Normalizing Flows as HMM Emissions for Learning from Demonstration

by

Farzin Negahbani

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

February 18, 2022

Normalizing Flows as HMM Emissions for Learning from Demonstration

Koç University

Graduate School of Sciences and Engineering

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

Farzin Negahbani

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. Aykut Erdem

_____

Assoc. Prof. Sanem Sarıel

Date: _____

*To my **"Family"** that are the most precious ones in my life*

ABSTRACT

Normalizing Flows as HMM Emissions for Learning from Demonstration
Farzin Negahbani
Master of Science in Computer Science and Engineering
February 18, 2022

On top of being used in many different industries, robots are getting out of factories and into our everyday lives in the form of greeter robots, telepresence robots, toys, autonomous cars and perhaps most ubiquitously vacuum cleaners. Soon we may see more capable robots, such as mobile manipulators, helping us in our homes. Programming and controlling robots to achieve certain tasks in controlled industrial environments with field experts is significantly different than using them in everyday environments. Being able to program a robot to achieve desired tasks without the presence of an expert is of importance for the near future.

Imitation learning or Learning from Demonstration (LfD) field aims to enable robots to learn from humans. In this framework, instead of analytically deriving and manually programming a skill, the robot learns the desired skill from human demonstrations. Due to the human-interaction aspects, LfD needs to contend with a low amount of demonstrations which leads to a low amount of data. Using reinforcement learning on top of demonstrations is not feasible when there is no one to engineer a reward function, let alone have the setup for trial and error. Thus, extracting as much information as possible from a limited set of demonstrations and utilizing previously learned skills via transfer is an attractive option.

In the LfD framework of this thesis, action and goal/perceptual models of skills are learned from keyframes. Action models are used to execute the skill and goal models are used to monitor this execution. Hidden Markov Models (HMMs) and their derivatives are suitable to learn action and goal models from a low amount of keyframe demonstrations. The first part of the thesis introduces the State Traversal Transfer algorithm to facilitate transfer learning of skills for a single user. We show that this algorithm leads to successful transfer compared to learning from scratch for goal models but do not significantly increase action model performance.

However, HMMs have some limitations with transfer learning and multiple sources of data (e.g. multiple users, multiple objects for the same skill, etc.), especially about dealing with perceptual states. These limitations partially stem from using multivariate Gaussian emissions and the difficulty of choosing the correct number of hidden states.

Towards this end, a generative model called Conditional Flow Hidden Markov Model (C-FlowHMM) is designed by combining conventional HMMs, normalizing flows, and robotic specific adaptations to improve model flexibility in learning goal/perceptual models of skills. The idea is to use a single normalizing flow model, conditioned on hidden states, instead of Gaussians so that a more general emission model can be learned. By using a single model, states share information which is suitable in a low data regime. Furthermore, a neural network model is more amenable to transfer learning. We develop an expectation-maximization (EM) algorithm to train C-FlowHMMs from human demonstrations which lead to better execution monitoring performance compared to HMMs. We also show that C-FlowHMMs result in better transfer learning performance when data is more varied. Finally, we demonstrate that C-FlowHMM is more robust to change in the number of hidden states compared to conventional HMMs.

ÖZETÇE

Gösterimlerden Öğrenme için Normalleştiren Akış Emisyonlu Saklı Markov Modelleri
Farzin Negahbani
Bilgisayar Bilimleri ve Mühendisliği, Yüksek Lisans
18 Şubat 2022

Robotlar birçok farklı endüstride kullanılmalarının yanı sıra fabrikalardan çıkıp günlük yaşamlarımıza karşılama robotları, uzaktan bulunma robotları, oyuncaklar, otonom araçlar ve belki de en yaygın olarak elektrikli süpürgeler şeklinde girmektedir. Yakın zamanda mobil manipülatörler gibi daha yetenekli robotların, evlerimizde bize yardımcı olduğunu görebiliriz. Sabit ve kontrol altındaki endüstriyel ortamlarda robotları belirli görevler için uzmanlar aracılığı ile programlamak ve kontrol etmek, onları günlük ortamlarda kullanmaktan önemli ölçüde farklıdır. Bir robotu, herhangi bir uzman olmadan istenilen görevleri gerçekleştirecek şekilde programlayabilmek yakın gelecek için önem arz etmektedir.

Gösterimlerdenden Öğrenme (GÖ) alanı, robotların insanlardan öğrenmesini sağlamayı amaçlar. Bu çerçevede, bir beceriyi analitik olarak türetip elle programlamak yerine robot, istenen beceriyi insan gösterimlerinden öğrenir. İnsan etkileşimi yönleri nedeniyle, GÖ'nün düşük miktarda veri ile başarılı olması gerekir. Deneme yanılma için kurulum bir yana, ödül fonksiyonu tasarlayacak bir uzman olmadığında, pekiştirmeli öğrenme yapmak gerçekçi değildir. Bu nedenle, sınırlı bir dizi gösterimden mümkün olduğunca fazla bilgi çıkarmak ve daha önce öğrenilen becerileri aktarım yoluyla kullanmak çekici bir seçenektir.

Bu tezin GÖ çerçevesinde, önemli noktalardan hareket ve hedef/algısal beceri modelleri öğrenilir. Beceriyi yürütmek için eylem modelleri ve yürütmeyi izlemek için hedef modelleri kullanılır. Saklı Markov Modelleri (SMM'ler) ve türevleri, düşük miktarda önemli nokta gösteriminden eylem ve hedef modellerini öğrenmek için uygundur. Tezin ilk kısmı, tek bir kullanıcı için becerilerin aktarma yoluyla öğrenimini kolaylaştırmak için Durum Geçişi Aktarımı algoritmasını tanıtmaktadır. Bu algoritmanın, hedef modeller için sıfırdan öğrenmeye kıyasla daha başarılı olduğunu ancak eylem modeli performansını önemli ölçüde artırmadığını gösterilmiştir. Bununla

birlikte, SMM'lerin, özellikle algısal durumlar için, aktarımlı öğrenim ve çoklu veri kaynakları (örneğin birden çok kullanıcı, aynı beceri için birden çok nesne, vb.) kullanımı konularında bazı sınırlamaları vardır. Bu sınırlamalar kısmen çok değişkenli Gauss emisyonlarının kullanılmasından ve doğru sayıda gizli durum seçmenin zorluğundan kaynaklanmaktadır.

Bu amaçla daha esnek hedef modelleri öğrenmek için, SMM'leri, normalleştiren akış modellerini, ve robotlara özel uyarlamaları birleştirererek Koşullu Akış Saklı Markov Modeli (C-FlowHMM) adı verilen bir üretici model tasarlanmıştır. Buradaki fikir, daha genel bir emisyon modelinin öğrenilebilmesi için Gauss emistonları yerine gizli durumlara göre koşullandırılmış tek bir normalleştiren akış modeli kullanmaktır. Gizli durumlar, tek bir model kullanıldığı için bilgi paylaşmış olurlar ki bu düşük veri rejimine uygundur. Ayrıca, sinir ağı modelleri, aktarımlı öğrenme için daha uygundur. İnsan gösterimlerinden C-FlowHMM öğrenmek için bir "expectation-maximization (EM)" tabanlı algoritma türetilmiştir. Yapılan hedef modeli öğrenme deneylerinde, C-FlowHMM'nin, SMM'lere kıyasla daha iyi yürütme izleme performansına yol açtığı gösterilmiştir. Ayrıca, veriler daha çeşitli olduğunda C-FlowHMM'lerin daha iyi aktarım öğrenme performansı olduğu gözlemlenmiştir. Son olarak, C-FlowHMM'nin geleneksel SMM'lere kıyasla gizli durumların sayısındaki değişime daha dayanıklı olduğu bulunmuştur.

ACKNOWLEDGMENTS

First of All, I want to thank my supervisor, Prof. Barış Akgün, for all he has done for me during these years. In addition to supervising me, he has always been a good friend for me who advised me a lot during this period to find the best choices in my future career. Not only did I learn from him about research, but it was also a good chance for me to learn professional academic ethics from him during these years.

After that, I should appreciate the KUIS AI center of Koc University and the Scientific and Technological Research Council of Turkey (TÜBİTAK) that provided financial support for me to accomplish this thesis.

In the end, I should thank my family and my friends that supported me during this career. In this difficult situation that pandemic has been created in these years, their support meant a lot to me, and I would like to express my deep gratitude to them.

TABLE OF CONTENTS

List of Tables **xii**

List of Figures **xiii**

Abbreviations **xv**

Chapter 1: Introduction **1**

1.1 Thesis Statement 2

1.2 Thesis Organization 2

Chapter 2: Literature Review **3**

2.1 Learning from Demonstration 3

2.2 Density Estimation 4

2.3 Transfer Learning 5

Chapter 3: The Utilized Learning from Demonstration Framework **7**

3.1 Overview 7

3.2 Learning from Demonstration 7

3.3 Monitoring 8

3.3.1 Reversed Demonstrations as Negative Samples 9

3.3.2 NM Demonstrations as Negative Samples 10

3.4 Utilized Setup 10

3.5 Dataset 11

3.6 Summary 12

Chapter 4: State Traversal Transfer **14**

4.1 Overview 14

4.2 Motivation 14
4.3 Methodology 15
4.4 Results and Discussion 17
4.4.1 Experimental Setup 17
4.4.2 Goal Model Transfer 17
4.4.3 Action Model Transfer 21
4.5 Conclusion 21

Chapter 5: Conditional Flow Hidden Markov Model 23

5.1 Overview 23
5.2 Motivation 23
5.3 Problem Definition 24
5.4 Emission Model 24
5.5 Learning Method 25
5.5.1 Expectation Step 26
5.5.2 Maximization Step 26
5.5.3 Transition and Prior Probability Update Formulas 26
5.5.4 Conditional Coupling Layers 27
5.5.5 Emission Model Update Formula 28
5.6 Pseudocode 29
5.7 Convergence Criteria 29
5.8 Results and Discussion 30
5.8.1 Experimental Setup 30
5.8.2 Skill Learning Monitoring 32
5.8.3 Multi-object Skill Learning Monitoring 34
5.8.4 Skill Transfer Monitoring 34
5.8.5 Loss Function Plots and Convergence Criteria 36
5.9 Limitations 36

Chapter 6: Conclusion **40**

6.1 Future Directions 41

Bibliography **42**

Appendix A: Additional C-FlowHMM Results **48**

LIST OF TABLES

3.1 **Keyframe Demonstration Dataset Information.** 12

4.1 **Baseline vs. Transfer Execution avg. Success**. For the following calculations, failure accounted as 0, NM as 1, and success as 2 points. Results are average of 5 retries. Bold data shows outperforming model's result. 20

4.2 **Baseline vs. Transfer Execution Monitoring accuracy** calculated for robot execution experiments. Reported results are average of 5 retries with shuffling the data. Bold data shows outperforming model's result. 22

LIST OF FIGURES

3.1 **Demonstration**. The picture depicts a user providing close box skill demonstration with a robotic hand (Pand robot by Franka Emika). The robotic hand in white, an object of interest on top of the table in blue and a RGB-D scanner (Microsoft Kinect) on a stand to capture perceptual data. 8

3.2 **Perception Pipeline**. Utilized perception pipeline is demonstrated in this picture. Procedure starts by capturing raw data from RGB-D scanner and then feeding it to an object segmentation method to extract segmented object point cloud data. Next, the skill agnostic PCAE encodes the point cloud data to 128 dimension feature space. Based on our preference, we may reduce the resulted 128 dimension data to lower dimensions using a skill specific method like PCA. . . . 9

3.3 **Dataset Objects and Skills**. 11

3.4 **LfD Overview**. Process of kinesthetic teaching, learning models and then execution and monitoring are depicted as an overview of LfD. . . 13

4.1 **State Traversal Transfer vs. Baseline HMM**: Depicts comparison of state traversal transfer and baseline model in different transfer scenarios where in (a) 1D monitoring used and 2D monitoring employed in (b). 18

4.2 **With Near-Miss vs. W/O Near-Miss** : This plot investigates effect of utilizing NM demonstrations in monitoring threshold calculation where (a) is the case without NM case, and (b) is where we utilized NM. 19

5.1 **Sequential Data Notation**. To clarify our sequential data notations, we visualize the number of demonstrations (R) and length of each demonstration (T^r) in this figure. 25
5.2 **Induced Distribution Flow**. 25
5.3 **Conditional Coupling Layers**. Depicts how hidden state s is passed to each coupling layer to make them conditional. 29
5.4 **HMM modeled using GenHMM and C-FlowHMM**. 31
5.5 **Box1 Object Open and Close Skill Monitoring.** 32
5.6 **Box2 Object Open and Close Skill Monitoring.** 33
5.7 **Cumulative set of Box1 and Box2 Objects Skill Monitoring.** 35
5.8 **Skill Transfer Results.** To clarify, close / box2 -¿ box1 means transferring close skill where the source object is box2 and the target object is box1. 36
5.9 **Negative Log-probability of Sequences and Convergence**. To depict how the convergence criteria works, the iteration that convergence criteria meets with an arrow. 37
5.10 **Toy dataset samples.** 37
5.11 **3 State Conditional RealNVP Heat map and Samples** 38
5.12 **4 State Conditional RealNVP Heat map and Samples** 38
5.13 **Samples from Likelihood-guided Sampling.** 39

6.1 **Abstract concept of using a State Encoder** 41

A.1 **Sqr1 Object Open and Close Skill Monitoring.** 48
A.2 **Tall1 Object Close Skill Monitoring.** 49
A.3 **Bowl Object Pour Skill Monitoring.** 49
A.4 **Oct1 Object Open, Close, and Pour Skill Monitoring.** 50

ABBREVIATIONS

LfD	Learning from Demonstration
HMM	Hidden Markov Model
GMM	Gaussian Mixture Models
ROS	Robot Operating System
PCAE	Point Cloud Auto Encoder
PCA	Principle Component Analysis
NICE	Non-linear Independent Components Estimation
C-FlowHMM	Conditional Flow Hidden Markov Model
EM	expectation maximization
PbD	Programming by Demonstration
RL	Reinforcement Learning
i.i.d	Independent and identically distributed
LSTM	Long short-term memory

Chapter 1

INTRODUCTION

Programming by demonstration (PbD), imitation learning, or Learning from Demonstration (LfD) are some of the names for a common technique of teaching a robot new behaviours by utilizing human provided demonstrations[1]. First challenge of LfD is coping with low amount of data that stems from human-interaction aspects of LfD. By considering this low data regime, squeezing as much as information possible from very few data or by transferring it from existing learned skills is highly desirable.

Although approaches like Hidden Markov Models[2] (HMM) are proved to be promising in LfD framework given the low data regime, HMMs similar to traditional machine learning techniques do not scale well to high degrees of freedom [3] and no useful transfer learning methods are available for HMMs. As a result, moving in direction of finding a transfer method for HMMs is attractive yet experience showed such methods are highly engineered and limited. Also, experiences showed in non-linear and complex spaces with multi-user demonstrations, HMMs are prone to change in number of hidden states.

To obtain a desirable method for our problem and motivated by studies in literature[4], keeping nice structure of HMMs but changing their emission models seemed to be a probable direction. Neural network based solutions seems viable because of high learning capacity and being able to train with gradient-based optimizer techniques that provides the chance of network-based transfer[5, 6]. This neural network based candidate should be tractable and bijective to be plugged in a generative model like HMM. Normalizing flow models[7, 8] meet all of the mentioned criteria but they do not comply with few data regime. As a result, conditional nor-

malizing flow models with motivation of making the model lighter and providing shared knowledge, is picked as emission model of HMMs.

In this study, we intend to redesign HMMs by employing conditional normalizing flow models as their emission to customize them for few data regime of LfD and providing a better infrastructure for skill transfer learning.

1.1 Thesis Statement

A customized generative model based Hidden Markov Models with conditional normalizing flows as emissions leads to better performance for learning of goal/perceptual models from few demonstrations. In addition to learning more complex state distributions, this method is more robust to change in the number of hidden states and enables transfer learning of skills.

1.2 Thesis Organization

This thesis is organized into six chapters plus one appendix. The first chapter, the introduction chapter, introduces the problem and thesis statement followed by thesis organization. Reviewing related studies in learning from demonstration, density estimation, and transfer learning literature is done in chapter 2. Then chapter 3 gives an overview of learning from demonstration and provides more detail about the framework and dataset used in this thesis. In chapter 4, State Traversal Transfer, a transfer method for Hidden Markov Models, is explained. Next, chapter 5 covers the suggested approach for using in learning from demonstration based on conditional normalizing flows. Finally, Chapter 6 concludes main points of this thesis and ends with suggesting future directions.

Chapter 2

LITERATURE REVIEW

2.1 Learning from Demonstration

The development of various robots such as robotic arms, humanoids, mobile robots, soft robots[9], plus significant advances in sensors and perception, is helping the field of service of robots to grow[10]. Although robots are getting cheaper and more precise, acting flexibly in changing environments needs an outstanding intelligence or a degree of autonomy in programming them[11]. An efficient approach in programming robots is by providing human demonstrations. Learning to achieve skill by leveraging human-provided demonstrations or observing a human during a task execution are called learning from demonstration (LfD) or imitation learning[12].

Literature in applicable models in LfD is vast and is different based on the usage. Any approach that infers a policy that can complement or replace the imitation is called indirect learning methods[13]. In this line of work, family of traditional methods such as HMM[14, 15], Gaussian Mixture Model[16], and Graph-based methods[17] utilized especially for object manipulation task. However, even directly imitating the demonstrations are not always accurate and robust especially to unseen cases mostly due to two reasons. First, due to errors in data acquisition especially for cases that need precise movements[1]. Secondly, poor generalization of methods that can come from i.i.d assumption of training data, which is usually not the case for sequential actions of humans[18]. In addition, usually experts demonstrate only the successful way of achieving a task hence the method will be highly biased and lacks in generalization[19]. Reinforcement Learning[20] (RL) based approaches usually model the problem as an MDP to learn how to take action in a certain state space. RL framework can enjoy an initial policy learned from other traditional methods and further fine tune the policy using both positive and negative demonstrations[21]

yet is not without complications. As a result, any improvement in learning from the demonstration whether using fewer data or increasing the modeling capability, is complementary to RL methods not a substitute for RL.

Other groups of models such as recurrent neural networks and especially LSTM[22] are used to model demonstrations as sequential data. Even recently transformers are utilized in deep imitation learning for dual-arm robot manipulation task[23]. However, neural network based methods require a lot of data which is in contrast with most of imitation learning scenarios that are restricted to few data.

2.2 Density Estimation

A core problem in machine learning is estimation of $p(x)$ from a set of data samples $x_1, x_2, ..., x_n$ which is called density estimation. Ability to obtain a good estimation of density $p(x)$ is the key component to many machine learning tasks. However, this problem can be very hard due to issues like the curse of dimensionality, data scarcity, or having complex distribution. Conventional methods usually fall short while dealing with non-linear and complex distributions, while neural network-based methods showed promising results thanks to their learning capabilities. [24, 25, 8, 26] are examples of some pioneering methods that are designed to model natural images. A group of the neural network-based density estimation methods learn to model density evaluations as opposed to variational autoencoders[27] or generative adversarial networks[28]. Neural network density estimation approaches proposed for problems such as importance sampling[29], or even conditional density estimators employed as inference networks inside variational autoencoders[27, 30]. Among these methods, only two groups of neural network-based estimators are tractable, normalizing flows[31] and autoregressive methods[32]. Although autoregressive flow models like [33] has great learning capacities, sampling from such methods are not as efficient as evaluation of $p(x)$ thus using them inside generative models can be challenging.

Normalizing flow models apply a transformation on a base distribution to obtain the best fit on the target set of data. Normalizing flow models utilize transformations such that their Jacobian is tractable; hence the transformation is invertible[7, 8].

Such invertible transformation makes normalizing flow models a suitable candidate to employ within many architectures that need a density estimator. GenHMM [4] use a mixture of these flow models to improve the modeling capabilities of Hidden Markov Models (HMM). Despite the novelty of their idea, GenHMM model is not suitable for many applications where data points are few because of their model complexity. While on the other hand, an HMM can have a rough estimate even with very few data. Another issue that comes with normalizing flow models that work with a standard Gaussian as base distribution, is that induced distribution assumed to be a single deformed continuous volume in space. In fact this issue may not be troublesome in high-dimensional space, yet can bring challenges in fitting to disjoint clusters. Although stochastic normalizing flows[34] alleviate this issue further, but still there is a lot of room for improvements.

In this work, we are interested in neural network-based estimators that have invertible transformation and can be used within generative models yet are able to learn from a few data samples.

2.3 Transfer Learning

Transfer learning, in general, is learning to learn[35] or in machine learning, is focusing on reuse or transfer knowledge among domains. In other words, we as humans have the ability to generalize our experiences, and we are able to detect two situations as similar, thus generalizing the solution. For example, someone who is learned to play ping pong can usually learn badminton faster because of experiences obtained in controlling limbs and movement predictions. However, there might be various motivations for transfer learning when it comes to different fields.

Nowadays, many promising machine learning techniques are developed and used in practice, yet in many real-world problems, they face serious challenges. The best problem environments for machine learning techniques are the ones with plentiful data. However, in real-life scenarios, usually, data is expensive to collect, or very time-consuming, or nearly impossible to annotate [36, 37]. In literature, these problems are called insufficient training data[38] where in the case of deep learning, is

even more severe since these models are data hungry[39, 38]. Mentioned issues are motivations for transfer learning in many fields such as robotics and bioinformatics. So by considering a few data available, it is desired to reuse existing data or models with the aim of improving or lowering the need for data. For example, reinforcement transfer learning is used to alleviate the imbalanced-class issue for loss detection in power grids[40]. In a line of works, mapping-based deep transfer methods try to map data from source and target domain to a new domain where both source and target data are similar[41, 42, 43]. Another type of transfer learning methods are adversarial-based approaches that are inspired by generative adversarial networks[28]. These adversarial-based deep transfer learning methods search for a transferable representation that can be applied to both the source and target domains[44, 45, 46]. Network-based transfer can be defined as reusing whole or part of a network that is trained in the source domain by transferring its parameter weights, structures, or connections to another network which will be used for target domain problem[47, 48, 49]. So using a pre-trained model as the initial state and further fine-tuning it for another problem is counted as network-based transfer learning.

In the context of robotics within learning from the demonstration framework, we can define one of the transfer learning problems as such. Given a set of data to learn specific task, it is desired to learn similar or related tasks with fewer data. We then define source skill as a skill that enough amount of data for learning is present and target skill as the desired skill to be learned with fewer data by incorporating any information from source skill.

Chapter 3

THE UTILIZED LEARNING FROM DEMONSTRATION FRAMEWORK

3.1 Overview

In the following chapter, we cover the utilized learning from the demonstration framework in our work. At first, we define some common terms and then define different types of demonstrations. Next, monitoring in our framework alongside two methods of monitoring using reversed and Near-miss demonstrations are explained. The final section of this chapter depicts and explains our setup for imitation learning and ends with a conclusion of the chapter.

3.2 Learning from Demonstration

In our case, a demonstration is kinesthetically showing a robot to execute a skill by providing a trajectory or a set of keyframes[50] (kf) where keyframes can be defined as a sequence of critical points in the desired state space that following them allows the robot to perform a skill. After having demonstrations in the format of trajectory or keyframe demonstrations, a robot can use different models to learn execution and understand the meaning of successful execution in a task. A demonstration is called a successful demonstration if following them by a robot, by assuming no change in the environment compared to demonstration time, leads to successfully executing a task. On the other hand, if following a certain demonstration can achieve execution of a task up to a point but not fully successful, we call it Near-miss (NM) demonstrations. Fig.3.1 shows an expert providing demonstration with the used setup.

Each demonstration contains the robot's joint space (joint angles) and corresponding perceptual features with our setting. Then we can learn a model using the

Figure 3.1: **Demonstration**. The picture depicts a user providing close box skill demonstration with a robotic hand (Pand robot by Franka Emika). The robotic hand in white, an object of interest on top of the table in blue and a RGB-D scanner (Microsoft Kinect) on a stand to capture perceptual data.

robot's joint space or so-called action data, which provides us with an action model that can be used to perform the execution. Also, the model learned from sequential perceptual features is called the goal model that helps us to reason and monitor the skill execution. We are mainly focused on the goal model in this work, and in the next section, we dive deeper into the definition and methods of monitoring.

3.3 Monitoring

One critical usage of a learned model from demonstrations is to determine whether a given demonstration is successful or not. For example, in the case of applying a Reinforcement Learning method, we can leverage the output of such a model and formulate a reward function. To this end, we can calculate the forward log-likelihood of a demonstration, or more specifically, the sequence of perception states, and then decide on the success or failure of the execution. However, this threshold varies from one skill to another and drastically may change based on the change of the object. Another way is to use the probability of ending a skill in a hidden state (terminal

probability), yet we again need empirical threshold tuning in the presence of multiple terminal states. Also, relying only on terminal probability is not practical, especially in a very low data regime. To automate and improve the threshold picking procedure, we need negative and positive training samples to use a simple classifier. We utilized two different approaches, first, using reverse demonstrations as negative samples and second, using NM demonstrations as negative samples.

3.3.1 Reversed Demonstrations as Negative Samples

Here we can consider closing a box wherein the beginning the object is fully open and at the end, it is fully closed. By reversing the perceptual states of this demonstration, we obtain a set of demonstrations where the start and end states are reversed. As a result, calculating the forward log-likelihood value for such demonstration can give us an estimation of the negative sample to leverage and pick a threshold for monitoring.

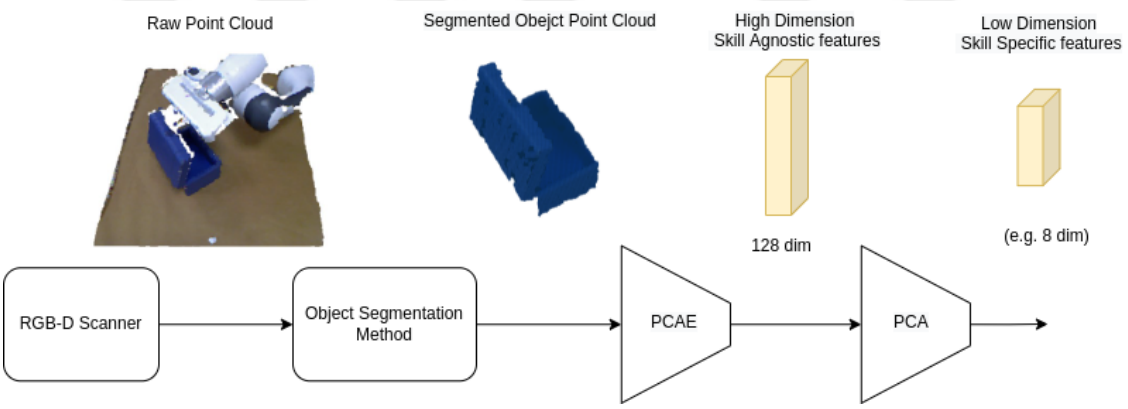


Figure 3.2: **Perception Pipeline**. Utilized perception pipeline is demonstrated in this picture. Procedure starts by capturing raw data from RGB-D scanner and then feeding it to an object segmentation method to extract segmented object point cloud data. Next, the skill agnostic PCAE encodes the point cloud data to 128 dimension feature space. Based on our preference, we may reduce the resulted 128 dimension data to lower dimensions using a skill specific method like PCA.

3.3.2 NM Demonstrations as Negative Samples

The assumption behind using reversed demonstrations is that end and start perceptual state of the object are not the same, which may not be the case in many skills like closing followed by opening a box. Although even in this case, transitions are different, still that is one issue with using reversed demonstrations. But most importantly, reversed demonstrations tend to give us a likelihood estimation of obvious failure cases, which most of the time is not the case. For example, imagine a robot tends to pour pasta inside a bowl; if the robot pours the whole pasta in a different place or doesn't pour anything at all, that is an obvious failure. However, in practice and especially during reinforcement learning, we observe that robots are able to perform tasks successfully up to a point like pouring most of the pasta inside the bowl. As e result, by picking threshold using reversed demonstrations, we can't easily catch these near-miss cases. So instead, we propose providing such NM demonstrations as an input during the teaching phase and then later leverage a set of NM demonstrations to obtain a better threshold automatically.

3.4 Utilized Setup

During each demonstration, synchronized robot joint states and perceptual features are being recorded as a demonstration. In the case of keyframe demonstrations, states are captured upon the user's request. Robot joint states are captured from the robot controller that communicates with our system through robot operating system (ROS) topics. We assume that our object of interest has a bluish color, but we make no other assumption on the type or shape of the object. A stream of RGB-D data in the form of point clouds is provided using the Microsoft Kinect device. The point cloud data is then fed into the object segmentation method that extracts points that belong to the object. Based on the state, size, and type of the object number of points in the segmented point cloud may vary, which is not desirable for applying many machine learning techniques. Plus, since point cloud data doesn't have a structure similar to images, we need to have permutation invariance features. To this end, a skill agnostic Point Cloud Auto Encoder (PCAE) is trained, which

applies to segmented objects and outputs a 128 dimension perceptual feature vector. Given our low data regime, this 128 dimension data is big, so based on our need, we may apply different skill-specific dimensionality reduction techniques such as Principle Component Analysis to reduce the size of the feature vector. Since PCA is fitted on demonstrations of a skill, it is called skill-specific, while the PCAE is used for all perception data regardless of the skill type. Fig.3.2 shows our perception pipeline utilized in our work.

3.5 Dataset

To evaluate suggested methods in this thesis, we collected a dataset of keyframe demonstrations that consists of 7 objects and 3 skills of closing, opening, and pouring. Perceptual states are encoded to 128 dimension skill agnostic features followed by the perception stack described in chapter.3. Fig.3.3 shows objects and available skills for each of them. To have diverse and comprehensive data, multiple expert users have provided both keyframe and NM demonstrations for some skills which more details in this regard in presented in table.3.5.

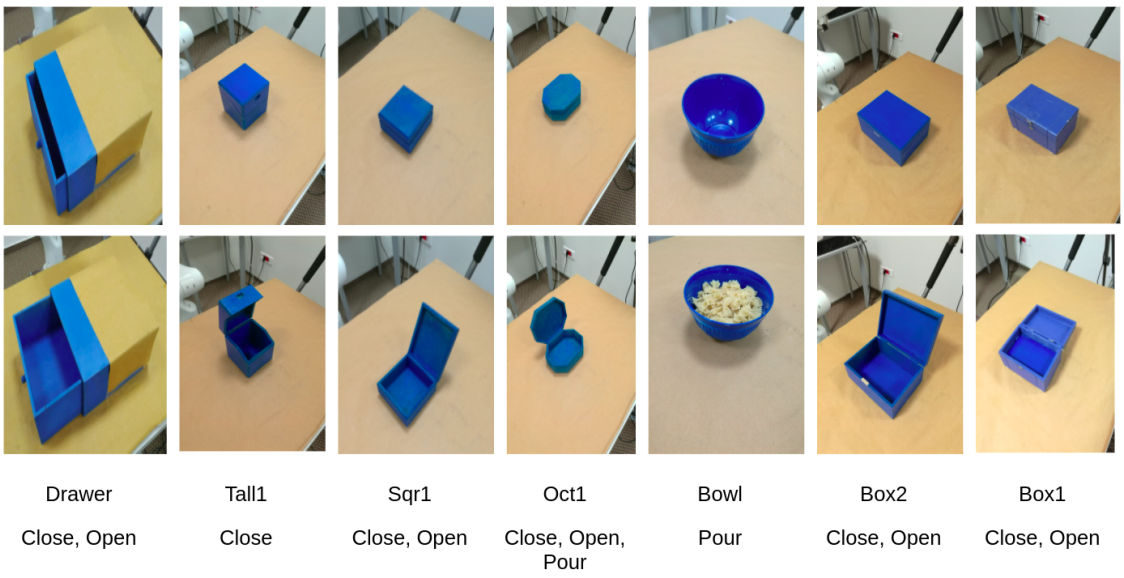


Figure 3.3: **Dataset Objects and Skills**. .

Table 3.1: **Keyframe Demonstration Dataset Information.**

Object Name	**Skill**						**# of Users**
	Close		**Open**		**Pour**		
	Success	Near-Miss	Success	Near-Miss	Success	Near-Miss	
Box1	40	40	40	40	-	-	4
Box2	40	40	40	40	-	-	3
Oct1	30	30	30	30	30	30	3
Sqr1	30	30	30	30	-	-	3
Drawer	20	20	20	20	-	-	2
Bowl	-	-	-	-	30	30	2
Tall1	10	10	-	-	-	-	1

3.6 Summary

Fig.3.4 shows the overview of LfD framework. First, a user provides demonstrations by moving the robot towards achieving a task and meanwhile recording action and perceptual data. Then based on given demonstrations, goal and action model are built. Then, action model can be leveraged to generate trajectory of joint angles that will be sent to the robot controller. Finally, during robot execution, perceptual data (also called perception data) are collected to determine success or failure (monitoring) of the execution with the help of the goal model.

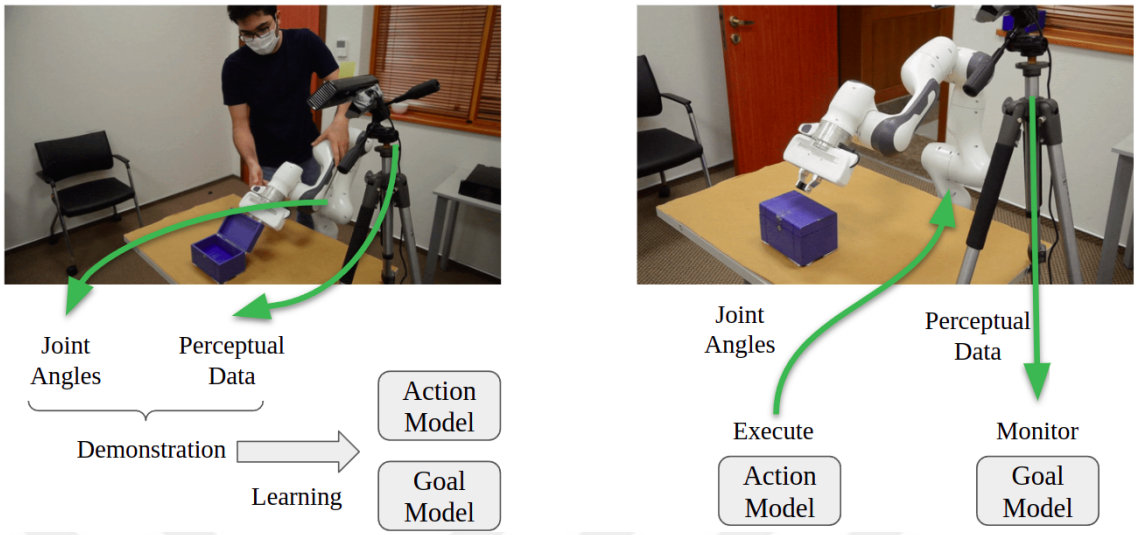


Figure 3.4: **LfD Overview**. Process of kinesthetic teaching, learning models and then execution and monitoring are depicted as an overview of LfD.

Chapter 4

STATE TRAVERSAL TRANSFER

4.1 Overview

The State Traversal Transfer chapter first introduces the motivation to look for a transfer learning method. Then we continue with defining some terms to share the same ground. Next, we go over our suggested method and demonstrate the State Traversal Transfer algorithm with a pseudocode. Finally, the skill monitoring methods of this algorithm are explained. Finally, experiment setup and results of State Traversal Transfer method are provided at the end of this chapter.

4.2 Motivation

Motivated by rich knowledge embedded in an HMM learned from enough amount of data, we aim to transfer some of this embedded knowledge to improve learning another task in a low data regime. Here we assume a transfer scenario where we have enough demonstrations for a specific skill and object (Source Skill), and given a few number of demonstrations, we want to learn another task (Target Skill) iteratively. Indeed a level of similarity is expected between source and target skill. We observed that in learning a skill in case of very few data, emission estimation is critical, while on the other hand, having a rough estimation of the transition model can be done relatively easier. So we aim to find a method that can iteratively update the emission model of an HMM and learn transitions using fewer data by reusing related and similar data.

Algorithm 1: HMM State Traversal Transfer Algorithm

Input: Target Demonstrations (D), Source Skill HMM (HMM_S),Likelihood threshold ($likelihood\_th$)

`/* Emission (`$Mean(\mu)$`,` $Covariance(Z)$`)` `*/`

Result: Transferred Target HMM (HMM_T)

$HMM_T \leftarrow HMM_S$

for *demo in* D **do**

  $log\_prob \leftarrow log\_pdf(\ HMM_T.emissions, demo)$

  $decoded\_seq \leftarrow argmax(log\_prob)$

  for kf *in demo.keyframes* **do**

    `/* Update a current state emission */`

    if $log\_prob[kf] \geq likelihood\_th$ **then**

      $\mu_{new} \leftarrow \mu_{old} + \alpha \times (kf - \mu_{old})$

      $Z_{estimated} \leftarrow (\mu_{new} - kf)(\mu_{new} - kf)^T$

      $Z_{new} \leftarrow Z_{old} + \alpha \times (Z_{estimated} - Z_{old})$

    end

    `/* Add a new state */`

    else

      $\mu_{new} \leftarrow kf$

      $Z_{new} \leftarrow Average(statesZ)$

    end

  end

end

Run EM by fixing the emissions to learn transitions

4.3 Methodology

As mentioned, we observed that emissions are of utmost importance for transfer. In our case, we assume HMMs with multivariate Gaussian distributions as emission model. So given the scenario of a very low data regime, we want to incrementally

update the covariance matrix and mean of emission model of a source HMM and build a candidate HMM for the target task (transferred HMM). For now, we assume that demonstrations are from a single user, and they are not vastly different, so a level of similarity between demonstrations is expected.

First, we learn the source HMM using all demonstrations provided for the source skill. Then we drop the source HMM transitions and treat the states of source HMM as only different multivariate Gaussian distributions. Next, we traverse keyframes in each demonstration; for each step, the log-probability of the current keyframe with respect to all states is calculated. This gives us a measurement of how likely a keyframe belongs to an existing state to be a candidate for an update. However, there might be cases in a keyframe that may not belong to any of the current states, and in fact, we need to add a new state. For this, a lower limit empirically is set for log-probability values to decide whether a keyframe belongs to an existing state in source HMM or not.

So, if a keyframe belongs to an existing state, we first update the mean using eq.(4.1), then using the updated mean(μ_{new}, we estimate a new covariance based on eq.(4.2) and finally we update the current covariance matrix using eq..(4.3). Otherwise, a new state is added with a mean (μ) equal to the keyframe point and an average of existing distributions covariances (Z) as this new state initial covariance. After traversing all keyframes in the given demonstrations, we fix the emissions and learn the transition model using the baum-welch [2] algorithm. The notion behind update equations is to gradually update the emission model where the factor α tunes this speed. State Traversal Transfer algorithm is demonstrated in Algorithm.1.

$$\mu_{new} = \mu_{old} + \alpha \times (kf - \mu_{old}) \tag{4.1}$$

$$Z_{estimated} = (\mu_{new} - kf)(\mu_{new} - kf)^T \tag{4.2}$$

$$Z_{new} = Z_{old} + \alpha \times (Z_{estimated} - Z_{old}) \tag{4.3}$$

4.4 Results and Discussion

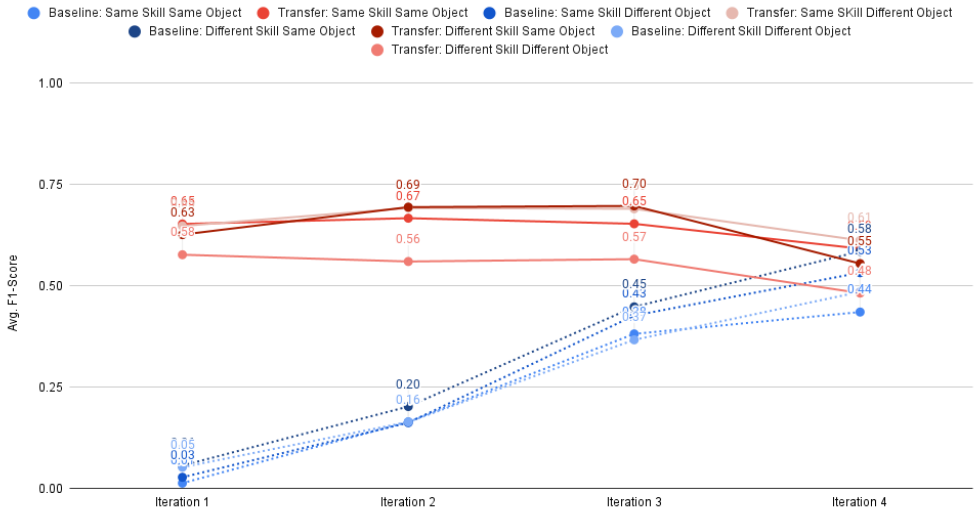
4.4.1 Experimental Setup

State Traversal Algorithm was designed to achieve transfer for single-user data. So during State Traversal experiments, we only use demonstrations from a single user. As shown in algorithm.2, we iteratively go over demonstrations, and in each iteration, we perform our evaluation to compare the results and obtain the trend of transfer versus the number of target skill data. Also, as a baseline to compare with, we have an HMM with multivariate Gaussian emissions that are fitted on target skill data. In other words, we want to investigate whether HMM transfer scenario helped us improve over the baseline or not. We pick new data from the test set and add it to the trainset in each iteration. For instance, in an experiment with 10 success and 10 NM demonstrations, in the case of using NM for picking threshold, in the first iteration, we have only 1 train success and 1 NM while we have 9 success and 9 NM test data.

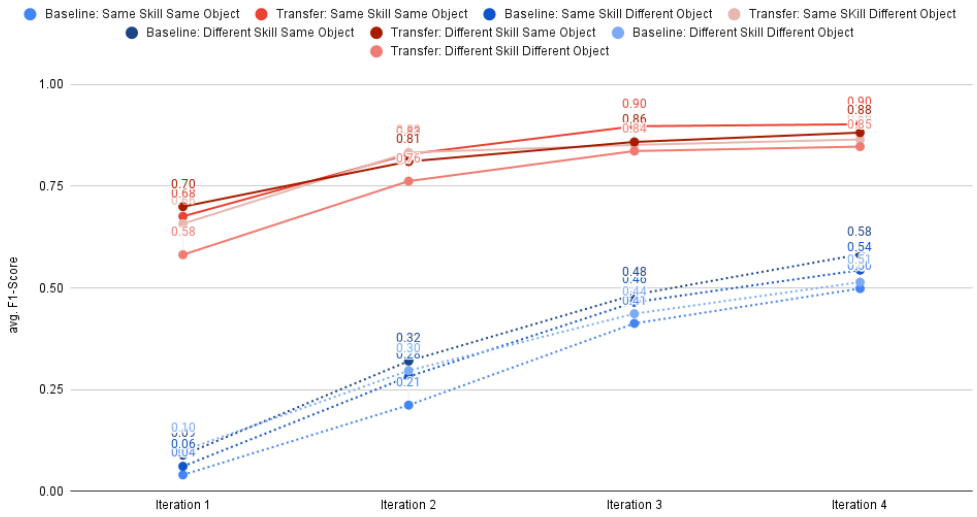
4.4.2 Goal Model Transfer

We evaluate the monitoring performance of transferred and baseline models for goal model transfer in terms of average F1-score. For this purpose, we leverage the forward likelihood of a demonstration and terminal probability to classify whether a skill was successful or not. To be practical, we pick this threshold using training demonstrations that are provided during each iteration. However, they are all successful demonstrations and can be highly biased towards the trainset. Thus, we provide the same number of NM demonstrations as the training set and use them as unsuccessful samples as we explained in 3. Since we only use single-user demonstrations, we can have many experiment variations. For example, transfer from box1, user1, close skill, to box1, user2, close skill. In this regard, we group experiments in 4 different scenarios:

- Same Skill Same Object (different user demonstrations)
- Same Skill Different Object (can be same or different user demonstrations)



(a) 1D Monitoring

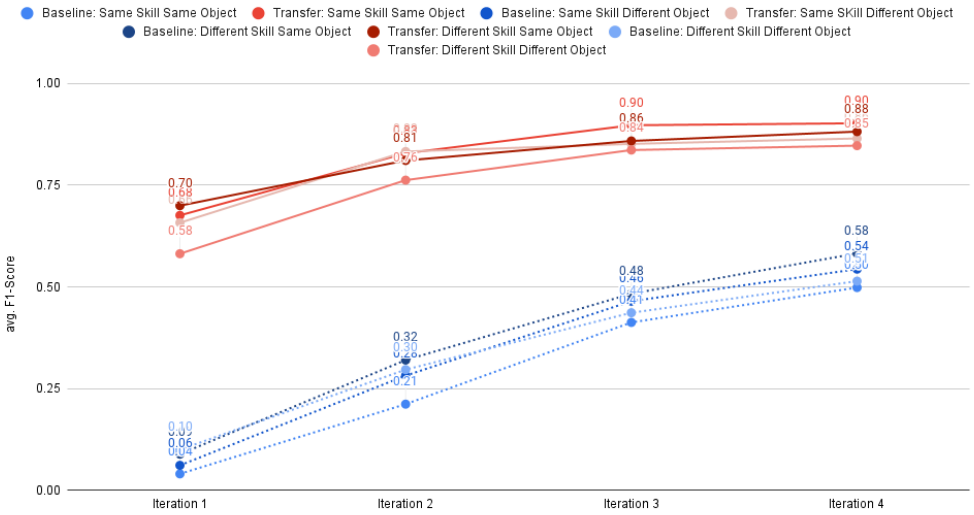


(b) 2D Monitoring

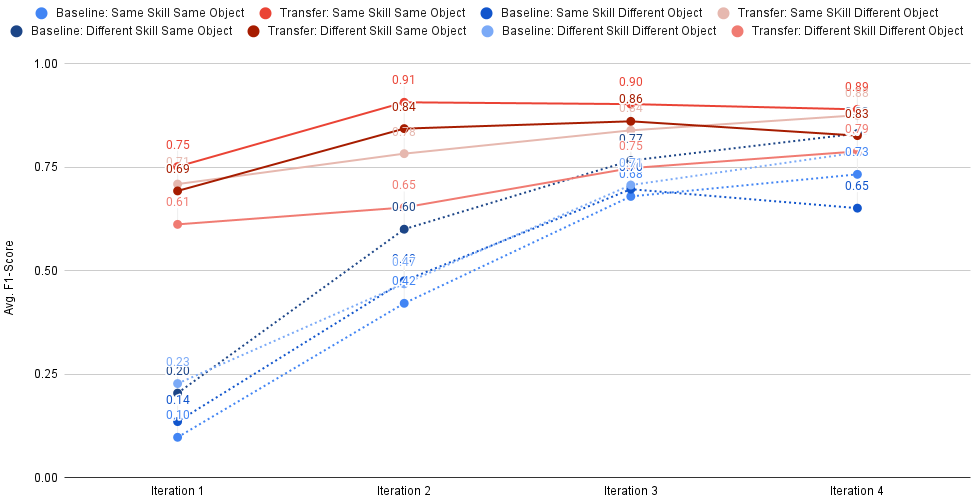
Figure 4.1: **State Traversal Transfer vs. Baseline HMM**: Depicts comparison of state traversal transfer and baseline model in different transfer scenarios where in (a) 1D monitoring used and 2D monitoring employed in (b).

- Different Skill Same Object (can be same or different user demonstrations)
- Different Skill Different Object (can be same or different user demonstrations)

Fig.4.1 demonstrates the state traversal transfer and baseline HMM results. In Fig.4.1a, we observe monitoring results comparison where $forward\_likelihood$ $\times$



(a) Without Near-Miss



(b) With Near-Miss

Figure 4.2: **With Near-Miss vs. W/O Near-Miss** : This plot investigates effect of utilizing NM demonstrations in monitoring threshold calculation where (a) is the case without NM case, and (b) is where we utilized NM.

terminal_probability used as a 1 dimension value to use in monitoring while in Fig.4.1b, forward and terminal probability used as a 2 dimension vector for monitoring. As the plots suggest, state traversal transfer has proved useful in almost all scenarios. In addition, using the 2 dimension vector of forward and terminal probability showed to provide more flexibility in monitoring.

Table 4.1: **Baseline vs. Transfer Execution avg. Success**. For the following calculations, failure accounted as 0, NM as 1, and success as 2 points. Results are average of 5 retries. Bold data shows outperforming model's result.

Transfer Experiment	Model	Iteration 1	Iteration 2	Iteration 3	Iteration 4
Box1 → Box2 Close	Baseline	0.5	0.5	1	1.75
	Transfer	**1.75**	**1.75**	**2**	**2**
Box2 → Box1 Close	Baseline	**2**	1.5	1.5	1.25
	Transfer	1.75	**1.75**	**2**	**2**
Box1 → Box2 open	Baseline	2	1.5	2	**2**
	Transfer	2	**1.75**	2	1.75
Box2 → Box1 open	Baseline	1.25	1	1.25	1
	Transfer	**2**	**1.75**	**1.5**	**1.5**

Next, now that we have observed 2D monitoring performs better compared to the 1D monitoring, we repeat the same experiments, but this time, instead of using near-misses only as negative samples, we split them to train and test sets similar to what we do for successful demonstrations to further use the near-miss training for improving the monitoring threshold calculation. Results of these experiments are provided in fig.4.2. Overall, we can infer that using near-miss demonstrations instead of reversed demonstration in success threshold calculation improves the baseline and state traversal transfer results. For example, we can see using near-miss helped the method to achieve higher performance in the first and second iteration in Same Skill Same Object and Different Skill Same Object while decreasing the performance of Same Skill Different Object and Different Skill Different Object transfer scenarios. Also, we should consider that by using NM we are accepting to increase the cost of providing near-miss demonstrations as well in which it may not always be feasible to provide natural near-misses by a demonstrator. Yet, having near-miss samples helps us achieve a more realistic monitoring evaluation as reversed demonstrations are hugely different.

4.4.3 Action Model Transfer

Similar method of transfer leveraged for transferring Action model. In this case, our evaluation metrics are being able to achieve the task with or without transfer successfully. To this end, we picked a subset of experiments and performed the action model transfer, while at the end of each iteration, we take the mean of action HMM and generate a mean trajectory for execution. Then an expert user observes the execution and decides whether it was a success, NM, or total failure. In addition to action transfer, we do goal transfer and monitor the execution to evaluate our goal model transfer again. Also, since we work with keyframe demonstrations and models are trained in a similar manner, perceptual states during executions are picked based on passing through action model keyframes.

Table.4.1 demonstrates the average score of baseline and transferred model executions by scoring success as 2 points, NM as 1 point, and failure as 0. In addition table.4.2 compares monitoring accuracy of a baseline HMM and the HMM obtained from state traversal method. As best performing results in bold format show, we obtained consistent results with the offline goal model transfer experiments. Although State Traversal Transfer method showed improvement compared to the baseline in transferring goal model, it is not without limitations and no noticeable improvement observed in transferring action model. This can be from the fact that Gaussian assumption for action data is a reasonable assumption. As a result we chose the path of goal model to transfer and improve.

4.5 Conclusion

This chapter has provided path and motivations towards a transfer method for HMMs. We suggested a simple approach to iteratively updating a current HMM with multivariate Gaussian emissions. Indeed we assume that the HMM is used for goal or action model and learned from keyframe demonstrations of a single user. We also inferred that:

- Learning from various or multiple user demonstrations needs a lot of fine tuning in number of hidden states

Table 4.2: **Baseline vs. Transfer Execution Monitoring accuracy** calculated for robot execution experiments. Reported results are average of 5 retries with shuffling the data. Bold data shows outperforming model's result.

Transfer Experiment	Model	Iteration 1	Iteration 2	Iteration 3	Iteration 4
Box1 → Box2 Close	Baseline	0	0.5	0.5	1
	Transfer	**0.75**	**1**	**1**	1
Box2 → Box1 Close	Baseline	0.25	1	0.75	0.75
	Transfer	**1**	**0.75**	**1**	0.75
Box1 → Box2 open	Baseline	0.75	0.75	0.75	0.75
	Transfer	**1**	**1**	**1**	**1**
Box2 → Box1 open	Baseline	0.5	**0.75**	0.25	**1**
	Transfer	**0.5**	0.5	**1**	0.5

- Transferring goal model showed to have more room to improve especially in presence of mixed data compared to action model that was without significant improvement.

Chapter 5

CONDITIONAL FLOW HIDDEN MARKOV MODEL

5.1 Overview

After defining learning from the demonstration framework and the transfer learning motivations plus suggesting the State Traversal Transfer method as a starting point toward our aim, here we suggest a model called Conditional Flow Hidden Markov Model (C-FlowHMM) that is designed based on experiences that emerged from previous chapters. First, we recite more motivations and then define the problem, and finally, we suggest our solution, followed by details of the algorithm and mathematical derivations and then experiment details and related discussions. This chapter ends with a discussion about an observed limitation of normalizing flows.

5.2 Motivation

HMMs have been shown to be powerful in learning sequential data and learning the structure even in low data regimes. However, the emission model assumption, which in our case is multivariate Gaussian distribution, brings about shortcomings, especially when the distribution of data is complex. In addition, we approached to transfer and update the emission models and faced many challenges and had to make many simplifying assumptions. So we are motivated to replace the emission model of HMMs with a neural network density estimation model to improve the model's flexibility. Towards this end, normalizing flow models such as RealNVP[8] which is an extension of NICE[7], is a suitable candidates. First, it is simple, has fewer assumptions on the density model, and is tractable. This path is motivated by a recent work of Liu et al.[4] work that used a mixture of RealNVP models as each state emission. But this model is huge hence not suitable for a very low data regime. As a result, we aim to redesign HMM by leveraging robotic domain knowledge for

using the model as a flexible and powerful model in the LfD framework.

5.3 Problem Definition

Towards having a generative model with flexible emission models, we suggest realizing HMM emission model by a conditional RealNVP model conditioned on hidden states of the HMM. As such, we define our problem domain.

Assuming that our sequential data is $\underline{\mathrm{X}} = [x_1, x_2, .., x_T]$ where $x_t \in \mathbb{R}^N$. We are interested in finding an HMM $H \in \mathcal{H}$ where $\mathcal{H}$ is the space of all possible solutions. Note that T is the number of keyframes in the demonstration X or, in other words, the length of the sequence. Now we define the followings:

- $\mathcal{H} = \{H | H = \{S, \pi, A, p(x|s;\theta)\}$ where $\mathcal{H}$ is solution space.
- Set of hidden states of H is S.
- A is the transition matrix of states in H with size $|S|\times|S|$ where $\forall i, j \in S$, $A_{i,j} = p(s_{t+1} = j | s_t = i)$
- $\pi = \{\pi_1, \pi_2, ..., \pi_{|S|}\}$ are states prior probability distribution.
- Sample x_t is assumed to be generated using $p(x_t|s_t;\theta)$ where θ is parameters of emission model.

Next, we formulate C-FlowHMM in detail, but we should keep in mind that our aim is to build a probabilistic model that induces $p(X;H)$.

5.4 Emission Model

As mentioned we denote a demonstrations as a sequential data $\underline{\mathrm{X}} = [x_1, x_2, .., x_T]$ where $x_t \in \mathbb{R}^N$ assuming that they follow an unknown distribution $\hat{p}(x)$. As Fig.5.1 visually demonstrates, we assume having R demonstration or sequences of data where each can have different length that is denoted by T_r. Next, we replace the $p(x|s;\theta)$ with a conditional RealNVP generator as shown in (5.1) where $g : \mathbb{R}^N \rightarrow$

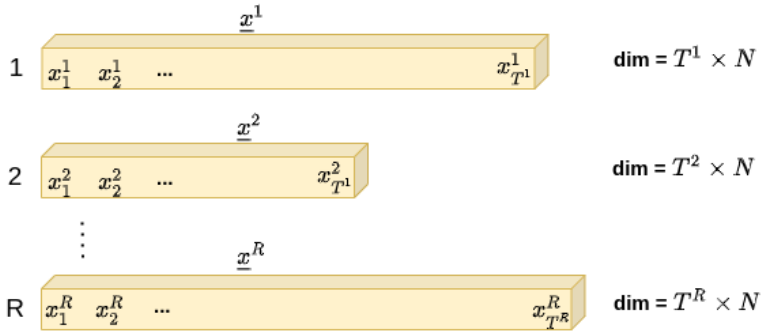


Figure 5.1: **Sequential Data Notation**. To clarify our sequential data notations, we visualize the number of demonstrations (R) and length of each demonstration (T^r) in this figure.

$\mathbb{R}^N$ is a transformation in which $x = f(z, s)$. Please not that we assume that g is invertible so $f^{-1}(x, s) = z$ holds. Here z is a latent variable with density function $p_{base}(z)$ such that $z \in \mathbb{R}^N$. p_{base} can be any known parametric distribution but her we assume it is a Standard Multivariate Gaussian distribution.

$$p(x|s;\theta) = p_{base}(z)|det(\frac{\partial f(z,s)}{\partial z})|^{-1} \tag{5.1}$$

Fig.5.2 demonstrates a signal flow of our suggested model where a sample x is transformed sample of p_{base} distribution given hidden state s.

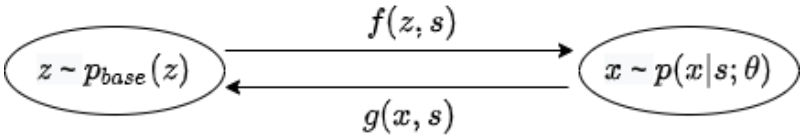


Figure 5.2: **Induced Distribution Flow**.

5.5 Learning Method

Given the following definitions, we want to learn parameters of C-flowHMM such that it estimates $\hat{p}(x)$ with $p(x; H)$ or in the other word maximizing the likelihood of $p(x; H)$ for all sequences of data as shown in eq.(5.2).

$$arg\max_{H\in\mathcal{H}} \frac{1}{R}\sum_{r=1}^{R} logp(\underline{\mathrm{x}}^r; H) \tag{5.2}$$

Note that $\underline{\mathrm{s}} = [s_1, s_2, .., s_T]$ is the sequence of hidden states corresponding to observations $\underline{\mathrm{x}}$. Finding a closed form solution is not possible for problem(5.2) hence we approch this problem in expectation maximization (EM) framework.

5.5.1 Expectation Step

First we calculate joint posterior probability of observation sequence $\underline{\mathrm{x}}$ and corresponding hidden state sequences. Then we calculate expected likelihood of $\hat{p}(x)$ and $p(s|x; H)$ using eq.(5.3).

$$\mathcal{L}(H) = E_{\hat{p}(x),p(s|x;H)}[logp(x|s; H)] \tag{5.3}$$

5.5.2 Maximization Step

In maximization step we need to maximize eq.(5.4) to update emission model(5.7), transition matrix(5.6) and prior probability distribution(5.5).

$$\max_{H} \mathcal{L}(H) = \max_{\pi} \mathcal{L}(\pi; H) + \max_{A} \mathcal{L}(A; H) + \max_{\theta} \mathcal{L}(\theta; H) \tag{5.4}$$

So the maximization step divides into three separate problem which are formulated below.

$$\mathcal{L}(\pi; H) = E_{\hat{p}(\underline{\mathrm{X}}),p(s|\underline{\mathrm{X}};H)}[logp(s_1; H)] \tag{5.5}$$

$$\mathcal{L}(A; H) = E_{\hat{p}(\underline{\mathrm{X}}),p(s|\underline{\mathrm{X}};H)}[\sum_{t=1}^{T-1} logp(s_{t+1}|s_t; H)] \tag{5.6}$$

$$\mathcal{L}(\theta; H) = E_{\hat{p}(\underline{\mathrm{X}}),p(s|\underline{\mathrm{X}};H)}[logp(\underline{\mathrm{x}}|s; H)] \tag{5.7}$$

5.5.3 Transition and Prior Probability Update Formulas

Since transition and prior probability update problems are known and, in fact, based on our architecture, same with an HMM model, derivations are already available in the literature, so we provide the resulted update equations.

Eq.(5.8) demonstrates the transition probability update rule by considering that $A_{i,j} = p(s^r_{t+1} = j | s^r_t = i; H)$ is definition of transition matrix entities.

$$A_{i,j} = \frac{\sum_{r=1}^{R} \sum_{t=1}^{T-1} p(s^r_t = i, s^r_{t+1} = j | \underline{\mathrm{x}}^r; H)}{\sum_{k=1}^{|S|} \sum_{r=1}^{R} \sum_{t=1}^{T-1} p(s^r_t = i, s^r_{t+1} = k | \underline{\mathrm{x}}^r; H)} \tag{5.8}$$

In addition, eq.(5.9) is the initial probability update update formula. Note that posterior probability calculation is also a known problem and in this study we used forward-backward[2] algorithm to calculate them.

$$\pi_i = \frac{1}{R} \sum_{r=1}^{R} p(s^r_1 = i | \underline{\mathrm{x}}^r; H), \forall i \in 1, 2, 3, ..., |S| \tag{5.9}$$

5.5.4 Conditional Coupling Layers

Now to obtain the solution to (5.7), we need to provide more details about conditional normalizing flows. Generator f is realized by concatenation of L-layer neural network that maps z given s to x where $f(., s) = f_L \circ f_{L-1} \circ ... f_1$ and f_l is the l-th layer. We need this mapping to be invertible. The concatenated mapping is invertible if the mapping of each layer is invertible; hence we have $f^{-1}(x, s) = z$. Carefully designed neural network layers suggested in NICE[7, 8] and extension of it in [8] are such layers and suitable for our purpose. In this work, we use the extended version suggested in RealNVP that provides both more flexibility in learning as well as an easy way of calculating the determinant of the Jacobian of this transformation by leveraging the change of variable formula in (5.1). Layers suggested in RealNVP paper are called affine coupling layers where each transformation is defined as follows given our $N - dimension$ observation x where y is the output of transformation, and $\odot$ is element-wise product.

$$y_{1:n} = x_{1:n} \tag{5.10}$$

$$y_{n+1:N} = x_{n+1:N} \odot exp(s(x_{1:n})) + t(x_{1:n}) \tag{5.11}$$

Here, $S(.)$ and $t(.)$ are function that map $\mathbb{R}^n \rightarrow \mathbb{R}^N - n$ space. If we try to derive the inverse transformations in (5.10) and (5.11) we can simply obtain the followings:

$$x_{1:n} = y_{1:n} \tag{5.12}$$

$$x_{n+1:N} = (y_{n+1:N} - t(y_{1:n})) \odot exp(-s(y_{1:n})) \tag{5.13}$$

There are two main points here; first, we can see based on (5.12) and (5.13) that inverse and forward transformations cost equal computations, and this is very desirable for our generative model where inference and generation can have the same cost. Secondly, transformation obtained from such coupling layer has known Jacobian that is presented in (5.14), and it is triangular.

$$\frac{\partial y}{\partial x} = \begin{bmatrix} \mathbb{I}_d & 0 \\ \frac{\partial y_{n+1:N}}{\partial x_{1:n}} & diag(exp(s(x_{1:n}))) \end{bmatrix} \tag{5.14}$$

As a result, we can observe that determinant of this Jacobian does not need $s(.)$ or $t(.)$ to be invertible and can be any complex functions which, in our case, we use multi-layer neural networks. Finally, the Jacobian determinant of the concatenated transformation can be obtained by multiplying each coupling layer's determinant(5.15) and hence similarly for the inverse of it.

$$det(A \circ B) = det(A)det(B) \tag{5.15}$$

To make such layers conditional, we follow a common practice that is used in deep neural networks, which is the concatenation of condition with the input of the layer. We should note that in practice, indexing inputs fed to shift ($s(.)$) and translation ($t(.)$) functions are achieved by masking the input. We follow the same approach, but indeed we do not mask the condition passed to each layer. Fig.5.3 shows a concept of consecutive conditional coupling layers used in this work.

5.5.5 Emission Model Update Formula

Now that we have defined our conditional normalizing flow model, we can continue to solve (5.7) that takes us to maximizing the cost function of our neural network based generator in (5.17).

$$\mathcal{L}(\theta; H) = \frac{1}{R} \sum_{r=1}^{R} \sum_{s^r} p(s^r | \underline{\mathbf{x}}^r; H) \sum_{t=1}^{T^r} logp(x_t^r | s_t^r; H) \tag{5.16}$$

$$= \frac{1}{R} \sum_{r=1}^{R} \sum_{s^r} \sum_{s_t^r=1}^{|S|} p(s_t^r | \underline{\mathbf{x}}^r; H) logp(x_t^r | s_t^r; H) \tag{5.17}$$

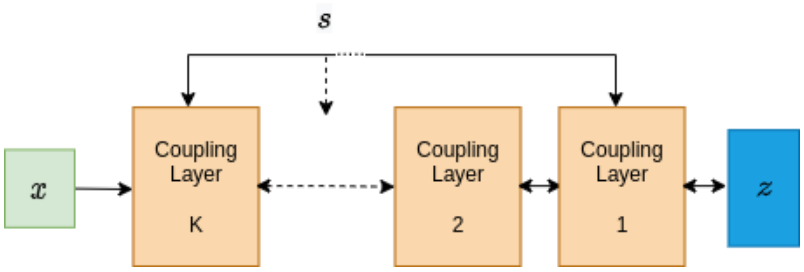


Figure 5.3: **Conditional Coupling Layers**. Depicts how hidden state s is passed to each coupling layer to make them conditional.

If we name $f^{-1} = g$ and calculate the state posterior values $p(s_t|\underline{x}; H)$ using forward-backward approach we can get:

$$= \frac{1}{R} \sum_{r=1}^{R} \sum_{s^r} \sum_{s_t^r=1}^{|S|} p(s_t^r|x^r; H)[logp_{base}(g(x_t^r, s)) + \sum_{l=1}^{L} log|det(\nabla g^l(x_t^r, s))|] \tag{5.18}$$

The equation in (5.18) is the emission model cost function of a C-FlowHMM that can be optimized using stochastic gradient descent. In fact, being able to learn the emission model of C-FlowHMM using gradient updates provides a way of network-based transfer for our method in which we update a current C-FlowHMM by fine-tuning it with new data.

5.6 Pseudocode

Finally with having all the update rules and formulating our cost function we demonstrate the C-FlowHMM method training algorithm in EM framework in algorithm.2.

5.7 Convergence Criteria

One important hyperparameter that usually is picked empirically is the number of training iterations. Since our method is explicitly designed to be a lightweight model and work in a low data regime, the number of iterations for training can vary based

Algorithm 2: Learning method of FlowHMM

Input: Data from unknown distribution $\hat{p}(x)$, Learning Rate (η)

Result: FlowHMM params

Initialize $H \in \mathcal{H}$ results:

$H = \{S, \pi, A, p(x|s;\theta)\}$

while *H not converged* **do**

`/* Expectation Step */`

Sample a batch of data $\{\underline{\mathrm{X}}\}_{r=1}^{R_b}$ from $\hat{p}(x)$ with batch size R_b

Compute posterior $p(s_t^r|\underline{\mathrm{x}}^r; H)$

Calculate loss $\mathcal{L}(\theta, H)$

`/* Maximization Step */`

$\partial\theta \leftarrow \nabla_\theta \mathcal{L}(\theta, H)$

$\theta \leftarrow \theta + \eta \times \partial\theta$

$\pi \leftarrow \arg\max_\pi \mathcal{L}(\pi)$

$A \leftarrow \arg\max_A \mathcal{L}(A, H)$

end

return H

on the number of training data. So instead, we suggest convergence criteria instead of deciding on the number of iterations. We propose leveraging change in forwarding log-probabilities of training sample in each iteration so that if this difference goes lower than a threshold, we stop the training.

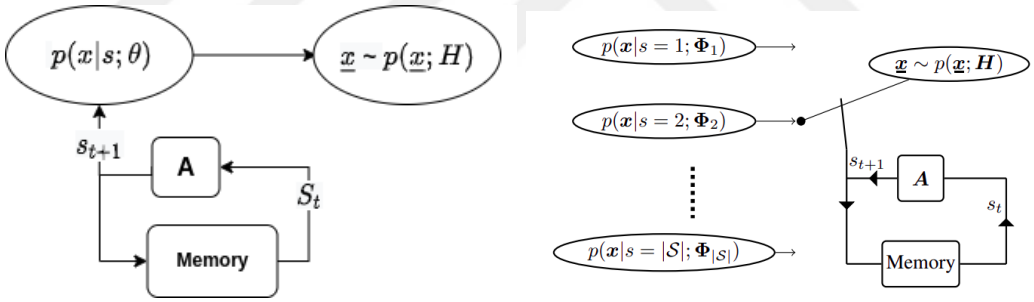
5.8 Results and Discussion

5.8.1 Experimental Setup

To evaluate the C-FlowHMM method for goal model transfer, we used the introduced dataset consisting of various demonstrations from 7 objects and 3 skills. However, contrary to state traverse experiments, here we combine demonstrations from all users to build more significant and various sets of data for our experiments. Since

each user may achieve a skill differently; as a result, different occlusion cases and perceptual object states happen, which makes our experiment scenarios more realistic.

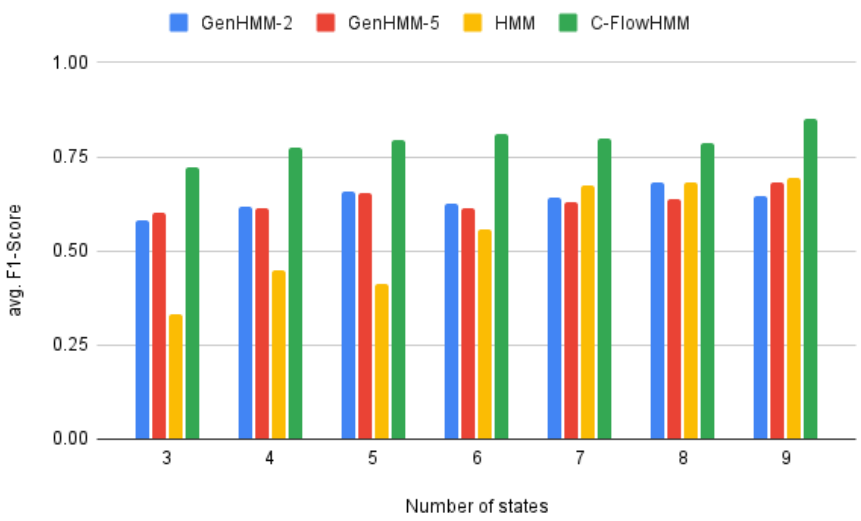
One motivation behind designing C-FlowHMM was to alleviate the issue with choosing the number of hidden states as we know HMM is highly dependent on this hyperparameter. Doing all experiments with changing the number of hidden states from 3 to 9 gives us a more fair comparison between the baseline HMM and C-FlowHMM. The reason behind stopping at 9 is that almost all of the demonstrations in our dataset have lower than 9 keyframes, so stopping until 9 hidden states is reasonable. In the following experiments, we randomly pick 30% of success and NM demonstrations for training while the rest will be used for testing. As the evaluation measure, similar to state traversal transfer, the average F1-score of monitoring is reported where experiments are performed 5 times.



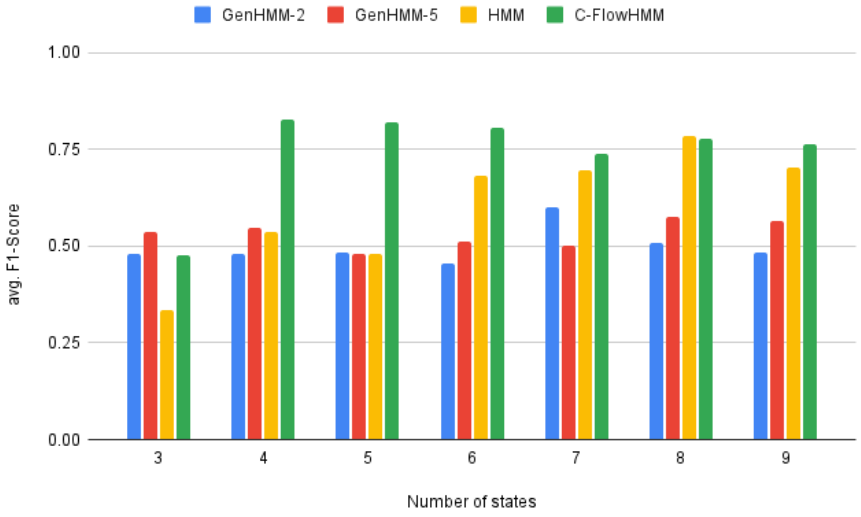
(a) C-FlowHMM (b) GenHMM

Figure 5.4: **HMM modeled using GenHMM and C-FlowHMM**.

In the following experiments, in addition to the HMM, we use the GenHMM method as the second baseline to compare our method with. Although, we should mention that the GenHMM model is not designed for our low data regime case, and the comparison might not be completely fair. In this regard, we use the GenHMM model with 2 and 5 generative components (mentioned as parameter K in their work), and we denote it as GenHMM-2 and GenHMM-5, respectively. Fig.5.4 illustrates a graphical comparison between the HMM obtained from GenHMM and C-FlowHMM model. While GenHMM uses a mixture of normalizing flow models for



(a) Close Skill Monitoring



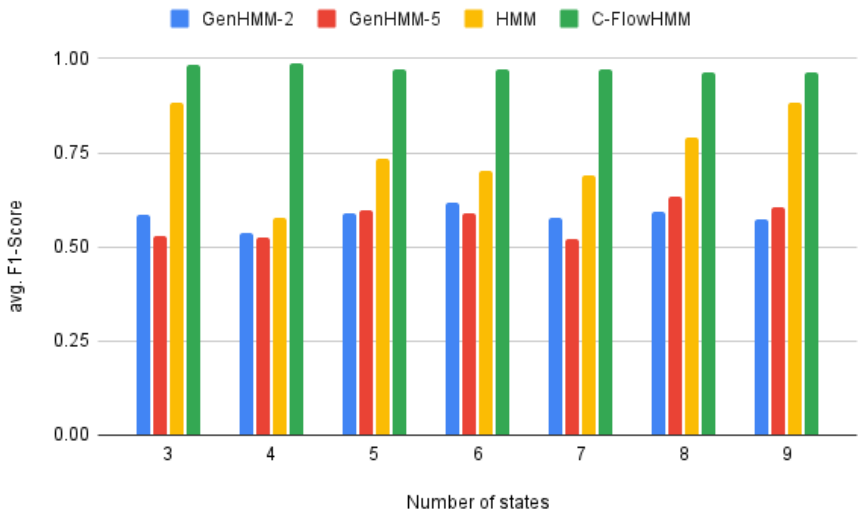
(b) Open Skill Monitoring

Figure 5.5: **Box1 Object Open and Close Skill Monitoring.**

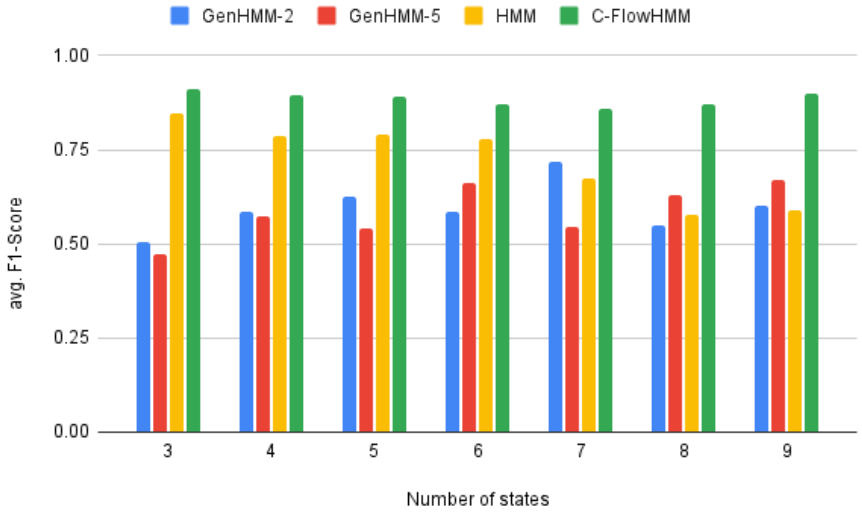
each hidden state emission, C-FlowHMM employs a single conditional normalizing flow to represent all emission distributions.

5.8.2 Skill Learning Monitoring

Since GenHMM results are not comparable to the baseline and C-FlowHMM, in fact, were not designed for our problem, GenHMM results are added to a subset of



(a) Close Skill Monitoring



(b) Open Skill Monitoring

Figure 5.6: **Box2 Object Open and Close Skill Monitoring.**

experiments to provide enough notion of the model performance. Note that more skill learning monitoring results are provided in Appendix A for the space structure and space constraints.

As figures 5.5, 5.6, A.4, A.1, A.2, and A.3 show, overall C-FlowHMM performs better in terms of avg. F1-score of monitoring. Also, we can claim that C-FlowHMM is more robust to number of hidden states compared to GenHMM and HMM yet it

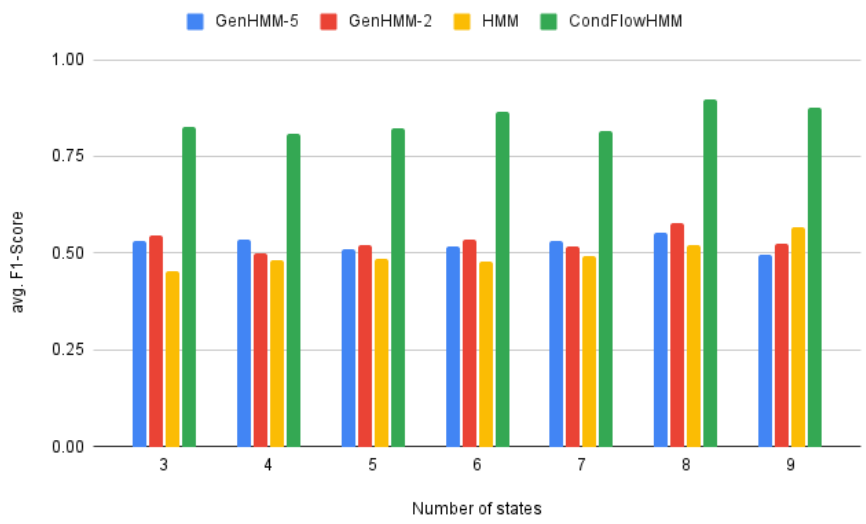
doesn't remove the need of an expert to estimate this hyperparameter empirically. For example, 5.5b presents that with 3 hidden states C-FlowHMM improved baseline yet fell short in competing with GenHMM approach suggestion that 3 states is not enough. This is completely dependent on variety of object perceptual state in demonstrations. Fig.5.6a and fig.5.6b further demonstrate this fact since box2 object has much less variation in perceptual states compared to box1.

5.8.3 Multi-object Skill Learning Monitoring

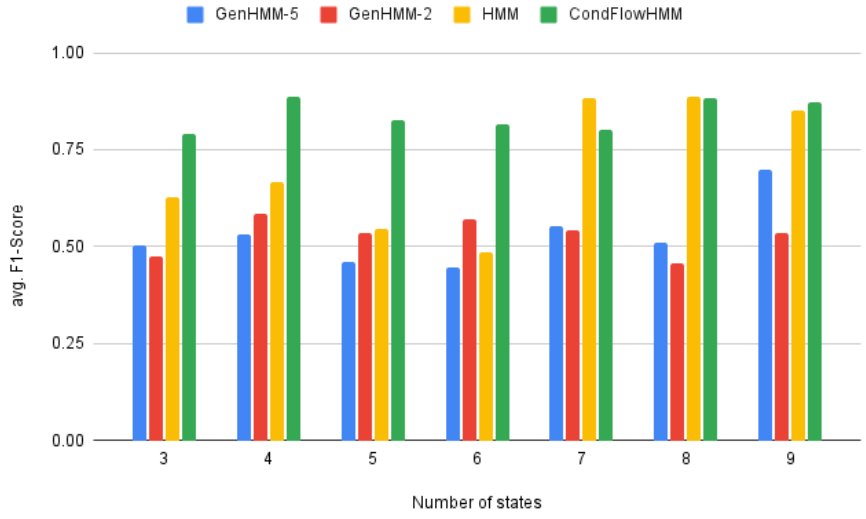
In addition to learning skills from an object, we investigate whether we are able to learn the same skill for similar objects or not. To this end, demonstrations from box1 and box2, that has the highest similarity in shape and dimensions among dataset objects, are combined. Next, a similar monitoring experiment was performed to evaluate this hypothesis. In this scenario, due to the diversity of perceptual states, issues related to the number of hidden states become more severe. As fig.5.7 plots, C-FlowHMM is able to handle the data thanks to the neural network generators employed in the model. On the other hand, the baseline shows great change in performance by change in the number of hidden states and, in fact, after having enough hidden states, is able to compete with C-FlowHMM.

5.8.4 Skill Transfer Monitoring

As opposed to state traversal transfer, now we have better infrastructure to move toward transfer. In this scenario, we learn a C-FlowHMM from a source skill, and then we try to use the resulted method as the initial state for learning the target skill. For learning source skill, again 30% of demonstrations used while for target skill only 15% of demonstrations is used to fine-tune the model on the target domain. Then we compare the transferred model with an HMM and C-FlowHMM trained with only 15% of target skill demonstrations. Similar to the previous experiments, avg. F1-score of monitoring with 5 retries are reported. As fig.5.8 suggests, a level of transfer is achieved, and the suggested transfer model performs better than HMM baseline. However, in two of the experiments, we only achieved comparable results



(a) Close Skill Monitoring



(b) Open Skill Monitoring

Figure 5.7: **Cumulative set of Box1 and Box2 Objects Skill Monitoring.**

compared to the C-FloHMM baseline. Yet, the investigated transfer method is naive and is only to show that with C-FlowHMM, directions for transfer can be explored more.

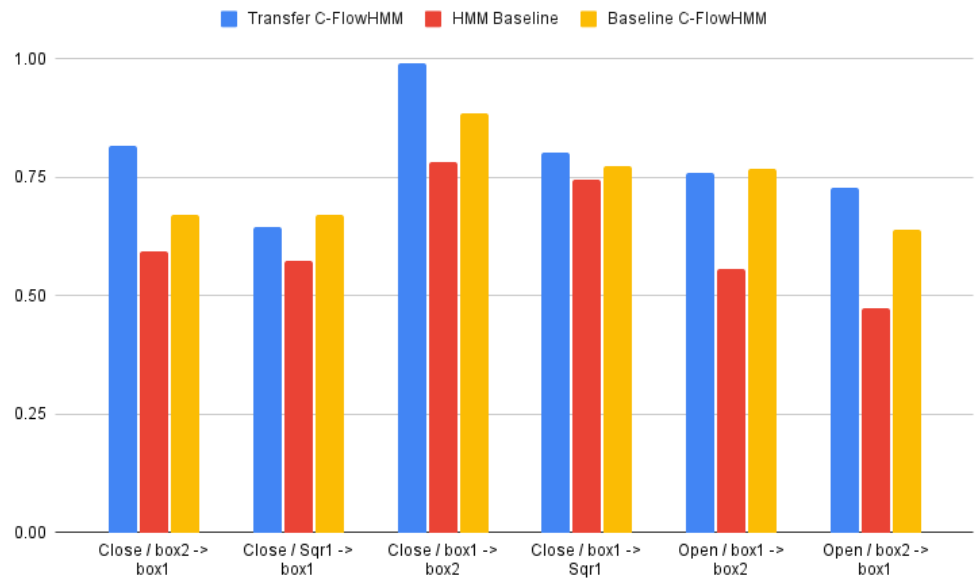


Figure 5.8: **Skill Transfer Results.** To clarify, close / box2 -¿ box1 means transferring close skill where the source object is box2 and the target object is box1.

5.8.5 Loss Function Plots and Convergence Criteria

Fig.5.9 demonstrates negative log-probability of demonstrations during training. On aim of C-FlowHMM is being able to distinguish between NM and success demonstrations. So, here we plot same values for train success, near-miss, and a validation set from the success demonstrations which only is used here for visualization purposes. Here, the mode is trained for a number of iterations but the iteration that convergence criteria is met is distinguished with an arrow.

5.9 Limitations

As mentioned during introduction of density estimation methods, normalizing flow models are tractable and equipped with high learning capacities. However, when assuming a standard Gaussian as base distribution, induced distribution inherits features of base Gaussian distribution. Few of the studies investigated this issue mostly due to usage of these methods in high dimension data. But inclined with our work, we present this limitation with a small toy dataset and discuss why it is

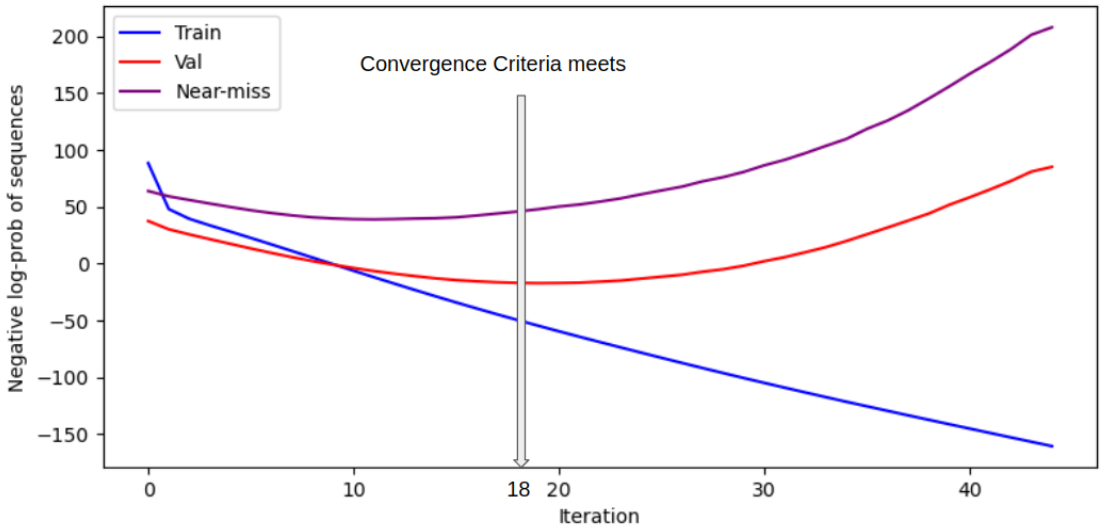


Figure 5.9: **Negative Log-probability of Sequences and Convergence**. To depict how the convergence criteria works, the iteration that convergence criteria meets with an arrow.

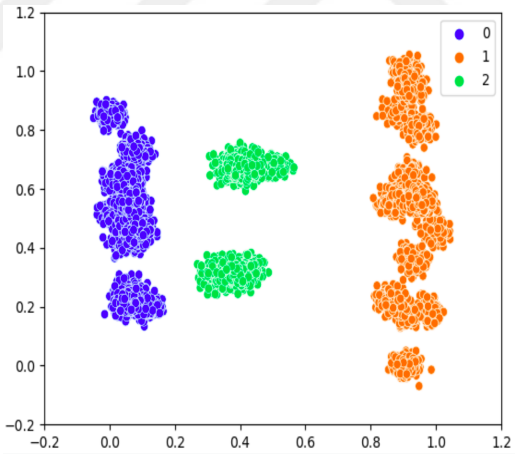
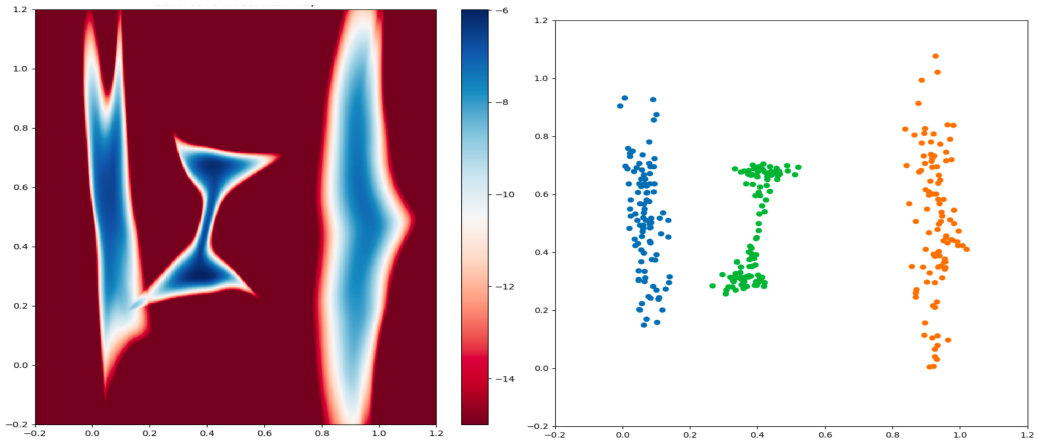


Figure 5.10: **Toy dataset samples.**

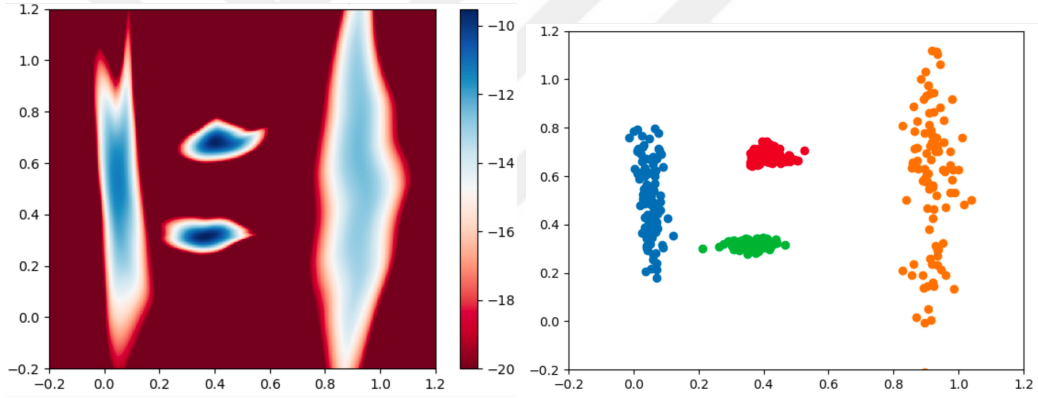
important in our field especially for modeling action states.

Fig.5.10 shows a toy dataset with 4 clusters and 3 states where points belonging to each state is represented with the same color. Let us imaging these are data points belonging to a task that can be achieved in 2 ways. All valid sequences start with cluster in left, then goes to one of the middle clusters, which both of them are from a same state, and then ends with the rightmost cluster. Now, lets say we are



(a) Induced Heat map (b) Sampled point

Figure 5.11: **3 State Conditional RealNVP Heat map and Samples**



(a) Induced Heat map (b) Sampled point

Figure 5.12: **4 State Conditional RealNVP Heat map and Samples**

interested in learning this task with conditional normalizing flow models. The aim is to obtain two separate induced distributions for the middle clusters. Fig.5.11a depicts the learned heat map of the generative model and fig.5.11b shows sampled points from this generative model. It is obvious that our base assumption imposes a limitation in achieving separate clusters in the middle while fig.5.12 illustrates how this models can fit perfectly on data in presence of enough number of states and we believe this issue is more critical in dealing with action data of any skill that can be achieved in multiple ways.

A simple workaround can be leveraging the density evaluations of generated

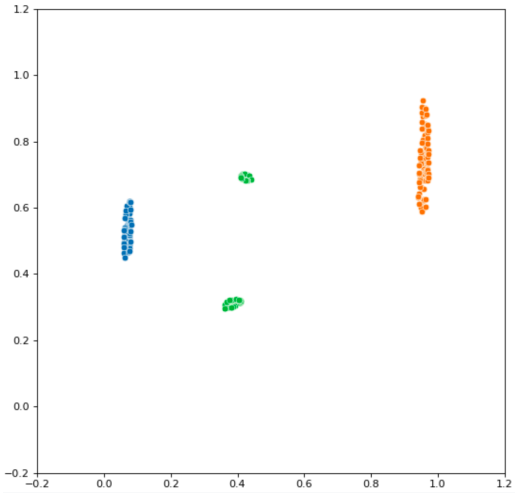


Figure 5.13: **Samples from Likelihood-guided Sampling.**

samples or as we call it likelihood-guided sampling. For this, we first generate enough samples then we only pick samples with high density values estimated by the model itself. However as shown in fig.5.13 this will harshly limits the samples to very high likely space and cannot model the distribution of data properly.

Chapter 6

CONCLUSION

In this thesis, we first presented a method for iterative transfer of HMM models, and we showed that using transfer in a very low data regime can help boost the monitoring performance of the goal model for single user demonstrations. In addition, we presented that using demonstration likelihood and terminal probability as a 2D feature vector improves the monitoring performance in terms of avg. F1-score. However, we observed that this approach is only suitable for a very low data regime and is not a substitute for the HMM, especially for the action model. In addition, we presented that using NM demonstrations instead of reversed demonstrations leads to better monitoring, but it will increase the effort of demonstrations needed.

Next, given our problem in a low data regime, efforts have been made to leverage promising conventional methods like HMM and further customize them with our field knowledge. In this line, C-FlowHMM is suggested, which is an HMM powered by neural network-based generators. Here, conditional normalizing flows are employed as the emission model to simplify the model while bringing more modeling flexibility. Then learning method of C-FlowHMM is suggested in the framework of expectation maximization. As the experiments present, C-FlowHMM has the capability of training in the presence of a low number of data and improve monitoring tasks for our existing dataset. Next, we showed that C-FlowHMM is more robust to change in the number of hidden states compared to HMM, and we further certified that by learning the same skill for multiple similar objects. Finally, we investigated a naive and simple approach for transfer using C-FlowHMM that showed their possibilities of skill transfer, but more experiments and specific transfer methods based on C-FlowHMM may be needed. We believe that this thesis represents the fact that with field-specific designs and combining conventional methods with deep neural network modeling power can make such models applicable for few data.

6.1 Future Directions

There are avenues for improvement and future work, but we illustrate a few here. Other than similarity in perceptual states of objects, there are similarities in order of execution to achieve a specific task. As a result, the investigation of paths to re-use transitions for transfer in C-FlowHMM seems interesting. In addition, in case of having more data with more complex or various transitions, replacing the transition matrix of C-FlowHMM with a recurrent method can be another direction. We represent this abstract idea in fig.6.1 where the hidden state of C-FlowHMM is decided using a recurrent model. Another line of improvement can be replacing the conditional normalizing flow model of emission with a stochastic conditional normalizing flow with aim of alleviating the seperate cluster issue.

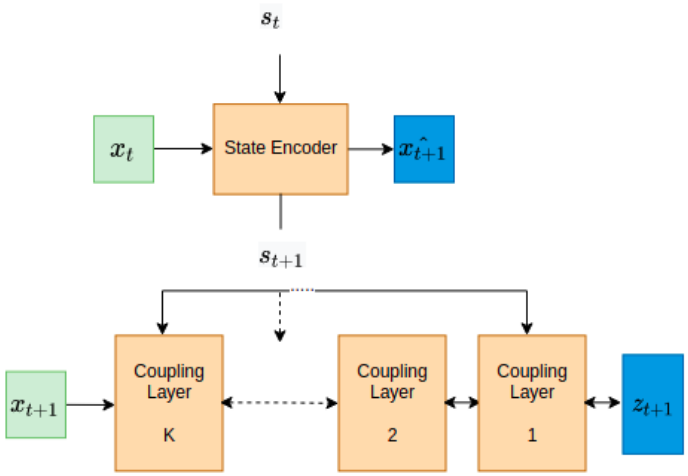


Figure 6.1: **Abstract concept of using a State Encoder**

BIBLIOGRAPHY

[1] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.

[2] L. Rabiner and B. Juang, "An introduction to hidden markov models," *ieee assp magazine*, vol. 3, no. 1, pp. 4–16, 1986.

[3] J. Kober and J. Peters, "Imitation and reinforcement learning," *IEEE Robotics Automation Magazine*, vol. 17, no. 2, pp. 55–62, 2010.

[4] D. Liu, A. Honoré, S. Chatterjee, and L. K. Rasmussen, "Powering hidden markov model by neural network based generative models," *arXiv preprint arXiv:1910.05744*, 2019.

[5] H. Zhu, M. Long, J. Wang, and Y. Cao, "Deep hashing network for efficient similarity retrieval," in *Proceedings of the AAAI conference on Artificial Intelligence*, vol. 30, 2016.

[6] D. George, H. Shen, and E. Huerta, "Deep transfer learning: A new deep learning glitch classification method for advanced ligo," *arXiv preprint arXiv:1706.07446*, 2017.

[7] L. Dinh, D. Krueger, and Y. Bengio, "Nice: Non-linear independent components estimation," *arXiv preprint arXiv:1410.8516*, 2014.

[8] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real nvp," *arXiv preprint arXiv:1605.08803*, 2016.

[9] G. M. Whitesides, "Soft robotics," *Angewandte Chemie International Edition*, vol. 57, no. 16, pp. 4258–4273, 2018.

[10] S. Ivanov, U. Gretzel, K. Berezina, M. Sigala, and C. Webster, "Progress on robotics in hospitality and tourism: a review of the literature," *Journal of Hospitality and Tourism Technology*, 2019.

[11] G.-Z. Yang, J. Bellingham, P. E. Dupont, P. Fischer, L. Floridi, R. Full, N. Jacobstein, V. Kumar, M. McNutt, R. Merrifield, *et al.*, "The grand challenges of science robotics," *Science robotics*, vol. 3, no. 14, p. eaar7650, 2018.

[12] S. Schaal, "Learning from demonstration," *Advances in neural information processing systems*, vol. 9, 1996.

[13] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, pp. 1–35, 2017.

[14] T. Asfour, P. Azad, F. Gyarfas, and R. Dillmann, "Imitation learning of dual-arm manipulation tasks in humanoid robots," *International Journal of Humanoid Robotics*, vol. 5, no. 02, pp. 183–202, 2008.

[15] L. Rozo, P. Jiménez, and C. Torras, "A robot learning from demonstration framework to perform force-based manipulation tasks," *Intelligent service robotics*, vol. 6, no. 1, pp. 33–51, 2013.

[16] S. Calinon and A. Billard, "Learning of gestures by imitation in a humanoid robot," tech. rep., Cambridge University Press, 2007.

[17] M. N. Nicolescu and M. J. Mataric, "Natural methods for robot task learning: Instructive demonstrations, generalization and practice," in *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pp. 241–248, 2003.

[18] S. Ross and D. Bagnell, "Efficient reductions for imitation learning," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 661–668, JMLR Workshop and Conference Proceedings, 2010.

[19] J. Togelius, R. De Nardi, and S. M. Lucas, "Towards automatic personalised content creation for racing games," in *2007 IEEE Symposium on Computational Intelligence and Games*, pp. 252–259, IEEE, 2007.

[20] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[21] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, "Survey: Robot programming by demonstration," tech. rep., Springrer, 2008.

[22] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[23] H. Kim, Y. Ohmura, and Y. Kuniyoshi, "Transformer-based deep imitation learning for dual-arm robot manipulation," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 8965–8972, IEEE, 2021.

[24] L. Theis and M. Bethge, "Generative image modeling using spatial lstms," *Advances in neural information processing systems*, vol. 28, 2015.

[25] A. Van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, *et al.*, "Conditional image generation with pixelcnn decoders," *Advances in neural information processing systems*, vol. 29, 2016.

[26] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma, "Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications," *arXiv preprint arXiv:1701.05517*, 2017.

[27] D. P. Kingma and M. Welling, "Stochastic gradient vb and the variational auto-encoder," in *Second International Conference on Learning Representations, ICLR*, vol. 19, p. 121, 2014.

[28] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[29] G. Papamakarios and I. Murray, "Distilling intractable generative models," in *Probabilistic Integration Workshop at Neural Information Processing Systems*, 2015.

[30] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," in *International conference on machine learning*, pp. 1278–1286, PMLR, 2014.

[31] D. Rezende and S. Mohamed, "Variational inference with normalizing flows," in *International conference on machine learning*, pp. 1530–1538, PMLR, 2015.

[32] B. Uria, M.-A. Côté, K. Gregor, I. Murray, and H. Larochelle, "Neural autoregressive distribution estimation," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 7184–7220, 2016.

[33] G. Papamakarios, T. Pavlakou, and I. Murray, "Masked autoregressive flow for density estimation," *Advances in neural information processing systems*, vol. 30, 2017.

[34] H. Wu, J. Köhler, and F. Noé, "Stochastic normalizing flows," *Advances in Neural Information Processing Systems*, vol. 33, pp. 5933–5944, 2020.

[35] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas, "Learning to learn by gradient descent by gradient descent," *Advances in neural information processing systems*, vol. 29, 2016.

[36] X.-W. Chen and X. Lin, "Big data deep learning: challenges and perspectives," *IEEE access*, vol. 2, pp. 514–525, 2014.

[37] F. Negahbani, R. Sabzi, B. Pakniyat Jahromi, D. Firouzabadi, F. Movahedi, M. Kohandel Shirazi, S. Majidi, and A. Dehghanian, "Pathonet introduced as a deep neural network backend for evaluation of ki-67 and tumor-infiltrating lymphocytes in breast cancer," *Scientific reports*, vol. 11, no. 1, pp. 1–13, 2021.

[38] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A survey on deep transfer learning," in *International conference on artificial neural networks*, pp. 270–279, Springer, 2018.

[39] P. P. Shinde and S. Shah, "A review of machine learning and deep learning applications," in *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, pp. 1–6, 2018.

[40] A. A. Esmael, H. H. da Silva, T. Ji, and R. da Silva Torres, "Non-technical loss detection in power grid using information retrieval approaches: A comparative study," *IEEE Access*, vol. 9, pp. 40635–40648, 2021.

[41] S. J. Pan, I. W. Tsang, J. T. Kwok, and Q. Yang, "Domain adaptation via transfer component analysis," *IEEE transactions on neural networks*, vol. 22, no. 2, pp. 199–210, 2010.

[42] J. Zhang, W. Li, and P. Ogunbona, "Joint geometrical and statistical alignment for visual domain adaptation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1859–1867, 2017.

[43] E. Tzeng, J. Hoffman, N. Zhang, K. Saenko, and T. Darrell, "Deep domain confusion: Maximizing for domain invariance," *arXiv preprint arXiv:1412.3474*, 2014.

[44] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell, "Adversarial discriminative domain adaptation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7167–7176, 2017.

[45] Y. Ganin and V. Lempitsky, "Unsupervised domain adaptation by backpropagation," in *International conference on machine learning*, pp. 1180–1189, PMLR, 2015.

[46] H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, and M. Marchand, "Domain-adversarial neural networks," *arXiv preprint arXiv:1412.4446*, 2014.

[47] J.-T. Huang, J. Li, D. Yu, L. Deng, and Y. Gong, "Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 7304–7308, IEEE, 2013.

[48] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1717–1724, 2014.

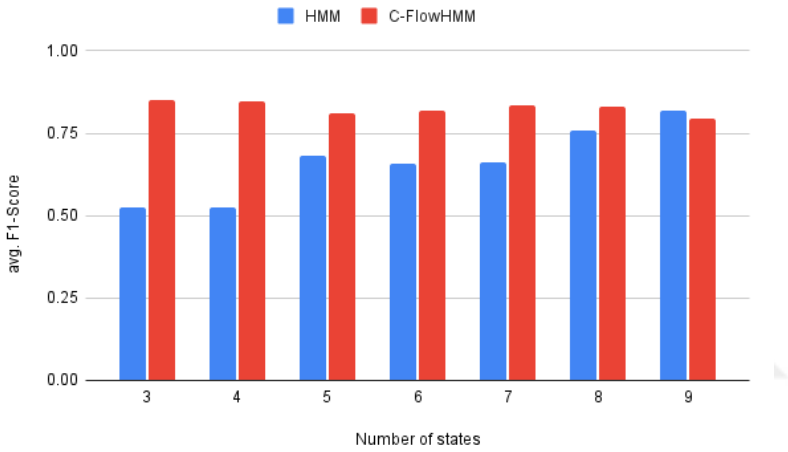
[49] M. Long, H. Zhu, J. Wang, and M. I. Jordan, "Unsupervised domain adaptation with residual transfer networks," *Advances in neural information processing systems*, vol. 29, 2016.

[50] B. Akgun, M. Cakmak, K. Jiang, and A. L. Thomaz, "Keyframe-based learning from demonstration," *International Journal of Social Robotics*, vol. 4, no. 4, pp. 343–355, 2012.

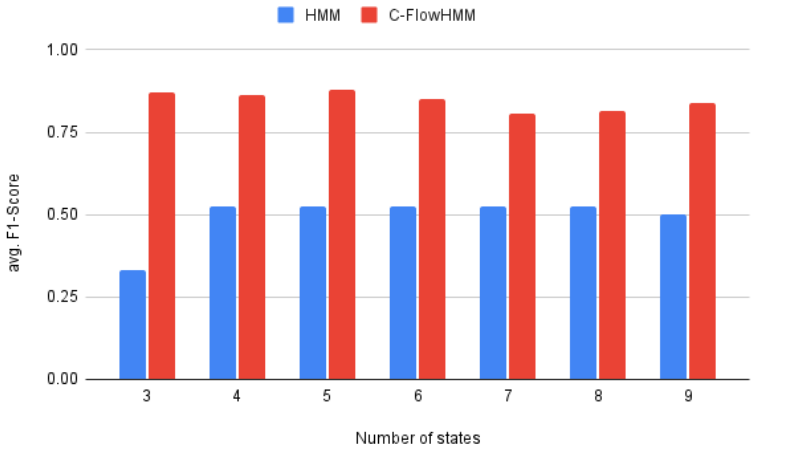
Appendix A

ADDITIONAL C-FLOWHMM RESULTS

In this appendix we have provided more skill monitoring results of C-FlowHMM.



(a) Close Skill Monitoring



(b) Open Skill Monitoring

Figure A.1: **Sqr1 Object Open and Close Skill Monitoring.**

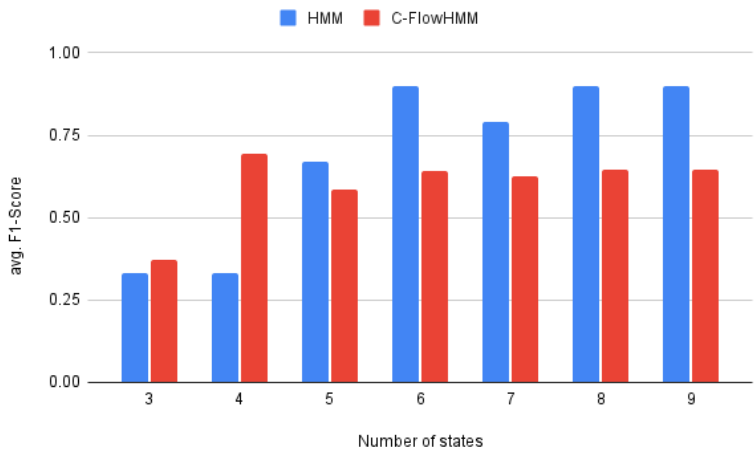


Figure A.2: **Tall1 Object Close Skill Monitoring.**

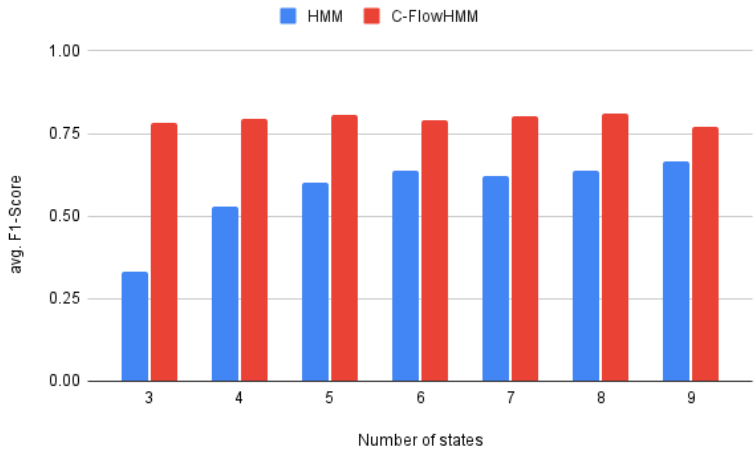
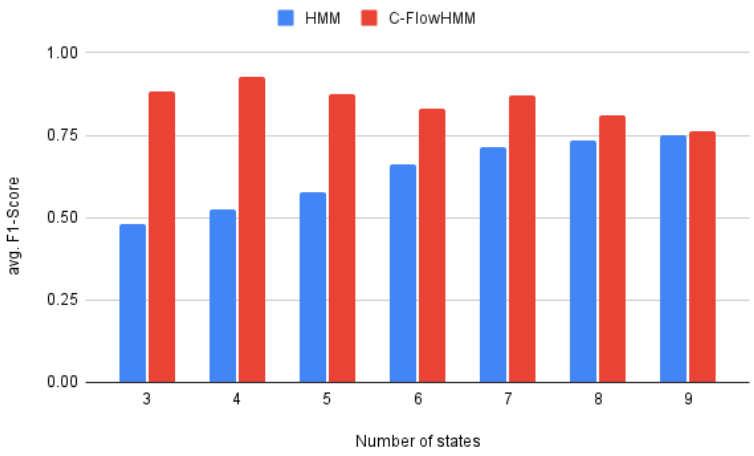
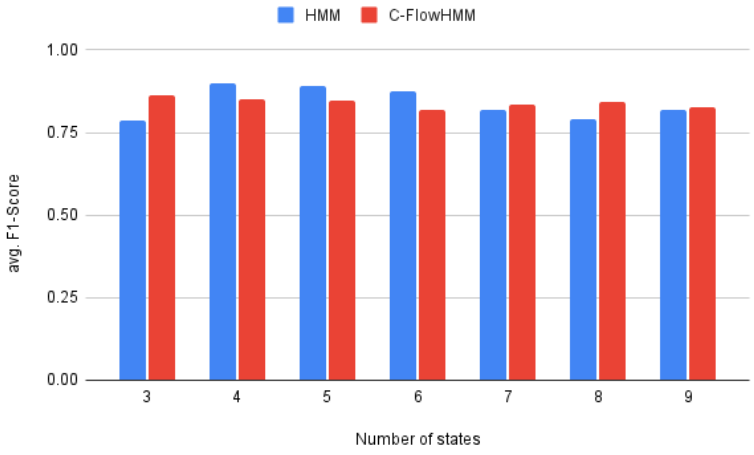


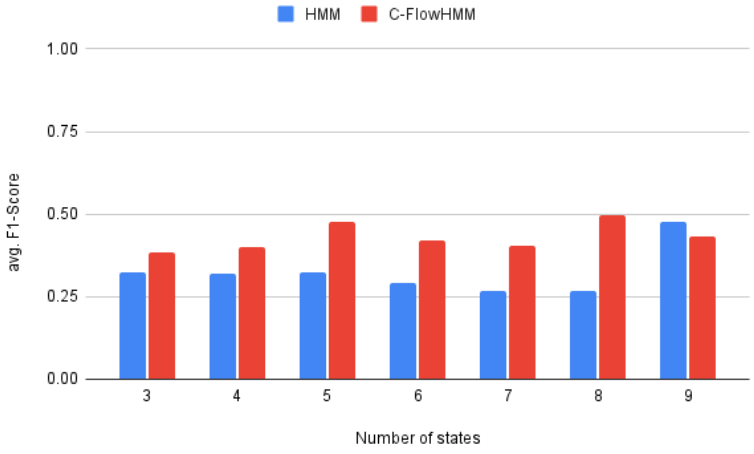
Figure A.3: **Bowl Object Pour Skill Monitoring.**



(a) Close Skill Monitoring



(b) Open Skill Monitoring



(c) Pour Skill Monitoring

Figure A.4: **Oct1 Object Open, Close, and Pour Skill Monitoring.**