The Dissertation Committee for Faraz Torabi
certifies that this is the approved version of the following dissertation:

Imitation Learning from Observation

Committee:

Peter Stone, Supervisor

Inderjit Dhillon, Supervisor

Garrett Warnell

Sergey Levine

Richard Tsai

Ufuk Topcu

Imitation Learning from Observation

by

Faraz Torabi

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2021

Acknowledgments

The work in this dissertation would have not been possible without the guidance and the support of my advisor, Peter Stone. Peter's way of advising and his kindness have helped to make this journey a very pleasant experience. Peter gave me a great deal of freedom to come up with my own research problem, and then offered his full support as I worked on that problem. I also want to thank Garrett Warnell who I have had the pleasure to closely collaborate with throughout my PhD journey and spend a lot of time discussing research ideas. All of the research discussed in this dissertation has been in collaboration with Peter and Garrett. I also would like to thank the rest of the dissertation committee members for their valuable advice and useful comments: Inderjit Dhillon, Sergey Levine, Richard Tsai, and Ufuk Topcu.

I have been a part of the Learning Agent Research Group (LARG), which has allowed me the pleasure of spending time with a group of smart people. This includes Sanmit Narvekar, Ishan Durugkar, Elad Liebman, Xuexu Xiao, Justin Hart, Harel Yadidsion, Reuth Mirsky, Yuqian Jiang, Shani Alkoby, Guni Sharon, Stephano Albrecht, Siddharth Desai, Yuqian Jiang, Bo Liu, Prahbat Nagarajan, Roland Fernandez Jr., and Harsh Goyal. Particularly, I want to thank Patrick MacAlpine, Josiah Hanna, Brahma Pavse, Haresh Karnan, Eddy Hudson, Ruohan Zhang, and Sean Geiger who I had the pleasure

to collaborate with.

I also would like to thank the CSEM graduate program coordinator, Stephanie Rodriguez, who has helped me with many complicated logistic problems.

I also would like to thank my undergraduate advisors, Hossein Nejat Pishkenari and Amir R. Khoei, who gave me the opportunity to explore my interests and collaborate with graduate students which led me to recognize my interest in research. I am forever grateful for their support.

Last but not least, I would like to thank my family; my parents, Farzaneh and Mehdi, and my sister, Farnaz, who have always supported me throughout my life. If I have accomplished anything, it is the result of their encouragement, love, and support. I also thank my friends that have always been there for me through ups and downs of this journey and made it a memorable experience.

Faraz Torabi

The University of Texas at Austin

August 2021

Imitation Learning from Observation

Publication No. _____

Faraz Torabi, Ph.D.
The University of Texas at Austin, 2021

Supervisors: Peter Stone
Inderjit Dhillon

Advances in robotics have resulted in increases both in the availability of robots and also their complexity—a situation that necessitates automating both the execution and acquisition of robot behaviors. For this purpose, multiple machine learning frameworks have been proposed, including reinforcement learning and imitation learning. Imitation learning in particular has the advantage of not requiring a human engineer to attempt the difficult process of cost function design necessary in reinforcement learning. Moreover, compared to reinforcement learning, imitation learning typically requires less exploration time before an acceptable behavior is learned. These advantages exist because, in the framework of imitation learning, a learning agent has access to an expert agent that demonstrates how a task should be performed. Broadly speaking, this framework has a limiting constraint in that it requires the learner to have access not only to the states (e.g., observable quantities such as spatial location) of the expert, but also to its actions (e.g., internal control signals such

as motor commands). This constraint is limiting in the sense that it prevents the agent from taking advantage of potentially rich demonstration resources that do not contain action information, e.g., YouTube videos. To alleviate this restriction, *Imitation Learning from Observation* (*IfO*) has recently been introduced as an imitation learning framework that explicitly seeks to learn behaviors by observing state-only expert demonstrations.

The *IfO* problem has two main components: (1) perception of the demonstrations, and (2) learning a control policy. This thesis focuses primarily on the second component, and introduces multiple algorithms to solve the control aspect of the problem. Each of the proposed algorithms has certain advantages and disadvantages over the others in terms of performance, stability and sample complexity. Moreover, some of the algorithms are model-based (i.e., a model of the dynamics of the environment is learned in the imitation learning process), and some are model-free. In general, model-based algorithms are more sample-efficient, whereas model-free algorithms are known for their performance. Though the focus of this thesis is on the control aspect of *IfO*, two algorithms are introduced that do integrate a perception module into one of the control algorithms. By doing so, the adaptability of that control algorithm to the general *IfO* problem is shown.

The work in this thesis is evaluated primarily in simulation, though in some cases experiments were carried out using real-world robots as well. The performance of the proposed algorithms is compared against well-known baselines and it is shown that they outperform the baselines in most cases.

Table of Contents

Acknowledgments iv

Abstract vi

List of Tables xiii

List of Figures xiv

Chapter 1. Introduction 1
- 1.1 Research Question . . . 5
- 1.2 Contributions . . . 5
- 1.3 Dissertation Overview . . . 7

Chapter 2. Preliminaries 11
- 2.1 Reinforcement Learning (RL) . . . 11
  - 2.1.1 Model-free Algorithms . . . 13
  - 2.1.2 Model-based Algorithms . . . 15
  - 2.1.3 Combination of Model-based and Model-free Algorithms 19
- 2.2 Imitation Learning . . . 20
  - 2.2.1 Behavioral Cloning (BC) . . . 21
  - 2.2.2 Inverse Reinforcement Learning (*IRL*) . . . 22
  - 2.2.3 Adversarial Imitation Learning (*AIL*) . . . 23

Chapter 3. Problem Definition 26
- 3.1 Imitation from Observation . . . 26
- 3.2 Experimental Environments . . . 27
  - 3.2.1 Simulation Environments . . . 28
    - 3.2.1.1 OpenAI Gym Environments . . . 28
    - 3.2.1.2 RoboCup3D Simulation . . . 36
  - 3.2.2 Physical Robot Environment . . . 38

Chapter 4. Behavioral Cloning from Observation **40**
4.1 Overview . . . 40
4.2 Problem Setting . . . 43
4.3 Behavioral Cloning from Observation . . . 45
4.3.1 Inverse Dynamics Model Learning . . . 46
4.3.2 Behavioral Cloning . . . 47
4.3.3 Model Improvement . . . 49
4.4 Implementation and Experimental Results . . . 50
4.4.1 Training Details and Results . . . 52
4.4.2 Discussion . . . 53
4.5 Summary . . . 58

Chapter 5. Generative Adversarial Imitation from Observation **59**
5.1 Overview . . . 60
5.2 Problem Setting . . . 61
5.3 A General Framework for Imitation from Observation . . . 61
5.4 Generative Adversarial Imitation from Observation . . . 63
5.5 Practical Implementation . . . 69
5.6 Experimental Setup and Implementation Details . . . 70
5.7 Results and Discussion . . . 71
5.8 Summary . . . 73

Chapter 6. Adversarial Imitation Learning from Video **75**
6.1 Overview . . . 75
6.2 Problem Setting . . . 77
6.3 Proposed algorithms . . . 78
6.3.1 Visual *GAIfO* with Self-Observation . . . 78
6.3.2 Visual *GAIfO* with Proprioceptive Information . . . 82
6.4 Experiments . . . 84
6.4.1 Visual *GAIfO* with Self-observation . . . 85
6.4.1.1 Setup . . . 86
6.4.1.2 Results . . . 87
6.4.2 Visual *GAIfO* with Proprioceptive Information . . . 88

6.4.2.1 Setup . . . 89
6.4.2.2 Results . . . 90
6.5 Summary . . . 94

Chapter 7. Reinforced Inverse Dynamics Modeling 96
7.1 Overview . . . 97
7.2 Problem Setting . . . 99
7.3 Reinforced Inverse Dynamics Modeling . . . 100
7.3.1 Inverse Dynamics Model Pre-training . . . 103
7.3.2 Inverse Dynamics Model Reinforcement . . . 104
7.4 Empirical Results . . . 105
7.4.1 Raw vs. Augmented State . . . 107
7.4.2 *RIDM* Applied to MuJoCo Simulation . . . 109
7.4.2.1 Baseline: *GAIfO+RL* . . . 110
7.4.2.2 Hypothesis Validation . . . 111
7.4.3 *RIDM* Applied to SimSpark RoboCup 3D Simulation . . . 112
7.4.4 *RIDM* Applied to a Physical UR5 Robot Arm . . . 116
7.5 Additional Results . . . 119
7.6 Summary . . . 120

Chapter 8. Data-Efficient Adversarial Learning for Imitation from Observation 122
8.1 Overview . . . 122
8.2 Problem Setting . . . 125
8.3 *DEALIO* . . . 127
8.4 Experiments . . . 133
8.4.1 Experimental Setup . . . 134
8.4.2 Implementation Details . . . 134
8.4.3 Results and Discussion . . . 135
8.5 Summary . . . 137

Chapter 9. Related Work **138**
9.1 Overview . . . 138
9.2 Imitation Learning . . . 140
9.2.1 Behavioral Cloning (*BC*) . . . 141
9.2.2 Inverse Reinforcement Learning (*IRL*) . . . 142
9.2.3 Adversarial Imitation Learning . . . 144
9.3 Imitation from Observation . . . 145
9.3.1 Perception . . . 145
9.3.1.1 Embodiment Mismatch . . . 146
9.3.1.2 Viewpoint Difference . . . 147
9.3.2 Control . . . 148
9.3.2.1 Model-based . . . 149
9.3.2.2 Model-free . . . 151
9.4 Summary . . . 156

Chapter 10. Conclusion and Future Work **157**
10.1 Contributions . . . 159
10.2 Future Work . . . 161
10.2.1 General Directions . . . 161
10.2.1.1 Perception . . . 161
10.2.1.2 Application to Physical Robots . . . 163
10.2.1.3 Integration . . . 164
10.2.2 Specific Directions . . . 164
10.2.2.1 Behavioral Cloning from Observation (*BCO*) . . . 164
10.2.2.2 Generative Adversarial Imitation from Observation (*GAIfO*) . . . 165
10.2.2.3 Visual Extensions of *GAIfO* . . . 166
10.2.2.4 Reinforced Inverse Dynamics Modeling (*RIDM*) 166
10.2.2.5 Data-Efficient Adversarial Learning for Imitation from Observation (*DEALIO*) . . . 167
10.3 Concluding Remarks . . . 168

Appendices **169**

Appendix A. Proofs and Additional Experimental Details for *GAIfO* **170**

A.1 Proofs 170

A.1.1 Proof of Lemma 5.4.2 170

A.1.2 Proof of the claims made for the generative adversarial regularizer 173

A.2 Experimental Details 175

Appendix B. Experimental Details for Visual Extensions of *GAIfO* **178**

B.1 *GAIfO* with Self-observation 178

B.2 *GAIfO* with Proprioceptive Information 179

Appendix C. Acronyms **182**

List of Tables

4.1 This is an extension to Figure 4.5 which provides the number of post-demonstration interactions for each case. First vertical column for each domain (d) shows the number of demonstrated trajectories. As an example, for the MountainCar domain with 1 demonstrated trajectory and $\alpha = 2E - 3$, the average number of post-demonstration interactions is 6825. We can also see at the top of the table that the number of pre-demonstration interactions is $2E3$ so the overall number of interactions would become 8825. It can be seen that almost always by increasing α or the number of demonstrated trajectories, the number of post-demonstration interactions increases. Also the overall number of interactions (combined pre- and post-demonstration interactions) in all the cases is far less than the overall number of interactions required by the other methods (*FEM* and *GAIL*). 57

7.1 *RIDM* vs. demonstrator for speed walking. 115

7.2 *RIDM* vs. demonstrator for long-distance kick offs. 116

7.3 *RIDM* vs. original PID controller vs. demonstrator. 117

A.1 Number of discriminator and policy updates performed at each iteration. 177

B.1 Number of discriminator and policy updates performed at each iteration. 179

B.2 Number of discriminator and policy updates performed at each iteration. 181

List of Figures

1.1 A block diagram showing the dependencies of the chapters to each other. The arrow shows the direction of the order in which the chapters should be read. 7

2.1 A block diagram of the agent-environment interaction in a Markov decision process (*MDP*). At each time-step (t), an agent observes a state (s_t), and takes an action (a_t). Consequently, it moves to another state (s_{t+1}) and receives feedback in the form of a cost (c_t). 13

2.2 A diagrammatic representation of *GAIL*. On the left, s (dark blue) and a (red) are the state and action features in a demonstration transition, respectively. On the right, dark blue neurons represent the imitator's states. Based on policy π (green), it performs action a (red) in the environment. *GAIL* aims to find a policy that generates state-action close to the demonstrations. To this end, it iteratively train the discriminator and the policy. The discriminator is trained in a way to output values v (brown) close to one for the data coming from the expert (left) and close to zero for the data coming from the imitator (right). The policy is trained to generate state-action pairs close to the demonstrations so that the discriminator is not able to distinguish them from the demonstrations. 25

3.1 Representative screenshots of the MuJoco domains considered in the thesis. The environments are as following: (a) InvertedPendulum, (b) InvertedDoublePendulum, (c) Reacher, (d) HalfCheetah, (e) Swimmer, (f) Hopper, (g) Ant, (h) Walker2d, (i) Disc, (j) PegInsertion, (k) GripperPusher, (l) DoorOpening. 30

3.2 Representative screenshots of the simulation domains (other than MuJoCo) considered in the thesis. The environments are as following: (a) CartPole, (b) MountainCar, MountainCarContinuous, (c) InvertedDoublePendulumBulletEnv, (d) InvertedPendulumSwingupBulletEnv, (e) HopperBulletEnv, (f) Walker2DBulletEnv. 31

3.3 Representative screenshot the simulated Nao. 37

3.4 Coordinate system used in the RoboCup 3D simulation domain. 37

3.5 The UR5 arm robot. 39

4.1 Behavioral Cloning from Observation ($BCO(\alpha)$) framework proposed in this chapter. The agent is initialized with a (random) policy which interacts with the environment and collects data to to learn its own agent-specific inverse dynamics model. Then, given state-only demonstration information, the agent uses this learned model to infer the expert's missing action information. Once these actions have been inferred, the agent performs imitation learning. The updated policy is then used to collect data and this process repeats. 41

4.2 Learning timelines for $BCO(0)$, $BCO(\alpha)$, and the *IRL* methods we compare against in this paper. The horizontal axis represents time, gray rectangles mark when each technique requires environment interactions. For *BCO*, the white and gray circles denote the inverse model and policy learning steps, respectively. For *BCO*, $\alpha|\tau^{pre}|$ is the number of post-demonstration environment interactions performed before each model- and policy-improvement step and for *IRL* methods, $|\tau^{IRL}|$ represents the total number of interactions. 43

4.3 Performance of each technique with respect to the number of post-demonstration interactions. For each domain, ten demonstrated trajectories were considered. $BCO(0)$ is depicted as a horizontal line since all environment interactions happen before the demonstration is provided. Performance values are scaled such that performance of a random policy is zero and the performance of the expert is one. Note that *GAIL* and *FEM* have access to demonstration action information whereas *BCO* does not. *The *BCO* line is not visible for the CartPole domain because *BCO* has the same performance as the expert. ***FEM* is not shown for the Reacher domain because its performance is much worse than the other techniques. 54

4.4 Performance of imitation agents with respect to the number of available demonstration trajectories. Rectangular bars and error bars represent the mean return and the standard error, respectively, as measured over 5000 trajectories. Returns have been scaled such that the performance of a random policy and the demonstrating agent are zero and one, respectively. *Note that *FEM* is not shown for the Reacher domain because its performance is much worse than the others. **Note that *BC*, *GAIL*, and *FEM* all have access to demonstration action information whereas *BCO(0) does not.* 55

4.5 The performance of *BC* and several *BCO*(α) techniques (varying α) with respect to the number of demonstrated trajectories provided. Rectangular bars and error bars represent the mean return and the standard error, respectively, as measured over 5000 trajectories. By increasing α, more post-demonstration environment interactions are allowed to occur, which increases the performance of the imitation policy. Note that *BC* has access to demonstration action information whereas *BCO* does not. Also note that the number of trajectories required for learning a fairly good policy is very small. Each demonstrated trajectory has 5, 50, and 50 transitions for each domain from left to right, respectively. Note that we did not demonstrate the results for CartPole because the results were equally perfect regardless of the value of α. 56

5.1 A diagrammatic representation of *GAIfO*. On the left, s (dark blue) and s' (light blue) are the state and next-state features in a demonstration transition, respectively. On the right, dark blue neurons represent the imitator's states. Based on policy π (green), it performs action a (red) in the environment, and encounters the next-state s' (light blue). We aim to find a policy that generates state-transitions close to the demonstrations. To this end, we iteratively train the discriminator and the policy. The discriminator is trained in a way to output values v (brown) close to zero for the data coming from the expert (left) and close to one for the data coming from the imitator (right). The policy is trained to generate state-transitions close to the demonstrations so that the discriminator is not able to distinguish them from the demonstrations. 67

5.2 Performance of algorithms in low-dimensional experiments with respect to the number of demonstration trajectories. Rectangular bars and error bars represent mean return and standard deviations, respectively. For comparison purposes, we have scaled all the performances such that a random and the expert policy score 0.0 and 1.0, respectively. **GAIL* has access to action information. 72

6.1 A diagrammatic representation of the implementation of our extension of *GAIfO* for processing visual state representations with self-observation. A stack of 4 grayscale images from $t-3$ to t (t being the current time-step) enters the policy *CNN* (top left). The policy outputs an action that the agent takes in the environment and goes to the next state in time $t+1$ (top right). A stack of 3 grayscale images from $t-1$ to $t+1$ of the agent is prepared along with a stack of 3 consecutive state images (grayscale) of the demonstrator (bottom right). When data from the imitation policy is provided, the stack from the imitator enters the discriminator and outputs the reward for taking that action (bottom left). This reward value is then used to both update the policy using *TRPO* and also update the discriminator using supervised learning (to drive the value closer to zero). When data from the demonstrator is provided, the stack from the demonstrator enters the discriminator and outputs a value which is then used to update the discriminator (to drive the value closer to one). 81

6.2 A diagrammatic representation of the extension of *GAIfO* algorithm for processing visual demonstration that leverages proprioceptive information. A multilayer perceptron (MLP) is used to model the policy, which takes the proprioceptive features s_t as the input and outputs an action a_t. The agent then executes the action in its environment. While the agent executes the policy, a video of the resulting behavior is recorded. Stacks of four consecutive grayscale images ($o_{t-2} : o_{t+1}$) from both the demonstrator and the imitator are then prepared as the input for the discriminator, which is trained to discriminate between data coming from these two sources. Finally, the discriminator function is then used as the reward function to train the policy using *PPO* (not shown). 86

6.3 Performance of algorithms in visual experiments with respect to the number of demonstration trajectories. Rectangular bars and error bars represent mean return and standard deviations, respectively. For comparison purposes, we have scaled all the performances such that a random and the expert policy score 0.0 and 1.0, respectively. 88

6.4 Performance comparison of visual *IfO* algorithms presented in this chapter. The rectangular bars and error bars represent the mean normalized return and the standard error, respectively, as measured over 1000 trials. The normalized values have been scaled in such a way that expert and random performance are 1.0 and 0.0, respectively. The x-axis represents the number of available video demonstration trajectories. 91

6.5 Performance of imitation agents with respect to the number of iterations (N). Solid colored lines represent the mean return and shaded areas represent standard errors. The returns are scaled so that the performance of the expert and random policies are zero and one, respectively. 93

7.1 *RIDM* uses a task-specific inverse dynamics model, $\mathcal{M}_\theta$, to trasnform the learner's current state s_t and the demonstrator's next state s^e_{t+1}, such that the sequence of executed actions, $\{a_t\}$, minimizes the cumulative cost from the environment. At each time step, the agent uses the demonstrator's next state, s^e_{t+1} (black dot), as the set point for $\mathcal{M}_\theta$. However, the agent actually reaches s_{t+1} (red dot) instead—which is typically *not* the set point—since *RIDM* optimizes $\mathcal{M}_\theta$ to *minimize environment task cost* instead of trajectory-tracking error. 100

7.2 Quantitative exhibition of the importance of an augmented state space for high performance on six MuJoCo domains for *GAIfO* and *TRPO*. Mean and standard deviations are over 100 policy runs. The results are brought here as we seek the *RIDM* to operate over raw state space and to show how challenging it is to learn when raw state space is considered compared to augmented space. Both methods use a neural network parameterized policy. 109

7.3 Establishment of *GAIfO+RL* as a reasonable *IfO+RL* baseline to compare against *RIDM*. All methods use the same single state-only demonstration consisting of only raw states (exclusively of joint angles). Mean and standard deviations are over 100 policy runs. All methods use a neural network parameterized policy. 111

7.4 Comparison of *RIDM* final performance against established baseline, *GAIfO+RL*, on the MuJoCo domains on the same single state-only demonstration consisting of only raw states (exclusively of joint angles). Mean and standard deviations for *GAIfO+RL* and *RIDM* are over 100 policy runs. *GAIfO+RL* uses a neural network parameterized policy. For each domain, in order of x-axis, the numbers of iterations required for *RIDM* are 700, 800, 400, 100, 900, and 1300 and for *GAIfO+RL* are 400, 800, 1000, 1000, 1200, and 1500 113

7.5 Comparison of *RIDM* learning process against established baseline, *GAIfO+RL*, on the Swimmer domain on the same single state-only demonstration consisting of only raw states (exclusively of joint angles). Solid lines represent the mean return and shaded areas represent standard deviations over 10 trials. While the shown graph is for Swimmer-v2, we observed the same qualitative trend on other domains as well. 114

7.6 Comparison of *RIDM* with PID as the IDM versus *RIDM* with NN as the IDM and the maximum performance between the randomly generated PID values on the MuJoCo domain on the same single state-only demonstration consisting of only raw states (exclusively of joint angles). Since *RIDM* with PID uses a deterministic inverse dynamics model, we do not report mean or standard deviations of our algorithm. ⋆PID version of *RIDM* used global PID gains for Walker2d-v2, unlike on other domains where it used local PD gains. 118

8.1 A block diagram representing the learning flow of *DEALIO*. A time-varying Gaussian controller $p(a|s)$ is initialized and used to collect trajectories $\{\tau^i\}$ which are then used to fit a linear Gaussian dynamics model $P(s'|s,a)$. The collected data $\{\tau^i\}$ is also used with the demonstration data D^e to train a discriminator D_θ which is a composition function of a quadratic function $h(.,s_t,s_{t+1})$ and a neural network $g_\theta(s_t,s_{t+1})$. The discriminator D_θ and the dynamics model $P(s'|s,a)$ are then used to extract the cost function $c(s,a)$ which is then used to update the controller $p(a|s)$. 123

8.2 Learning with *DEALIO* is compared against *GAIfO*. The plots represent the performance of the algorithms with respect to the number of trajectories sampled during the learning process. The solid lines represent the mean of the performances over 10 different training processes and the shaded areas represent the standard deviations. For comparison purposes, all the performances are scaled such that a random and the expert policy score 0.0 and 1.0, respectively. 136

9.1 A diagrammatic representation of categorization of the *IfO* control algorithm. The algorithms can be categorized into two groups: (1) model-based algorithms in which the algorithms may use either a forward dynamics model [30] or an inverse dynamics model [136, 85]. (2) Model-free algorithms, which itself can be categorized to adversarial methods [139, 81, 123] and cost engineering [118, 42, 77]. 148

Chapter 1

Introduction

As the number and availability of robots increases, it is becoming more and more vital that they be able to exhibit at least basic levels of autonomy. Driven by decreasing production costs, we are now starting to see the beginnings of a dramatic increase in the number of robots being deployed in a number of application areas. For example, it is estimated that sales of industrial robots reached a record of $380,550$ in 2017, 29% higher than the previous year.[1] While the number of robots is increasing, we do not expect the number of human controllers to scale accordingly, and so the ratio of people available to control the robots is decreasing. Therefore, with this increase in the numbers of robots that exhibit increasing diversity in capability, and the subsequent decrease in the ratio of people available to control these robots, it is becoming more and more vital that we devise general schemes that enable these robots to act and learn tasks autonomously.

Recent advances in technology have enabled robot manufacturers to produce more complex and sophisticated types of robots that are capable of performing many different types of tasks. As humans are prone to mistakes,

[1] https://www.businesswire.com/news/home/20180621005453/en/IFR%C2%A0Industrial-Robot-Sales-Increase-Worldwide-29-Percent/

letting them control these sophisticated robots may result in disaster. One well-studied way in which artificially-intelligent agents are currently able to learn to perform tasks autonomously is via *reinforcement learning (RL)* [128] techniques. Using these techniques, if agents are able to interact with the world and receive feedback (known as *reward*) based on how well they are performing with respect to a particular task, they are able to use their own experience to improve their future behavior. However, designing a proper feedback mechanism for complex tasks can sometimes prove to be extremely difficult for system designers. Moreover, learning based solely on one's own experience can be exceedingly slow.

Concerns such as the ones above have given rise to the study of *imitation learning* [112, 12, 3], where agents, instead, attempt to learn a task by observing another, more expert agent perform that task. Because the information about how to perform the task is communicated to the imitating agent via a demonstration, this paradigm does not require the explicit design of a reward function. Furthermore, because the demonstrations directly provide rich information regarding how to perform the task correctly, imitation learning typically learns faster (less exploration) than *RL*. Imitation learning methods have been used to successfully learn a variety of tasks such as navigation for quadrotors [38] and autonomous ground vehicles [14], and in-home service robots [33].

Importantly, most of the imitation learning literature has thus far concentrated only on situations in which the imitator not only has the ability to

observe the demonstrating agent's *states* (e.g., observable quantities such as spatial location), but also the ability to observe the demonstrator's *actions* (e.g., internal control signals such as motor commands). While this extra information can make the imitation learning problem easier, requiring it is also limiting. In particular, requiring action observations makes a large number of valuable learning resources – e.g., vast quantities of online videos of humans performing different tasks [158] – useless. For the demonstrations present in such resources, the actions of the expert are unknown. This limitation has recently motivated increased interest in the area of *imitation from observation (IfO)* [125, 77], in which agents seek to perform imitation learning using state-only demonstrations.

It is worth mentioning that this idea of imitation learning from observational data is extremely intuitive to most people. Humans and other animals have a natural ability to learn skills from observation, i.e., seeing the effects of these skills without direct knowledge of the underlying actions being taken. For example, after observing an actor doing a jumping jack, a human child can typically imitate this behavior despite not knowing anything about what's going on inside the actor's brain and nervous system. The main focus of this thesis is extending this ability to artificial autonomous agents.

Imitation learning from observation is especially relevant now due to the rise of deep learning and increasing number of freely-accessible online videos. As an example, 300 hours of video are uploaded to YouTube every minute[2],

[2]http://videonitch.com/2017/12/13/36-mind-blowing-youtube-facts-figures-statistics-

many of which include diverse types of tasks being performed. At the same time, recent breakthroughs in visual recognition due to advances in computational capability and deep learning have resulted in advanced techniques for object detection [159, 134], semantic segmentation [157, 60], etc. Take, for example, recent advances in pose estimation [89, 23, 22, 126], where previous approaches necessitated the use of extra equipment such as motion capture suits. New algorithms—based on deep learning—can extract skeletal joint positions from pictures or videos of people alone, allowing the community to extract rich information from existing video in which subjects were not outfitted with special equipment. The availability and accuracy of these tools has made it possible to study visual imitation in new ways.

The imitation learning from observation problem can be decomposed into two main components: (1) perception of the demonstrations, and (2) learning an autonomous control policy. While extensive current work can be directly applied to the perception part of the problem [110, 42, 126], there remains much work to be done to address the control component. The bulk of this thesis is concerned with new algorithms for this particular control problem; however, in some cases, the perception problem is also considered in the course of integration so as to demonstrate the overall capability of imitation from raw video inputs.

2017-re-post/

1.1 Research Question

The goal of this thesis research is to answer the following question:

In what ways can autonomous agents learn to imitate experts using state-only observations?

More specifically, this thesis is focused on developing algorithms for imitation learning from observation. As a means by which to answer this question, this thesis focuses on developing and evaluating two sets of new algorithms for learning controllers in the course of imitation from observation. One set of algorithms can be characterized as *model-based*, in which a model of the environment must be learned during the process of imitation. The other set of introduced algorithms can be characterized as *model-free*, where the agent attempts to directly imitate the demonstrator without learning any explicit model.

1.2 Contributions

This thesis provides the following contributions:

1. **One model-free algorithm for imitation learning from observation, and two extensions:** Model-free algorithms directly learn tasks without having explicit access to—or a learned model of—the environment. In this thesis, a model-free algorithm for imitation learning from observation, called *Generative Adversarial Imitation from Observation*

(*GAIfO*), is introduced (Chapter 5). Furthermore, two extensions of this algorithm are presented, allowing the approach to imitate directly from visual data (Chapter 6).

2. **Three model-based algorithms for imitation learning from observation:** Model-based algorithms are the algorithms that, during the process of task learning, rely upon a model of the environment. This model could either be known or learned. In this thesis, three model-based algorithms for imitation learning from observation are introduced: *Behavioral Cloning from Observation* (*BCO*) – introduced in Chapter 4 – *Reinforced Inverse Dynamics Modeling* (*RIDM*) – introduced in Chapter 7 – Data-Efficient Adversarial Learning for Imitation from Observation (*DEALIO*) – introduced in Chapter 8.

3. **Theory on applicability of the introduced model-free algorithm:** In this thesis, theoretical results that establish the applicability of *GAIfO* to the general model-free imitation from observation framework formulation are presented.

4. **Empirical evaluation of the developed algorithms on problems both in simulation and in the real world:** In this thesis, all of the algorithms are thoroughly tested either in simulation, in the real world, or both, and the performance of the algorithms is compared against well-known baselines.

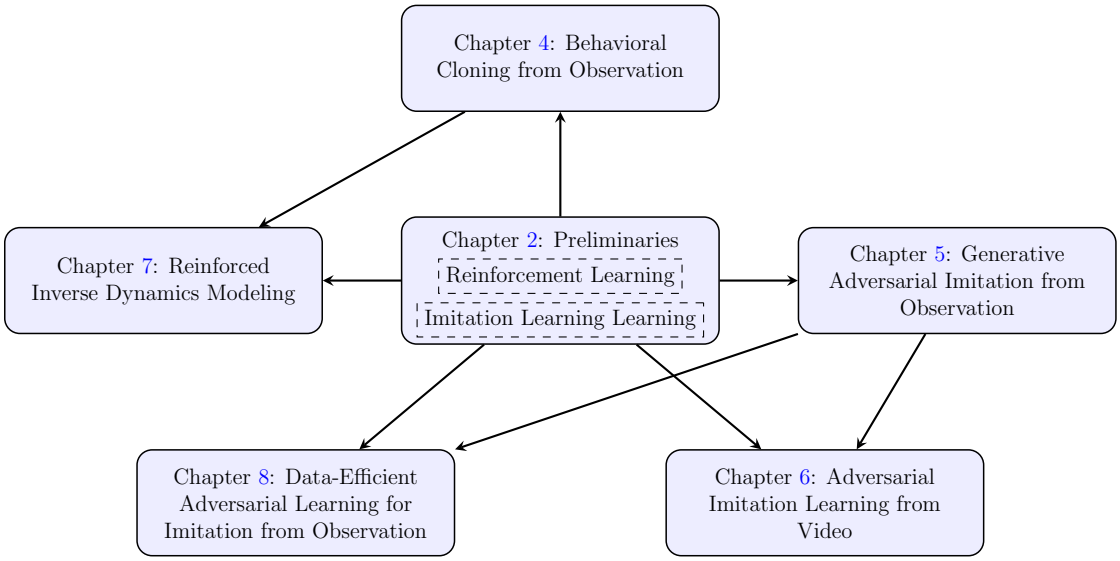

Figure 1.1: A block diagram showing the dependencies of the chapters to each other. The arrow shows the direction of the order in which the chapters should be read.

1.3 Dissertation Overview

The remainder of this document is organized as follows (for the ease of navigation, a flow chart of the dependencies of the chapters of this thesis is shown in Figure 1.1):

- **Chapter 2 - Preliminaries:** This chapter provides the notation and background information required for understanding subsequent chapters. It gives an overview of model-based reinforcement learning that is necessary for Chapter 8. It also provides background information about two categories of imitation learning algorithms: 1) behavioral cloning which is required for understanding Chapter 4, and 2) inverse reinforcement

learning which is needed for Chapters 5, 6 and 8.

- **Chapter 3 - Problem Definition:** This chapter defines the problem of imitation learning from observation and formulates it with the notation provided in Section 2.

- **Chapter 4 - Behavioral Cloning from Observation:** A model-based algorithm, Behavioral Cloning from Observation (BCO), is introduced and studied, where missing actions are first inferred from state observations and then a policy is learned using both the states and inferred actions. This algorithm partially addresses Contribution 2. Furthermore, this algorithm is evaluated in multiple MuJoCo domains to partially cover Contribution 4.

- **Chapter 5 - Generative Adversarial Imitation from Observation:** This chapter first proposes a formulation of a general model-free framework for imitation from observation and then, introduces a specific model-free algorithm, Generative Adversarial Imitation from Observation ($GAIfO$), to partially address Contribution 1. The approach encourages the imitator's state transition distribution to match that of the demonstrator. The theory of the derivation of this algorithm from the introduced model-free framework is presented to address Contribution 3. The algorithm is also evaluated in multiple MuJoCo domains to partially cover Contribution 4.

- **Chapter 6 - Adversarial Imitation Learning from Video:** Two extensions of *GAIfO* that incorporate perception modules are introduced which partially address Contribution 1. These algorithms would enable an agent to imitate directly from visual observation. These algorithms are evaluated in the MuJoCo simulator and results of experiments on multiple domains are provided to partially cover Contribution 4.

- **Chapter 7 - Reinforced Inverse Dynamics Modeling:** In this chapter, imitation learning from observation and reinforcement learning are combined to introduce another model-based algorithm. This algorithm requires fewer samples compared to *BCO* and partially addresses Contribution 2. Moreover, this algorithm is thoroughly evaluated in the MuJoCo simulator [135], in the 3D SimSpark simulator [13, 152], and on a UR5 robot to partially cover Contribution 4.

- **Chapter 8 - Data-Efficient Adversarial Learning for Imitation from Observation:** This chapter introduces another extension of the algorithm presented in Chapter 5 with much lower sample complexity that would enable agents to imitate in the real world. This chapter also partially addresses Contribution 2. This chapter provides experiments and results of this algorithm in the MuJoCo simulator to partially cover Contribution 4.

- **Chapter 9 - Related Work:** This chapter discusses work related to this thesis. At the beginning, a short overview of categories of imita-

tion learning and related algorithms are given. Then, a categorization of imitation learning from observation algorithms is introduced and a thorough survey of the works in *IfO* is provided and how they can be placed in each category.

- **Chapter 10 - Conclusion and Future Work:** This chapter presents ideas for future work and concludes the thesis.

Chapter 2

Preliminaries

We now review some preliminary concepts and notation. Section 2.1 discusses the reinforcement learning (RL) framework and the two broad categories of existing RL approaches. Afterwards, Section 2.2 briefly describes the imitation learning (IL) framework and two of its main existing approaches. Finally, at the end of this section, the state of the art approach in imitation learning is briefly mentioned.

2.1 Reinforcement Learning (RL)

Reinforcement learning [128] refers to a class of techniques that enable artificial agents to learn tasks through their own experience by minimizing a predefined cost function. RL problems are usually considered in the framework of Markov Decision Processes (*MDP*s) in which agents make decisions at discrete time intervals in an environment with the goal of minimizing their long-term cost. Given the current state of the environment, the agent takes an action that subsequently results in a change in state, where this next state is drawn from a particular distribution. Additionally, taking the action and transitioning to a new state also results in the agent receiving feedback, usu-

ally called cost (or reward), and the objective of the agent is to minimize this cost (or maximize the reward).

An MDP is denoted by the tuple $M = \langle \mathcal{S}, \mathcal{A}, P, c \rangle$ where $\mathcal{S}$ and $\mathcal{A}$ denote the state and action spaces, respectively. An agent at a particular state $s \in \mathcal{S}$, chooses an action $a \in \mathcal{A}$, based on a policy $\pi : \mathcal{S} \to \mathcal{A}$, which results in the agent moving to new state s' with probability $P(s'|s, a)$ which is specified by the environment's transition dynamics. As a result of taking an action, the agent receives feedback $c(s, a)$ specified by a cost function $c : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. The typical objective for learning agents operating in an *MDP* is to find the best policy that results in minimum overall cost. This cost is calculated by taking the expectation over the trajectories that a policy generates. For the sake of brevity, in this thesis, the expectation over the trajectories that a policy generates is referred as the expectation over that policy:

$$\mathbb{E}_{\pi}[c(s, a)] \triangleq \mathbb{E}_{(s_t, a_t) \sim \pi}[\sum_{t=0}^{\infty} \gamma^t c(s_t, a_t)], \tag{2.1}$$

where γ is a discount factor that determines how to weight future costs relative to the current cost, $s_0 \sim p_0$, $a_t \sim \pi(.|s_t)$, and $s_{t+1} \sim P(.|s_t, a_t)$. Using this notation, the objective of a reinforcement learning agent can be written as

$$\underset{\pi \in \Pi}{\operatorname{argmin}} \ \mathbb{E}_{\pi}[c(s, a)] \ . \tag{2.2}$$

Alternatively, an entropy-regularized version of this problem can be written as

$$\underset{\pi \in \Pi}{\operatorname{argmin}} \ \mathbb{E}_{\pi}[c(s, a)] - \lambda_H H(\pi) \ , \tag{2.3}$$

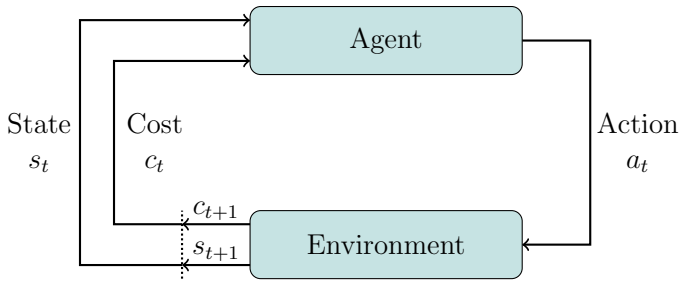

Figure 2.1: A block diagram of the agent-environment interaction in a Markov decision process (MDP). At each time-step (t), an agent observes a state (s_t), and takes an action (a_t). Consequently, it moves to another state (s_{t+1}) and receives feedback in the form of a cost (c_t).

where $H(\pi)$ and λ_H are the entropy function of the policy π and its weighting parameter, respectively. This statement aims to find a policy that minimizes the cost function and maximizes the entropy. Entropy maximization helps with exploration in the process of policy learning by encouraging a more stochastic policy and therefore improves policy optimization.

In general, there are two main categories of reinforcement learning algorithms: (1) model-free, and (2) model-based which are discussed in Sections 2.1.1 and 2.1.2, respectively.

2.1.1 Model-free Algorithms

Model-free reinforcement learning algorithms aim to allow agents to learn tasks without explicitly representing or learning the environment dynamics model. Two sets of algorithms exist in this category: (1) value-based methods, and (2) policy-based methods.

Value-based methods: Value-based methods for model-free reinforcement learning rely on functions known as value functions. For a given policy, a state-value function is defined as the negative of the expected sum of costs when starting from a given state:

$$v^{\pi}(s) = -\sum_{k=0}^{\infty} \mathbb{E}_{\pi}[c(s_{t+k}, a_{t+k})|s_t = s] \tag{2.4}$$

Given a policy and a state, the higher the value, the better the state. The process of determining the value function for a given policy is called *policy evaluation*. Using the value function for a given policy, a better policy can be found using a process known as *policy improvement*. Policy improvement modifies the policy in such a way that, at each state, an action is selected that is expected to result in a transition of the agent to a state with a higher value. Policy evaluation and policy improvement can be repeated until the optimal value function is found, i.e.,

$$v^{*}(s) = \min_{\pi} \mathbb{E}_{\pi}[\sum_{k=0}^{\infty} c(s_{t+k}, a_{t+k})|s_t = s] \text{ for } \forall s \in \mathcal{S} \tag{2.5}$$

Usually the environment's transition dynamics are not known, and so it is hard to figure out what state the agent transitions into after taking an action. Therefore, even though state-values are enough to define optimality, state-action-values are defined which are more useful:

$$q^{\pi}(s, a) = -\sum_{k=0}^{\infty} \mathbb{E}_{\pi}[c(s_{t+k}, a_{t+k})|s_t = s, a_t = s] \tag{2.6}$$

in which the higher the value, the better the state-action pair. Two popular value-based approaches are Q-learning [149, 148] and *SARSA* [108].

Policy-based methods: Policy-based methods for model-free RL seek to directly learn a parametrized mapping from states to actions (ie., a policy, π_θ) that maximizes the return [129]:

$$J(\theta) = v^{\pi_\theta}(s_0) \tag{2.7}$$

where $J(\theta)$ is the value of the initial state under policy π_θ. Policy-based methods typically use numerical optimization techniques such as gradient descent. By taking the gradient of this return and moving a step in the direction of the gradient, the parametrized policy can be improved.

Many policy-based algorithms have been developed. Some popular ones are REINFORCE [150], Trust Region Policy Optimization ($TRPO$) [115] and Proximal Policy Optimization (PPO) [116].

2.1.2 Model-based Algorithms

In contrast to model-free algorithms, model-based reinforcement learning algorithms involve the agent learning or having access to explicit representations of the transition dynamics model. If the dynamics model is available, then the problem of finding a policy is called *planning*, and it can be solved using different derivative-based or derivative-free methods such as stochastic optimization [122] or Monte-Carlo tree search ($MCTS$) [18]. One popular derivative-based method is linear quadratic regulator (LQR) [75, 121] which will be used in Chapter 8. Therefore, LQR is discussed in more detail in the following.

Linear quadratic regulator (*LQR*): *LQR* makes several strict assumptions regarding the features of the system such as the environment's model and the cost function. Specifically, *LQR* assumes that both the environment model and the policy can be represented as linear functions (in this thesis, the word policy, denoted π, is used when the policy is modeled with a neural network and the word controller, denoted p, is used when the policy is modeled with a linear function. This distinction is made to avoid confusion because of the frequent switching between these two models later in this chapter and in Chapter 8. These terms are used in previous work as well to refer to these types of policy modelings [24].). *LQR* is a trajectory-centric reinforcement learning algorithm, meaning that this algorithm results in a local controller, $p(a|s)$, for a very specific task with a specific initial state and goal. In the case of deterministic dynamics:

$$s_{t+1} = f(s_t, a_t) = F_t \begin{bmatrix} s_t \\ a_t \end{bmatrix} + f_t \tag{2.8}$$

where F_t and f_t are known matrices and vectors, respectively. Furthermore, the cost function is considered to be quadratic:

$$c(s_t, a_t) = \frac{1}{2} \begin{bmatrix} s_t \\ a_t \end{bmatrix}^T C_t \begin{bmatrix} s_t \\ a_t \end{bmatrix} + \begin{bmatrix} s_t \\ a_t \end{bmatrix}^T c_t \tag{2.9}$$

where C_t and c_t are also known matrices and vectors, respectively.

LQR attempts to solve an optimization problem that returns the actions that have the highest return in the course of an episode. Assuming that the episode has T time-steps, the optimization problem becomes:

$$\min_{a_1,\ldots,a_T} c(s_t, a_t) + c(f(s_1, a_1), a_2) + \ldots + c(f(f(\ldots)\ldots), a_T) \tag{2.10}$$

Solving this optimization problem, results in optimal actions in the form of:

$$a_t = K_t s_t + k_t \tag{2.11}$$

where K_ts and k_ts are matrices and vectors which are combinations of F_ts, C_ts, f_ts, and c_ts that can be computed for each time-step. This optimization problem can be solved efficiently in terms of the number of interactions required with the environment since everything is defined explicitly (no approximation is needed). This feature has made the algorithm particularly popular in robotics, where environment interaction can be very expensive.

K_t and k_t in Equation 2.11 are computed using an algorithm known as LQR backward recursion. First, starting from time T (terminal), the algorithm takes the derivative of the return, Q_T (which is just the cost for the last time-step), with respect to the parameters of a_T, i.e. K_T and k_T. Setting the expression for the gradient to zero and solving for K_T and k_T yields expressions for these parameters as functions of C_T and c_T. Then, going backwards, for time $T-1$, again the return, Q_{T-1} (sum of cost from $T-1$ to T), is computed, and expressions for K_{T-1} and k_{T-1} are found using the same procedure as at time step T. This backward recursion process which is presented in Algorithm 1 continues until time 0. Finally, K_t and k_t are used, either in an open-loop or a closed-loop fashion, to generate a_t according to Equation 2.11. This process is called forward recursion.

LQR can also be used under the assumption of linear Gaussian dy-

Algorithm 1 *LQR Backward Recursion*

for $t = T$ to 1 **do**
  $Q_t = C_t + F_t^T V_{t+1} F_t$
  $q_t = c_t + F_t^T V_{t+1} f_t + F_t^T v_{t+1}$
  $Q(s_t, a_t) = \frac{1}{2} \begin{bmatrix} s_t \\ a_t \end{bmatrix}^T Q_t \begin{bmatrix} s_t \\ a_t \end{bmatrix} + \begin{bmatrix} s_t \\ a_t \end{bmatrix}^T q_t$
  $a_t \leftarrow \text{argmin}_{a_t} Q(s_t, a_t) = K_t s_t + k_t$
  $K_t = -Q_{a_t,a_t}^{-1} Q_{a_t,s_t}$
  $k_t = -Q_{a_t,a_t}^{-1} q_{a_t}$
  $V_t = Q_{s_t,s_t} + Q_{s_t,a_t} K_t + K_t^T Q_{a_t,s_t} + K_t^T Q_{a_t,a_t} K_t$
  $v_t = q_{s_t} + Q_{s_t,a_t} + K_t^T Q_{a_t} + K_t^T Q_{a_t,a_t} k_t$
  $V(s_t) = \frac{1}{2} s_t^T V_t s_t + s_t^T v_t$

namics, i.e.,

$$s_{t+1} \sim P(s'|s,a) = \mathcal{N}(f(s_t, a_t), \Sigma_t) = \mathcal{N}(F_t \begin{bmatrix} s_t \\ a_t \end{bmatrix} + f_t, \Sigma_t) \tag{2.12}$$

regardless of what the covariance is, the solution is exactly the same as the deterministic case:

$$a_t = K_t s_t + k_t$$

where K_t and k_t can again be calculated following LQR backward recursion.

Note that to compute the controllers as described above, one must know the dynamics model parameters F and f. If the model is known but non-linear or the cost function is known but non-quadratic, one can use the iterative Linear Quadratic Regulator ($iLQR$) method [75], which employs Taylor expansion. In brief, at each iteration, $iLQR$ generates linear and quadratic estimates of the dynamics and cost, respectively, using the Taylor expansion of each function about some fixed points. LQR backward recursion is then

used to find the controller, which is then used to generate new rollouts and subsequently update the fixed points used for expansion. The entire process is repeated until convergence.

To use LQR in cases where the dynamics model is unknown, the agent must try to learn that model through experience [73]. Because full dynamics models are typically complex and non-local, typical approaches to model-learning attempt to fit a set of local, linear Gaussian dynamics models to the available data at each iteration. However, with such local models, if the agent deviates drastically from the trajectory that the dynamics is learned on, the learned model will not be able to produce accurate predictions. Therefore, at each LQR iteration, the current trajectory distribution should be kept close to the previous one. To achieve this, the KL-divergence of two distributions is kept below a threshold, i.e., $D_{KL}(p(\tau)||\bar{p}(\tau)) < \epsilon$, where $p(\tau)$ and $\bar{p}$ are the trajectory distributions of the new and old controllers, respectively. Accordingly, the optimization problem becomes:

$$min \sum_{t=1}^{T} E_{p(x_t,u_t)}[c(x_t,u_t)] \text{ s.t. } D_{KL}(p(\tau)||\bar{p}(\tau)) < \epsilon$$

This optimization problem can be solved, e.g., by using the Lagrangian and Dual Gradient Descent (DGD).

2.1.3 Combination of Model-based and Model-free Algorithms

As mentioned at the beginning of Section 2.1, reinforcement learning algorithms are usually divided into two categories: model-free and model-

based. Since model-free algorithms can deal with complex dynamics, they typically perform better in the absence of any prior knowledge. However, they also suffer from high sample complexity [67, 116]. Model-based algorithms, on the other hand, are more sample-efficient [27], but add additional complexity in that they require specific model estimates.

Combining the sample-efficiency of model-based algorithms and the high final performance of model-free algorithms motivated the development of *PILQR* [24]. This algorithm integrates model-based updates from the Linear Quadratic Regulator using fitted linear models [73] with Policy Improvement with Path Integrals (PI^2) [133], which is a model-free reinforcement learning algorithm based on stochastic optimal control. This is accomplished by forming a quadratic approximation of the cost, using $iLQR$ to find an optimal policy with respect to the approximated cost, and performing a subsequent update with PI^2 on the residual cost [24]. This algorithm is later used in Chapter 8

2.2 Imitation Learning

Imitation learning is a more efficient learning framework compared to reinforcement learning. In reinforcement learning, agents must interact with the world and receive feedback (known as cost) based on how well they are performing with respect to a particular task, and they use their own experience to improve their future behavior. However, designing a proper feedback mechanism for complex tasks can sometimes prove to be extremely difficult for system designers. Moreover, learning based solely on one's own experience can

be exceedingly slow. Concerns such as the ones above have given rise to the study of imitation learning [112, 12, 3], where agents instead attempt to learn a task by observing another, more expert agent perform that task. Because the information about how to perform the task is communicated to the imitating agent via demonstrations, this paradigm does not require the explicit design of a cost function. Moreover, because the demonstrations directly provide rich information regarding how to perform the task correctly, imitation learning typically requires fewer interactions with the environment compared to RL. Imitation learning is typically defined in the context of an MDP *without* an explicitly-defined cost function, i.e., $M \setminus c$. The learning problem is for an agent to determine an *imitation policy*, $\pi : S \rightarrow A$ that the agent may use in order to behave like the expert, using a provided set of state-action expert demonstrations $D^e = \{\tau_i^e\} = \{\tau_1^e, \tau_2^e, \cdots\}$ in which τ_i^e is a demonstrated state-action trajectory $\{(s^e, a^e)\}_i = \{(s_0^e, a_0^e), (s_1^e, a_1^e), \cdots, (s_N^e, a_N^e)\}_i$.

There are three main imitation learning approaches: (1) behavioral cloning, (2) approaches based on inverse reinforcement learning, and (3) adversarial methods. These methods are described in Sections 2.2.1, 2.2.2, and 2.2.3, respectively.

2.2.1 Behavioral Cloning (BC)

Behavioral cloning (BC) [9, 107, 26] is a common approach to imitation learning. The agent receives as training data both the encountered states and actions of the demonstrator, and then uses a regressor to replicate the

expert's policy [106]. To be more precise, these methods require access to some state-action demonstrated trajectories, $\{\tau_i^e\}$, and then learn a direct mapping from states to actions using a supervised learning algorithm such as maximum likelihood estimation (MLE)

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \prod_{(s,a)\in\{\tau_i^e\}} \pi_\theta(a|s) \tag{2.13}$$

where π is a policy with parameter θ that takes state as input and outputs a Gaussian distribution that the action can be sampled from. This method is powerful in the sense that it is capable of imitating the demonstrator immediately without having to interact with the environment. However, the disadvantage of this algorithm is that it requires large amounts of data because of the well-known compounding error problem caused by covariate shift [106, 107]

2.2.2 Inverse Reinforcement Learning (*IRL*)

Another general approach to imitation learning is based on inverse reinforcement learning (IRL). The first step of this approach is exactly IRL, i.e., one seeks to learn a cost function based on the given state-action demonstrations. This cost function is learned such that it is minimal for the trajectories demonstrated by the expert and maximal for every other policy [1]. However, since the problem is underconstrained — many policies can lead to the same (demonstrated) trajectories — additional constraints are typically imposed as regularizers, e.g., encouraging policies with high entropy ($MaxEnt\ IRL$) [162].

IRL problems can be formulated as the following optimization problem

$$IRL_{\psi}(\pi^e) = \underset{c \in \mathbb{R}^{\mathcal{S} \times \mathcal{A}}}{\operatorname{argmax}} - \psi(c) + (\min_{\pi \in \Pi} - \lambda_H H(\pi) + \mathbb{E}_{\pi}[c(s,a)]) - \mathbb{E}_{\pi^e}[c(s,a)] \ , \tag{2.14}$$

where $\psi(c) : \mathbb{R}^{\mathcal{S} \times \mathcal{A}} \to \overline{\mathbb{R}}$ is a convex cost function regularizer, π^e is the expert policy, Π is the space of all the possible policies, and $H(\pi)$ and λ_H are the entropy function of the policy π and its weighting parameter respectively. The output here is the desired cost function. The second step of this framework is to input the learned cost function into a standard reinforcement learning problem as formulated in Equation 2.3.

2.2.3 Adversarial Imitation Learning (*AIL*)

Recently, Ho and Ermon [49] have developed an algorithm, Generative Adversarial Imitation Learning (*GAIL*), showing that, by considering a specific function as the cost regularizer $\psi(c)$, the pipeline described in Section 2.2.2 can be solved instead as

$$\min_{\pi \in \Pi} \max_{D \in (0,1)^{\mathcal{S} \times \mathcal{A}}} - \lambda_H H(\pi) + \mathbb{E}_{\pi}[\log(D(s,a)] + \mathbb{E}_{\pi^e}[\log(1 - D(s,a))] \ , \tag{2.15}$$

where $D : \mathcal{S} \times \mathcal{A} \to (0,1)$ is a classifier trained to discriminate between the state-action pairs that arise from the demonstrator and the imitator. Excluding the entropy term, the loss function in (2.15) is similar to the loss of generative adversarial networks [40]. Instead of first learning the cost function and then learning the policy on top of that, this method directly learns

the optimal policy by bringing the distribution of the state-action pairs of the imitator as close as possible to that of the demonstrator. In order to do so, it uses a neural network structure depicted in Figure 2.2. On the right, the imitator uses policy π that gets state s as input and outputs action a. Then a discriminator D is trained to discriminate between the data that is coming from the imitator and the demonstrator by outputing values close to one and zero for the demonstrator and the imitator, respectively. After the discriminator is updated, when it outputs a value for a given state-action pair, that value represents the closeness of the data to that of the demonstrator. Therefore, the output value can be used as a cost indicating how good the state-action pair is. Using this feedback, the policy can be learned by an *RL* method. Recently, there has been research on methods that seek to improve on *GAIL* by, e.g., increasing sample efficiency [69, 111] and improving reward representation [35, 102].

The imitation learning algorithms described thus far are concentrated on situations that the learning agents have access to both the demonstrators' states and actions. Requiring access to the demonstrators' actions is restrictive in the sense that it precludes using a large amount of demonstration data where action sequences are not given. For instance, there is a great number of tutorial videos on YouTube that only provide the observer knowledge of the demonstrators state trajectory. This limitation has motivated the research described in the rest of this thesis to develop algorithms for imitation learning from observation (IfO), in which agents seek to perform imitation learning

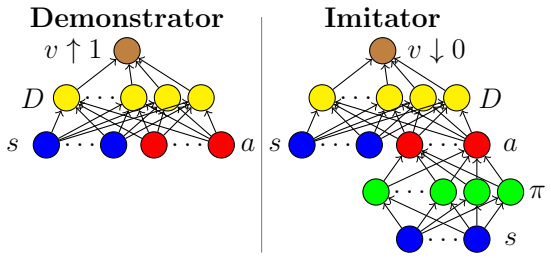

Figure 2.2: A diagrammatic representation of *GAIL*. On the left, s (dark blue) and a (red) are the state and action features in a demonstration transition, respectively. On the right, dark blue neurons represent the imitator's states. Based on policy π (green), it performs action a (red) in the environment. *GAIL* aims to find a policy that generates state-action close to the demonstrations. To this end, it iteratively train the discriminator and the policy. The discriminator is trained in a way to output values v (brown) close to one for the data coming from the expert (left) and close to zero for the data coming from the imitator (right). The policy is trained to generate state-action pairs close to the demonstrations so that the discriminator is not able to distinguish them from the demonstrations.

using state-only demonstrations.

Chapter 3

Problem Definition

In this chapter, the problem of interest, Imitation from Observation (*IfO*), and the experimental settings in which it will be studied will be fully discussed. More specifically, the *IfO* problem problem setting is described in Section 3.1 using the notations introduced Chapter 2. In Section 3.2, the environments used for experiments throughout this thesis will be described.

3.1 Imitation from Observation

Imitation from Observation is the problem faced by autonomous agents attempting to learn tasks by observing state-only demonstrations of those tasks. In traditional imitation learning, these demonstrations consist of both states and actions, and the goal for the imitator at each state is to take an action that is as close as possible to what the expert would take. As described in Chapter 2, there are multiple categories of approaches developed for imitation learning such as behavioral cloning and inverse reinforcement learning. In the case of imitation from observation, however, the demonstrations that the agent receives are limited to the expert's state-only trajectories. In this framework, the goal is for the actions of the demonstrator and imitator to have

the same effect on the environment (performing the task), rather than taking exactly the same actions. Thus, in imitation from observation, the goal is for the imitator to learn behaviors that allow it to exhibit state transitions that look like the state transitions exhibited by the demonstrator.

More formally, similar to imitation learning, the imitating agents are considered to operate in the context of an MDP *without* an explicitly-defined cost function, i.e., $M \backslash c$. Instead of a cost function, the agent instead has access to state-only expert demonstrations $D^e = \{\tau_i^e\} = \{\tau_1^e, \tau_2^e, \cdots\}$, in which each τ_i^e denotes a demonstrated state-only trajectory $\{(s^e)\}_i = \{s_0^e, s_1^e, \cdots, s_N^e\}_i$. In this context, the imitation from observation learning problem is for the agent to learn an *imitation policy*, $\pi : S \rightarrow A$ that the agent may use to replicate the behavior of the demonstrator for the purposes of having the same effect on the environment and, ultimately, successfully performing the demonstrated task.

3.2 Experimental Environments

Throughout this thesis, multiple algorithms for imitation learning from observation will be designed, each of which will be tested in multiple experimental domains. Depending on the algorithm, these may be simulation domains, real-world domains, or both. Each of the experimental environments that will be used in this thesis are described in the remainder of this section.

3.2.1 Simulation Environments

The algorithms proposed in this thesis are tested in simulation environments which approximately model the physics of the real world. Leveraging simulation as an experimental domain has several advantages over real-world domains. One such advantage is that evaluation in simulation domains is less time consuming in that a higher number of experiments can be run in parallel on different machines and that the experiments can be run at speeds that are faster than real time. Another advantage is that simulation domains are relatively cheaper than real world experiments, both in the sense that there is no need for the purchase of equipment (such as robots), and also in the sense that expensive accidents and collisions can be avoided. Moreover, simulated experiments require less supervision than experiments done in the real world.

3.2.1.1 OpenAI Gym Environments

Most of the simulation domains used in this thesis are developed by OpenAI, an AI research company.[1] The toolkit developed by this company for developing and comparing *RL* algorithms is called OpenAI Gym. Gym maintains a collection of environments that are easy to interact with in Python. One set of these environments consists of classic control domains which have been very popular in control theory literature such as the cartpole problem where the goal is to balance a pole on a car. The state space, action space, or both are discrete for most of these domains. Another set of domains have

[1] https://openai.com

continuous state or action spaces, and they utilize a physics engine called MuJoCo [135] (Multi-Joint dynamics with Contact). MuJoCo has been developed to facilitate research in different areas of science, one of which is reinforcement learning. MuJoCo enables researchers to develop their own experiment domains, and the Gym library provides multiple environments of this kind such as the Reacher task, in which the goal is for a 2D arm robot to reach a randomly specified target point.

One other popular module in *RL* that is used in this thesis is Pybullet [25]. This module uses a physics engine called Bullet Physics[2] which is open source and freely available to public. Pybullet has adapted most of the MuJoCo environments developed by OpenAI Gym for use with the Bullet physics engine. In the following, the domains that are used in thesis are described in detail.

- **CartPole:** This environment is one of Gym's classic control domains in which the goal is to balance a pole on a car as shown in Figure 3.2(a). The state space is four dimensional and continuous, and the action space is two dimensional and discrete. This domain is used in the experiments in Chapter 4.

- **MountainCar:** Similar to the CartPole domain, MountainCar is also a Gym classic control domain. In this domain, there is a car between two mountains as shown in Figure 3.2(b) and the goal is to drive the car

[2]https://github.com/bulletphysics/bullet3

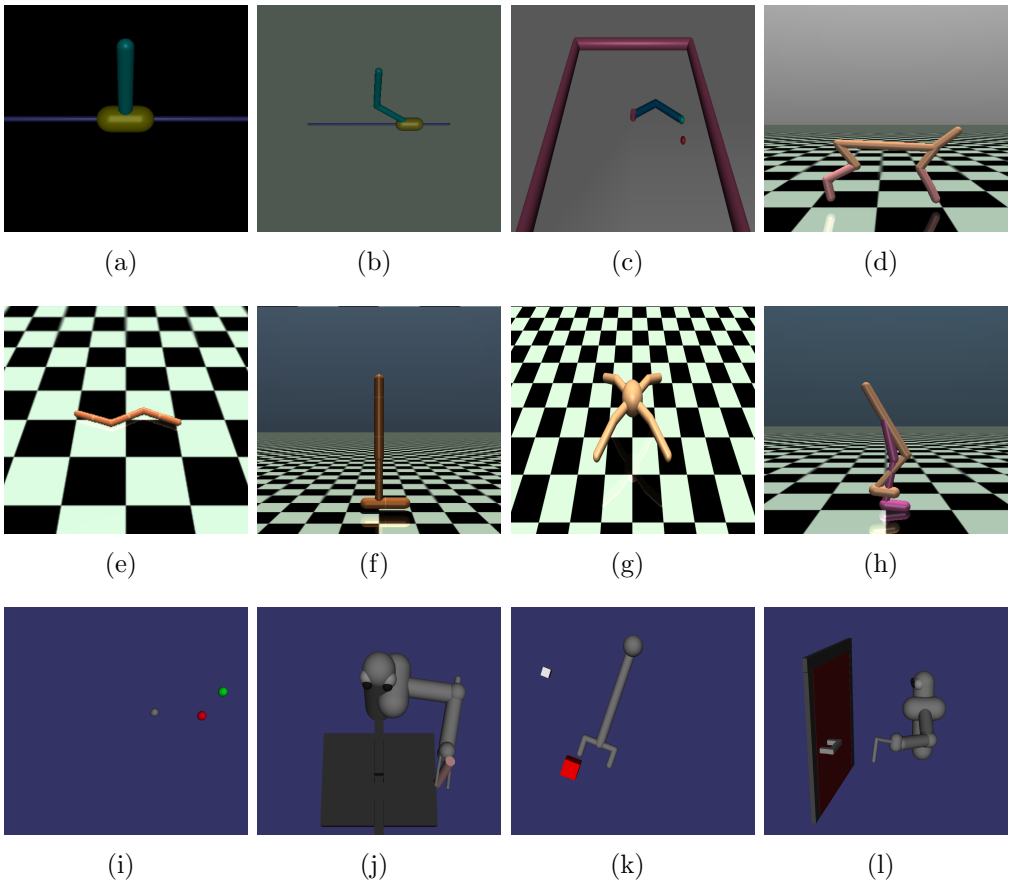

Figure 3.1: Representative screenshots of the MuJoco domains considered in the thesis. The environments are as following: (a) InvertedPendulum, (b) InvertedDoublePendulum, (c) Reacher, (d) HalfCheetah, (e) Swimmer, (f) Hopper, (g) Ant, (h) Walker2d, (i) Disc, (j) PegInsertion, (k) GripperPusher, (l) DoorOpening.

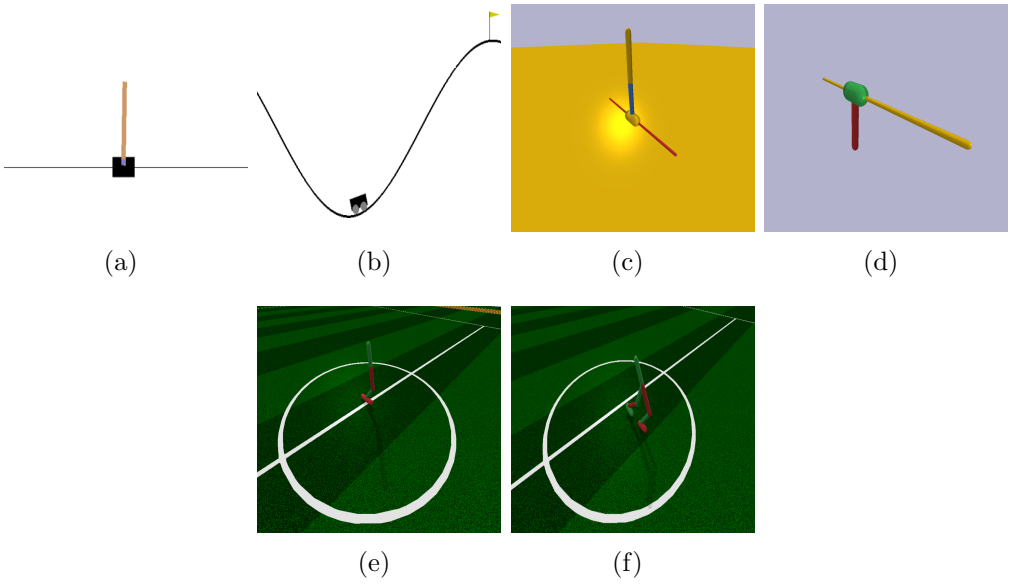

Figure 3.2: Representative screenshots of the simulation domains (other than MuJoCo) considered in the thesis. The environments are as following: (a) CartPole, (b) MountainCar, MountainCarContinuous, (c) InvertedDoublePendulumBulletEnv, (d) InvertedPendulumSwingupBulletEnv, (e) HopperBulletEnv, (f) Walker2DBulletEnv.

up to the top of one of them. The state space is two dimensional and continuous, and the action space is three dimensional and discrete. This domain is used in the experiments in Chapter 4.

- **MountainCarContinuous:** This domain is the same as MountainCar except that in MountainCarContinuous, the action space is one dimensional and it is continuous. This domain is used in the experiments in Chapter 6.

- **InvertedPendulum:** This environment is one of Gym's domains developed for MuJoCo in which the goal is to balance a pole on a cart (shown in Figure 3.1(a)). In this domain, the state space and the action space are four and one dimensional, respectively, and they both are continuous. This domain is used in the experiments in Chapter 6.

- **InvertedDoublePendulum:** This environment is another MuJoCo domain in which two poles are connected to each other and the bottom one is connected to a cart as shown in Figure 3.1(b). The goal is to balance the poles on the cart. The state and action spaces are eleven and one dimensional, respectively, and they both are continuous. This domain is used in the experiments in Chapter 6.

- **Reacher:** This MuJoCo domain has a 2D arm robot and randomly specified target point (Figure 3.1(c)) and the goal is to have the fingertip of the arm reach the target point. The continuous state space and the

action space are eleven and two dimensional, respectively. This domain is used in the experiments in Chapters 4 and 7.

- **HalfCheetah:** This MuJoCo domain includes a 2D cheetah robot as shown in Figure 3.1(d) and the goal is for the robot to run as fast as possible. The state and action spaces are seventeen and six dimensional, respectively. This domain is used in the experiments in Chapters 6 and 7.

- **Swimmer:** This domain is also a MuJoCo environment in which there is a swimming robot that has three links as shown in Figure 3.1(e) and the goal is for the agent to move forward as fast as possible. The state and action spaces are eight and two dimensional, respectively. This domain is used in the experiments in Chapter 7.

- **Hopper:** This MuJoCo domain includes a 2D one legged robot as shown in Figure 3.1(f) and the goal is for the robot to run as fast possible. This domain's continuous state and action spaces are eleven and three dimensional, respectively. This domain is used in the experiments in Chapters 6 and 7.

- **Ant:** The goal is to have the ant to run as fast as possible (Figure 3.1(g)). This domain has a continuous action space. The state and action space are 111 and 8 dimensional, respectively. This domain is used in the experiments in Chapters 4 and 7.

- **Walker2d:** This domain is a MuJoCo environment in which there is a two dimensional bipedal robot as shown in Figure 3.1(h) and the goal is for it to walk forward as fast as possible. The state and action spaces are seventeen and six dimensional, respectively. This domain is used in the experiments in Chapters 6 and 7.

- **InvertedDoublePendulumBulletEnv:** A Bullet's continuous control domain in which the task is to keep a double pendulum upright (Figure 3.2(c)). This domain's state and action spaces are nine and one dimensional, respectively. This domain is used in the experiments in Chapters 5 and 6.

- **InvertedPendulumSwingupBulletEnv:** Another of Bullet's continuous control domain in which a pendulum is initially hanging downward, and the task is to swing it in a way to cause it to become upward (Figure 3.2(d)). This environment's state and action spaces are five and one dimensional, respectively. This domain is used in the experiments in Chapters 5 and 6.

- **HopperBulletEnv:** A continuous control domain developed for Bullet Physics in which the task is to enable a two-dimensional one-legged robot to hop forward as fast as possible (Figure 3.2(e)). The state and action spaces are fifteen and three dimensional, respectively. This domain is used in the experiments in Chapters 5 and 6.

- **Walker2DBulletEnv:** A Bullet Physics continuous control domain in which the task is to enable a two-dimensional bipedal robot walk forward as fast as possible (Figure 3.2(f)). The state and action spaces are twenty two and six dimensional, respectively. This domain is used in the experiments in Chapter 5.

- **Disc**: A MuJoCo domain that includes a gray point particle on a disc and two target points—one red and one green—as shown in Fig. 3.1(i). The agent can push the particle in the x and y directions, and the task is for the agent to first move the particle to the red target point, and then move the particle to the green target point. The state and action spaces are ten- and two-dimensional, respectively. This domain is used in the experiments in Chapters 8.

- **PegInsertion**: Also MuJoCo domain that includes an arm with a gripper, a peg, and a plate with a hole in it as shown in Fig. 3.1(j). The task is for the agent to manipulate the arm such that the peg is inserted into the hole. The state and action spaces are twenty-six- and seven-dimensional, respectively. This domain is used in the experiments in Chapters 8.

- **GripperPusher**: This domain includes an arm with a gripper, a white particle, and a red target point as shown in Fig. 3.1(k). The task is for the agent to manipulate the arm such that it reaches the particle, grabs it, and moves it to the target point. The state and action spaces

are thirty-six- and four-dimensional, respectively. This domain is used in the experiments in Chapters 8.

- **DoorOpening**: This domain includes a robot arm and a door with a handle as shown in in Fig. 3.1(l). The task is for the agent to manipulate the arm such that it reaches the handle and opens the door. The state and action spaces are thirty-six- and six-dimensional, respectively. This domain is used in the experiments in Chapters 8.

3.2.1.2 RoboCup3D Simulation

This thesis also uses the RoboCup 3D Simulator as an experimental simulation domain. Both SimSpark [13, 13] and the Open Dynamics Engine (ODE) are used within this simulation environment. The ODE library helps model realistic simulation of rigid body dynamics, and the SimSpark software models the simulated physical multiagent system.

The RoboCup 3D Simulator environment is mainly used as part of the 3D simulation league of the annual RoboCup competition. Many teams participate each year, each of which comprises eleven players (as in human soccer), where each player is a simulated Nao humanoid robot as shown in Figure 3.3. Each Nao has 22 degrees of freedom: the neck constitutes two degrees of freedom, the arms eight, and the legs the additional 12. The state space of the simulated Nao includes the angular measurements of these joints. The games take place on a simulated soccer field similar to the one shown in Figure 3.4, and each game has two halves, each of which takes five minutes.

Figure 3.3: Representative screenshot the simulated Nao.

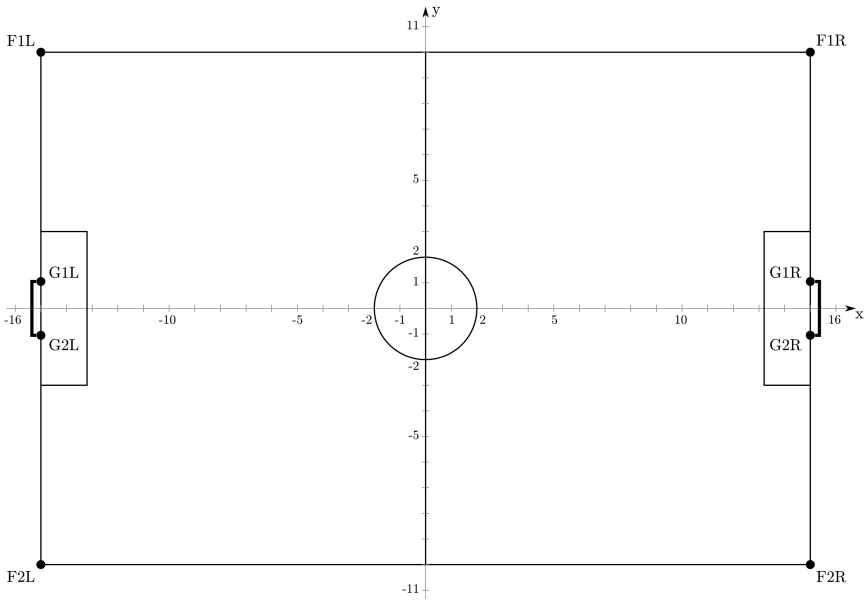

Figure 3.4: Coordinate system used in the RoboCup 3D simulation domain.

In this competition, all of the players' skills, such as walking and kicking, are developed by the participating teams. Developing these skills is challenging, and if a team finds a way to endow their simulated agents with a better walk or kick, that team would have an advantage over the others. Additionally, at the end of the competition, each of the teams is required to release a binary of their approach which can be used to play against them. The only information that other teams can glean from that binary is the joint angles (states) of the simulated agents. The two tasks that are considered for training in this thesis are a fast walk and a long kick. The tasks are described in more details in Chapter 7.

3.2.2 Physical Robot Environment

Some of the algorithms developed in this thesis are evaluated in the real world as well. While, as mentioned above, simulation environments are great for testing algorithms in many ways, they are not sufficient as a means by which to evaluate the applicability of a particular algorithm in real world. In order for an algorithm to be useful in the real world, it should be both sample efficient and safe. For these reason, in this thesis, we use a UR5 (Figure 3.5) arm robot to evaluate some of the proposed algorithms. This robot is a collaborative industrial robot that is also popular in robotics research and has six degrees of freedom[3].

[3]https://www.universal-robots.com/products/ur5-robot/

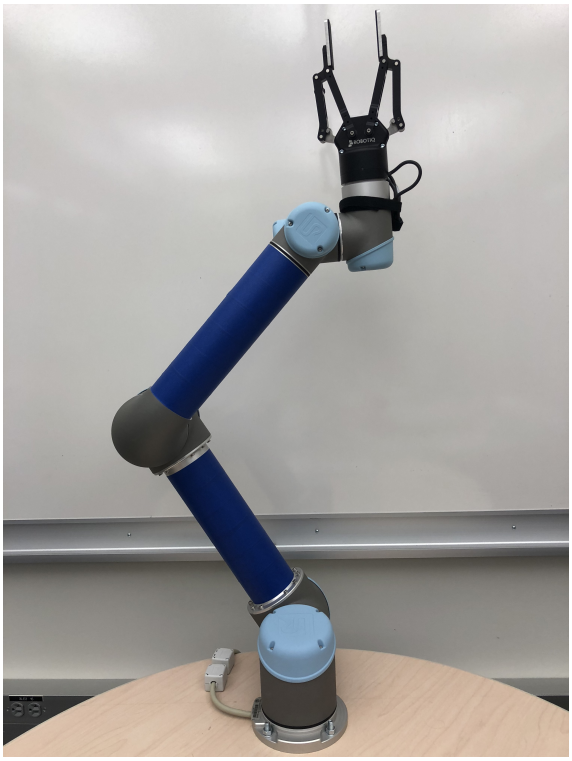

Figure 3.5: The UR5 arm robot.

Chapter 4

Behavioral Cloning from Observation

In this chapter, a model-based imitation from observation algorithm is introduced which is called Behavioral Cloning from Observation (BCO). Two versions of this algorithm are presented, a zero-shot version $BCO(0)$, and an iterative version $BCO(\alpha)$. The latter ourperforms the former but it is less sample-efficient. The proposed algorithm is evaluated in the MuJoCo simulator, and the results of experiments on multiple domains are provided.

Work in this chapter is based on a paper, *Behavioral Cloning from Observation* [136], published in the Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI 2018).

4.1 Overview

As mentioned in Chapter 1, our goal is to present imitation learning from observation algorithms. Meaning that we aim to propose algorithms for autonomous agents to learn tasks by observing a demonstrator perform the task. The challenge in this framework is that the agents only have access to the states of the demonstrator (and not the actions).

Also another challenge that the imitation learning community faces is

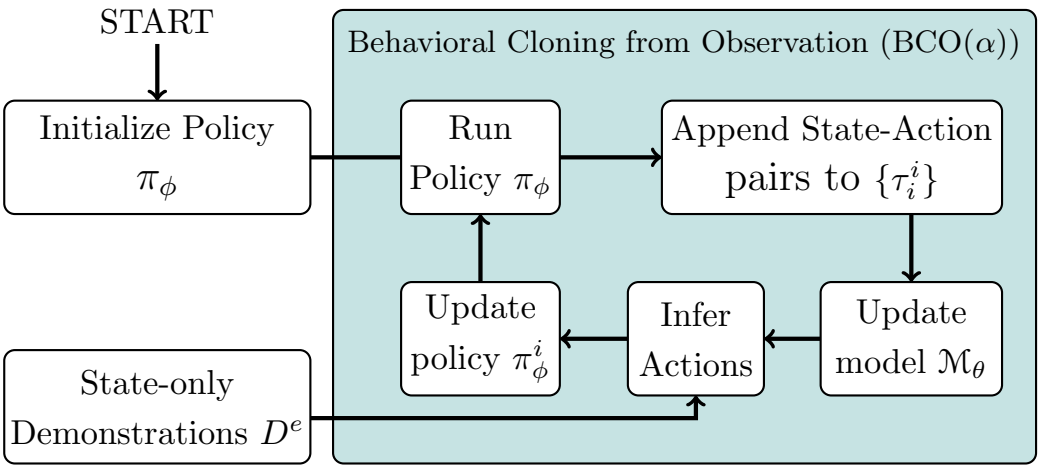

Figure 4.1: Behavioral Cloning from Observation ($BCO(\alpha)$) framework proposed in this chapter. The agent is initialized with a (random) policy which interacts with the environment and collects data to to learn its own agent-specific inverse dynamics model. Then, given state-only demonstration information, the agent uses this learned model to infer the expert's missing action information. Once these actions have been inferred, the agent performs imitation learning. The updated policy is then used to collect data and this process repeats.

the necessity of environment interaction, which can be expensive in several regards. One is the amount of time it requires: executing actions, either in the real world or in simulation, takes time. If a learning algorithm requires that a large number of actions must be executed in order to find a good imitation policy after a demonstration is presented, then there will be an undesirable amount of delay before the imitating agent will be successful. Furthermore, algorithms that require post-demonstration interaction typically require it again and again for each newly-demonstrated task, which could result in even more delay. Beyond delay, environment interaction can also be risky. For example, when training autonomous vehicles, operating on city streets while learning might endanger lives or lead to costly damage due to crashes. Therefore, we desire an algorithm for which environment interactions can be performed as a pre-processing step - perhaps in a safer environment - and where the information learned from those interactions can be re-used for a variety of demonstrated tasks.

In this chapter, we introduce an imitation learning algorithm called *behavioral cloning from observation* (*BCO*). *BCO* simultaneously addresses both of the issues discussed above, i.e., it provides reasonable imitation policies almost immediately upon observing state-trajectory-only demonstrations. First, it calls for the agent to learn a task-independent, inverse dynamics model in a pre-demonstration, exploratory phase. Then, upon observation of a demonstration without action information, *BCO* uses the learned model to infer the missing actions. Finally, *BCO* uses the demonstration and the inferred actions

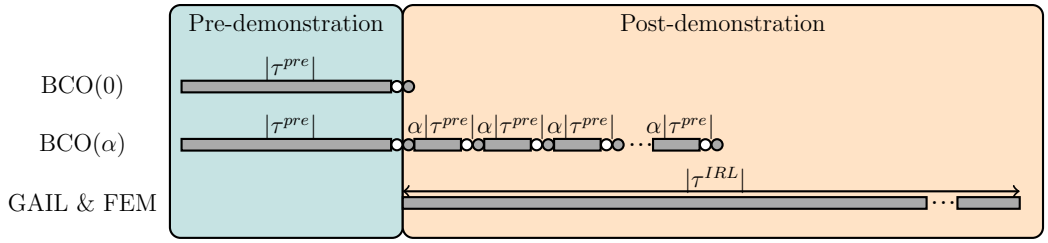

Figure 4.2: Learning timelines for $BCO(0)$, $BCO(\alpha)$, and the IRL methods we compare against in this paper. The horizontal axis represents time, gray rectangles mark when each technique requires environment interactions. For BCO, the white and gray circles denote the inverse model and policy learning steps, respectively. For BCO, $\alpha|\tau^{pre}|$ is the number of post-demonstration environment interactions performed before each model- and policy-improvement step and for IRL methods, $|\tau^{IRL}|$ represents the total number of interactions.

to find a policy via behavioral cloning. If post-demonstration environment interaction is possible, BCO additionally specifies an iterative scheme where the agent uses the extra interaction time in order to learn a better model and improve its imitation policy. This iterative scheme therefore provides a trade-off between imitation performance and post-demonstration environment interaction. We experimentally test this algorithm in the MuJoCo simulator on several domains and find that it results in comparable (and, in some cases, better) performance to other algorithms that require either more information, more environment interaction, or both.

4.2 Problem Setting

Recall that, as defined in Chapter 2, we consider agents acting within the broad framework of Markov decision processes (MDPs). We denote an

MDP using the 4-tuple $M = \{\mathcal{S}, \mathcal{A}, P, c\}$, where $\mathcal{S}$ is the agent's state space, $\mathcal{A}$ is its action space, $P(s'|s,a)$ is a function denoting the probability of the agent transitioning from state s to s' after taking action a, $c : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is a function specifying the immediate cost that the agent receives for taking a specific action in a given state. In this framework, agent behavior can be specified by a policy, $\pi : \mathcal{S} \rightarrow \mathcal{A}$, which specifies the action (or distribution over actions) that the agent should use when in a particular state.

In this chapter, we will be interested in the *inverse dynamics model*, $\mathcal{M} = P(a|s,s')$, which is the probability of having taken action a given that the agent transitioned from state s to s'. Moreover, we specifically seek task-independent models. We assume that some of the state features are specifically related to the task and others specifically to the agent, i.e., a given state s can be partitioned into an *agent-specific state*, s^a, and a *task-specific state*, s^t, which are members of sets $\mathcal{S}^a$ and $\mathcal{S}^t$, respectively (i.e., $\mathcal{S} = \mathcal{S}^a \times \mathcal{S}^t$) [68, 42]. Using this partitioning, we define the *agent-specific inverse dynamics model* to be a function $\mathcal{M}_\theta : \mathcal{S}^a \times \mathcal{S}^a \rightarrow \mathcal{A}$ that maps a pair of agent-specific state transitions, (s^a_t, s^a_{t+1}), to a distribution of agent actions that is likely to have given rise to that transition.

In the setting of imitation from observation, one seeks to find an imitation policy from a set of state-only demonstrations $D^e = \{\tau^e_i\}$ in which each $\tau^e_i = \{(s^e)\}_i$. The specific problem that we are interested in is imitation from observation under a constrained number of environment interactions. By *environment interactions* we mean time-steps for which we require

our agent to gather new data by executing an action in its environment and observing the state outcome. We are concerned here in particular with the cost of the learning process, in terms of the number of environment interactions, both before and after the expert demonstrations are provided. Pre- and post-demonstration environment interactions are represented by τ^{pre} and τ^{post}, respectively, to denote sets of interactions (s, a) that must be executed by a learner before and after a demonstration becomes available. In this context, we are concerned here with the following specific goal: *given a set of state-only demonstration trajectories, D^e, find a good imitation policy using a minimal number of post-demonstration environment interactions, i.e., $|\tau^{post}|$.*

In pursuit of this goal, we introduce a new algorithm for imitation learning that can operate both in the absence of demonstrator action information and while requiring no or very few post-demonstration environment interactions. Our framework consists of two components, each of which considers a separate part of this problem. The first of these components considers the problem of learning an agent-specific inverse dynamics model (Subsections 4.3.1 and 4.3.2), and the second one considers the problem of learning an imitation policy from a set of demonstration trajectories (Subsection 4.3.3).

4.3 Behavioral Cloning from Observation

We now describe our imitation learning algorithm, *BCO*, which combines inverse dynamics model learning with learning an imitation policy. We are motivated by the fact that humans have access to a large amount of prior

experience about themselves, and so we aim to also provide an autonomous agent with this same prior knowledge. To do so, before any demonstration information is observed, we allow the agent to learn its own agent-specific inverse dynamics model. Then, given state-only demonstration information, we use this learned model to infer the expert's missing action information. Once these actions have been inferred, the agent performs imitation learning via a modified version of behavioral cloning (Figure 4.1). The pseudo-code of the algorithm is given in Algorithm 2.

4.3.1 Inverse Dynamics Model Learning

In order to infer missing action information, we first allow the agent to acquire prior experience in the form of an agent-specific inverse dynamics model. In order to do so, we let the agent perform an exploration policy, π. In this work, we let π be a random policy (Algorithm 2, Line 2). While executing this policy, the agent performs some number of interactions with the environment, i.e., τ^{pre}. Because we seek an agent-specific inverse dynamics model as described in Section 4.2, we extract the agent-specific part of the states in τ^{pre} and store them as $\{(s^a_t, s^a_{t+1})\}$, and their associated actions, $\{a_t\}$ (Algorithm 2, Lines 5-8). Given this information, the problem of learning an agent-specific inverse dynamics model is that of finding the parameter θ for which $\mathcal{M}_\theta$ best describes the observed transitions. We formulate this problem as one of maximum-likelihood estimation, i.e., we seek θ^* as

$$\theta^* = \arg\max_\theta \Pi_{t=0}^{|\tau^{pre}|} p_\theta(a_t \mid s_t^a, s_{t+1}^a), \tag{4.1}$$

where p_θ is the conditional distribution over actions induced by $\mathcal{M}_\theta$ given a specific state transition. Any number of supervised learning techniques, denoted as "modelLearning" in Algorithm 2, may be used to solve Equation 4.1.

Some details regarding particular choices made in this chapter: For domains with a continuous action space, we assume a Gaussian distribution over each action dimension and our model estimates the individual means and standard deviations. We use a neural network for $\mathcal{M}_\theta$, where the network receives a state transition as input and outputs the mean for each action dimension. The standard deviation is also learned for each dimension, but it is computed independently of the state transitions. In order to train this network (i.e., to find θ^* in Equation 4.1), we use the Adam variant [65] of stochastic gradient decent. Intuitively, the gradient for each sample is computed by finding a change in θ that would increase the probability of a_t with respect to the distribution specified by $\mathcal{M}_\theta(s_t, s_{t+1})$. When the action space is discrete, we again use a neural network for $\mathcal{M}_\theta$, where the network computes the probability of taking each action via a softmax function.

4.3.2 Behavioral Cloning

Our overarching problem is that of finding a good imitation policy from a set of state-only demonstration trajectories, $D^e = \{\tau_i^e\}$ where $\tau_i^e = \{(s^e)\}_i$.

Note that, although the inverse dynamics model is learned using a set of agent-generated data in Section 4.3.1, the data used there is not utilized in this step. In order to use the learned agent-specific inverse dynamics model, we first extract the agent-specific part of the demonstrated state sequences and then form the set of demonstrated agent-specific state transitions $\{(s_t^{ea}, s_{t+1}^{ea})\}$ (Algorithm 2, Line 10) in which e and a stand for expert and agent-specific, respectively. Next, for each transition (s_t^{ea}, s_{t+1}^{ea}), the algorithm computes the model-predicted distribution over demonstrator actions, $\mathcal{M}_{\theta^*}(s_t^{ea}, s_{t+1}^{ea})$ and uses the maximum-likelihood action as the inferred action, $\tilde{a}_t^e$ (Algorithm 2, Line 11). Using these inferred actions, we then build the set of complete state-action pairs $\{(s^e, \tilde{a}^e)\}$.

With this new set of state-action pairs, we may now seek the imitation policy π_ϕ. We cast this problem as one of behavioral cloning, i.e., given a set of state-action tuples $\{(s_t^e, \tilde{a}_t^e)\}$, the problem of learning an imitation policy becomes that of finding the parameter ϕ for which π_ϕ best matches this set of provided state-action pairs (Algorithm 2, Line 12). We find this parameter using maximum-likelihood estimation, i.e., we seek ϕ^* as

$$\phi^* = \arg\max_\phi \Pi_{t=0}^{N} \pi_\phi(\tilde{a}_t^e \mid s_t^e) \; . \tag{4.2}$$

Some details regarding particular choices made in this chapter: For continuous action spaces, we assume our policy to be Gaussian over each action dimension, and, for discrete actions spaces we use a softmax function to

Algorithm 2 BCO(α)

Initialize the model $\mathcal{M}_\theta$ as random approximator
Set π_ϕ to be a random policy
Set $I = |\tau^{pre}|$
while policy improvement **do**
    for time-step t=1 to I **do**
        Generate samples (s_t^a, s_{t+1}^a) and a_t using π_ϕ
        Append samples (s_t^a, s_{t+1}^a), and a_t
    Improve $\mathcal{M}_\theta$ by modelLearning($\{(s_t^a, s_{t+1}^a)\}$, $\{a_t\}$)
    Generate set of agent-specific state transitions $\{(s_t^{ea}, s_{t+1}^{ea})\}$ from the demonstrated state trajectories D^e
    Use $\mathcal{M}_\theta$ with $\{(s_t^{ea}, s_{t+1}^{ea})\}$ to approximate $\{\tilde{a}_t^e\}$
    Improve π_ϕ by behavioralCloning($\{(s^e, \tilde{a}^e)\}$)
    Set $I = \alpha|\tau^{pre}|$

represent the probability of selecting each value. We let π_ϕ be a neural network that receives as input a state and outputs either the Gaussian distribution parameters or the action probabilities for continuous or discrete action spaces, respectively. We then solve for ϕ^* in (4.2) using Adam SGD, where the intuitive view of the gradient is that it seeks to find changes in ϕ that increase the probability of each inferred demonstrator action, $\tilde{a}_t$, in the imitation policy's distribution $\pi_\phi(\cdot \mid s_t)$.

4.3.3 Model Improvement

The techniques described above form the building blocks of *BCO*. If one is willing to tolerate post-demonstration environment interaction, a modified version of our algorithm can further improve both the learned model and the resulting imitation policy. This modified algorithm proceeds as follows. After

the behavioral cloning step, the agent executes the imitation policy in the environment for a short period of time. Then, the newly-observed state-action sequences are used to update the model, and, accordingly, the imitation policy itself. The above procedure is repeated until there is no more improvement in the imitation policy. We call this modified version of our algorithm $BCO(\alpha)$, where α is a user-specified parameter that is used to control the number of post-demonstration environment interactions at each iteration, $\alpha|\tau^{pre}|$. The total number of post-demonstration interactions required by $BCO(\alpha)$ can be calculated as $|\tau^{post}| = T\alpha|\tau^{pre}|$, where T is the total number of model-improvement iterations required by $BCO(\alpha)$. Using a nonzero α, the model is able to leverage post-demonstration environment interaction in order to more accurately estimate the actions taken by the demonstrator, and therefore improve its learned imitation policy. If one has a fixed budget for post-demonstration interactions, one could consider terminating the model-improvement iterations early, i.e., specify both α and T.

4.4 Implementation and Experimental Results

We evaluated $BCO(\alpha)$ in several domains available in OpenAI Gym [17]. Continuous tasks are simulated by MuJoCo [135]. These domains have different levels of difficulty, as measured by the complexity of the dynamics and the size and continuity of the state and action spaces. Ordered from easy to hard, the domains we considered are: CartPole, MountainCar, Reacher, and Ant which are described in Section 3.2. Any RL algorithm could be used

with the problem's pre-defined reward function to generate an expert policy. Since Trust Region Policy Optimization (*TRPO*) [115] has shown promising performance in simulation [77], we generated our demonstrations using agents trained with this method.

We evaluated our algorithm in two senses. First, with respect to the number of environment interactions required to attain a certain performance. In a real-world environment, interactions can be expensive which makes it a very important criterion. The second way in which we evaluate our algorithm is with respect to data efficiency, i.e., the imitator's task performance as a function of the amount of available demonstration data. In general, demonstration data is scarce, and so making the best use of it is very important.

We compared $BCO(\alpha)$ to the following methods:

1. **Behavioral Cloning (*BC*):** As presented in Section 2.2.1, this method applies supervised learning over state-action pairs provided by the demonstrator.

2. **Feature Expectation Matching (*FEM*) [50]:** A modified version of the approach presented by Abbeel and Ng [1]. It uses *TRPO* with a linear cost function in order to train neural network policies.

3. **General Adversarial Imitation Learning (*GAIL*) [49]:** As presented in Section 2.2.3, this method is state-of-the-art in *IRL*. It uses a specific class of cost functions which allows for the use of generative adversarial networks in order to do apprenticeship learning.

Note in particular that **our method is the only method that does not have access to the demonstrator's actions**. However, as our results will show, $BCO(\alpha)$ can still achieve comparable performance to these other techniques, and do so while requiring far fewer environment interactions.

4.4.1 Training Details and Results

Because both BC and $BCO(\alpha)$ rely on supervised learning methods, we use only 70% of the available data for training and use the rest for validation. We stop training when the error on the 30% validation data starts to increase. For the other methods, all available data was used in the training process. We will now discuss the architecture details for each domain.

- **CartPole:** In this domain, we considered linear models over the predefined state features for both the inverse dynamics model and the imitation policy and we only used $I = 1000$ interactions to learn the dynamics.

- **MountainCar:** In this domain, the data set for learning the inverse dynamics model is acquired by letting the agent to explore its action space for $I = 2000$ time steps. For both the imitation policy and inverse dynamics model, we used neural networks with two hidden layers, 8 nodes each, and leaky rectified linear activation functions (LReLU).

- **Reacher:** In this domain, we can partition the state-space to agent-specific features (i.e., those only related to the arm) and task-specific

features (i.e., those related to the position of the target). A neural network architecture with two hidden layers of 100 LReLU nodes are used with $I = 5000$ agent-specific state transition-action pairs in order to learn the dynamics and then this model is used to learn a policy which also has two layers but with 32 LReLU nodes.

- **Ant:** This is the most complex domain considered in this work. The state and action space are 111 and 8 dimensional, respectively. The number of interactions needed to learn the dynamics was $I = 5e5$ and the architectures for inverse dynamics learning and the policy are similar to those we used in Reacher.

4.4.2 Discussion

Each experiment was executed twenty times, and all results presented here are the average values over these twenty runs. We selected twenty trials because we empirically observed very small standard error bars in our results. This is likely a reflection of the relatively low variance in the expert demonstrations.

In our first experiment, we compare the number of environment interactions needed for $BCO(0)$ with the number for other methods (Figure 4.3.) We can clearly see how imitation performance improves as the agent is able to interact more with the environment. In the case of $BCO(0)$, the interactions with the environment happen before the policy-learning process starts,

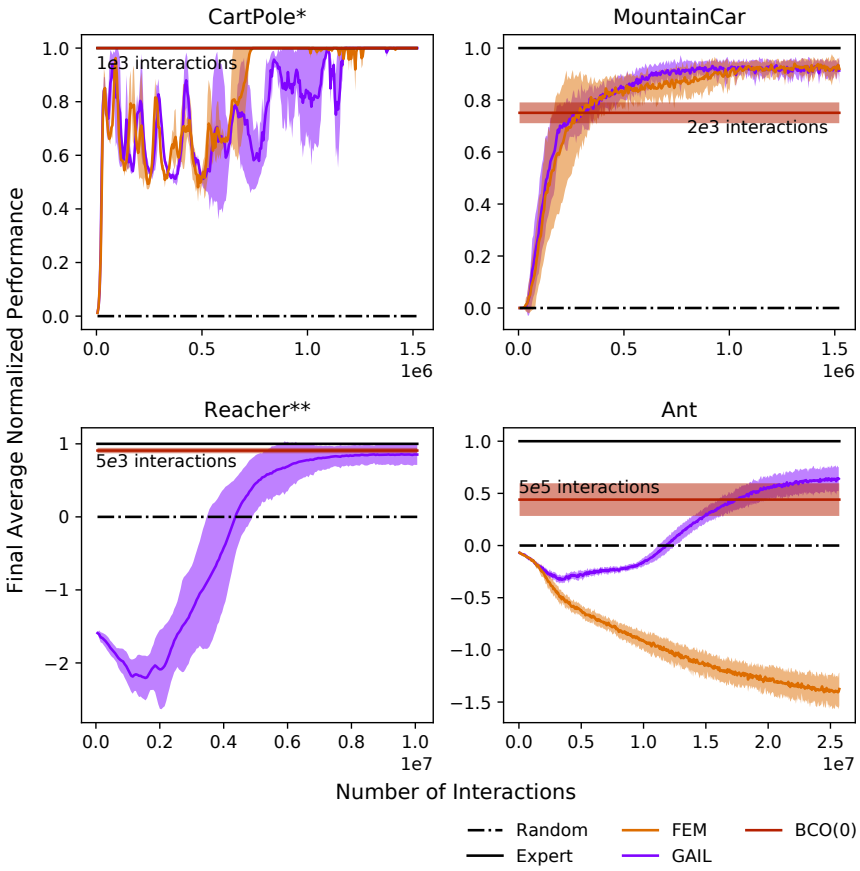

Figure 4.3: Performance of each technique with respect to the number of post-demonstration interactions. For each domain, ten demonstrated trajectories were considered. $BCO(0)$ is depicted as a horizontal line since all environment interactions happen before the demonstration is provided. Performance values are scaled such that performance of a random policy is zero and the performance of the expert is one. Note that *GAIL* and *FEM* have access to demonstration action information whereas *BCO* does not. *The *BCO* line is not visible for the CartPole domain because *BCO* has the same performance as the expert. ***FEM* is not shown for the Reacher domain because its performance is much worse than the other techniques.

.

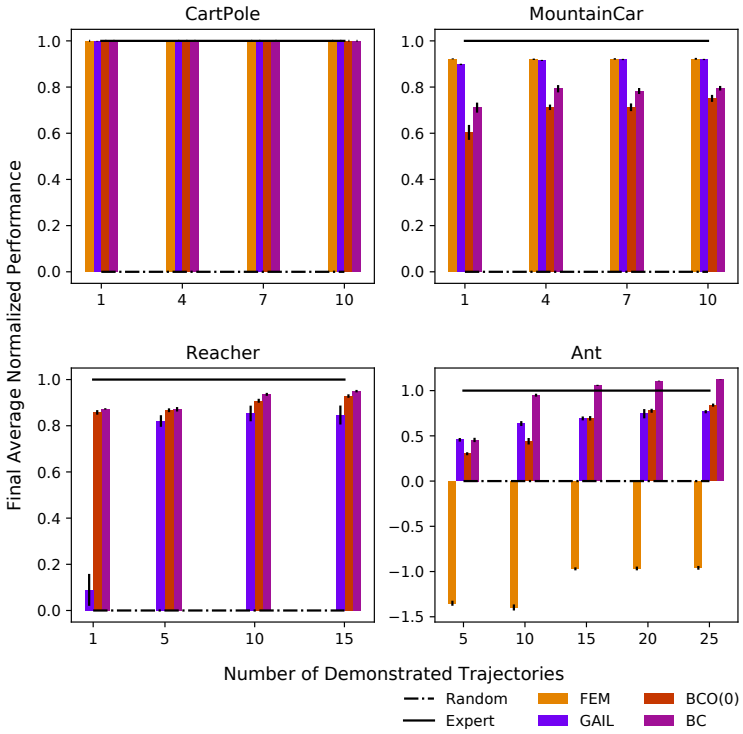

Figure 4.4: Performance of imitation agents with respect to the number of available demonstration trajectories. Rectangular bars and error bars represent the mean return and the standard error, respectively, as measured over 5000 trajectories. Returns have been scaled such that the performance of a random policy and the demonstrating agent are zero and one, respectively. *Note that *FEM* is not shown for the Reacher domain because its performance is much worse than the others. **Note that *BC*, *GAIL*, and *FEM* all have access to demonstration action information whereas *BCO(0) does not.*

so we represent its performance with a horizontal line. The height of the line indicates the performance, and we display the number of provided pre-demonstration environment interactions next to it. The random and expert policies also do not benefit from post-demonstration environment interaction, and so they are also shown using horizontal lines. From these plots, we can

see that it takes at least 40 times more interactions required by *GAIL* or *FEM* to gain the same performance as $BCO(0)$.

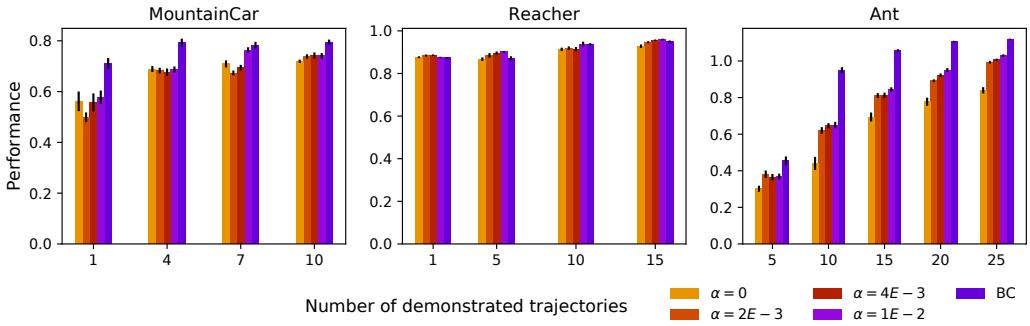

Figure 4.5: The performance of *BC* and several $BCO(\alpha)$ techniques (varying α) with respect to the number of demonstrated trajectories provided. Rectangular bars and error bars represent the mean return and the standard error, respectively, as measured over 5000 trajectories. By increasing α, more post-demonstration environment interactions are allowed to occur, which increases the performance of the imitation policy. Note that *BC* has access to demonstration action information whereas *BCO* does not. Also note that the number of trajectories required for learning a fairly good policy is very small. Each demonstrated trajectory has 5, 50, and 50 transitions for each domain from left to right, respectively. Note that we did not demonstrate the results for CartPole because the results were equally perfect regardless of the value of α.

Now, we compare the performance of our algorithm $BCO(\alpha)$ with the other algorithms. To do so, we use each algorithm to train the agents, and then calculate the final performance by computing the average return over 5000 episodes. For comparison purposes, we scale these performance values such that the performance of the expert and a random policy are 1.0 and 0.0, respectively. This comparison is shown in Figures 4.4, and 4.5 where we have plotted the performance of each algorithm with respect to the number of available demonstrated trajectories. Figure 4.4 shows the comparison between the

Table 4.1: This is an extension to Figure 4.5 which provides the number of post-demonstration interactions for each case. First vertical column for each domain (d) shows the number of demonstrated trajectories. As an example, for the MountainCar domain with 1 demonstrated trajectory and $\alpha = 2E-3$, the average number of post-demonstration interactions is 6825. We can also see at the top of the table that the number of pre-demonstration interactions is $2E3$ so the overall number of interactions would become 8825. It can be seen that almost always by increasing α or the number of demonstrated trajectories, the number of post-demonstration interactions increases. Also the overall number of interactions (combined pre- and post-demonstration interactions) in all the cases is far less than the overall number of interactions required by the other methods (*FEM* and *GAIL*).

MountainCar (pre-demo = $2E3$)				**Reacher (pre-demo = $5E3$)**				**Ant (pre-demo = $5E5$)**			
d\α	**$2E-3$**	**$4E-3$**	**$1E-2$**	**d\α**	**$2E-3$**	**$4E-3$**	**$1E-2$**	**d\α**	**$2E-3$**	**$4E-3$**	**$1E-2$**
1	6825	23475	28950	**1**	210052	358736	912368	**5**	602500	1270000	3362500
4	8387	12000	31200	**5**	270500	486578	1837500	**10**	940000	2075000	5000000
7	6300	23100	122200	**10**	221421	569736	1055921	**15**	1387500	2855000	7325000
10	45462	61450	88600	**15**	509289	852210	1859210	**20**	1925000	4055000	9687500

performance of $BCO(0)$ with all the other methods, and Figure 4.5 compares the performance of $BCO(\alpha)$ across different values of α. In Figure 4.4, we can see that performance of our method is comparable with other methods even though our method is the only one without access to the actions. In the case of Reacher, the transferability of the learned inverse model is highlighted by the high performance of BCO. In the case of the Reacher and Ant domains, we can see that *FEM* performs poorly compared to others, perhaps because the cost functions are not simple enough to be approximated by linear functions. In the CartPole domain, each of the methods performs as well as the expert and so all the lines are over each other. In the case of MountainCar, BCO's performance is worse than the other methods. Conversely, for Reacher,

BCO is more sample efficient than *GAIL*, i.e., with smaller number of demonstrations we get much better results. In the case of Ant, *BCO* performs almost as well as *GAIL*. In Figure 4.5, we can see that *BCO*'s performance improves with larger α since the extra environment interactions allow it to make better estimates of the demonstrator's actions.

4.5 Summary

In this thesis, we have presented *BCO*, an algorithm for performing imitation learning that requires neither access to demonstrator actions nor post-demonstration environment interaction. Our experimental results show that the resulting imitation policies perform favorably compared to those generated by existing imitation learning approaches that *do* require access to demonstrator actions. Moreover, *BCO* requires fewer post-demonstration environment interactions than these other techniques, meaning that a reasonable imitation policy can be executed with less delay.

Chapter 5

Generative Adversarial Imitation from Observation

In this chapter, first, a formulation of a general model-free framework for imitation from observation is proposed. Then, a specific model-free algorithm, Generative Adversarial Imitation from Observation (*GAIfO*), is introduced. This algorithm encourages the imitator's state transition distribution to match that of the demonstrator through adversarial learning. A derivation of this algorithm is presented and, at the end of the chapter, the algorithm is evaluated in multiple MuJoCo domains.

Work in this chapter is based on a paper, *Adversarial Imitation Learning from State-only Demonstrations* [138], published in the Proceedings of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), and another paper, *Generative Adversarial Imitation from Observation* [139], presented in the International Conference on Machine Learning Workshop on Imitation, Intent, and Interaction (I3).

5.1 Overview

Broadly speaking, the *IfO* problem consists of two major subproblems: (1) perception of the demonstrations, i.e., extracting useful features from raw visual data, and (2) learning a control policy using the extracted features. In this chapter, we propose a general framework for the subproblem of *IfO* concerned with learning a control policy in which we characterize the cost as a function of state transitions only. Using this characterization, the *IfO* problem becomes one of trying to recover the state-transition cost function of the expert. Inspired by the work of Ho and Ermon [49], we introduce a novel, model-free algorithm called *Generative Adversarial Imitation from Observation* (*GAIfO*) and prove that it is a specific version of the general framework proposed for *IfO*. We then experimentally evaluate *GAIfO* in high-dimensional MuJoCo environments. Note that, in the settings considered in this chapter, the state space for both the demonstrator and imitator consists of manually-defined features (as opposed to raw visual observations, which will be considered in Chapter 6). We show that the proposed method compares favorably to *BCO* and another recently-developed *IfO* method, and also that it performs comparably to state-of-the-art conventional imitation learning methods that *do* have access the the demonstrator's actions.

The rest of this chapter is organized as follows. The problem setting is presented in Section 5.2. In Section 5.3, we introduce our proposed general framework for *IfO* problems and, in Sections 5.4 and 5.5, we discuss our *IfO* algorithm, *GAIfO*. Then, we describe and discuss our experiments in Sections

5.6 and 5.7, respectively. Finally, we conclude the chapter by summarizing the method and the experiments in Section 5.8.

5.2 Problem Setting

We again operate within the framework of a Markov Decision Processes (MDP), $M = \langle \mathcal{S}, \mathcal{A}, P, c \rangle$ as discussed in Chapter 2. However, since the goal of the work presented in this thesis is that of imitation, the agent does not have access to the cost function. Instead, state-only demonstration trajectories, $D^e = \{\tau_i^e\}$, are available, where $\tau_i^e = \{s_0^e, s_1^e, \cdots, s_N^e\}$. These trajectories are assumed to have been generated using an expert policy π^e. In this chapter, it is also assumed that the demonstrations consist of low-level, hand-crafted state information, and the study of raw visual state observations is deferred to Chapter 6. As for notation, $\overline{\mathbb{R}}$ denotes the extended real numbers $\mathbb{R} \cup \{+\infty\}$, and, as noted in Chapter 2, expectation over a policy means the expectation over all the trajectories that it generates. Finally, the goal here is, first, to formulate a general, model-free approach for imitation from observation, and second, to also develop a specific algorithm of this type inspired by the Generative Adversarial Imitation Learning approach discussed in Chapter 2.

5.3 A General Framework for Imitation from Observation

In *IRL*, both states and actions are available, and the goal is to find a cost function that, on average, results in a smaller cost for the trajecto-

ries generated by the expert policy compared to trajectories generated by any other policy. In the case of imitation from observation, however, the demonstration trajectories that the agent observes contain sequences of states only. In the context of the *IRL*-based approaches to imitation learning discussed in Chapter 2, this lack of action information makes it impossible to calculate the $\mathbb{E}_{\pi^e}[c(s,a)]$ term in (2.14). Consequently, none of the approaches described in Chapter 2 is directly applicable in this setting.

In imitation from observation, the goal is for the actions of the demonstrator and imitator to have the same effect on the environment (performing the task), rather than taking exactly the same actions. Therefore, instead of characterizing the cost signal as a function of states and actions $c : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, we instead define it as a function of the state transitions $c : \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{R}$. Based on this characterization, we formulate Inverse Reinforcement Learning from observation as

$$\begin{aligned} IRLfO_{\psi}(\pi^e) = & \underset{c \in \mathbb{R}^{\mathcal{S} \times \mathcal{S}}}{\operatorname{argmax}} - \psi(c) + (\min_{\pi \in \Pi} \mathbb{E}_{\pi}[c(s,s')]) \\ & - \mathbb{E}_{\pi^e}[c(s,s')] \ , \end{aligned} \tag{5.1}$$

which outputs $\tilde{c}$ (the notation is as discussed in Section 2.2.2). Note that, in (5.1), we ignore the entropy term so as to simplify the theoretical analysis presented in Section 5.4. Evidence form Ho and Ermon [49] suggests that doing so is valid from an empirical perspective (they set $\lambda_H = 0$ in more than 80% of their successful experiments), but we leave detailed analysis of the effect of this choice to future work. From a high-level perspective, the goal is to enable the agent to extract what the task is by observing some state sequences.

Intuitively, this extraction is possible because we expect the beneficial state transitions for any given task to form a low-dimensional manifold within the $\mathcal{S} \times \mathcal{S}$ space. Thus, the intuition behind our definition of the cost function is to penalize based on how close each transition is to that manifold.

Now, using an RL algorithm for $\tilde{c}$ amounts to solving

$$RL(\tilde{c}) = \underset{\pi \in \prod}{\operatorname{argmin}} \, \mathbb{E}_{\pi}[\tilde{c}(s, s')] \, , \tag{5.2}$$

where the output, $\tilde{\pi}$, is the imitation policy. The overall process can be summarized as:

$$\begin{aligned} RL \circ IRLfO_{\psi}(\pi^e) = \underset{\pi \in \prod}{\operatorname{argmin}} \underset{c \in \mathbb{R}^{\mathcal{S} \times \mathcal{S}}}{\operatorname{argmax}} - \psi(c) + (\min_{\pi \in \prod} \mathbb{E}_{\pi}[c(s, s')]) \\ - \mathbb{E}_{\pi^e}[c(s, s')]) \, , \end{aligned} \tag{5.3}$$

5.4 Generative Adversarial Imitation from Observation

Having developed the general framework in Section 5.3, we now propose a specific algorithm for the general framework mentioned in the previous section called Generative Adversarial Imitation from Observation (*GAIfO*). To this end, we first define the state-transition occupancy measure for a specific policy, $\rho^s_{\pi} : \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{R}$ as

$$\rho^s_{\pi}(s_i, s_j) = \sum_a P(s_j | s_i, a) \pi(a | s_i) \sum_{t=0}^{\infty} \gamma^t P(s_t = s_i | \pi) \, . \tag{5.4}$$

This occupancy measure corresponds to the distribution of state transitions that an agent encounters when using policy π. We define the set of valid state-transition occupancy measures as $\mathrm{P}^s \triangleq \{\rho^s_{\pi} : \pi \in \Pi\}$.

We now introduce a proposition which is the foundation of our algorithm. In the following proposition we use the convex conjugate concept which is defined as follows: for a function $f : X \to \overline{\mathbb{R}}$, the convex conjugate $f^* : X^* \to \bar{\mathbb{R}}$ is defined as $f^*(x^*) \triangleq \sup_{x \in X} \langle x^*, x \rangle - f(x)$.

Proposition 5.4.1. *$RL \circ IRLfO_\psi(\pi^e)$ and $\operatorname{argmin}_{\pi \in \Pi} \psi^*(\rho^s_\pi - \rho^s_{\pi^e})$ induce policies that have the same state-transition occupancy measure, ρ^s_π.*

In the rest of this section, we prove this proposition. Then, by choosing a specific regularizer, we present our algorithm. Further, we propose a practical implementation of the algorithm.

To prove the proposition, we first define another problem, $\overline{RL} \circ \overline{IRLfO}_\psi(\pi^e)$, and argue that it outputs a state-transition occupancy measure which is the same as ρ^s_π induced by $RL \circ IRLfO_\psi(\pi^e)$. We define

$$\begin{aligned} \overline{IRLfO}_\psi(\pi^e) = \operatorname*{argmax}_{c \in \mathbb{R}^{\mathcal{S} \times \mathcal{S}}} \min_{\rho^s_\pi \in \mathrm{P}^s} & \sum_{s,s'} \rho^s_\pi(s,s') c(s,s') \\ & - \sum_{s,s'} \rho^s_{\pi^e}(s,s') c(s,s') - \psi(c) \ , \end{aligned} \tag{5.5}$$

where, the output is a cost function $\bar{c}$. Note that $\mathbb{E}_\pi[c(s,s')] = \sum_{s,s'} \rho^s_\pi(s,s') c(s,s')$ which implies that (5.1) and (5.5) are similar except that the former is optimized over $\pi \in \Pi$ and the latter over $\rho^s_\pi \in \mathrm{P}^s$. If we consider using an RL method to find a state-transition occupancy measure under $\bar{c}$, (5.2) can be rewritten as

$$\overline{RL}(\bar{c}) = \min_{\rho^s_\pi \in \mathrm{P}^s} \sum_{s,s'} \rho^s_\pi(s,s') \bar{c}(s,s') \ , \tag{5.6}$$

which would now output the desired state-transition occupancy measure $\bar{\rho}^s_\pi$.

Lemma 5.4.1. *$\overline{RL} \circ \overline{IRLfO}_{\psi}(\pi^e)$ outputs a state-transition occupancy measure, $\bar{\rho}^s_\pi$, which is the same as $\rho^s_{\tilde{\pi}}$ induced by $RL \circ IRLfO_{\psi}(\pi^e)$.*

Proof. From the definition of P^s, the mapping from Π to P^s is surjective, i.e., for every $\rho^s_\pi \in \mathrm{P}^s$, there exists at least one $\pi \in \Pi$. Therefore, we can say $\bar{\rho}^s_\pi = \rho^s_{\tilde{\pi}}$ (where $\tilde{\pi}$ and $\bar{\rho}^s_\pi$, as already defined, are the outputs of (5.2) and (5.6), and $\rho^s_{\tilde{\pi}}$ is the state-transition occupancy measure that corresponds to $\tilde{\pi}$). Therefore, solving $RL \circ IRLfO_{\psi}(\pi^e)$ results in the same ρ^s_π as applying $\overline{RL}$ using the cost function returned by $\overline{IRLfO}$ in (5.5). □

Note that, in Lemma 5.4.1, the returned policies from these two problems are not necessarily the same. The reason is that the mapping from Π to P^s is not injective, i.e., there could be one or multiple $\pi \in \Pi$ that corresponds to the same $\rho^s_\pi \in \mathrm{P}^s$. Consequently, it is not necessarily the case that a policy that gives rise to $\bar{\rho}_\pi$ is the same as $\tilde{\pi}$. However, as we discussed in the previous section, in imitation from observation, we are primarily concerned with the effect of the policy on the environment so this situation is acceptable.

Now we introduce another lemma that helps us in the proof of Proposition 5.4.1.

Lemma 5.4.2. $\overline{RL} \circ \overline{IRLfO}_{\psi}(\pi^e) = \operatorname{argmin}_{\rho^s_\pi \in \mathrm{P}^s} \psi^*(\rho^s_\pi - \rho^s_{\pi^e})$

This lemma is proven in the appendix using the minimax principle [82]. Thus far, by combining Lemmas 5.4.1 and 5.4.2, we can conclude that $\rho^s_{\tilde{\pi}}$

induced by $\text{RL} \circ \text{IRLfO}_\psi(\pi^e)$ is the same as the output of $\operatorname{argmin}_{\rho^s_\pi \in \text{P}^s} \psi^*(\rho^s_\pi - \rho^s_{\pi^e})$. Now, we only need one more step to prove Proposition 5.4.1:

Lemma 5.4.3. $\operatorname{argmin}_{\pi \in \Pi} \psi^*(\rho^s_\pi - \rho^s_{\pi^e})$ *is a policy that has a state-transition occupancy measure that is the same as the output of* $\operatorname{argmin}_{\rho^s_\pi \in \text{P}^s} \psi^*(\rho^s_\pi - \rho^s_{\pi^e})$.

The proof of Lemma 5.4.3 is similar to that of Lemma 5.4.1. Now based on Lemmas 5.4.1, 5.4.2, and 5.4.3 we can conclude that Proposition 5.4.1 holds.

Having proved this proposition, we can solve $\operatorname{argmin}_{\pi \in \Pi} \psi^*(\rho^s_\pi - \rho^s_{\pi^e})$ instead of $\text{RL} \circ \text{IRLfO}_\psi(\pi^e)$. To this end, we consider the generative adversarial regularizer

$$\psi_{GA}(c) \triangleq \begin{cases} \mathbb{E}_{\pi^e}[g(c(s, s'))] & \text{if } c < 0 \\ +\infty & \text{otherwise} \end{cases} \tag{5.7}$$

where

$$g(x) = \begin{cases} -x - \log(1 - e^x) & \text{if } x < 0 \\ +\infty & \text{otherwise} \end{cases} \tag{5.8}$$

which is a closed, proper, convex function and has convex conjugate

$$\begin{aligned} \psi^*_{GA}(\rho^s_\pi - \rho^s_{\pi^e}) = \max_{D \in (0,1)^{\mathcal{S} \times \mathcal{S}}} \sum_{s,s} & \rho^s_\pi(s, s') \log(D(s, s')) + \\ & \rho^s_{\pi^e}(s, s') \log(1 - D(s, s')) \ , \end{aligned} \tag{5.9}$$

where $D : \mathcal{S} \times \mathcal{S} \to (0, 1)$ is a discriminative classifier. A similar convex conjugate is derived by Ho and Ermon [49]; However, for the sake of completeness, we prove the properties claimed for (5.7) and show that (5.9) is its convex conjugate in the appendix.[1]

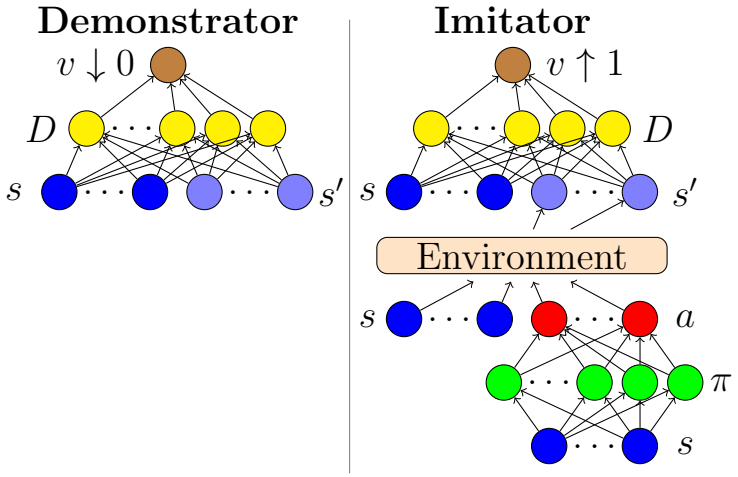

Figure 5.1: A diagrammatic representation of *GAIfO*. On the left, s (dark blue) and s' (light blue) are the state and next-state features in a demonstration transition, respectively. On the right, dark blue neurons represent the imitator's states. Based on policy π (green), it performs action a (red) in the environment, and encounters the next-state s' (light blue). We aim to find a policy that generates state-transitions close to the demonstrations. To this end, we iteratively train the discriminator and the policy. The discriminator is trained in a way to output values v (brown) close to zero for the data coming from the expert (left) and close to one for the data coming from the imitator (right). The policy is trained to generate state-transitions close to the demonstrations so that the discriminator is not able to distinguish them from the demonstrations.

Algorithm 3 *GAIfO*

1: Initialize parametric policy π_ϕ with random ϕ
2: Initialize parametric discriminator D_θ with random θ
3: Obtain state-only expert demonstration trajectories $D^e = \{\tau_i^e\}$ where $\tau_i^e = \{(s^e)\}_i$
4: **while** Policy Improves **do**
5: Execute π_ϕ and store the resulting state transitions $\tau_i^i = \{(s^i)\}_i$
6: Update D_θ using loss

$$-\Big(\mathbb{E}_{\pi_\phi}[\log(D_\theta(s,s'))] + \mathbb{E}_{\pi^e}[\log(1 - D_\theta(s,s'))]\Big)$$

7: Update π_ϕ by performing *TRPO* updates with cost function

$$\Big(\mathbb{E}_{\pi_\phi}[\log(D_\theta(s,s'))]\Big)$$

Using the above, the imitation from observation problem can be solved as:

$$\begin{aligned}\min_{\pi\in\Pi} \psi^*_{GA}(\rho^s_\pi - \rho^s_{\pi^e}) = &\min_{\pi\in\Pi} \max_{D\in(0,1)^{\mathcal{S}\times\mathcal{S}}} \mathbb{E}_\pi[\log(D(s,s'))] + \\ &\mathbb{E}_{\pi^e}[\log(1 - D(s,s'))]\end{aligned} \tag{5.10}$$

We can see that the loss function in (5.10) is similar to the generative adversarial loss. We can connect this to general *GAN*s if we interpret the expert's demonstrations as the real data, and the data coming from the imitator as the generated data. The discriminator seeks to distinguish the source of the data, and the imitator policy (i.e., the generator) seeks to fool the discriminator to make it look like the state transitions it generates are coming from the ex-

[1]This proof closely follows the proofs of Proposition A.1. and Corollary A.1.1. of Ho and Ermon [49] and it is included here for the sake of completeness. The only substantive difference is that in our case we consider state-transition occupancy measure (s, s') instead of (s, a).

pert. The entire process can be interpreted as bringing the distribution of the imitator's state transitions closer to that of the expert. We call this process Generative Adversarial Imitation from Observation (*GAIfO*).

5.5 Practical Implementation

Based on the preceding analysis, we now specify our practical implementation of the *GAIfO* algorithm. We represent the discriminator, D, using a multi-layer perceptron with parameters θ that takes as input a state transition and outputs a value between 0 and 1. We represent the policy, π, using a multi-layer perceptron with parameters ϕ that takes as input a state and outputs an action. We begin by randomly initializing each of these networks, after which the imitator selects an action according to π_ϕ and executes that action. This action leads to a new state, and we feed both this state transition and the entire set of expert state transitions to the discriminator. The discriminator is updated using the Adam optimization algorithm [65], with cross-entropy loss that seeks to push the output for expert state transitions closer to 0 and the imitator's state transitions closer to 1. After the discriminator update, we perform trust region policy optimization (*TRPO*) [115] to improve the policy using a cost function that encourages state transitions that yield small outputs from the discriminator (i.e., those that appear to be from the demonstrator). This process continues until convergence. The algorithm is shown in Algorithm 3 and the framework is summarized in Figure 5.1.

5.6 Experimental Setup and Implementation Details

We evaluate our algorithm in domains from OpenAI Gym [17] based on the Pybullet simulator [25] which are discussed extensively in Section 3.2. In each of the domains, we used Trust Region Policy Optimization (*TRPO*) [115] to train the expert agents, and we recorded the demonstrations using the resulting policy.

The results shown in the figures are the average over ten independent trials. We compare our algorithm against three baselines:

- **Behavioral Cloning from Observation (*BCO*)[136]:** As described in Chapter 4, *BCO* first learns an inverse dynamics model through self-supervised exploration, and then uses that model to infer the missing actions from state-only demonstrated trajectories. *BCO* then uses the inferred actions to learn an imitation policy using conventional behavioral cloning.

- **Time Contrastive Networks (*TCN*)[117]:** *TCN*s use a triplet loss to train a neural network to learn an encoded form of the task at each time step. This loss function brings the states that occur in a small time-window closer together in the embedding space and pushes the ones from distant time-steps far apart. A cost function is then defined as the Euclidean distance between the embedded demonstration and the embedded agent's state at each time step. The imitation policy is learned using *RL* techniques that seek to optimize this cost function.

- **Generative Adversarial Imitation Learning (*GAIL*) [49]:** This method is as specified in Section 2.2.3. Note that, this method has access to the demonstrator's actions while the others do not.

5.7 Results and Discussion

In this section, we present the results of the experiments described above. Figure 5.2 illustrate the comparative performance of *GAIfO* in our experimental domains using the low-dimensional state representations. We can see that, for the domains considered here, *GAIfO (a)* performs very well compared to other *IfO* techniques, and *(b)* is surprisingly comparable to *GAIL* even though *GAIfO* lacks access to explicit action information.

Figure 5.2 compares the final performance of the imitation policies learned by different algorithms. We can clearly see that *GAIfO* outperforms the other imitation from observation algorithms by a large margin in most of the experiments. For the InvertedDoublePendulum domain, we can see that the *TCN* method does not perform well at all. We hypothesize that this is the case because *TCN* relies on time synchronization in order to find the imitating policy, i.e., it learns what the state should be at each time step. However, successfully performing the InvertedDoublePendulum task requires the agent to simply keep the pendulum upright, and requiring it to time synchronize with the demonstrator may be too restrictive a requirement. *BCO*, on the other hand, performs very well in this domain, which demonstrates that, here, the inverse dynamics model learned by *BCO* is accurate and that the compound-

ing error problem is negligible. We can see that *GAIfO* also performs very well here, achieving performance similar to that of the expert, which shows that the algorithm has been able to extract the goal of the task and find a reasonable cost function from which to learn the policy.

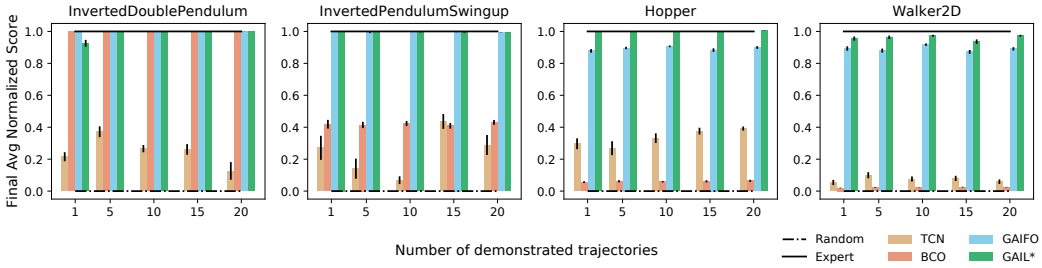

Figure 5.2: Performance of algorithms in low-dimensional experiments with respect to the number of demonstration trajectories. Rectangular bars and error bars represent mean return and standard deviations, respectively. For comparison purposes, we have scaled all the performances such that a random and the expert policy score 0.0 and 1.0, respectively. **GAIL* has access to action information.

For the InvertedPendulumSwingup domain, we can see that *TCN* again does not perform well, perhaps because the goal of the task is not well-represented in the encoding-learning phase. *BCO* also does not perform well. We hypothesize that this is the case because of the compounding error problem since performing this task successfully is contingent on taking several specific actions consecutively – deviation from those actions would cause the pendulum to drop down and not reach the goal. *GAIfO* and *GAIL*, on the other hand, perform as well as the expert, which reveals that these algorithms have successfully extracted the goal and learned the task.

For both the Hopper and Walker2D domains, it can be seen that, again,

TCN does not work well. We posit that this might be due to the fact that these tasks require behavior that is cyclic in nature, i.e., the expert demonstrations contain repeated states. Because *TCN* learns a time-dependent representation of the task, it cannot appropriately handle this periodicity and, therefore, the learned representations are not sufficient. *GAIfO*, however, learns a distribution of the state transitions that is not time-dependent; therefore, periodicity does not affect its performance. *BCO* also does not perform well in either of these two domains, perhaps again due to the compounding error problem. Learning in these domains has two steps: first, the agent needs to learn to stand, and then the agent needs to learn to walk or hop. With *BCO*, it would seem that the imitating agent begins to deviate from the expert early in the task, and this early deviation ultimately leads to the imitating agent being unable to learn the secondary walking and hopping behaviors. *GAIfO*, on the other hand, does not suffer from this issue because it learns by executing its own policy in the environment (on-policy learning) and is therefore able to address deviation from the expert during the learning process.

5.8 Summary

In this chapter, we presented a general framework for imitation from observation ($RL \circ IRLfO_{\psi}(\pi^e)$) and then proposed a specific algorithm (*GAIfO*) for doing so. *GAIfO* removes the need for several restrictive assumptions that are required for some other *IfO* techniques, including the need for multiple demonstrations to be time-synchronized. Moreover, the on-policy nature

of *GAIfO* allows it to avoid the compounding error problem experienced by more brittle imitation techniques. The result is an approach that is able to find better imitation policies without the need for action information, and is also able to find imitation policies that perform very close to those found by techniques that do have access to this information.

Chapter 6

Adversarial Imitation Learning from Video

In this chapter, two extensions of the algorithm presented in Chapter 5 (i.e., *GAIfO*) are introduced. These extensions enable an agent to imitate directly from visual observations. They are evaluated in the Bullet Physics and MuJoCo simulators, and the results of experiments on multiple domains are provided.

Work in this chapter is based on a paper, *Imitation Learning from Video by Leveraging Proprioception* [140], published in the Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019), and another paper, *Generative Adversarial Imitation from Observation* [139], presented in the International Conference on Machine Learning Workshop on Imitation, Intent, and Interaction (I3).

6.1 Overview

As mentioned previously in Section 5.1, the *IfO* problem consists of two major subproblems: (1) perception of the demonstrations, i.e., extracting useful features from raw visual data, and (2) learning a control policy using the extracted features. Most of the previous work in this area is focused on one

of these subproblems. For example, the two algorithms proposed in Chapters 4 and 5 concentrate exclusively on the second subproblem. However, in order to move towards the ultimate goal of *IfO*, i.e., imitating directly from videos, these subproblems should be considered simultaneously, culminating in one algorithm. In this chapter, two extensions of *GAIfO*[139] are presented in which a perception module is incorporated into the algorithm.

The first extension of the algorithm attempts to learn imitation policies by relying solely on self-observation through video, i.e., it uses a convolutional neural network (CNN) that maps images of the agents themselves to actions. In our experiments, we compare this method with other state-of-the-art visual imitation algorithms and show that this approach outperforms them. On the other hand, in many cases, the imitating agent also has access to its own *proprioceptive* state information, i.e., direct knowledge of itself such as the joint angles and torques associate with limbs. We argue that *IfO* algorithms that ignore this information are missing an opportunity that could potentially improve the performance and the efficiency of the learning process. Therefore, another extension of *GAIfO* is proposed as well that can make use of both visual and proprioceptive state information. This extension uses proprioceptive information from the imitating agent during the learning process. We hypothesize that the addition of such information will improve both learning speed and the final performance of the imitator, and we test this hypothesis experimentally in several standard simulation domains. We compare our method with other, state-of-the-art approaches that do not leverage proprio-

ception, and our results validate our hypothesis, i.e., the proposed technique outperforms the others by a large margin.

The rest of this chapter is organized as follows. In Section 6.2, we review technical details surrounding Markov Decision Processes and the *IfO* problem, and explain this Chapter's specific problem of interest. The proposed algorithms are presented in Section 6.3, and we describe the experiments that we have performed in Section 6.4.

6.2 Problem Setting

Similar to previous chapters, we consider artificial reinforcement learning agents that operate in the framework of Markov decision processes (*MDP*s), $M = \langle \mathcal{S}, \mathcal{A}, P, c \rangle$. As in all imitation learning problems, in the setting that we consider in this chapter, the imitating agent does not have access to the cost function, i.e., the agent operates in the context of $M \setminus c$. Instead, it has access to a set of observation-only expert demonstrations $D^e = \{\tau_i^e\} = \{\tau_1^e, \tau_2^e, \cdots\}$ in which τ_i^e is a demonstrated observation-only trajectory $\tau_i^e = \{o_0^e, o_1^e, \cdots, o_N^e\}_i$. In this chapter, we refer to s as the *proprioceptive* state, i.e., s is the most basic, internal state information available to the agent (e.g., the joint angles of a robotic arm). Since we are also concerned with visual observations of agent behavior, we denote these observations as $o \in \mathcal{O}$, i.e., an image of the agent at time t is denoted as o_t. The visual observations of the agent are determined both by the agent's current proprioceptive state s, and also other factors relating to image formation such as camera position. Importantly, due

to phenomena such as occlusion, it is not always possible to infer s from o alone. The goal in this chapter is to modify the *GAIfO* algorithm in a way that enables the algorithm to imitate directly from the set of observation-only expert demonstrations.

6.3 Proposed algorithms

As presented in Section 6.2, we are interested in the problem of imitation from observation (*IfO*), where an imitating agent has access to visual demonstrations, $D^e = \{\tau_i^e\}$, of an expert performing a task, and seeks to learn a behavior that is approximately the same as the expert's. In this section, we introduce the two proposed extensions of *GAIfO* that enable the algorithm to imitate directly from visual data.

6.3.1 Visual *GAIfO* with Self-Observation

As presented in Chapter 5, *GAIfO* is an algorithm that imitates from low-level, state-only expert demonstrations. The algorithm starts with a randomly-initialized multi-layer perceptron (MLP) as the policy and executes that policy to generate states of the imitator's behavior, and then trains a discriminator to differentiate between the states of the demonstrator and the states of the imitator. Then the discriminator is used as a cost function to train the policy. This process is repeated until convergence. Here, in order to incorporate a perception module, instead of using the multi-layer perceptron, we instead use convolutional neural networks to model both the policy and

the discriminator.

Adopting these new models, the imitating CNN policy is randomly initialized, and deployed so as to generate recorded video of the imitator's behavior, $\{\tau_i^i\}$. Then, the CNN discriminator is trained to differentiate between videos of the demonstrator, $\{\tau_i^e\}$, and videos of the imitator, $\{\tau_i^i\}$. *GAIfO* as presented in Chapter 5 is feasible for cases in which *(a)* the states can be assumed to be fully-observable, and *(b)* the system is strictly Markovian. However, when considering visual observations, neither of these assumptions is necessarily valid. Therefore, agents operating in such state spaces are typically provided with a recent state history instead. Providing this information is useful because, for example, having knowledge about the velocity of the agent at each time step is important in order to select the correct action, and velocity information is not available when considering a single image. We adopted this same approach by considering stacks of three frames, $\{o_{t-1}, o_t, o_{t+1}\}$ as the input of the discriminator.

The discriminator is trained with the objective of outputting values that are closer to one for image sequences that come from the imitator, and values that are closer to zero for image sequences that come from the demonstrator. To achieve this goal, the discriminator attempts to solve

$$\max_{D\in(0,1)^{\circ 3}} \Big(\mathbb{E}_{\pi_\theta}[\log(D(o_{t-1} : o_{t+1}))] + \mathbb{E}_{\pi^e}[\log(1 - D(o_{t-1} : o_{t+1}))] \Big) . \tag{6.1}$$

Algorithm 4

1: Initialize policy π_θ (modeled with CNNs) randomly
2: Initialize discriminator D (modeled with CNNs) randomly
3: Obtain visual demonstrations $D^e = \{\tau_i^e\}$ from an expert policy π^e
4: **for** $i \leftarrow 0$ ***to*** N **do**
5: Execute π_θ and record video observation $\{\tau_i^i\}$
6: Update the discriminator D using loss

$$-\Big(\mathbb{E}_{\pi_\theta}[\log(D(o_{t-1} : o_{t+1}))] + \mathbb{E}_{\pi^e}[\log(1 - D(o_{t-1} : o_{t+1}))]\Big)$$

7: Update π_θ by performing PPO updates with gradient steps of

$$\mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|o) Q(s, o)] - \lambda \nabla_\theta H(\pi_\theta),$$

where

$$Q(\hat{o}_t, \hat{a}_t) = - \mathbb{E}_{\pi_\theta}[\log(D(o_{t-1} : o_{t+1}))|o_0 = \hat{o}_t, a_0 = \hat{a}_t]$$

In order for the imitator to learn the task, it should behave similar to the demonstrator. Therefore, it needs to fool the discriminator and so

$$\Big(\mathbb{E}_{\pi_\theta}[\log(D(o_{t-1} : o_{t+1}))]\Big) \tag{6.2}$$

is used as the cost to update the CNN imitation policy using RL. In particular, we use either Trust Region Policy Optimization ($TRPO$) or Proximal Policy Optimization (PPO) [116][1] with gradient steps of

[1]These two RL algorithms were selected based on their performance. In the experiments, we use whichever one results in better performance for a given experimental domain.

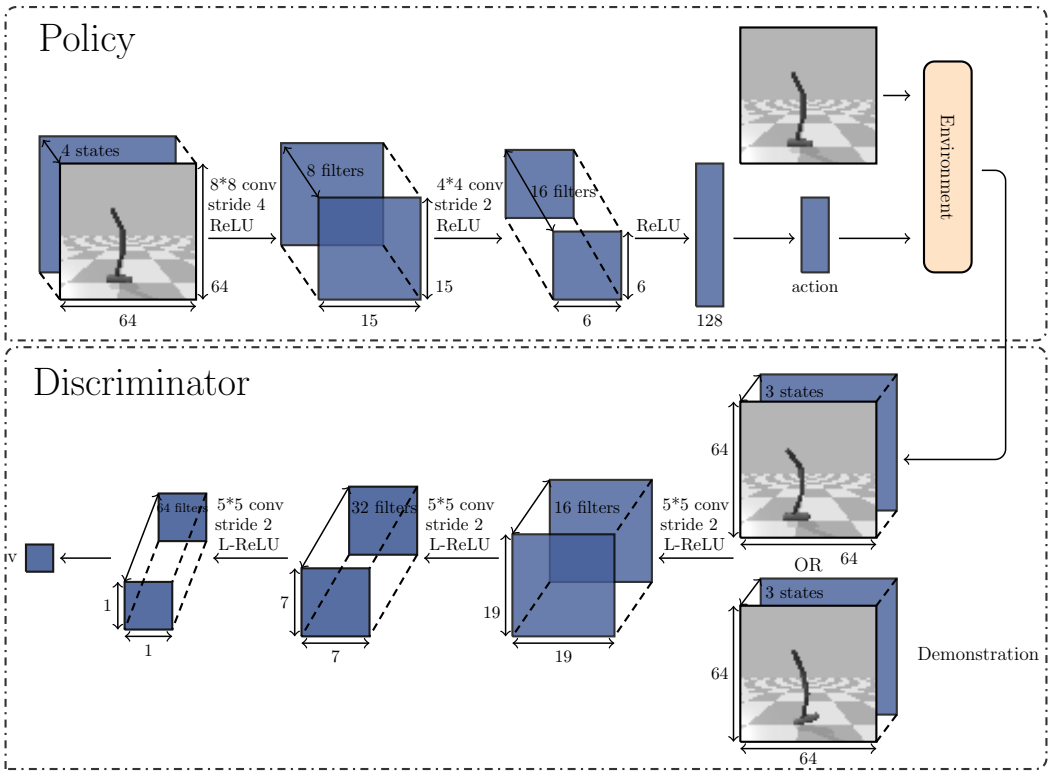

Figure 6.1: A diagrammatic representation of the implementation of our extension of *GAIfO* for processing visual state representations with self-observation. A stack of 4 grayscale images from $t-3$ to t (t being the current time-step) enters the policy *CNN* (top left). The policy outputs an action that the agent takes in the environment and goes to the next state in time $t+1$ (top right). A stack of 3 grayscale images from $t-1$ to $t+1$ of the agent is prepared along with a stack of 3 consecutive state images (grayscale) of the demonstrator (bottom right). When data from the imitation policy is provided, the stack from the imitator enters the discriminator and outputs the reward for taking that action (bottom left). This reward value is then used to both update the policy using *TRPO* and also update the discriminator using supervised learning (to drive the value closer to zero). When data from the demonstrator is provided, the stack from the demonstrator enters the discriminator and outputs a value which is then used to update the discriminator (to drive the value closer to one).

$$\mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|o) Q(o,a)] - \lambda \nabla_\theta H(\pi_\theta), \tag{6.3}$$

where $Q(o,a)$ is the observation-action value, i.e. the potential cost that the agent receives starting from o and taking action a:

$$\begin{aligned} &Q(\hat{o}_t, \hat{a}_t) = \\ &- \mathbb{E}_{\pi_\theta}[\log(D(o_{t-1} : o_{t+1}))|o_0 = \hat{o}_t, a_0 = \hat{a}_t]. \end{aligned} \tag{6.4}$$

The implementation of this algorithm is summarized in Algorithm 4 and Figure 6.1.

6.3.2 Visual *GAIfO* with Proprioceptive Information

We now propose another visual extension of *GAIfO* that leverages the agent's proprioceptive information. The algorithm presented in Section 6.3.1 differs from what is proposed here in that *GAIfO* with self-observation uses visual data *both* in the process of discriminator *and* policy learning. That is, the learned behavior policy maps images o to actions using a convolutional neural network. We hypothesize, however, that also leveraging available proprioceptive state information, s, during the learning process will result in better and faster learning. Therefore, the technique proposed in this section leverages proprioceptive information in the policy learning step, instead learning policies that map proprioceptive states s to actions using a multilayer perceptron architecture.

Inspired by *GAIfO*, this technique comprises two pieces: (1) a generator, which corresponds to the imitation policy, and (2) a discriminator, which

serves as the reward function for the imitator. We model the imitation policy as a multilayer perceptron (MLP), π_θ. The imitating agent, being aware of its own proprioceptive features s, feeds them into the policy network and receives as output a distribution over actions from which the selected action a can be sampled. The imitator then executes this action and we record a video of the resulting behavior. After several actions have been executed, we have accumulated a collection of visual observations of the imitator's behavior, $\{\tau_i^i\}$.

Meanwhile, a convolutional neural network is used as a discriminator D. Given visual observations of the demonstrator, $\{\tau_i^e\}$, and observations of the imitator, $\{\tau_i^i\}$, the discriminator is trained to differentiate between the data coming from these different sources. Since single video frames lack observability in most cases, stacks four frames, $\{o_{t-2}, o_{t-1}, o_t, o_{t+1}\}$ are generated and fed as input to the discriminator.

Similar to *GAIfO* the discriminator is trained to output values closer to zero for the transitions coming from the expert, and values closer to one for those coming from the imitator. Therefore, the discriminator aims to solve the following optimization problem:

$$\max_{D\in(0,1)^{\mathcal{O}^4}} \Big(\mathbb{E}_{\pi_\theta}[\log(D(o_{t-2}:o_{t+1}))]+ \mathbb{E}_{\pi^e}[\log(1-D(o_{t-2}:o_{t+1}))]\Big)\ . \tag{6.5}$$

The lower the value outputted by the discriminator, the higher the chance of the input being from the expert. Recall that the objective for the imitator is to

mimic the demonstrator, which can be thought of as fooling the discriminator. Therefore, we use

$$\Big(\mathbb{E}_{\pi_\theta}[\log(D(o_{t-2}:o_{t+1}))]\Big) \tag{6.6}$$

as the cost to update the imitation policy using *RL*. In particular, we use proximal policy optimization (*PPO*) [116] with gradient steps of

$$\mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q(s,a)] - \lambda \nabla_\theta H(\pi_\theta), \tag{6.7}$$

where $Q(s,a)$ is the state-action value, i.e. the potential reward that the agent receives starting from s and taking action a:

$$\begin{aligned} &Q(\hat{s}_t, \hat{a}_t) = \\ &-\mathbb{E}_{\pi_\theta}[\log(D(o_{t-2}:o_{t+1}))|s_0 = \hat{s}_t, a_0 = \hat{a}_t]. \end{aligned} \tag{6.8}$$

As presented, the proposed algorithm uses the visual information in order to learn the reward function by comparing visual data generated by the imitator and the demonstrator. It also takes advantage of proprioceptive state features in the process of policy learning by learning a mapping from those features to actions using a reinforcement learning algorithm. Pseudocode and a diagrammatic representation of the proposed algorithm are presented in Algorithm 5 and Figure 6.2, respectively.

6.4 Experiments

In this section, we provide the results of the experiments performed for the two extensions of *GAIfO* that handle video demonstrations.

Algorithm 5

1: Initialize policy π_θ randomly
2: Initialize discriminator D randomly
3: Obtain visual demonstrations $D^e = \{\tau_i^e\}$ from an expert policy π^e
4: **for** $i \leftarrow 0$ ***to*** N **do**
5: Execute π_θ and record video observations $\{\tau_i^i\}$
6: Update the discriminator D using loss

$$-\Big(\mathbb{E}_{\pi_\theta}[\log(D(o_{t-2} : o_{t+1}))] + \mathbb{E}_{\pi^e}[\log(1 - D(o_{t-2} : o_{t+1}))]\Big)$$

7: Update π_θ by performing PPO updates with gradient steps of

$$\mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q(s,a)] - \lambda \nabla_\theta H(\pi_\theta),$$

where

$$Q(\hat{s}_t, \hat{a}_t) = - \mathbb{E}_{\pi_\theta}[\log(D(o_{t-2} : o_{t+1}))|s_0 = \hat{s}_t, a_0 = \hat{a}_t]$$

6.4.1 Visual *GAIfO* with Self-observation

The first extension of the *GAIfO* algorithm uses self-observation to enable autonomous agents to imitate from raw visual demonstrations. In this section, we describe the experiments performed to evaluate the proposed algorithm in comparison to other visual imitation algorithms. We hypothesize that the proposed algorithm will outperform the baselines due to the fact that the adversarial framework is able to use a customized, learned cost function at each new iteration.

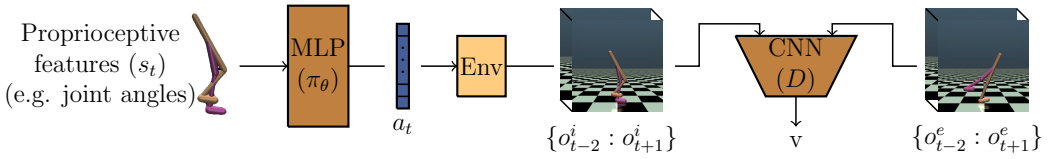

Figure 6.2: A diagrammatic representation of the extension of *GAIfO* algorithm for processing visual demonstration that leverages proprioceptive information. A multilayer perceptron (MLP) is used to model the policy, which takes the proprioceptive features s_t as the input and outputs an action a_t. The agent then executes the action in its environment. While the agent executes the policy, a video of the resulting behavior is recorded. Stacks of four consecutive grayscale images ($o_{t-2} : o_{t+1}$) from both the demonstrator and the imitator are then prepared as the input for the discriminator, which is trained to discriminate between data coming from these two sources. Finally, the discriminator function is then used as the reward function to train the policy using *PPO* (not shown).

6.4.1.1 Setup

The proposed method, *GAIfO* with self-observation, is evaluated on a set of continuous control tasks in BulletPhysics: ReacherBulletEnv, InvertedPendulumBulletEnv, InvertedPendulumSwingupBulletEnv, and HopperBulletEnv. These domains are described in detail in Section 3.2.

In these experiments, like the ones done using the lower-dimensional state representations, the expert is trained with *TRPO* or *PPO* (the one with better performance is selected) using low-level state features and the ground truth reward function provided by Pybullet. Then, 64×64, 30-fps video demonstrations are recorded.

We illustrate the comparison between the performance of this *GAIfO* extension, *BCO* and *TCN*[2]. The *BCO* algorithm is as described in Chapter 4 with the only difference being that observations are used instead of states to learn the inverse dynamics model and the policy and *TCN* is as described in Section 5.6. Furthermore, for a more-representative baseline, we also learn a policy with *TRPO* using visual states only as opposed to the low-dimensional state observations. This line is important in our comparison because it shows (everything being similar to *IfO* methods) what would have been the resulting performance if the agent had access to the reward.

6.4.1.2 Results

In this section, we evaluate our hypothesis that the proposed algorithm outperforms the baselines. Figure 6.3 shows the results of the experiments on the Pybullet domains in which the rectangular bars and error bars represent mean return and standard deviations and quantities 0 and 1 represent the performance of a random agent and the expert, respectively. This figure shows that *GAIfO* with self-observation outperforms other approaches by a large margin.

It is interesting to notice that, even though *GAIfO* with self-observation (like the other *IfO* techniques) does not achieve the performance of the expert agent (solid line), it *does* achieve the performance of the *TRPO*-trained agent

[2]Here, we do not compare against *GAIL* because doing so would require a drastic change to the structure of its discriminator in order to process raw visual data, i.e., the discriminator would need to be altered to appropriately mix action and visual data.

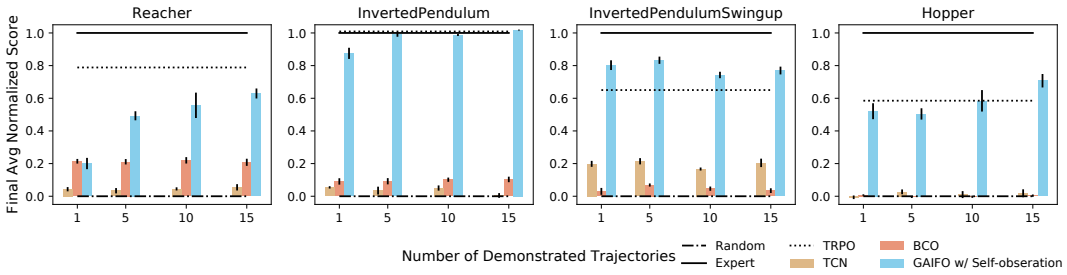

Figure 6.3: Performance of algorithms in visual experiments with respect to the number of demonstration trajectories. Rectangular bars and error bars represent mean return and standard deviations, respectively. For comparison purposes, we have scaled all the performances such that a random and the expert policy score 0.0 and 1.0, respectively.

that used visual state representations. These results suggest that, in these cases, the drop in imitation performance is perhaps due to a fundamental limitation of learning the task from visual data (i.e., partial state observability).

Finally, it can be seen that *BCO* does not perform well in any of the domains, perhaps due to *(a)* the complexity of learning dynamics models over visual states, and *(b)* compounding error. *TCN* also does not work well, perhaps due to the demonstrations not being time-synchronized.

6.4.2 Visual *GAIfO* with Proprioceptive Information

The second extension of the *GAIfO* algorithm combines proprioceptive state information with video observations in an adversarial imitation learning paradigm. We hypothesize that using the extra state information in the proposed way will lead to both faster imitation learning and better performance on the imitated task when compared to similar techniques that ignore propri-

oception. In this section, we describe the experimental procedure by which we evaluated this hypothesis, and discuss the results.

6.4.2.1 Setup

We evaluated our method on a subset of the continuous control tasks available via OpenAI Gym [17] and the MuJoCo simulator [135]: MountainCarContinuous, InvertedPendulum, InvertedDoublePendulum, Hopper, Walker2d, HalfCheetah. These domains are discussed in detail in Section 3.2.

To generate the demonstration data, we first trained an expert agent using pure reinforcement learning (i.e., not from imitation). More specifically, we used Trust Region Policy Optimization (*TRPO*) and Proximal Policy Optimization (*PPO*) [116] with the ground truth reward function provided by OpenAI Gym and selected the best agent for each of the domains. After the expert agents were trained, we recorded 64×64, 30-fps video demonstrations of their behavior.

We compared the proposed method with three other imitation from observation algorithms that do *not* exploit the imitator's proprioceptive state information: **Time Contrastive Networks (*TCN*)** [117], Behavioral Cloning from Observation (*BCO*) [136], and Generative Adversarial Imitation Learning (*GAIfO*) with self-observation.

6.4.2.2 Results

We hypothesized that the visual extension of *GAIfO* that uses proprioceptive information would outperform the baselines with respect to two criteria: (1) the final performance of the trained imitator, i.e., how the imitator performs the task compared to the demonstrator (as measured by the ground truth reward functions), and (2) the speed of the imitation learning process as measured by number of learning iterations. The results shown here were generated using ten independent trials, where each trial used a different random seed to initialize the environments and model parameters.

Figure 6.4 depicts our experimental results pertaining to the first criterion, i.e., the final task performance of trained imitating agents in each domain. The rectangular bars and error bars represent the mean return and the standard error, respectively, as measured over 1000 trajectories. We report performance using a normalized task score, i.e., scores are scaled in such a way that the demonstrating agent's performance corresponds to 1.0 and the performance of an agent with random behavior corresponds to 0.0. The x-axis represents the number of demonstration trajectories, i.e., videos, available to the imitator. In general, it can be seen that the proposed method indeed outperforms the baselines in almost all cases, which shows that using the available proprioceptive state information can make a remarkable difference in the final task performance achieved by imitation learning. In the particular case of InvertedPendulum, both *GAIfO* with self-observation and *GAIfO* with proprioceptive information achieve a final task performance equal to that of the

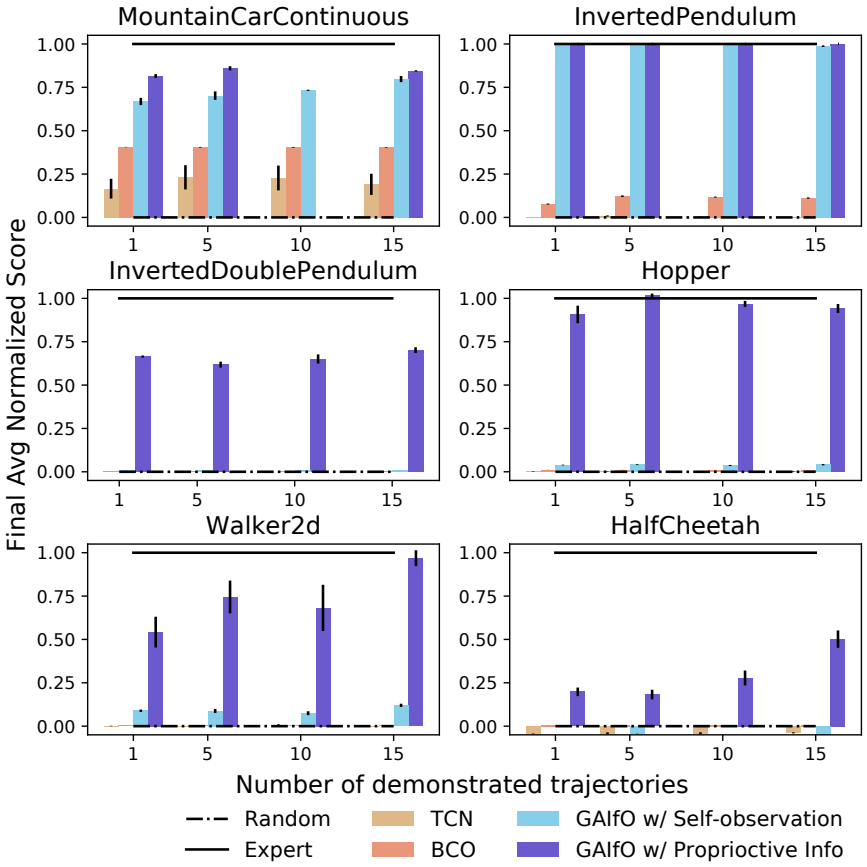

Figure 6.4: Performance comparison of visual *IfO* algorithms presented in this chapter. The rectangular bars and error bars represent the mean normalized return and the standard error, respectively, as measured over 1000 trials. The normalized values have been scaled in such a way that expert and random performance are 1.0 and 0.0, respectively. The x-axis represents the number of available video demonstration trajectories.

demonstrator, likely due to the simplicity of the task. However, for the rest of the tasks, it can be clearly seen that *GAIfO* with proprioceptive information performs better than *GAIfO* with self-observation.[3] Further, we can see that

[3]Note that the performance of *GAIfO* with self-observation on Hopper is different from

increasing the number of demonstrated trajectories results in increased task performance.

To validate our hypothesis with respect to learning speed, we also studied the transient performance of the various learning algorithms. Because only one other method, *GAIfO* with self-observation, performed as well as the expert in only one domain, InvertedPendulum, Figure 6.5 only depicts the results for these algorithms in that domain. The x-axis shows the number of iterations, i.e., the number of update cycles for both the policy and the discriminator. Since updating the policy requires interaction with the environment, a smaller number of iterations also corresponds to less overhead during the learning process. As shown in the figure, *GAIfO* with proprioceptive information converges to expert-level performance much faster than *GAIfO* with self-observation, which supports our hypothesis that leveraging proprioception speeds the imitation learning process.

In Figure 6.4, we can see that two of the baseline methods—*BCO* and *TCN*—do not achieve task performance anywhere near that of the expert.

For InvertedPendulum and InvertedDoublePendulum, we suspect that *TCN* performs poorly due to possible overfitting of the learned state embedding to the specific demonstrations and, therefore, does not generalize well

what was presented in the previous section. We hypothesize that the reason is twofold: (1) different physics engines—MuJoCo is used in this paper, but in the previous section Pybullet [25] was used, and (2) differences in video appearance—in this work we do not alter the default simulator parameters, whereas in the previous section some of the parameters were modified such as the colors used in the video frames in order to increase the contrast between the agent and the background.

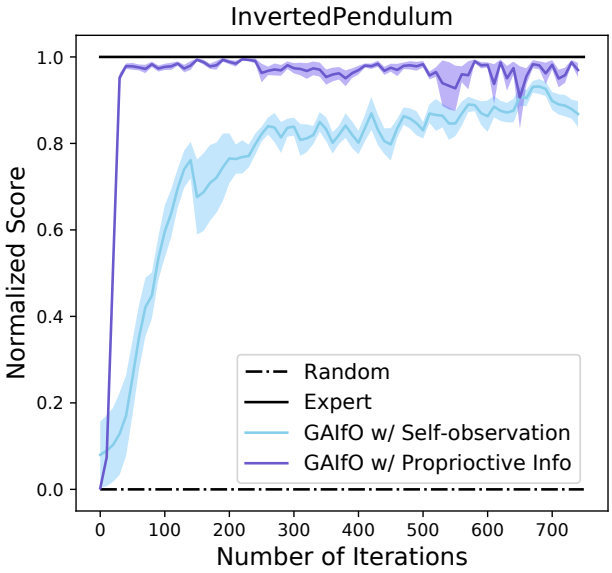

Figure 6.5: Performance of imitation agents with respect to the number of iterations (N). Solid colored lines represent the mean return and shaded areas represent standard errors. The returns are scaled so that the performance of the expert and random policies are zero and one, respectively.

toward supporting the overall goal of keeping the pendulum balanced above the rod. For Hopper, Walker2d, and HalfCheetah, the poor performance of *TCN* may be due to the fact that the tasks are cyclical in nature and therefore not well-suited to the time-dependent learned state embedding. *TCN* performs relatively better in MountainCarContinuous, compared to other domains because this domain does have the properties required by *TCN*. As for *BCO*, we posit that the low performance is due to the well-known compounding-error issue present in behavioral cloning.

One interesting thing to note is that results in Walker2d have larger error bars for our technique than those seen for any of the other domains.

We hypothesize that the reason for this is that the video frames provide very poor information regarding the state of the demonstrator—here, the agent has two legs, which sometimes results in occlusion and, therefore, uncertainty regarding which action the agent should take.

Finally, we can see that *GAIfO* with proprioceptive information performs the most poorly in the HalfCheetah domain. We hypothesize that this is due to the speed at which the demonstrator acts: frame-to-frame differences are large, e.g., three to four consecutive frames cover a complete cycle of the agent jumping forwards. This rate of change may make it difficult for our discriminator to extract a pattern of behavior, which, consequently, would make it much more difficult for the agent to move its behavior closer to that of the demonstrator. Therefore, one way that performance might be improved is to increase the frame rate at which the demonstrations are sampled. Another way, as suggested by Figure 6.4, would be to increase the number of demonstration trajectories beyond what is shown here.

6.5 Summary

In this chapter, we proposed two extensions for *GAIfO* that allow autonomous agents to imitate from visual demonstrations. The first algorithm involved using convolutional neural networks two model both the policy and the discriminator. We hypothesized that this algorithm would outperform other algorithms with the same ability and verified this hypothesis on continuous control domains in Pybullet. The second algorithm was another extension

of *GAIfO* that leverages both available visual and proprioceptive information and we hypothesized that including proprioception would be beneficial to the learning process in the *IfO* paradigm. This algorithm uses visual information to compare the imitator's behavior to that of the demonstrator, and uses this comparison as a cost function for training a policy over proprioceptive states. We showed that leveraging this state information can significantly improve both the performance and the efficiency of the learning process.

Chapter 7

Reinforced Inverse Dynamics Modeling

In this chapter, an algorithm called reinforced inverse dynamics modeling (*RIDM*) is introduced that is model-based – similar to *BCO* – and combines imitation learning from observation and reinforcement learning in order to take advantage of the strengths of both frameworks. This algorithm requires a relatively small number of environment interactions and works with only one state-only demonstration. *RIDM* is thoroughly evaluated in the MuJoCo simulator [135], in the 3D SimSpark simulator [13, 152], and on a UR5 robot.

Work in this chapter is based on a paper, RIDM*: Reinforced Inverse Dynamics Modeling for Learning from a Single Observed Demonstration* [95] in collaboration with Brahma Pavse and Josiah Hanna, published in IEEE Robotics and Automation Letter and presented in 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2020), and another paper, *UT Austin Villa: RoboCup 2019 3D Simulation League Competition and Technical Challenge Champions* [79] in collaboration with Patrick MacAlpine and Brahma Pavse, published in Robot World Cup 2019.

7.1 Overview

Two of the most prevalent paradigms for behavior learning in artificial agents are imitation learning (*IL*) [112, 3] and reinforcement learning (*RL*) [128]. Agents that use imitation learning receive a strong training signal in the form of an expert demonstration, but because their goal is to imitate, their task performance is typically bounded above by that of the expert. Agents using reinforcement learning, on the other hand, can theoretically learn behaviors that are optimal with respect to a predefined task cost, but often have difficulty doing so due to practical challenges such as large state spaces and sparse cost functions. Because of the relative advantages and disadvantages of each of these paradigms, it is natural to investigate whether one can integrate them in order to get the best of both methods.

While combining imitation learning and reinforcement learning has been explored to a certain extent in the literature [132, 70, 161], several important issues remain. Most importantly, these techniques require access to the actions used by a demonstrator in order to be able to leverage the demonstration information [49, 161, 48], meaning that imitation from observation has not been explored in this setting. A second limitation of many existing techniques is the requirement for *many* expert demonstrations [14], which makes obtaining sufficient demonstration data difficult in that it requires a high level of access to expert demonstrators. Finally, existing methods typically assume that they have access to *task-specific* state features during the learning process that can be used to make learning easier [49, 136, 139]. Task-

specific state features are ones that somehow augment the agent's natural (or *raw*) state information using additional domain knowledge—like the distance to certain important subgoals—designed to make cost function representation easier (see, e.g., Figure 7.2). While providing this domain knowledge may be fairly easy for a specific task, it will, in general, need to be specified anew for each new task encountered and therefore represents a practical impediment to using existing methods.

In this chapter, a new technique for integrating imitation learning from observation and reinforcement learning called *reinforced inverse dynamics modeling (RIDM)* is introduced that bypasses the issues identified above. *RIDM* leverages recent ideas from model-based imitation from observation (*IfO*) to enable integrated imitation and reinforcement learning from a *single, action-free* demonstration consisting of only *raw states*. Moreover, *RIDM* represents a new paradigm for combining imitation learning and reinforcement learning in that the agent's behavior is based on following a fixed demonstration trajectory using a parameterized task-specific inverse dynamics model (IDM) (see, Figure 7.1). A task-specific IDM is one that maps state-transitions to actions for a specific task only and may not generalize to other tasks that have different cost functions. While *RIDM* requires the demonstration trajectory during execution, its overall objective is *not* to imitate, but rather to minimize the external environment cost. *RIDM* accomplishes the cost minimization by using reinforcement learning to tune the IDM that *attempts* to follow the fixed demonstration such that the resulting behavior leads to the

lowest environmental cost. Formulating the overall reinforcement learning problem in this way allows *RIDM* to diverge from the demonstration if doing so will lead to lower task cost, which is helpful when the demonstration is sub-optimal. To the best of our knowledge, *RIDM* is the first algorithm that combines *IfO* and *RL*.

To evaluate our algorithm, a baseline algorithm is established by modifying the algorithm presented in Chapter 5, *GAIfO*, to incorporate an external cost signal. *RIDM* is expected to be able to outperform this baseline in the problem of interest where one, raw-state demonstration is provided. Several quantitative experiments focused on both simulated and real robot control tasks are performed, and it is shown that *RIDM*'s unique, model-driven approach results in high-quality behavior trajectories that lead to better performance than the baseline.

7.2 Problem Setting

In this chapter, an algorithm is proposed that integrates reinforcement learning and imitation from observation. The idea is to combine both approaches in a manner that can take advantage of the strengths of each.

The agent-environment interaction is modeled as an *MDP*, denoted by $M = \langle \mathcal{S}, \mathcal{A}, P, c \rangle$. As noted in Chapter 2, $\mathcal{S}$ is the state space of the agent, $\mathcal{A}$ is the action space of the agent, P defines the environment transition function that gives the probability of the agent moving from one state to another given that the agent took a particular action, and c is the scalar-

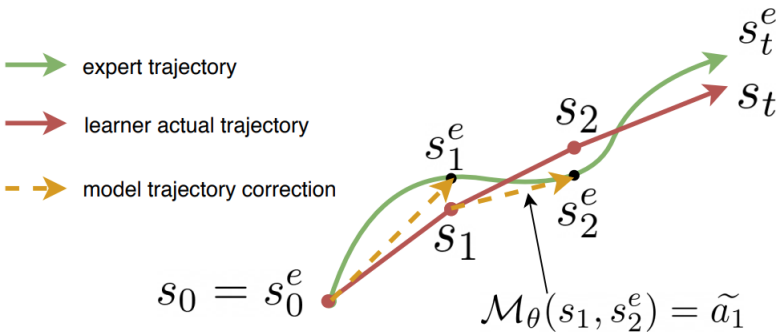

Figure 7.1: *RIDM* uses a task-specific inverse dynamics model, $\mathcal{M}_\theta$, to trasnform the learner's current state s_t and the demonstrator's next state s^e_{t+1}, such that the sequence of executed actions, $\{a_t\}$, minimizes the cumulative cost from the environment. At each time step, the agent uses the demonstrator's next state, s^e_{t+1} (black dot), as the set point for $\mathcal{M}_\theta$. However, the agent actually reaches s_{t+1} (red dot) instead—which is typically *not* the set point—since *RIDM* optimizes $\mathcal{M}_\theta$ to *minimize environment task cost* instead of trajectory-tracking error.

valued cost function that dictates the cost incurred by the agent when moving from one state to another via a particular action. In the context of the problem setting considered in this chapter, the agents have access to the cost function c and also a single, state-only demonstration $\tau^e = \{s^e_0, s^e_1, \cdots, s^e_N\}$. The agent's goal is to find a policy that it can use to select actions a_t to take at each time step that allow it follow the demonstration as closely as possible while also minimizing the cost.

7.3 Reinforced Inverse Dynamics Modeling

Reinforced inverse dynamics modeling (*RIDM*) – a new method for integrating *IfO* and *RL* – is now introduced. *RIDM* learns a strategy by

which an agent can select actions $\{a_t\}$ that allow it to achieve a high level of task performance when it has available a single, state-only demonstration, $D^e = \{s_t^e\}$ and a cost function, C_{env}.

RIDM finds such a strategy by learning and using a task-specific inverse dynamics model (IDM), $\mathcal{M}_\theta$, that computes which action to take at any given time instant based on both the agent's current state and a desired next state, the *set point*. Under *RIDM*, the agent's actions are computed as $a_t = \mathcal{M}_\theta(s_t, s_{t+1}^e)$, where s_t is the learner's current state and s_{t+1}^e is the state of the demonstrator at the next time instant. The goal of *RIDM* is to find an optimal θ such that the generated action sequence minimizes the cumulative cost from the environment, $C_{env}(\mathcal{M}_\theta)$. That is, while *RIDM* selects which actions to take by using the demonstrator's state sequence as a sequence of set points, it evaluates its policy in relation to the *environmental cost* as opposed to, e.g., the trajectory-tracking error. Note that it may actually be desirable for the induced state sequence to differ from that of the demonstrator's if this difference ends up leading to a lower environment cost. Figure 7.1 depicts this process. To the best of our knowledge, using such a scheme to perform integrated *IfO* and *RL* is unique in the literature.

RIDM consists of two phases: *(1)* the IDM initialization, and *(2)* the IDM reinforcement. The initialization of the IDM can either be done by selecting θ at random, or – if a known policy is available to the learner – by having the agent generate its own set of state-action-next-state triples and using supervised learning to fit θ to those triples. In the second phase, *RIDM* alternates

between generating agent behavior according to θ and the demonstration, and optimizing θ in response to the amount of environment cost obtained by the generated behavior. During this phase, the learner uses the demonstration to guide the agent's behavior (i.e., imitation), but uses the observed environment cost to adjust θ such that actions leading to low costs are generated (i.e., reinforcement). The goal of this two-phase procedure is to find the optimal policy in terms of total task cost (which may outperform the demonstrator) by using the demonstration as a guide. The pseudo-code for *RIDM* is given in Algorithm 6, and each phase is described in more detail below[1].

Algorithm 6 RIDM

Require: Single, state-only demonstration $D^e := \{s_t^e\}$
if π^{pre} available **then**
    Generate $D^{pre} := \{(s_t^{pre}, a_t^{pre})\}$ using π^{pre}
    Initialize θ as the solution to (7.1)
else
    Initialize θ uniformly at random
while θ not converged **do**
    for $t = 0 : |D^e| - 1$ **do**
        $a_t := \mathcal{M}_\theta(s_t, s_{t+1}^e)$
        Execute a_t and record s_{t+1} and cost $c(s_t, a_t)$
    Compute cumulative episode cost $C_{env} = \sum_t c(s_t, a_t)$
    Update θ by solving (7.2)
return θ^*

[1]Even though *RIDM* is described as a method that requires both environment costs and state-only demonstrations, the algorithm can be used even if the cost is not available for instance by defining the cost as the distance between the demonstrated state and the imitator's state at each time step.

7.3.1 Inverse Dynamics Model Pre-training

During *RIDM*'s optional first phase, an initial value for θ is sought. This initialization is accomplished either through the use of data collected by the learner using a pre-defined exploration policy or, if such a policy is not available, by selecting the parameter value at random. We allow for *RIDM* to take advantage of an available exploration policy so that it can achieve a reasonable level of task performance, which is likely to get us into a good basin of attraction within the optimization landscape. From a high level point of view, this step is similar to inverse dynamics model learning step of *BCO* presented in Section **??** in that in both cases, they are pre-training the IDM. However, they are different in the sense that here, this step is only happening if a reasonable exploration policy that could perform relatively well is available. In *BCO* , on the other hand, the exploration policy was considered to be random.

In the case where an exploration policy π^{pre} is available (e.g. if a slow-walk policy is available and we want the agent to learn a fast walk), *RIDM* computes an initial value for θ as follows. First, the learner executes π^{pre} in the environment and records the resulting experience as a trajectory of length T that we denote as $D^{pre} = \{(s_t^{pre}, a_t^{pre}, s_{t+1}^{pre})\}$. The initial value for θ is then computed by solving the following supervised learning problem:

$$\theta^* = \arg max \left(-\frac{1}{T} \sum_{t=1}^{T} \sum_{n=1}^{N} \frac{|\mathcal{M}_\theta(s_t^{pre}, s_{t+1}^{pre})_n - a_{tn}^{pre}|}{\max(a_n^{pre}) - \min(a_n^{pre})} \right), \tag{7.1}$$

where N is the dimensionality of the action space, a_{tn}^{pre} denotes the scalar value of the n^{th} component of the action vector a_t^{pre}, and $\max(a_n^{pre})$ denotes

the maximum value of a_{tn}^{pre} across all t. Above, notice that the goal of the optimization problem is to select θ such that $\mathcal{M}_\theta(s_t^{pre}, s_{t+1}^{pre})_n$ is a good approximation of the true action value a_{tn}^{pre}. We adopt the particular loss given above because we found that it worked well in practice. It is able to effectively trade off short-term errors in order to optimize the differences across a full trajectory, and the normalization term ensures greater accuracy for actions which vary over a smaller range. *RIDM* solves (7.1) using a blackbox optimization technique (e.g., CMA-ES [46]). Note, however, that this pre-training phase is optional, and only possible when *RIDM* has access to an exploration policy that generates a behavior that is qualitatively similar to the desired end behavior.

7.3.2 Inverse Dynamics Model Reinforcement

RIDM's required second phase seeks to iteratively update the inverse dynamics model parameters in response to the environment return. The process executed here is illustrated in Figure 7.1, where one can see that *RIDM* uses the demonstration as a behavior template in the sense that the demonstrator's state trajectory is used as a sequence of set points.

The iterative updates to θ are computed as follows. First, the learner uses $\mathcal{M}_\theta$ and the demonstration to generate a trajectory of experience. It does so by, when in state s_t at time step t, executing action $a_t = \mathcal{M}_\theta(s_t, s_{t+1}^e)$, which results in a transition to state s_{t+1} and the observation of cost $c(s_t, a_t)$. After an entire trajectory has been generated, the learner computes the cumulative

environment cost $C_{env}(D^e\ ;\ \theta) = \sum_t c(s_t, a_t)$, which is dependent on both the (fixed) demonstration data D^e and the (tunable) model parameters θ. In a given iteration, i, an update to θ is computed as the solution to:

$$\theta_i = \arg\max C_{env}(D^e\ ;\ \theta_{i-1})\ . \tag{7.2}$$

It is important to note that, here, the demonstrator's actions are *unknown*. While $C_{env}(D^e\ ;\ \theta)$ is used to reinforce the learning of the inverse dynamics model parameters, the learner is always guided by the same, fixed, state-only demonstration trajectory.

For each iteration of the above procedure, *RIDM* solves (7.2) again using a blackbox optimization technique (eg., CMA-ES [46] or Bayesian optimization [96]).

7.4 Empirical Results

We now empirically validate our hypothesis, i.e., that behaviors learned using *RIDM* will outperform those learned by the established baseline, i.e., a modified version of *GAIfO*. We focus on the case in which only a single, state-only demonstration is available to the agent and when no task-specific state augmentation can be performed. Our experiments are executed in multiple robot control domains: simulated tasks are carried out in the MuJoCo and SimSpark simulators, and several manipulation tasks are carried out on a UR5 robot arm. These environments are extensively discussed in Chapter 3.

In the first set of experiments with the MuJoCo simulator, neural net-

works are used to model IDMs since other literature has shown that such models are effective in the corresponding domains. The selected neural network IDM receives a single (state, next-state) pair as input, and outputs the parameters of a Gaussian distribution from which an action is to be sampled and executed by the agent. For the experiments in the SimSpark simulator and the real world, PID controllers are used as IDMs since, again, such controllers have proven effective in these domains in the past[2].

During training, the next state (or set point) at each time step is fixed according to the demonstration, and *RIDM* uses reinforcement learning to tune the IDM parameters (either NN parameters or PID gains) such that the overall agent behavior can best minimize the external environment cost. *RIDM* uses CMA-ES[46] in all the simulated experiments, but uses Bayesian optimization[96] in the physical robot experiments to learn the inverse dynamics model due to its superior sample complexity.

This rest of this section is organized as follows. First, we provide experimental motivation for studying *RIDM* in the single, state-only demonstration and raw state space case. Next, we establish a reasonable *IfO+RL* baseline that can also operate in this regime, and we validate our hypothesis by comparing *RIDM* to this baseline. In each of our evaluations, we scale the reported performance metrics such that a score of 0 corresponds to the behavior of a

[2]PID controllers are not exactly inverse dynamics models due to their differential and integral terms. However, they retain the essential characteristic that *RIDM* requires, i.e., that, given a current state and desired next state (or set point), they will generate an action that attempts to reach the set point.

random policy, and a score of 1 corresponds to the behavior of the demonstrator. Note that, when the demonstrator performance is sub-optimal, because we are combining imitation learning with reinforcement learning, it is reasonable to expect that the algorithm will outperform the demonstrator on some tasks. Finally, we conclude this section by reporting additional empirical results for applying *RIDM* to both simulated robot soccer skill learning and to learning to perform a behavior on a physical UR5 arm robot.

7.4.1 Raw vs. Augmented State

Prior research has established that the availability of demonstrator action information and many demonstration trajectories are critical for the success of existing imitation learning algorithms [136, 139]. While *RIDM* is advantageous in that it does not depend on the availability of the above information, we have also claimed that it can operate directly on raw state information that has not been augmented using extra knowledge of the task which, of course, is typically unknown or difficult to obtain. We now seek to experimentally motivate the need for overcoming this issue by showing the level of reliance on these augmented state spaces in many existing imitation and reinforcement learning algorithms.

In Figure 7.2, we compare the scaled performance of an imitation from observation algorithm (*GAIfO* [139] introduced in Chapter 5) and a reinforcement learning algorithm (*TRPO* [115]) when they are given access to an augmented state space vs. when they are exposed to only the raw state space on

six tasks from the MuJoCo domain [135][3]. Here, the raw state space refers to the list of joint angles, and the augmented state space also includes task-specific information. For instance, in the Hopper task, the augmented state space includes the agent's global position which is advantageous in that it is more highly correlated with the cost signal (see, e.g., [139, 140, 49, 115, 116]). In this specific task, the agent's goal is that of controlling the limbs of a 2D, one-legged robot such that it moves forward as fast as possible. The task cost given per time step corresponds to the change in global position of the agent, and since this information appears in the augmented state information, both learning algorithms perform much better (except for *GAIfO* in the Ant domain, perhaps due to the very large (111 dimensional) augmented state space). This advantage can be seen in Figure 7.2, where the high reliance of *GAIfO*[139] and *TRPO*[115] on the augmented state space is readily apparent. Designing state spaces is, in general, a relatively hard task. The difficulty arises in that, to get the best performance possible, the state space should have just the right amount of information, which often requires a significant amount of domain knowledge and, therefore, our preference here is to remove this requirement.

Because we seek to remove the above restriction, we use only the joint angles as the raw state information in our experiments. Many robots are composed of joints, and therefore a joint-only representation is reasonably task-

[3]The MuJoCo experiments use all the standard settings, e.g., the cost functions, the goals, and the augmented state spaces, as defined in the MuJoCo code base.

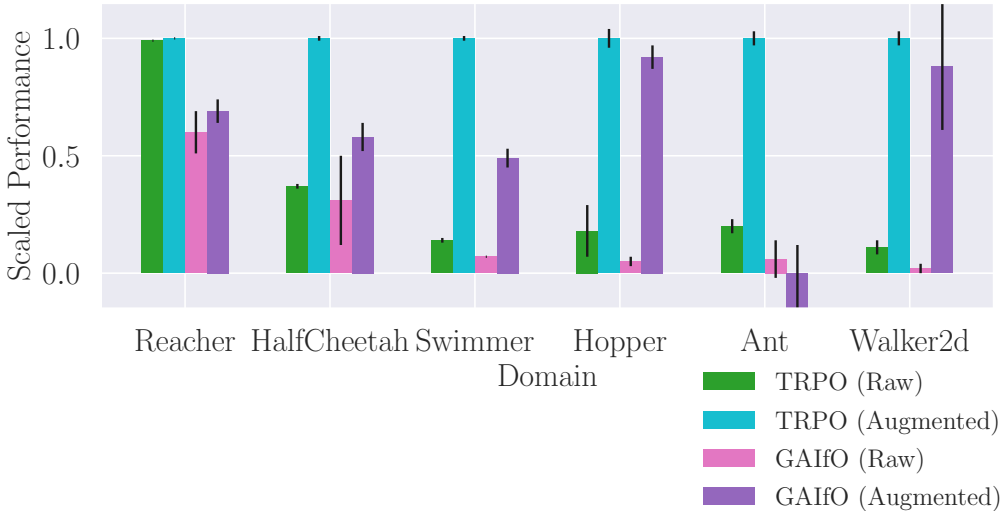

Figure 7.2: Quantitative exhibition of the importance of an augmented state space for high performance on six MuJoCo domains for *GAIfO* and *TRPO*. Mean and standard deviations are over 100 policy runs. The results are brought here as we seek the *RIDM* to operate over raw state space and to show how challenging it is to learn when raw state space is considered compared to augmented space. Both methods use a neural network parameterized policy.

independent. The core results of our algorithm are exclusively concerned with dealing with the above case, i.e., single state-only demonstration consisting of joint angles.

7.4.2 *RIDM* Applied to MuJoCo Simulation

We now present our core results. This section is organized as follows. Section 7.4.2.1 proposes a reasonable baseline to compare against our method. Section 7.4.2.2 presents the performance of our algorithm against this baseline.

7.4.2.1 Baseline: *GAIfO+RL*

To the best of our knowledge, there is no method in the existing literature which can operate in the experimental setting of interest. Therefore, in order to understand the effectiveness of *RIDM*, we first introduce a natural combination of the best existing algorithms for the components of *RIDM*, namely *IfO* and *RL* as the baseline, which we refer to as *GAIfO+RL*and then compare the performance of *RIDM* against this algorithm in MuJoCo environments[4].

GAIfO+RL is based on the current state-of-the-art imitation from observation algorithm, *GAIfO* [139]. Starting from *GAIfO*, *GAIfO+RL* integrates *RL* by modifying the cost function used during the agent update step. Instead of the cost function being determined solely by the discriminator as in *GAIfO*, *GAIfO+RL* integrates imitation and reinforcement learning by defining a new cost function that is a linear combination of the discriminator's output and the task cost [161].

In Figure 7.3, we establish that *GAIfO+RL* is a strong baseline by evaluating the performance of *GAIfO* alone, *RL* alone (*TRPO*/*PPO*), and *GAIfO+RL*. All three methods operate in the raw (un-augmented) state space, and *GAIfO* and *GAIfO+RL* are also given access to a single, state-only demonstration. While the performance of either pure *IfO* or pure *RL* alone is

[4]The baseline algorithm's high sample complexity prevents us from using it on SimSpark simulator and UR5 and therefore, this algorithm is only considered as the baseline for MuJoCo experiments

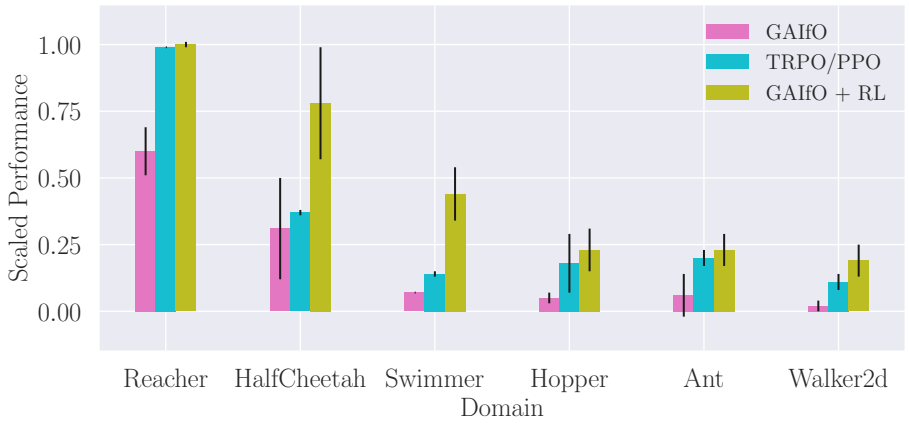

Figure 7.3: Establishment of *GAIfO+RL* as a reasonable *IfO+RL* baseline to compare against *RIDM*. All methods use the same single state-only demonstration consisting of only raw states (exclusively of joint angles). Mean and standard deviations are over 100 policy runs. All methods use a neural network parameterized policy.

relatively poor, we can see that *GAIfO+RL* achieves significantly higher performance than its parts. Moreover, *GAIfO+RL* operates in the same established regime and belongs to the same class of *IfO+RL* algorithms as *RIDM*, and therefore seems to be a reasonable imitation from observation + reinforcement learning algorithm.

7.4.2.2 Hypothesis Validation

We conducted an experiment comparing the scaled performance of *RIDM* against that of the *GAIfO+RL* baseline on six tasks from the OpenAI Gym domain. The demonstrators are generated using *TRPO/PPO* with augmented states. However, the demonstrated trajectories only include the

raw state information. Here, we first model the inverse dynamics model using a neural network and train the network to minimize the received cost while attempting to follow the demonstrator trajectory. The results are presented in Figure 7.4. It can be seen that *RIDM* outperforms *GAIfO+RL* in five of the domains. The only domain that the performance is worse than the baseline is the Ant domain. We speculate that the neural network IDM is not able to learn a meaningful model due to the complexity of the domain resulting from the large number of joints compared to each of the other domains. In Section 7.5, we show that if *RIDM* uses a lower-dimensional parameterized IDM (e.g. a PID controller), the performance of the learning agents is improved.

7.4.3 *RIDM* Applied to SimSpark RoboCup 3D Simulation

We now report the results of using *RIDM* to learn agent behaviors in the RoboCup 3D simulation environment, SimSpark[13, 152]. Specifically, our goal was to determine whether or not *RIDM* could imitate agent skills exhibited by the agents of other teams that participate in the RoboCup 3D simulation competition [78]. Since the opponent's policies are unknown, we obtain the demonstration by executing the teams' computer-readable but non-human-readable code in the environment.

In our experiments, we are interested in two tasks: (1) speed walking, and (2) long distance kick-offs. We collect demonstration data of two teams, FC Portugal (FCP) [104] and FUT-K [59]. *RIDM* pre-trained the model (see Section 7.3.1) using walk and kick exploration policies from our own team,

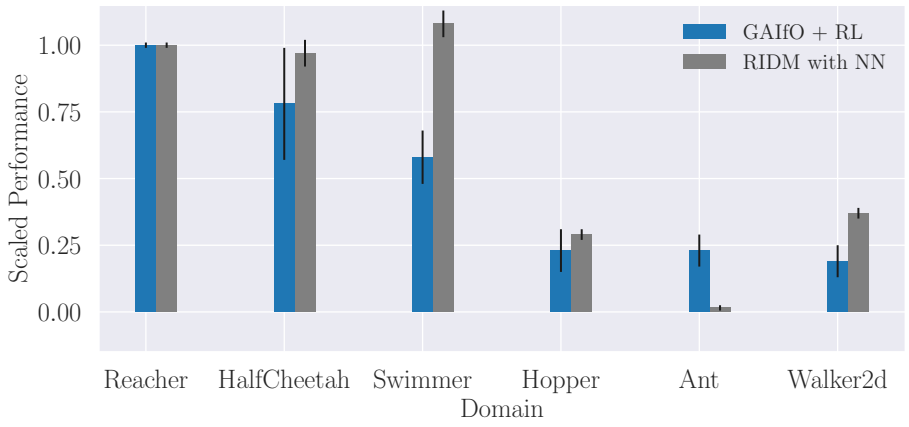

Figure 7.4: Comparison of *RIDM* final performance against established baseline, *GAIfO+RL*, on the MuJoCo domains on the same single state-only demonstration consisting of only raw states (exclusively of joint angles). Mean and standard deviations for *GAIfO+RL* and *RIDM* are over 100 policy runs. *GAIfO+RL* uses a neural network parameterized policy. For each domain, in order of x-axis, the numbers of iterations required for *RIDM* are 700, 800, 400, 100, 900, and 1300 and for *GAIfO+RL* are 400, 800, 1000, 1000, 1200, and 1500

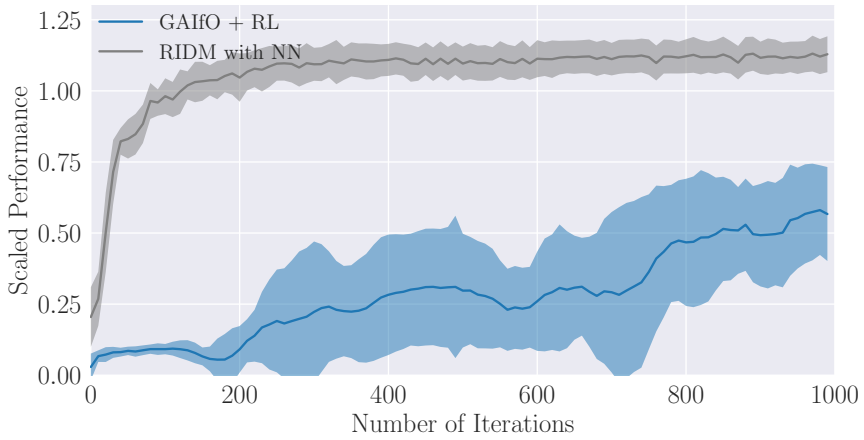

Figure 7.5: Comparison of *RIDM* learning process against established baseline, *GAIfO+RL*, on the Swimmer domain on the same single state-only demonstration consisting of only raw states (exclusively of joint angles). Solid lines represent the mean return and shaded areas represent standard deviations over 10 trials. While the shown graph is for Swimmer-v2, we observed the same qualitative trend on other domains as well.

UT Austin Villa[78]. Here, we report results using only *RIDM* since it proved infeasible to evaluate *GAIfO+RL* in this domain due to the computational time complexity. We found that *RIDM* performed best with global PD gains common to all joints as the inverse dynamics model.

Below are the cost function details of each task:

- Speed walking: Summation of distances (meters) travelled per time-step with a -5 penalty for falling down.
- Long-distance kick-off:

$$C_{kick} = -(1 + x_{total}) \cdot \exp\left(\frac{-\theta^2}{180}\right) - x_{air} \cdot 100$$

with a penalty -5 for bumping the ball, -10 for falling down, where x_{total} is the x-axis distance traveled by the ball, θ is the angle between the ball's trajectory and the line between the agent and center of the goal, and x_{air} is the x-axis distance for which the ball was traveling in the air. Distances are in meters, and θ is in degrees.

Since we defined these cost functions independently from the demonstrations, the demonstrations do not optimize the cost signals. The demonstrators are trained for the RoboCup task; their performances are sub-optimal with regards to our designed cost functions.

Tables 7.1 and 7.2 and summarize our results. We report both the performance of the demonstrator and our agent. We can see that, since *RIDM* takes advantage of both the cost functions and the demonstrations, it allows our agents to *outperform* the sub-optimal demonstrators.

Table 7.1: *RIDM* vs. demonstrator for speed walking.

Demonstrator	Agent	Speed (m/s)	Cost
FCP	***RIDM* (ours)**	**0.81**	**-9.82**
	Demonstrator	0.69	-8.35
FUT-K	***RIDM* (ours)**	**0.89**	**-10.70**
	Demonstrator	0.70	-8.47

Table 7.2: *RIDM* vs. demonstrator for long-distance kick offs.

Demonstrator	Agent	x_{air} (m)	x_{total} (m)	Cost
FCP	***RIDM* (ours)**	**13.78**	**24.05**	**-1386.00**
	Demonstrator	8.00	17.00	-808.00
FUT-K	***RIDM* (ours)**	**10.62**	**16.23**	**-1064.00**
	Demonstrator	0.00	10.00	-1.00

7.4.4 *RIDM* Applied to a Physical UR5 Robot Arm

We also used *RIDM* for behavior learning on a physical robot. Specifically, we used a UR5, a 6-degree-of-freedom robotic arm. We considered a reaching task in which the arm begins in a consistent, retracted position, then must move its end effector (i.e., the gripper at the end of the arm) to a target point in Cartesian space, and finally must stop moving once the end effector has reached the target point. We trained the demonstrator by iterating between iLQR [130] and dynamics learning with a specified cost function. We then executed this demonstrator policy and recorded the resulting trajectories to create the demonstration data [137].

For the physical arm experiments, we skip the pre-training phase for two reasons: 1) we did not have access to a sub-optimal policy for each task, and 2) for safety concerns, we did not want to use a random exploration policy. For *RIDM*'s second phase, we used a cost function defined as the Euclidean distance of the end effector of the arm to the target point at each time step. We used Bayesian optimization[96] as the blackbox optimization algorithm to update the model parameters in response to the environment cost. Bayesian

optimization works by constructing a posterior distribution over the space of functions being optimized over. Here, this distribution was represented using a Gaussian process over functions that map PID values to the episode returns. As the training proceeds and more data is observed, Bayesian optimization techniques sharpen the posterior, resulting in more certainty as to which regions of the parameter space are worth exploring further with more trials and which are not. For simpler optimization problems, Bayesian optimization is more sample efficient compared to CMA-ES and converges within a few iterations.

Table 7.3 represents the results of our experiments on the UR5, where we compare *RIDM* to a baseline behavior generated by using the demonstration state sequence as set points for the platform's pre-defined, hard-coded PID controller parameterization. The reported numbers are the averages and standard deviations of episode returns over five separate experiments all of which are reaching tasks with different target points. Table 7.3 shows that while *RIDM* outperforms the original PID, it is worse than the demonstrator. The reason is that the demonstrator is optimal with regards to the designed cost function.

Table 7.3: *RIDM* vs. original PID controller vs. demonstrator.

Agent	Reaching	Pushing	Pouring
***RIDM* (ours)**	**11.94 (1.55)**	**19.01 (1.03)**	**5.87 (0.08)**
Original PID controller	36.57 (0.97)	58.98 (0.15)	15.67 (0.68)
Demonstrator	5.64 (0.76)	8.43 (0.11)	2.31 (0.04)

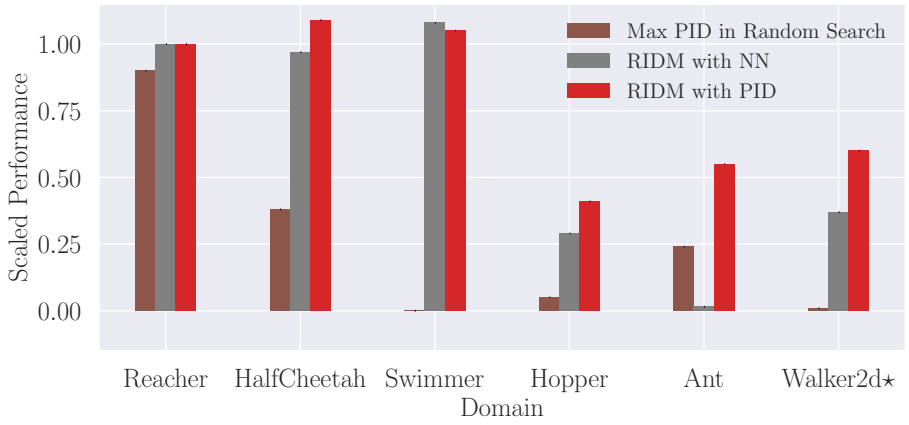

Figure 7.6: Comparison of *RIDM* with PID as the IDM versus *RIDM* with NN as the IDM and the maximum performance between the randomly generated PID values on the MuJoCo domain on the same single state-only demonstration consisting of only raw states (exclusively of joint angles). Since *RIDM* with PID uses a deterministic inverse dynamics model, we do not report mean or standard deviations of our algorithm. ⋆PID version of *RIDM* used global PID gains for Walker2d-v2, unlike on other domains where it used local PD gains.

7.5 Additional Results

Due to the impressive results acquired by *RIDM* with PID controllers presented in previous sections, we performed another set of experiments on the MuJoCo domains that used a PID controller as the IDM instead of a neural network to see the effects that these two types of IDM have on the performance of the algorithm. We compare *RIDM* with with PID controller to 1) *RIDM* with a NN IDM (from Figure 7.4) and 2) the best-performing randomly generated PID gain. We determined the best-performing random PID gain by sampling 100 PID gains from a Gaussian distribution with mean $[0.45, 0.75, 0.15]$ and standard deviation $[.5, .5, .5]$ (each index in the list corresponds to P, I, and D. In order for these values to be reasonable, they are selected using domain knowledge), and selecting the best-performing set. Figure 7.6 provides experimental results for all six of the domains. One can see that *RIDM* with a PID controller performs similarly to *RIDM* with an NN, and in more complex domains such as Ant and Walker2d, significantly outperforms it. The reasonably good performance of random PID gains shows us that even an un-trained PID controller is an effective IDM. *RIDM* with a PID controller is able to focus on optimizing just the (very few) parameters of the PID controller (i.e., the gains) as opposed to a neural network policy, where the policy space is much larger. Furthermore, no matter what the gains are, the PID has a lot of knowledge built into it in that it is designed to try to reach the set point whereas the NN is not predisposed to exhibit any such behavior.

7.6 Summary

In this chapter, we investigated whether or not several restrictive assumptions common to many techniques that integrate imitation and reinforcement learning – access to demonstrator action information, access to several demonstrations, and knowledge of task-specific state augmentations – are necessary. We hypothesized that they are not, and we proposed a new algorithm called *RIDM* in order to validate that hypothesis. *RIDM* is a fundamentally new method for integrated imitation and reinforcement learning that operates in scenarios for which only a single, raw-state-only demonstration is provided. We experimentally demonstrated that *RIDM* can find behaviors that achieve good task performance in these scenarios. Moreover, our results show that it outperforms a reasonable baseline technique while doing so. We posit that the success of *RIDM* is due to the way in which it generates behavior trajectories and performs learning – *RIDM* generates behavior by directly using the demonstration data as the set points for a parameterized but robust inverse dynamics model, and iteratively optimizes the model parameters in response to the environment cost. The above procedure not only generates reasonable trajectories over which to learn, but also reduces the learning problem to one over a relatively low-dimensional set of parameters when compared to other approaches.

This chapter opens up many possible directions for future work. For one, while *RIDM* is designed for single-demonstration scenarios, it may be useful in some scenarios to try to extend it such that a generalized controller

can be learned using numerous demonstrations of a specific task. For example, in an arm-reaching task, we may have two different demonstrations with two different target reaching points. Another open question is how *RIDM* will perform when using optimization algorithms other than CMA-ES, such as *TRPO* or *PPO*. Furthermore, in all of our experiments, the state spaces are low-level features (only joint angles). Another possible future direction is to investigate how *RIDM* performs with video demonstrations.

Chapter 8

Data-Efficient Adversarial Learning for Imitation from Observation

In this chapter, an extension of the algorithm presented in Chapter 5 (i.e., *GAIfO*) is introduced. This extension, Data-Efficient Adversarial Learning for Imitation from Observation (*DEALIO*) improves the sample-efficiency of *GAIfO* in order to enable agents to learn tasks in the real world. The proposed algorithm is evaluated in the MuJoCo simulator, and the results of experiments on multiple domains are provided.

Work in this chapter is based on a paper, DEALIO*: Data-Efficient Adversarial Learning for Imitation from Observation* [142], published in Proceedings of the International Conference on Intelligent Robots and Systems (IROS 2021).[1]

8.1 Overview

Imitation learning from observation (*IfO*) [77, 141] considers explicitly the case where artificial agents seek to learn behaviors from demonstrations

[1] An earlier version of this work was presented in ICML Workshop on Imitation, Intent, and Interaction (I3) [137]. This paper [142] however, is drastically different both in the design of the proposed algorithm and the experiments.

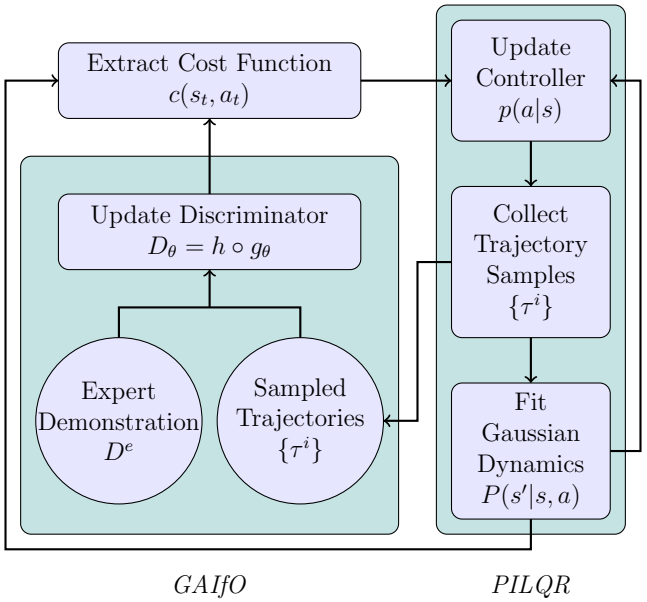

Figure 8.1: A block diagram representing the learning flow of *DEALIO*. A time-varying Gaussian controller $p(a|s)$ is initialized and used to collect trajectories $\{\tau^i\}$ which are then used to fit a linear Gaussian dynamics model $P(s'|s, a)$. The collected data $\{\tau^i\}$ is also used with the demonstration data D^e to train a discriminator D_θ which is a composition function of a quadratic function $h(., s_t, s_{t+1})$ and a neural network $g_\theta(s_t, s_{t+1})$. The discriminator D_θ and the dynamics model $P(s'|s, a)$ are then used to extract the cost function $c(s, a)$ which is then used to update the controller $p(a|s)$.

consisting only of state information. As discussed in Chapter 5 and 6, one class of algorithms that have achieved state-of-the-art performance on *IfO* problems rely on techniques from adversarial learning [40]. Generative Adversarial Imitation from Observation (*GAIfO*) [139, 138] as an example uses adversarial learning to bring the state-transition distribution of the imitator closer to that of the demonstrator. *GAIfO* relies on a model-free *RL* algorithm as part of the learning process, in which the goal is to learn a policy directly from samples without forming or leveraging any intermediate representations of the environment or cost function. These model-free techniques typically exhibit extremely high sample complexity, which can prove problematic in real-world settings in which each sample required for learning incurs a high cost in terms of time, energy, and/or risk.

One classical way in which the *RL* community has dealt with poor sample efficiency is through the use of model-based *RL*. Unlike model-free techniques, model-based *RL* algorithms make assumptions about the environment dynamics such as it being linear or differentiable, and learn a model of the dynamics during the learning process. Model-free *RL* algorithms are known to perform well at the cost of data inefficiency [67, 115], i.e., the final learned policy is able to perform close to optimality, but learning it requires many interactions with the environment. On the other hand, model-based algorithms are more sample efficient, but, since they rely on assumptions to build models, they typically result in sub-optimal policies, especially in environments that exhibit complex dynamics [27]. Some *RL* algorithms, such

as *PILQR* [24] (as presented in Section 2.1.3), have been proposed to take advantage of the benefits of both model-based and model-free algorithms.

In this chapter, we propose to address sample inefficiency in adversarial *IfO* algorithms by integrating ideas from model-based *RL*. In particular, we propose to integrate the sample-efficient *RL* updates from *PILQR* with the high-performing *GAIfO* algorithm for *IfO*. The resulting algorithm, Data-Efficient Adversarial Learning for Imitation from Observation (*DEALIO*), is able to learn a time-varying linear Gaussian controller that can imitate a demonstrator using state-only demonstrations of a behavior. We evaluate *DEALIO* in several simulation domains and compare it with *GAIfO*. Our results show that *DEALIO* can achieve good imitation performance using far fewer environment interactions during learning. Moreover, in some cases, *DEALIO* is even able to outperform *GAIfO*.

8.2 Problem Setting

As in previous chapters, the agents are modeled in the framework of Markov Decision Processes (*MDP*s) which is often represented by a tuple $M = \{\mathcal{S}, \mathcal{A}, P, c\}$, where $\mathcal{S}$ and $\mathcal{A}$ are state and action spaces, respectively, P is a transition probability distribution, and c is the cost function. At a given time instant, a decision-making agent operating within M finds itself in state $s \in \mathcal{S}$ and takes an action $a \in \mathcal{A}$ based on a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$. As a result, agent moves to a new state $s' \in \mathcal{S}$ with probability $P(s'|s, a)$, at which point the agent also receives feedback $c(s, a)$ based on the cost function

$c : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. The goal of reinforcement learning agents is to use their own experience to learn a policy that results in a behavior that incurs minimal expected cumulative cost. Our problem of interest is *IfO* and therefore, the agent does not have access to the cost function. Instead, it has access to demonstrations that only consist of state information, i.e., $D^e = \{\tau_i^e\}$, where $\tau_i^e = \{(s^e)\}_i$.

Our work builds off of *GAIfO* [139, 138, 140] presented in Chapter 5. Recall that *GAIfO* attempts to learn a policy such that the imitator's state-transition distribution matches that of the demonstrator. The algorithm works as follows. First, a random neural network policy π_ϕ is initialized as the imitator's policy, which is used by the imitator to generate state trajectories $\{\tau_i^i\}$ where $\tau_i^i = \{(s^i)\}_i$. Next, a neural network discriminator is trained to discriminate between the state transitions provided by the demonstrator and the state transitions generated by the imitator. Specifically, the discriminator training is done using supervised learning with the following loss function:

$$-\Big(\mathbb{E}_{\{\tau_i^i\}}[\log(D_\theta(s_t, s_{t+1}))] + \mathbb{E}_{\{\tau_i^e\}}[\log(1 - D_\theta(s_t, s_{t+1}))]\Big) \,, \tag{8.1}$$

where D is the discriminator network parameterized by θ and s_t and s_{t+1} are two consecutive states. Finally, the model-free *RL* algorithm Trust Region Policy Optimization (*TRPO*) [115] is used to update the imitator's policy using the cost function

$$\mathbb{E}_{\{\tau_i^i\}}[\log(D_\theta(s_t, s_{t+1}))] \,. \tag{8.2}$$

While *GAIfO* has shown very promising performance, it also exhibits high sample complexity. This chapter introduces an algorithm, *DEALIO*, that integrates *PILQR* into *GAIfO* to make it more sample efficient.

8.3 *DEALIO*

We now introduce *DEALIO*, an algorithm for *IfO* that has the explicit goal of making adversarial imitation learning from observation techniques more sample efficient through the use of model-based *RL*. *DEALIO* is based on the *GAIfO* algorithm discussed in Section 5.4, in which a neural network discriminator attempts to distinguish between state transitions generated by the current imitation policy and state transitions generated by the demonstrator. The discriminator's output is used as the cost function that drives imitation policy learning using the model-free *RL* algorithm *TRPO*. To improve sample efficiency, *DEALIO* replaces *TRPO* with the *PILQR* algorithm that computes policy updates using a combination of model-based and model-free *RL*.

Making this change to *GAIfO* is nontrivial due to the functional form of the cost function required by *PILQR*. First, *PILQR* requires a quadratic approximation of the cost function in order to compute updates, whereas the cost function used in *GAIfO* is specified by the discriminator, which is a continually updating deep neural network. As mentioned in Section 2.1.3, it is not required for the cost function to be in the form of Equation 2.9. Instead it only needs to be twice-differentiable. Therefore, we aim to develop a cost

function in the form of

$$c(s_t, a_t) = \frac{1}{2} \begin{bmatrix} s_t \\ a_t \end{bmatrix}^T C_t \begin{bmatrix} s_t \\ a_t \end{bmatrix} + \begin{bmatrix} s_t \\ a_t \end{bmatrix}^T c_t + cc_t \tag{8.3}$$

for which the quadratic approximation would be the first two terms of the right hand side.[2] Second, *PILQR* requires a cost function over both states and actions (as suggested by Equation 8.3), whereas, because *GAIfO* is an *IfO* technique, the discriminator that specifies the cost function is a function of state information only.

Overcoming these challenges and learning the cost function presented in Equation 8.3 requires us to find a method by which we can compute C_t, c_t, and cc_t in the context of the *GAIfO* algorithmic structure. To do so, we first propose to modify the structure of the discriminator by considering it to be a composition of two functions, $g_\theta(s_t, s_{t+1})$ and $h(., s_t, s_{t+1})$. The first function, $g_\theta(s_t, s_{t+1})$, is a neural network (with parameters θ) that takes state transitions as inputs and outputs the following quantities:

- The elements of a matrix $C^{ss}(s_t, s_{t+1})$
- The elements of a vector $c^{ss}(s_t, s_{t+1})$
- A constant $cc^{ss}(s_t, s_{t+1})$

[2] Another option is to use a neural network discriminator (the same as *GAIfO*) and take the second order Taylor expansion of the neural network to calculate the quadratic approximate of the cost function. This approach also meets the requirements of *PILQR*. We implemented this approach and saw that *DEALIO* outperforms it by a large margin. We hypothesize that the reason is that the manifold learned using the neural network is so complex that its second order Taylor expansion is not meaningful enough.

The second function, $h(., s_t, s_{t+1})$, is a quadratic function in which the outputs of $g_\theta(s_t, s_{t+1})$, i.e. $C^{ss}(s_t, s_{t+1})$ and $c^{ss}(s_t, s_{t+1})$, can be used as its matrix and vector and the overall function composition becomes

$$\begin{aligned} D_\theta(s_t, s_{t+1}) &= h(g_\theta(s_t, s_{t+1}), s_t, s_{t+1}) \\ &= \frac{1}{2} \begin{bmatrix} s_t \\ s_{t+1} \end{bmatrix}^T C^{ss}(s_t, s_{t+1}) \begin{bmatrix} s_t \\ s_{t+1} \end{bmatrix} \\ &+ \begin{bmatrix} s_t \\ s_{t+1} \end{bmatrix}^T c^{ss}(s_t, s_{t+1}) \ . \end{aligned} \tag{8.4}$$

which yield the discriminator we use in *DEALIO*. By imposing this structure on the discriminator, we now have a quadratic cost function, $h(g_\theta(s_t, s_{t+1}), s_t, s_{t+1})$, as required by *PILQR*. However, two issues still remain. First, $h(g_\theta(s_t, s_{t+1}), s_t, s_{t+1})$ is still a function of state transitions rather than state-action pairs. Second, $C^{ss}(s_t, s_{t+1})$ and $c^{ss}(s_t, s_{t+1})$ are functions of states rather than time.

In order to find a quadratic cost function with state-action pairs as input, we use the linear dynamics model assumed by *PILQR* in Equation 2.8, which allows us to rewrite s_{t+1} in $h(g_\theta(s_t, s_{t+1}), s_t, s_{t+1})$ as a function of s_t and a_t. To do so, we first partition quantities from Equations 2.8 and 8.4 as follows:

$$F_t = \begin{bmatrix} F_{s_t} \\ F_{a_t} \end{bmatrix} \ , \tag{8.5}$$

$$C^{ss}(s_t, s_{t+1}) = \begin{bmatrix} C^{ss}_{s_t, s_t} & C^{ss}_{s_t, s_{t+1}} \\ C^{ss}_{s_{t+1}, s_t} & C^{ss}_{s_{t+1}, s_{t+1}} \end{bmatrix} \ , \tag{8.6}$$

and

$$c^{ss}(s_t, s_{t+1}) = \begin{bmatrix} c^{ss}_{s_t} \\ c^{ss}_{s_{t+1}} \end{bmatrix} . \tag{8.7}$$

Next, substituting s_{t+1} from Equation 2.8 into Equation 8.4 and doing some linear algebra, we can write

$$\begin{aligned} h(s_t, a_t) = \frac{1}{2} \begin{bmatrix} s_t \\ a_t \end{bmatrix}^T C^{sa}(s_t, s_{t+1}) \begin{bmatrix} s_t \\ a_t \end{bmatrix} \\ + \begin{bmatrix} s_t \\ a_t \end{bmatrix}^T c^{sa}(s_t, s_{t+1}) , \end{aligned} \tag{8.8}$$

where the new matrix and vector, $C^{sa}(s_t, s_{t+1})$ and $c^{sa}(s_t, s_{t+1})$, respectively, are given by

$$C^{sa}(s_t, s_{t+1}) = \begin{bmatrix} C^{sa}_{s_t,a_t} & C^{sa}_{s_t,a_t} \\ C^{sa}_{a_t,s_t} & C^{sa}_{a_t,a_t} \end{bmatrix} , \tag{8.9}$$

and

$$c^{sa}(s_t, s_{t+1}) = \begin{bmatrix} c^{sa}_{s_t} \\ c^{sa}_{a_t} \end{bmatrix} , \tag{8.10}$$

and the individual partition terms above are computed as

$$\begin{aligned} C^{sa}_{s_t,s_t} &= C^{ss}_{s_t,s_t} + F^T_{s_t} C^{ss}_{s_{t+1},s_t} + C^{ss}_{s_t,s_{t+1}} F_{s_t} \\ &+ F^T_{s_t} C^{ss}_{s_{t+1},s_{t+1}} F_{s_t} , \end{aligned} \tag{8.11}$$

$$C^{sa}_{s_t,a_t} = C^{ss}_{s_t,s_{t+1}} F_{a_t} + F^T_{s_t} C^{ss}_{s_{t+1},s_{t+1}} F_{a_t} , \tag{8.12}$$

$$C^{sa}_{a_t,s_t} = F^T_{a_t} C^{ss}_{s_{t+1},s_t} + F^T_{a_t} C^{ss}_{s_{t+1},s_{t+1}} F_{s_t} \ , \tag{8.13}$$

$$C^{sa}_{a_t,a_t} = F^T_{a_t} C^{ss}_{s_{t+1},s_{t+1}} F_{a_t} \ , \tag{8.14}$$

$$\begin{aligned} c^{sa}_{s_t} = & \frac{1}{2} C^{ssT}_{s_{t+1},s_t} f_t + \frac{1}{2} C^{ss}_{s_t,s_{t+1}} f_t + \frac{1}{2} F^T_{s_t} C^{ssT}_{s_{t+1},s_{t+1}} f_t \\ & + \frac{1}{2} F^T_{s_t} C^{ss}_{s_{t+1},s_{t+1}} f_t + c^{ss}_{s_t} + F^T_{s_t} c^{ss}_{s_{t+1}} \ , \end{aligned} \tag{8.15}$$

and

$$c^{sa}_{a_t} = \frac{1}{2} F^T_{a_t} C^{ssT}_{s_{t+1},s_{t+1}} f_t + \frac{1}{2} F^T_{a_t} C^{ss}_{s_{t+1},s_{t+1}} f_t + F^T_{a_t} c^{ss}_{s_{t+1}} \ . \tag{8.16}$$

Finally, in order to find cost function parameters C_t, c_t, and cc_t (as shown in Equation 8.3) that are functions of time only (i.e., independent of states), we use the mean state transition at time t as input to $C^{sa}(\cdot,\cdot)$, $c^{sa}(\cdot,\cdot)$, and $cc^{ss}(\cdot,\cdot)$, i.e.,

$$C_t = C^{sa}(\bar{s}_t, \bar{s}_{t+1}) \ , \tag{8.17}$$

$$c_t = c^{sa}(\bar{s}_t, \bar{s}_{t+1}) \ , \tag{8.18}$$

and

$$cc_t = cc^{ss}(\bar{s}_t, \bar{s}_{t+1}) \ , \tag{8.19}$$

Algorithm 7 *DEALIO*

1: Initialize controller $p(a|s)$
2: Initialize a neural network discriminator D_θ with random parameter θ
3: Collect demonstration trajectories $D^e = \{\tau^e\}$
4: **while** Controller Improves **do**
5: Execute the controller to collect state-action trajectories $\{\tau_i^i\}$
6: Fit a linear Gaussian dynamics model $P(s'|s,a)$
7: Update D_θ using loss

$$-\Big(\mathbb{E}_{\{\tau_i^i\}}[\log(D_\theta(s_t, s_{t+1}))] + \mathbb{E}_{\{\tau_i^e\}}[\log(1 - D_\theta(s_t, s_{t+1}))]\Big)$$

and store $C^{ss}(s_t, s_{t+1})$, $c^{ss}(s_t, s_{t+1})$, and $cc^{ss}(s_t, s_{t+1})$
8: Compute the cost $c(s_t, a_t)$ by calculating C_t, c_t, and cc_t
9: Perform a *PILQR* update to update the controller $p(a|s)$

where $\bar{s}_t$ represents the mean taken over all the available sample states at each time step.[3] With the above quantities defined, the final cost function (as presented in Equation 8.3) used to perform *PILQR* updates in *DEALIO* is prepared.

DEALIO Having resolved the incompatibility between *GAIfO* and *PILQR* above, we now describe the full proposed algorithm, *DEALIO*, detailed in Algorithm 7. First, we initialize a time-varying, linear Gaussian imitation controller $p(a|s)$ and a neural network discriminator D_θ (Lines 1 and 2). The imitator uses p to generate multiple state-action trajectories $\tau^i = \{(s^i, a^i)\}$ (Line

[3]One alternative is to calculate the average of $C^{sa}(s_t, s_{t+1})$ and $c^{sa}(s_t, s_{t+1})$ over the available states, however, our experiments showed that calculating $C^{sa}(\bar{s}_t, \bar{s}_{t+1})$ and $c^{sa}(\bar{s}_t, \bar{s}_{t+1})$ results in better performance.

5), and these trajectories are used to fit a linear Gaussian dynamics model $P(s'|s,a)$, as shown in Equation 2.12 (Line 6). Next, state transitions from both the imitator and the demonstrator are used to train the discriminator D_θ (Equation 8.4) using the same loss function as in *GAIfO* (Line 7). Then, h and P are used to calculate the quadratic cost function parameters in Equation 8.3 (Line 8). Finally, the cost function is used to perform a *PILQR* update (Line 9).

8.4 Experiments

In order to evaluate *DEALIO*, we ran several experiments using a benchmark robotics simulation environment. In particular, we sought to determine whether or not *DEALIO*, which integrates a model-based *RL* technique with the *GAIfO* algorithm for *IfO*, exhibited better data efficiency than the original *GAIfO* algorithm, which instead uses a model-free *RL* technique. We compared the data efficiency of each approach by analyzing learning curves—more data-efficient approaches exhibit steeper learning curves (i.e., they rise to a higher level of task performance using fewer samples). As we will detail below, our results do indeed confirm that *DEALIO* is more data-efficient than *GAIfO*, suggesting that integrating model-based *RL* approaches is a promising path forward for designing *IfO* approaches that are practical for real-world systems. Moreover, we also found that, even given fewer training samples, *DEALIO* also led to better overall task performance than *GAIfO* in our experimental tasks.

8.4.1 Experimental Setup

We conducted our experiments using four robotics tasks simulated using the MuJoCo physics engine [135]. These tasks are Disc, PegInsertion, GripperPusher, and DoorOpening which are summarized in Section 3.2.

In order to generate demonstration data D^e, expert demonstrators are trained for each task until convergence using *PILQR* with predefined task cost functions that have been previously used in related work [72, 84]. The trained policies are then used to generate 20 demonstration trajectories for each task. The performance of these expert agents is represented as the green horizontal lines in Figure 8.2, where the performance metric (negative expected cumulative cost) is normalized to the expert's level.

8.4.2 Implementation Details

In our experiments, the neural network component of the *DEALIO* discriminator was modeled with two hidden layers, each with one hundred neurons. For fair comparison, the *GAIfO* discriminator was similarly modeled. One important hyperparameter that must be specified for both *DEALIO* and *GAIfO* is the number of trajectory samples generated by the imitator at each iteration (e.g., for *DEALIO*, Line 5 of Algorithm 7); we empirically determined the best value of this hyperparameter separately for each algorithm and environment. For *DEALIO*, we collect ten, three, twenty, and twenty trajectories (each trajectory is one hundred time steps) for each domain in the order mentioned in the beginning of Section 8.4.1. For *GAIfO*, these same pa-

rameters were set as five, five, twenty, and twenty with the same order again. Another important set of hyperparameters required by both algorithms is the number of updates to make to both the discriminator and the controller at each iteration. Again, we empirically determined the best values for these hyperparameters, and we found that ten discriminator updates per iteration and one update for the controller per iteration worked best regardless of algorithm or domain. Above, ten discriminator updates means that we divide the overall data collected at that iteration into ten batches and updated the discriminator once for each batch.

8.4.3 Results and Discussion

For each environment and algorithm, we generated the learning curves shown in Figure 8.2 by computing the mean and standard deviation of the developing imitation policies over ten independent training runs. The results confirm our hypothesis that integrating a model-based *RL* algorithm leads to improved data efficiency: the *DEALIO* learning curves all exhibit a significantly steeper rise than those obtained with *GAIfO*. The results also show that, even over much longer training horizons, *DEALIO* still achieves better final performance than *GAIfO*. We posit that this is possible because *PILQR* itself combines model-free and model-based reinforcement learning algorithms such that it is both data efficient and high performing. Taken together, our results suggest that model-based *RL* can play an important role in bringing adversarial *IfO* methods to real-world scenarios.

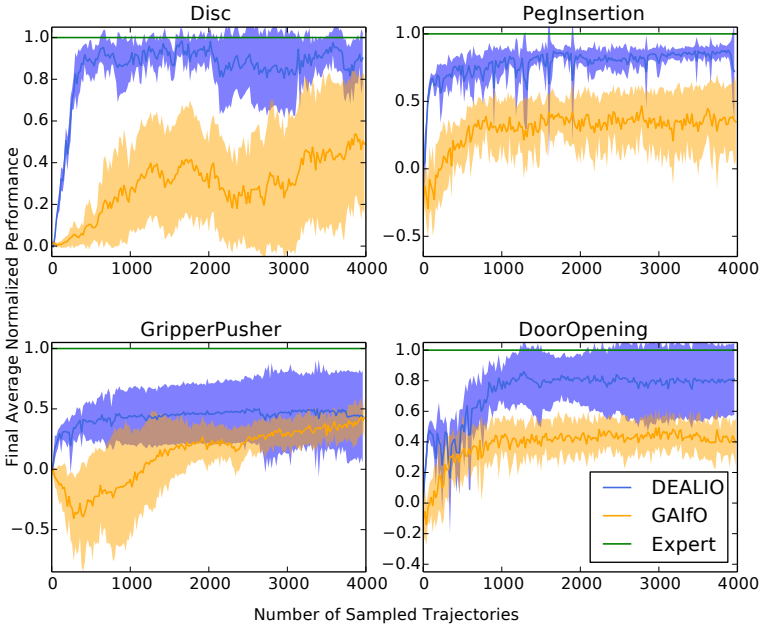

Figure 8.2: Learning with *DEALIO* is compared against *GAIfO*. The plots represent the performance of the algorithms with respect to the number of trajectories sampled during the learning process. The solid lines represent the mean of the performances over 10 different training processes and the shaded areas represent the standard deviations. For comparison purposes, all the performances are scaled such that a random and the expert policy score 0.0 and 1.0, respectively.

8.5 Summary

In this chapter, we studied the effect of integrating sample-efficient, model-based *RL* techniques with recent adversarial *IfO* algorithms. To do so, we introduced one way to perform this integration through our novel algorithm, *DEALIO*, and demonstrated experimentally that (1) it exhibits significantly reduced sample complexity compared to *GAIfO*, and (2) it can do so seemingly without sacrificing performance.

Chapter 9

Related Work

In this chapter, an overall categorization of imitation learning from observation work is proposed, and the existing work in each category is discussed. The presented categorization and the survey of previous work is based on three papers: (1) *Recent Advances in Imitation Learning from Observation* [141], published in Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019), (2) *Leveraging Human Guidance for Deep Reinforcement Learning Tasks* [155] in collaboration with Ruohan Zhang, Lin Guan, and Dana H. Ballard, published in Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019), and (3) *Recent Advances in Leveraging Human Guidance for Sequential Decision-Making Tasks* [156] in collaboration with Ruohan Zhang which is published in the Journal of Autonomous Agents and Multi-Agent System.

9.1 Overview

Recall that, as described in Section 2.2, imitation learning [112, 3, 92] is a problem in machine learning that autonomous agents face when attempting to learn tasks from another, more-expert agent. The expert provides demon-

strations of task execution, from which the imitator attempts to mimic the expert's behavior. Conventionally, methods developed in this framework require the demonstration information to include not only the expert's *states* (e.g., robot joint angles), but also its *actions* (e.g., robot torque commands). For instance, a human expert might provide a demonstration of an object-manipulation task to a robot by manually moving the robot's arms in the correct way, during which the robot can record both its joint angles and also the joint torques induced by the human demonstrator. Unfortunately, requiring demonstration action information prevents imitating agents from using a large number of existing valuable demonstration resources such as online videos of humans performing a wide variety of tasks. These resources provide state information only—the actions executed by the demonstrator are not available.

In order to take advantage of these valuable resources, the more-specific problem of *imitation learning from observation* (*IfO*) must be considered. The *IfO* problem arises when an autonomous agent attempts to learn how to perform tasks by observing *state-only* demonstrations generated by an expert. Compared to the typical imitation learning paradigm described above, *IfO* is a more natural way to consider learning from an expert, and exhibits more similarity with the way many biological agents appear to approach imitation.

Considering the problem of imitation learning using state-only demonstrations is not new [57, 11]. However, with recent advances in deep learning and visual recognition, researchers now have much better tools than before with which to approach the problem, especially with respect to using raw vi-

sual observations. These advances have resulted in a litany of new imitation from observation techniques in the literature, which can be categorized in several fundamentally-different ways. In this chapter, we discuss our proposed categorization and how each work can be placed in each category.

The rest of this chapter is organized as follows. As my thesis can be placed in the broad field of imitation learning, Section 9.2 first offers a short summary of imitation learning and it's general categorization and some of the most recognized works in each category. Then, Section 9.3 introduces our proposed categorization of *IfO* research, where the research in this thesis can be put in these categories, where other *IfO* work can be placed, and how they are related to our work. Finally, Section 9.4 summarizes the chapter.

9.2 Imitation Learning

The learning frameworks surveyed in this chapter are inspired by, an extension of, or combined with traditional imitation learning algorithms. The standard imitation learning setting can be formulated as MDP\c, i.e. there is no cost function c available. Instead, a learning agent (the *imitator*) records expert (the *demonstrator*, could be expert humans or artificial agents) demonstrations in the format of state-action pairs $\{(s^e, a^e)\}$ at each timestep, and then attempts to learn the task using that data.

One approach is for the agent to learn to mimic the demonstrated policy using supervised learning, which is known as behavioral cloning [9]. A second approach to imitation learning is called inverse reinforcement learning (*IRL*)

[1] which involves learning a cost function based on the demonstration data and learning the imitation policy using RL with the learned cost function. These two approaches constitute the major learning frameworks used in imitation learning. Comprehensive reviews of these two approaches can be found in [3, 54, 92, 5, 31]. More recently, generative adversarial imitation learning ($GAIL$) [49] has been proposed, which utilizes the notion of generative adversarial networks (GAN) [40].

Importantly, all of these approaches assume that both s and a are available to the agent. In the rest of this section, we briefly describe the three IL frameworks described above, i.e., *(1)* behavioral cloning (BC), *(2)* inverse reinforcement learning (IRL), and *(3)* adversarial imitation learning (AIL).

9.2.1 Behavioral Cloning (BC)

As described in Chapter 2, behavioral cloning [99, 9] is one of the main methods to approach an imitation learning problem. The agent receives as training data both the encountered states and actions of the demonstrator, then uses supervised learning techniques such as classification or regression to estimate the demonstrator's policy. This method is powerful in the sense that it is capable of imitating the demonstrator immediately without having to interact with the environment, and it has been successfully applied in many application domains. For instance, it has been used to train a quadrotor to fly down a forest trail [38]. There, the training data consists of images of the forest trail gathered by a camera mounted to the quadrotor and labeled with

the actions that the demonstrating quadrotor used. The policy is modeled as a convolutional neural network classifier, and trained using supervised learning. In the end, the quadrotor managed to fly down the trail successfully. *BC* has also been used in autonomous driving [14]. The training data is acquired using a human demonstrator, and a convolutional neural network is trained to map raw pixels from a single front-facing camera directly to platform steering commands. After training, the vehicle was capable of driving in traffic on local roads. *BC* has also been successfully used to teach robotic manipulators complex, multi-step, real-world tasks using kinesthetic demonstrations [91].

One of *BC*'s major drawbacks is potential performance degradation due to the well-studied compounding error caused by covariate shift [106, 107], i.e., that training and testing data distribution mismatch results in deviation of the learned behavior from the demonstration [136]. Ross et al. [107] proposed an interactive training method to correct the shift called DAgger (Dataset Aggregation) which attempts to bring the distribution of demonstration data closer to that of the learned behavior. It does so by collecting demonstration data on the states observed by the imitator at each iteration. Retraining the policy on the aggregated dataset ultimately prevents the imitator from deviating from the demonstration behavior.

9.2.2 Inverse Reinforcement Learning (*IRL*)

Inverse reinforcement learning [1, 162] is another category of imitation learning, *IRL* techniques seek to learn a cost function that has the minimum

value for the demonstrated actions. The learned cost function is then used in combination with *RL* methods to find an imitation policy. To be more specific, most *IRL* algorithms first initialize a random policy. Next, the agent executes that policy in the environment to collect state-action data, and then the algorithms estimate the expert's cost function based on the data generated by the policy and the demonstration data. Finally, standard *RL* algorithms are used to learn an optimal policy for that cost function. The process of cost learning and policy learning is repeated until the agent policy becomes sufficiently close to the demonstrator's policy. Like *BC* techniques, *IRL* methods usually assume that state-action pairs are available [33], and also that the cost is a function of both states and actions. The algorithms developed in this category have shown impressive results in a variety of tasks such as autonomous helicopter aerobatics [2], robot object manipulation [33], and autonomous navigation in complex unstructured terrains [120], etc.

One major drawback of most algorithms developed for *IRL* is that at each iteration, they have to solve a complete *RL* problem to find an optimal policy given the current estimated cost function which is computationally very expensive. However, the learned policies are often more robust than the policies learned by *BC* algorithms as they do not suffer from the covariate shift problem. This shift does not happen in the case of *IRL* because the agent is able to interact with the environment while training and the distribution mismatch diminishes during the process.

9.2.3 Adversarial Imitation Learning

As mentioned in Chapter 2, recently an imitation learning algorithm, generative adversarial imitation learning (*GAIL*) [49], has been developed that alleviates the *IRL*'s drawback just set forth. This algorithm directly learns the policy given demonstration bypassing the optimal cost recovery. *GAIL* formulates the problem of finding an imitating policy as that of solving the following optimization problem:

$$\min_{\pi \in \Pi} \max_{D \in (0,1)^{\mathcal{S} \times \mathcal{A}}} - \lambda_H H(\pi) + \mathbb{E}_{\pi}[\log(D(s,a)] + \mathbb{E}_{\pi^e}[\log(1 - D(s,a))] \,, \quad (9.1)$$

where $\prod$ is a set of agent policies, $\mathbb{E}_{\pi^e}$ is the demonstrator's policy, λ_H is a weight factor, H is the entropy function, and the discriminator function $D : \mathcal{S} \times \mathcal{A} \rightarrow (0, 1)$ can be thought of as a classifier trained to differentiate between the state-action pairs provided by the demonstrator and those experienced by the imitator. The objective in (2.15) is inspired by the one used in generative adversarial networks (*GAN*s) [40]. A *GAN* system is trained in a competitive process: the generator tries to fool the classifier while the classifier tries to distinguish the generated data from the real data. This competitive training process makes both models do better by trying to beat the other. In *GAIL* the associated algorithm can be thought of as trying to induce an imitator state-action occupancy measure that is similar to that of the demonstrator. Even more recently, there has been research on methods that seek to improve on *GAIL* by, e.g., increasing sample efficiency [69, 111] and improving cost representation [35, 102].

9.3 Imitation from Observation

We now turn to the problem that is the focus of this thesis, i.e., that of imitation learning from observation (*IfO*), in which the agent has access to state-only demonstrations (visual observations) of an expert performing a task, i.e., $\tau_i^e = \{o_0^e, o_1^e, \cdots, o_N^e\}_i$. As in *IL*, the goal of the *IfO* problem is to learn an imitation policy π that results in the imitator exhibiting similar behavior to the expert. Broadly speaking, there are two major components of the *IfO* problem: *(1)* perception, and *(2)* control. Most of the *IfO* methods proposed in this thesis are mainly concerned with the control aspect of the *IfO* problem.

9.3.1 Perception

Because *IfO* depends on observations of a more expert agent, processing these observations perceptually is extremely important. In the existing *IfO* literature, multiple approaches have been used for this part of the problem. One approach to the perception problem is to record the expert's movements using sensors placed directly on the expert agent [56]. Using this style of perception, previous work has studied techniques that can allow humanoid or anthropomorphic robots to mimic human motions, e.g., arm-reaching movements [57, 11], biped locomotion [87], and human gestures [21]. A more recent approach is that of motion capture [32], which typically uses visual markers on the demonstrator to infer movement. *IfO* techniques built upon this approach have been used for a variety of tasks, including locomotion, acrobatics, and

martial arts [97, 81, 119]. The methods discussed above often require costly instrumentation and pre-processing [51]. Moreover, one of the goals of *IfO* is to enable task imitation from available, passive resources such as YouTube videos, for which these methods are not helpful.

Recently, however, convolutional neural networks and advances in visual recognition have provided promising tools to work towards visual imitation where the expert demonstration consists of raw video information (e.g., pixel color values). The *IfO* algorithms in this thesis that consider the perception module in Chapter 6, utilize convolutional neural networks to extract the useful information from video demonstrations [139, 140]. However, as the focus of this thesis is mainly on the control aspect of the *IfO* problem, in the cases that perception module is considered, the imitator and the demonstrator have exactly the same observation space. In general, however, the imitating agent is faced with a number of challenges: *(1)* embodiment mismatch, and *(2)* viewpoint difference.

9.3.1.1 Embodiment Mismatch

One challenge that might arise is if the demonstrating agent has a different embodiment from that of the imitator. For example, the video could be of a human performing a task, but the goal may be to train a robot to do the same. Since humans and robots do not look exactly alike (and may look quite different), the challenge is in how to interpret the visual information such that *IfO* can be successful. One *IfO* method developed to address this

problem learns a correspondence between the embodiments using autoencoders in a supervised fashion [42]. The autoencoder is trained in such a way that the encoded representations are invariant with respect to the embodiment features. Another method learns the correspondence in an unsupervised fashion with a small amount of human supervision [118]. A more recent approach integrates the correspondence learning process into an adversarial *IfO* algorithm in which the learning of the policy, mapping, and the cost function are interconnected [53]. The paper demonstrates high performance on a large set of simulation tasks.

9.3.1.2 Viewpoint Difference

Another perceptual challenge that might arise in *IfO* applications comes when demonstrations are not recorded in a controlled environment. For instance, video background may be cluttered, or there may be mismatch in the point of view present in the demonstration video and that with which the agent sees itself. One *IfO* approach that attempts to address this issue learns a context translation model to translate an observation by predicting it in the target context [77]. The translation is learned using data that consists of images of the target context and the source context, and the task is to translate the frame from the source context to that of the target. Another approach uses a classifier to distinguish between the data that comes from different viewpoints and attempts to maximize the domain confusion in an adversarial setting during the training [123]. Consequently, the extracted features can be invariant

with respect to the viewpoint. A more recent *IfO* approach [62] resolves the viewpoint mismatch by detecting keypoints and descriptors and determining keypoint matches based on two nearest neighbors in descriptor space [28].

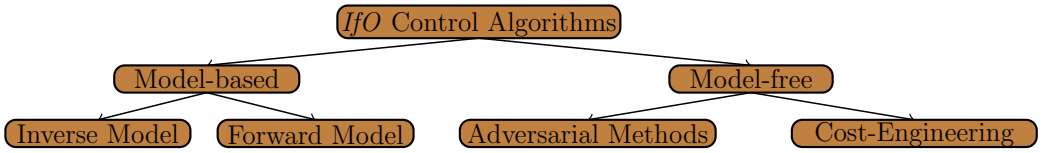

Figure 9.1: A diagrammatic representation of categorization of the *IfO* control algorithm. The algorithms can be categorized into two groups: (1) model-based algorithms in which the algorithms may use either a forward dynamics model [30] or an inverse dynamics model [136, 85]. (2) Model-free algorithms, which itself can be categorized to adversarial methods [139, 81, 123] and cost engineering [118, 42, 77].

9.3.2 Control

Another main component of *IfO* is control, i.e., the approach used to learn the imitation policy, typically under the assumption that the agent has access to clean state demonstration data $\tau_i^e = \{s_0^e, s_1^e, \cdots, s_N^e\}_i$. Since the action labels are not available, this is a very challenging problem, and many approaches, including the ones in this thesis, have recently been proposed in the literature. We organize *IfO* control algorithms in the literature into two general groups: *(1)* model-based algorithms, and *(2)* model-free algorithms. The algorithms presented in Chapters 4, 7, and 8 are model-based and the ones in Chapters 5, and 6 are model-free. In the following subsections, we discuss the features of each group and present relevant example algorithms from the literature.

9.3.2.1 Model-based

Model-based approaches to *IfO* are characterized by the fact that they learn some type of dynamics model during the imitation process. The learned models themselves can be either *(1)* inverse dynamics models, or *(2)* forward dynamics model.

Inverse Dynamics Model

An inverse dynamics model is a mapping from state-transitions $\{(s_t, s_{t+1})\}$ to actions $\{a_t\}$ [45]. One algorithm that learns and uses this kind of model for *IfO* is that of Nair et al. [85]. Given a single video demonstration, the goal of the proposed algorithm is to allow the imitator to reproduce the observed behavior directly. To do so, the algorithm first allows the agent to interact with the environment using an exploratory policy to collect data $\{(s_t, a_t, s_{t+1})\}$. Then, the collected data is used to learn a pixel-level inverse dynamics model which is a mapping from observation transition, $\{(o_t, o_{t+1})\}$, to actions, $\{a_t\}$. Finally, the algorithm computes the actions for the imitator to take by applying the inverse dynamics model to the video demonstration. Critically, this method makes the assumption that each observation transition is reachable through the application of a single action. Pathak et al. [94] attempt to remove this assumption by allowing the agent to execute multiple actions until it gets close enough to the next demonstrated frame. Then this process is repeated for the next frame, and so on.

Both of the algorithms mentioned above attempt to *exactly* reproduce

single demonstrations. Behavioral Cloning from Observation (*BCO*) [136], introduced in Chapter 4, on the other hand, is instead concerned with learning generalized imitation policies using multiple demonstrations. The approach also learns an inverse dynamics model using an exploratory policy, and then uses that model to infer the actions from the demonstrations. Then, however, since the states and actions of the demonstrator are available, a regular imitation learning algorithm (behavioral cloning) is used to learn the task. Later, Robertson and Walter [105] proposed another version of *BCO* and has provided guarantees on the convergence of the new version to behavioral cloning.

In another work, Guo et al. [41] proposed a hybrid algorithm that assumes that the agent also has access to both visual demonstrations and cost information as in the *RL* problem. A method similar to *BCO* is formulated for imitating the demonstrations, and a gradient-based *RL* approach is used to take advantage of the additional cost signal. The final imitation policy is learned by minimizing a linear combination of the behavioral cloning loss and the *RL* loss. Reinforced Inverse Dynamics Modeling (*RIDM*) [95], introduced in Chapter 7, on the other hand, combines *RL* and *IfO* by training the inverse dynamics policy with the cost signal using black-box optimization techniques.

Forward Dynamics Model

A forward dynamics model is a mapping from state-action pairs, $\{(s_t, a_t)\}$, to the next states, $\{s_{t+1}\}$. One *IfO* approach that learns and uses this type of dynamics model is called imitating latent policies from observation (*ILPO*)

[30]. *ILPO* creates an initial hypothesis for the imitation policy by learning a latent policy $\pi(z|s_t)$ that estimates the probability of latent (unreal) action z given the current state s_t. Since actual actions are not needed, this process can be done offline without any interaction with the environment. In order to learn the latent policy, they use a latent forward dynamics model which predicts s_{t+1} and a prior over z given s_t. Then they use a limited number of environment interactions to learn an action-remapping network that associates the latent actions with their corresponding correct actions. Since most of the process happens offline, the algorithm is efficient with regards to the number of interactions needed. Another algorithm of this type is developed by Wu et al. [151] in which a forward dynamics model is learned which is used to predict the future state of the imitating agent and then future state similarity is used to learn an imitation policy. Data-Efficient Adversarial Learning for Imitation from Observation (*DEALIO*), introduced in Chapter 8, also learns a forward dynamics model in the learning process. The learned model is used both in the policy learning process and the cost learning process.

9.3.2.2 Model-free

The other broad category of *IfO* control approaches is that of model-free algorithms. Model-free techniques attempt to learn the imitation policy without any sort of model-learning step. Within this category, there are two fundamentally-different types of algorithms: (1) adversarial methods, and (2) cost-engineering methods.

Adversarial Methods

Adversarial approaches to *IfO* are inspired by the generative adversarial imitation learning (*GAIL*) algorithm described in Section 2.2.3. Motivated by this work, Merel et al. [81] proposed an *IfO* algorithm that assumes access to proprioceptive state-only demonstrations $\{(s^e)\}_i$ and uses a *GAN*-like architecture in which the imitation policy is interpreted as the generator. The imitation policy is executed in the environment to collect data, $\{(s^i)\}_i$, and single states are fed into the discriminator, which is trained to differentiate between the data that comes from the imitator and data that comes from the demonstrator. The output value of the discriminator is then used as a cost to update the imitation policy using *RL*. Another algorithm, called *OptionGAN* [47], uses the same algorithm combined with option learning to enable the agent to decompose policies for cases in which the demonstrations contain trajectories result from a diverse set of underlying cost functions rather than a single one.

In both of the algorithms discussed above, the underlying goal is to achieve imitation policies that generate a state distribution similar to the expert. However, two agents having similar state distributions does not necessarily mean that they will exhibit similar behaviors. For instance, in a ring-like environment, two agents that move with the same speed but different directions (i.e., one clockwise and one counter-clockwise) would result in each exhibiting the same state distribution even though their behaviors are opposite one another. Generative Adversarial Imitation from Observation (*GAIfO*)

[139, 138], introduced in Chapter 5, on the other hand, considers state *transitions*, $\{(s_t, s_{t+1})\}$, as the input to the discriminator instead of single states. Visual *GAIfO* with self-observation, introduced in Chapter 6, considers the cases that the imitator has only access to visual demonstrations $\{o_t\}$ by incorporating CNNs into *GAIfO*, and shows that using multiple video frames instead of single frames resulted in good imitation policies for the demonstrated tasks. In this algorithm, we consider policies to be a mapping from observations $\{o_t\}$ to actions $\{a_t\}$. In follow up work [140], presented in Chapter 6, motivated by the fact that agents often have access to their own internal states (i.e., *proprioception*), we introduced a modified version of this algorithm that leverages this information in the policy learning process. Zolna et al. [163] has built on this work, and proposed an approach to adapt the algorithm to cases in which the imitator and the expert have different action spaces. Instead of using consecutive states as the input of the discriminator, they use pairs of states with random time gaps, and show that this change helps improve imitation performance.

Another adversarial *IfO* approach developed by Stadie et al. [123] considers cases in which the imitator and demonstrator have different viewpoints. To overcome this challenge, a new classifier is introduced that uses the output of early layers in the discriminator as input, and attempts to distinguish between the data coming from different viewpoints. Then they train early layers of the discriminator and the classifier in such a way as to maximize the viewpoint confusion. The intuition is to ensure that the early layers of the

discriminator are invariant to viewpoint. Sun et al. [127] have also developed an adversarial *IfO* approach in which, from a given start state, a policy for each time-step of the horizon is learned by solving a minimax game. The minimax game learns a policy that matches the state distribution of the next state given the policies of the previous time-steps.

GAIL and *GAIfO* consider the problem of minimizing the discrepancy over state-action and state-transition occupancy measures, respectively. Later Yang et al. [153] showed the gap between these problems lies in the disagreement of inverse dynamics models between the imitator and the expert. Therefore, they attempted to minimize that gap as well the occupancy measure discrepancy in order to get better performance.

Cost Engineering

Another class of model-free approaches developed for *IfO* control are those that utilize cost engineering. Here, cost engineering means that, based on the expert demonstrations, a manually-designed cost function is used to find imitation policies via *RL*. Importantly, the designed cost functions are not necessarily the ones that the demonstrator used to produce the demonstrations—rather, they are simply estimates inferred from the demonstration data. One such method, developed by Kimura et al. [64], first trains a predictor that predicts the demonstrator's next state given the current state. The manually-designed cost function is then defined as the Euclidean distance of the actual next state and the one that the approximator returns. An imitation policy

is learned via *RL* using the designed cost function. Another cost-engineering approach is that of time-contrastive networks (*TCN*) [118]. *TCN* considers settings in which demonstrations are generated by human experts performing tasks and the agent is a robot with arms. A triplet loss is used to train a neural network that is used to generate a task-specific state encoding at each time-step. This loss function brings states that occur in a small time-window closer together in the embedding space and pushes others farther apart. The engineered cost function is then defined as the Euclidean distance between the embedded demonstration and the embedded agent's state at each time-step, and an imitation policy is learned using *RL* techniques. Dwibedi et al. [29] claims that, since *TCN* uses single frames to learn the embedding function, it is difficult for *TCN* to encode motion cues or the velocities of objects. Therefore, they extend *TCN* to the multi-frame setting by learning an embedding function that uses multiple frames as the input, and they show that it results in better imitation. Also *TCN* required the imitator and the demonstrator to be time-aligned, therefore, in order to remove this constraint, Liu et al. [76] has proposed an algorithm that is inspired by hierarchical reinforcement learning. This approach selects a sub-goal from the expert demonstration every few steps and attempts to reach that sub-goal using *RL* with manually-designed cost function being Euclidean distance between the observations. Another approach of this type is developed by Goo and Niekum [39] in which the algorithm uses a formulation similar to shuffle-and-learn [83] to train a classifier that learns the order of frames in the demonstration. The manually-specified

cost function is then defined as the progress toward the task goal based on the learned classifier. Aytar et al. [6] also take a similar approach, learning an embedding function for the video frames based on the demonstration. They use the closeness between the imitator's embedded states and some checkpoint embedded features as the cost function.

Gupta et al. [42] consider settings in which the demonstrator and the imitator have different state spaces. First, they train an autoencoder that maps states into an invariant feature space where corresponding states have the same features. Then, they define the cost as the Euclidean distance of the expert and imitator state features in the invariant space at each time-step. Finally, they learn the imitation policy using this cost function with an *RL* algorithm. Liu et al. [77] also uses the same cost function to solve the task however in a setting where the expert demonstrations and the imitator's viewpoints are different. Karnan et al. [62] also consider viewpoint mismatch, however, they consider a deep-learning-based keypoint extractors such as SUPERPOINT [28] to detect keypoints and descriptors and the ratio of the number of keypoint matches between the observations of the imitator and expert is used to generate the cost at each time-step.

9.4 Summary

In this chapter, we reviewed recent advances in imitation learning from observation (*IfO*) and provided an organization of the research that is being conducted in this field.

Chapter 10

Conclusion and Future Work

This chapter summarizes the ideas and contributions of this thesis and also provides directions for future work.

The increasing number and availability of robots, coupled with an increase in the complexity of their capability and a limited number of human operators, has necessitated that robots be able to perform and learn tasks autonomously. Currently, the most well-known approaches used for robot learning are reinforcement learning [128] and imitation learning [1]. The former approach utilizes a framework of learning by exploring the environment and searching for behaviors that optimize the feedback received from a predefined cost function that defines the task. The latter approach utilizes a framework of learning from an expert that demonstrates how the task should be executed. Between these two frameworks, imitation learning in particular is known for (1) its convenience in that it alleviates the laborious process of cost design, and (2) its sample-efficiency in that it reduces the amount of exploration required by the learning agent. However, imitation learning itself imposes some constraints on the learning problem such as requiring access to both states and *actions* of the demonstrator. Requiring access to the actions is limiting as

it prevents the agent to take advantage of a large number of existing valuable demonstration resources such as online videos of humans performing a wide variety of tasks. These resources include state information (i.e., video frames) of the demonstrator, but the actions are not available.

The problem of learning from state-only demonstrations is called *imitation learning from observation* which is the title of this thesis. This problem can be divided into two main components, (1) perception of the demonstrations, and (2) learning an autonomous control policy. The main focus of this thesis has been primarily on the latter, i.e., the control aspect of the problem. To approach this problem, this thesis has introduced multiple algorithms.

This thesis has provided multiple model-based *IfO* algorithms, one of which is called *Behavioral Cloning from Observation* (*BCO*), which was introduced in Chapter 4. In *BCO*, an inverse dynamics model is learned using data collected by the learner exploring the environment. The learned model is then used to infer the missing actions of the demonstrator. Using the inferred actions, the problem is then treated as a regular imitation learning problem and behavioral cloning is used to learn an imitation policy. Another model-based algorithm called *Reinforced Inverse Dynamics Modeling* (*RIDM*) was introduced in Chapter 7, which has access to both a single, state-only demonstration and a cost function. This algorithm combines reinforcement learning and imitation learning from observation so as to take advantage of the sparse cost feedback available to the agent in order to improve the performance.

This thesis also introduced multiple model-free algorithms, one of which

is called *Generative Adversarial Imitation from Observation* (*GAIfO*), which was discussed in Chapter 5. This algorithm is inspired by Generative Adversarial Networks (*GAN*s) and attempts to match the distribution of the imitator's state transitions to that of the demonstrator. Two extensions of this algorithm were introduced in Chapter 6 that incorporate the perception module of the *IfO* problem intro *GAIfO* by using convolutional neural networks. Another extension, called Data-Efficient Adversarial Learning for Imitation from Observation (*DEALIO*), was introduced in Chapter 8. *DEALIO* integrates model-based reinforcement learning algorithms into *GAIfO* in order to make it efficient enough that it could be deployed in the real world.

The remainder of this chapter is organized as follows. Section 10.1 summarizes the contributions made in this thesis, then Section 10.2 provides some directions for future work, and finally Section 10.3 concludes.

10.1 Contributions

As first presented in Section 1.2, this thesis has provided the following contributions:

1. **One model-free algorithm for imitation learning from observation, and two extensions:** Model-free algorithms directly learn tasks without having explicit access to—or a learned model of—the environment. In this thesis, a model-free algorithm for imitation learning from observation, called *Generative Adversarial Imitation from Observation*

(*GAIfO*), was introduced (Chapter 5). Furthermore, two extensions of this algorithm were presented, both of which allow the approach to imitate directly from visual data (Chapter 6).

2. **Three model-based algorithms for imitation learning from observation:** Model-based algorithms are algorithms that, while learning, rely upon a model of the environment. The model itself could either be known or learned. In this thesis, three model-based algorithms for imitation learning from observation were introduced: *Behavioral Cloning from Observation* (*BCO*) – introduced in Chapter 4 – *Reinforced Inverse Dynamics Modeling* (*RIDM*) – introduced in Chapter 7 – Data-Efficient Adversarial Learning for Imitation from Observation (*DEALIO*) – introduced in Chapter 8.

3. **Theory on applicability of the introduced model-free algorithm:** In this thesis, theoretical results that establish the applicability of *GAIfO* to the general model-free imitation from observation framework formulation are presented.

4. **Empirical evaluation of the developed algorithms on problems both in simulation and in the real world:** In this thesis, all of the algorithms are thoroughly tested either in simulation, in the real world, or both, and we have shown that our introduced algorithms outperform well-known baselines in most cases.

10.2 Future Work

In this section, some directions for future work are provided. First, we will provide general directions for future work and then we'll provide some more specific directions for the specific research conducted in this thesis.

10.2.1 General Directions

In this section, general directions for future *IfO* researchers are presented.

10.2.1.1 Perception

This thesis was mostly focused on the control aspect of the imitation from observation problem. However, to reach the end goal of *IfO* (i.e., imitating from passive visual demonstrations such as YouTube videos), both main components of the problem (i.e., control and perception) should be solved jointly. While the vision community has conducted extensive research on the perception aspect alone [159, 134], there has been only limited work that integrates both perception and control of these modules in an attempt to address the overall *IfO* problem [140, 77]. That said, the research in the perception community is evolving very fast, and one promising direction for future work is to bring the latest ideas there to bear in solving the perception aspect of *IfO*.

As stated in Section 9.3.1, in order to achieve the end goal of true imitation from observation, several additional challenges exist. For example,

IfO algorithms should be able to overcome embodiment mismatch (i.e., the situation in which the imitator and the demonstrator have different physical bodies), and viewpoint mismatch (i.e., when the visual demonstrations are recorded from different viewpoints). Resolving these limitations is a natural next step for extending this research.

Embodiment Mismatch In computer vision, there has been a number of advancements in the area of visual domain adaptation [147], which is concerned with transferring learned knowledge to different visual contexts than the ones in which the system was trained. Domain adaptation has successfully made noticeable progress in image-to-image translation, where the goal is to learn a mapping from one domain to another domain. Some of the approaches that have had great impact are CycleGAN [160], pix2pix [58], dual *GAN* [154], disco *GAN* [63], and others [143]. These approaches may be applicable to *IfO* problems that require solutions to embodiment mismatch, in that if the embodiment of the demonstrator is different from that of the imitator, these approaches potentially could be used to transfer the demonstrator's embodiment to the imitator's. Therefore, it is a promising research direction to explore the use of these techniques in the context of the *IfO* algorithms introduced in this thesis.

Viewpoint Mismatch As mentioned in Section 9.3.1.2, there has been a small amount of recent work that attempts to address the viewpoint mismatch

challenge of *IfO* problem [77, 123, 62]. However, several recent and exciting advances in the computer vision community—led by adversarial training techniques—have resulted in creative ideas that could be used to address the viewpoint mismatch challenge. One such advance is in the area of pose estimation, where recent work based on keypoint detection [22, 146] has enabled the detection of object position and orientation even in cluttered video. This keypoint information would also be invariant to the viewpoint. While there has been a small amount of effort to incorporate these advances in *IfO* [98], there is still much to investigate.

10.2.1.2 Application to Physical Robots

Only a few of the *IfO* algorithms discussed in this thesis have actually been successfully tested on physical robots [118, 77, 95]. That is, many of the algorithms have been studied only in domains developed in simulation engines such as MuJoCo and Pybullet. One factor at play is that, while adversarial methods currently provide state-of-the-art performance for a number of baseline experimental *IfO* problems, these methods exhibit high sample complexity and therefore using them on a real robot is prohibitively expensive (in terms of time, risk to the physical platform, etc.). One approach proposed in this thesis, *DEALIO*, does seek to address this issue by lowering the sample complexity of such methods, but it has yet to have been tested on a physical robot. Thus, an open problem in *IfO* is that of finding ways to adapt the techniques developed thus far such that they can be used in scenarios for which high

sample complexity is prohibitive, i.e., tasks in robotics.

10.2.1.3 Integration

Most of the algorithms introduced in this thesis, along with the ones reviewed in Chapter 9, are exclusively concerned with the *IfO* problem, i.e., finding imitation policies from state-only demonstrations. However, to achieve the overall goal of developing fully-intelligent agents, algorithms that have been developed for other learning paradigms (e.g., reinforcement learning or imitation learning) should be integrated with these techniques. Leveraging the advantages of a myriad of techniques is a promising path to get closer to the goal of creating complete autonomous agents, one of the longstanding goals of AI. While there is some previous work that considers a combination of imitation learning and reinforcement learning [161] or imitation from observation and reinforcement learning [41, 95, 114], there is still much to investigate.

10.2.2 Specific Directions

In this section, future directions are presented for each algorithm introduced in this thesis.

10.2.2.1 Behavioral Cloning from Observation (*BCO*)

In Chapter 4, Behavioral Cloning from Observation (*BCO*) was presented, in which an inverse dynamics model is learned to infer the action information missing from the demonstration, and then a policy is learned us-

ing behavioral cloning over the completed demonstration data. In *BCO*, the inverse dynamics model is first learned using data collected by the imitating agent exploring the environment randomly. This random exploration results in higher sample complexity and probably lower performance. Therefore, one future direction is to look at performing this exploration with different policies. For instance, a known policy that exhibits a state-action distribution close to that of the expert would likely perform better than a random policy. Indeed, there is extensive research towards smart exploration strategies in the reinforcement learning literature [93, 19], and it would be interesting to investigate how these ideas could be transferred to *BCO*.

Furthermore, in the imitation learning step of *BCO*, behavioral cloning is used to learn an imitating policy. One future direction is to use other methods such as inverse reinforcement learning [33] or generative adversarial imitation learning [49] and investigate how they affect the performance of the learning agents.

10.2.2.2 Generative Adversarial Imitation from Observation (*GAIfO*)

In Chapter 5, Generative Adversarial Imitation from Observation (*GAIfO*) was introduced, in which a *GAN*-like structure was used to move the state-transition distribution of the imitator closer to that exhibited by the demonstrator. Regarding future work, note that the analysis presented in this thesis did not consider policy entropy terms in either the *IRLfO* step or in the *RL*step. Therefore, it would be interesting to include entropy in these equations—as has

been shown to be beneficial in some cases [43, 44]—and investigate its effect on the overall problem and results. Another way to improve upon *GAIfO* as presented here would be to attempt to make the training more reliable by incorporating techniques developed to improve the stability of *GAN*s, such as the work of Arjovsky et al. [4].

10.2.2.3 Visual Extensions of *GAIfO*

In Chapter 6, two extensions of *GAIfO* were presented that integrate a perception module into the algorithm in order to enable the autonomous agent to imitate from visual demonstrations. These algorithms, however, are not able to imitate from demonstrations that exhibit mismatch in terms of embodiment or viewpoint. Therefore, following what has been described in Section 10.2.1.1, utilizing the ideas from computer vision in order to overcome these challenges would be an interesting direction to pursue for future work.

10.2.2.4 Reinforced Inverse Dynamics Modeling (*RIDM*)

In Chapter 7, Reinforced Inverse Dynamics Modeling (*RIDM*) was presented, in which reinforcement learning is used to tune an inverse dynamics policy to follow a state-only demonstration. This algorithm opens up many possible directions for future work. For instance, *RIDM* attempts to follow only a single demonstration trajectory. Therefore, one future direction could be extending *RIDM* to learn a generalized controller from numerous demonstrations of a specific task. For example, in an arm-reaching task, *RIDM* learns

an inverse dynamics policy for starting from one specific starting point and reaching one specific target point. Therefore, for this specific task, the improvement would be enabling *RIDM* to take different demonstrations and provide a policy that could generalize to unseen starting and target points. Another open question is how *RIDM* will perform when using optimization algorithms other than *CMA-ES*, such as *TRPO* or *PPO*. Furthermore, in all of our experiments, the state spaces are low-level features (only joint angles). Another possible future direction is to integrate *RIDM* with perception modules, and investigate how it performs with video demonstrations.

10.2.2.5 Data-Efficient Adversarial Learning for Imitation from Observation (*DEALIO*)

In Chapter 8, Data-Efficient Adversarial Learning for Imitation from Observation (*DEALIO*) was introduced, in which a model-based reinforcement learning algorithm, *PILQR*, is integrated with *GAIfO* in order to lower the sample complexity. While our work suggests that model-based *RL* can play an important role in improving adversarial techniques for *IfO*, there are several ways in which we might improve upon the exemplar algorithm we proposed here. One such way concerns our use of *PILQR*, which is a trajectory-centric reinforcement learning algorithm, i.e., it finds a local controller, $p(a|s)$, only for a very specific task with a very specific initial state and goal. While this approach appears to have been sufficient for our experimental domains, an interesting direction for future work is to take advantage of algorithms such as Guided Policy Search (*GPS*) [73] to integrate general neural network policy

learning using a supervised learning algorithm to mimic the combined behavior of all the learned local controllers for different initial states and goals.

10.3 Concluding Remarks

This thesis introduced multiple imitation learning from observation control algorithms that allow agents to learn by observing just the state information of an expert performing a task. Moreover, two algorithms were proposed that integrate a perception module in order to get closer to the ultimate goal of *IfO*, i.e., being able to imitate from general visual observations. Furthermore, this thesis provided a survey and a categorization of the research conducted in this area. We hope that the research presented here will contribute to the AI and machine learning community's ongoing effort to develop fully-autonomous agents in the real world.

Appendices

Appendix A

Proofs and Additional Experimental Details for *GAIfO*

In this appendix, we provide proofs and experimental details for Generative Adversarial Imitation from Observation (*GAIfO*), introduced in Chapter 5.

A.1 Proofs

Proofs of the lemmas, propositions and claims are provided in this section.

A.1.1 Proof of Lemma 5.4.2

In order to prove Lemma 5.4.2, we first define a state-action occupancy measure, $\rho^a_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ as

$$\rho^a_\pi(s,a) = \pi(a|s) \sum_{t=0}^{\infty} \gamma^t P(s_t = s|\pi) \ . \tag{A.1}$$

This occupancy measure corresponds to the distribution of state-action pairs that an agent encounters when navigating the environment with policy π [49]. The set of valid state-action occupancy measures can be defined as $\mathrm{P}^a \triangleq \{\rho^a_\pi : \pi \in \Pi\} = \{\rho^a : \rho^a \geq 0, \ \ \sum_a \rho^a(s,a) = p_0(s) + \gamma \sum_{s',a} P(s|s',a)\rho^a(s',a) \ \ \forall s \in$

$\mathcal{S}\}$ where ρ^a could be any distribution, $\rho^a : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$.

Lemma A.1.1. *(Theorem 8.9.4 of Puterman [101]).* P^a *is compact and convex.*

Having established Lemma A.1.1, we now introduce another lemma which helps us in the proof of Lemma 5.4.2:

Lemma A.1.2. P^s *is compact and convex.*

Proof. Convexity: Based on (A.1) and (5.4), we know that that $\rho^s_\pi(s_i, s_j) = \sum_a P(s_j|s_i, a)\rho^a_\pi(s_i, a)$. We need to show that P^s is convex, meaning that for any $\rho^s_{\pi_1}, \rho^s_{\pi_2} \in \mathrm{P}^s$ and any λ with $0 \leq \lambda \leq 1$, we have $\lambda\rho^s_{\pi_1} + (1-\lambda)\rho^s_{\pi_2} \in \mathrm{P}^s$ [15].

$$\begin{aligned}\lambda\rho^s_{\pi_1}(s_i, s_j) + (1-\lambda)\rho^s_{\pi_2}(s_i, s_j) &= \lambda \sum_a P(s_j|s_i, a)\rho^a_{\pi_1}(s_i, a) + \\ &\quad (1-\lambda) \sum_a P(s_j|s_i, a)\rho^a_{\pi_2}(s_i, a) \\ &= \sum_a P(s_j|s_i, a)(\lambda\rho^a_{\pi_1}(s_i, a) + (1-\lambda)\rho^a_{\pi_2}(s_i, a))\end{aligned}$$

Based on Lemma A.1.1, we know that P^a is convex and so $\lambda\rho^a_{\pi_1}(s_i, a) + (1-\lambda)\rho^a_{\pi_2}(s_i, a) \in \mathrm{P}^a$. Therefore, $\sum_a P(s_j|s_i, a)(\lambda\rho^a_{\pi_1}(s_i, a) + (1-\lambda)\rho^a_{\pi_2}(s_i, a))$ is a valid state-transition occupancy measure and belongs to P^s so we can conclude that P^s is convex. Compactness: Based on the definition of $\rho^s_\pi(s_i, s_j)$ in (5.4), we have $\sum_{s_i \in \mathcal{S}} \sum_{s_j \in \mathcal{S}} \rho^s_\pi(s_i, s_j) = 1$. This equality and the non-negativity constraint imply that P^s is bounded and compact [101]. □

Having proven Lemma A.1.2, we now prove Lemma 5.4.2:

proof of Lemma 5.4.2. Let $\bar{c} \in \overline{IRLfO}_{\psi}(\pi^e)$, $\bar{\rho}^s_\pi \in \overline{RL}(\bar{c}) = \overline{RL} \circ \overline{IRLfO}_{\psi}(\pi^e)$, and

$$\hat{\rho}^s_\pi \in \underset{\rho^s_\pi}{\operatorname{argmin}} \ \psi^*(\rho^s_\pi - \rho^s_{\pi^e}) = \underset{\rho^s_\pi}{\operatorname{argmin}} \max_c \ -\psi(c) + \sum_{s,a}(\rho^s_\pi(s,s') - \rho^s_{\pi^e}(s,s'))c(s,s') \tag{A.2}$$

Here we argue that $\bar{\rho}^s_\pi = \hat{\rho}^s_\pi$ which are the outputs of the two sides of the equation that we want to prove. If we consider loss function $L : \mathrm{P}^s \times \mathbb{R}^{\mathcal{S}\times\mathcal{S}} \to \mathbb{R}$ to be

$$L(\rho_\pi, c) = -\psi(c) + \sum_{s,s'}(\rho_\pi(s,s') - \rho_{\pi^e}(s,s'))c(s,s') \tag{A.3}$$

then we can write:

$$\hat{\rho}^s_\pi \in \underset{\rho^s_\pi \in \mathrm{P}^s}{\operatorname{argmin}} \max_c \ L(\rho^s_\pi, c) \tag{A.4}$$

$$\bar{c} \in \max_c \underset{\rho^s_\pi \in \mathrm{P}^s}{\operatorname{argmin}} \ L(\rho^s_\pi, c) \tag{A.5}$$

$$\bar{\rho}^s_\pi \in \underset{\rho^s_\pi \in \mathrm{P}^s}{\operatorname{argmin}} \ L(\rho^s_\pi, \bar{c}) \tag{A.6}$$

Based on Lemma A.1.2, P^s is compact and convex and $\mathbb{R}^{\mathcal{S}\times\mathcal{S}}$ is convex; Moreover, $L(., c)$ is convex over all c, and $L(\rho^s_\pi, .)$ is concave over all ρ^s_π. Therefore, based on minimax duality:

$$\min_{\rho^s_\pi \in \mathrm{P}^s} \max_c \ L(\rho^s_\pi, c) = \max_c \min_{\rho^s_\pi \in \mathrm{P}^s} \ L(\rho^s_\pi, c) \tag{A.7}$$

Therefore based on (A.4) and (A.5), $(\hat{\rho}^s_\pi, \bar{c})$ is a saddle point of L, which implies that $\hat{\rho}^s_\pi \in \underset{\rho^s_\pi \in \mathrm{P}^s}{\operatorname{argmin}} \ L(\rho^s_\pi, c)$ and so $\bar{\rho}^s_\pi = \hat{\rho}^s_\pi$. □

A.1.2 Proof of the claims made for the generative adversarial regularizer

In this section, we prove the claims made that (5.7) is convex and its convex conjugate is (5.9).

Proposition A.1.1. *Suppose* $\psi_\kappa : \mathbb{R}^{\mathcal{S}\in\mathcal{S}} \to \overline{\mathbb{R}}$

$$\begin{aligned}
\psi_\kappa(c) &= \begin{cases} \sum_{s,s'} \rho_{\pi^e}(s,s') g_\kappa(c(s,s')) & \text{if } c(s,s') \in T \text{ for all } s,s' \\ +\infty & \text{otherwise} \end{cases}, \\
g_\kappa(x) &= \begin{cases} -x + \kappa(-\kappa^{-1}(-x)) & \text{if } x \in T \\ +\infty & \text{otherwise} \end{cases},
\end{aligned} \tag{A.8}$$

where $\kappa : \mathbb{R} \to \mathbb{R}$ *is a strictly decreasing convex function and* $g_\kappa : \mathbb{R} \to \overline{\mathbb{R}}$ *and* T *is the range of* $-\kappa$*. Then* ψ_κ *is closed, proper, and convex, and if* $\kappa = \log(1 + e^{-x})$ *(note that with this logistic loss, (A.8) becomes equivalent to (5.7)), then:*

$$\psi^*_{GA}(\rho^s_\pi - \rho^s_{\pi^e}) = \max_{D\in(0,1)^{\mathcal{S}\times\mathcal{S}}} \sum_{s,s} \rho^s_\pi(s,s') \log(D(s,s')) + \rho^s_{\pi^e}(s,s') \log(1 - D(s,s')) \,. \tag{A.9}$$

Proof. Closedness: κ is strictly decreasing; consequently, the range of this function is either $\mathbb{R}$ or (a,∞). In both cases, $\kappa(-\kappa^{-1}(-x))$ is closed and g_κ is closed; therefore ψ_κ is closed as well. Properness: T is nonempty so g_κ is proper. Convexity: κ is a strictly decreasing convex function; therefore, κ^{-1} is convex and so $-\kappa^{-1}$ is concave. Hence, $\kappa(-\kappa^{-1}(-x))$ is convex and as a

result g_κ and ψ_κ are convex as well. Now we prove the second claim made in the proposition:

$$\psi^*_\kappa(\rho^s_\pi - \rho^s_{\pi^e}) = \max_{c\in\mathcal{C}} \sum_{s,s'} (\rho^s_\pi(s,s') - \rho^s_{\pi^e}(s,s'))c(s,s') - \sum_{s,s'} \rho^s_{\pi^e}(s,s') g_\kappa(c(s,s')) \tag{A.10}$$

$$= \sum_{s,s'} \max_{c\in T} \ (\rho^s_\pi(s,s') - \rho^s_{\pi^e}(s,s'))c - \rho^s_{\pi^e}(s,s')[-c + \kappa(-\kappa^{-1}(-c))] \tag{A.11}$$

$$= \sum_{s,s'} \max_{c\in T} \ \rho^s_\pi(s,s')c - \rho^s_{\pi^e}(s,s')\kappa(-\kappa^{-1}(-c)). \tag{A.12}$$

Since T is the range of $-\kappa$, by change of variables $c \to -\kappa(\gamma)$

$$\psi^*_\kappa(\rho^s_\pi - \rho^s_{\pi^e}) = \sum_{s,s'} \max_{\gamma\in\mathbb{R}} \ \rho^s_\pi(s,s')(-\kappa(\gamma)) - \rho^s_{\pi^e}(s,s')\kappa(-\kappa^{-1}(\kappa(\gamma))) \tag{A.13}$$

$$= \sum_{s,s'} \max_{\gamma\in\mathbb{R}} \ \rho^s_\pi(s,s')(-\kappa(\gamma)) - \rho^s_{\pi^e}(s,s')\kappa(-\gamma). \tag{A.14}$$

Substituting the logistic function $\kappa(x) = \log(1 + e^{-x})$,

$$\psi^*_{GA}(\rho^s_\pi - \rho^s_{\pi^e}) = \sum_{s,s'} \max_{\gamma\in\mathbb{R}} \ \rho^s_\pi(s,s') \log\left(\frac{1}{1+e^{-\gamma}}\right) + \rho^s_{\pi^e}(s,s') \log\left(\frac{1}{1+e^{\gamma}}\right) \tag{A.15}$$

$$= \sum_{s,s'} \max_{\gamma\in\mathbb{R}} \ \rho^s_\pi(s,s') \log\left(\frac{1}{1+e^{-\gamma}}\right) + \rho^s_{\pi^e}(s,s') \log\left(1 - \frac{1}{1+e^{-\gamma}}\right) \tag{A.16}$$

$$= \sum_{s,s'} \max_{\gamma\in\mathbb{R}} \ \rho^s_\pi(s,s') \log(\sigma(\gamma)) + \rho^s_{\pi^e}(s,s') \log(1 - \sigma(\gamma)), \tag{A.17}$$

where $\sigma(x) = 1/(1+e^{-x})$ is the sigmoid function. Because the range of σ is $(0,1)$, we can write

$$\psi^*_{GA}(\rho^s_\pi - \rho^s_{\pi^e}) = \sum_{s,s'} \max_{d\in(0,1)} \; \rho^s_\pi(s,s')\log d + \rho^s_{\pi^e}(s,s')\log(1-d) \tag{A.18}$$

$$= \max_{D\in(0,1)^{\mathcal{S}\times\mathcal{S}}} \; \sum_{s,s'} \rho^s_\pi(s,s')\log(D(s,s')) + \rho^s_{\pi^e}(s,s')\log(1-D(s,s')), \tag{A.19}$$

which is the desired expression. □

A.2 Experimental Details

We now discuss some detail regarding the experimental setup and the implementation of *GAIfO* and the baselines.

For *GAIL* and *GAIfO*, each iteration consists of three main components: *(1)* executing the current policy in the environment, *(2)* updating the discriminator, and *(3)* updating the policy. For *(1)*, we executed the policy for 1000 time steps in each domains. For *(2)* and *(3)*, we found that the number of updates performed at each iteration could affect results significantly, and so we performed an extensive hyper-parameter search to find the numbers that performed the best. These numbers are presented in Table B.1. For DoublePendulum and PendulumSwingup, we use deep neural networks with two 64-neuron layers for both the model and policy. For Hopper and Walker2D, we use networks with two 100-neuron layers.

For *BCO*, in DoublePendulum and PendulumSwingup, we use $50,000$ and $5,000$ exploration actions for training and validation in the model learning

stage, respectively. In the case of Hopper and Walker2D, we use $500,000$ and $50,000$ interactions for training and validation, respectively. For *BCO* and *TCN*, in order to be consistent with the generative adversarial methods, we use the same network architectures for each domain.

We compare the methods both in terms of the final performance (mean score at the end of training) and in terms of the learning process. We also vary the number of demonstrations in order to investigate the effect of more or less demonstration data. In this set of experiments, since *GAIL* has access to the demonstrator's actions, we expect it to perform best.

Table A.1: Number of discriminator and policy updates performed at each iteration.

		GAIL		***GAIfO***	
Domain	Demonstration trajectories	Discriminator steps	Policy steps	Discriminator steps	Policy steps
InvertedDoublePendulum	1	10	1	5	3
	5	10	3	10	3
	10	10	3	10	3
	15	5	3	10	3
	20	10	2	10	3
InvertedPendulumSwingup	1	10	1	1	2
	5	10	2	2	3
	10	10	2	1	3
	15	10	3	1	3
	20	5	3	1	3
Hopper	1	3	1	3	2
	5	3	1	10	1
	10	10	1	3	1
	15	10	2	10	2
	20	10	1	10	1
Walker2D	1	10	1	10	2
	5	10	2	10	2
	10	10	2	10	2
	15	1	1	10	2
	20	10	2	10	1

Appendix B

Experimental Details for Visual Extensions of *GAIfO*

In this appendix, information regarding the experimental details for the visual extensions of *GAIfO* presented in Chapter 6 (i.e. *GAIfO* with self-observation and *GAIfO* with proprioceptive information) is presented.

B.1 *GAIfO* with Self-observation

Each iteration of *GAIfO* implemented for visual state representations consists of the same three components as the version discussed in the Section A.2 for *GAIfO*: *(1)* executing the current policy in the environment, *(2)* updating the discriminator, and *(3)* updating the policy. For component *(1)*, we executed the policy for 5120 time steps in each domain. For components *(2)* and *(3)*, we again did an extensive hyperparameter search (the best number of updates for the policy and the discriminator are presented in Table B.1). We use the convolutional architecture presented in Figure 6.1 for each domain.

For these experiments, we compare our approach against *BCO* and *TCN*. For each of these competing techniques, we use the same architecture as the one considered for the policy in Figure 6.1. Note in particular that we

use the grayscale version of the images as inputs.

Table B.1: Number of discriminator and policy updates performed at each iteration.

		GAIfO *with Self-Observation*	
Domain	Demonstration trajectories	Discriminator steps	Policy steps
Reacher	1	10	5
	5	10	5
	10	10	5
	15	5	5
InvertedDoublePendulum	1	10	5
	5	10	5
	10	10	5
	15	10	3
InvertedPendulumSwingup	1	10	5
	5	10	5
	10	10	3
	15	3	3
Hopper	1	10	5
	5	10	5
	10	10	5
	15	5	5

B.2 *GAIfO* with Proprioceptive Information

For the proposed algorithm, we model imitation policies as multi-layer perceptrons with 2 64-neuron layers. We model the discriminator as a convo-

lutional neural network with three convolutional layers with 5-by-5 filters that use a stride of 2. These layers have 16, 32, and 64 filters, respectively. The last layer is fully-connected to a single output, which represents the output value. For training of these networks, we use the Adam variant of stochastic gradient descent [66] with a learning rate of $3e^{-4}$. One important parameter in the training process is the the number of updates performed on the discriminator and the policy at each iteration. Therefore, we performed an extensive hyper-parameter search to find the numbers that performed the best for each experiment which are presented in Table B.2. Also, for each of the baseline algorithms above, we used the hyper-parameters and the architectures used in the original paper. Finally, the same as the previous section, visual observations are represented as 64×64 grayscale images and they are the same across the environment.

Table B.2: Number of discriminator and policy updates performed at each iteration.

		***GAIfO* w\ Proprioception**		***GAIfO* w\ Self-observation**	
Domain	Demonstration trajectories	Policy steps	Discriminator steps	Policy steps	Discriminator steps
MountainCarContinuous	1	1	25	1	10
	5	1	25	1	5
	10	1	25	1	10
	15	1	25	2	10
InvertedPendulum	1	2	5	2	3
	5	3	5	3	3
	10	2	3	2	5
	15	3	5	3	5
InvertedDoublePendulum	1	3	10	3	1
	5	3	5	3	1
	10	3	1	3	1
	15	3	5	3	1
Hopper	1	3	10	1	10
	5	1	10	1	10
	10	1	10	3	10
	15	1	10	1	3
Walker2D	1	3	10	1	10
	5	3	10	1	10
	10	1	10	3	10
	15	1	10	1	3
HalfCheetah	1	1	10	1	10
	5	2	10	1	10
	10	2	10	1	10
	15	1	10	1	10

Appendix C

Acronyms

Acronym	Definition
AIL	Adversarial Imitation Learning
BC	Behavioral Cloning
BCO	Behavioral Cloning from Observation
DEALIO	Data-Efficient Adversarial Learning for Imitation from Observation
GAIfO	Generative Adversarial Imitation from Observation
GAIL	Generative Adversarial Imitation Learning
GAN	Generative Adversarial Networks
GPS	Guided Policy Search
IDM	Inverse Dynamics Model
IfO	Imitation Learning from Observation
IL	Imitation Learning
iLQR	iterative Linear Quadratic Regulator
ILPO	Imitating Latent Policies from Observation
IRL	Inverse Reinforcement Learning
IRLfO	Inverse Reinforcement Learning from Observation
LQG	Linear Quadratic Gaussian
LQR	Linear Quadratic Regulator
MCTS	Monte Carlo Tree Search
MDP	Markov Decision Processes
PI^2	Path Integrated Policy Improvement
PPO	Proximal Policy Optimization
RIDM	Reinforced Inverse Dynamics Modeling
TCN	Time Contrastive Networks
TRPO	Trust Region Policy Optimization

Bibliography

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

[2] Pieter Abbeel, Adam Coates, and Andrew Y Ng. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 29(13):1608–1639, 2010.

[3] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[4] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, pages 214–223, 2017.

[5] Saurabh Arora and Prashant Doshi. A survey of inverse reinforcement learning: Challenges, methods and progress. *arXiv preprint arXiv:1806.06877*, 2018.

[6] Yusuf Aytar, Tobias Pfaff, David Budden, Thomas Paine, Ziyu Wang, and Nando de Freitas. Playing hard exploration games by watching

youtube. In *Advances in Neural Information Processing Systems*, pages 2930–2941, 2018.

[7] JA Bagnell, Joel Chestnutt, David M Bradley, and Nathan D Ratliff. Boosting structured prediction for imitation learning. In *Advances in Neural Information Processing Systems*, pages 1153–1160, 2007.

[8] Michael Bain and Claude Sammut. A framework for behavioral cloning. *Machine Intelligence 14*, 1995.

[9] Michael Bain and Claude Sommut. A framework for behavioural claning. *Machine Intelligence 15*, 15:103, 1999.

[10] Chris L Baker, Rebecca Saxe, and Joshua B Tenenbaum. Action understanding as inverse planning. *Cognition*, 113(3):329–349, 2009.

[11] Darrin C Bentivegna, Ales Ude, Christopher G Atkeson, and Gordon Cheng. Humanoid robot learning and game playing using pc-based vision. In *IROS*, volume 3, pages 2449–2454. IEEE, 2002.

[12] Aude Billard, Sylvain Calinon, Ruediger Dillmann, and Stefan Schaal. Robot programming by demonstration. In *Springer handbook of robotics*, pages 1371–1394. Springer, 2008.

[13] Joschka Boedecker and Minoru Asada. Simspark–concepts and application in the robocup 3d soccer simulation league. *Autonomous Robots*, 174:181, 2008.

[14] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[15] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[16] Douglas A Bristow, Marina Tharayil, and Andrew G Alleyne. A survey of iterative learning control. *IEEE control systems magazine*, 26(3):96–114, 2006.

[17] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[18] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[19] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *International Conference on Learning Representations*, 2018.

[20] Roberto Calandra, André Seyfarth, Jan Peters, and Marc Peter Deisenroth. Bayesian optimization for learning gaits under uncertainty. *Annals of Mathematics and Artificial Intelligence*, 76(1-2):5–23, 2016.

[21] Sylvain Calinon and Aude Billard. Incremental learning of gestures by imitation in a humanoid robot. In *HRI*. ACM, 2007.

[22] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7291–7299, 2017.

[23] Joao Carreira, Pulkit Agrawal, Katerina Fragkiadaki, and Jitendra Malik. Human pose estimation with iterative error feedback. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4733–4742, 2016.

[24] Yevgen Chebotar, Karol Hausman, Marvin Zhang, Gaurav Sukhatme, Stefan Schaal, and Sergey Levine. Combining model-based and model-free updates for trajectory-centric reinforcement learning. In *34th International Conference on Machine Learning (ICML)*, pages 703–711. PMLR, 2017.

[25] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. 2016-2017. URL http://pybullet.org/.

[26] Shreyansh Daftry, J Andrew Bagnell, and Martial Hebert. Learning transferable policies for monocular reactive mav control. In *International Symposium on Experimental Robotics*, pages 3–11. Springer, 2016.

[27] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. A survey on policy search for robotics. *Foundations and trends in Robotics*, 2(1-2): 388–403, 2013.

[28] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superpoint: Self-supervised interest point detection and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 224–236, 2018.

[29] Debidatta Dwibedi, Jonathan Tompson, Corey Lynch, and Pierre Sermanet. Learning actionable representations from visual observations. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1577–1584. IEEE, 2018.

[30] Ashley Edwards, Himanshu Sahni, Yannick Schroecker, and Charles Isbell. Imitating latent policies from observation. In *International Conference on Machine Learning*, pages 1755–1763, 2019.

[31] Bin Fang, Shidong Jia, Di Guo, Muhua Xu, Shuhuan Wen, and Fuchun Sun. Survey of imitation learning for robotic manipulation. *International Journal of Intelligent Robotics and Applications*, pages 1–8, 2019.

[32] Matthew Field, David Stirling, Fazel Naghdy, and Zengxi Pan. Motion capture in robotics review. In *2009 IEEE-ICCA*, pages 1697–1702. IEEE, 2009.

[33] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning*, pages 49–58, 2016.

[34] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot visual imitation learning via meta-learning. In *Conference on Robot Learning*, pages 357–368, 2017.

[35] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adverserial inverse reinforcement learning. In *International Conference on Learning Representations*, 2018.

[36] Tanmay Gangwani and Jian Peng. State-only imitation with transition dynamics mismatch. In *International Conference on Learning Representations*, 2019.

[37] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *JMLR*, 17(1): 2096–2030, 2016.

[38] Alessandro Giusti, Jérôme Guzzi, Dan C Cireşan, Fang-Lin He, Juan P Rodríguez, Flavio Fontana, Matthias Faessler, Christian Forster, Jürgen

Schmidhuber, Gianni Di Caro, et al. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters*, 1(2):661–667, 2016.

[39] Wonjoon Goo and Scott Niekum. One-shot learning of multi-step tasks from observation via activity localization in auxiliary video. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7755–7761. IEEE, 2019.

[40] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, 2014.

[41] Xiaoxiao Guo, Shiyu Chang, Mo Yu, Gerald Tesauro, and Murray Campbell. Hybrid reinforcement learning with expert state sequences. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3739–3746, 2019.

[42] Abhishek Gupta, Coline Devin, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Learning invariant feature spaces to transfer skills with reinforcement learning. In *The International Conference on Learning Representations (ICLR)*, 2017.

[43] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *Proceedings*

of the 34th International Conference on Machine Learning-Volume 70, pages 1352–1361, 2017.

[44] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870, 2018.

[45] Josiah P Hanna and Peter Stone. Grounded action transformation for robot learning in simulation. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 3834–3840, 2017.

[46] Nikolaus Hansen, Sibylle D Müller, and Petros Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary computation*, 11 (1):1–18, 2003.

[47] Peter Henderson, Wei-Di Chang, Pierre-Luc Bacon, David Meger, Joelle Pineau, and Doina Precup. Optiongan: Learning joint reward-policy options using generative adversarial inverse reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[48] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John Agapiou, et al. Learning from demonstrations for real world reinforcement learning. *arXiv preprint arXiv:1704.03732*, 2017.

[49] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pages 4565–4573, 2016.

[50] Jonathan Ho, Jayesh Gupta, and Stefano Ermon. Model-free imitation learning with policy optimization. In *International Conference on Machine Learning*, pages 2760–2769, 2016.

[51] Daniel Holden, Jun Saito, and Taku Komura. A deep learning framework for character motion synthesis and editing. *ACM Transactions on Graphics (TOG)*, 35(4):138, 2016.

[52] Ionel-Alexandru Hosu and Traian Rebedea. Playing atari games with deep reinforcement learning and human checkpoint replay. *arXiv preprint arXiv:1607.05077*, 2016.

[53] Eddy Hudson, Garrett Warnell, Faraz Torabi, and Peter Stone. Skeletal feature compensation for imitation learning with embodiment mismatch. *arXiv preprint arXiv:2104.07810*, 2021.

[54] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):21, 2017.

[55] Jemin Hwangbo, Christian Gehring, Hannes Sommer, Roland Siegwart, and Jonas Buchli. Rockefficient black-box optimization for policy learn-

ing. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 535–540. IEEE, 2014.

[56] Auke Jan Ijspeert, Jun Nakanishi, and Stefan Schaal. Trajectory formation for imitation with nonlinear dynamical systems. In *IROS*, volume 2, pages 752–757. IEEE, 2001.

[57] Auke Jan Ijspeert, Jun Nakanishi, and Stefan Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *ICRA*, volume 2, pages 1398–1403. IEEE, 2002.

[58] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[59] Tomohiro Iwanaga, Kosuke Onda, and Teruya Yamanishi. Fut-k team description paper 2017.

[60] Simon Jégou, Michal Drozdzal, David Vazquez, Adriana Romero, and Yoshua Bengio. The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 11–19, 2017.

[61] Shengyi Jiang, Jingcheng Pang, and Yang Yu. Offline imitation learning

with a misspecified simulator. *Advances in Neural Information Processing Systems*, 33, 2020.

[62] Haresh Karnan, Garrett Warnell, Xuesu Xiao, and Peter Stone. Voila: Visual-observation-only imitation learning for autonomous navigation. *AAAI Spring Symposium Series - Machine Learning for Mobile Robot Navigation*, 2021. URL https://drive.google.com/file/d/135FCPsHu8sg72zzGhv6CJg3AHYajDF5O/view.

[63] Taeksoo Kim, Moonsu Cha, Hyunsoo Kim, Jung Kwon Lee, and Jiwon Kim. Learning to discover cross-domain relations with generative adversarial networks. In *International Conference on Machine Learning*, pages 1857–1865. PMLR, 2017.

[64] Daiki Kimura, Subhajit Chaudhury, Ryuki Tachibana, and Sakyasingha Dasgupta. Internal model from observations for reward shaping. *arXiv preprint arXiv:1806.01267*, 2018.

[65] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[66] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2015.

[67] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning

in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[68] George D Konidaris. A framework for transfer in reinforcement learning. In *ICML-06 Workshop on Structural Knowledge Transfer for Machine Learning*, 2006.

[69] Ilya Kostrikov, Kumar Krishna Agrawal, Debidatta Dwibedi, Sergey Levine, and Jonathan Tompson. Discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning. In *The International Conference on Learning Representations (ICLR)*, 2019. URL https://openreview.net/forum?id=Hk4fpoA5Km.

[70] Aravind S Lakshminarayanan, Sherjil Ozair, and Yoshua Bengio. Reinforcement learning with few expert demonstrations. In *NIPS Workshop on Deep Learning for Action and Interaction*, volume 2016, 2016.

[71] Michael Laskey, Sam Staszak, Wesley Yu-Shu Hsieh, Jeffrey Mahler, Florian T Pokorny, Anca D Dragan, and Ken Goldberg. Shiv: Reducing supervisor burden in dagger using support vectors for efficient learning from demonstrations in high dimensional state spaces. In *In International Conference on Robotics and Automation (ICRA)*, pages 462–469. IEEE, 2016.

[72] Matteo Leonetti, Petar Kormushev, and Simone Sagratella. Combining local and global direct derivative-free optimization for reinforcement learning. *Cybernetics and Information Technologies*, 12(3):53–65, 2012.

[73] Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *NIPS*, volume 27, pages 1071–1079. Citeseer, 2014.

[74] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[75] Weiwei Li and Emanuel Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *ICINCO (1)*, pages 222–229, 2004.

[76] Shanqi Liu, Junjie Cao, Wenzhou Chen, Licheng Wen, and Yong Liu. Hilonet: Hierarchical imitation learning from non-aligned observations. *arXiv preprint arXiv:2011.02671*, 2020.

[77] YuXuan Liu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Imitation from observation: Learning to imitate behaviors from raw video via context translation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1118–1125. IEEE, 2018.

[78] Patrick MacAlpine, Faraz Torabi, Brahma Pavse, John Sigmon, and Peter Stone. Ut austin villa: Robocup 2018 3d simulation league champions. In *Robot World Cup*, pages 462–475. Springer, 2018.

[79] Patrick MacAlpine, Faraz Torabi, Brahma Pavse, and Peter Stone. Ut austin villa: Robocup 2019 3d simulation league competition and techni-

cal challenge champions. In *Robot World Cup*, pages 540–552. Springer, 2019.

[80] Alonso Marco Valle. Gaussian process optimization for self-tuning control. Master's thesis, Universitat Politècnica de Catalunya, 2015.

[81] Josh Merel, Yuval Tassa, Sriram Srinivasan, Jay Lemmon, Ziyu Wang, Greg Wayne, and Nicolas Heess. Learning human behaviors from motion capture by adversarial imitation. *arXiv preprint arXiv:1707.02201*, 2017.

[82] P Warwick Millar. The minimax principle in asymptotic statistical theory. In *Ecole d'Eté de Probabilités de Saint-Flour XI-1981*, pages 75–265. Springer, 1983.

[83] Ishan Misra, C Lawrence Zitnick, and Martial Hebert. Shuffle and learn: unsupervised learning using temporal order verification. In *European Conference on Computer Vision*, pages 527–544. Springer, 2016.

[84] William Montgomery and Sergey Levine. Guided policy search via approximate mirror descent. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 4015–4023, 2016.

[85] Ashvin Nair, Dian Chen, Pulkit Agrawal, Phillip Isola, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Combining self-supervised learning and imitation for vision-based rope manipulation. In *2017 IEEE*

International Conference on Robotics and Automation (ICRA), pages 2146–2153. IEEE, 2017.

[86] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6292–6299. IEEE, 2018.

[87] Jun Nakanishi, Jun Morimoto, Gen Endo, Gordon Cheng, Stefan Schaal, and Mitsuo Kawato. Learning from demonstration and adaptation of biped locomotion. *Robotics and autonomous systems*, 47(2-3):79–91, 2004.

[88] Matthias Neumann-Brosig, Alonso Marco, Dieter Schwarzmann, and Sebastian Trimpe. Data-efficient autotuning with bayesian optimization: An industrial control study. *IEEE Transactions on Control Systems Technology*, 2019.

[89] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *European conference on computer vision*, pages 483–499. Springer, 2016.

[90] Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, pages 663–670, 2000.

[91] Scott Niekum, Sarah Osentoski, George Konidaris, Sachin Chitta, Bhaskara Marthi, and Andrew G Barto. Learning grounded finite-state representations from unstructured demonstrations. *The International Journal of Robotics Research*, 34(2):131–157, 2015.

[92] Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J Andrew Bagnell, Pieter Abbeel, Jan Peters, et al. An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, 7(1-2):1–179, 2018.

[93] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning*, pages 2778–2787. PMLR, 2017.

[94] Deepak Pathak, Parsa Mahmoudieh, Guanghao Luo, Pulkit Agrawal, Dian Chen, Yide Shentu, Evan Shelhamer, Jitendra Malik, Alexei A Efros, and Trevor Darrell. Zero-shot visual imitation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 2050–2053, 2018.

[95] Brahma S Pavse, Faraz Torabi, Josiah Hanna, Garrett Warnell, and Peter Stone. Ridm: Reinforced inverse dynamics modeling for learning from a single observed demonstration. *IEEE Robotics and Automation Letters*, 5(4):6262–6269, 2020.

[96] Martin Pelikan, David E Goldberg, and Erick Cantú-Paz. Boa: The bayesian optimization algorithm. In *Proceedings of the 1st Annual Con-*

ference on Genetic and Evolutionary Computation-Volume 1, pages 525–532. Morgan Kaufmann Publishers Inc., 1999.

[97] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (TOG)*, 37(4): 143, 2018.

[98] Xue Bin Peng, Angjoo Kanazawa, Jitendra Malik, Pieter Abbeel, and Sergey Levine. Sfv: Reinforcement learning of physical skills from videos. In *SIGGRAPH Asia 2018*. ACM, 2018.

[99] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems*, pages 305–313, 1989.

[100] Dean A Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural computation*, 3(1):88–97, 1991.

[101] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[102] Ahmed H. Qureshi, Byron Boots, and Michael C. Yip. Adversarial imitation via variational inverse reinforcement learning. In *The International Conference on Learning Representations (ICLR)*, 2019. URL https://openreview.net/forum?id=HJlmHoR5tQ.

[103] Ilija Radosavovic, Xiaolong Wang, Lerrel Pinto, and Jitendra Malik. State-only imitation learning for dexterous manipulation. *arXiv preprint arXiv:2004.04650*, 2020.

[104] Luís Paulo Reis, Nuno Lau, Abbas Abdolmaleki, Nima Shafii, Rui Ferreira, Artur Pereira, and David Simões. Fc portugal 3d simulation team: Team description paper 2017. In *RoboCup Symposium*, 2017.

[105] Zachary W Robertson and Matthew R Walter. Concurrent training improves the performance of behavioral cloning from observation. *arXiv preprint arXiv:2008.01205*, 2020.

[106] Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668, 2010.

[107] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

[108] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, England, 1994.

[109] Stuart Russell. Learning agents for uncertain environments. In *Proceed-*

ings of the eleventh annual conference on computational learning theory, pages 101–103. ACM, 1998.

[110] Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. Adapting visual category models to new domains. In *European conference on computer vision*, pages 213–226. Springer, 2010.

[111] Fumihiro Sasaki, Tetsuya Yohira, and Atsuo Kawaguchi. Sample efficient imitation learning for continuous control. In *The International Conference on Learning Representations (ICLR)*, 2019. URL https://openreview.net/forum?id=BkN5UoAqF7.

[112] Stefan Schaal. Learning from demonstration. In *Advances in neural information processing systems*, pages 1040–1046, 1997.

[113] Stefan Schaal, Auke Ijspeert, and Aude Billard. Computational approaches to motor learning by imitation. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 358(1431):537–547, 2003.

[114] Karl Schmeckpeper, Oleh Rybkin, Kostas Daniilidis, Sergey Levine, and Chelsea Finn. Reinforcement learning with videos: Combining offline observations with interaction. *arXiv preprint arXiv:2011.06507*, 2020.

[115] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[116] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[117] Pierre Sermanet, Corey Lynch, Jasmine Hsu, and Sergey Levine. Time-contrastive networks: Self-supervised learning from multi-view observation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 486–487. IEEE, 2017.

[118] Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, Sergey Levine, and Google Brain. Time-contrastive networks: Self-supervised learning from video. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1134–1141. IEEE, 2018.

[119] Adam Setapen, Michael Quinlan, and Peter Stone. Marionet: Motion acquisition for robots through iterative online evaluative training. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 1435–1436, 2010.

[120] David Silver, J Andrew Bagnell, and Anthony Stentz. Learning from demonstration for autonomous navigation in complex unstructured terrain. *The International Journal of Robotics Research*, 29(12):1565–1592, 2010.

[121] Eduardo D Sontag. *Mathematical control theory: deterministic finite*

dimensional systems, volume 6. Springer Science & Business Media, 2013.

[122] James C Spall. *Introduction to stochastic search and optimization: estimation, simulation, and control*, volume 65. John Wiley & Sons, 2005.

[123] Bradly C Stadie, Pieter Abbeel, and Ilya Sutskever. Third-person imitation learning. In *The International Conference on Learning Representations (ICLR)*, 2017.

[124] Kaushik Subramanian, Charles L Isbell Jr, and Andrea L Thomaz. Exploration from demonstration for interactive reinforcement learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 447–456, 2016.

[125] Wael Suleiman, Eiichi Yoshida, Fumio Kanehiro, Jean-Paul Laumond, and André Monin. On human motion imitation by humanoid robot. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 2697–2704. IEEE, 2008.

[126] Ke Sun, Bin Xiao, Dong Liu, and Jingdong Wang. Deep high-resolution representation learning for human pose estimation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5693–5703, 2019.

[127] Wen Sun, Anirudh Vemula, Byron Boots, and Drew Bagnell. Prov-

ably efficient imitation learning from observation alone. In *International Conference on Machine Learning*, pages 6036–6045, 2019.

[128] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[129] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[130] Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4906–4913. IEEE, 2012.

[131] Matthew E Taylor, Nicholas K Jong, and Peter Stone. Transferring instances for model-based reinforcement learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 488–505. Springer, 2008.

[132] Matthew E Taylor, Halit Bener Suay, and Sonia Chernova. Integrating reinforcement learning with human demonstrations of varying ability. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 617–624, 2011.

[133] Evangelos Theodorou, Jonas Buchli, and Stefan Schaal. A generalized path integral control approach to reinforcement learning. *The Journal of Machine Learning Research*, 11:3137–3181, 2010.

[134] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: Fully convolutional one-stage object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 9627–9636, 2019.

[135] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[136] Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 4950–4957, 2018.

[137] Faraz Torabi, Sean Geiger, Garrett Warnell, and Peter Stone. Sample-efficient adversarial imitation learning from observation. *arXiv preprint arXiv:1906.07374*, 2019.

[138] Faraz Torabi, Garrett Warnell, and Peter Stone. Adversarial imitation learning from state-only demonstrations. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2229–2231, 2019.

[139] Faraz Torabi, Garrett Warnell, and Peter Stone. Generative adversarial

imitation from observation. In *International Conference on Machine Learning Workshop on Imitation, Intent, and Interaction (I3)*, 2019.

[140] Faraz Torabi, Garrett Warnell, and Peter Stone. Imitation learning from video by leveraging proprioception. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 3585–3591. AAAI Press, 2019.

[141] Faraz Torabi, Garrett Warnell, and Peter Stone. Recent advances in imitation learning from observation. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 6325–6331. AAAI Press, 2019.

[142] Faraz Torabi, Garrett Warnell, and Peter Stone. Dealio: Data-efficient adversarial learning for imitation from observation. *arXiv preprint arXiv:2104.00163*, 2021.

[143] Eric Tzeng, Coline Devin, Judy Hoffman, Chelsea Finn, Pieter Abbeel, Sergey Levine, Kate Saenko, and Trevor Darrell. Adapting deep visuomotor representations with weak pairwise constraints. In *Algorithmic Foundations of Robotics XII*, pages 688–703. Springer, 2020.

[144] Mor Vered, Gal A Kaminka, and Sivan Biham. Online goal recognition through mirroring: Humans and agents. In *The Fourth Annual Conference on Advances in Cognitive Systems*, 2016.

[145] Mor Vered, Ramon Fraga Pereira, Maurício C Magnaguagno, Gal A Kaminka, and Felipe Meneguzzi. Towards online goal recognition combining goal mirroring and landmarks. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2112–2114, 2018.

[146] Keze Wang, Liang Lin, Chenhan Jiang, Chen Qian, and Pengxu Wei. 3d human pose machines with self-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[147] Mei Wang and Weihong Deng. Deep visual domain adaptation: A survey. *Neurocomputing*, 312:135–153, 2018.

[148] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[149] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

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

[151] Alan Wu, AJ Piergiovanni, and Michael S Ryoo. Model-based behavioral cloning with future image similarity learning. In *Conference on Robot Learning*, pages 1062–1077. PMLR, 2020.

[152] Yuan Xu and Hedayat Vatankhah. Simspark: An open source robot simulator developed by the robocup community. In *Robot Soccer World Cup*, pages 632–639. Springer, 2013.

[153] Chao Yang, Xiaojian Ma, Wenbing Huang, Fuchun Sun, Huaping Liu, Junzhou Huang, and Chuang Gan. Imitation learning from observations by minimizing inverse dynamics disagreement. In *Advances in Neural Information Processing Systems*, pages 239–249, 2019.

[154] Zili Yi, Hao Zhang, Ping Tan, and Minglun Gong. Dualgan: Unsupervised dual learning for image-to-image translation. In *Proceedings of the IEEE international conference on computer vision*, pages 2849–2857, 2017.

[155] Ruohan Zhang, Faraz Torabi, Lin Guan, Dana H. Ballard, and Peter Stone. Leveraging human guidance for deep reinforcement learning tasks. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 6339–6346. International Joint Conferences on Artificial Intelligence Organization, 7 2019. doi: 10.24963/ijcai.2019/884. URL https://doi.org/10.24963/ijcai.2019/884.

[156] Ruohan Zhang, Faraz Torabi, Garrett Warnell, and Peter Stone. Recent advances in leveraging human guidance for sequential decision-making tasks. *Autonomous Agents and Multi-Agent Systems*, 35(2):1–39, 2021.

[157] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2881–2890, 2017.

[158] Luowei Zhou, Chenliang Xu, and Jason J Corso. Towards automatic learning of procedures from web instructional videos. *arXiv preprint arXiv:1703.09788*, 2017.

[159] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. *arXiv preprint arXiv:1904.07850*, 2019.

[160] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

[161] Yuke Zhu, Ziyu Wang, Josh Merel, Andrei Rusu, Tom Erez, Serkan Cabi, Saran Tunyasuvunakool, János Kramár, Raia Hadsell, Nando de Freitas, et al. Reinforcement and imitation learning for diverse visuomotor skills. *arXiv preprint arXiv:1802.09564*, 2018.

[162] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

[163] Konrad Zolna, Negar Rostamzadeh, Yoshua Bengio, Sungjin Ahn, and Pedro O Pinheiro. Reinforced imitation learning from observations. *NeurIPS 2018 Workshop*, 2018.