Interactive Learning in State-space

Enabling robots to learn from non-expert humans

Snehal Jauhri

Master Thesis
Embedded Systems
May 14th, 2020

TU Delft

Interactive Learning in State-space

Enabling robots to learn from non-expert humans

by

Snehal Jauhri

to obtain the degree of Master of Science in Embedded Systems
at the Delft University of Technology,
to be defended publicly on Thursday May 14, 2020 at 01:00 PM.

Student number:	4772202	
Project duration:	August 15, 2019 – May 14, 2020	
Thesis committee:	dr. ing. Jens Kober,	TU Delft, supervisor, chair
	dr. Luka Peternel,	TU Delft
	dr. ir. A. J. van Genderen	TU Delft
	dr. Carlos Celemin,	TU Delft, supervisor

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Abstract

Imitation Learning is a powerful technique that enables programming the behavior of agents through demonstration, as opposed to manually engineering behavior. It has been successfully used to train agents in a short amount of time in areas such as computer games, robotic control and motion-planning. However, Imitation Learning methods require demonstration data (in the form of state-action labels) and in many scenarios, the demonstrations can be expensive to obtain or too complex for a demonstrator to execute. This lack or sub-optimality of demonstrations limits the applicability and performance of many Imitation Learning methods.

Advancements in Interactive Imitation Learning techniques however, have made it easier for demonstrators to train agents and improve their performance. These techniques involve demonstrators interacting with and guiding the agent as it performs the requisite task. This guidance is typically in the form of corrections or feedback on the current actions being executed by the agent.

In this work, we propose a novel Interactive Learning technique that uses human corrective feedback in *state-space* to train and improve agent behavior. This technique is beneficial since providing guidance to the agent in terms of 'changing its state' is often easier or more intuitive for the human demonstrator (as opposed to changing the actions being executed). For instance, in manipulation tasks using a robotic arm, it is easier for the demonstrator to provide state information such as the Cartesian position of the end-effector rather than low-level action information such as joint torques. Keeping such scenarios in mind, we propose our method titled: Teaching Imitative Policies in State-space (TIPS). In TIPS, the agent is provided on-line human feedback in terms of modifications to its current state. To perform the instructed state transitions, the requisite actions to be taken are computed by querying a learnt forward dynamics model.

We evaluate the performance of TIPS for various control tasks as part of the OpenAI Gym toolkit as well as for a manipulation task using a KUKA LBR iiwa robotic arm. We show that our state-space feedback mechanism allows non-expert demonstrators to teach agents that achieve high task performance. Moreover, through continuous improvement via feedback, agents trained using TIPS outperform the demonstrator and in-turn outperform conventional Imitation Learning agents.

“Admission of ignorance is often the first step in our education.”

— *Stephen R. Covey*

Acknowledgments

I am immensely grateful to my advisors Dr. Ing. Jens Kober and Dr. Carlos Celemin for giving direction to this thesis project, entertaining my long discussions at meetings, patiently answering all my queries and continuously providing feedback on my work. I would also like to thank them for administering the course titled 'Knowledge Based Control Systems' at TU Delft which introduced me to the exciting world of learning-based robotics and led me to this thesis.

I would like to thank Brinda Mohan for her constant support in keeping me focused as well as proof-reading this document and providing valuable feedback. The paper that this thesis resulted in was made possible by her selfless efforts in understanding the material and helping condense it.

Lastly I would like to thank my family for always supporting me in my endeavors and giving me the opportunity to excel.

Delft, University of Technology — Snehal Jauhri
May 7, 2020

Contents

1 Introduction

1.1. Motivation

We have been promised an autonomous future in which intelligent agents and robots are prevalent in many spheres of our lives and can be tasked with driving, mowing our lawns and even performing surgery. To program agents that can perform such complex and diverse sets of tasks requires developing robust control algorithms [Argall et al., 2009]. Typically, this involves engineers using their understanding of the task to mathematically devise algorithms resulting in the necessary performance. However, this requires considerable expertise and can still be limited by the number of situations considered by the engineer [Kober and Peters, 2009]. Another promising approach however, is to use techniques such as Reinforcement and Imitation Learning that allow behavior to be learnt from experience or from demonstration.

Imitation Learning (IL) involves using example data, gathered during execution of a task by a demonstrator, to 'imitate' behavior. The demonstration data is described in the form of sequential states (or observations) and actions taken in those states. Imitation Learning (IL) is useful since it is often much easier for humans to demonstrate desired behavior as opposed to engineering the robot/agent's behavior [Osa et al., 2018]. Consider the case of automating a task performed by an operator in a factory setting. Using Imitation Learning, the operator (a domain expert for the task) can demonstrate and teach the task to a robot without much knowledge about robotics or control engineering. Such an approach is also important as we are moving into a future where robots work closely with humans in applications such as elderly care, domestic housework etc. With this view, there have also been research efforts to make training via Imitation Learning easier for demonstrators. This is done by allowing demonstrators to interact with and provide corrections/feedback to the agent as it performs the task [Argall et al., 2008; Chernova and Veloso, 2009; Celemin and Ruiz-del Solar, 2019]. Learning using such interactive methods can further increase the applicability of Imitation Learning and often results in better agent performance [Ross et al., 2011].

There are however, limitations to current Imitation and Interactive Imitation Learning techniques. These techniques typically require demonstration or feedback in the action-space of the agent. However, this is different from how humans learn behavior. Humans can often be taught through information about changes in state required for a task, without providing the actions to be taken [Liu et al., 2018]. Moreover, providing demonstration or feedback in the action-space can be difficult for demonstrators. For example, consider a learning agent for a robotic arm manipulation task. Providing action-space information to the agent in the form of angles or torques at each

of the joints requires considerable demonstrator expertise. It would be easier to instead provide state-space information such as the Cartesian position of the end-effector. It is thus beneficial to develop learning methods that do not require action information but can instead utilize feedback in state-space to learn behavior. Devising such a method and **minimizing the required demonstrator expertise in Imitation Learning** is the main focus of this thesis.

1.2. Current Literature

Advancements in Imitation Learning (IL) techniques have led to several successes in learning tasks such as robot locomotion [Zucker et al., 2011], helicopter flight [Abbeel et al., 2010] etc. as well as learning to play games [Silver et al., 2016]. However, such Imitation Learning (IL) methods typically assume the availability of optimal demonstrations. In addition, several Interactive Imitation Learning methods have been developed which enable demonstrators to guide agents by providing corrective action labels [Ross et al., 2011], corrective feedback [Argall et al., 2011; Celemin and Ruiz-del Solar, 2019] or evaluative feedback [Knox and Stone, 2009; Christiano et al., 2017].

In the context of learning using state-space information, there has been some recent research into methods that can learn using states/observations only. These methods learn behavior using recorded state trajectories or videos of task execution by a demonstrator and this technique is known as Imitation from Observation (IfO). Though Imitation from Observation (IfO) methods may not use human interaction, they provide useful insights into learning using state-space information. Recent advances in Imitation from Observation (IfO) have led to some success in learning simulated as well as physical robotic tasks using observations [Nair et al., 2017; Torabi et al., 2018; Liu et al., 2018; Sun et al., 2019].

For this thesis, a survey of current IL and IfO literature was performed, details of which are provided in Chapter 2. Some of the key findings and conclusions of the survey are as follows:

- The use of human interaction and feedback during the learning process has made it easier for demonstrators to guide agents and continuously improve agent behavior. However, further research is required into methods that enable such interactive feedback in state-space.

  Providing corrections or feedback to the agent during learning can increase the applicability and performance of IL [Ross et al., 2011; Chernova and Thomaz, 2014]. To this end, some methods utilize corrections in the action-space [Celemin and Ruiz-del Solar, 2019; Pérez-Dattari et al., 2019], while others use pre-defined or learnt primitives [Argall et al., 2008, 2011] to guide agents. However, learning by providing corrective feedback to the agent in state-space (which is often more intuitive for demonstrators) is still an open problem. Developing such a method could make it easier for non-expert demonstrators to train agents.

- Techniques such as Inverse Dynamics Model (IDM) learning have been used in an Imitation from Observation (IfO) setting to learn from demonstrative state trajectories. Such techniques could be applied in an Interactive Learning setting with state-space feedback.

  Inverse Dynamics Models (IDMs) map state transitions to actions that lead to these transitions. Recent IfO methods have used an estimated IDM to infer actions in order to train agents using state-trajectories only [Nair et al., 2017; Torabi et al., 2018]. Such an approach could also be applied for an Interactive Imitation Learning method in order to compute requisite actions when provided corrections/feedback in state-space.

1.3. Contribution

In this thesis, we aim to devise a viable Interactive Imitation Learning method for training agents that requires minimal demonstrator expertise.

To this end, we propose using a learning framework that utilizes human corrective feedback. This is because providing corrective feedback in the form of adjustments to the current states/actions being executed by the agent is easier for non-experts (as opposed to providing exact state/action labels that should be executed). Moreover, since the action-space is often not conducive for providing demonstration or feedback, we evaluate how feedback provided in state-space can be utilized to train agents.

The main contributions of this thesis are the following:

- **Development of a novel Interactive Imitation Learning method that utilizes corrective feedback in state-space.**

  This thesis proposes and evaluates an Interactive IL method titled: 'Teaching Imitative Policies in State-space (TIPS)'. The method uses state-space corrective feedback (in the form of changes to the current state) to learn behavior. To change the state as per the feedback received, appropriate actions to be taken need to be computed. We propose a mechanism to infer actions using indirect inverse dynamics i.e. by querying a forward dynamics model and predicting the future state.

- **Display of improved performance upon current IL methods in sub-optimal demonstration scenarios.**

  Our proposed method 'Teaching Imitative Policies in State-space (TIPS)' was evaluated and compared with IL methods such as Behavioral Cloning (BC) and Generative Adversarial Imitation Learning (GAIL). Through continuous improvement, TIPS agents outperform the demonstrator as well as IL agents for benchmark OpenAI Gym tasks [Brockman et al., 2016] in scenarios where demonstrations are sub-optimal. To validate the method for real world application, experiments were also run on a KUKA LBR iiwa Robotic arm for learning a manipulation task.

1.4. Thesis Outline

This thesis report is structured in the following manner:

In Chapter 2 a detailed background of Imitation Learning (IL) and Imitation from Observation (IfO) is provided. Firstly, some preliminary information is provided about how Imitation Learning problems are formulated and mathematically represented. We then elaborate on Interactive Imitation Learning methods and their advantages. This is followed by a discussion on current IfO literature.

In Chapter 3, we explain the proposed method: 'Teaching Imitative Policies in State-space (TIPS)'. The learning framework as well as the technique used to infer actions from state-space feedback are elaborated.

In Chapter 4 we elaborate on the experimental environment used to evaluate our method. This includes explanation of the Open AI Gym tasks [Brockman et al., 2016] as well as the robotic manipulation task.

In Chapter 5 we provide the results of our experiments and delineate the performance improvements provided by TIPS.

In Chapter 6, some conclusions and possible future improvements to this work are provided.

Appendix A consists of a draft version of the paper that this thesis work resulted in. The paper will be submitted to the Conference on Robot Learning (CoRL) 2020.

2

Background and Related Work

This chapter is meant to provide background information about Imitation Learning (IL) and the different approaches to solving the IL and Imitation from Observation (IfO) problems. Moreover, as a precursor to the method proposed in this thesis, a discussion on current Interactive IL methods is presented. We aim to provide a broad view of different types of IL and IfO techniques and discuss their advantages/disadvantages. Throughout this discussion an emphasis is placed on practical, real-world robotic applications.

In section 2.1, preliminary information about IL is provided, followed by details about two of the most popular IL approaches, namely, Behavioral Cloning (BC) and Inverse Reinforcement Learning (IRL).

Section 2.2 elaborates on Interactive Imitation Learning i.e. methods that utilize interaction with a demonstrator during learning to accelerate learning or improve performance.

Section 2.3 delves into Imitation from Observation (IfO) and the types of methods used in this relatively new field. Key techniques in IfO such as Inverse Dynamics Model (IDM) learning are delineated.

In section 2.4, we summarize the chapter and offer a concluding hypothesis on improving Interactive Imitation Learning (IL) methods and minimizing the required demonstrator expertise.

2.1. Imitation Learning (IL)

Imitation Learning (IL) is a machine learning technique through which an agent can learn to perform a task using example demonstrations of the task [Schaal, 1997; Daumé III, 2012]. The agent is trained to perform actions that lead to similar behavior as seen in the example demonstrations [Argall et al., 2009; Torabi et al., 2019].

In the context of learning techniques, IL offers some key advantages. Firstly, IL significantly reduces the search space for learning using the provided desirable examples (in contrast to most Reinforcement Learning (RL) settings where the entire state-action space needs to be searched) [Billard et al., 2008]. Furthermore, it eliminates the need for humans to pre-program the required behavior for a task, instead utilizing the sometimes easier or more intuitive mechanism of demonstrating the task [Osa et al., 2018].

IL also provides some useful features when applied to the field of robotics [Argall et al., 2009]. For instance, instead of the tedious tuning of multiple control parameters, robots can be taught motor skills through interaction with a human demonstrator [Kober and Peters, 2009]. Additionally, IL offers a lot of flexibility in terms of the tasks that the robot can be taught to perform [Billard and Matarić, 2001].

2.1.1. Preliminaries

Some preliminary information about the IL problem and its framework is presented in this subsection. We also provide necessary background about Reinforcement Learning (RL) since it goes hand-in-hand with many IL methods we will elaborate.

2.1.1.1. Learning framework:

Imitation learning problems are defined using the following elements (as described by Osa et al. [2018] and Sutton and Barto [1998]):

Environment This is the scenario or setting in which the necessary task needs to be performed. For example, in a self-driving car application, the environment consists of the dynamics of the car as well as the real world the car is in. A situation in the environment can be captured by it's current state $s \in S$. The dynamics of the environment may be deterministic or stochastic.

Agent A learning agent interacts with the environment to achieve a desired goal. The agent must have the ability to sense the environment it is in, either directly in the form of the state s (full observability) or through observation o (partial observability). Moreover, the agent can take an action $a \in A$ to transition from the current state at time t i.e. s_t to the next state s_{t+1}.

Policy A policy π defines the agent's behavior (actions to be taken) in the environment. This is essentially what we aim to learn in our IL or RL setting. It maps the state in which an agent is to an action to be taken in that state i.e. a function $\pi : S \rightarrow A$ (Note that in the case of partial observability, the policy can instead be an observation to action mapping). The policy could be deterministic (always choosing the same action in a given state) or stochastic.

Demonstrations These are example executions of the task. This information is typically provided in the form of a sequence of state-action pairs that were observed during task execution by a demonstrator [Argall et al., 2009]. We denote such a trajectory over time T as: $\tau = \{(s_0, a_0), ...(s_T, a_T)\}$ and the total demonstration data as a set of these trajectories: $D = \{(\tau_i)\}_{i=1}^{N}$.

The aim of IL is to learn a policy $\pi(s)$ that leads to similar behavior as seen in the demonstrations D during the policy's 'roll-out' (an execution over time). A common approach is to consider this as an optimization problem to find the optimal policy π^* [Osa et al., 2018]:

$$\pi^* = \arg\ \min\ D(q(\phi), p(\phi)), \tag{2.1}$$

where ϕ represents features of the trajectory (typically state or observation trajectories), $q(\phi)$ and $p(\phi)$ represent the distribution of features produced by executing the demonstrator and learner policies respectively, and $D(q,p)$ is a dissimilarity measure between q and p.

While standard IL methods assume demonstrations as state-action pairs, there exist other IL formulations with different forms of demonstrative information. Moreover, in this survey we will also consider scenarios where demonstrator feedback is used while learning. Thus we additionally define the following **types of demonstrative information**:

- Observations: These are observations of the example execution of the task. This is typically in the form of a sequence of observations that are recorded as the task was executed. Observations can vary from state-trajectories of the system $\tau = \{(s_0)..(s_T)\}$ (in a full observability scenario) to videos i.e. sequences of images of the task execution.
- Corrective Labels: These are action labels that are provided by the demonstrator *during* the learning phase in order to correct the agent's behavior. These action labels can be appended to the current state of the agent thus enabling addition of state-action demonstrations on-the-fly.
- Corrective Feedback: Demonstrator feedback in the form of adjustments to the current states/actions being executed by the agent. This type of information is useful when the demonstrator cannot provide exact labels but can give information related to how states/actions can be modified.
- Evaluative Feedback: This is feedback in the form of qualitative evaluation of the current behavior. This is useful when the demonstrator cannot provide any information about exact states/actions but can give qualitative advice or provide preferences between different behaviors.

IL using observations is discussed in detail in section 2.3. Details about IL using the other mentioned types of demonstrative information are provided in section 2.2 of this chapter.

2.1.1.2. Reinforcement Learning (RL)

Reinforcement Learning (RL) methods involve agents interacting with the environment over time to learn the desired policy. Thus, in contrast to IL, in the RL paradigm agents learn from their own experience rather than examples [Argall et al., 2009]. RL systems consist of the following additional elements [Sutton and Barto, 1998]:

Reward A reward value or signal $r \in R$ is used to define the goal of the agent. Upon interacting with the environment, the agent receives these rewards (represented by a function $r : S \times A \rightarrow R$). Rewards can also be negative to represent undesirable events. The maximization of total reward received over time ($\sum r_t$) (also known as the *return*) becomes the basis for choosing the policy. Additionally, a *discount factor* γ can be used to reduce the weight of future rewards when calculating the return ($\sum \gamma^t r_t$).

Markov Decision Process (MDP) The RL problem is typically modeled as a decision process (choosing actions that maximize total reward). Moreover, the process is assumed to obey the *Markov property* and hence called a Markov Decision Process (MDP). In simple terms, the Markov

property implies that a future state s_{t+1} is dependent on the history of states only through the current state s_t [Serfozo, 2009]. We define an MDP by the tuple: (S, A, P, r, γ), where S and A are the state and action spaces, $P(s_{t+1}|s_t, a_t)$ represents the transition probability i.e. the probability of reaching state s_{t+1} from s_t after executing action a_t, r is the reward function and γ is the discount factor.

The goal in RL is to learn a policy that maximizes the expected total reward over time i.e. the *return*. Thus we can write an objective function J of the form [Osa et al., 2018]:

$$J(\pi) = \mathbb{E}\left[\sum_{t=0}^{T} r_t \middle| \pi\right]. \tag{2.2}$$

So, we can now write the RL optimization problem to find the desired policy π^* as:

$$\pi^* = \arg\ \max\ J(\pi). \tag{2.3}$$

Typically, reward functions are defined by experienced programmers even though for the agent, we consider these as coming from the environment. The design of the reward function greatly affects the performance of a RL method. For instance, if the rewards in an environment are sparse, the agent may never encounter a state that leads to a reward, thus never learning any useful policy. **Reward shaping** is the concept of choosing the right reward function that guides the agent in a controlled way [Grzes and Kudenko, 2009].

RL agents use the rewards obtained to learn the policy and visit the state-action space that maximizes return. However, the reward obtained at any time-step can be due to multiple actions that were taken at previous time-steps. It can thus be challenging to assign credit for the reward to an appropriate past state-action pair. This is known as the **credit assignment** problem and is another key challenge in RL problems.

Since RL methods involve some 'trial and error', another key consideration is the strategy around trials. Choosing actions known to provide maximum return is termed *exploitation* of the knowledge of these actions. Conversely, choosing actions with unknown return is known as *exploration*. Exploration helps improve the knowledge of actions and in the long run may lead to higher return [Sutton and Barto, 1998]. However, increased exploration comes at the cost of an increased number of trials required for learning. Moreover, exploring all possible state-actions is typically infeasible in most environments. This inherent trade-off in RL is known as the **exploration vs. exploitation** trade-off.

More details about RL methods and solving MDPs can be found in [Sutton and Barto, 1998]. Specific to robotics, a detailed survey is also provided by Kober et al. [2013].

2.1.2. Behavioral Cloning (BC)

A direct approach to learning a policy in IL is to leverage supervised learning [Shalev-Shwartz and Ben-David, 2014]. Treating policy learning as a regression problem, the example demonstrations can be used to build a state to action mapping [Osa et al., 2018]. This method essentially clones the demonstrated behavior and is known as Behavioral Cloning (BC) [Bain and Sammut, 1995; Ross and Bagnell, 2010] .

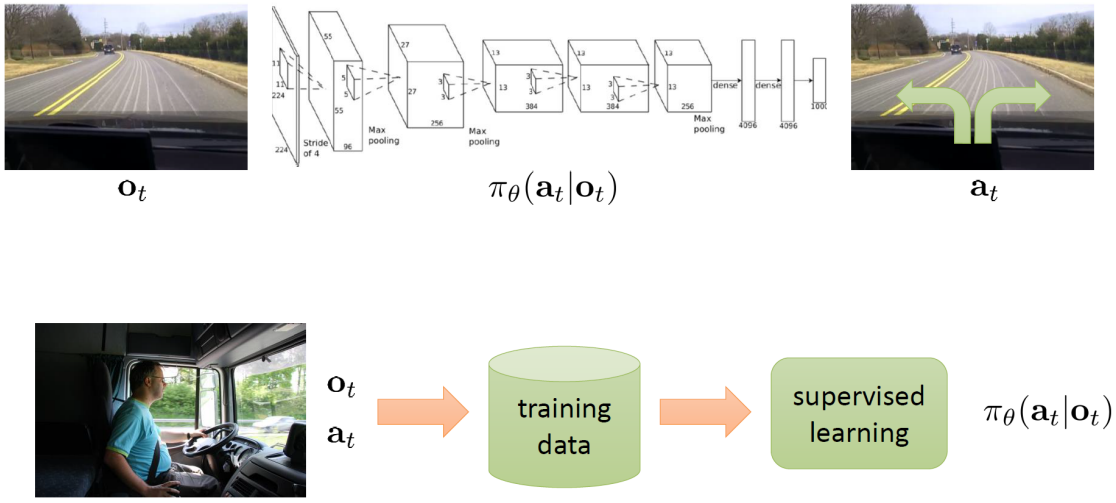


Figure 2.1: An example behavioral cloning strategy for a driving agent. The observations $\boldsymbol{o_t}$ (images of the road) and the actions $\boldsymbol{a_t}$ (steering commands) demonstrated by the human are used to learn the policy $\boldsymbol{\pi_\theta}$. From Levine [2018] and Bojarski et al. [2016].

Regression technique The choice of regression technique is one of the main considerations in such a supervised learning approach. This includes the representation function for the policy and its training method. A chosen representation function must be able to capture the complexity and non-linearity of the specific problem [Osa et al., 2018]. However, a more complex function requires a larger amount of data for training. It is also useful to have methods that can capture uncertainty in demonstrations as well as the policy. For instance, Gaussian Mixture Models (GMMs) can explicitly model uncertainty and have been used to teach robots manipulation skills through Gaussian Mixture Regression (GMR) [Calinon and Billard, 2009; Gribovskaya et al., 2011]. One of the most typical representations used in IL however are artificial neural networks. In the field of robotics, neural network based BC methods have been used for autonomous driving [Bojarski et al., 2016] and quad-rotor control [Giusti et al., 2015]. Moreover, different architectures of neural networks such as Recurrent Neural Networks (RNNs) can capture the entire history of states/observations in the demonstrations. The usage of RNNs along with Long short-term memory units (LSTMs) [Hochreiter and Schmidhuber, 1997] has shown promise in imitating robot manipulation tasks [Rahmatizadeh et al., 2018].

Model-free vs. Model-learning Another consideration in BC is whether to create a model of the system dynamics. Based on this, BC methods can be divided into model-free and model-learning. Model-free methods attempt to directly map observations to actions without modeling the system dynamics. This can lead to a simpler learning process with fewer iterations [Osa et al., 2018]. However, in some cases this may lead to trajectories being learnt that are infeasible to execute since they violate the physical constraints of the system. For instance, consider a case where the embodiment of the demonstrator is different from the learner and the demonstrated trajectory needs to be executed at a higher velocity. If the learner's system is under-actuated (Eg. An under-actuated robotic manipulator), executing the policy at this velocity may not be possible [Billard et al., 2008]. Model-learning BC methods on the other hand do not suffer from this issue since they involve learning a dynamics model of the system. Though this requires a larger number of learning iterations, it

ensures that the learnt trajectory is both feasible and close to the demonstrated trajectories. Such BC techniques have been used to great effect for learning manipulation tasks using representations such as Gaussian Mixture Models (GMMs) [Grimes et al., 2007] and neural networks [Nair et al., 2017].

The BC approach has some inherent challenges and drawbacks. Firstly, in a supervised learning formulation, the training and test data need to be independent and identically distributed. This does not hold true in an IL setting where we need to learn a policy for sequential decision making [Osa et al., 2018]. Another problem is that supervised learning typically **requires significant training data**. Obtaining demonstration data might be very expensive, especially for robotics applications, and hence there is the need to develop less data hungry IL methods. Some of these will be discussed in section 2.2.

Finally, arguably the biggest drawback of BC is that it fails to learn appropriate actions in states that were not visited during the demonstrations. This is known as the **covariate shift** problem where the source distribution (demonstrator's states) is different from the target distribution (learner's visited states) [Ross and Bagnell, 2010; Osa et al., 2018]. This is a major issue since it leads to compounding errors. During execution, if an agent reaches a state where the correct action is not known, there is a high chance of again reaching such states in the future leading to a large trajectory deviation [Bagnell, 2015; Ross and Bagnell, 2010]. To solve this problem, one common approach is to use a combination of IL and RL [Nair et al., 2018; Hester et al., 2018] thus utilizing exploration and reward strategies in unknown states. Another approach is to leverage human interaction and feedback which we will elaborate in section 2.2.

2.1.3. Inverse Reinforcement Learning (IRL)

In Inverse Reinforcement Learning (IRL) [Russell, 1998; Ng et al., 2000], the provided demonstrations are used to infer a reward function which can then be used to learn an optimal policy using RL. It is assumed that the demonstrator's behavior is based on maximizing a reward function and thus the main aim is to estimate this reward function. Conversely, this can also be thought of as learning a cost function that penalizes behavior that is dissimilar to the demonstrator. IRL has been successfully used to learn many tasks from robot locomotion [Zucker et al., 2011] to helicopter flight [Abbeel et al., 2010].

IRL involves two key steps: (i) **reward function estimation** and (ii) **policy optimization** based on the reward function. In the former step, a reward function is chosen that measures the difference between relevant features of the demonstrator and learning agent's behavior (similar to Eq. 2.1) and penalizes this difference. In the latter step, an optimum learner policy is chosen using this reward function in a RL fashion. For instance, let us say that task execution by the demonstrator and the agent policy leads to distributions of state-action trajectories $q(\tau)$ and $p(\tau)$ respectively. A suitable reward function is such that it leads to an agent policy being learnt (in the policy optimization step) that minimizes the difference between these distributions. In order to estimate the agent policy's trajectory distribution $p(\tau)$, some methods utilize a model of the system dynamics (model-based) [Abbeel and Ng, 2004; Levine et al., 2011] while other methods use sampling-based approaches (model-free) [Finn et al., 2016; Ho and Ermon, 2016].

One drawback of the IRL approach is that reward estimation typically needs to be done iteratively with policy optimization for every reward function (RL) running in an 'inner loop' [Torabi et al., 2019]. This can be highly **computationally expensive**. However, some methods try to avoid

this pitfall using ideas such as solving an approximate local control problem [Levine and Koltun, 2012] or by instead performing policy optimization with reward estimation in-the-loop [Finn et al., 2016].

Uniqueness of Reward Function Finding a unique solution to the IRL problem is a challenge since there can be many reward functions that lead to the same optimal policy and state-action trajectory [Osa et al., 2018]. One solution to this problem is to choose a reward function that additionally maximizes a margin between the optimal trajectory and others in the agent's trajectory distribution [Ratliff et al., 2006; Ng et al., 2000]. This typically works well in case where one reward function is considerably better than the others [Osa et al., 2018]. However, a more general and popular solution is to use the *maximum entropy principle* [Jaynes, 1957]. As per this principle, in the case where multiple trajectory distributions can be considered equivalent, the distribution which maximizes the entropy should be chosen [Ziebart et al., 2008]. In this manner, the ambiguity is resolved by choosing the distribution that does not exhibit any additional bias beyond matching the demonstrator's distribution [Ziebart et al., 2008].

The maximum entropy method has been used successfully to estimate unique IRL solutions for tasks such as highway driving [Levine et al., 2011] and robotic manipulation [Finn et al., 2016]. Another popular entropy based IRL approach, proposed by Ho and Ermon [2016] is GAIL. This method uses a Generative Adversarial Network (GAN) architecture [Goodfellow et al., 2014] in which the reward/cost function acts as a *discriminator* that distinguishes the trajectory distributions of the demonstrator and the learner, while the RL optimization acts as the *generator* and provides candidate distributions. Thus, direct policy optimization (generation) is done while the reward function is implicitly represented by the discriminator network. GAIL and its extensions [Henderson et al., 2018] have been used to learn relatively complex tasks simulated in OpenAI Gym [Brockman et al., 2016] and achieve high performance, highlighting the promise of the adversarial learning approach. However, a drawback of such methods is that they require a relatively large number of environment interactions during learning.

When compared to Behavioral Cloning, a drawback of IRL stems from the fact that it requires an RL step. In the absence of a model, this necessitates interaction with the environment which can be expensive (in terms of time) and is also unsafe for many applications (Eg. Self-driving vehicles). Another limitation, even in model-based IRL, is that the reward function is estimated under an assumption that the provided demonstrations are optimal. When this assumption does not hold, the learnt behavior can be sub-optimal and may not improve over time (unlike standard RL).

2.2. Interactive Imitation Learning

A typical drawback of agents trained using IL is the compounding error problem [Bagnell, 2015]. Due to small errors (especially in stochastic environments), agents will inevitably reach states that were not in the demonstrations. This leads to further mistakes and eventually causes significant deviation from the demonstrated trajectory [Osa et al., 2018; Ross et al., 2011]. Moreover, since providing demonstration data to cover all possible situations encountered by the agent is infeasible or highly expensive, there is a need for alternative methods that provide information to the agent. One way to do this is to use interaction and feedback during learning to guide the agent as it executes its policy. This can be considered as an 'on-policy' approach to IL [Laskey et al., 2017] or Interactive Machine Learning (IML) [Chernova and Thomaz, 2014] and in this section we discuss some of the methods in this domain.

2.2.1. Corrective Labels

Pomerleau [1989] argues that it is insufficient to provide the agent example demonstrations of a task. Instead, it must also be taught recovery actions to correct its errors. One way to achieve this is to have demonstrators provide **corrective action labels** to IL agents as the policy is executed.

Interactive learning methods can further be classified as *passive* or *active*. Passive learning techniques require the demonstrator to observe the agent's behavior and provide corrections when necessary [Ross et al., 2011]. On the other hand, in active learning, the agent explicitly requests the demonstrator for input, typically at times when the agent lacks knowledge of the correct action to be taken in its current state. For example, a successful active learning method in this domain is proposed by Chernova and Veloso [2009] who utilize a confidence measure of the agent based on its state. Upon visiting states where the agent's confidence is low, the agent requests a human expert to provide correct actions which then get added to the training data. Such active techniques are aimed at reducing the demonstrator's observation effort and ensuring safe operation. However, a drawback of such an approach is that it requires the human demonstrator to be constantly attentive.

Dataset Aggregation (DAgger) One of the most popular methods utilizing corrective labels is the DAgger algorithm proposed by Ross et al. [2011]. In this method, the agent is allowed to execute its IL policy while keeping an expert demonstrator in the loop. Initially, the agent is trained using an IL algorithm such as Behavioral Cloning (BC). The agent then executes its policy while the demonstrator simultaneously provides action labels that the agent should have taken in the visited states. These state-action labels are added to the training dataset and the BC step is carried out again (data aggregation). The entire process is iterated until necessary performance is obtained. This method has been shown to significantly improve the learning of tasks such as steering a cart [Ross et al., 2011].

DAgger surpasses BC in terms of performance since it **solves the distribution mismatch problem** between the states in the learner's and demonstrator's trajectories [Ross and Bagnell, 2010]. Unlike BC, when the agent makes mistakes and visits states in the 'neighborhood' of the initial demonstrated trajectory, the correct recovery actions are added to its dataset and subsequently learnt. Asymptotically, with enough training data accumulated over iterations, DAgger has been shown to achieve the same expected error bound as supervised learning [Ross et al., 2011].

The main obstacle in implementing DAgger is that it can be expensive to query the demonstrator a large number of times. This is especially the case with human demonstrators. The demonstrators also need to be experts in that they know the correct actions in every state. This might not always be feasible. Moreover, since the agent makes mistakes during execution, it raises safety concerns in many applications such as driving. However, some of these problems can be alleviated by using a confidence measure of the agent and only querying the human when this confidence is below a threshold [Zhang and Cho, 2017; Menda et al., 2018].

2.2.2. Corrective Feedback

Methods that leverage demonstration data (IL) and corrective labels (DAgger) can still be insufficient for some problems such as when the embodiment of the learner is different than that of the demonstrator [Argall et al., 2011]. Furthermore, the requirement of visiting a state in order to receive information about which action to perform in it is a bottleneck, especially if the environment consists of unreachable or unsafe states. In these situations, it can be beneficial to use an alternate means of human interaction: corrective feedback. In this setting, the human provides feedback in

the form of ***modifications* or *adjustments* to the current agent behavior**. This type of interaction is also desirable since it enables teaching by non-expert or sub-optimal human demonstrators who may not be able to provide accurate state/action labels, but can still provide feedback on how to *modify* the states/actions.

In Argall et al. [2008], the *advice-operator* paradigm for corrective feedback is introduced. In this setting the human provides *advice* on what *operation* should be performed on the current state-action data point (typically a state/action modification). With this operation, a new data-point is generated that gets added to the demonstration dataset. Thus, an initial policy learnt from demonstration can be improved upon with the updated dataset [Argall et al., 2008]. This feedback technique is especially advantageous in problems with continuous action spaces since it is unreasonable to always expect the demonstrator to provide accurate continuous valued actions. The technique has been extended by Argall et al. [2011] who use it to learn motion primitives. In their method titled 'Feedback for Policy Scaffolding (FPS)', corrective feedback is used to learn primitive policies. These primitives can then be used to learn more complex policies for whom demonstration may be difficult. The advice-operator framework has been used to learn simulated tasks such as driving on a racetrack Argall et al. [2011] as well as motion control task for Segway robots [Argall et al., 2008].

A limitation of the advice-operator approach is that devising the operators requires significant prior knowledge about the environment as well as the task to be performed. Moreover, another drawback is that the effect of the corrections is only seen after the policy is re-derived with the new data. This means that until the policy is updated, the demonstrator may still observe the agent's incorrect behavior and provide further corrections that are harmful or contradictory [Celemin and Ruiz-del Solar, 2019].

COACH (COrrective Advice Communicated by Humans) An important method that uses human advice is COACH (COrrective Advice Communicated by Humans), proposed by Celemin and Ruiz-del Solar [2019]. Aimed at problems with continuous action spaces, this method uses feedback (h_t) in the form of a binary signal implying an increase/decrease in value of an action (i.e. $h_t \in \{-1, 0, +1\}$). Moreover, during execution, a human feedback model is learnt that predicts the magnitude of this correction. The intuition is that if a sequence of corrections provided in a state are in the same direction (increase/decrease), the demonstrator is suggesting a larger magnitude correction. Conversely, if the feedback alternates between increase/decrease, the demonstrator is suggesting a smaller change around a set-point [Celemin and Ruiz-del Solar, 2019]. This helps resolve the ambiguity around the magnitude of correction. Another feature of COACH is that the corrected action is executed immediately by the agent, thus making it more intuitive for non-expert demonstrators to observe the behavior and provide further corrections. The method has been used to successfully learn tasks such as a 'Reacher' (with a 3DoF robot) as well as ball-dribbling (with a humanoid robot) even when trained by non-expert humans who make occasional mistakes.

Pérez-Dattari et al. [2019] have subsequently proposed an extension of COACH to specifically work with policies that are represented using neural networks. In their method called Deep-COACH (D-COACH), the human feedback model of COACH is replaced by a demonstration replay mechanism. The demonstration replay memory stores previously computed state-action pairs based on previous feedback received. Updation of the agent's policy is done using samples from this memory (similar to experience replay [Lin, 1992]). The replay memory mechanism thus resolves the ambiguity around the magnitude of correction by considering prior corrections. Moreover, it also helps avoid the catastrophic forgetting problem [French, 1999] commonly observed in learning neural network representations. To ensure sufficient learning iterations to train the neural network, a periodic training step is also taken. The method has been used to learn tasks such as car racing using

a Duckietown robot as well as 'reacher' and 'pusher' tasks using a 3-DoF robot manipulator. The pseudo-code of D-COACH is shown in Algorithm 1.

Algorithm 1: D-COACH [Pérez-Dattari et al., 2019]

Require: Error constant: e, Periodic update interval: T_{update}
Initialize: Replay buffer $B = []$
for *time = 1,2,....* **do**
  observe state s_t
  execute action $a_t = \pi(s_t)$
  get human corrective **feedback** h_t
  if h_t *is not* ***0*** **then**
    $error_t = h_t.e$
    $a_{label(t)} = a_t + error_t$
    update policy π using pair $(s_t, a_{label(t)})$
    update policy π using a batch sampled from replay buffer B
    append $(s_t, a_{label(t)})$ to buffer B
  end
  if $mod(t, T_{update})$ **then**
    update policy π using a batch sampled from replay buffer B
  end
end

2.2.3. Evaluative Feedback

The final type of interaction we will discuss is in the form of qualitative evaluation i.e. evaluative feedback. This kind of interaction is especially useful in cases where non-expert humans are unable to demonstrate or correct behavior but can evaluate and specify whether the observed behavior (states/actions) are good or bad. This type of feedback is typically either *reward-based* [Knox and Stone, 2009; Griffith et al., 2013] or *preference-based* [Akrour et al., 2011; Christiano et al., 2017].

In **reward-based** evaluative feedback methods, demonstrator feedback is in the form of scalar reward signals indicating desirable/undesirable behavior. An important work in this domain is TAMER (Training an Agent Manually via Evaluative Reinforcement) [Knox and Stone, 2009]. In the TAMER framework, human feedback is modeled as $H : S \times A \rightarrow R$ (mapping from state-action space to reward). During task execution, in addition to the existing RL reward function, the agent also queries the model (H) to receive 'human feedback reward'. This is then used to modify the policy and has been used to learn tasks such as mountain-car and later for robot navigation [Knox et al., 2013]. In other works by Suay and Chernova [2011] and Najar et al. [2016], this type of human evaluation and guidance is used to teach a sorting task to a robotic manipulator. A drawback of the reward-based feedback approach however, is the sensitivity to relative values of the human feedback reward signals. Another limitation is that a relatively large amount of feedback data may be required before correct actions can be learnt. Thus, the approach does not scale well to problems with large state-action spaces [Suay and Chernova, 2011].

Preference-based evaluative feedback methods elicit preferences between multiple behaviors or policies and then use these to build scoring/reward functions. The idea is that policies with a higher rank/preference result in a higher reward and with a large enough number of preferences obtained, a suitable reward function can be estimated. This approach is advantageous since it requires less teacher expertise as compared to providing accurate reward signals. The preference-

based feedback technique has been used to learn motion control tasks for humanoid Nao robots [Akrour et al., 2014] as well as robotic manipulators [Jain et al., 2013]. Moreover, for learning in high dimensional state-spaces such as images, Christiano et al. [2017] propose an extension that leverages deep neural networks. The method uses feedback in the form of preferences between video clips of trajectories, and has been successfully applied to simulated OpenAI Gym [Brockman et al., 2016] tasks such as Hopper. However, a limitation of preference-based feedback methods is that in cases where multiple sub-optimal policies are presented, providing a preference becomes ambiguous. Another drawback is that to converge to an optimal policy, a large number of preferences may need to be elicited.

2.3. Imitation from Observation (IfO)

Though Imitation Learning (IL) has proved successful for learning robotic skills, the required demonstrations can be expensive to obtain or too complex for a demonstrator to execute. This can limit the applicability of IL. Thus there has been recent interest in methods that learn by utilizing existing resources such as videos of humans performing the task. This technique of Imitation Learning using only observations or states is termed as Imitation from Observation (IfO).

Imitation from Observation (IfO) is more representative of how humans learn tasks [Liu et al., 2018]. Humans are able to imitate tasks by just observing their execution by a demonstrator without requiring exact action labels. To this end, the goal in IfO is to learn a policy using only observations which can range from state trajectories (in full-state observability scenarios) to high dimensional camera images/videos. In the case of demonstrative videos, difficulties can arise due to variations in embodiment or viewpoint. For instance, a robot might be provided with a video demonstration of a human performing the task which may also be in a different context (Eg. different environment). This is known as the embodiment mismatch problem and is a key challenge in imitation from videos.

In this section, we elaborate various types of successful IfO methods. A typical approach in IfO is to learn an Inverse Dynamics Model (IDM) through which actions can be inferred from the provided state/observation trajectories (section 2.3.1). Alternatively, many methods use the provided observations to build a reward function and then learn a policy in an IRL fashion (section 2.3.2).

2.3.1. Inverse Dynamics Model (IDM)-learning

An Inverse Dynamics Model (IDM) maps states transitions to the actions that produce the state transitions [Hanna and Stone, 2017]. The model can be written as a function f_{inv} such that:

$$f_{inv}(s_t, s_{t+1}) = a_t, \tag{2.4}$$

where s_t is the current state, s_{t+1} is the next state and a_t is the action that leads to the transition from s_t to s_{t+1}. Such models are in principle the inverse of 'forward' dynamics models which map state-action pairs to their resulting subsequent states ($f(s_t, a_t) = s_{t+1}$). Since in an IfO problem the desired state trajectory is already known (or obtained using the observations), the appropriate action labels can be computed through the IDM. Thus if an accurate IDM is estimated, it **can reduce the IfO problem to an IL problem**.

We note that in a model-based problem setting (i.e. when the forward dynamics model is already known), inferring the action labels for imitation is further simplified. Using the forward model and demonstrated state trajectories, a minimum variance and bias estimation technique can be

used to compute the actions [Gillijns and De Moor, 2007]. This has been successfully demonstrated by Curi et al. [2018] who use an adaptive linear estimator to compute actions and then use Behavioral Cloning (BC) for imitation. However, in this discussion, we will consider scenarios where the forward model is not already known.

For learning the IDM, a typical approach is to sample state-action trajectories and then fit model parameters through regression. A consideration when sampling is that the data acquired is sufficiently rich i.e. sufficiently captures the dynamics of the system. To this end, some methods acquire samples while executing pre-defined trajectories while others may use randomized execution of actions (typically requiring a larger number of samples to cover sufficient state-action space). For model representation and regression, there exist many techniques. For instance, Gaussian Process Regression can be used to learn the parameters of a model represented using Gaussian kernels [Nguyen-Tuong et al., 2008]. Moreover, neural network based models, trained on random execution samples, are popular among IfO methods [Torabi et al., 2018; Nair et al., 2017].

For the remainder of this subsection we will discuss various IfO methods that use the IDM-learning approach:

Behavioral Cloning from Observation (BCO) Proposed by Torabi et al. [2018], BCO is an important IfO method that uses the IDM-learning approach. The method extends Behavioral Cloning (BC) by learning an IDM in two stages:

In the first *pre-demonstration* stage, the agent is allowed to randomly execute actions in it's environment and record the states visited. Through these samples of state-trajectories (T) and actions (A) generated, the IDM (M_θ) is learnt via supervised learning (in Torabi et al. [2018] the model is represented using a neural net).

The second *post-demonstration* stage further involves learning both the policy (π_ϕ) and IDM in-the-loop. The policy is learnt in a BC fashion using state-action pairs. The states are selected from the provided state-trajectory demonstrations while actions for these states are inferred using the currently learnt IDM. Crucially, in the loop, the agent rolls out its current policy and the actual states visited are again sampled. These samples are then used to again learn and improve the IDM and in turn, the policy. Thus, BCO iteratively learns an accurate IDM and the appropriate policy for imitating the demonstrations. The pseudo-code of the method is provided in Algorithm 2.

BCO has shown promising results for simulated OpenAI Gym tasks such as 'Reacher' and 'Ant' [Brockman et al., 2016]. The method converges to a good model and policy that lead to high returns even when the state space is high dimensional [Torabi et al., 2018] (Eg. The state space of the 'Ant' task in OpenAI Gym has over a hundred dimensions). Although, unlike standard Behavioral Cloning, BCO requires a significant amount of environment interactions during the pre and post-demo iterations, the total number of interactions are relatively fewer as compared to many other methods that utilize RL [Torabi et al., 2018].

A drawback of the BCO method is that the significant number of environment interactions immediately makes it infeasible for applications where such interaction is expensive or unsafe. Moreover, the pre-demonstration stage involves completely random interaction with the environment. Due to this, learning can be quite sensitive to the number of iterations initially trained on (I^{pre}) since the agent needs to explore its entire state-action space through random interactions to learn a reasonable initial IDM. Another demerit is that BCO suffers from the same problems as Behavioral Cloning (BC) such as distribution-mismatch between the states observed by the agent and the demonstrations. A possible solution to this problem however, is to combine the method with RL

Algorithm 2: Behavioral Cloning from Observation (BCO) [Torabi et al., 2018]

Initialize the model M_θ (random)
Initialize the policy π_ϕ (random)
Set *iterations* = I^{pre}
while *policy improvement* **do**
  for *iterations* **do**
    Generate samples $\left(s^a_t, s^a_{t+1}\right)$ and (a_t) using π_ϕ
    Append samples $\mathrm{T}^a_{\pi_\phi} \leftarrow \left(s^a_t, s^a_{t+1}\right), A_{\pi_\phi} \leftarrow (a_t)$
  end
  Improve M_θ by model learning $\left(\mathrm{T}^a_{\pi_\phi}, A_{\pi_\phi}\right)$
  Select a set of state transitions $\mathrm{T}^a_{\text{demo}}$ from the demonstrated state trajectories
  Use M_θ with $\mathrm{T}^a_{\text{demo}}$ to infer actions A_{demo}
  Improve π_ϕ by behavioral cloning $(S_{\text{demo}}, A_{\text{demo}})$
  Set *iterations* = I^{post}
end

[Pavse et al., 2019; Guo et al., 2019]. The additional reward function helps the agent learn optimal policy actions even for the states not present in the demonstrations.

IDM learning has also led to some success in the context of imitation from videos. For instance, Nair et al. [2017] propose a **vision-based self-supervised method for IDM learning**. The agent (robot) is allowed to physically interact with the environment and self-supervise its learning of a Convolutional Neural Network (CNN) based IDM that maps the image sequences observed to actions. This is successfully applied to the challenging problem of manipulation of a deformable object (in the mentioned case, a rope). A limitation of this method is that it suffers from the embodiment-mismatch problem generally observed in IfO. The robot fails to learn a good policy when there are differences in the background of the environment or object being manipulated. Furthermore, training the CNN through interaction can be very costly in terms of robot learning hours (over 60K interactions in the mentioned case).

Though IDM learning is useful in practice, Sun et al. [2019] argue that the approach is 'ill-defined'. From a probabilistic viewpoint, the IDM can be denoted as $P(a|s_t, s_{t+1})$. Using Bayes rule, we can say that $P(a|s_t, s_{t+1}) \propto P(s_{t+1}|s_t, a)P(a|s_t)$. This means that the IDM, by definition, is dependent on the policy $P(a|s_t)$ (or $\pi(s_t)$). Thus, to learn an IDM that predicts the correct actions that lead to the demonstrator's state trajectory, we need to estimate $P^*(a|s_t, s_{t+1}) \propto P(s_{t+1}|s_t, a)\pi^*(s_t)$, which requires sampling actions from the optimal policy π^* [Sun et al., 2019]. Since we don't have access to π^*, there are no guarantees on learning $P^*(a|s_t, s_{t+1})$. Thus, IDM learning may work in practice, but is not guaranteed to always converge to the correct model.

2.3.1.1. Alternative Model-learning Approaches

It is important to note some general limitations of IfO using model-learning. Most methods that use this approach operate under an assumption that the state transitions shown in the demonstrations can be executed by the agent using only a single action [Torabi et al., 2019]. Moreover, learning a model can also be a challenge in the case of multi-modal behavior i.e. when there exist multiple actions that lead to the same state transition. Another significant issue is the typical requirement of environment interaction for model learning, limiting the applicability for some physical systems. To overcome these limitations, some IfO methods learn alternative dynamics models.

In one such method, 'Imitating Latent Policies from Observation' [Edwards et al., 2018], a forward dynamics model is learnt in a latent action (z) space. Using the demonstrative state trajectories, an initial latent policy $\pi_z(s_t)$ and a forward model $f(s_{t+1}|s_t, z)$ is estimated. This does not require environment interactions and can be thought of as estimating what needs to be done (latent actions z) based on just observing the demonstrator. Subsequently, using interactions with the environment, the latent actions are mapped to actual actions and thus a real policy π is learnt. The method requires a lower number of environment interactions as compared to IDM learning methods such as BCO while still performing well for OpenAI Gym tasks such as 'Acrobot' and 'CoinRun' [Edwards et al., 2018]. However, the method is only applicable to environments with deterministic state transitions and discrete action spaces since the number of possible actions need to be known beforehand to build the latent space.

Another method in this domain is proposed by Pathak et al. [2018] titled: 'Zero-Shot Visual Imitation'. It involves learning a model that predicts a *sequence of actions* to be executed from the current state that lead to the goal state. This model, known as a 'Goal Conditioned Skill Policy' (GSP), is implemented using a recurrent neural network and is learnt by interacting with the environment. The method also attempts to solve the multi-modal behavior problem i.e. when there exist multiple actions that lead to the same state transition. First, a forward dynamics model ($f(s_t, a_t) = s_{t+1}$) is estimated using the samples from environment interactions. Then, when training the GSP, a 'forward consistency loss' is used which only penalizes actions a_t if they result in a different next state s_{t+1} than the one predicted by the forward dynamics model f. In this way, multiple actions that lead to the same goal state are not penalized and can be learnt by the GSP. The method outperforms 'Vision-based self-supervised learning' [Nair et al., 2017] in rope manipulation and has also been used to successfully navigate a robot through an office environment [Pathak et al., 2018].

2.3.2. Inverse Reinforcement Learning (IRL)-based methods

Another popular IfO approach is to leverage Inverse Reinforcement Learning (IRL). This implies learning a reward function using only the demonstrated state/observation sequences and then learning the policy using RL. In this section we elaborate on such methods while making a distinction between methods aimed at developing suitable reward functions i.e. *reward-shaping method*s (section 2.3.2.1), and *adversarial methods* which involve learning a discriminator that provides rewards (section 2.3.2.2).

2.3.2.1. Reward-shaping based methods

These IfO methods utilize reward-shaping in the IRL step (shaping involves choosing/designing a viable reward function that guides the agent during RL).

Given demonstrations in the form of state trajectories, a natural approach is to reward the agent if it executes actions that result in these state trajectories. Such an approach is used by Kimura et al. [2018] who propose learning a sequence model that uses the current state to predict the next state ($f(s_t) = s_{t+1}$) based on the behavior observed in the demonstrations. The reward function is based on the Euclidean distance between the actual next state reached and the next state predicted by the sequence model. Imitation policies learnt using this reward function outperform Behavioral Cloning (BC) as well as other sparse reward functions for simulated tasks such as 'Reacher' and video games such as Super Mario Bros. [Kimura et al., 2018].

Another method, proposed by Brown et al. [2019], can not just imitate but also potentially improve upon the demonstrator's performance. In this method, using a set of *ranked* demonstra-

tion trajectories, a reward function is estimated that best explains the rankings. With sub-optimal demonstrations, the method has been used to learn policies that surpass the performance of the demonstrations for tasks such as 'HalfCheetah' and 'Ant' in OpenAI Gym as well as several Atari games [Brown et al., 2019].

Reward shaping has also been used by many methods in the domain of video based demonstrations [Liu et al., 2018; Sermanet et al., 2018; Dwibedi et al., 2018; Aytar et al., 2018; Goo and Niekum, 2019]. To overcome problems in this domain such as embodiment mismatch, a core idea behind many of these approaches is to devise a **reward function that is invariant to changes in embodiment or viewpoint**. Liu et al. [2018] propose such a method which is able to imitate behavior even when there exist context differences between the demonstrator and imitator. This is done by learning a context translation model through which observations of a task can be converted from one context to the other. The internal feature space of the learnt model captures key information about a task. Thus the Euclidean distance between the demonstrator and imitators observations in this feature space is used to specify the reward. The method has been validated by learning 'Reacher' and 'Pusher' tasks with both simulated and real robots [Liu et al., 2018]. However, a limitation of the technique is that to learn the translation model many demonstrations are required in *each* of the contexts.

2.3.2.2. Adversarial methods

While the reward-shaping based IfO approach has let to some success, developing suitable reward functions requires sufficient domain knowledge which can restrict applicability. An alternative is to learn both the reward function and the imitation policy in an adversarial manner (inspired by the Generative Adversarial Network (GAN) learning approach [Goodfellow et al., 2014]). These methods have also shown state-of-the-art performance for current IfO techniques [Torabi et al., 2019] and we will detail some them in this subsection.

Adversarial IfO methods typically extend Generative Adversarial Imitation Learning (GAIL) [Ho and Ermon, 2016] to enable learning using only states/observations. In one such method proposed by Merel et al. [2017], a discriminator network is trained to output differences between the demonstrator and imitator based on only the state distributions observed during their execution of the task. The discriminator's output is effectively used as reward to train the generator network i.e. the imitation policy in an RL fashion. This work has been extended by Torabi et al. [2019] in their method titled GAILfO (GAIL from Observations). In this method, the discriminator compares distributions of *state transitions* (as opposed to only states) which is more suitable for capturing the agent's behavior [Torabi et al., 2019]. The method has also been tested with visual demonstrations and has been shown to learn good policies for simulated tasks such as 'Hopper' in Open AI Gym as well as a physical 'Reacher' task using a robotic arm.

Another adversarial approach using visual observations has been proposed by Stadie et al. [2017] who also try to solve the viewpoint difference problem. In their method titled 'Third-person Imitation Learning', demonstrations are in the form of *third-person* observations rather than *first-person*. The discriminator network in this case is divided into the feature extraction layers (early layers of the network) and the classification layers (that do the discrimination between demonstrator and imitator). The network is then trained in such a way that the feature extraction layers are invariant to viewpoint differences. This approach has led to some success for tasks such as 'Reacher' in OpenAI Gym with observations from different camera angles.

The main drawback of the IRL-based IfO approach is that the RL formulation requires significant environment interactions during learning. This is especially true for Adversarial methods

which tend to have very high sample complexities [Torabi et al., 2019].

2.4. Conclusions

In this chapter, we outlined the basic framework of Imitation Learning (IL) and discussed important methods and algorithms in this domain. We also saw how human interaction has proved to be a powerful tool that can improve agent performance and increase the applicability of IL. Moreover, we elaborated on the Imitation from Observation (IfO) problem and briefly discussed some approaches to learning using state-space information.

The main conclusions we note from this chapter are the following:

- Among interaction-based learning approaches, the usage of corrective feedback is the most promising approach in scenarios with non-expert demonstrators. This is because providing corrective feedback in the form of adjustments to the current states/actions being executed by the agent is easier for non-experts (as opposed to providing exact state/action labels that should be executed). Moreover, evaluative feedback methods are sensitive to the demonstrator's scoring of good and bad behavior and evaluation is ambiguous when comparing multiple sub-optimal agent behaviors.
- Among corrective feedback learning techniques, a typical approach is to utilize corrections in the action-space [Celemin and Ruiz-del Solar, 2019; Pérez-Dattari et al., 2019] or to use predefined or learnt primitives [Argall et al., 2008, 2011] to guide agents. However, the action-space is often not conducive for providing feedback to the agent (Eg. Action-space as joint torques for a robotic arm). Moreover, defining primitives requires significant prior knowledge about the environment as well as the task to be performed, thus limiting the generalizability of such methods. An alternative approach could be to use corrective feedback in state-space to guide agents. This is however, still an open problem.
- To enable learning using state-space information, many Imitation from Observation (IfO) methods propose using an Inverse Dynamics Model (IDM) [Nair et al., 2017; Torabi et al., 2018]. Such an approach could also be applied for an Interactive Imitation Learning method in order to compute requisite actions when provided corrective feedback in state-space. An implication for such a method however, is the additional focus towards learning an accurate dynamics model. A sufficiently rich state-action transition dataset should be sampled which also implies a significant number of environment interactions.

With these insights in mind, we propose a new Interactive Learning method, details of which are provided in the next chapter.

3

Teaching Imitative Policies in State-space (TIPS)

As noted in the previous chapter, there are currently no Interactive Imitation Learning techniques that utilize human corrective feedback in state-space. Since providing such state feedback is often easier for a demonstrator (as opposed to feedback in the action-space), developing such a technique can reduce the required demonstrator expertise in Imitation Learning (IL). This chapter details our proposed method: Teaching Imitative Policies in State-space (TIPS). With this we aim to provide an intuitive mechanism for non-expert demonstrators to teach agents. Throughout the presentation of this method, we assume environments with continuous state-spaces ($s \in S$) and unknown dynamics. The method is however applicable to both continuous and discrete action spaces ($a \in A$).

Section 3.1 details the learning framework of the method. In TIPS, on-line binary feedback is obtained from the human demonstrator (implying an increase/decrease of particular states) and the agent policy is accordingly modified.

Section 3.2 elaborates on the indirect inverse dynamics mechanism used in TIPS. This is used to compute actions that realize the state changes requested by the demonstrator.

Section 3.3 delineates the algorithm in full while some comments and conclusions about TIPS are discussed in section 3.4.

3.1. Learning Framework

The general idea in TIPS is to let the agent execute its policy while a human demonstrator observes and suggests modifications to the state visited by the agent at any given time. This feedback is then used to **update the agent's policy on-line** i.e. during the execution itself. The initial policy of the agent could be random or could be pre-trained using prior knowledge/demonstration (if available).

The learning framework of TIPS is similar to another corrective feedback method: D-COACH [Pérez-Dattari et al., 2019; Celemin and Ruiz-del Solar, 2019]. However, while D-COACH uses feedback in the action-space, TIPS uses feedback in state-space. In this framework, human feedback (h_t) is in the form of binary signals implying an increase/decrease in the value of a state (i.e. $h_t \in \{-1, 0, +1\}$ where 0 implies no feedback). Each dimension of the state has a corresponding feedback signal. To convert this binary signal into a modification value, an error constant hyper-

parameter (e) is chosen. Thus the human desired state (s_{t+1}^{des}) is computed as:

$$s_{t+1}^{des} = s_t + h_t.e. \tag{3.1}$$

It is important to note that the feedback (h_t) and the desired modification can be both in the full state or partial state. Thus, the demonstrator is allowed to just suggest modifications in the partial state dimensions that are well understood or easy to observe for the demonstrator. Moreover, the binary feedback mechanism is simpler for the demonstrator as opposed to providing an exact value of the desired state. Even though the state computed using binary feedback (s_{t+1}^{des}) may be larger than what the demonstrator is suggesting, Celemin and Ruiz-del Solar [2019] and Pérez-Dattari et al. [2019] have shown that it is **sufficient to capture the trend of modification**. This is because, when updating the policy, information from past feedback is also used via a replay memory mechanism. The idea is that if a sequence of feedback provided in a state is in the same direction (increase/decrease), the demonstrator is suggesting a large magnitude change. Conversely, if the feedback alternates between increase/decrease, a smaller change around a set-point is suggested [Celemin and Ruiz-del Solar, 2019].

With the desired state known, we compute the action (a_t^{des}) required to realize the transition from the current state to the desired state i.e $s_t \rightarrow s_{t+1}^{des}$. To do this we use an Indirect Inverse Dynamics computation. This is because, since the desired state transition could be in the partial state dimension or could be infeasible, directly using an Inverse Dynamics Model (IDM) is ill-suited. Details about the action computation are provided in section 3.2.

After computing the required action a_t^{des}, the policy ($\pi(s)$) can be trained in a supervised-learning fashion using the state-action pair (s_t, a_t^{des}). Training can differ based on the type of representation function used for the policy. We represent policies using feed-forward artificial neural networks and hence use a similar training mechanism as D-COACH [Pérez-Dattari et al., 2019]. This

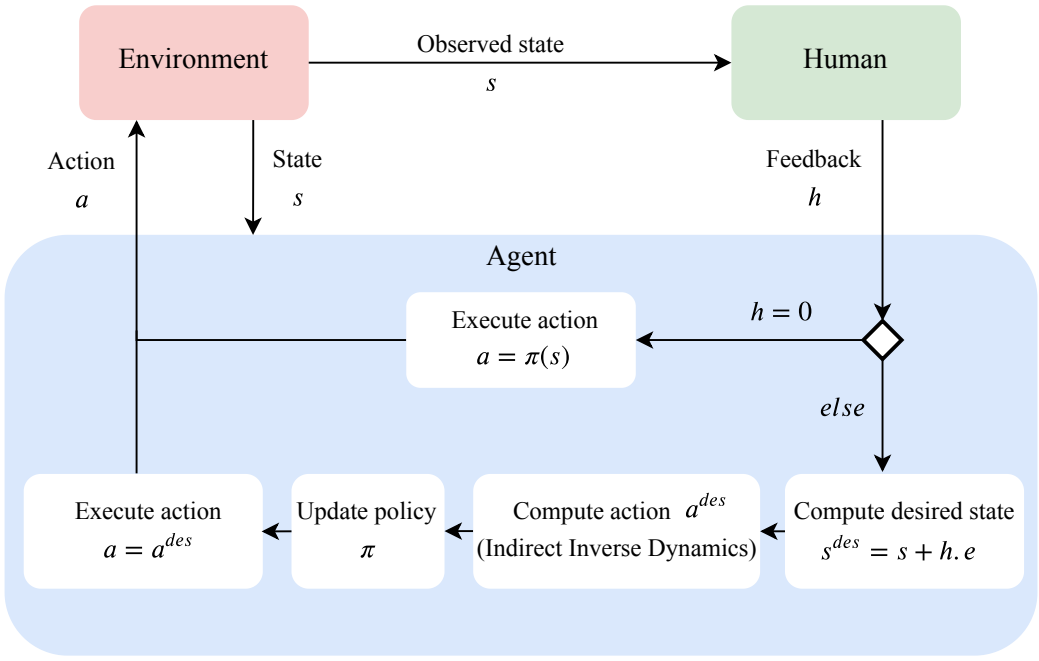


Figure 3.1: High-level representation of the learning framework of TIPS.

involves an immediate training step using the current state-action sample as well as a training step using a batch sampled from a 'demonstration replay memory' (similar to experience replay [Lin, 1992]). The demonstration replay memory stores previously computed state-action pairs based on previous feedback received. As noted before, this replay mechanism ensures that the learning is as per the sequence of changes suggested by the demonstrator. Moreover, the combination of the two training steps ensures that the policy is updated based on recent feedback while also avoiding the catastrophic forgetting problem in neural networks [French, 1999]. Lastly, to ensure sufficient learning iterations to train the neural network, a batch replay training step is also carried out periodically every T_{update} time-steps. The full algorithm is detailed in section 3.3.

Crucially, the computed action a_t^{des} is also executed immediately by the agent. This helps in visiting the subsequent states that the demonstrator wants to visit. It also speeds up the learning process since further feedback can be received in subsequent states to learn the next actions to be taken. Moreover, executing the computed action a_t^{des} also makes it easy for the human demonstrator to observe the effect of the feedback that they have just provided. An implication of this however is that the action needs to be computed in real-time. A high-level view of the learning framework can be seen in Figure 3.1.

3.2. Computing Actions via Indirect Inverse Dynamics

For the computation of actions (a_t) that lead to the desired state transitions ($s_t \rightarrow s_{t+1}^{des}$), many methods have proposed using an Inverse Dynamics Model (IDM) [Nair et al., 2017; Torabi et al., 2018]. However, IDMs are ill-suited in our case for two main reasons. Firstly, the feedback provided by the demonstrator can be in the partial state-dimension, leading to ambiguity regarding the desired state transition in the remaining dimensions. For instance, the state-space of the environment could include both position and velocity while the demonstrator only provides feedback on the position. In this case the necessary desired velocity transition is unknown. Secondly, the desired state transition ($s_t \rightarrow s_{t+1}^{des}$) may be infeasible. There may not exist an action that leads to the human suggested state transition in a single time-step. In such cases the IDM provides no solution and the method fails.

To avoid these pitfalls, we propose an indirect inverse dynamics mechanism. This involves sampling possible actions ($a \in A$) and using a **learnt forward dynamics model** (f) to predict the next states ($\hat{s}_{t+1} = f(s_t, a)$) for these actions. The action that results in a subsequent state that is closest to the desired state *in the partial state dimensions* is chosen. Mathematically, we can write this as:

$$a_t^{des} = \arg\min_a \left|\left| f(s_t, a) - s_{t+1}^{des} \right|\right|, \quad \text{where } a \in A \text{ (N_a uniform samples).} \tag{3.2}$$

The actions are uniformly sampled from the action-space A which can be discrete or continuous and the number of samples N_a is a hyper-parameter. This indirect inverse dynamics formulation allows us to compare only the state dimensions of interest and compute actions accordingly. Moreover, it avoids the infeasibility problem altogether since the objective is to choose the action that brings us *closest* to the desired next state, regardless of whether the desired state transition is feasible.

The success of this mechanism depends on a few key factors. Firstly, an accurate Forward Dynamics Model (FDM) needs to be learnt. For this, our proposed model-learning technique is

explained in the next sub-section 3.2.1. Secondly, the chosen number of samples N_a should be a sufficiently large to ensure that an appropriate action is sampled that brings us close enough to the desired state. Lastly, there can be scenarios in which there exist multiple actions that lead to the same partial state transition, leading to ambiguity around action selection. For example, consider a task performed by a robotic manipulator. There can be multiple viable actions that result in the desired state transition in the position of the end-effector. In such cases, to break ties among such actions, we propose selecting the action that results in the *least predicted change in the full state* i.e. minimizes $||s_t - \hat{s}_{t+1}||$.

3.2.1. Model Learning

We propose a sampling-based approach to learn the Forward Dynamics Model (FDM) in a supervised-learning fashion. An implication of this is that a sufficiently rich state-action transition dataset needs to be sampled, requiring a significant number of environment interactions. We propose a two-stage learning procedure similar to the one used in Behavioral Cloning from Observation (BCO) [Torabi et al., 2018].

In an initial stage (before any teaching by the demonstrator), the agent is allowed to gain experience of the dynamics by letting it execute an exploration policy π_e. This could be a random exploration policy or could be pre-defined based on prior information about the environment. The number of experience samples to be generated is a hyper-parameter (N_e) chosen specific to the environment. The generated samples, $\{(s_i, a_i, s_{i+1})_{i=1}^{i=N_e}\}$, are stored in an experience buffer E and used to train a FDM f_θ. This involves learning the model parameters θ such that:

$$\theta^* = \arg\min_{\theta} \sum_{i=1}^{i=N_e} \left|\left| s_{i+1} - f_\theta(s_i, a_i) \right|\right|. \tag{3.3}$$

Solving this equation for parameters θ can be done using any supervised-learning procedure based on the representation function used for the FDM (f_θ).

This initial FDM is then further improved using the experience gained in the demonstrator teaching phase. The state-transitions and actions taken by the agent are continuously added to the experience buffer E and the FDM is updated at the end of every episode. The update is done using all prior experience i.e. batches sampled from the experience buffer E. This second learning stage not only fine-tunes the model but also helps in learning the dynamics in state-action spaces that are newly visited by the agent as it learns its policy.

3.3. Algorithm in full

The full procedure of TIPS can be seen in Algorithm 3.

As mentioned previously, in an initial Model-learning phase, samples are generated by executing an exploration policy π_e (random policy in our implementation) and used to learn an initial Forward Dynamics Model (FDM) f_θ (solving equation 3.3). Moreover, the samples are added to an experience buffer E that is used later to update the model. In our implementation we represent the FDM using a feed-forward neural network.

In a second Teaching phase, the human demonstrator suggests changes to the state visited by the agent i.e. $s_t \rightarrow s_{t+1}^{des}$. To compute the action required for this state transition we use the indirect inverse dynamics as detailed in section 3.2. The computed action a_t^{des} is also executed

Algorithm 3: Teaching Imitative Policies in State-space (TIPS)

Initial Model-Learning Phase:

Require: Number of exploration samples N_e
Initialize:
Forward-dynamics model f_θ,
Exploration policy π_e (random/prior knowledge),
Experience buffer $E = []$
for $i = 1, 2...N_e$ **do**
Generate sample (s_i, a_i, s_{i+1}) by executing policy π_e
Append sample to experience buffer E
end
Learn model f_θ using inputs $\{s_i, a_i\}_1^{N_e}$ and targets $\{s_{i+1}\}_1^{N_e}$

Teaching Phase:

Require: Forward-dynamics model f_θ, Experience buffer E, Number of action samples N_a, Error constant e, Periodic policy update interval T_{update}
Initialize:
Agent policy π_ϕ (random/prior knowledge),
Demonstration buffer $D = []$
for *episodes* **do**
for $t = 0, 1, 2.....T$ **do**
Visit state s_t
Get human corrective feedback h_t
if h_t is not 0 **then**
$error_t = h_t.e$
Compute desired state $s_{t+1}^{des} = s_t + error_t$
Compute action $a_t^{des} = \arg\min_a \left|\left| f_\theta(s_t, a) - s_{t+1}^{des} \right|\right|$, using N_a sampled actions
Append (s_t, a_t^{des}) to demonstration buffer D
Update policy π_ϕ using pair (s_t, a_t^{des}) and using batch sampled from D
Execute action $a_t = a_t^{des}$, reach state s_{t+1}
Append (s_t, a_t, s_{t+1}) to experience buffer E
else
No feedback
Execute action $a_t = \pi_\phi(s_t)$, reach state s_{t+1}
Append (s_t, a_t, s_{t+1}) to experience buffer E
end
if $\mathrm{mod}(t, T_{update})$ **then**
Update policy π_ϕ using batch sampled from demonstration buffer D
end
end
Update learnt FDM f_θ using samples from experience buffer E
end

immediately. The policy π_ϕ is updated using the pair (s_t, a_t^{des}) as well as a batch sampled from the demonstration buffer D that stores previous corrections. We represent the policy using a feed-forward neural network which is randomly initialized with small weights.

Finally, to ensure enough training iterations, the policy π_ϕ is trained periodically using samples from the demonstration buffer D. Moreover, to improve the FDM, it is trained every episode using the experience gathered in E.

3.4. Discussion

In this chapter we delineated our proposed method TIPS that uses human corrective feedback in state-space to teach agent behavior. We explained our binary feedback mechanism (inspired by COACH [Celemin and Ruiz-del Solar, 2019]) and how it makes providing demonstration simple for the demonstrator. Unlike COACH however, our method allows the demonstrator to provide relative corrections in both discrete and continuous action spaces. We also noted how computing the actions required to perform the suggested state transitions is a challenge when the feedback received is in partial state dimensions. To tackle this, we elaborated on our indirect inverse dynamics computation.

There are however, some limitations of our approach. Firstly, model-learning through random sampling can be difficult in many environments and typically requires a significant number of environment interactions. This can be a limiting factor in scenarios where environment interaction is expensive. Another factor for consideration is that our action computation technique requires uniformly sampling N_a possible actions, followed by real-time execution of the chosen action. In high-dimensional continuous action spaces, this requires significant computational power, affecting the scalability of the method to such spaces. Another problem that is unaddressed by TIPS is credit assignment. The feedback provided by the demonstrator is assumed to be applicable to the current state s_t. However, in many relatively fast environments, the feedback may have been intended for a past state. Nevertheless, as we will show in the next chapters, these limitations were not significant in practice when learning several control tasks. Overall, we were able to learn a sufficiently accurate forward dynamics model, compute correct actions quickly and avoid credit assignment problems even with non-expert demonstrators.

In the next chapters, we detail our experiments and provide results obtained from the evaluation of TIPS.

4

Experimental Setting

In this chapter, we provide information about the experimental setting for the evaluation of our method: Teaching Imitative Policies in State-space (TIPS). Experiments are setup with human teachers who are non-experts and our main criteria for evaluation are (i) task performance of the trained agent over time and (ii) the ease of demonstration for the human teacher.

Firstly, some implementation details of the method are provided, followed by information about the different domains and tasks used for evaluation. The techniques/methods that are used for comparison are then mentioned. Finally, we elaborate on the setup for conducting trials with human teachers.

4.1. Implementation of Method

Our implementation of TIPS (in python) is available at: github.com/sjauhri/Interactive-Learning-in-State-space. As mentioned previously, we represent the forward dynamics model (f) and policy (π) as feed-forward artificial neural networks. We use the TensorFlow python library to implement and train the neural networks. Training of the neural networks is done in a standard supervised learning fashion and for optimization we use the Adam variant of stochastic gradient descent [Kingma and Ba, 2014]. The number of layers and sizes of the neural networks vary as per the task being learnt and these parameter settings can be seen in Table 4.1. In discrete action-spaces, the neural networks consist of an additional soft-max layer at the output. Both the model and the policy networks are initialized with random small weights.

For model-learning, we use a random exploration policy (π_e) and the number of initial exploration samples (N_e) vary with the size of state-action space and complexity of the task to be learnt. The states for which feedback is provided are also chosen specific to the task. The number of actions to be sampled (N_a) for computing the inverse dynamics is based on the size of the action space. The parameter settings for each of the domains/tasks we use for evaluation can be seen in Table 4.1.

Table 4.1: Parameter settings in the implementation of TIPS for different evaluation tasks.

Parameter	**Value**			
	CartPole	Reacher	LunarLander	Robot-Fishing
No. of exploration samples (N_e)	500	10000	20000	5000
States for feedback	Pole tip position	x-y position of end effector	Vertical, angular position	x-z position of end effector
Error constant (e)	0.1	0.008	0.15	0.05
No. of action samples (N_a)	10	500	500	1000
Periodic policy update interval (T_{update})	10	10	10	10
FDM Network (f_θ)				
Network layer sizes	16, 16	64, 64	64, 64	32, 32
Learning rate	0.005	0.005	0.005	0.005
Batch size	16	32	32	32
Policy Network (π_ϕ)				
Network layer sizes	16, 16	32, 32	32, 32	32, 32
Learning rate	0.005	0.005	0.005	0.005
Batch size	16	32	32	32

4.2. Evaluation Domains

4.2.1. OpenAI Gym

The OpenAI gym toolkit [Brockman et al., 2016] provides standardized simulated environments and tasks for the evaluation of learning agents. We use three such environments for the evaluation of agents trained via TIPS, namely: CartPole, Reacher and LunarLanderContinuous (depicted in Figure 4.1). Each OpenAI gym environment has a pre-defined reward function and thus the net reward obtained by the agent during execution i.e. return can be used as a performance metric.

4.2.1.1. CartPole

This is a classic control problem which involves balancing a pole attached to a cart which moves along a friction-less track. The cart is controlled by applying a force to it. Thus the action-space is discrete with the actions being a force applied to the cart towards the left or right. A reward of +1 is provided for every time-step that the pole remains upright. The maximum length of an episode is 200 time-steps but can also end if the pole is more than 40 degrees from the vertical or if the cart moves more than 2.4 units from the center. To make the task viable for demonstration, the duration of each execution time-step is set to 75ms.

In our TIPS implementation for this task, we chose the position of the tip of the pole as the state based on which feedback is provided. Thus, the human demonstrator needs to observe the tip of the pole and suggest which direction the tip should move (left/right) in order to balance the pole.

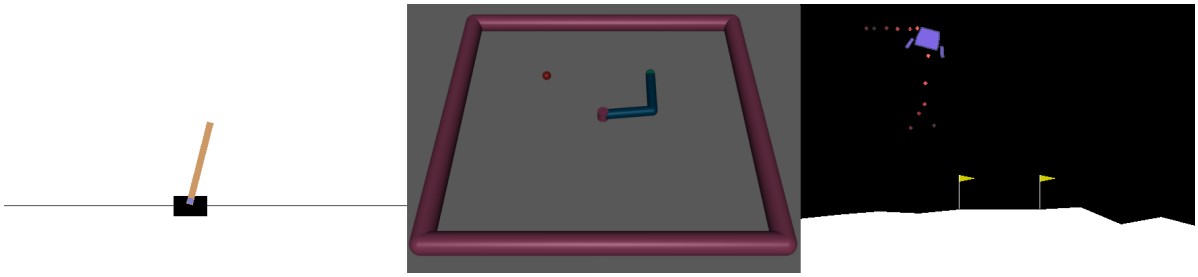


Figure 4.1: Representative screenshots of (from left to right) the CartPole, Reacher and LunarLanderContinuous tasks in OpenAI Gym [Brockman et al., 2016].

4.2.1.2. Reacher

The Reacher task involves using a 2 DOF robotic arm to reach a particular target position in a flat plane. This task uses the Mujoco physics engine [Todorov et al., 2012] for simulation. The actions (continuous) are in the form of forces at the two joints which move the two arms clockwise or anticlockwise. A negative reward is provided at every time-step based on the the force applied at the joints as well as the distance between the end effector and the target position. The length of an episode is set to 500 time-steps. The duration of each execution time-step is set to 50ms.

In our TIPS implementation for the Reacher task, the demonstrator provides feedback based on the x-y Cartesian position of the end effector. Thus instead of providing actions in the form of forces, feedback is provided in the form of up/down/left/right signals which accordingly move the end-effector in the x-y plane. Since, the movement of the arm is quite sensitive to actions/forces applied at the joints, we restrict the force to a maximum of 50% of the total capacity. Importantly, the Reacher problem is simplified in our experiments by keeping the target position fixed over all episodes during training and evaluation. This makes it easier to learn a policy via demonstration since the learnt behavior does not need to generalize to different target positions. This simplification is done mainly to make it feasible for the human demonstrators to teach a good policy in a relatively short amount of time. The simplification also makes it easier for the human demonstrator to compare state and action-space feedback since the target position is the same in both types of experiments.

4.2.1.3. LunarLanderContinuous

This problem involves flying a Lunar-Lander craft to a landing pad between the two flags (Figure 4.1). The actions are two continuous values that control the firing of the landers' main engine and steering/side engines. The first action powers the main (bottom) engine while the second action powers the side engines (left or right but never both). The engines do not fire with less than 50% power. The lander starts from the top of the screen and needs to navigate to the landing pad at the bottom. This entails a reward of 140, though a -20 reward is received for each leg of the lander that is damaged in the landing process. Moreover, the firing of engines is punished with a small negative reward every time-step. The episode ends if the lander crashes or comes to rest, receiving an additional -100 or +100 respectively. The maximum episode length is set to 1000 time-steps. The duration of each execution time-step is set to 80ms.

The states chosen for providing feedback in the implementation of TIPS for this task include the vertical and angular positions of the lander (in the world frame). The human teacher provides feedback in the form of up/down signals to modify the vertical position and left/right signals to suggest a anti-clockwise/clockwise rotation of the lander. Such feedback has a stabilizing effect when flying the lander since the vertical position is maintained even when modifying the angular

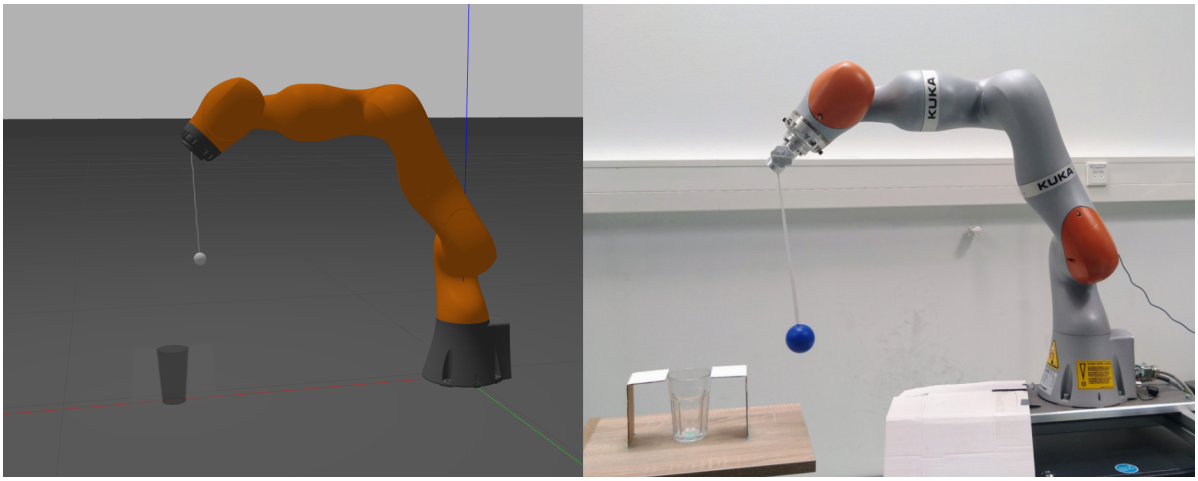


Figure 4.2: The robotic 'fishing' task using a KUKA lbr iiwa 7 robot, visualized in Gazebo (left) and with the real robot (right). The task is to place the swinging ball tied to the end effector of the robot into the cup.

position and vice versa. The teacher is also allowed to provide feedback in both state dimensions simultaneously (i.e. up+left, down+right etc.).

4.2.2. Robotic Fishing Task

To validate the application of TIPS on a real robot, we design a simple experiment for a robotic 'fishing' task. For this task we use a KUKA LBR iiwa7 robotic manipulator with a ball attached to its end-effector by a thread. The task is to swing the ball into a nearby cup/slot (similar to handling a fishing rod with a bait attached). To reduce the complexity of the task, the movement of the robot end-effector (and ball) is restricted to a 2-D x-z Cartesian plane i.e. height and forward/backward movement. This is done by restricting the usage of the robot's joints (only the 2nd, 4th and 6th joints are used). The task is visualized in Figure 4.2.

The experiment is first validated using simulations of the task in the Gazebo simulator followed by execution using the real robot. For simulations, as well as for interfacing with the robot, we use a ROS interface and the iiwa stack [Hennersperger et al., 2017]. The robot is controlled using a iiwa stack position controller that controls the joint angles. The actions taken by the robot are in the form of relative changes to the joint angles. An external camera is used to track the position of the ball using OpenCV color segmentation and blob detection. The overall state-space of the environment consists of the joint angles, joint angular velocities and the position and velocity of the ball in the x and z axes.

To teach the task using TIPS, feedback is provided in the x-z Cartesian position of the end-effector. The demonstrator provides up/down/left/right signals that should accordingly move the end-effector in the x-z plane. To enable this, a forward dynamics model in the joint-angle space is learnt. Thus the model is used to predict the position of the end-effector based on the joint angle commands (actions) sent to the robot. An appropriate action is chosen from a sample size of 1000 actions (Parameter settings in Table 4.1).

To evaluate the task, we define a reward function to be used as a performance metric. The function is inspired from the reacher task and consists of a negative reward at every time-step based on the magnitude of the action command sent to the robot as well as the distance between the ball and the center of the cup/slot ($r_t = -\|a_t\| - \|dist_t\|$). Note that this experiment is run only to validate

the application of TIPS and comparisons are not made with other learning methods.

4.3. Methods for Comparison

We evaluate and compare TIPS to other techniques based on two main criteria: (i) the task performance of the trained agent over time and (ii) the total demonstration effort required by the human teacher. Firstly, we compare the performance of the TIPS agent with the demonstrator's own performance when executing the task via tele-operation. We further compare with other agents trained using standard Imitation Learning (IL) techniques. It is also of interest to highlight the differences between demonstration in state-space versus action-space. For this, we compare both tele-operation and corrective feedback learning techniques in state and action spaces.

Following is the list of techniques used for the comparison:

- **Tele-operation in Action-space:** The demonstrator is asked to execute the task by providing action values. Actions are in the form of forces to the cart/joints for the CartPole and Reacher tasks and firing the main/side engines for the LunarLander task.
- **Tele-operation in State-space:** The demonstrator is asked to execute the task by providing state-space information and computing actions via the indirect inverse dynamics mechanism of TIPS. The states that the demonstrator provides feedback on are the same as the ones used to teach in TIPS (Table 4.1).
- **Behavioral Cloning (BC):** Supervised learning to imitate the demonstrator using state-action demonstration data recorded during tele-operation. Notably, we only use demonstrations from tele-operation episodes where the demonstrator was successful in performing the task. Success is defined as a return of atleast 40% in the range of minimum to maximum return for the three tasks. Agents are trained using both action-space and state-space tele-operation data and the better performing agent is used for performance comparison.
- **Generative Adversarial Imitation Learning (GAIL) [Ho and Ermon, 2016]:** Inverse Reinforcement Learning (IRL) method that uses adversarial learning to learn a reward function and policy. Similar to BC above, the successful state-action demonstration data recorded during tele-operation is used for imitation. Agents are trained using both action-space and state-space tele-operation data and the better performing agent is used for performance comparison. The implementation of GAIL by Hill et al. [2018] is used for the tests.
- **D-COACH [Pérez-Dattari et al., 2019]:** Interactive Imitation Learning method that uses binary corrective feedback in the action space. The demonstrator suggests modifications to the current actions being executed to train the agent as it executes the task.

4.4. Experiment Setup with Human Teachers

We run experiments with non-expert human participants who have no prior knowledge of the tasks. A total of 22 sets of experiments are performed, with 8, 8 and 6 participants for the CartPole, Reacher and LunarLander tasks respectively. All the participants belong to an age group of 25 to 30 years.

Participants are first allowed to tele-operate and get acquainted with/understand the goals of the task. During execution, the performance metric i.e. the return after every episode is provided to

the participants. Subsequently, the participants perform four different experiments, namely: Tele-operation in action-space, Tele-operation in state-space, training an agent using D-COACH and training an agent using TIPS. To compensate for learning effect (i.e. participants learning and improving over time), the order of the experiments is changed for every participant. For consistency between the four experiments with respect to the environment stochasticity, the same random seed is used for all four experiments. The seed is changed for every new participant.

When performing tele-operation, the demonstrator provides actions and the corresponding states are recorded. Tele-operation is deemed to be complete once no new demonstrative information can be provided. When training interactively using D-COACH and TIPS, the demonstrator provides feedback until no more agent performance improvement is observed. The maximum number of episodes for all the experiments is set to 50 episodes.

It is also of interest to capture and compare the demonstration effort required for performing and teaching the tasks. For this, the human teachers are asked to fill out the NASA **Task Load Index Questionnaire** [Hart and Staveland, 1988] after each experiment. In this questionnaire, ratings related to the mental demand, physical demand, time pressure, performance, effort and frustration during the different experiments are obtained. The questionnaire is provided in Appendix B.

The results obtained from the experiments are provided in the next chapter.

5

Results

In this chapter, we provide the results from our experiments. We delineate the performance improvements provided by our method TIPS in trials where agents are trained by human participants. We compare the TIPS agent's performance to the demonstrator (tele-operation) as well as to agents trained via Behavioral Cloning (BC) and GAIL [Ho and Ermon, 2016] using the tele-operation data. Furthermore, a comparison of state-space and action-space demonstration is done by comparing tele-operation in both spaces as well as interactive learning in both spaces using TIPS and D-COACH [Pérez-Dattari et al., 2019].

To assess the demonstration effort required in state-space vs action-space teaching, we use the ratings filled out by the human teachers in the NASA Task Load Index Questionnaire [Hart and Staveland, 1988].

Finally we validate the application of TIPS to a real robot by displaying the performance obtained when learning the robotic fishing task.

5.1. Performance Improvements

Figure 5.1 shows the performance in terms of average return obtained for the tasks (averaged over all participants) using tele-operation, agents trained via IL techniques and agents trained using our method: TIPS.

We notice that tele-operation is challenging for the demonstrator, especially for time-critical tasks such as CartPole and LunarLander where the system is inherently unstable. The Reacher task is solved by the demonstrator albeit with a relatively large overall force applied and large position error between the end effector and target (as compared to TIPS). The LunarLander task is the most challenging for demonstrators with no participant successful in reliably performing the landing.

Agents trained using IL techniques (Behavioral Cloning (BC) and GAIL), with the successful tele-operation data (*40% return in the normalized range), perform better than the demonstrator for the CartPole task. For this task, the state-action space is the smallest and the consolidated successful demonstration data over multiple episodes of tele-operation is sufficient to learn a good policy and outperform the demonstrator. However, this is not true for the Reacher and LunarLander tasks. In the Reacher task, the IL agents tend to overshoot the desired target position and keep performing recovery actions, leading to further negative rewards and thus a low return. For the LunarLander

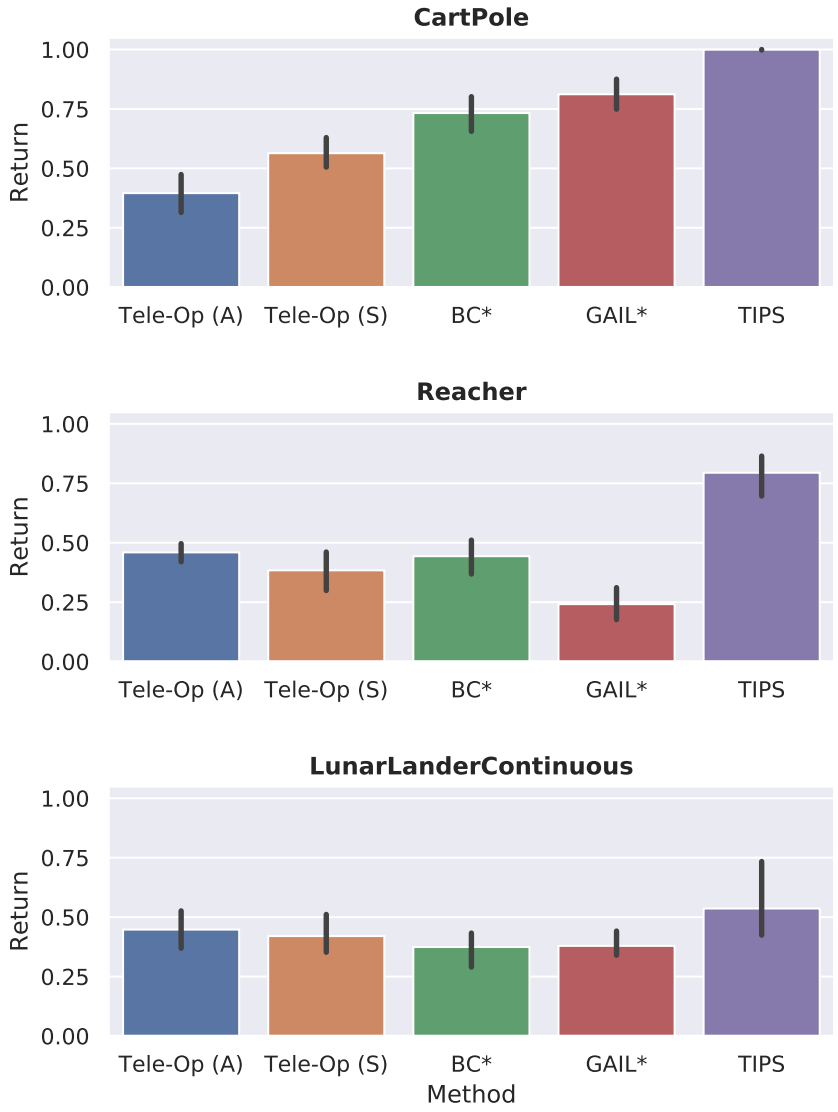


Figure 5.1: Performance achieved using the different methods in the experiments: Tele-Operation (Action space), Tele-Operation (State space), Behavioral Cloning (BC), GAIL and our method TIPS. The return is normalized for each environment and averaged over multiple episodes and over all participants (with the variance as indicated). Behavioral Cloning* and GAIL* use only successful tele-operation data (return of atleast 40% in the normalized range.)

task, the provided demonstrations do not generalize well to the situations encountered by the agent. This can be attributed to the higher stochasticity of the task as well as inconsistent demonstrations provided by the teachers.

Interactively learning via TIPS leads to the highest agent performance. It is observed that the first few episodes of training are similar to tele-operation in that the demonstrator provides a large amount of feedback based on which actions are immediately executed. After this stage, the demonstrator feedback reduces significantly and only corrections and fine-tuning of the behavior is done. This second learning stage especially leads to the learning of well performing policies. All participants are successful in achieving the maximum return for the CartPole task. High performing policies are also learnt for the Reacher task. The performance improvement with TIPS for the LunarLander task however, is not significant. While the participants are successful in teaching the lander to not crash, they struggle to teach it to land and thus the agent ends up flying out of the frame which leads to a low return.

5.2. State-space vs Action-space

5.2.1. Performance

Figure 5.2 compares the performance of state-space interactive learning (TIPS) and action-space interactive learning (D-COACH) over training episodes. The performance obtained during tele-operation in state-space and action-space can be seen in Figure 5.1.

In tele-operation, state-space demonstration leads to better performance for the CartPole task. Participants find it easier to provide feedback in terms of the position of the pole since balancing the pole is the main goal of the task. For the Reacher task however, tele-operation in state-space does not perform better. This is mainly due to inconsistencies of actions resulting from the provided feedback. Since the forward dynamics model is learnt and not perfect, the computed actions are always slightly different, even in similar state-spaces. This can make performing the task difficult for the demonstrator. For the LunarLander task, the difference in tele-operation performance in state and action spaces was insignificant.

When comparing interactive learning techniques (TIPS and D-COACH), the advantage of state-space feedback is significant for the Reacher task. Higher agent performance is achieved by all of the participants when teaching using TIPS as compared to D-COACH. For the CartPole task, all participants are able to train agents that achieve the highest return using both TIPS and D-COACH though fewer training episodes are needed when using TIPS (Figure 5.2). For the LunarLander task there is no significant performance improvement observed when using state-space demonstration. The overall demonstration mechanism in state-space using angular position is not too different from demonstration in action-space. Though state-space feedback does provide a stabilizing effect when flying the lander and leads to fewer crashes, it does not translate to a higher return. This is mainly due to the lander flying out of the frame. Since crashing the lander entails a smaller penalty as compared to flying out of the frame, no significant performance improvement is observed when using state-space feedback.

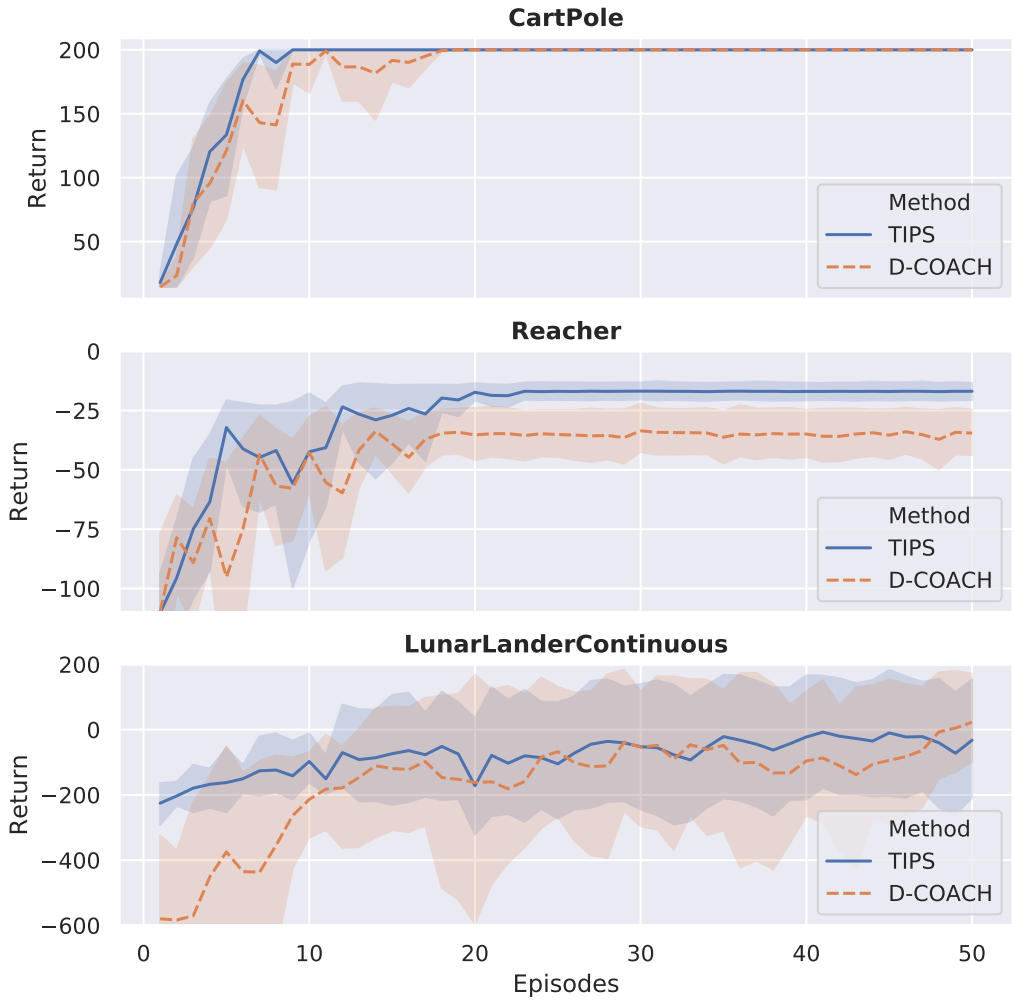


Figure 5.2: Average performance of TIPS and D-COACH agents over training episodes when trained by human participants (with variance between participants as indicated). TIPS uses state-space feedback while D-COACH uses action-space feedback.

Table 5.1: Average ratings provided by the participants in the NASA Task Load Index questionnaire [Hart and Staveland, 1988]. Values are normalized, with smaller magnitude implying lower mental demand etc. (S) and (A) are used to denote state-space and action-space techniques respectively.

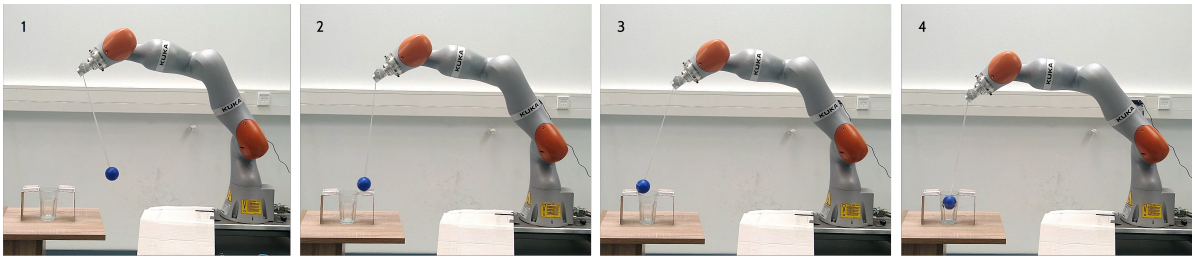
	Mental Demand	Physical Demand	Temporal Demand	1-Performance	Effort	Frustration
CartPole						
Tele-Operation (S)	0.54	0.53	0.56	0.19	0.56	0.37
Tele-Operation (A)	0.71	0.61	0.7	0.33	0.71	0.69
TIPS (S)	**0.29**	0.33	**0.33**	0.11	0.37	**0.19**
D-COACH (A)	**0.49**	0.37	**0.43**	0.14	0.44	**0.3**
Reacher						
Tele-Operation (S)	0.8	0.79	0.73	0.46	0.76	0.66
Tele-Operation (A)	0.66	0.56	0.63	0.19	0.69	0.47
TIPS (S)	**0.53**	0.64	0.61	0.17	0.63	**0.3**
D-COACH (A)	**0.63**	0.66	0.57	0.2	0.61	**0.41**
LunarLanderContinuous						
Tele-Operation (S)	0.8	0.8	0.7	0.37	0.77	0.73
Tele-Operation (A)	0.8	0.77	0.7	0.4	0.8	0.6
TIPS (S)	0.8	0.7	0.67	0.3	0.73	0.73
D-COACH (A)	0.8	0.77	0.67	0.27	0.73	0.6

5.2.2. Demonstrator Effort

To capture the the demonstrator's effort during teaching and tele-operation in state vs. action spaces, the NASA Task Load Index ratings provided by the participants are used (Table 5.1). The questionnaire is provided in Appendix B.

For the CartPole task, there is a significant reduction in demonstrator task load with state-space demonstration, both in the tele-operation case and for interactive teaching (TIPS, D-COACH). When teaching using TIPS, participants report lower ratings for all the parameters with the mental demand rating reduced by about 40% as compared to D-COACH. Moreover, temporal demand and participant frustration is reduced by about 25% and 35% respectively. These figures are highlighted in Table 5.1.

For the Reacher task, teaching using TIPS leads to a 25% lower participant frustration and a reduction in mental demand of about 16% as compared to D-COACH. For the other ratings, the improvements are not as significant as CartPole. One of the reasons for this could be the aforementioned inconsistencies of actions resulting from state-space feedback. Models learnt for the Reacher task are more complex and not as accurate as the ones in CartPole and thus the demonstrator needs to be attentive to these inconsistencies in computed actions for the Reacher task. For the LunarLander task, demonstration in state and action-spaces is equally challenging, backed up by little change in the ratings.



(a)

(b)

Figure 5.3: Results from the validation experiment on the KUKA robot. (a) Left to right, the fishing task performed by the robot after being taught by the demonstrator for 60 episodes using TIPS. (Episode length is 30 seconds) (b) Return obtained as per our defined reward function and demonstrator feedback rate over learning episodes. (Values are averaged over a rolling window of size 10)

5.3. Validation: Robotic fishing task

For validation on a real system, the robotic-fishing task (as described in subsection 4.2.2) is taught to an agent using TIPS. The experiment was first successfully carried out in simulation using the Gazebo simulator, followed by experiments on the real robot with a demonstrator in a laboratory setting. In this section we provide the final results from experiments on the real system (visualized in Figure 5.3). Since the objective is to validate the application of TIPS, comparisons with other learning techniques are not made.

To teach the task using TIPS, the teacher provides corrective feedback in terms of the desired end effector position. Based on some initial trials, a correction magnitude (i.e. error constant e) of 5 centimeters is found to work well for the task. The minimum height of the end effector is limited such that the ball does not hit the ground plane. For safety reasons, the joint angles are also limited to between zero and 90 degrees. To avoid the ball rolling off the sides of the cup and thus moving

out of the x-z plane, two supports are attached to the sides of the cup.

The learning is evaluated over a period of 100 episodes. Each episode is 30 seconds long, after which the robot is reset to its starting position. The initial position of the joints and the ball is randomized by making the robot perform a random motion at the start of every episode. The return, used as a performance metric, is calculated using a defined reward penalty at every time-step based on the magnitude of the executed action and the distance between the ball and the center of the cup.

The agent performance and demonstrator feedback rate over learning episodes can be seen in Figure 5.3b. The agent successfully learns to reliably perform the task (return of -15) after 60 episodes of training. After about 90 episodes, the agent performance is further improved in terms of speed at which the task is completed (return of -10). The feedback rate reduces over time as the agent performs better and only some fine-tuning of the behavior is needed after 60 episodes.

The learnt behavior depends on the strategy used by the demonstrator to perform the task. In our experiments the demonstrator's strategy is to move the end effector towards a position above the cup and choose the appropriate moment to bring the end effector down such that the ball falls into the cup. This behavior is sucessfully imitated by the agent (shown in Figure 5.3a). Notably, the robot has to be taught to stay still at the desired final state i.e. when the ball is in the cup. Thus, in such states, the demonstrator alternately provides feedback signals in opposite directions through which the agent eventually learns the average behavior at that state i.e. to stay still. A video of the learnt behavior is available at: https://youtu.be/aN1r8IytsXY.

5.4. Discussion

In this Chapter, we provided the results from our experiments for the evaluation of TIPS.

Through our comparison of TIPS with tele-operation and IL techniques (BC and GAIL), we have shown that agents trained using TIPS achieve significantly higher performance. This illustrates the viability of TIPS as an interactive learning technique suitable for scenarios where demonstrators are non-experts.

Through our comparison of state-space interactive learning (TIPS) with action-space interactive learning (D-COACH), we have shown that state-space feedback is advantageous. Both in terms of performance over time as well as demonstration effort, TIPS compares favorably to D-COACH for the CartPole and Reacher tasks. Thus, the merits of state-space interactive learning are clear. However, it is noted that these advantages are task specific. For the LunarLander task, state-space feedback helps avoid crashing the lander but poses a new problem of avoiding flying outside the limits which can still be difficult for demonstrators. It is also noted that actions computed based on feedback using TIPS can be irregular due to inaccuracies in model learning. Since handling such scenarios requires demonstrator effort, this can diminish the advantages provided by state-space feedback.

In the experiment carried out on the KUKA iiwa robot to learn the fishing task, we have shown that the agent is able to successfully learn a high performing policy in a reasonable amount of time through state-space feedback (end-effector position). This validates the application of TIPS to a real robotic task.

We conclude the discussion and this thesis work in the next chapter.

6

Conclusion

Imitation Learning (IL) techniques enable the programming of agent behavior through demonstration and eliminate the need for humans to manually engineer behavior. Moreover, Interactive Learning techniques have made it easier for demonstrators to train and improve agent performance by guiding the agent's actions as it performs the requisite task. However, providing guidance in terms of 'change of state' is often easier for human teachers as opposed to 'change of actions' being executed. In this thesis we proposed a novel Interactive Learning technique: 'Teaching Imitative Policies in State-space (TIPS)', that uses human corrective feedback in state-space to train agents.

The proposed method TIPS provides a simple binary corrective feedback mechanism (in the form of increase/decrease signals) in state-space through which demonstrators can train agents. Unlike similar techniques that use feedback in action space (COACH [Celemin and Ruiz-del Solar, 2019]), the method allows the demonstrator to provide relative corrections in both continuous and discrete action-space environments (since the actions are computed internally).

Through experiments with non-expert human demonstrators, it is observed that TIPS outperforms IL techniques such as BC and GAIL as well as Interactive Learning in action-space (D-COACH [Pérez-Dattari et al., 2019]). Moreover, the state-space feedback mechanism also leads to a significant reduction in demonstrator effort (captured using the NASA Task Load Index [Hart and Staveland, 1988]). With these results we have illustrated the viability of TIPS to non-expert demonstration scenarios and have also highlighted the merits of state-space Interactive Learning.

A caveat of the proposed method however is that, to compute actions, an accurate forward dynamics model needs to be learnt using random environment interaction samples. This can be challenging in many environments and can require a large number of environment interactions. Moreover, as observed in the experiments, inaccuracies in the learnt model can lead to irregular actions computed. Handling such scenarios requires demonstrator effort and thus diminishes the advantages provided by state-space feedback. Another consideration is that, for action selection, candidate actions are sampled from the entire action-space. This is not scalable to high-dimensional continuous action spaces, thus affecting the viability of the method to such spaces.

Despite these caveats, we have demonstrated the practical viability of TIPS on a real system by training an agent for a robotic fishing task. We are able to learn a sufficiently accurate forward dynamics model, compute correct actions and successfully teach the robotic task to an agent with good performance.

In conclusion, we have successfully demonstrated a method that makes corrective feedback learning more feasible in non-expert demonstration scenarios. Moreover, the advantages of Interactive Learning in state-space have been established.

Future recommendations

For the extension of this work, a main focus would be towards improving on the current action computation mechanism (i.e. indirect inverse dynamics). Following are some recommendations for this:

- Model learning: Improving the accuracy of the learnt model is of course crucial for computing more accurate actions. For this, one direction to explore could be to use smarter exploration strategies for acquiring experience samples. With a more complete experience dataset, the learnt model can be more representative of the ground truth. Another direction could be to use better knowledge representations for the model, such as Recurrent Neural Networks that can capture the history of states (although such representations can be especially data hungry).
- Action selection: In the proposed method, action selection is done by minimizing a simple cost function based on deviation of the predicted next state with the desired next state. However, in practice, the demonstrator may not actually require the desired state to be reached in a single time-step or using a single action. Thus, a time horizon may instead be used with multiple actions selected over the horizon to reach the desired state. Uniformly sampling actions from the action-space may not suffice in this case and an optimization method could instead be used to compute the actions. With these changes, the action computation mechanism starts looking more and more like a Model Predictive Control scheme and it would be very interesting to see if the scheme could be combined with our learning framework.

A

Paper

A draft version of the research paper presenting the method proposed in this thesis: Teaching Imitative Policies in State-space (TIPS), is provided below. The paper will be submitted to the Conference on Robot Learning (CoRL) 2020.

Interactive Imitation Learning in State-space

Snehal Jauhri
TU Delft
Delft, Netherlands
snehal.jauhri@gmail.com

Carlos Celemin
TU Delft
Delft, Netherlands
c.e.celeminpaez@tudelft.nl

Jens Kober
TU Delft
Delft, Netherlands
j.kober@tudelft.nl

Abstract: Imitation Learning techniques enable programming the behavior of agents through demonstration rather than manual engineering. However, they are limited by the quality of available demonstration data. Interactive Imitation Learning techniques, which involve providing feedback while the agent executes its task, can improve the efficacy of learning. In this work, we propose a novel Interactive Learning technique that uses human feedback in state-space to train and improve agent behavior (as opposed to alternative methods that use feedback in action-space). Our method titled Teaching Imitative Policies in State-space (TIPS) enables providing guidance to the agent in terms of 'changing its state' which is often more intuitive for a human demonstrator. The performance of TIPS is evaluated for various control tasks as part of the OpenAI Gym toolkit and for a manipulation task using a KUKA LBR iiwa robotic arm. Through continuous improvement via feedback, agents trained by non-expert demonstrators using TIPS outperform the demonstrator and conventional Imitation Learning agents.

Keywords: Imitation Learning, Interactive Imitation Learning, Learning from Demonstration

1 Introduction

We have been promised an autonomous future in which intelligent agents and robots can be tasked with driving, mowing our lawns and even performing surgery. To program agents that can perform such complex and diverse tasks requires developing robust control algorithms [1]. Typically, this involves engineers using their understanding of the task to mathematically devise algorithms to achieve the necessary performance. However, this requires considerable expertise and can still be limited by the number of situations considered by the engineer [2]. In this regard, techniques such as Reinforcement Learning and Imitation Learning, that allow behavior to be learnt from experience or from demonstration, provide an advantage.

Imitation Learning (IL) is a machine learning technique through which an agent learns to perform a task using example demonstrations of the task [3, 4]. It eliminates the need for humans to preprogram the required behavior for a task, instead utilizing the more intuitive mechanism of demonstrating it [5]. Advancements in Imitation Learning techniques have led to successes in learning tasks such as robot locomotion [6], helicopter flight [7] and learning to play games [8]. There have also been research efforts to make training via Imitation Learning easier for demonstrators. This is done by allowing them to interact by providing feedback to the agent as it performs the task [9, 10, 11]. Learning using such interactive methods has further increased the applicability of IL and often results in better agent performance [12].

One limitation of current Imitation and Interactive Imitation Learning techniques is that they typically require demonstration or feedback in the *action-space* of the agent. However, humans commonly learn behaviors by understanding the changes in *state* required for a task, not the precise actions to be taken [13]. Additionally, providing demonstration or feedback in the action-space can be difficult for demonstrators. E.g. consider a learning agent for a robotic arm manipulation task. Providing action-space information to the agent for the movement of individual joints requires considerable demonstrator expertise. It would be easier to instead provide state-space information such as the Cartesian position of the end effector.

4th Conference on Robot Learning (CoRL 2020), Cambridge MA, USA.

In this paper, a novel Interactive Learning method is proposed that utilizes feedback in state-space to learn behavior. The main focus of this work is to enable non-expert demonstrators to interactively train agents.

2 Related Work

In recent literature, several Interactive Imitation Learning methods have been proposed that enable demonstrators to guide agents by providing corrective action labels [12], corrective feedback [14, 11] or evaluative feedback [15, 16]. Celemin and Ruiz-del Solar [11] have argued that the usage of corrective feedback is the most promising approach in scenarios with non-expert demonstrators. This is because providing corrective feedback in the form of adjustments to the current states/actions being visited/executed by the agent is easier for non-experts (as opposed to providing exact state/action labels). Moreover, evaluative feedback methods require demonstrator's to score good and bad behavior which can be ambiguous when comparing multiple sub-optimal agent behaviors.

Among corrective feedback learning techniques, a typical approach is to utilize corrections in the action-space [11, 17] or to use pre-defined or learnt primitives [9, 14] to guide agents. However, the action-space is often not conducive for providing feedback to the agent (e.g. Action-space as joint torques of a robotic arm). Further, defining primitives requires significant prior knowledge about the environment as well as the task to be performed, thus limiting the generalizability of such methods. An alternative approach, proposed in this work, is to use corrective feedback in state-space to guide agents.

There has been recent interest in Imitation Learning methods that learn using state/observation information only. This problem is termed as Imitation from Observation (IfO) and enables learning from state trajectories of humans performing the task. To compute the requisite actions, many IfO methods propose using a learnt Inverse Dynamics Model (IDM) [18, 19] which maps state transitions to the actions that produce those state transitions. However, the usage of human interaction to guide agent behavior in an IfO setting has not been studied.

In our approach, we combine the concept of action computation by learning inverse dynamics with an Interactive Learning framework. The demonstrator provides state-space corrective feedback to guide the agent's behavior towards desired states. Meanwhile an indirect inverse dynamics scheme is used to ensure the availability of the requisite actions to learn the policy.

3 Teaching Imitative Policies in State-space (TIPS)

The principle of TIPS is to allow the agent to execute its policy while a human demonstrator observes and suggests modifications to the state visited by the agent at any given time. This feedback is then used to update the agents policy on-line i.e. during the execution itself. The learning framework of TIPS is similar to another corrective feedback method: D-COACH (Deep-COrrective Advice Communicated by Humans) [17, 11]. However, while D-COACH uses feedback in the action-space, TIPS uses feedback in state-space.

Human feedback (h_t) is in the form of binary signals implying an increase/decrease in the value of a state (i.e. $h_t \in \{-1, 0, +1\}$). Each dimension of the state has a corresponding feedback signal. To convert this binary signal into a modification value, an error constant hyper-parameter (e) is chosen. Thus the human desired state (s^{des}_{t+1}) is computed as:

$$s^{des}_{t+1} = s_t + h_t.e. \tag{1}$$

The feedback (h_t) and the desired modification can be both in the full state or partial state. Thus, the demonstrator is allowed to only suggest modifications in the partial state dimensions that are well understood or easy to observe for the demonstrator. Moreover, the binary feedback mechanism is simpler than providing an exact value of the desired state. Even though the state computed using binary feedback (s^{des}_{t+1}) may be larger than what the demonstrator is suggesting, Celemin and Ruiz-del Solar [11] and Pérez-Dattari et al. [17] have shown that it is sufficient to capture the *trend* of modification. This is because, when updating the policy, information from past feedback is also used via a replay memory mechanism. The idea is that if a sequence of feedback provided in a state is in the same direction (increase/decrease), the demonstrator is suggesting a large magnitude

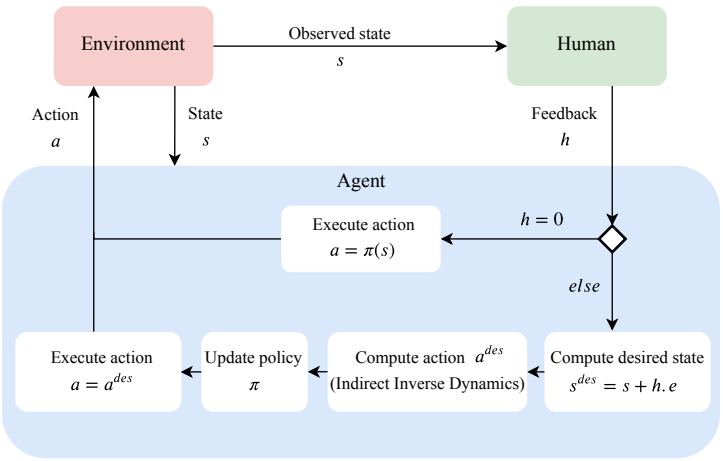


Figure 1: High-level representation of the learning framework of TIPS

change. Conversely, if the feedback alternates between increase/decrease, a smaller change around a set-point is suggested [11].

The action (a_t^{des}) required to realize the transition from the current state to the desired state i.e $s_t \rightarrow s_{t+1}^{des}$ is computed using a indirect inverse dynamics computation. Since the desired state transition could be in the partial state dimension or could be infeasible, directly using an IDM is ill-suited. Instead, we sample possible actions ($a \in A$) and use a learnt forward dynamics model (f) to predict the next states ($\hat{s}_{t+1} = f(s_t, a)$) for these actions. The action that results in a subsequent state that is closest to the desired state *in the partial state dimensions* is chosen. Mathematically, we can write this as:

$$a_t^{des} = \arg\min_a \left|\left| f(s_t, a) - s_{t+1}^{des} \right|\right|, \quad \text{where } a \in A \text{ (N_a uniform samples).} \tag{2}$$

The policy ($\pi(s)$) is trained in a supervised learning fashion using the state-action pair (s_t, a_t^{des}). Training can differ based on the type of representation function used for the policy. We represent policies using feed-forward artificial neural networks and use a similar training mechanism as D-COACH [17]. This involves an immediate training step using the current state-action sample as well as a training step using a batch sampled from a demonstration replay memory. Lastly, to ensure sufficient learning iterations to train the neural network, a batch replay training step is also carried out periodically every T_{update} time-steps.

Crucially, the computed action a_t^{des} is also executed immediately by the agent. This helps speed up the learning process since further feedback can be received in the demonstrator requested state to learn the next action to be taken. Moreover, executing the computed action a_t^{des} also makes it easy for the human demonstrator to observe the effect of the feedback that they have just provided. An implication of this however is that the action needs to be computed in real-time. A high-level view of the learning framework can be seen in Figure 1.

The full procedure of TIPS can be seen in Algorithm 1.

- In an initial Model-learning phase, samples are generated by executing an exploration policy π_e (random policy in our implementation) and used to learn an initial FDM f_θ. The samples are added to an experience buffer E that is used later when updating the model. The FDM is represented by a feed-forward neural network.
- In the teaching phase, the policy π_ϕ is trained using an immediate training step as well as a periodic step using past feedback from a demonstration buffer D. Moreover, to improve the

FDM, it is trained after every episode using the consolidated new and previous experience gathered in E.

Algorithm 1: Teaching Imitative Policies in State-space (TIPS)

Initial Model-Learning Phase:

Require: Number of exploration samples N_e
Initialize:
Forward-dynamics model f_θ,
Exploration policy π_e (random/prior knowledge),
Experience buffer $E = []$
for $i = 1, 2...N_e$ **do**
  Generate sample (s_i, a_i, s_{i+1}) by executing policy π_e
  Append sample to experience buffer E
end
Learn model f_θ using inputs $\{s_i, a_i\}_1^{N_e}$ and targets $\{s_{i+1}\}_1^{N_e}$

Teaching Phase:

Require: Forward-dynamics model f_θ, Experience buffer E, Number of action samples N_a, Error constant e, Periodic policy update interval T_{update}
Initialize:
Agent policy π_ϕ (random/prior knowledge),
Demonstration buffer $D = []$
for *episodes* **do**
  for $t = 0, 1, 2.....T$ **do**
    Visit state s_t
    Get human corrective feedback h_t
    if h_t is not 0 **then**
      $error_t = h_t.e$
      Compute desired state $s_{t+1}^{des} = s_t + error_t$
      Compute action $a_t^{des} = \arg\min_a \left|\left| f_\theta(s_t, a) - s_{t+1}^{des} \right|\right|$, using N_a sampled actions
      Append (s_t, a_t^{des}) to demonstration buffer D
      Update policy π_ϕ using pair (s_t, a_t^{des}) and using batch sampled from D
      Execute action $a_t = a_t^{des}$, reach state s_{t+1}
      Append (s_t, a_t, s_{t+1}) to experience buffer E
    else
      ## No feedback
      Execute action $a_t = \pi_\phi(s_t)$, reach state s_{t+1}
      Append (s_t, a_t, s_{t+1}) to experience buffer E
    end
    if $\mathrm{mod}(t, T_{update})$ **then**
      Update policy π_ϕ using batch sampled from demonstration buffer D
    end
  end
  Update learnt FDM f_θ using samples from experience buffer E
end

4 Experimental Setting

Our implementation of TIPS is available at github.com/sjauhri/Interactive-Learning-in-State-space. The forward dynamics model (f) and policy (π) are represented as feed-forward artificial neural networks. Training is done in a standard supervised learning fashion and for optimization the Adam variant of stochastic gradient descent [20] is used. Both the model and the policy networks are initialized with random small weights. The parameter settings for each of the domains/tasks in the experiments can be seen in Table 1.

Table 1: Parameter settings in the implementation of TIPS for different evaluation tasks

	CartPole	Reacher	LunarLander	Robot-Fishing
Number of exploration samples (N_e)	500	10000	20000	5000
States for feedback	Pole tip position	x-y position of end effector	Vertical, angular position	x-z position of end effector
Error constant (e)	0.1	0.008	0.15	0.05
Number of action samples (N_a)	10	500	500	1000
Periodic policy update interval (T_{update})	10	10	10	10
FDM Network (f_θ) layer sizes	16, 16	64, 64	64, 64	32, 32
Policy Network (π_ϕ) layer sizes	16, 16	32, 32	32, 32	32, 32
Learning rate	0.005	0.005	0.005	0.005
Batch size	16	32	32	32

4.1 Evaluation:

The tasks used for the evaluation of TIPS are chosen from the OpenAI gym toolkit [21]. The environments chosen are: CartPole, Reacher and LunarLanderContinuous. The reward obtained by the agent during execution is used as a performance metric.

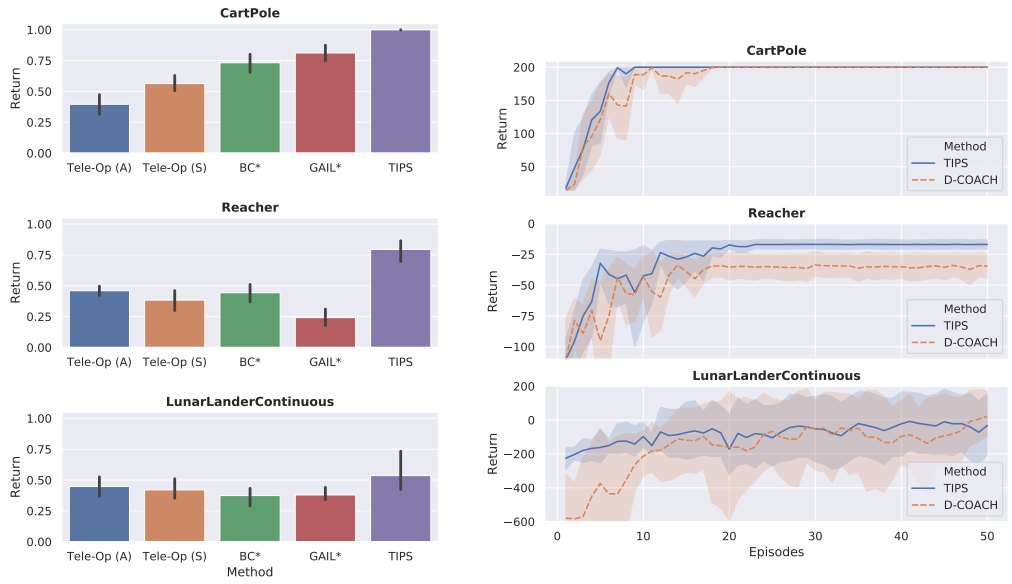
TIPS is evaluated and compared to other techniques based on two main criteria: (i) the task performance of the trained agent over time and (ii) the total demonstration effort required by the human teacher. The performance of a TIPS agent is compared against the demonstrator's own performance when executing the task via tele-operation, and against other agents trained via IL techniques using the tele-operation data. It is also of interest to highlight the differences between demonstration in state-space versus action-space. For this, both tele-operation and corrective feedback learning techniques in state and action spaces are compared.

The following techniques are used for comparison.

- **Tele-operation in Action-space:** Demonstrator executes task by providing action values.
- **Tele-operation in State-space:** Demonstrator executes task by providing state-space information (as per Table 1) with actions computed in a similar way as TIPS.
- **Behavioral Cloning (BC):** Supervised learning to imitate the demonstrator using state-action demonstration data recorded during tele-operation. (Only successful demonstrations are used, i.e. those with a return of atleast 40% in the min-max range).
- **Generative Adversarial Imitation Learning (GAIL) [22]:** Method that uses adversarial learning to learn a reward function and policy. Similar to BC, the successful state-action demonstration data is used for imitation. GAIL implementation by Hill et al. [23] is used.
- **D-COACH [17]:** Interactive IL method that uses binary corrective feedback in the action space. The demonstrator suggests modifications to the current actions being executed to train the agent as it executes the task.

Experiments are run with non-expert human participants who have no prior knowledge of the tasks. A total of 22 sets of experiments are performed, with 8, 8 and 6 participants for the CartPole, Reacher and LunarLander tasks respectively. The participants belong to an age group of 25 to 30 years. Participants performed four experiments: Tele-operation in action-space and state-space, training an agent using D-COACH and training an agent using TIPS. To compensate for learning effect the order of the experiments is changed for every participant.

When performing tele-operation, the demonstrator provided actions and the corresponding states are recorded. Tele-operation is deemed to be complete once no new demonstrative information can be provided. When training interactively using D-COACH and TIPS, the demonstrator provides feedback until no more agent performance improvement is observed. The maximum number of episodes for all the experiments is set to 50. To compare the demonstration effort, participants are asked to fill out the NASA Task Load Index Questionnaire [24] after each experiment wherein ratings related to the mental demand, time pressure etc. of the experiment are obtained.



(a) (b)

Figure 2: Evaluation results of TIPS. (a) Performance comparison of Tele-Operation (Action space), Tele-Operation (State space), BC, GAIL and TIPS. The return is normalized for each environment and averaged over multiple episodes and over all participants. BC* and GAIL* use only successful tele-operation data (return of atleast 40% in the normalized range). (b) Performance of TIPS (state-space feedback) and D-COACH (action-space feedback) agents over training episodes.

4.2 Validation task:

To validate the application of TIPS on a real robot, a robotic 'fishing' task is designed. The KUKA LBR iiwa 7 robotic manipulator is used, with a ball attached to its end-effector by a thread and the task is to swing the ball into a nearby cup. To reduce the complexity of the task, the movement of the robot end-effector (and ball) is restricted to a 2-D x-z Cartesian plane. This is done by restricting the usage of the robot's joints (only the 2nd, 4th and 6th joints are used). An illustration of the task can be seen in Figure 3a.

The robot is controlled using a position controller that controls the joint angles. The actions taken by the robot are in the form of relative changes to the joint angles. To teach the task using TIPS, feedback is provided in the x-z Cartesian position of the end-effector. A forward dynamics model in the joint-angle space is learnt. Thus the model is used to predict the position of the end-effector based on the joint angle commands (actions) sent to the robot.

To measure task performance, a reward function is defined for use as a metric. The function is inspired from the Reacher task and consists of a negative reward at every time-step based on the magnitude of the action command sent to the robot as well as the distance between the ball and the centre of the cup ($r_t = -\|a_t\| - \| dist_t \|$). Note that since this experiment is run only to validate the application of TIPS to a real system, comparisons are not made with other learning methods.

5 Results

5.1 Performance

Figure 2a shows the performance obtained for the tasks (averaged over all participants) using tele-operation, agents trained via IL techniques and agents trained using TIPS. It is noticed that tele-operation is challenging for the demonstrator, especially for time-critical tasks such as CartPole and

Table 2: Average ratings provided by the participants in the NASA Task Load Index questionnaire [24]. Values are normalized, with smaller magnitude implying lower mental demand etc. (S) and (A) are used to denote state-space and action-space techniques respectively.

	Mental Demand	Physical Demand	Temporal Demand	1-Performance	Effort	Frustration
CartPole						
TIPS (S)	**0.29**	0.33	**0.33**	0.11	0.37	**0.19**
D-COACH (A)	**0.49**	0.37	**0.43**	0.14	0.44	**0.3**
Reacher						
TIPS (S)	**0.53**	0.64	0.61	0.17	0.63	**0.3**
D-COACH (A)	**0.63**	0.66	0.57	0.2	0.61	**0.41**
LunarLanderContinuous						
TIPS (S)	0.8	0.7	0.67	0.3	0.73	0.73
D-COACH (A)	0.8	0.77	0.67	0.27	0.73	0.6

LunarLander where the system is inherently unstable. Agents trained using IL techniques (BC and GAIL) suffer from inconsistency as well as lack of generalization of the demonstrations. For the CartPole task, this problem is not as significant given the small state-action space. Interactively learning via TIPS enables continuous improvement over time and leads to the highest agent performance.

Figure 2b compares the performance of state-space interactive learning (TIPS) and action-space interactive learning (D-COACH) over training episodes. The advantage of state-space feedback is significant for the Reacher and CartPole tasks with higher agent performance and learning efficiency observed. For the LunarLander task, no performance improvement is observed. While state-space feedback provides a stabilizing effect on the lander and leads to fewer crashes, participants struggle to teach it to land and thus the agent ends up flying out of the frame.

5.2 Demonstrator Effort

The NASA Task Load Index ratings are used to capture demonstrator effort when teaching using state-space (TIPS) and action-space (D-COACH) feedback and the results can be seen in Table 2. Significant differences in rating are highlighted.

When teaching using TIPS, participants report lower ratings for the CartPole and Reacher tasks with the mental demand rating reduced by about 40% and 16% and participant frustration reduced by about 35% and 25% respectively. Thus, the merits of state-space interactive learning are clear. However, these advantages are task specific. For the LunarLander task, demonstration in state and action-spaces is equally challenging, backed up by little change in the ratings.

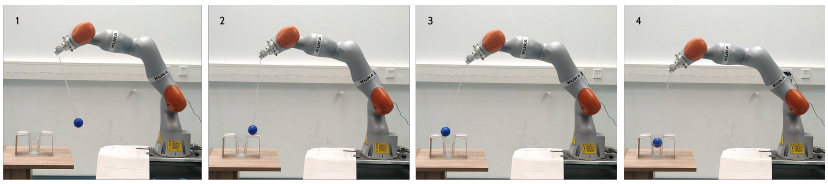
It is noted that actions computed based on feedback using TIPS can be irregular due to inaccuracies in model learning. This was observed for the Reacher and LunarLander tasks where model learning is relatively more complex as compared to CartPole. Since handling such irregular action scenarios requires demonstrator effort, this can diminish the advantage provided by state-space feedback.

5.3 Validation Results

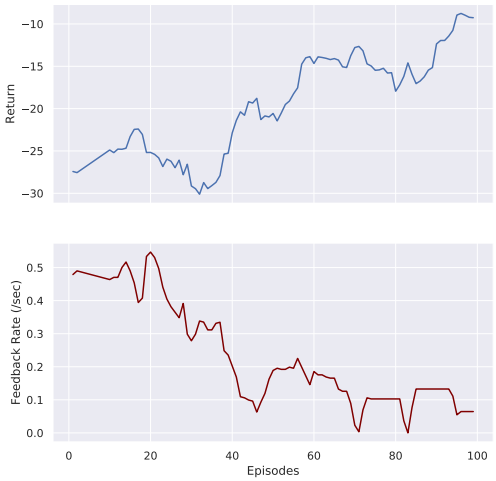
The robotic-fishing task is taught to an agent using TIPS, the results of which are visualized in Figure 3. Since the objective of the experiment is to validate the application of TIPS on a real system, comparisons with other learning techniques are not made.

In our experiment, the demonstrator's strategy is to move the end effector towards a position above the cup and choose the appropriate moment to bring the end effector down such that the ball falls into the cup. This behavior is sucessfully imitated by the agent (shown in Figure 3a).

The agent performance and demonstrator feedback rate over learning episodes can be seen in Figure 3b. The agent successfully learns to reliably place the ball in the cup (return of -15) after 60 episodes of training (each episode is 30 seconds long). After about 90 episodes, the agent performance is further improved in terms of speed at which the task is completed (return of -10). The feedback rate reduces over time as the agent performs better and only some fine-



(a)



(b)

Figure 3: Results from the validation experiment on the KUKA robot. (a) Left to right, the fishing task performed by the robot after being taught by the demonstrator for 60 episodes using TIPS. (Episode length is 30 seconds) (b) Return obtained as per our defined reward function and demonstrator feedback rate over learning episodes. (Values are averaged over a rolling window of size 10)

tuning of the behavior is needed after 60 episodes. A video of the learnt behavior is available at: https://youtu.be/aN1r8IytsXY.

6 Conclusion

In experiments with non-expert human demonstrators, our proposed method TIPS outperforms IL techniques such as BC and GAIL [22] as well as Interactive Learning in action-space (D-COACH [17]). Moreover, the state-space feedback mechanism also leads to a significant reduction in demonstrator effort. With these results we have illustrated the viability of TIPS to non-expert demonstration scenarios and have also highlighted the merits of state-space Interactive Learning.

A caveat of the proposed method however is that, to compute actions, an accurate forward dynamics model needs to be learnt which can be challenging and can require a large number of environment interactions. Another consideration is that, for action selection, candidate actions are sampled from the entire action-space. This is not scalable to high-dimensional continuous action spaces. Despite these caveats, we have demonstrated the practical viability of TIPS on a real system by training an agent for a robotic fishing task.

For the extension of this work, a main focus would be towards improving on the current action computation mechanism. One approach could be to use smarter exploration strategies for acquiring experience samples to learn a more accurate forward dynamics model. Another direction to explore would be to compute multiple actions over a time horizon to reach the human feedback desired state. An optimization method could be used, making the action computation similar to a Model Predictive Control scheme and it would be interesting to see such a scheme could be combined with our learning framework.

References

[1] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009. doi:10.1016/j.robot.2008.10.024.

[2] J. Kober and J. Peters. Learning motor primitives for robotics. In *2009 IEEE International Conference on Robotics and Automation*, pages 2112–2118. IEEE, 2009. doi:10.1109/ROBOT.2009.5152577.

[3] S. Schaal. Learning from demonstration. In *Advances in neural information processing systems*, pages 1040–1046, 1997. URL `http://papers.nips.cc/paper/1224-learning-from-demonstration.pdf`.

[4] H. Daumé III. A course in machine learning. *Publisher, ciml. info*, 5:69, 2012.

[5] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, J. Peters, et al. An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, 7(1-2):1–179, 2018. doi:10.1561/9781680834116.

[6] M. Zucker, N. Ratliff, M. Stolle, J. Chestnutt, J. A. Bagnell, C. G. Atkeson, and J. Kuffner. Optimization and learning for rough terrain legged locomotion. *The International Journal of Robotics Research*, 30(2):175–191, 2011.

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

[8] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 01 2016. doi:10.1038/nature16961.

[9] B. D. Argall, B. Browning, and M. Veloso. Learning robot motion control with demonstration and advice-operators. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 399–404. IEEE, 2008. doi:10.1109/IROS.2008.4651020.

[10] S. Chernova and M. Veloso. Interactive policy learning through confidence-based autonomy. *Journal of Artificial Intelligence Research*, 34:1–25, 2009. doi:10.1613/jair.2584.

[11] C. Celemin and J. Ruiz-del Solar. An interactive framework for learning continuous actions policies based on corrective feedback. *Journal of Intelligent & Robotic Systems*, 95(1):77–97, 2019. doi:10.1007/s10846-018-0839-z.

[12] S. Ross, G. Gordon, and D. 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.

[13] Y. Liu, A. Gupta, P. Abbeel, and S. 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. doi:10.1109/ICRA.2018.8462901.

[14] B. D. Argall, B. Browning, and M. M. Veloso. Teacher feedback to scaffold and refine demonstrated motion primitives on a mobile robot. *Robotics and Autonomous Systems*, 59(3-4):243–255, 2011. doi:10.1016/j.robot.2010.11.004.

[15] W. B. Knox and P. Stone. Interactively shaping agents via human reinforcement: The tamer framework. In *Proceedings of the Fifth International Conference on Knowledge Capture*, K-CAP 09, page 916, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605586588. doi:10.1145/1597735.1597738.

[16] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, pages 4299–4307, 2017. URL `https://arxiv.org/abs/1706.03741v3`.

[17] R. Pérez-Dattari, C. Celemin, J. Ruiz-del Solar, and J. Kober. Continuous control for high-dimensional state spaces: An interactive learning approach. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7611–7617. IEEE, 2019. URL https://arxiv.org/abs/1908.05256.

[18] A. Nair, D. Chen, P. Agrawal, P. Isola, P. Abbeel, J. Malik, and S. 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. doi:10.1109/ICRA.2017.7989247.

[19] F. Torabi, G. Warnell, and P. Stone. Behavioral cloning from observation. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 4950–4957. International Joint Conferences on Artificial Intelligence Organization, 7 2018. doi:10.24963/ijcai.2018/687.

[20] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. URL https://arxiv.org/abs/1412.6980.

[21] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016. URL https://arxiv.org/abs/1606.01540.

[22] J. Ho and S. Ermon. Generative adversarial imitation learning. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4565–4573. Curran Associates, Inc., 2016. URL http://papers.nips.cc/paper/6391-generative-adversarial-imitation-learning.pdf.

[23] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu. Stable baselines. https://github.com/hill-a/stable-baselines, 2018.

[24] S. G. Hart and L. E. Staveland. Development of nasa-tlx (task load index): Results of empirical and theoretical research. In *Advances in Psychology*, volume 52, pages 139–183. Elsevier, 1988. doi:10.1016/S0166-4115(08)62386-9.

[25] C. Hennersperger, B. Fuerst, S. Virga, O. Zettinig, B. Frisch, T. Neff, and N. Navab. Towards mri-based autonomous robotic us acquisitions: a first feasibility study. *IEEE transactions on medical imaging*, 36(2):538–548, 2017.

B

Task Load Index Questionnaire

Following is the NASA Task Load Index [Hart and Staveland, 1988] questionnaire provided to participants after each experiment to capture the demonstration effort:

Question	Response options
Mental Demand: How much mental and perceptual activity was required (e.g., thinking, deciding, calculating, remembering, looking, searching etc.)?	Low [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10] High
Physical Demand: How much physical activity was required (e.g.. turning, controlling, activating etc.)?	Low [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10] High
Temporal Demand: How much time pressure did you feel due to the rate or pace at which the tasks or task elements occurred?	Low [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10] High
Performance: How successful do you think you were in accomplishing the goals of the task set by the experimenter (or yourself)?	Poor [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10] Good
Effort: How hard did you have to work (mentally and physically) to accomplish your level of performance?	Low [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10] High
Frustration: How insecure, discouraged, irritated, stressed and annoyed versus secure, gratified, content, relaxed and complacent did you feel during the task?	Low [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10] High

Bibliography

Abbeel, P., Coates, A., and Ng, A. Y. (2010). Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 29(13):1608–1639.

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

Akrour, R., Schoenauer, M., and Sebag, M. (2011). Preference-based policy learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 12–27. Springer.

Akrour, R., Schoenauer, M., Sebag, M., and Souplet, J.-C. (2014). Programming by feedback. In *International Conference on Machine Learning*, volume 32, pages 1503–1511.

Argall, B. D., Browning, B., and Veloso, M. (2008). Learning robot motion control with demonstration and advice-operators. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 399–404. IEEE.

Argall, B. D., Browning, B., and Veloso, M. M. (2011). Teacher feedback to scaffold and refine demonstrated motion primitives on a mobile robot. *Robotics and Autonomous Systems*, 59(3-4):243–255.

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

Aytar, Y., Pfaff, T., Budden, D., Paine, T., Wang, Z., and de Freitas, N. (2018). Playing hard exploration games by watching youtube. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31*, pages 2930–2941. Curran Associates, Inc.

Bagnell, J. A. D. (2015). An invitation to imitation. Technical Report CMU-RI-TR-15-08, Carnegie Mellon University, Pittsburgh, PA.

Bain, M. and Sammut, C. (1995). A framework for behavioural cloning. In *Machine Intelligence 15*, pages 103–129.

Billard, A., Calinon, S., Dillmann, R., and Schaal, S. (2008). Robot Programming by Demonstration. In Siciliano, B. and Khatib, O., editors, *Springer Handbook of Robotics*, pages 1371–1394. Springer Berlin Heidelberg, Berlin, Heidelberg.

Billard, A. and Matarić, M. J. (2001). Learning human arm movements by imitation:: Evaluation of a biologically inspired connectionist architecture. *Robotics and Autonomous Systems*, 37(2-3):145–160.

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., et al. (2016). End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym.

Brown, D. S., Goo, W., Nagarajan, P., and Niekum, S. (2019). Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. *arXiv preprint arXiv:1904.06387*.

Calinon, S. and Billard, A. (2009). Statistical learning by imitation of competing constraints in joint space and task space. *Advanced Robotics*, 23(15):2059–2076.

Celemin, C. and Ruiz-del Solar, J. (2019). An interactive framework for learning continuous actions policies based on corrective feedback. *Journal of Intelligent & Robotic Systems*, 95(1):77–97.

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

Chernova, S. and Veloso, M. (2009). Interactive policy learning through confidence-based autonomy. *Journal of Artificial Intelligence Research*, 34:1–25.

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., and Amodei, D. (2017). Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, pages 4299–4307.

Curi, S., Levy, K. Y., and Krause, A. (2018). Unsupervised imitation learning. *arXiv preprint arXiv:1806.07200*.

Daumé III, H. (2012). A course in machine learning. *Publisher, ciml. info*, 5:69.

Dwibedi, D., Tompson, J., Lynch, C., and Sermanet, P. (2018). Learning actionable representations from visual observations. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1577–1584.

Edwards, A. D., Sahni, H., Schroecker, Y., and Isbell, C. L. (2018). Imitating latent policies from observation. *arXiv preprint arXiv:1805.07914*.

Finn, C., Levine, S., and Abbeel, P. (2016). Guided cost learning: Deep inverse optimal control via policy optimization. In *International conference on machine learning*, pages 49–58.

French, R. M. (1999). Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135.

Gillijns, S. and De Moor, B. (2007). Unbiased minimum-variance input and state estimation for linear discrete-time systems with direct feedthrough. *Automatica*, 43(5):934–937.

Giusti, A., Guzzi, J., Cireşan, D. C., He, F.-L., Rodríguez, J. P., Fontana, F., Faessler, M., Forster, C., Schmidhuber, J., Di Caro, G., et al. (2015). A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters*, 1(2):661–667.

Goo, W. and Niekum, S. (2019). 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.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc.

Gribovskaya, E., Khansari-Zadeh, S. M., and Billard, A. (2011). Learning non-linear multivariate dynamics of motion in robotic manipulators. *The International Journal of Robotics Research*, 30(1):80–117.

Griffith, S., Subramanian, K., Scholz, J., Isbell, C. L., and Thomaz, A. L. (2013). Policy shaping: Integrating human feedback with reinforcement learning. In *Advances in neural information processing systems*, pages 2625–2633.

Grimes, D. B., Rashid, D. R., and Rao, R. P. (2007). Learning nonparametric models for probabilistic imitation. In *Advances in Neural Information Processing Systems*, pages 521–528.

Grzes, M. and Kudenko, D. (2009). Theoretical and empirical analysis of reward shaping in reinforcement learning. In *2009 International Conference on Machine Learning and Applications*, pages 337–344. IEEE.

Guo, X., Chang, S., Yu, M., Tesauro, G., and Campbell, M. (2019). Hybrid reinforcement learning with expert state sequences. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3739–3746.

Hanna, J. P. and Stone, P. (2017). Grounded action transformation for robot learning in simulation. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)*.

Hart, S. G. and Staveland, L. E. (1988). Development of nasa-tlx (task load index): Results of empirical and theoretical research. In *Advances in Psychology*, volume 52, pages 139–183. Elsevier.

Henderson, P., Chang, W. D., Bacon, P. L., Meger, D., Pineau, J., and Precup, D. (2018). OptionGAN: Learning joint reward-policy options using generative adversarial inverse reinforcement learning. *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pages 3199–3206.

Hennersperger, C., Fuerst, B., Virga, S., Zettinig, O., Frisch, B., Neff, T., and Navab, N. (2017). Towards mri-based autonomous robotic us acquisitions: a first feasibility study. *IEEE transactions on medical imaging*, 36(2):538–548.

Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Osband, I., et al. (2018). Deep q-learning from demonstrations. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Hill, A., Raffin, A., Ernestus, M., Gleave, A., Kanervisto, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., and Wu, Y. (2018). Stable baselines. `https://github.com/hill-a/stable-baselines`.

Ho, J. and Ermon, S. (2016). Generative adversarial imitation learning. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems 29*, pages 4565–4573. Curran Associates, Inc.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Jain, A., Wojcik, B., Joachims, T., and Saxena, A. (2013). Learning trajectory preferences for manipulators via iterative improvement. In *Advances in neural information processing systems*, pages 575–583.

Jaynes, E. T. (1957). Information theory and statistical mechanics. *Phys. Rev.*, 106:620–630.

Kimura, D., Chaudhury, S., Tachibana, R., and Dasgupta, S. (2018). Internal model from observations for reward shaping. *arXiv preprint arXiv:1806.01267*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Knox, W. B. and Stone, P. (2009). Interactively shaping agents via human reinforcement: The tamer framework. In *Proceedings of the Fifth International Conference on Knowledge Capture*, K-CAP '09, page 9–16, New York, NY, USA. Association for Computing Machinery.

Knox, W. B., Stone, P., and Breazeal, C. (2013). Training a robot via human feedback: A case study. In *International Conference on Social Robotics*, pages 460–470. Springer.

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

Kober, J. and Peters, J. (2009). Learning motor primitives for robotics. In *2009 IEEE International Conference on Robotics and Automation*, pages 2112–2118. IEEE.

Laskey, M., Lee, J., Hsieh, W., Liaw, R., Mahler, J., Fox, R., and Goldberg, K. (2017). Iterative noise injection for scalable imitation learning. *arXiv preprint arXiv:1703.09327*.

Levine, S. (2018). Deep reinforcement learning - cs 294-112, lecture slides.

Levine, S. and Koltun, V. (2012). Continuous inverse optimal control with locally optimal examples. *arXiv preprint arXiv:1206.4617*.

Levine, S., Popovic, Z., and Koltun, V. (2011). Nonlinear inverse reinforcement learning with gaussian processes. In *Advances in Neural Information Processing Systems*, pages 19–27.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321.

Liu, Y., Gupta, A., Abbeel, P., and Levine, S. (2018). 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.

Menda, K., Driggs-Campbell, K., and Kochenderfer, M. J. (2018). Ensembledagger: A bayesian approach to safe imitation learning. *arXiv preprint arXiv:1807.08364*.

Merel, J., Tassa, Y., Srinivasan, S., Lemmon, J., Wang, Z., Wayne, G., and Heess, N. (2017). Learning human behaviors from motion capture by adversarial imitation. *arXiv preprint arXiv:1707.02201*.

Nair, A., Chen, D., Agrawal, P., Isola, P., Abbeel, P., Malik, J., and Levine, S. (2017). 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.

Nair, A., McGrew, B., Andrychowicz, M., Zaremba, W., and Abbeel, P. (2018). Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6292–6299. IEEE.

Najar, A., Sigaud, O., and Chetouani, M. (2016). Training a robot with evaluative feedback and unlabeled guidance signals. In *2016 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 261–266. IEEE.

Ng, A. Y., Russell, S. J., et al. (2000). Algorithms for inverse reinforcement learning. In *Icml*, volume 1, pages 663–670.

Nguyen-Tuong, D., Peters, J., Seeger, M., and Schölkopf, B. (2008). Learning inverse dynamics: a comparison. In *European symposium on artificial neural networks*.

Osa, T., Pajarinen, J., Neumann, G., Bagnell, J. A., Abbeel, P., Peters, J., et al. (2018). An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, 7(1-2):1–179.

Pathak, D., Mahmoudieh, P., Luo, G., Agrawal, P., Chen, D., Shentu, Y., Shelhamer, E., Malik, J., Efros, A. A., and Darrell, T. (2018). Zero-shot visual imitation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 2050–2053.

Pavse, B. S., Torabi, F., Hanna, J. P., Warnell, G., and Stone, P. (2019). RIDM: Reinforced Inverse Dynamics Modeling for Learning from a Single Observed Demonstration. *arXiv e-prints*, page arXiv:1906.07372.

Pérez-Dattari, R., Celemin, C., Ruiz-del Solar, J., and Kober, J. (2019). Continuous control for high-dimensional state spaces: An interactive learning approach. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7611–7617. IEEE.

Pomerleau, D. A. (1989). Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313.

Rahmatizadeh, R., Abolghasemi, P., Behal, A., and Bölöni, L. (2018). From virtual demonstration to real-world manipulation using lstm and mdn. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Ratliff, N. D., Bagnell, J. A., and Zinkevich, M. A. (2006). Maximum margin planning. In *Proceedings of the 23rd international conference on Machine learning*, pages 729–736.

Ross, S. and Bagnell, D. (2010). Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668.

Ross, S., Gordon, G., and Bagnell, D. (2011). 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.

Russell, S. (1998). Learning agents for uncertain environments. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 101–103.

Schaal, S. (1997). Learning from demonstration. In *Advances in neural information processing systems*, pages 1040–1046.

Serfozo, R. (2009). *Basics of applied stochastic processes*. Springer Science & Business Media.

Sermanet, P., Lynch, C., Chebotar, Y., Hsu, J., Jang, E., Schaal, S., Levine, S., and Brain, G. (2018). Time-contrastive networks: Self-supervised learning from video. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1134–1141. IEEE.

Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press.

Silver, D., Huang, A., Maddison, C., Guez, A., Sifre, L., Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489.

Stadie, B. C., Abbeel, P., and Sutskever, I. (2017). Third-Person Imitation Learning. *arXiv e-prints*, page arXiv:1703.01703.

Suay, H. B. and Chernova, S. (2011). Effect of human guidance and state space size on interactive reinforcement learning. In *2011 Ro-Man*, pages 1–6. IEEE.

Sun, W., Vemula, A., Boots, B., and Bagnell, J. A. (2019). Provably Efficient Imitation Learning from Observation Alone. *arXiv e-prints*, page arXiv:1905.10948.

Sutton, R. S. and Barto, A. G. (1998). Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, 9(5):1054–1054.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033.

Torabi, F., Warnell, G., and Stone, P. (2018). Behavioral cloning from observation. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 4950–4957. International Joint Conferences on Artificial Intelligence Organization.

Torabi, F., Warnell, G., and Stone, P. (2019). Generative adversarial imitation from observation. In *ICML Workshop on Imitation, Intent, and Interaction (I3)*.

Torabi, F., Warnell, G., and Stone, P. (2019). Recent Advances in Imitation Learning from Observation. *arXiv e-prints*, page arXiv:1905.13566.

Zhang, J. and Cho, K. (2017). Query-efficient imitation learning for end-to-end simulated driving. *31st AAAI Conference on Artificial Intelligence, AAAI 2017*, pages 2891–2897.

Ziebart, B. D., Maas, A. L., Bagnell, J. A., and Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA.

Zucker, M., Ratliff, N., Stolle, M., Chestnutt, J., Bagnell, J. A., Atkeson, C. G., and Kuffner, J. (2011). Optimization and learning for rough terrain legged locomotion. *The International Journal of Robotics Research*, 30(2):175–191.

C

Glossary

List of Acronyms

IL	Imitation Learning
IIL	Interactive Imitation Learning
RL	Reinforcement Learning
IRL	Inverse Reinforcement Learning
DRL	Deep Reinforcement Learning
BC	Behavioral Cloning
MDP	Markov Decision Process
IfO	Imitation from Observation
LfD	Learning from Demonstration
IDM	Inverse Dynamics Model
FDM	Forward Dynamics Model
BCO	Behavioral Cloning from Observation
GAIL	Generative Adversarial Imitation Learning
GAILfO	GAIL from Observations
CNN	Convolutional Neural Network
GAN	Generative Adversarial Network
TRPO	Trust Region Policy Optimization
DAgger	Dataset Aggregation

COACH	COrrective Advice Communicated by Humans
D-COACH	Deep-COACH
TAMER	Training an Agent Manually via Evaluative Reinforcement
TIPS	Teaching Imitative Policies in State-space
TU Delft	Delft University of Technology