigma_t(a)) \tag{4.8}$$

At the end of each episode, BO-PI^2 uses the update rules of the PI^2-ES-Cov to update emission parameters using return weighted averaging. We use modified softmax function using the the return of the trajectories to calculate probabilities (Eqn. 4.9). These probabilities then used to update the emission parameters (Eqn. 4.10, 4.11).

The final version of BO-PI^2 can be seen in Alg. 4, where highlighted lines show

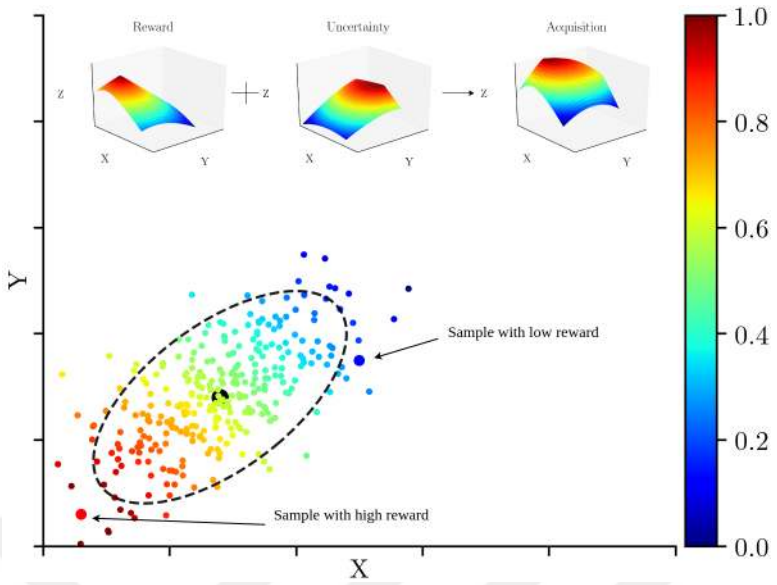

Figure 4.2: (Bottom) One step sampling of BO-PI^2 projected into 2-D with candidate sample points with corresponding returns. (Top) 3-D projections of 2D return, uncertainty and acquisition values during the sampling process. BO-PI^2 selects the following sample using acquisition function.

the additions of BO-PI^2 over PI^2-ES-Cov. We start by finding the sub-goals with Alg. 2 using the initial trajectory generated by the DBN (line 1). Then we run episodic RL for each sub-goal in our list. In each episode, we select the hidden state BO-PI^2 will focus on (line 4). After each episode (lines 3-10), we update the model action emissions and execute the new model to check whether successful; if we succeed, we move into the next sub-goal in our list (lines 16-19).

$$P_c = \frac{e^{r_c/C}}{\sum_c e^{r_c/C}} \tag{4.9}$$

$$\hat{\mu}_i^a = \sum_c P_c \mu_c^a \tag{4.10}$$

$$\hat{\Sigma}_i^a = \sum_c P_c (\mu_c^a - \mu_i^a)^T (\mu_c^a - \mu_i^a) \tag{4.11}$$

Algorithm 4 BO-PI^2

```
Execute Sub-Goal Assessment
for SubGoal in SubGoals do
    for Episodes do
        Update Candidate
        for Rollouts do
            Select the next parameter
            Execute the skill
            Calculate the return
            Update GP
        end for
        Calculate the probability of rollout
        Update the parameters
        Update the exploration covariances
        Execute the mean skill
        Update GP
        Calculate Success
        if Success then
            break
        end if
    end for
end for
```

Chapter 5

EXPERIMENTS

We compared BO-PI^2 and PI^2-ES-Cov methods in three metrics; skill success, cumulative rewards, and convergence speed. Each metric was tested with demonstrations from expert and naive users to cover the characteristic of each user type.

Each experiment was repeated five times to minimize the effect of randomness. Our return figures are normalized (between 0-250) to aid in visualization. The shaded area around the mean shows the standard deviation calculated using repeated experiments. The number of episode for each trial and experiment is different due to early termination criteria and the sub-goal assessment algorithm.

In order to make a reliable comparison between the algorithms, we adapt PI^2-ES-Cov to keyframes. This version of the algorithm works on our DBN and utilize the hidden state selection algorithms we introduced. The only distinction between BO-PI^2 and PI^2-ES-Cov is how the following sample is generated; compared to the BO-PI^2's return predictive approach, PI^2-ES-Cov uses adaptive random sampling.

5.1 Setup

Our experiment setup consists of Franka Emika Panda arm and Microsoft Kinect V1 (Fig. 3.3). The demonstration data only consist of keyframe information. During the Reinforcement Learning we collect robot end effector and perceptual pipeline data with 10Hz pipelines.

5.2 Skills

We select three skills by considering their difficulty to demonstrate on High-DoF robotic arms. Both expert and naive users gave a demonstration of three different skills; *open the box*: robot opens a box, *close the box*: robot closes a box, and *open*

Table 5.1: Object used in experiments

Name	**Close**	**Open**
Box1		
Box2		
Drawer		

the drawer: robot pulls to open a wooden drawer. Objects for each skill can be seen in Fig. 5.1.

5.3 Model Initialization

We initialize the l^g depending on the skill. We choose three for *open and close the box* skill, and four for *open the drawer* skill.

5.4 Expert Demonstrations

We performed four experiments with expert user demonstrations. Initial DBN models learned from the demonstrations were successful. To start Reinforcement Learning from a fail state, we manually perturb the means of the hidden states. The perturbations were random and can be seen in Table 5.2 for all experiments.

5.4.1 Results

Our results show BO-PI^2 outperforms PI^2-ES-Cov for almost all the skills. On average, BO-PI^2 (1) reaches 95% skill success vs 40% (Table 5.3.), (2) accumulates 2.5 more returns (Table 5.4), and (3) terminates faster with 4.5 number of episodes vs 5.7 (Table 5.5). These results imply that BO-PI^2 has faster learning capabilities, and the learned return model better exploits the return function to end up with

Table 5.2: Random mean perturbations for each hidden state*

Skill	**Object Type**	**First**	**Second**	**Third**	**Fourth**
Close	Box 1	-	-	9	6
Close	Box 2	-	-	10.2	14
Open	Box 2	-	5	-	10.4
Open	Drawer	-	-	9.8	-

* Units in centimeter.

Table 5.3: Expert - Skill Success Ratios*

Skill	**Object Type**	**BO-PI^2**	**PI^2-ES-Cov**
Close	Box 1	**100%**	40%
Close	Box 2	**100%**	20%
Open	Box 2	**100%**	20%
Open	Drawer	80%	80%
Avg.	-	**95%**	40%

* Out of 5 experiments.

high-return regions. Furthermore, BO-PI^2 accomplishes tasks even when started from sparse regions where standard BBO approaches fail to converge.

Table 5.5 indicates that our Sub-Goal Assessment and Update Candidate Selection algorithms perform better than applying episodic RL on each hidden state of the trajectory. Results indicate that BO-PI^2 outperforms the minimum amount of episodes if such an approach is used.

Figure 5.5 visualize our results and shows the reward of intermediate episodes. Figures imply BO-PI^2 outperforms not only in the final episode but in-between episodes as well.

Table 5.4: Expert - Avg. Normalized Returns*

Skill	**Object Type**	**BO-PI^2**	**PI^2-ES-Cov**
Close	Box 1	**213**	76
Close	Box 2	**198**	11
Open	Box 2	**146**	25
Open	Drawer	171	**174**
Avg.	-	**182**	71.5

* Out of 5 experiments.

Table 5.5: Expert - Avg. Termination Time*

Skill	**Object Type**	**BO-PI^2**	**PI^2-ES-Cov**
Close	Box 1	**3.2**	6.4
Close	Box 2	**4.6**	5.8
Open	Box 2	**5.6**	5.8
Open	Drawer	**4.6**	4.8
Avg.	-	**4.5**	5.7

* Out of 5 experiments.

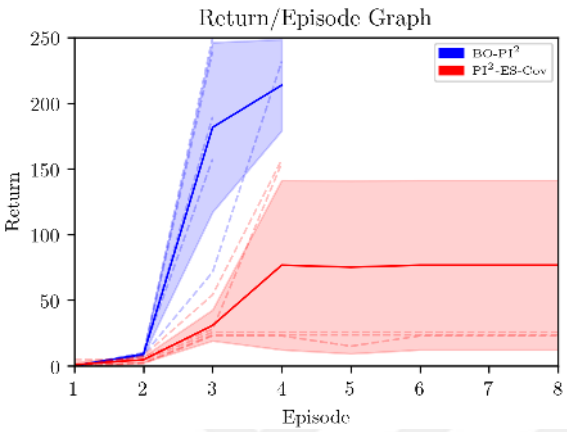

Figure 5.1: (A) Close-Box1

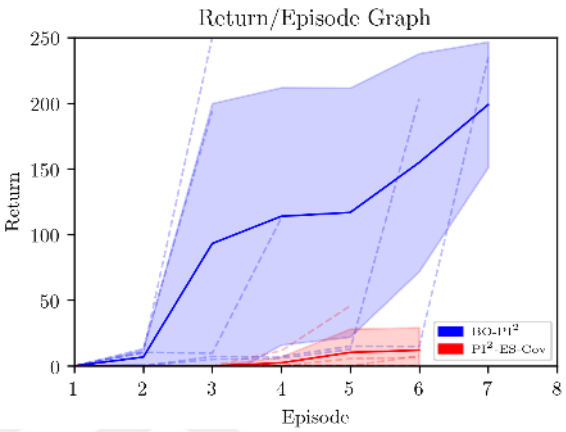

Figure 5.2: (B) Close-Box2

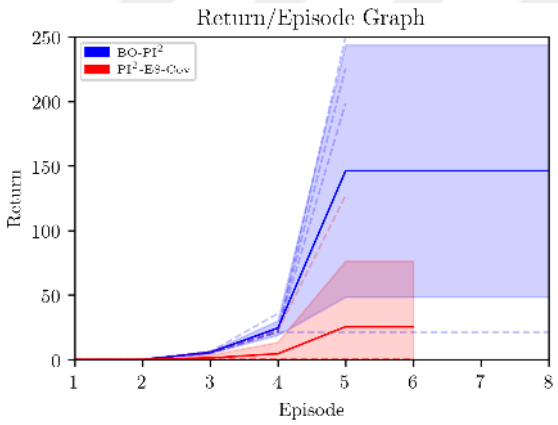

Figure 5.3: (C) Open-Box2

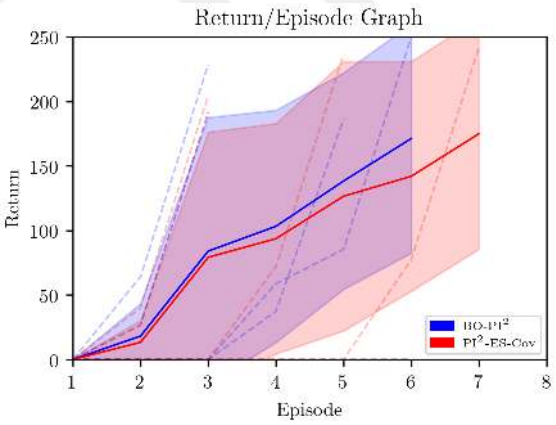

Figure 5.4: (D) Open-Drawer

Figure 5.5: Expert User Normalized Returns

5.5 HRI experiments

We further validate our results with HRI study. We collect data from 8 (4 female, 4 male) naive users aged 22-28 who do not have any experience teaching a robot. Participants are selected from the campus community.

5.5.1 Data Collection

We divide the data collection protocol into 4 phases; First, the experiment conductor asks the user to sign the consent form and gives a brief verbal description of kinesthetic teaching, environment, keyframes, and introduction flow. After the consent and greetings, the user completes two simple toy problems; (1) setting the robot into specific configurations and (2) teaching the robotic arm to move between two points. During the demonstration, the user gives verbal commands to start, capture and end demonstrations. These toy problems are selected to make the user get used to keyframe demonstrations and the 7-DoF robotic arm. Finally, the experiment conductor asks the user to demonstrate three skills; the order of skills is counterbalanced to minimize the effects of experience. Each user gives 6 demonstrations for each skill and is allowed to see the learned DBN model after 4 demonstrations.

5.5.2 Success Ratio of DBNs

We compare the performance of learned DBN models to select the ones that will be used in RL. We learned DBN using the provided demonstrations and noted the success of the trajectory generated with the Alg. 1. The table 5.6 shows the average success ratios, where each demonstration is used to learn five DBNs from scratch.

We choose 6 demonstrations (2 for each skill) to compare algorithms. We randomly selected the demonstrations amongst the participants, we did not use the ones where the goal model achieved poor performance, or the initial learned model was successful.

Table 5.6: Participant Skill Success*

ID	**Close the box**	**Open the box**	**Open the drawer**
1	100%	0%	20%
2	40%	80%	40%
3	0%	0%	60%
4	0%	0%	60%
5	0%	100%	60%
6	20%	0%	20%
7	100%	0%	60%
8	0%	0%	80%
Avg	32.5%	22.5%	60%

Out of 5 trials.

5.5.3 Results

In this setup, BO-PI^2 is either better than PI^2-ES-Cov, or they perform about the same. On average, BO-PI^2 (1) reaches 86.67% skill success vs 63.33% (Table 5.7), (2) accumulates 1.5 more returns (Table 5.8) , and (3) terminates faster with 3.7 number of episodes vs 4.3 (Table 5.9).

The figures 5.12 visualize our results and show that on average BO-PI^2 continues to outperform PI^2-ES-Cov in intermediate episodes as well. One interesting point is in Fig. 5.7, where both approaches fail to maintain the same reward between consecutive episodes. Further inspection of the demonstrations shows that participant 3 (5.7), moves the robot near the joint limits, where exploration has negative effects.

One important point is RL perfomance of the models learned from both HRI and Expert user demonstration are similar for the *open the drawer* skill. We believe this is due to difficulty of the skill, where robot end effector must hold the drawer from an thin space. Furthermore, contrary to other skills, *open the drawer* skills is harder to demonstrate with a high-DoF robotic arm.

Our results prove that BO-PI^2 shows promising results compared against the PI^2-ES-Cov, regardless of the demonstration type, and return predictive exploration strategies are beneficial to improve RL performance and to reduce the number of trials to endow robot arms with real-life manipulation skills.

5.6 Demonstration Noise

A critical point in LfD is to evaluate the effect of demonstration noise on the underlying algorithm. Our results show that BO-PI^2 can handle noise generated by naive users. To quantify this noise, we measured the Euclidean and angular mean displacement between initial and learned models and compared these values with the manual perturbations we applied. Our analysis shows that compared to hand perturbations (Fig. 5.2), on average, HRI perturbations are smaller, and the performance difference between BO-PI^2's and PI^2-ES-Cov becomes more apparent with higher noise levels. Furthermore, we see that model policies learned through naive users tend to fail only one sub-goal in contrast to our perturbations on multiple

Table 5.7: HRI - Skill Success Ratios*

Skill	**Object Type**	**Participant ID**	**BO-PI^2**	**PI^2-ES-Cov**
Close	Box 2	2	100%	100%
Close	Box 2	3	**80%**	20%
Open	Box 2	6	**100%**	60%
Open	Box 2	7	**100%**	80%
Open	Drawer	4	60%	60%
Open	Drawer	1	**80%**	60%
Avg.	-	-	**86.67%**	63.33%

* Out of 5 experiments.

Table 5.8: HRI - Avg. Normalized Returns*

Skill	**Object Type**	**BO-PI^2**	**PI^2-ES-Cov**
Close	Box 1	**198**	108
Close	Box 2	**110**	56
Open	Box 2	**195**	109
Open	Box 2	**170**	112
Open	Drawer	127	**133**
Open	Drawer	**164**	129
Avg.	-	**160**	107

* Out of 5 experiments.

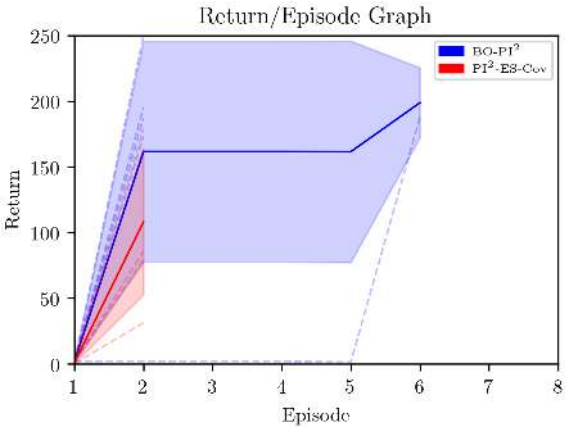

Figure 5.6: (A) Close-Box

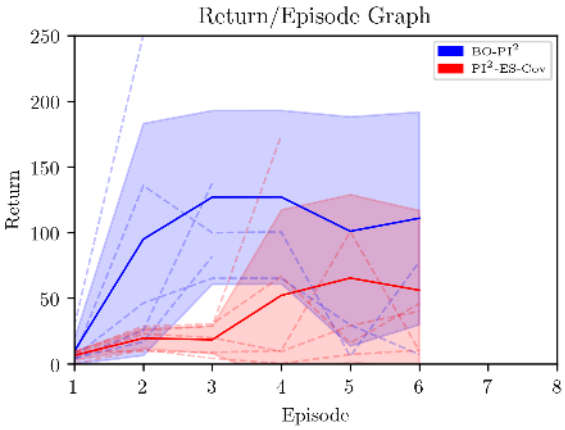

Figure 5.7: (B) Close-Box

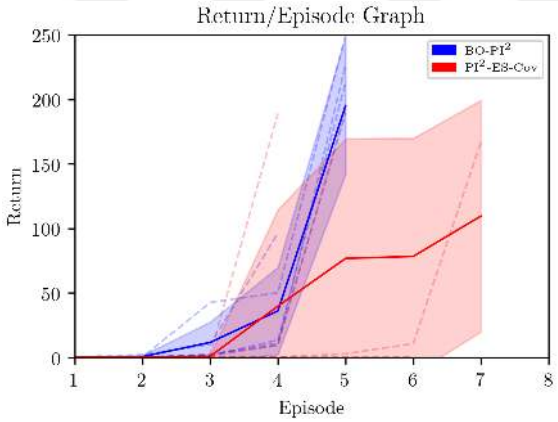

Figure 5.8: (C) Open-Box

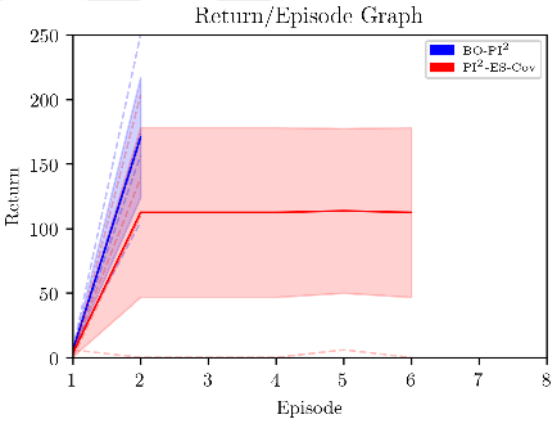

Figure 5.9: (D) Open-Box

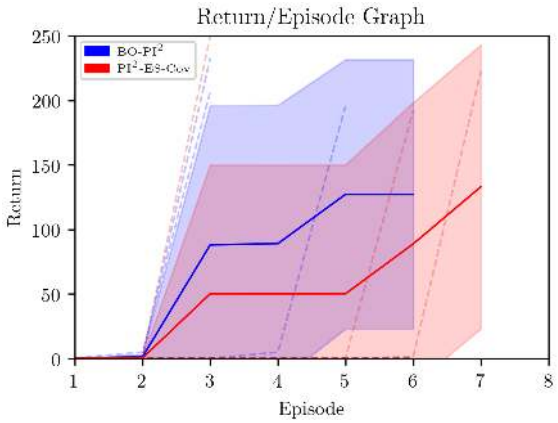

Figure 5.10: (E) Open-Drawer

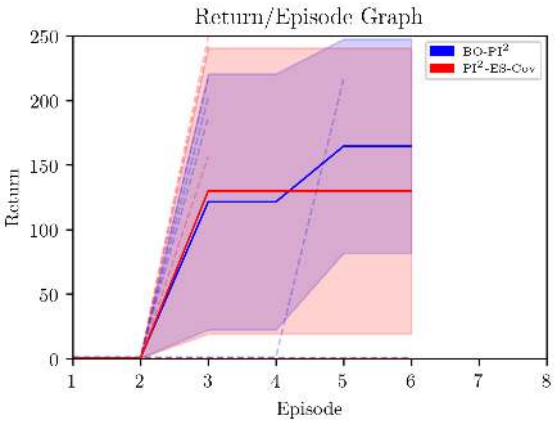

Figure 5.11: (F) Open-Drawer

Figure 5.12: HRI Normalized Returns

Table 5.9: HRI - Avg. Termination Time*

Skill	**Object Type**	**BO-PI^2**	**PI^2-ES-Cov**
Close	Box 2	2.8	**2.0**
Close	Box 2	**4.0**	5.6
Open	Box 2	**4.8**	5.6
Open	Box 2	**2.0**	2.8
Open	Drawer	**4.6**	5.6
Open	Drawer	**4.0**	4.2
Avg.	-	**3.7**	4.3

* Units in episode.

hidden states. We also see that, on average angular displacement is around the same for BO-PI^2 and PI^2-ES-Cov while being less than its Euclidean counterpart. These results can be interpreted as the noise for Euclidean displacement is higher. In overall, we also see that BO-PI^2 tends to make bigger steps in the policy space due to the utilization of GP, UCB, and our exploration constant.

Chapter 6

DISCUSSION

Generalizability: Previous work shows that goal models learned through HMM's are relatively generalizable [7] unlike the action models. A possible solution to increase the generalization of our underlying approach is to learn from more demonstration data. However, this approach is limited to the patience of the participants. In our work, we do not assume learned models are generalizable enough, but the goal part as a reward signal can be used with RL to improve the individual action demonstration.

State Selection Ablation Study: In our evaluations, both BO-PI^2 and PI^2-ES-Cov methods employ our state selection algorithms. We did not perform experiments without these i.e. we did not perform an ablation study regarding them. Going over each hidden state one-by-one is trivially worse than concentrating on the failed states. Furthermore, updating all the action emissions at once would have high-variance and limit the applicability of GP due to increased dimensionality, both resulting in high sample complexities. GP used in BO-PI^2 strongly leverages the reduced state space dimensionality by sampling only from a single hidden state.

Dynamically Changing Trajectories: Our approach assumes the trajectory generated by the DBN correctly identifies the sub-goals of the skill. If that is not the case, an additional loop is necessary to try trajectories with different likelihoods to find a successful policy. In our work, we did not include an additional outer loop due to increased time to convergence.

Using Continuous Trajectories with Dynamic Movement Primitives: Using Dynamic Movement Primitives [24] (DMP) parameters as state space is not directly possible. The easiest approach, using one GP for all dimensions of high-DoF robots, is unsuitable due to increased computational complexity. Furthermore, using separate GPs for each dimension of the state space might mislead the exploitation-

exploration trade-off of GPs. High-return behavior due to one dimension might unnecessarily affect exploring other dimensions.

Policy with Multiple Hidden States: Our approach only improves one keyframe in each episode by using the state space of one hidden state, which limits the exploration and results in faster convergence. It is possible to use multiple hidden states by deriving a new policy that encodes state spaces of multiple hidden states. In our preliminary 2D works, we tried different approaches and saw that using separate state spaces for each keyframe resulted in the fastest convergence with the highest return.

Starting GP from the demonstrations: Preliminary studies indicated that using only returns from the keyframes on the trajectory resulted in sparse return signals, which is insufficient to learn complex skills. In order to use denser returns, our approach uses the return calculated from the whole trajectory, which blocks us from initializing the GP with the demonstration data. As a possible solution, increasing number of action hidden states might increase the density of the keyframe returns and utilize the return information from the demonstrations, which in turn might also increase the performance of our approach.

Effect of Hidden State Selection Algorithm, There are multiple ways to update hidden states of the failed sub-goals. One possible approach is to start the trajectory from a successful state and set a goal state to the first failed sub-goal in the list. Unfortunately, this approach spends too many episodes for unnecessary improvement (i.e., when failure depends on the goal state, it still requires N episode for the start state.). As an alternative, both the start and hidden state can be improved simultaneously. However, this approach doubles the state space dimensions and is not feasible for low-data, sparse returns scenarios. Our approach solves this problem without increasing the state space and spending time on unnecessary updates. The Table 5.5 shows that our algorithm performs better than iterating over all possible failed sub-goals.

Future Work, Our approach only uses goal information to learn the reward model and disregards the crucial information related to the object of interest from the goal model. It is possible to utilize this information by learning a joint action-goal

Table 6.1: HRI - Avg. Euclidean* displacement

Skill	**Object Type**	**BO-PI^2**	**PI^2-ES-Cov**
Close	Box 1	9.42	5.86
Close	Box 2	15.74	16.40
Open	Box 2	3.87	1.92
Open	Box 2	8.74	5.33
Open	Drawer	7.18	6.83
Open	Drawer	7.79	5.76
Avg.	-	8.79	7.01

* Units in centimeter.

Table 6.2: HRI - Avg. angular* displacement

Skill	**Object Type**	**BO-PI^2**	**PI^2-ES-Cov**
Close	Box 1	4.50	2.70
Close	Box 2	12.59	12.58
Open	Box 2	3.58	2.63
Open	Box 2	2.94	4.08
Open	Drawer	5.03	6.18
Open	Drawer	7.07	5.37
Avg.	-	5.95	5.59

* Units in degree.

reward model with an approach like Conditional Neural Processes [25] (CNP). CNPs can be trained with multiple GPs learned from the demonstrations and experiments. Using goal-related information inside the reward model further allows us to use transfer learning capabilities to learn the same skills for different objects.

Chapter 7

CONCLUSION

This thesis aims to develop algorithms that allow robots to improve skill execution independently when the behaviors learned from experts and naive users are insufficient. To achieve this, a new model is used, which learns the connection between the robot movements and the change in the object during the demonstrations. A policy search algorithm for self-learning has been proposed in case our model fails and showed that this approach outperforms existing methods.

Our approach uses the connection between action (robot pose) and goal (changes in the object of interest) models to find the failed sub-goals. We use keyframe demonstrations to model these. Our novel state and hidden state selection algorithm utilize the connection between the action and goal part to focus on failed sub-goals. Our results show this approach provides faster convergence to successful skill demonstrations.

We further propose a novel policy search algorithm to improve the efficiency of the action part's samples. In contrast to the previous approaches, our approach uses reward-weighted averaging due to noisy and complex reward space. To improve the efficiency of the sampling process, we use the Bayesian Optimization technique with Gaussian Process. In each step, we refine our predictions about the return space to select the following sample.

We test our approach on three different skills collected from both expert and naive users. Our results state that, on average, BO-PI^2 outperforms the previous state-of-the-art PI^2-ES-Cov in all metrics by a salient margin. This concludes that keyframe seeded and Bayesian Optimized Policy search is a prominent approach to improve the self-learning capabilities of robots.

BIBLIOGRAPHY

[1] C. E. Rasmussen and C. K. I. Williams, "Gaussian processes for machine learning". MA, USA: MIT Press, 2006.

[2] E. Brochu, V. M. Cora, and N. De Freitas, "A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," 2010, arXiv:1012.2599.

[3] B. Akgun, M. Cakmak, K. Jiang, A. L. Thomaz, "Keyframe-based Learning from Demonstration," Int J of Soc Robotics 4, 343–355 2012.

[4] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," in Proceedings of the IEEE, vol. 77, no. 2, pp. 257-286, Feb. 1989, doi: 10.1109/5.18626.

[5] C. Eteke, D. Kebüde and B. Akgün, "Reward Learning From Very Few Demonstrations," in IEEE Transactions on Robotics, vol. 37, no. 3, pp. 893-904, June 2021, doi: 10.1109/TRO.2020.3038698.

[6] Sonia C. and A. L. Thomaz, "Robot Learning from Human Teachers," USA: Morgan & Claypool Publishers, 2014.

[7] B. Akgun and A .Thomaz, "Simultaneously learning actions and goals from demonstration," Auton. Robots 40, 2 (February 2016), 211–227. https://doi.org/10.1007/s10514-015-9448-x

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

[9] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas, "Learning representations and generative models for 3d point clouds," arXiv preprint arXiv:1707.02392, 2017.

[10] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su et al., "Shapenet: An information-rich 3d model repository," arXiv preprint arXiv:1512.03012, 2015. Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, BIntroduction. Thirion, O. Grisel, M. Blondel, A. Müller, J. Nothman, G. Louppe, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine Learning in Python," 2003. [Online]. Available:https://arxiv.org/abs/1201.0490

[12] F. Stulp and O. Sigaud, "Policy improvement: Between black-box optimization and episodic reinforcement learning," in Proc. Journées Francophones Planification, Décision, et Apprentissage pour la conduite de systèmes, 2013.

[13] K. Chatzilygeroudis, V. Vassiliades, F. Stulp, S. Calinon, and J.-B. Mouret, "A survey on policy search algorithms for learning robot controllers in a handful of trials," IEEE Transactions on Robotics, 2019.

[14] S. Mannor, R. Rubinstein, G. Yohai, "The Cross Entropy method for Fast Policy Search," in Proceedings of the Twentieth International Conference on Machine Learning, 2003

[15] N. Hansen and A. Ostermeier, "Completely Derandomized Self-Adaptation in Evolution Strategies," in Evolutionary Computation, vol. 9, no. 2, pp. 159-195, June 2001, doi: 10.1162/106365601750190398.

[16] F. Stulp, O. Sigaud, "Path Integral Policy Improvement with Covariance Matrix Adaptation," in Proceedings of the 10th European Workshop on Reinforcement Learning, 2012

[17] E. Theodorou, J. Buchli, and S. Schaal, "A generalized path integral control approach to reinforcement learning," journal of machine learning research, vol. 11, no. Nov, pp. 3137–3181, 2010.

[18] E. Brochu, V. M. Cora, and N. De Freitas, "A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," 2010, arXiv:1012.2599.

[19] K. Chatzilygeroudis, R. Rama, R. Kaushik, D. Goepp, V. Vassiliades, and J.-B. Mouret, "Black-Box Data-efficient Policy Search for Robotics," in IROS, 2017

[20] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret, "Robots that can adapt like animals," Nature, vol. 521, no. 7553, pp. 503–507, 2015.

[21] R. Antonova, A. Rai, and C. G. Atkeson, "Sample efficient optimization for learning controllers for bipedal locomotion," in Humanoids, 2016.

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

[23] R. Calandra, A. Seyfarth, J. Peters and M. P. Deisenroth, "An experimental comparison of Bayesian optimization for bipedal locomotion," 2014 IEEE International Conference on Robotics and Automation (ICRA), 2014, pp. 1951-1958, doi: 10.1109/ICRA.2014.6907117.

[24] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Learning attractor landscapes for learning motor primitives," in Advances in neural information processing systems, 2003, pp. 1547–1554.

[25] M. Garnelo et al., "Conditional neural processes," Proc. 35th Int. Conf. Mach. Learn., 2018, pp. 1704–1713.

[26] H. G. Beyer and B. Sendhoff, "Covariance matrix adaptation revisited—The CMSA evolution strategy—," in Proceedings of the 10th International Conference on Parallel Problem Solving From Nature (PPSN X), Dortmund, Germany (Lecture Notes in Computer Science) vol. 5199, G. Rudolph et al., Eds. Berlin, Germany: Springer, Sep. 2008, pp. 123–132.

[27] Marc Peter Deisenroth; Gerhard Neumann; Jan Peters, A Survey on Policy Search for Robotics , now, 2013.

[28] J. Luo, C. Papin and K. Costello, "Towards Extracting Semantically Meaningful Key Frames From Personal Video Clips: From Humans to Computers," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 19, no. 2, pp. 289-301, Feb. 2009, doi: 10.1109/TCSVT.2008.2009241.

[29] E. D. Nerurkar, K. J. Wu and S. I. Roumeliotis, "C-KLAM: Constrained keyframe-based localization and mapping," 2014 IEEE International Conference on Robotics and Automation (ICRA), 2014, pp. 3638-3643, doi: 10.1109/ICRA.2014.6907385.

[30] C. Mueller, J. Venicx and B. Hayes, "Robust Robot Learning from Demonstration and Skill Repair Using Conceptual Constraints," 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, pp. 6029-6036, doi: 10.1109/IROS.2018.8594133.

[31] A. Saran, E. S. Short, A. Thomaz, and S. Niekum, "Understanding Teacher Gaze Patterns for Robot Learning," in 3rd Conference on Robot Learning (CoRL 2019), Osaka, Japan, 2019.

[32] F. Stulp and O. Sigaud, "Robot skill learning: From reinforcement learning to evolution strategies," Paladyn, Journal of Behavioral Robotics, vol. 4, no. 1, pp. 49–61, 2013.

[33] N. Hansen, "The cma evolution strategy: a comparing review," in Towards a new evolutionary computation. Springer, 2006, pp. 75–102.

[34] K. Chatzilygeroudis, R. Rama, R. Kaushik, D. Goepp, V. Vassiliades, and J.-B. Mouret, "Black-Box Data-efficient Policy Search for Robotics," in IROS, 2017

Appendix A

MARKOVIAN DYNAMIC BAYESIAN NETWORK

This section provides the transition update rules of the Markovian Dynamic Bayesian Network in Fig. 1. The model parameters are given below:

N^A : Number of action hidden states

N^G : Number of goal hidden states

T : Number of keyframes, different for each demonstrations

$S^A = \{s_1^A,\ s_2^A, \ldots, s_{N^A}^A\}$: The set of action hidden states

$S^G = \{s_1^G,\ s_2^G, \ldots, s_{N^G}^G\}$: The set of goal hidden states

X_t^A : Action state at time t, $t \in 1, \ldots, T$

X_t^G : Goal state at time t, $t \in 1, \ldots, T$

O_t^A : Action observation at time t, $t \in 1, \ldots, T$

O_t^G : Goal observation at time t, $t \in 1, \ldots, T$

π^A : Action prior probability vector, $\pi_i^A = \Pr(X_1^A = s_i^A)$

π^G : Goal prior probability vector, $\pi_j^G = \Pr(X_1^G = s_j^G)$

τ^A : Action terminal probability vector, $\tau_i^A = \Pr(X_T^A = s_i^A)$

τ^G : Goal terminal probability vector, $\tau_j^G = \Pr(X_T^G = s_j^G)$

P^A : The action state transition probability tensor of shape (N^A, N^G, N^A), $P_{ijk}^A = \Pr\left(X_{t+1}^A = s_k^A \middle| X_t^A = s_i^A, X_t^G = s_j^G\right)$

P^G : The goal state transition probability tensor of shape (N^A, N^G, N^G), $P_{ijk}^G = \Pr\left(X_{t+1}^G = s_k^G \middle| X_t^A = s_i^A, X_t^G = s_j^G\right)$

$\Theta^A = \{\mu_i^A, \Sigma_i^A\}\ |_{i=1}^{N^A}$: Means and covariances of the action observations

$\Theta^G = \{\mu_i^G, \Sigma_i^A\}\ |_{i=1}^{N^G}$: Means and covariances of the goal observations

We learn the prior (π), terminal (τ), and observation parameters (Θ) using the same update rules as the Baum-Welch Algorithm for Hidden Markov Models. In

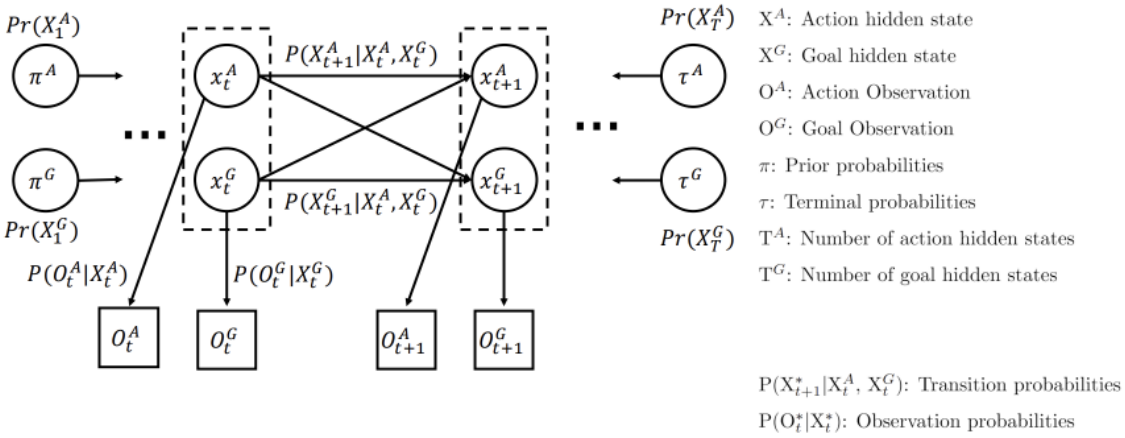

Figure A.1: Parameters and visualization of MDBN architecture

this Appendix, we derive the update rules to learn transition probabilities (P). The parameters of the MDBA will be shown as $\lambda = \left(\pi^A, \pi^G, P^A, P^G, \Theta^A, \Theta^G, \tau^A, \tau^G\right)$ for the next steps.

An "Expectation-Maximization" (EM) based approach is used to learn the transition parameters. First, the likelihood of the MDBN architecture must be written (only for one demonstration):

$$L\left(O^A, O^G, X^A, X^G \,|\, \lambda\right) =$$
$$\pi^A_{X_1^A}\, \pi^G_{X_1^G} \left(\prod_{t=1}^{t=T-1} P^A\left(X_t^A, X_t^G, X_{t+1}^A\right) P^G\left(X_t^A, X_t^G, X_{t+1}^G\right) f_{X_t^A}\left(O_t^A\right) f_{X_t^G}\left(O_t^G\right) \right)$$
$$\tau^A_{X_T^A} \tau^G_{X_T^G} \quad \text{(A.1)}$$

In Equation A.1, $O^* = \{O_1^*, O_2^*, \ldots, O_T^*\}$ denotes observations, $X^* = \{X_1^*, X_2^*, \ldots, X_T^*\}$ denotes hidden states and $f_{X_t^*}(O_t)$, denotes the multivariate Gaussian distributions parameterized with μ_{X_t}, Σ_{X_t} where the superscript * is a place holder for the action or goal.

Let's set $\alpha_\lambda = L\left(O^A, O^G, X^A, X^G \,|\, \lambda\right)$ and λ' is parameter set from the previous step. The Q function to be used in deriving the EM algorithm rules is then:

$$Q\left(\lambda, \lambda'\right) = \mathbb{E}\left[\log\left(L\left(O^A, O^G, X^A, X^G | \lambda\right)\right) \,\middle|\, O^A, O^G, \lambda'\right] = \sum_{X^A, X^G} \log(\alpha_\lambda)\alpha_{\lambda'}$$
$$\text{(A.2)}$$

Taking the logarithm of the Equation A.1 results in separate and summed model parameter. After taking the logarithm:

$$\begin{aligned}\log(\alpha_\lambda) = &\log(\pi^A_{X^A_1}) + \log(\pi^G_{X^G_1}) \\ &+ \sum_{t=1}^{T-1} \log(P^A\left(X^A_t, X^G_t, X^A_{t+1}\right)) + \sum_{t=1}^{T-1} \log(P^G\left(X^A_t, X^G_t, X^G_{t+1}\right)) \\ &+ \sum_{t=1}^{T-1} \log(f_{X^A_t}\left(O^A_t\right) f_{X^G_t}\left(O^G_t\right)) + \sum_{t=1}^{T-1} \log(f_{X^A_t}\left(O^A_t\right) f_{X^G_t}\left(O^G_t\right)) \\ &+ \log(\tau^A_{X^A_T}) + \log(\tau^G_{X^G_T})\end{aligned} \tag{A.3}$$

Observations, prior probabilities and terminal probabilities of the MDBN have same form as the Hidden Markov Models. This similarity allow to use methods in the literature to calculate these parameters. This report will focus on the maximization steps of the transition tensors. Additionally, parts related to action and goal transition tensors are also seperate terms which allows us to simplify the Q function:

$$\begin{aligned}\bar{Q}\left(\lambda,\ \lambda'\right) &= \sum_{X^A, X^G} \alpha_{\lambda'} \sum_{t=1}^{T-1} \log(P^*\left(X^A_t, X^G_t, X^*_{t+1}\right)) \\ &= \sum_{i \in S^A} \sum_{j \in S^G} \sum_{k \in S^*} \sum_{t=1}^{T} \log(P^*(i,j,k)) \Pr(O^A, O^G, X^A_t = i, X^G_t = j, X^*_{t+1} = k | \lambda')\end{aligned} \tag{A.4}$$

Equation A.4, will be maximized over the transition tensors shown as $\boldsymbol{P}$. There are two constraints for this maximization step:

- Summation of the transition probabilities must equal to 1:
  $\sum_{k=1}^{N^*} P^*\left(i,j,k\right) = 1, \quad i \in S^A, j \in S^G$

- Transition probabilities must be equal or greater than 0:
  $P^*\left(i,j,k\right) \geq 0$

The constrained maximization problem can be solved using Lagrange multipliers. During the derivation, it will become obvious that second constraint is unnecessary. If we rewrite the Equation A.4 with the Lagrange multipliers and take the derivative

of the related variables and equate them to 0 (with some abuse of notation):

$$\frac{\partial}{\partial P^*_{abc}}\left(\sum_{i,j,k,t} \log(P^*_{ijk}) \Pr(O^A, O^G, X^A_t = i, X^G_t = j, X^*_{t+1} = k|\lambda') - \chi\left(\sum_k P^*_{ijk} - 1\right)\right) = 0$$

$$\Rightarrow \frac{\sum_t \Pr(O^A, O^G, X^A_t = a, X^G_t = b, X^*_{t+1} = c|\lambda')}{P^*_{abc}} - \chi = 0 \tag{A.5}$$

We rearrange and sum over the next hidden states (c) to get:

$$\Rightarrow \sum_{c,t} \Pr(O^A, O^G, X^A_t = a, X^G_t = b, X^*_{t+1} = c|\lambda') = \chi \sum_c P^*_{abc}$$

The left hand side marginalizes out $X^*_{t+1} = c$ and the sum multiplied by χ in the right hand side is equal to 1 due to our constraint (take the derivative wrt χ and equate to zero to verify). Rearranging the equation:

$$\Rightarrow \chi = \sum_t \Pr(O^A, O^G, X^A_t = a, X^G_t = b|\lambda')$$

If we rearrange Equation A.5, change abc to ijk and plug in χ obtained from partial derivative steps we get:

$$P^*(i, j, k) = \frac{\sum_t \Pr(O^A, O^G, X^A_t = i, X^G_t = j, X^*_{t+1} = k|\lambda')}{\sum_t \Pr(O^A, O^G, X^A_t = i, X^G_t = j|\lambda')} \tag{A.6}$$

Equation A.6, shows the calculation of the transition probabilities using maximization step in EM algorithm. This approach requires to calculate probabilities in the equation. We use the help of the recursive forward and backward variables towards this end. The formula for these are given below (λ' removed from the equation to make them concise):

$$\alpha^*_t(i) = \Pr\left(X^*_t = i, O^*_{1:t}\right), \quad \text{forward variable} \tag{A.7}$$

$$\beta^*_t(i) = \Pr\left(O^*_{t+1:T} \middle| X^*_t = i\right), \quad \text{backward variable} \tag{A.8}$$

Recursive Forward Calculation: This step derives the formulas of α_t^* variables. This is done only for action parts; goal parts are symmetric. Recursive steps $(t = 1 \ldots T-1)$:

$$
\begin{aligned}
\alpha_1^A(k) &= \pi_k^A f_k^A(O_1^A) \\
\alpha_{t+1}^A(k) &= \Pr\left(O_{1:t+1}^A, X_{t+1}^A = s_k\right) \\
&= \Pr\left(O_{1:t+1}^A | X_{t+1}^A = s_k\right) \Pr\left(X_{t+1}^A = s_k\right) \\
&= \Pr\left(O_{1:t}^A | X_{t+1}^A = s_k\right) \Pr\left(O_{t+1}^A | X_{t+1}^A = s_k\right) \Pr\left(X_{t+1}^A = s_k\right) \\
&= \Pr\left(O_{1:t}^A, X_{t+1}^A = s_k\right) \Pr\left(O_{t+1}^A | X_{t+1}^A = s_k\right) \\
&= \Pr\left(O_{1:t}^A, X_{t+1}^A = s_k\right) f_k^A\left(O_{t+1}^A\right) \\
&= f_k^A\left(O_{t+1}^A\right) \sum_{i=1}^{N^A} \Pr\left(O_{1:t}^A, X_t^A = s_i, X_{t+1}^A = s_k\right) \\
&= f_k^A\left(O_{t+1}^A\right) \sum_{i=1}^{N^A} \Pr(O_{1:t}^A, X_{t+1}^A = s_k \mid X_t^A = s_i) \Pr(X_t^A = s_i) \\
&= f_k^A\left(O_{t+1}^A\right) \sum_{i=1}^{N^A} \Pr(O_{1:t}^A \mid X_t^A = s_i) \Pr(X_{t+1}^A = s_k \mid X_t^A = s_i) \Pr\left(X_t^A = s_i\right) \\
&= f_k^A\left(O_{t+1}^A\right) \sum_{i=1}^{N^A} \Pr(O_{1:t}^A, X_t^A = s_i) \Pr(X_{t+1}^A = s_k \mid X_t^A = s_i) \\
&= f_k^A\left(O_{t+1}^A\right) \sum_{i=1}^{N^A} \alpha_t^A(i) \sum_{j=1}^{N^G} \Pr(X_{t+1}^A = s_k, X_t^G = s_j \mid X_t^A s_i) \\
&= f_k^A\left(O_{t+1}^A\right) \sum_{i=1}^{N^A} \alpha_t^A(i) \sum_{j=1}^{N^G} \Pr(X_{t+1}^A = s_k \mid X_t^G = s_j, X_t^A = s_i) \Pr(X_t^G = s_j)
\end{aligned}
$$

$$
\alpha_{t+1}^A(k) = f_k^A\left(O_{t+1}^A\right) \sum_{i=1}^{N^A} \alpha_t^A(i) \sum_{j=1}^{N^G} P^A(i,j,k) \Pr(X_t^G = s_j) \tag{A.9}
$$

$$
\begin{aligned}
\Pr(X_1^G = s_j) &= \pi_j \\
\Pr(X_t^G = s_j) &= \sum_{u=1}^{N^A} \sum_{v=1}^{N^G} \Pr\left(X_{t-1}^A = s_u\right) \Pr\left(X_{t-1}^G = s_v\right) P^G(u,v,j)
\end{aligned}
$$

Equation A.9, shows the calculation of the forward variable.

Recursive Backward Calculation: This step derives the formulas of β^* variable. This will be done only for action states; goal states have the symmetric counterparts. Recursive steps ($t = 1 \ldots T-1$):

$$
\begin{aligned}
\beta_T^A(i) &= 1 \\
\beta_t^A(i) &= \Pr(O_{t+1:T}^A \mid X_t^A = s_i) \\
&= \sum_{k=1}^{N^A} \Pr(O_{t+1:T}^A, X_{t+1}^A = s_k \mid X_t^A = s_i) \\
&= \sum_{k=1}^{N^A} \Pr\left(O_{t+1:T}^A \middle| X_{t+1}^A = s_k, X_t^A = s_i\right) \Pr\left(X_{t+1}^A = s_k \middle| X_t^A = s_i\right) \\
&= \sum_{k=1}^{N^A} \Pr\left(O_{t+1:T}^A \middle| X_{t+1}^A = s_k\right) \Pr\left(X_{t+1}^A = s_k \middle| X_t^A = s_i\right) \\
&= \sum_{k=1}^{N^A} \Pr\left(O_{t+2:T}^A \middle| X_{t+1}^A = s_k\right) \Pr(O_{t+1}^A \middle| X_{t+1}^A = s_k) \Pr\left(X_{t+1}^A = s_k \middle| X_t^A = s_i\right) \\
&= \sum_{k=1}^{N^A} \beta_{t+1}^A(k) f_k^A\left(O_{t+1}^A\right) \sum_{j=1}^{N^G} \Pr\left(X_{t+1}^A = s_k, X_t^G = s_j \middle| X_t^A = s_i\right) \\
&= \sum_{k=1}^{N^A} \beta_{t+1}^A(k) f_k^A\left(O_{t+1}^A\right) \sum_{j=1}^{N^G} \Pr\left(X_{t+1}^A = s_k \middle| X_t^G = s_j, X_t^A = s_i\right) P\left(X_t^G = s_j \middle| X_t^A = s_i\right) \\
&= \sum_{k=1}^{N^A} \beta_{t+1}^A(k) f_k^A\left(O_{t+1}^A\right) \sum_{j=1}^{N^G} \Pr\left(X_{t+1}^A = s_k \middle| X_t^G = s_j, X_t^A = s_i\right) P\left(X_t^G = s_j\right) \\
\beta_t^A(i) &= \sum_{k=1}^{N^A} \beta_{t+1}^A(k) f_k^A\left(O_{t+1}^A\right) \sum_{j=1}^{N^G} P_{ijk}^A \Pr\left(X_t^G = s_j\right) \qquad \text{(A.10)}
\end{aligned}
$$

Equation A.10, shows the calculation of the backward variable.

We define two more variables, the denuminator and numerator of Equation A.6 which defines the transition tensors, to simplify the notation and derivations. For only the action part:

$$\begin{aligned}
\xi_t^A(i,j,k) &= \Pr\left(X_t^A = i, X_t^G = j, X_{t+1}^A = k | O^A, O^G\right) \\
&= \frac{\Pr\left(X_t^A = i, X_t^G = j, X_{t+1}^A = k, O^A, O^G\right)}{\Pr(O^A, O^G)} \\
&= \frac{\Pr(X_t^A = i, X_t^G = j, X_{t+1}^A = k, O^A, O^G)}{\sum_{i,j,k} \Pr(X_t^A = i, X_t^G = j, X_{t+1}^A = k, O^A, O^G)} \\
&= \frac{\alpha_t^A(i)\alpha_t^G(j)P_{ijk}^A f_{X_{t+1}^A}(O_{t+1}^A)\beta_{t+1}^A(k)\beta_t^G(j)}{\sum_{ijk} \alpha_t^A(i)\alpha_t^G(j)P_{ijk}^A f_{X_{t+1}^A}(O_{t+1}^A)\beta_{t+1}^A(k)\beta_t^G(j)}
\end{aligned} \tag{A.11}$$

$$\begin{aligned}
\varphi_t(i,j) &= \Pr\left(X_t^A = i, X_t^G = j | O^A, O^G\right) \\
&= \frac{\Pr\left(X_t^A = i, X_t^G = j, O^A, O^G\right)}{\Pr(O^A, O^G)} \\
&= \frac{\Pr(X_t^A = i, X_t^G = j, O^A, O^G)}{\sum_{i,j} \Pr(X_t^A = i, X_t^G = j, O^A, O^G)} \\
&= \frac{\alpha_t^A(i)\alpha_t^G(j)\beta_t^A(k)\beta_t^G(j)}{\sum_{ij} \alpha_t^A(i)\alpha_t^G(j)\beta_t^A(k)\beta_t^G(j)}
\end{aligned} \tag{A.12}$$

Finally, transition tensor probabilities can be calculated using Equation A.13:

$$P^*(i,j,k) = \frac{\sum_{t=1}^{T-1} \xi_t^*(i,j,k)}{\sum_{t=1}^{T} \varphi_t(i,j)} \tag{A.13}$$

At this point, everything is ready to calculate Equations A.9, A.10, A.11, A.12 and A.13. This means maximization step of the EM algorithm is derived.

These equations are for only one observation. If there are more than one observation, Equations A.9 and A.10 must be calculated for each observation seperately and ξ ve φ variables generated by them must be summed up. If we show an observation with d, equation becomes:

$$P^*(i,j,k) = \frac{\sum_d \sum_{t=1}^{T_d-1} \xi_t^*(i,j,k)_d}{\sum_d \sum_{t=1}^{T_d} \varphi_t(i,j)_d} \tag{A.14}$$