LEARNING PERCEPTION TO ACTION MAPPING

FOR FUNCTIONAL IMITATION

by

BHUPENDER SINGH

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

NOVEMBER 2016

Acknowledgements

Firstly, I would like to express my sincere gratitude to my supervising professor, Dr. Manfred Huber, who has been a great motivating factor and a constant source of encouragement throughout my masters' research. Without his guidance and excellent foresight this thesis would have only remained a great idea. I could not have imagined having a better advisor and mentor for my master's thesis. I am sincerely thankful to Dr. Gergely V. Zaruba and Dr. Farhad Kamangar for giving valuable suggestions and serving on my committee.

I thank to all my lab mates, especially Sourabh Bose for the stimulating discussions, for the sleepless nights we were working together before project meetings and other deadlines, and for all the fun we have had in the last few years. I also greatly appreciate Sandeep, Akash, Arun, Azmat, Dr. Vamsikrishna and few other friends for their love and support.

Last but not the least, I would like to thank my family: my mother (Suman), my niece (Kanika) and to my sister (Lalita) and brother-in-law (Arvind) for providing me with heartfelt love, unfailing support, and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

November 21, 2016

Abstract

LEARNING PERCEPTION TO ACTION MAPPING

FOR FUNCTIONAL IMITATION

Bhupender Singh, MS

The University of Texas at Arlington, 2016

Supervising Professor: Manfred Huber

Imitation leaning is the learning of advanced behavior whereby an agent acquires a skill by observing another's behavior while performing the same skill. The main objective of imitation learning is to make robots usable for a variety of tasks without programming them but by simply demonstrating new tasks. The power of this approach arises since end users of such robots will frequently not know how to program the robot, might not understand the dynamics and behavioral capabilities of the system, and might not know how to program these robots to get different/new tasks done. Some challenges in achieving imitation capabilities exist, include the difference in state space where the robot observes demonstrations of task in terms of different features compared to the ones describing the space in which it acts. The proposed approach to imitation learning in this thesis allows a robot to learn new tasks just by observing someone doing that task. For achieving this, the robot system uses two models. The first is an Internal model which represents all behavioral capabilities of the robot and consists of all possible states, actions, and the effects of executing the actions. The second is a demonstration model which represents

the perception of the task demonstration and is a continuous time, discrete event model consisting of a stream of state behavior sequences. Examples of perceived behavior can include a rolling behavior or a falling behavior of objects, etc. The approach proposed here then learns the similarity between states of the internal model and the states of the demonstrated model using a neural network function approximator and reinforcement learning with a reward feedback signal provided by the demonstrator. Using this similarity function, a heuristic search algorithm is used to find the action sequence that leads to the state and action sequence that is most similar to the observed task demonstrations. In this way, a robot learns to map its internal states to the sequence of observed states, yielding a policy for performing the corresponding task.

Table of contents

Acknowledgements iii
Abstract iv
List of Illustrations viii
List of Tables x
List of Algorithms xi
Chapter 1 1
Introduction 1
Motivations 1
Challenges 2
Approach 4
How to read this thesis 6
Chapter 2 7
Related work 7
Learning from demonstration(LfD)/ programming by demonstration(PbD) 9
Goal-Based / Functional Imitation 12
Proposed approach of thesis 14
Chapter 3 17
Proposed Imitation Learning Approach 17
Overview 17
Proposed approach 19
Chapter 4 22
Data Generation and Modeling 22
Internal Model 22
Demonstration Model 24
Chapter 5 28
Learning Cost metric 28
Mapping process 28
Learning Distance Function Approximator 33
Learning Action Behavior Mapping 37
Overview 38
Behavior 39
Generic Policy 41

Chapter 7 44
Experimental Evaluation 44
State Space 44
Action Space 45
Reward Function 45
Results 45
Multi-Task Experiments 47
Behavior to Action Mapping Experiments 48
Chapter 8 50
Conclusions and Future Work 50
Conclusion 51
Future Work 51
References 53
Bibliographical Information 56

List of Illustrations

Figure 3.1 Demonstration Task Scenario……..19

Figure 3.2 Proposed Solution Approach……20

Figure 4.1 Explaining Environment……23

Figure 4.2 Observed Data……25

Figure 4.3 Observed State Transitions……..26

Figure 5.1 Mapping Process……29

Figure 5.2 Problem Space Traversal……..30

Figure 5.3 Exploitation and Exploration Policies……..33

Figure 5.4 Learning Optimized Distance Function Using Function Approximator (NN)….34

Figure 6.1 Action Behavior Mapping……..38

Figure 6.2 Bouncing and Falling Behavior Example……39

Figure 6.3 Start and End States of a Behavior……40

Figure 6.4 Generic Policy Learning……41

Figure 7.1 Results of Single Demonstration for A Cleaning Task……46

Figure 7.2 Intermediate Results for Imitating a Task in Different Environment……..47

Figure 7.3 Intermediate Results of Learning Observed Behavior Mapping

to Robot's Internal Actions for Functional Imitation……..48

List of Tables

Table 4.1 Extracted States Using Time Slicing……27

List of Algorithms

Algorithm 2.1 Robot Learning Distance Function as Well as An Optimal Policy……15

Algorithm 6.1 SARSA Algorithm……..43

Chapter 1

Introduction

Motivations

Imitation is an advanced capability of an agent to replicate or learn behavior from the observation of the performance of a skill by another agent. Imitation bootstraps for an observer the process of learning behaviors and acquiring knowledge from a skilled individual. Over recent years, researchers have been getting more and more interest in applying imitation learning in the Artificial Intelligence (AI) and robotics fields.

The basic need for imitation learning for robots is to enable them to perform new tasks by just observing demonstrations from an individual or another robot system in situations where explicit programming of the system by a trained, knowledgeable expert is not easily possible. This arises particularly in fields such as service robotics where systems mainly interact with end users who may or may not know programming or understand the capabilities and physics of the robotic platform. In these situations, these robots should be able to acquire new tasks given task demonstrations without the need to change the programming logic.

By using imitation learning, the potential for the application of robots will dramatically increase in various fields. Generally, as robotic systems move into more complex and realistic environments, the need for learning capabilities increases as the potential for complete pre-programming of all situations becomes increasingly expensive and hard to achieve. With learning capabilities, the robot can more easily deal with a dynamic

environment that is tedious or even impossible to completely capture by a programmer.

Several significant advantages attached to imitation are:

Easy learning: The imitator (robot) can learn a new task by simply observing how a demonstrator (human or artificial agent) executes the task.

Fast learning: Imitation speeds up the learning process by transferring knowledge from the demonstrator to the imitator as the demonstration represents a possible solution of the task.

Easy training: The demonstrator can be an expert in a task domain without the additional need for an extensive technical background in robotics or programming.

Low training overhead: The demonstrator can continue its own work without consideration of and/or the need to pay significant attention to the imitator.

While the benefits of imitation learning are significant, we face many challenges, including when to imitate, what to imitate, how to imitate, and how to evaluate a successful imitation in particular in situations where the embodiment of agent and demonstrator are different.

Challenges

Learning by imitation in real-world domains poses a number of challenges that have to be overcome by the imitator to be able to be successful. Many of these arise as these agents will generally continuously observe their environment and act within it. It is thus essential for them to be able to determine when an observation is actually useful to imitate, what of all the things it observes actually forms a demonstration and how to achieve the same task.

When to imitate? A robot should not only work on command but it should also learn whether it is appropriate to imitate observed behavior in the present social scenario. It should also learn whether it is having good models covering enough information for doing imitation, and purpose to imitate. In case the robot determines that it does not have a good observation model, it might be beneficial if it could indicate this and/or motivate the demonstrator to perform the task again.

What to imitate? The observed demonstration model of any task may have extra detail that is not essential for the task (such as, for example, the exact foot placement locations or the places fingers get put on an object). It is not necessary to imitate all these minute details of observation. The robot should be able to determine and extract the aspects of the demonstration that are important for performing the corresponding skill.

How to imitate? As soon as a robot understands what to imitate, it will try to perform the task. In order to do this correctly, however, it has to determine the correct way to do this from all the possible approximate replications of the observations. To do this it can, over the course of multiple imitations, attempt to determine which imitation strategy maximizes a task reward that might be provided either by the environment or by the demonstrator or robot owner. In this form, the system can infer what to imitate and improve its solution. This is related to the ‘’Correspondence Problem’’ [15], which requires to establish the correct relation between observed behaviors and internal actions, or states of the model in its environment to states of one's own body and environment, or both.

How to evaluate a successful imitation? Once a robot is provided with a demonstration of some skill, it will try to imitate that. To compare its actions with respect to those of the demonstration model, the robot must be able to identify the desired outcome and to judge how successful its policy was in the achievement of that outcome. One way to evaluate this is through a separate reward signal received at end of policy execution that is directly proportional to the desirability of the outcome. In the presence of such a signal, the robot must be able to diagnose its own errors to incrementally improve performance using reinforcement learning.

Are embodiment of agent and demonstrator different? (Physical equivalence) When the demonstrator and imitator have similar bodies and capabilities, internal states usually closely match observed states. But when the imitator is substantially different from the demonstrator, the imitator may not be able to easily establish this mapping and the imitation may fail. In these situations, the challenge of learning how to imitate becomes more difficult and state/behavior mapping becomes a complex problem.

Approach

In this thesis, the imitation learning framework we have developed uses a task demonstration model and its own internal model in order to imitate task(s). The demonstration is a sequence of observed states, representing the state of the demonstrator and the environment. Transitions happen whenever a significant change has been detected. The internal model is a Markov decision process model containing all possible

states and actions encountered by and available to the robot. The internal states represent the state of the imitator and the environment, which can be achieved using its own action capabilities. The imitation learning problem is then addressed through two solution approaches.

In the first approach, the robot learns to map similar states from its internal model to states of the demonstrated model. Similarity between states of these two models is defined by an Artificial Neural Network (ANN) that learns a distance function based on the states' feature values. Treating the distance returned by the trained ANN as a cost metric, an optimal and efficient path finding algorithm is used to generate a cost-effective policy for imitating the given task(s). The learned policy consists of a sequence of internal states and actions which approximates the demonstration even if the imitator has a different body and different capabilities from the demonstrator

In the second approach, the robot learns individual policies from its internal model directly for each of the observed behaviors. Each policy is learned using a reinforcement learning technique called SARSA [21]. Here, the first step is to learn an action-value function. For an on-policy method we must estimate $Q^{\pi}(s, a)$ for the current behavior policy π and for all states s and actions a. Then we consider transitions from state-action pair to state-action pair, and learn the value of state-action pairs. The goal of learning these direct observed behaviors to policy mappings is to further improve the ability to obtain valid imitation strategies from a minimal set of observations through the accumulation of knowledge about the physics of the environment and the preferences of the demonstrator.

How to read this thesis

The rest of this thesis is organized as follows:

Chapter 2 reviews the related work involving the use of functional imitation learning by observing a task performed by some other person and provides a review of existing methods.

Chapter 3 introduces the technical background of the proposed method of functional imitation and discusses its notable features.

Chapter 4 introduces the methods for the construction of our internal and demonstration model. It provides a description of the parameters that are extractable and the possible interpretations. The background and the underlying characteristics of imitation learning,

Chapter 5 describes how an imitation state action sequence is generated using a heuristic search algorithm and how subsequently an optimized distance function is learned with the help of feedback from the demonstrator.

Chapter 6 provides the technical background of the proposed mapping of internal actions to observed behaviors and discusses the main benefits and features of this mapping.

Chapter 7 introduces the experimental setup used for the demonstration of the methods and provides an analysis of the results.

Finally, Chapter 8 concludes the work and discusses possible future work and expansions of this project.

Chapter 2

Related work

Robotics research has made significant progress in the past decades. One area where this has become apartment is the area of assistive and service robots which have entered the commercial market and are inching closer to their introduction into the end-user market in larger numbers. These assistive and service robots can be used in space exploration, recreational household activities, healthcare, military services etc. Therefore, a robust framework of these control algorithms is required. Over the years, several approaches have been proposed for imitation leaning in these robots.

The work presented in this thesis can most directly be related to two approaches in the field of robotics. The first one is the area of learning from demonstration, a technique that develops policies from example state to action mappings [2]. Second is the field of functional imitation where this approach focuses on important aspects of demonstrated task. This is a powerful framework which is not dependent on embodiments of teacher or imitator [3].

Imitation is the replication of observed behavior and can be of two types, namely action/movement copying and goal based or functional imitation. Action-level imitation is the detailed and sequential specification and copying of individual actions whereas imitation at the goal or function level is a high-level mechanism used for complex tasks and neglects copying of each state detail in favor of only achieving the same outcome as the demonstration. Therefore action/movement level replication of observation has several

limitations. In this technique, few actions may be not important for imitation and thus this usually requires the imitator to have an embodiment that allows it to replicate all aspects and details of the observed behavior. If embodiments differ, then the imitation might or might not succeed. Therefore, it is frequently not an applicable way to address the imitation learning problem in situations where service robots have dramatically different forms and individual control capabilities compared to humans. In particular, there are certain problems associated with action-level imitation that make it difficult for real environments. These problems include: Actions taken by the individual/demonstrator may have varying importance to the achievement of the task. For example, throwing a ball to a goal state and walking to a goal state with the ball and depositing it there might lead to the same outcome and might thus have the same reward in many tasks. However, when the object to be transferred is a glass the outcome of the two actions differs and thus the importance of imitating the exact action becomes more important

On the other hand, goal based or functional imitation focuses on the ability to infer the important aspects of a human teacher's demonstration and only attempting to achieve the same (or similar) functional outcomes. The proposed research focuses on this version of this problem. In this approach, different embodiment of the demonstrator and imitator does not matter. The imitator tries to learn the correspondence between observed and internal states to produce a policy even if its structure, shape, and capabilities are different from those of the demonstrator. For example, if we have a human as a demonstrator that can pick up an object and move it to a specific location by walking, and a robot (without a gripper) as an imitator, then action-level imitation may fail due to correspondence problem but function-level imitation can lead to the robot performing the same task by pushing the

object to the target location. The following discusses the two approaches in more detail and introduces related work in these two areas.

Learning from demonstration(LfD)/ programming by demonstration(PbD)

In the last decades, robots have turned from simple preprogrammed machines into highly flexible and advanced systems. Along with this the programming of such a robot has become a tedious and highly time consuming task and some new solution approaches were required [2]. Learning from Demonstration (LfD) or Programming by Demonstration (PbD) is a technique for building automation into robots. The main principle of robot LfD–PbD is that new tasks are learned without programming but just with the help of demonstration. Within LfD, a task is learned from a demonstration or multiple demonstrations of the same task, provided by a teacher. Robot Programming by Demonstration (PbD) aims at allowing even end users who lack programming skills and familiarity with robotics to program new capabilities for a general-purpose robot by demonstrating the desired behavior. Given a set of demonstrations, the robot builds a model of the demonstrated task, which is used in reproducing the action in a new situation.

Here demonstrations can be of various types, such as teleoperation, shadowing, sensors mounted on the teacher, or external observation. Types of demonstration are categorized based on the form of teacher (the human or robot) and its available action capabilities.

Teleoperation: a human will operate the robot and task demonstration information is collected by the robot's own sensors.

Shadowing: the robot tries to replicate observed actions while collecting demonstration information using its sensors.

External observation: information collection sensors are independent of the demonstrator and may or may not be located on the robot learner [2].

In the presented approach, data is collected by the robot's own sensors in a global Cartesian space and the robot executes imitation in an egocentric space. A global Cartesian space is a coordinate system that specifies each point uniquely in the form of orthonormal, linear coordinates and are represented as the signed distances of the point's projection onto fixed perpendicular directed axes, measured in identical units of length. On the other hand, egocentric space, specifies each point relative the robot or human.

There are many ways in which a robot can be made to replicate the movement of a human. Animatronic devices (such as those used in amusement parks) continuously replay movements that have been recorded either by manually putting the machine into a sequence of postures or by using devices that record the joint angles of a human actor. Although these machines can perform very high fidelity playback, they are non-interactive; they neither respond to changes in their environment nor do they adapt to new situations.

Other research has focused on the development of robots that can learn to perform tasks by observing a person perform that action. This technique is often called 'learning from demonstration'. Early explorations did not focus on perceiving the movement of the human demonstrator, but rather focused on observing the effects of those movements on objects in the environment. In other work, the robot observes the human's performance as well,

using both object and human movement information to estimate a control policy for the desired task. Providing the robot with knowledge of the goal (in the form of an evaluation function) allows the robot to further improve its performance through trial and error, for instance, for a 'ball-in-cup' task [5]. Atkeson and Schaal [6] demonstrated that far fewer real-world practice trials were needed if the robot could simulate its experience using a predictive forward model for a pendulum-swing-up task.

More general solutions to the problem of perceiving human movement through vision have yet to be realized [7,8], but many researchers are turning to techniques such as hidden Markov models [9], to provide basic information on how a human is moving in a visual scene. These techniques combine task-based knowledge with predictive models in an attempt to link expectations of what the scene should look like with sensory data. Although these techniques can provide information on how a person is moving, subsequent extensive tuning to the robot and environment are often necessary to produce usable data.

Another important part of what to imitate is attention. The problems of perception are closely tied to models of attention. Some attention models selectively direct computational resources to areas containing task-related information. They do this either by using fixed criteria [10,11] (such as 'always look at red objects when trying to pick apples') or by using adaptive models that modify the attentional process based on the robot's social context and internal state. For example, the humanoid robot Cog was biased to attend to objects with colors that matched skin tones when it was 'lonely', and to attend to objects that were brightly colored when 'bored' [12]. Another strategy is to use imitative behavior as an implicit attentional mechanism that allows the imitator to share a similar perceptual state

with the demonstrator [14,13]. This approach is used in the 'learning-by-imitation' paradigm, in which the ability to imitate is given a priori and acts as a mechanism for reinforcing further learning and understanding.

Besides how to obtain observation data, there is another important question related to imitation learning, in particular how does a robot know how to imitate? Once a relevant action has been perceived, the robot must convert that perception into a sequence of its own motor responses to achieve the same result. Nehaniv and Dautenhahn have termed this the correspondence problem [15]. Although it is possible to specify the solution to the correspondence problem a priori, this is practical only in simple systems that use the learning-by-imitation paradigm described above. When the solution to the correspondence problem is acquired through experience, more complex perceptions and actions can be accommodated, and this is then referred to as 'learning to imitate'.

Goal-Based / Functional Imitation

In [4], most sophisticated forms of imitation are those that require an ability to infer the underlying goals and intentions of a teacher. In this case, the imitating agent attributes not only visible behaviors to others, but also utilizes the idea that others have internal mental states that underlie, predict, and generate these visible behaviors. For example, infants that are about 18 months old can readily imitate actions on objects, e.g., pulling apart a dumbbell shaped object. More interestingly, they can imitate this action even when the adult actor accidentally under- or overshoots his target, or the hands slipped several times,

leaving the goal-state unachieved [16]. They were thus presumably able to infer the actor's goal, which remained unfulfilled, and imitate not the observed action but the intended one.

The authors in [4] proposed a new model for intent inference and goal-based imitation based on probabilistic inference in graphical models. The model assumes an initial learning phase where the agent explores the environment, and learns a graphical model capturing the sensory consequences of motor actions. The learned model is then used for planning action sequences to goal states and for learning policies. The resulting graphical model then serves as a platform for intent inference and goal-based imitation. It extends the approach of [17] from planning in a traditional state-action Markov model to a full-fledged graphical model involving states, actions, and goals with edges for capturing conditional distributions denoting policies. The authors of the referenced paper used maze examples that were learned using a relatively small number of trials due to the small size of the state space. On the other hand, the proposed approach of this thesis has used quite large continuous state space example to illustrate the approach in the experiments.

In [18], the authors begin by showing how an agent can learn the mapping between goals, states, and actions in order to plan how to fixate on a specific goal location. The agent first learns a transition model, (e.g., through exploration or "body babbling") which translates an initial position and an action to a final head position. The author used computational model, which makes it possible to infer the goal of a head movement given observations of the starting and ending head poses, initial position, and final position, respectively. To accomplish this, the agent must be able to recover the inputs to each goal of a head movement, given the outputs. Results from [22] allow [18], research to estimate a

distribution over the inputs given the outputs. As such, inferring a distribution over actions given initial position and final position can be used to estimate a distribution over goals. This paper is closely related to functional imitation and inverse planning models and makes similar assumptions than the ones in this thesis in terms of state and action space, but the size of the state space is significantly smaller than in this thesis. Another difference is that to evaluate the inverse models, the imitation in it performs an exact match between the predicted state and the actual state. Since the imitator has no way to know which aspects of the demonstration are more important than others, it may sometimes fail to recognize a match resulting in an unsuccessful imitation strategy.

Proposed approach of thesis

In this thesis, a framework is proposed to allow different variety of robots to perform assistive and supportive tasks by observing demonstrations performed by a human. It is extension to [24], where internal model and observed model had same state representations. In proposed thesis, internal model and observed model have different state space which is comparatively complex and large in size as well. This thesis adds on top of past work by addressing, mapping of observed behavior to given actions of robot [Chapter 6]. The proposed approach to imitation learning in this thesis allows a robot to learn new tasks just by observing someone doing that task. Robot learns to map its internal states to the sequence of observed states, yielding a policy for performing the corresponding task. Mapping of states is based on distance between two states of these models. Similarity between two states is inversely proportional to distance in between states of internal model and observed model, and it is learned using neural network function approximator. Less distance means more similarity, and vice-versa. Heuristic

search algorithm: A* [20], is used by robot firstly, to find state-action sequence by examining smaller area of problem search space, called exploitation. Once this sequence is executed; a reward signal is provided to quantify the quality of imitation. Secondly, a larger area of problem search space is examined in virtue of getting to better solution which Provides better reward than what present state-action sequence already have. If exploration over policy space provides better reward, means new/explored state-action sequence is better match when compared to current state-action sequence for imitating the task. Therefore, new similarity metric is derived using resulted policy and this derived similarity metric is used to train neural network function approximator. Training of neural network takes place until reward reaches to its optimum value

```
Given demonstration(s) of task
REPEAT
    Generate a policy using current distance function and get reward (current_policy)
    Generate another policy using perturbed distance function and get reward(new_policy)
    IF(new_policy_reward > current_policy_reward )
        Pull current_dist_func towards new_dist_func
        Feed neural netwrok with new_dist_func
    ELSE
        Push current_dist_func away from new_dist_func
        Feed neural netwrok with new_dist_func
UNTILL an optimized distance function has been learned
```

Algorithm 2.1 Robot Learning Distance Function as Well as An Optimal Policy.

Given a demonstration, the robot uses a heuristic search algorithm [20], over problem space, and derives the policy that it best according to current distance function, executes the policy, and receives reward from the environment. Then it generates a new policy by

adding a normally distributed randomness into current distance function, and receives reward from the environment. If the new policy receives a better reward then, imitator derives a new distance function from the new policy and backpropagate it into the neural network. This completes a cycle of learning functional imitation in terms of distance function.

Training cycles are repeated until imitator receives maximum reward for the task. When an optimal policy is found, it can be assumed that the imitator has learned the correspondence between states of internal and observation model and function approximator has inferred important aspects of the demonstrations.

Chapter 3

Proposed Imitation Learning Approach

Overview

Imitation learning is a form of social learning which occurs through observing behaviors of others. It is commonly used by humans and animals. According to social cognitive learning theory [19], there are four stages involved in this type of learning. First, the observer must know its environment and identify its own capabilities in form of some model. Second, the observer should be able to correlate observed behavior in the past with similar or different situations. Third, the observer must be capable of imitating demonstrated behavior, and last, the observer should be able to incorporate/recognize the feedback/reward signal provided by the demonstrator. Feedback signals during learning can be provided by some external entity in the form of reinforcement. Artificial intelligence has always been motivated by biological and psychological systems. In similar fashion, social learning in natural systems has frequently guided us to apply imitation learning in artificial agents.

A robot must also know its environment and identify its own capabilities in the form of some model, referred to as the internal model which represents all behavioral capabilities of the robot and consists of all possible states, actions, and the effects of executing the actions. The result of an observation of a task is referred to as the demonstration model which represents the perception of the task demonstration and is here a continuous time, discrete event model consisting of a stream of state behavior sequences. During the journey of imitation learning, the robot should be able to map/correlate its internal model with the demonstrated model. The degree of correctness of the correlation can be learned by using

some external reinforcement signal. This feedback signal represents then to what extent an imitation task has been accomplished.

For example, assume that a robot should learn a task of gathering all trash items and putting them into a trash bin. In the demonstration model of this task, there is information related to the precise states of trash items, the trash bin and the demonstrator of the task. This model also has information about behaviors of objects during demonstration of task where the demonstrator can be another artificial agent or a human. In the internal model, in contrast, no location of the demonstrator is present and the locations of the items might be different. The task of imitation learning is then to learn to correlate the internal model information to demonstrator model parameters, realizing that only relative location of the trash items and the trash bin is important.

Another example could be, where a robot needs to learn grouping similar items at one place. Let's say there are 9 items of 3 categories, i.e. spherical (3), cubical (3) and cylindrical (3). In a given environment, all items are scattered on to floor and a human demonstrator will pick up one spherical item and drop into a corner of the room. Again, picking up another spherical item and dropping it into the same corner of room where the last spherical item was dropped. At last, all 3 kinds of item will be placed in three corners of the room. Therefore, the demonstration model comprises of all the states and behavior sequences of items and demonstrator. Figure 3.1 shows this scenario of problem.

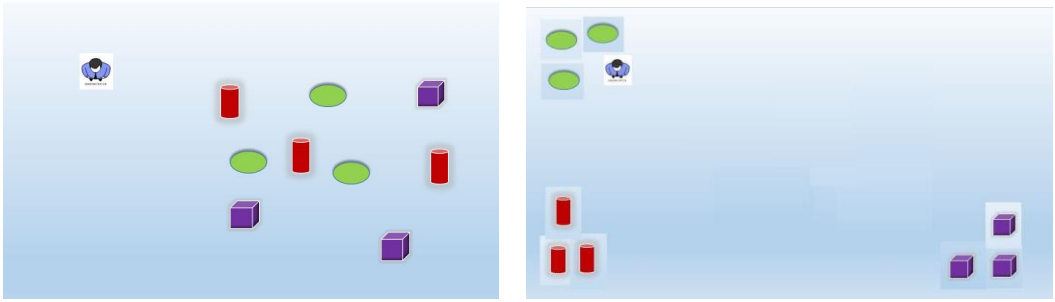

Figure 3.1 Demonstration Task Scenario

For imitating the task, robot will solve the state correspondence problem between these two models. In any of these task imitations, the object locations may not be the same as when the demonstrator is performing the task. For behaving/acting appropriately in novel situations across multiple demonstration, the robot must train itself over all acquired demonstrations and learn important features of this task. Here, for example, only the locations related to the corners (object destinations) are important while for objects only relative locations with respect to the robot and the corners are relevant.

Proposed approach

Proposed approach to imitation learning in this thesis allows a robot to learn new tasks just by observing someone doing that task. Robot learns to map its internal states to the sequence of observed states, yielding a policy for performing the corresponding task. Mapping of states is based on distance between two states of these models. Similarity between two states is directly proportional to distance in between states of internal model and observed model, and it is learned using neural network function approximator. Heuristic search algorithm: A* [20], is used by robot firstly, to find state-action sequence by examining smaller area of problem search space, called exploitation. Once this

sequence is executed; a reward signal is provided to quantify the quality of imitation. Secondly, a larger area of problem search space is examined in virtue of getting to better solution which Provides better reward than what present state-action sequence already have. If exploration over policy space provides better reward, means new/explored state-action sequence is better match when compared to current state-action sequence for imitating the task. Therefore, new similarity metric is derived using resulted policy and this derived similarity metric is used to train neural network function approximator. Training of neural network takes place until reward reaches to its optimum value. Figure 3.2 gives an idea of proposed solution approach.

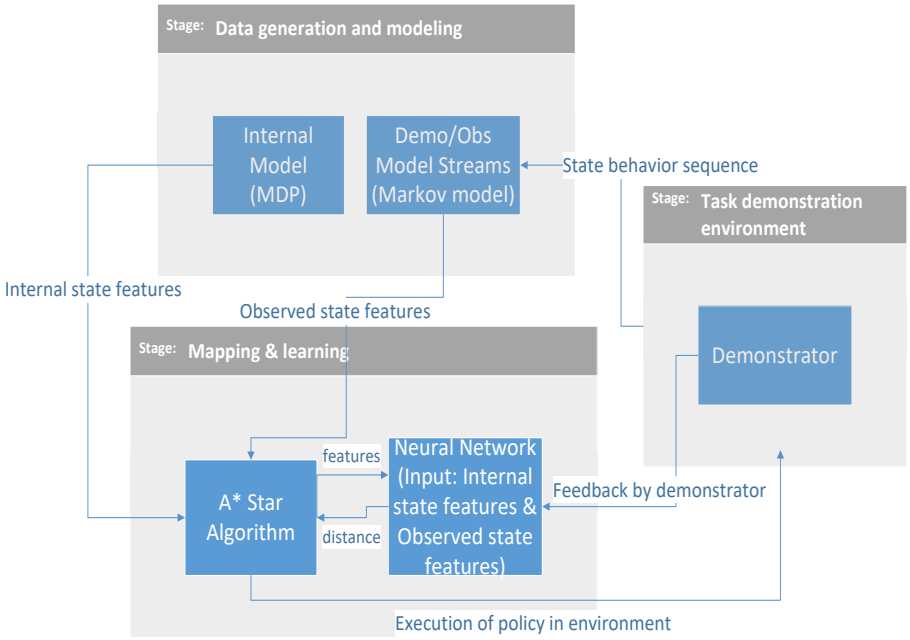

Figure 3.2 Proposed Solution Approach

The problem search space consists of a tree like structure, where each node of tree is a pair of internal and observed states. A node's actual cost is defined as the distance between paired states. An admissible heuristic for each node is considered as 0, therefore cost accumulation during search over policy space consists only of the actual cost provided by the neural network. To start this search process, the initial (i.e. current) state of the internal model is assumed to be associated with the initial state of demonstration model.

The robot can take a set of possible actions from its initial internal state, these actions will lead to a set of next states, which may be similar to the initial observed state or to the next observed state. Similarly, the current internal state may also be similar to the next observed state, without moving to any other internal state. While expanding to the next internal state, the neural network provides the actual cost of the resulting state pair. Once cost for every successor of the current internal state is calculated, they are added to the open-set of the search algorithm. The open list is a set where nodes are placed which are currently discovered and not evaluated yet. Now the node is selected from the open-set which has smallest accumulated distance for expanding towards its successors. Those successors, which are not currently in the open-set are added. If the successors which need to be expanded are already in the open-set, then their cost is compared and a least cost and path to reach this state pair is finally updated in the open-set. This process is repeated until all internal model states are mapped to corresponding demonstration model states. The state mapping provided by the above algorithm is referred to as the current policy, which need to be executed to obtain a reward signal.

Chapter 4

Data Generation and Modeling

The imitator uses two models for doing functional imitation. One of the models it uses is referred to as internal model and another as demonstration model.

Internal Model

To obtain an internal model, the imitator is learning objects' generic movements and behaviors by exploring and interacting with the environment. The imitator uses its primitive set of actions while learning about the environment. Example of a robot's initial skillset include a path planner, a kinesthetic controller, a force controller, a visual controller, a grasp controller etc. The environment where the imitator is observing demonstrations has various states. All these states are defined based on some attributes such as, for example, the location of objects in some coordinate frame.

In the experiments presented in this thesis, a simulator is designed to produce the internal model, which will consider environment information while generating the model. The environment in this case is defined in terms of an area which contains specific locations of interest where objects are located, where every location has a unique id. There are some objects included in the environment such as the demonstrator, trash-bin and, trash items. Objects are also provided with unique ID's and particular properties such as color and shape. Initially these objects are assigned to some of the locations. Figure 4.1 illustrates the described scenario.

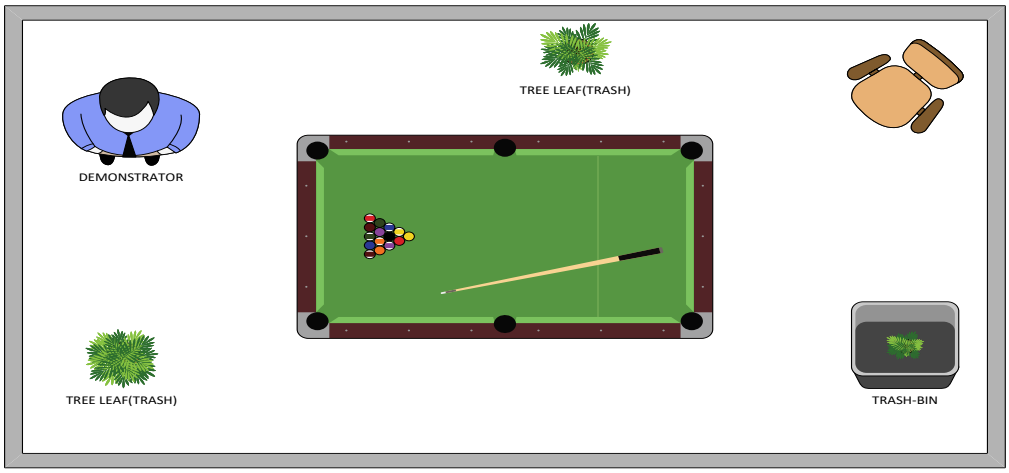

Figure 4.1: Explaining Environment

Once the imitator has knowledge about the environment, it can take any of the actions from its primitive set of actions. Actions include going to some location, grasping, and dropping and are only successful in some states. For example, the imitator can only pick-up/grab trash if it is nearby and the imitator can only drop trash into a trash-bin if the imitator is carrying trash and standing next to a trash-bin.

Every internal state in this model is described by some attributes like: number of probabilistic/deterministic action to take, which all next states it can go to, using allowed actions and feature values. A state's feature value provides information about locations of all objects, whether the imitator is carrying trash, etc. These state attributes play an important role in transferring information and providing a similarity measure for the correspondence problem between two models. The internal model is comprised of and represents all behavioral capabilities of the robot.

Demonstration Model

While the internal model of the robot can contain the internal state of the imitator, the demonstration model in this case can not contain internal variables of the demonstrator but only information that can be observed by the imitator. This effectively is the model where the robot has observed a demonstrator performing a skill. Humans and animals observe their environment using sight, sound, taste, smell, touch, pressure, and similar senses. Similarly, artificial systems do make use of various sensors (vision, sound sensor, tactile etc.) to obtain information about the environment. The proposed approach is aiming at using a Kinect one or similar as a vision sensor for observing demonstrations of task performed. The data stream coming from the Kinect and other sensors are pre-processed and the streams of pre-processed, object-centric data are referred to as fluents and needs to be processed to obtain useful information for model building from. The Kinect or any similar sensor outputs the stream of raw data when observing a demonstration of a task. Then a stream of fluents is extracted from that raw data. A fluent is defined as an abstracted property, such as the movement, of an object captured during observing all objects in the environment. One fluent may look like x_loc, y_loc, z_loc, their orientation in degrees, shape, color, and object id etc. Fluents play an important role in defining state features of all the objects. These state features will be used while performing the mapping between the robot's internal model and the observed model of the demonstration. Figure 4.2 describes this scenario. This process of observing a task is being captured in from of algorithms and equations, which generates a demonstration model. In proposed thesis experiments, computer simulation is used instead of real system.

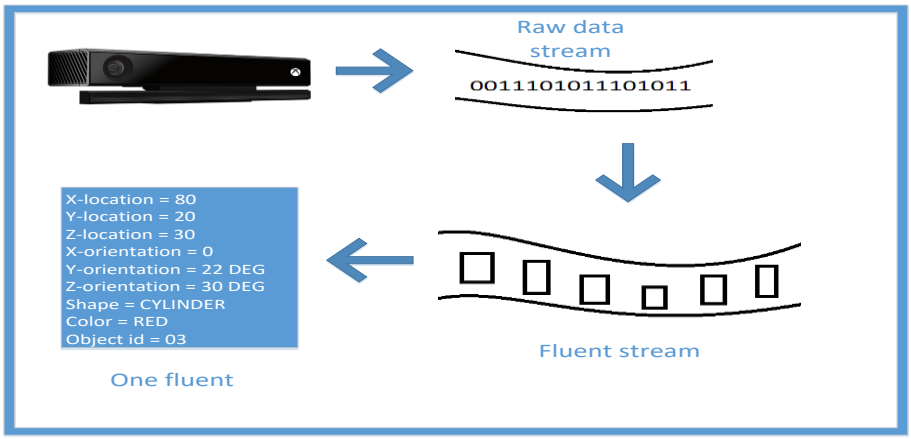

Figure 4.2 Observed Data

While generating demonstration model, information about environment and type of behaviors used in demonstration of task is required. In our experiments, the environment is the same as explained above using Figure 4.1. To further abstract the fluent streams produced here to obtain a more compact description of the observations, the fluents are further processed to transform the time-driven fluents which produce a state observation every camera frame, into an event-driven fluent representation where states transmitted are at points where discrete behavior changes of the corresponding object are observed.

A behavior is defined as a repeatable, reconstructable change in the time-driven fluent and occur between two states; for example: an object is stationary (behavior) in a particular location (state) and at some point, in time will move south with uniform velocity (behavior) for some time when it switches behavior again at some location (state). The event-driven fluent encodes these behaviors in the demonstration model. Demonstration of a task here provides some information about every object that is present. This information includes all

the states transited through by an object during that period. It also contains all behaviors which the same object has undergone during that period. States are described by using feature values like location, orientation, and time. Similarly, behaviors are described in terms of an ID associated with a time-driven model of the prototypical fluent sequence corresponding to this behavior. There is always a behavior between any two states. For example, if an object has gone through a sequence of 4 types of behavior during the demonstration of the task, this objects must have been through 5 states. To synchronize the states and behavior of all the objects, they are processed using dynamic time warping techniques [23]. This is shown in the Figure 4.3 and Table 4.1.

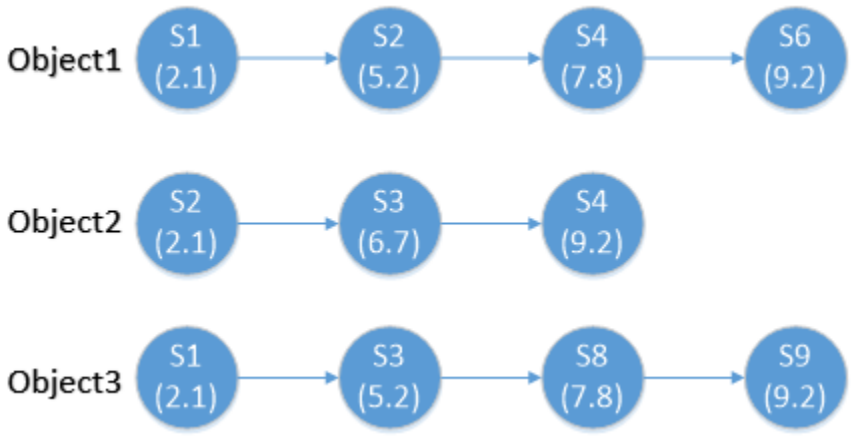

Figure 4.3 Observed State Transitions

The diagram in Figure 4.3 represents the fluents encoding the event-driven transitions made by different objects while observing a demonstration of a task by the demonstrator. Object 1 is in State 1 at time 2.1, then it made a transition to State 2 at time 5.2; finally,

Object 1 ended up in State 6 at the end of the demonstration at time 9.2. The fluents provide this information for all objects in the demonstration environment. To construct a single model of all objects from these fluents, all transition points have to be extracted. In this example, it can be seen that state transitions for at least one object occurred at times 2.1, 5.2, 6.7, 7.8, and 9.2.

Therefore, 5 states for the final observation model are extracted from this data. Table 4.1 shows how the 5 states are extracted. One thing to notice here is that, Object 1 has no state transition at time 6.7 so it is treated as the same state at time 5.2. Therefore state 3 for Object 1 is State 2 but its referred to as State 3 because Object 2 made a state transition from State 2 to State 3 at time 6.7.

Objects	**Time (2.1)**	**Time (5.2)**	**Time (6.7)**	**Time (7.8)**	**Time (9.2)**
Object1	S1	S2	S2	S4	S6
Object2	S2	S2	S3	S3	S4
Object3	S1	S2	S2	S8	S9

Table 4.1 Extracted States Using Time Slicing

Chapter 5

Learning Cost metric

To learn the correspondence between states of the two models, the imitator uses a heuristic search algorithm for traversing through problem search space. This algorithm derives tree traversal cost through a state similarity/distance function provided by a function approximator. We elaborate on the traversal of the search algorithm in the given problem space using Figure 5.2. Every node in this tree is a combination of internal state and corresponding observed state from two models. Every state in internal model has some fixed set of actions. Using these action, imitator can go to next/successor states. All the successor states of a node, inherit correspondence of current observed and next observed state.

Mapping process

The mapping process defines how states from two models are matched to each other. In all cases of a successful mapping, the first state of the internal model must be mapped to the first state of the observed state behavior sequence and the last state of the state action sequence must match to the last state of the observed state behavior sequence. Intermediate states have the flexibility to match to a corresponding state of the other model with the limitation that time can not move backwards and no state can be skipped in the mapping sequence. As a result there can be one to many and many to one mappings, i.e. two or more than two states from the internal model can be mapped to the same state in

the observed model and similarly, two or more than two states of the observed model can be mapped to the same state of the internal model.

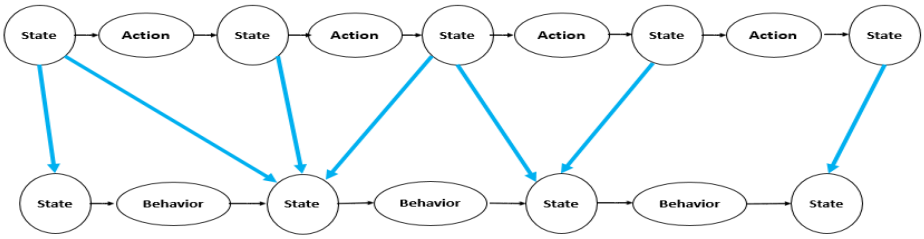

Figure 5.1 Mapping Process

Using the example shown in Figure 5.2, imitator has two actions in any internal state. In this case, the current node has five successors. Two successors correspond to the same observed state as the current internal state and two successors correspond to next internal state, with the last corresponding to the next internal and the next observed state (i.e. the situation where both sequences advanced one time step). There can be a condition when there is no state transition from the internal model side but the current observed state changed to the next observed state.

Action cost and heuristic cost are treated as zero in the experiments performed here but could be replaced by any fixed cost function and admissible heuristic, respectively. Using the function in our experiments, the cost of transitions in states is fully determined by the distance between internal and its corresponding observed state.

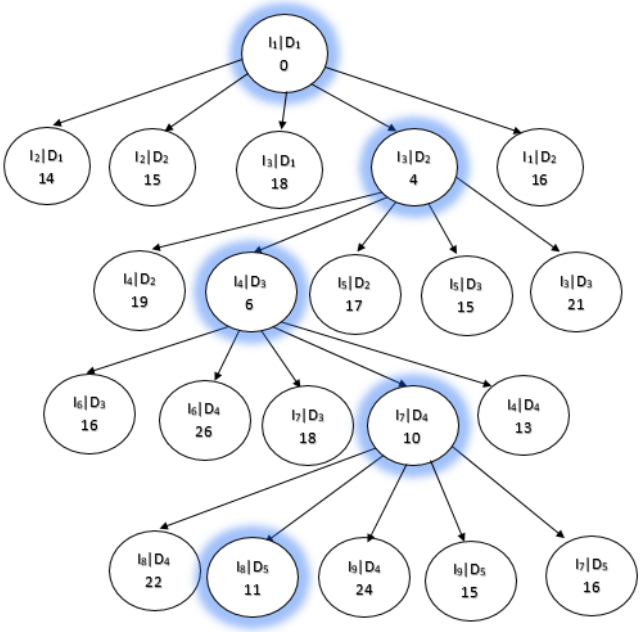

Figure 5.2: Problem Space Traversal

$$f(n) = g(n) + h(n) \tag{5.1}$$

$$g(n) = \sum_{i=1}^{n} \mathrm{D}\ (\mathrm{SIi} \mid \mathrm{SOj}) \tag{5.2}$$

where $g(n)$) is the cost of the path from the start node to node *n* and is given by the sum of distance function values. $h(n)$ is the heuristic cost that estimates the cost of the cheapest path from *n* to the goal (which in the experiments performed here is set to 0). SIi is the imitator's internal model state i, and O_j is the corresponding demonstration model state in the search tree. D $(\mathrm{SIi} \mid \mathrm{SOj})$ refers as distance between Internal state i, and observation state j. $g(n)$ represents the distance between the sequence of internal and observed states and at each step increases by the distance value for states i and j, respectively.

This distance is a treated as the inverse of a similarity metric between these two state. A lower distance represents a higher similarity and vice-versa. This function encapsulates which aspects of the state are relevant to the task and how they map between the feature representations of the two state spaces (internal and observed). As such it is usually not known beforehand and instead has to be learned. Initially the function approximator used to calculate this value provides some distance which is not accurate because it is randomly initialized. In later stages, the function approximator will learn to provide optimized distances using a reward signal.

During the search, as soon as all these successors are added into the open set (the set of nodes which need to be explored), the current node is moved to the closed set (the set of already explored nodes), and the lowest cost node is picked as the current node in the search. Now the successors of this new current node are added into the open set and again the lowest cost node is picked from the entire open set and the current node is moved to the closed set. This process will continue until a node is expanded that corresponds to an internal state and last sate of observation model.

Once this final state goal node is reached, the imitator fond the lowest cost state action sequence which maps to all observed states, the imitator executes this policy described by these actions in the environment. As soon as the selected policy is executed, it is assumed that the demonstrator of the task (or some other entity) provides a numeric reward to the imitator indicating the appropriateness/quality of the performed policy for the task to be imitated. A positive feedback signal is treated as reward and a negative as a penalty. The

policy found using the actual distance function is referred to as the current policy. The process of finding the current policy is referred to as *EXPLOITATION* since the imitator has used the current, learned distance function (and thus all previous knowledge) for finding the solution.

To be able to determine whether the current policy, and with it the current distance function, can be improved upon, the system also tests policies that are not optimal with respect to the current state of knowledge. For this, the problem space is again traversed to find a required sate action sequence which completes the demonstrated task. However, this time, every node's cost is calculated by adding a normally distributed random number to the current distance function, thus perturbing it. The perturbed distance function is thus a value drawn from a normal distribution with the mean equal to the current distance function and some small variance which allows the imitator to explore the problem space. Note that the distance function used in this case is slightly different from the current distance function, thus potentially resulting in a different best match in the search. Once the imitator has found a new solution (state action sequence) for performing the perceived task, it is again executed in the environment and a new reward is received from the demonstrator. The newly found policy and reward are referred to as new policy and new reward. The process used for finding the new policy is called *EXPLORATRION* as the imitator has tried to search the problem space in the vicinity of the current knowledge.

Figure 5.3 represents these two policies found by the imitator for performing a demonstrated task.

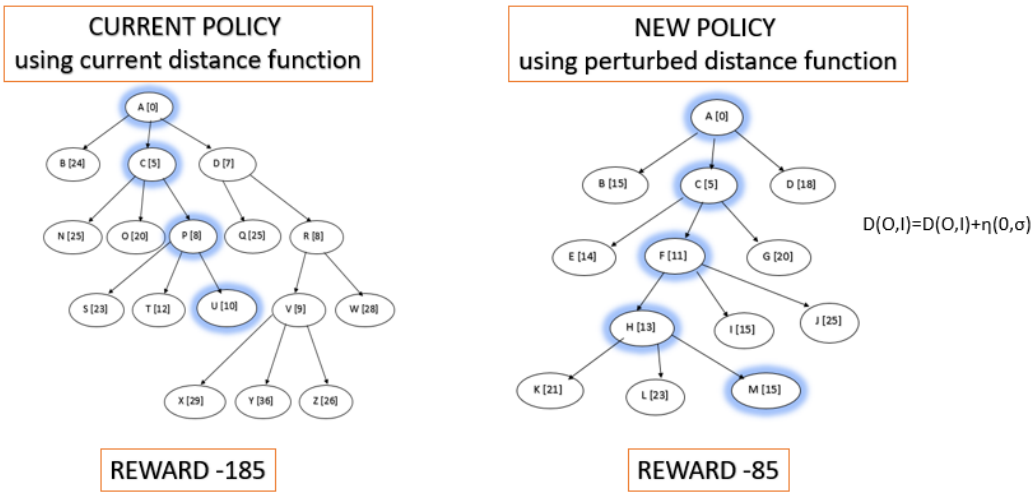

Figure 5.3 Exploitation and Exploration Policies

Learning Distance Function Approximator

As indicated, the result of the imitator utilized its current knowledge to get a policy using A* is said to be current policy. If the imitator, exploring and searching using the same distance function perturbed with uniform noise, found a higher or lower reward receiving policy, this new policy guides a learning process to derive a distance function that yields a better result. For this, the distance function has to be modified such as to increase the likelihood of obtaining the search tree of the better policy and to reduce the likelihood for the search tree of the worse policy.

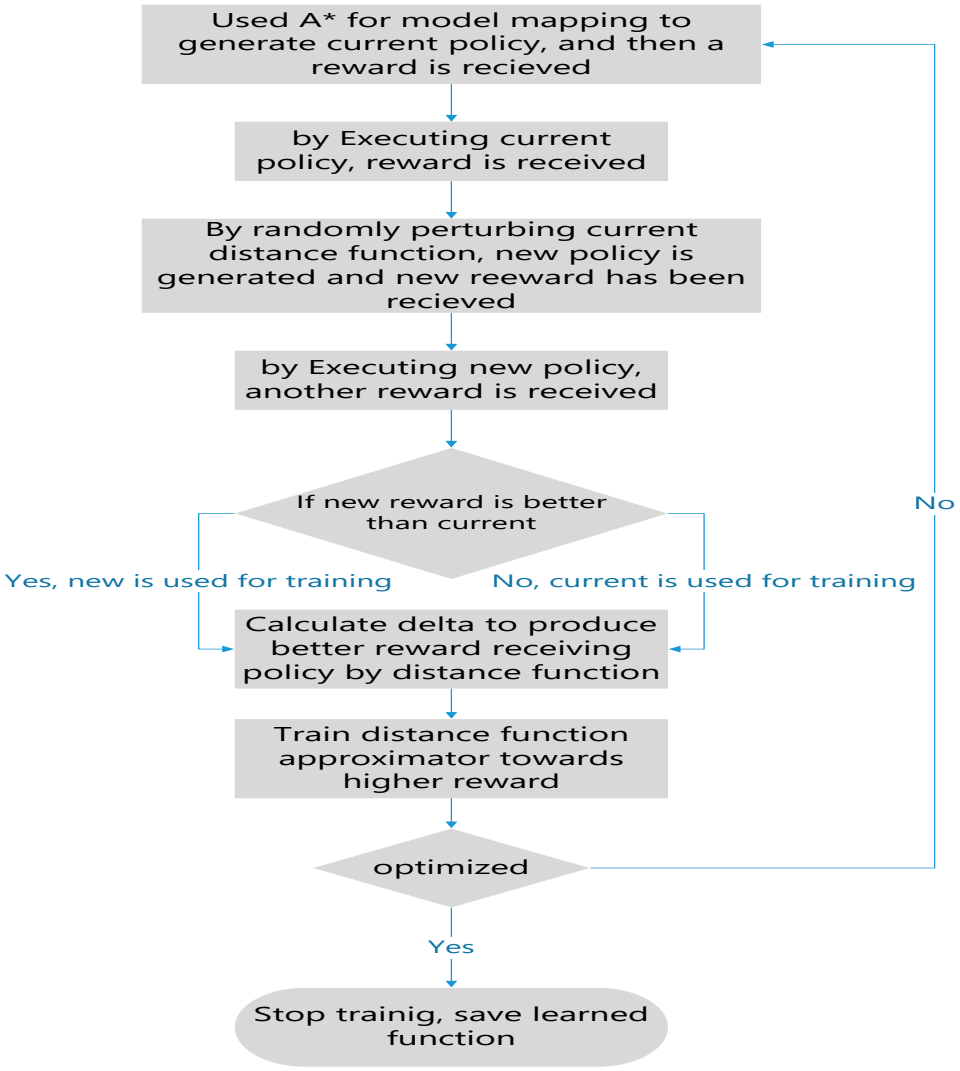

Figure 5.4 Learning Optimized Distance Function Using Function Approximator (NN)

For learning this new distance function to produce a higher reward policies in case the explored policy has a higher reward, the imitator increases the cost of those nodes in the current A* tree which have maximum cost in their corresponding branches such that they have a higher value than the maximum cost of the new policy. It also increases the cost of

the new policy's siblings cost and sets them to be higher than the policy node itself and all the successors of that policy node. This way the imitator would ensure that before picking any siblings of the new policy, the imitator must pick the nodes of the new policy. Lastly, the imitator decreases the cost of the new policy node such as to set it to be higher than the maximum cost of the current policy and the new policy's siblings and its predecessors' siblings.

In general, these above changes make sure of not picking the policy which has received lower reward but a policy which has received higher reward. However, since rewards can be noisy, implying that the same task execution might result in different rewards at different times, it might be misleading to adjust the distance function all the way to these values and instead only part of the updates required to achieve these changes is performed at each step. To also address the amount of improvement and deal with the case where the explored policy is worse, the computed changes are multiplied by the difference of the explored and the current policy rewards, leading effectively pulling the distance function towards the better of the two policies and pushing it away from generating the lower reward one. In particular, the distance function update is calculated as

$$\Delta_i = \frac{\sum_{\kappa i}(\Delta_i + \varepsilon) * (R_k - \widetilde{R_k})}{\max(\left|\max R_k - \min \widetilde{R_k}\right|, |max\widetilde{R_k} - \min R_k|)}$$

In this equation, Δ_{ki} is the change required for node i of A* tree in the demonstrated task k to achieve the abovementioned criteria where ε is the margin by which the value should be raised above the other tree's value. $\Delta_i\ is\ the$ amount of change to suggested node that should ultimately be applied. R_k is the reward of the explored policy and $\tilde{R}_k$ is the expected

reward of the current policy. This whole amount is divided by a normalizing factor made by comparing the reward ranges from these two policies.

This provides the gradient of the distance function used in the search tree which is used while training the neural network. This delta calculation also tells the function how important this update is by its magnitude. For multiple task or multiple demonstrations of the same task, a batch update method is used for training as indicated in this equation. In this way, training the distance function in the form of a function approximator towards a better reward receiving policy will make the imitator achieving increasingly better imitation of the demonstrated tasks.

Chapter 6

Learning Action Behavior Mapping

Sometimes it is not only important, what the goal of a task is, for which the start and end states are most important. In some cases, it is also important to know how a task should be done. In this case, intermediate sates become more important. Given that the imitator has observed a behavior in the middle of some task demonstration, to imitate that behavior, it is now important to know what is the starting state of the behavior and where it has ended. For example, to diffuse a bomb, it is important to achieve a successful diffusion of the bomb. But to enable this, it is important to cut/disconnect wires in a specific order. Action to behavior mapping helps to manage all these situations and make imitation faster in collaboration with the learned distance function as it provides additional information to guide the search process towards the most effective action choices.

Once the distance function has been learned, an optimal state mappings of the internal model to the observed model across various demonstrations of a task, or across various tasks itself, can be used to derive an action to behavior mapping by providing training examples to infer this mapping.

Figure 6.1 shows a block diagram for this strategy.

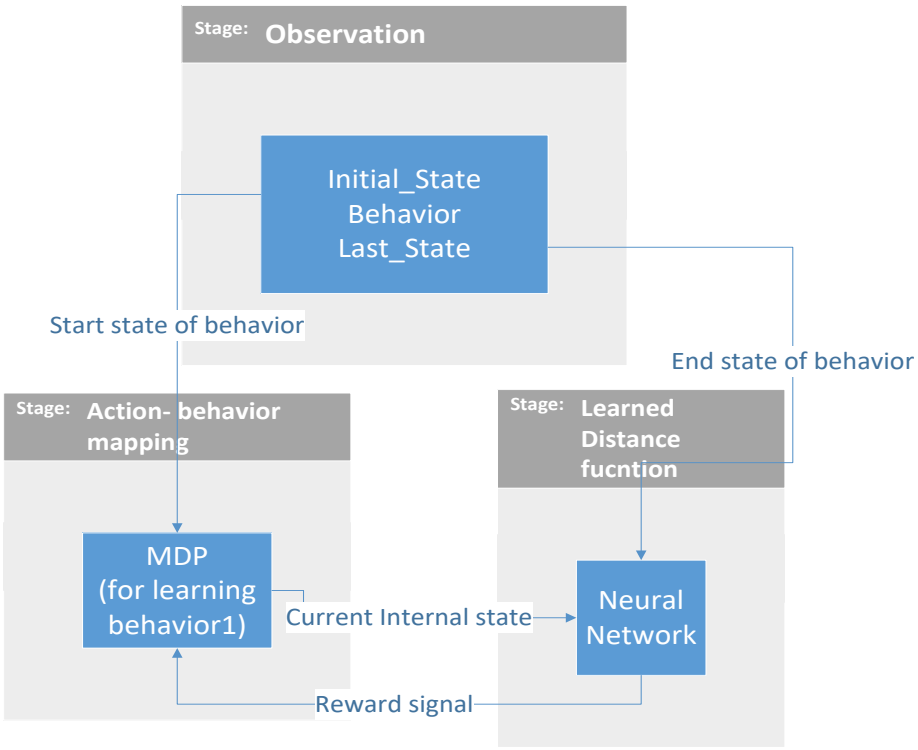

Figure 6.1 Action Behavior Mapping

Overview

This process has three main components. First, the observation model which consists of a state behavior sequence needs to be mapped onto given internal model. This provides starting and end states of the contained behaviors. Secondly there is an internal model, which is defined in the form of a Markov decision process. The MDP contains the state space where the imitator explores and finds an optimal policy for the assigned task. The MDP also has a set of given actions which an imitator can take to achieve state transitions while exploring the state space and a special STOP action. Once the imitator considers the current internal state as goal state, it executes the STOP action and reports its goal state to a reward function. The reward function also takes the last state of the

behavior as input and provides a feedback to the imitator. Using this, the imitator learns a policy to represent a behavior that optimizes the reward. In the following we will discuss this process in details by detailing all components of it.

Behavior

As discussed earlier, fluents capture all the movements of objects during the demonstration of task. Common subparts of these movements across all objects define a behavior. Examples of behaviors include rolling of an object, staying stationary at a location, bouncing, and moving to another location, etc...

Figure 6.2 shows two example behaviors of objects.

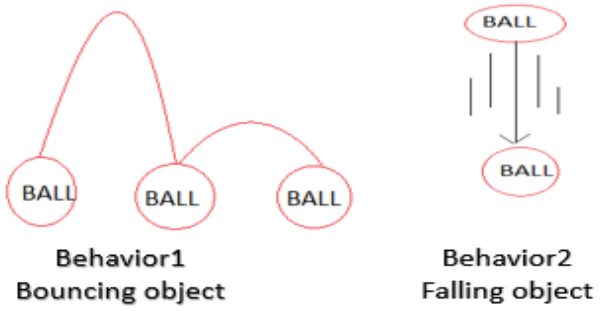

Figure 6.2 Bouncing and Falling Behavior Example

The fluent stream captured Behavior 1 as an object that is changing its x, y, and z locations over a period of time. This is identified as a bouncing behavior. Another example shown in Behavior 2 is represented as a z location change of the object representing a falling behavior. The same behaviors can be seen across different demonstrations of the same task or of different tasks as well. Figure 6.3 shows an example of falling behavior across three different tasks. The imitator can learn a policy to create this behavior using its internal MDP. The behavior is being observed between any two states and therefore it always has

a start and an end state. Figure 6.3 takes data from 3 tasks to show a single falling behavior. By looking onto start and end states, change is detected only in the z coordinate of object.

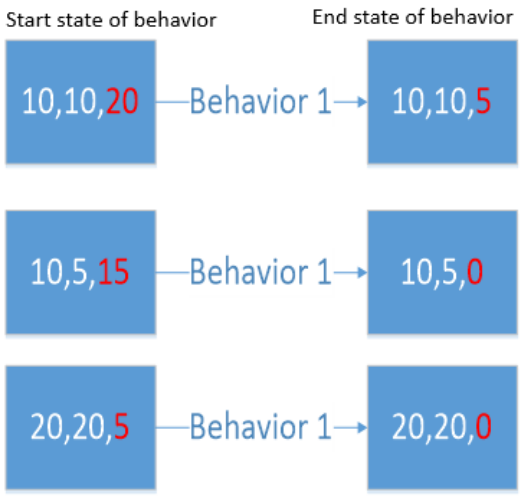

Figure 6.3 Start and End States of a Behavior

These start and end states will be used to map/learn this individual behavior by the imitator, as the imitator will report a goal state for mapping this behavior to the learned distance function. After achieving the goal state in the internal model of the robot, the learned distance function matches it to end states of behavior from the observed side and generates a reward. Using a reinforcement learning algorithm, the robot learns to achieve this behavior by optimizing its reward in terms of a policy over the MDP's state space. The policy learned by the robot has to be generic enough that it can identify a particular behavior being in its start state and then executes that behavior, irrespective of the environment (i.e. it has to match all observed demonstrations, thus promising to extend beyond this to new task environments). The following section discuss the generic policy in more detail.

Generic Policy

A policy (corresponding to a particular Markov chain) is a function that tells a robot to pick a fixed action (or action distribution) in a given state of the MDP. The optimal solution found in terms of a policy is the one which will try to maximize some cumulative function of the rewards. A specific policy is defined in one environment, whereas a generic policy is defined as a best general solution for the combination of more than one environment. Figure 6.4 shows the concept of a generic policy using individual and a combination of environments.

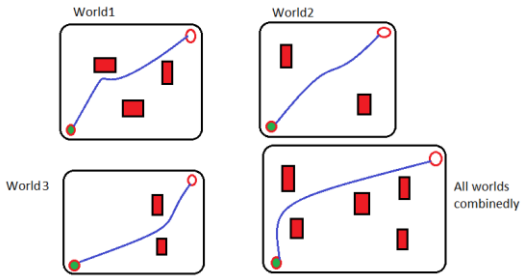

Figure 6.4 Generic Policy Learning

In this figure, red blocks represent obstacles, green circles represent start states, red circles represent goal states and blue lines represent policy paths from start to goal state in the given world/environment. This example considers 3 worlds where obstacles are at different locations and where there are individual policies learned by the robot to reach each goal state. When considering a combination of all these worlds in one environment, these optimal policies corresponding to each world may not be the optimal policy for the

environment where all these 3 worlds are combined. The robot must learn a new policy which is generic and optimal in that environment which is a mixture of all other worlds. This optimal generic policy learned may not be optimal in these given individual worlds. However, when dealing with imitation in new environments and trying to address mapping observed object's behavior to the given set of actions of the robot, the robot needs to learn a policy which is generic for all the environments in order to obtain successful imitation of that behavior in the new world.

As discussed earlier, the action behavior mapping stage of Figure 6.1, contains an MDP which will learn this policy for mapping to observed behavior. Here the imitator has to find a goal state (a state corresponding to the last state of observed behavior) in the MDP defined over the given demonstration environment. The MDP has some states and a given set of action for the robot to take and to achieve state transitions. Whenever the imitator finds a corresponding current internal state to the end state(s) of a behavior, the stop action is called where the imitator's policy treats the current internal state as a goal state. Then it receives a reward from the neural network. If the current internal state is the best match with the end state of the observed behavior, it returns the best reward for the imitator, otherwise it returns some reward which is not the optimal value of reward. Using these rewards, the imitator learns to represent the observed behavior as an optimal policy by updating its Q table values. A reinforcement learning algorithm, SARSA [21], is used for learning this policy from the MDP.

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(terminal\text{-}state, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Algorithm 6.1 SARSA Algorithm [21]

Initially, a Q table (MDP_states x Actions) is initialized with random values. Then the robot starts to interact with the environment by taking actions from the given set of actions, starting in the initial state (S0) and make state transitions to the next state (S1). Exploration and exploitation guides the process of choosing actions being in some state S. In the proposed approach, epsilon-greedy exploration is used for choosing actions. As soon as the imitator makes a state transition to the next state from the current state, a reward is provided by a reward function for choosing that chosen action. Again, an action for the next state is chosen using the same epsilon greedy algorithm. In the end, the current state is updated to the next state and the current action is updated to the next action. The Q value for a state-action is updated by an error, adjusted by the learning rate alpha. Q values represent the possible reward received in the next time step for taking action (A) in state (S), plus the discounted future reward received from the next state-action observation. These episodes will continue until a terminal state has been reached. This completes an episode of this policy learning algorithm called SARSA. Once robot has done enough exploration and exploitation over the entire problem space, it has learned a policy for mapping a behavior to its given set of actions in MDP space.

Chapter 7

Experimental Evaluation

This chapter presents the experimental setup which includes the environment description from Figure 4.1. There are two trash objects and a trash basket. A demonstrator who is demonstrating the skill has used 10 predefined locations.

State Space

The internal state space and robot capabilities are represented as an MDP model containing thirteen thousand states, where objects are placed in various locations. The demonstrator of the task can also be in any of these ten predefined locations. In the demonstrations, the demonstrator moves to a location where a trash object is placed and then it picks it up. After picking up the trash item, it moves towards the trash basket. Once the trash item is dropped into the trash basket, the item disappears from the environment and can not be involved in the state representation anymore. While working in this state space, the imitator has knowledge whether it is carrying any object right now or not. This is defined in term of state features. Features of states here include the current location of the imitator and of the objects, what the type of the object is, what's the pose/orientation of the object is or whether the imitator is carrying any object or not. While doing state to state mapping of both models, the imitator makes use of these state features to relate them between internal and observation model and find a corresponding state pair.

Action Space

The robot/imitator has a given set of actions (14 actions) which represent moving to a location, picking up an object (trash) or dropping an object. The robot makes use of these actions to interact with the world/environment. One thing to note here is that the robot may not have the same actions as those performed by the demonstrator during demonstration of task. For example, a ball might be thrown by the demonstrator and it went to a corner of the room by bouncing. However, the robot does not have any capability to make that ball bounce using its given capabilities. But robot can always push or grab the ball and take it to the corner of the room. Therefore, the robot must infer important aspects of the observed task where there is no guarantee that the objects will be in the same place where they were during the observation of the task.

Reward Function

When the robot acts in the environment to perform the observed task, it receives a feedback signal from the demonstrator after completion of execution The feedback signal enables a robot to understand how good it was while doing a task and in these experiments corresponds to the distance between the trash and the trash bin at the end of the execution of the imitation policy.

Results

The following shows the result of imitating a household trash cleaning task. Throughout the robot explores the policy space and learns an increasingly beneficial distance function using its function approximator. The graph in Figure 7.1 shows two policies, namely a new

policy (achieved by exploring the problem space) and a current policy (achieved by exploiting the problem space).

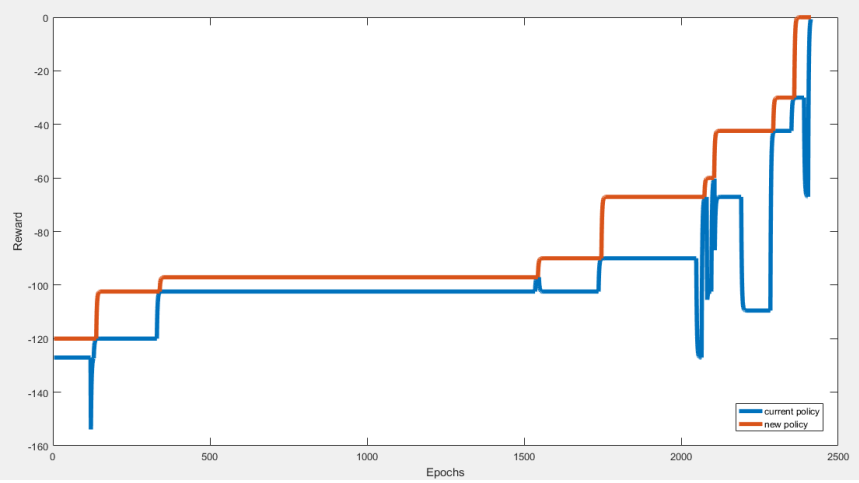

Figure 7.1 Results of Single Demonstration for A Cleaning Task

In this result, a single demonstration of the task has been given to the robot for learning to imitate the demonstrated task. Reward 0 is the maximum possible reward generated by the reward function. The reward is calculated by calculating the Euclidean distance between trash items and the trash basket at the end of policy execution. Zero reward represents that all trash items have been placed into the trash basket while executing a policy by the imitator. Figure 7.1, shows that the function approximator has learned an optimum distance representation between states of the internal model and the observed model. As a result, the current policy has achieved a reward equal to the maximum i.e. zero.

Multi-Task Experiments

The Robot/imitator has also been tested to imitate a task to learn from multiple demonstration of task. In this experiment, the same task is demonstrated in different environments. Initial results in these experiments show improvements but also point at weaknesses in the exploration strategy where the system encounters long plateaus in which the explored policy produces the same reward as the current policy and thus no learning improvement is possible. Figure 7.2 represents intermediate results of imitating the same task of household cleaning in a wide range of environments. Important fact about this experiment is that the demonstration of task is from ten different environments. Here the imitator is trying to learn a generalized policy by learning a generic distance function across multiple environment. It is observable from Figure 7.2 that the imitator is learning a better policy as its moving towards a better reward. Reward is referred as average reward over these multiple demonstration

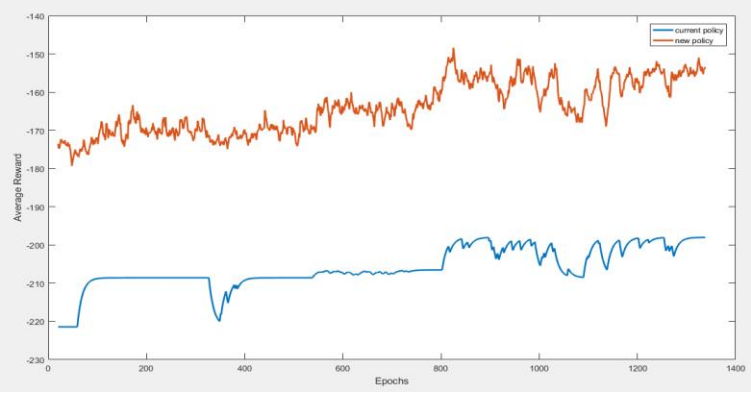

Figure 7.2 Intermediate Results for Imitating a Task in Different Environment.

Behavior to Action Mapping Experiments

Once the distance function is learned by the imitator for successfully imitating a task, the imitator will now try to learn to represent the observed behaviors in terms of MDP policies to map it to actions taken by the imitator.

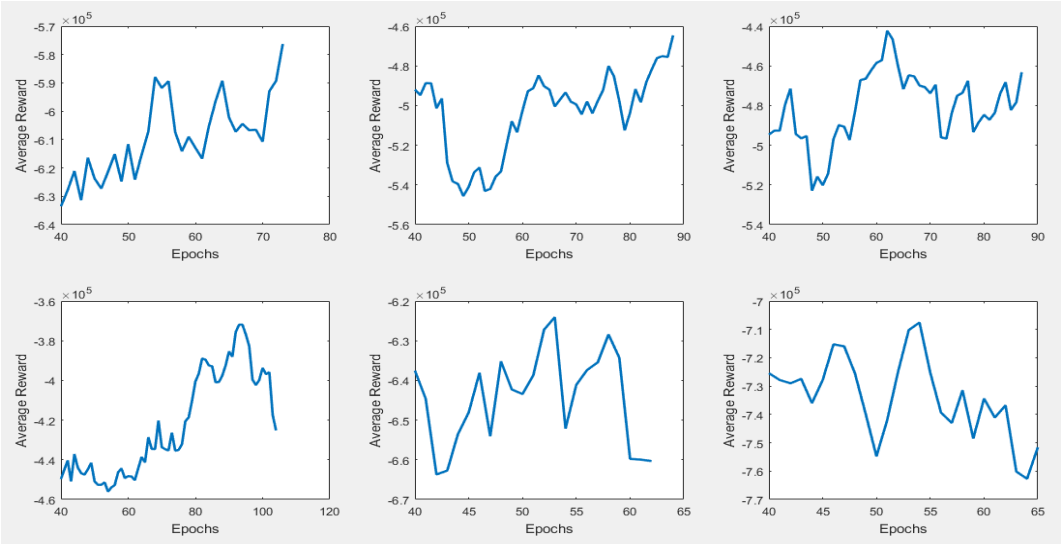

Figure 7.3 Results of Learning Observed Behavior Mapping to Robot's Internal Actions for Functional Imitation

The behavior's starting state is given as input to the internal model and then the robot comes up with a policy to map the desired behavior. As discussed earlier, the robot will report its goal internal state to a reward function which in return will provide a reward to the robot. Preliminary experiments as shown in Figure 7.3 show that the robot is successfully learning some of the behaviors while it is still exploring a number of other behaviors. These intermediate results show the learning of a behavior from multiple demonstration of a task. There are twenty-nine unique states found from one hundred demonstrations of the task.

This, together with the internal state space with 13,000 states makes a total problem space of nearly five hundred thousand states and fifteen actions to be take in each of these states. After successful completion of the preliminary experiments, the expectation is that the imitator will learn a generalized policy for representing a behavior across different tasks.

Chapter 8
Conclusions and Future Work

Robots as complex and intelligent machines have attracted significant interest from research and industry in past decades. They are getting used more and more in daily life and are applied in increasingly complex contexts. The purpose of this research is to develop methods towards the goal of making robots smart enough and sufficiently easy to use to assist people in various upcoming requirements from household assistance to military services. To make them smart for assisting mankind, they need to be trained based on their own capabilities and the environment they are working on. Until now, such robots are already assisting in very predefined structured environments where objects locations don't change and robots are preprogrammed to do their jobs in that environment. Progress should be made towards making these robots smart enough such that even if there is no fixed, robust environment, they can do their intended jobs.

In couple of past years, there has been a significant amount of progress to make these complex systems work in flexible environment. As a result, the need for approaches that allow to operate these systems without the need for complete pre-programming is increasing in order to allow these systems to enter real environments with end users. Imitation learning is one way to address this problem with a solution of making these robots useful by just demonstrating the task and having the robot learn to perform the task by observation only, without preprogramming for any task.

Conclusion

In the approach proposed in this thesis, the imitator learns an optimal correspondence between its internal model and an observed model of the demonstration. Correspondence learning is enabled by learning a distance function using a neural network based function approximator. This makes a robot capable of imitating an observed task even if its embodiment is different from the one of the demonstrator. A reinforcement learning algorithm then allows the robot to learn an optimized policy with the help of reward signals provided as feedback by the demonstrator in response to the execution of the imitator's found policy. Using the policy resulting from the current learned distance function and a policy obtained through exploration in policy space, the current distance function can be updated to improve the quality of the learned imitation policies with the help of a neural network. To test this learning of imitating demonstrated tasks, the robot has been tested to imitate tasks based on demonstrations, showing that the robot can successfully learn correspondences between internal model states and observed model states.

Future Work

The present approach needs to be extended by learning functional imitation on abstracted internal and observed model. An abstracted internal model will have complex actions like pick object and drop it in some location which will be made up of primitive actions like grasping an object, moving to a location, and releasing the object. It will dramatically decrease the state space. On the other side, hierarchical Markov task models should be developed using learned models of observation to obtain abstracted observed models. Abstracted observed model will have more abstract and complex behaviors such as a ball

is moving straight then it started bouncing and later it stopped at some location. This complex behavior is a combination of multiple simple behaviors like the ball moving in straight direction (first behavior), bouncing ball (second behavior), and ball becoming stationary (third behavior). For the sake of functional imitation, now the imitator should perform state mappings at different levels of abstraction between these internal and observed models. A similar approach can be used for state to state mapping but at corresponding hierarchies. This extension to the present work can speed up the process of search through problem space as the problem space size will be dramatically smaller than now.

References

[1] Demiris, J., & Hayes, G. (1996). Imitative learning mechanisms in robots and humans. University of Edinburgh, Department of Artificial Intelligence.

[2] Argall, B. D., Chernova, S., Veloso, M., & Browning, B. (2009). A survey of robot learning from demonstration. Robotics and autonomous systems, 57(5), 469-483.

[3] Verma, D., & Rao, R. P. (2005). Goal-based imitation as probabilistic inference over graphical models. In Advances in neural information processing systems (pp. 1393-1400).

[4] Breazeal, C., & Scassellati, B. (2002). Robots that imitate humans. Trends in cognitive sciences, 6(11), 481-487.

[5] Miyamoto, H., Schaal, S., Gandolfo, F., Gomi, H., Koike, Y., Osu, R., ... & Kawato, M. (1996). A kendama learning robot based on bi-directional theory. Neural networks, 9(8), 1281-1302.

[6] Atkeson, C. G., & Schaal, S. (1997, April). Learning tasks from a single demonstration. In Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on (Vol. 2, pp. 1706-1712). IEEE.

[7] Essa, I. A. (1999). Computers seeing people. AI magazine, 20(2), 69.

[8] Ude, A. (1999). Robust estimation of human body kinematics from video. In Intelligent Robots and Systems, 1999. IROS'99. Proceedings. 1999 IEEE/RSJ International Conference on (Vol. 3, pp. 1489-1494). IEEE.

[9] Yang, J. et al. (1997) Human action learning via hidden Markov model. In IEEE Trans. On Systems, Man and Cybernetics A: Systems and Humans 27, 34–44

[10] Mataric, M. J. (2000). Sensory-motor primitives as a basis for imitation: Linking perception to action and biology to robotics. In Imitation in Animals and Artifacts.

[11] Matarić, M. J., & Pomplun, M. (1998). Fixation behavior in observation and imitation of human movement. Cognitive Brain Research, 7(2), 191-202.

[12] Matarić, M. J., & Pomplun, M. (1998). Fixation behavior in observation and imitation of human movement. Cognitive Brain Research, 7(2), 191-202.

[13] Dautenhahn, K. (1995). Getting to know each other—artificial social intelligence for autonomous robots. Robotics and autonomous systems, 16(2), 333-356.

[14] Billard, A. (2002). ft Imitation: A Means to Enhance Learning of a Synthetic Protofanguage in Autonomous Robots. Imitation in animals and artifacts, 281.

[15] Nehaniv, C. L., & Dautenhahn, K. (2002). 2 the correspondence problem. Imitation in animals and artifacts, 41.

[16] Meltzoff, A. N. (1995). Understanding the intentions of others: re-enactment of intended acts by 18-month-old children. Developmental psychology, 31(5), 838.

[17] Attias, H. (2003, January). Planning by Probabilistic Inference. In AISTATS.

[18] Friesen, A. L., & Rao, R. P. (2011). Gaze following as goal inference: A Bayesian model. In Annual Conference of the Cognitive Science Society.

[19] Bandura, A. (2003). Observational learning. Encyclopedia of learning and memory, 2, 482-484.

[20] Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. IEEE transactions on Systems Science and Cybernetics, 4(2), 100-107.

[21] Andrew, A. M. (1999). REINFORCEMENT LEARNING: AN INTRODUCTION by Richard S. Sutton and Andrew G. Barto, Adaptive Computation and Machine Learning series, MIT Press (Bradford Book), Cambridge, Mass., 1998, xviii+ 322 pp, ISBN 0-262-19398-1,(hardback,£ 31.95).

[22] Ghahramani, Z., & Rasmussen, C. E. (2002). Bayesian monte carlo. In Advances in neural information processing systems (pp. 489-496).

[23] Wang, X., Mueen, A., Ding, H., Trajcevski, G., Scheuermann, P., & Keogh, E. (2013). Experimental comparison of representation methods and distance measures for time series data. Data Mining and Knowledge Discovery, 26(2), 275-309.

[24] Kou, H. (2006). Learn to Relate Observation to the Internal State for Robot Imitation. ProQuest.

Bibliographical Information

Bhupender Singh received his Bachelor of Science degree in Computer Science from Branch of Jawaharlal Nehru Technological University, Hyderabad, India. He started his graduate studies in August 2014 and received his Master of Science degree in Computer Science and Engineering from The University of Texas at Arlington in December 2016. His research interest is in machine perception, reinforcement learning, and machine learning.