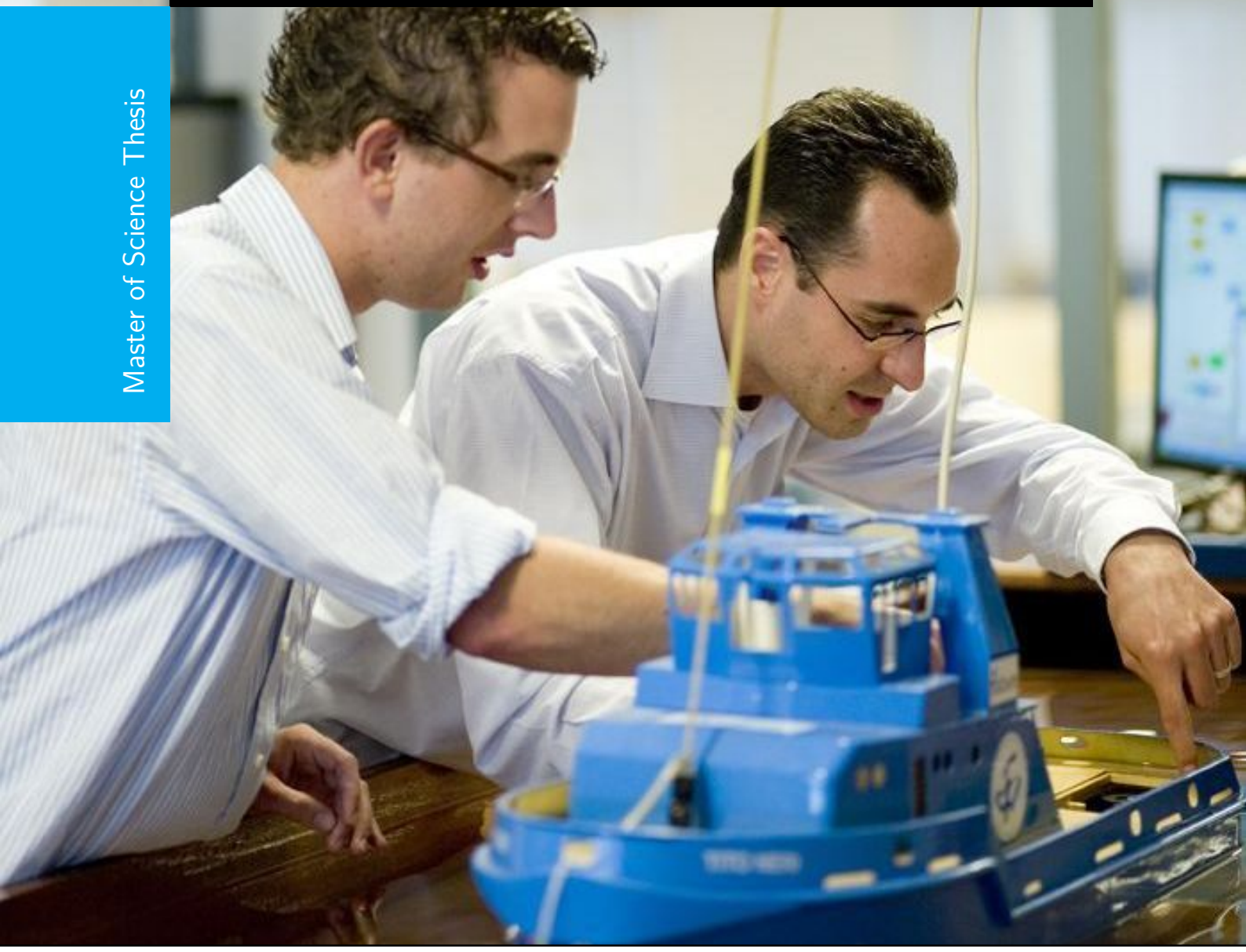
Learning Task Space Policies from Demonstration

Lalith Keerthan Suresh Kumar

Master of Science Thesis



TU Delft Delft University of Technology

Depart of Cognitive Robotics

Learning Task Space Policies from Demonstration

Master of Science Thesis

Lalith Keerthan Suresh Kumar

October 6, 2021

Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS) · Delft University of Technology

Abstract

In this thesis, we propose a method titled "Task Space Policy Learning (TaSPL)", a novel technique that learns a generalised task/state space policy $\pi(s_{t+1}|s_t)$, as opposed to learning a policy in state-action space $\pi(a_t|s_t)$, from interactive corrections in the observation space or from state only demonstration data. This task/state space policy enables the agent to execute the task when the dynamics of the environment is changed from the original environment, without the need of additional demonstrative effort from a human teacher. We achieve this by decoupling the objective task into a Task space policies and dynamics model. A Task space policy, describes how the observable states transit in order to reach the goal of a task and an Indirect Inverse Dynamics model, which is responsible for performing the action that obtains the desired transition. Thus, effectively decoupling the task objective from the dynamics of the environment. In case, the dynamics of the environment changes, only the agent's dynamics model has to be relearnt, while the task space policy can be reused.

The method was tested and compared to other imitation learning methods, for various control tasks of the OpenAI Gym toolkit in their original environment. The obtained policies were also tested in the modified environments, showing that this method can be used to obtain imitation policies with the benefits of interactive IL methods, while also being able to generalize that knowledge to several varied conditions unseen during the teacher interventions. The method was validated in two different tasks with a KUKA iiwa robot manipulator, testing generalization capabilities of the learnt policies.

Table of Contents

Acknowledgements vii

1 Introduction 1
1-1 Thesis Outline . . . 2

2 Model Learning 5
2-1 Markov decision process . . . 6
2-2 Types of models . . . 7
2-3 Different function approximators . . . 8
2-4 Data collection methods . . . 11
2-5 Region of validity . . . 11
2-6 State abstraction . . . 12
2-7 Conclusion . . . 13

3 Imitation Learning 15
3-1 Preliminaries . . . 16
3-1-1 Model-Free and Model-Based Imitation Learning . . . 16
3-2 Sequential Imitation Learning . . . 17
3-2-1 Behavioral Cloning . . . 17
3-2-2 Inverse Reinforcement Learning . . . 18
3-3 Interactive imitation learning . . . 20
3-3-1 Evaluative feedback . . . 20
3-3-2 Corrective feedback . . . 21
3-4 Imitation Learning from Observation . . . 24
3-4-1 Model based and Model free IfO algorithms . . . 24
3-5 Imitation learning in latent space . . . 26
3-6 Conclusions . . . 29

4 Task-Space Policy Learning **31**
4-1 Learning Framework . . . 32
4-2 Computing Actions via Indirect Inverse Dynamics . . . 33
4-2-1 Dynamic Model Learning . . . 34
4-3 Algorithms in Full . . . 35
4-3-1 Offline TaSPL . . . 35
4-3-2 Interactive TaSPL . . . 35
4-4 Discussion . . . 39

5 Experimental Setting **43**
5-1 Implementation of the algorithms . . . 43
5-2 Evaluation Domain - OpenAI Gym . . . 44
5-2-1 CartPole - Original environment . . . 45
5-2-2 Cart-pole - Modified environment . . . 45
5-2-3 MountainCar - Original environment . . . 46
5-2-4 MountainCar - Modified environment . . . 46
5-2-5 Pendulum - Original environment . . . 46
5-2-6 Pendulum - Modified environment . . . 46
5-2-7 Acrobot - Original environment . . . 47
5-2-8 Acrobot - Modified environment . . . 47
5-2-9 LunarLander - Original environment . . . 47
5-2-10 LunarLander - Modified environment . . . 47
5-3 Experiment Setup for Offline Policy learning . . . 48
5-3-1 Ablation Study . . . 49
5-4 Experiment Setup for Interactive Policy learning . . . 49
5-4-1 Ablation study . . . 50
5-5 Validation . . . 51
5-5-1 Car Balance Task . . . 51
5-5-2 Object pulling task . . . 53

6 Results **55**
6-1 Offline TaSPL . . . 55
6-1-1 Performance in original environment . . . 55
6-1-2 Performance in modified environment . . . 57
6-1-3 Ablation study . . . 57
6-2 Interactive TaSPL . . . 57
6-2-1 Ablation study . . . 63
6-3 Robotic Task . . . 66
6-3-1 Car Balancing task . . . 66
6-3-2 Object pulling task . . . 67
6-4 Discussion . . . 69

7 Conclusion **71**

A All Plots **73**
A-1 Offline TaSPL 73
A-1-1 Cartpole Environment Modification 73
A-1-2 Pendulum Environment Modification 75
A-1-3 Acrobot Environment Modification 76
A-2 Interactive TaSPL 77
A-2-1 Cartpole Environment Modification 77
A-2-2 Pendulum Environment Modification 79
A-2-3 Lunar Lander Environment Modification 80

B Paper **81**

Bibliography **95**

Acknowledgements

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.

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
October 6, 2021

Lalith Keerthan Suresh Kumar

Chapter 1

Introduction

In 2000, Honda Corporation created a general purpose humanoid robot called ASIMO (Advanced Step in Innovative Mobility), which could carry-out simple tasks like move objects with its 6-DOF robotic arms and move around within its environment [Sakagami et al., 2002]. The behaviour of ASIMO and other similar humanoid robots was either hard-coded for each task or requires live teleoperation by a human. Without any feedback from the robot, it is up to the human operator to gauge the environment and control the robot robustly. Furthermore, in many cases like space and deep-sea applications, communication time delay interferes with the human operator's ability to control the robot. Hence there is a need for learning approach for robot control.

The most popular approach nowadays for learning control in robots is reinforcement learning. RL enables an agent to learn an optimal behavior through trial-and-error interactions with its environment. In order to find the optimal solution to a given task, the agent needs to execute new actions and explore the state-action space. Robotic RL faces a few unique challenges in this regard as explained in Kober et al. [2013]. One, continuous interaction of the robot with the environment can be expensive (w.r.t time and w.r.t hardware, due the to wear and tear of the robot). Two, modern robots have high degrees of freedom (54 in the case of ASIMO), which makes it nearly impossible to search the entire state space for an optimal action. Hence it is necessary to either find more compact state action representations or to learn a portion of the state–action space that is relevant for the task at hand [Schaal, 1999].

If the demonstrations provided cover the entire state space, then the robot could perform the task by simply executing the recorded demonstration. However, in the real world, state and action spaces are continuous and possibly infinite, the necessary data cannot be gathered. Thus, as the goal of imitation learning is to enable the robot to accomplish the task in a novel situation, the robot must generalize the demonstrations in the form of a policy.

For Learning a policy from a set of recorded demonstrations, there are two main approaches in standard IL methods: i) directly deriving a policy from the data with supervised learning, known as Behavioral Cloning (BC) [Michie et al., 1990; Daftry et al., 2016]; ii) using Inverse Reinforcement Learning (IRL) to obtain the objective function of the task which is implicitly

described in the demonstrations, then using that learned objective function and Reinforcement Learning (RL) to train an imitation policy [Ng et al., 2000; **?**].

The former family of methods has the advantage of deriving an explicit policy without the additional computationally expensive/data hunger RL process. However, the latter, also known as indirect IL, has the advantage of being able to generalize the obtained knowledge encoded in the objective function, that could be used to learn an imitation policy in environments with different dynamics, or with mismatches between the embodiment of the teacher and the learner, i.e. without the need to record new teacher demonstrations in the changed environment.

Directly deriving a policy after collecting demonstrations has two major limitations: i) the distribution shift, given by the compound error that occurs when the learned policy shifts towards states unseen during the demonstration recording, and it is not able to recover and fulfill the task; ii) the policy performance is at most as good as the demonstrations, therefore, the methods only work for teaching tasks that the teachers are able to perform successfully. Interactive IL or IL with humans in the loop methods have recently become popular in the robot learning community [Cui et al., 2021], especially because they overcome the mentioned limitations of standard approaches. These methods iteratively keep collecting information from the teachers, while rolling out the current learning policy. These approaches allow the teacher to correct the agent when the policy performs wrong actions and/or needs to recover actions to move towards desired states.

Although interactive IL methods have been demonstrated to obtain more robust policies than with standard IL, most of them obtain explicit policies mapping from states to actions, that cannot be reused to fulfill the task objectives when the dynamics of the environment change. For example wear of the system, change of the manipulator, the tools, or the objects to be manipulated. Therefore, whenever the learned policy is not valid anymore, these methods require the intervention of the teacher to tune the policy for the new situation, i.e. requiring a higher workload from the teachers who may not be available.

In this thesis, we propose a method titled Task Space Policy Learning from demonstration, a novel imitation learning technique that learns a generalised state space policy $\pi(s_{t+1}|s_t)$, as opposed to learning a policy in state-action space $\pi(a_t|s_t)$. The learnt task space policy, enables the agent to execute the task even when the dynamics of the environment is changed, without the need additional intervention by the human teacher, therefore, reducing their workload.

1-1 Thesis Outline

This thesis report is structured in the following manner:

Chapter 2 provides information about Dynamic Model learning techniques. Firstly, some preliminary information is provided about how dynamics models are formulated and mathematically represented. We then elaborate on various model learning techniques and their advantages. Learning a dynamics model is necessary to utilise a generalised state space policy (that is independent of the dynamics of the environment), however, a reader who has basic background on model learning for control could skip it and go to chapter 3 that has more relevant literature to the topic.

Chapter 3 gives a detailed background of sequential imitation learning and interactive imitation learning, followed by a discussion on the current imitation learning from Observation (IfO) and imitation learning in latent space methods.

In Chapter 4, we explain the proposed algorithm : 'Task-Space Policy Learning' (TaSPL)- an imitation learning method that learns a generalized task policy from either state only demonstrations or interactively from corrections in state space. The chapter elaborates on the learning framework for the two proposed algorithms and the method used to infer actions that result in the desired state transition are also presented.

In Chapter 5, we elaborate on the experimental environment used to evaluate our method. This includes an explanation of the Open AI Gym tasks as well as the robotic manipulation task.

In Chapter 6, we provide the results of our experiments and show the advantage provided by TaSPL over baseline imitation learning techniques.

In Chapter 7, some conclusions and possible future improvements to this work are provided.

Appendix A consists of all the results of the algorithm tested.

Appendix B consists of a draft version of the paper that this thesis work resulted in.

Chapter 2

Model Learning

A model is a mathematical representation of systems or events. In control theory, a model is used for understanding the underlying working mechanism of a system, for instance, in order to make predictions on how the system will behave with or without external input. Models are commonly used for optimal control, by deciding on an action that will lead to the next desirable state before they are executed. The most commonly used method for modeling dynamical systems is by using ordinary differential equations (ODEs), which are derived using physics-based approaches that rely on the knowledge of the process. In cases where the system's dynamics are too complex and cannot be accurately modelled, an empirical model can be derived from data collected from a series of experiments. And in cases where a theoretical model is known, experimentation data can be used for tuning the theoretical model by parameter estimation. These methods are commonly known as **model learning**.

For most systems, the model dynamics can be derived using the first principles. For a car, we know the vehicle dynamics, we can predict how the motion of the car is affected by the input actions, steering angle, velocity, etc. There are systems for which models of dynamics are very tedious to construct like robotic manipulators with deformable surfaces. Another example is when a robot interacts with humans; human users are unpredictable/ difficult to model. Therefore there are situations where one might not know exactly how executing an action in the current state can lead to the next state. But in order to control the system, you'd like to have an algorithm that can predict the model's next state without actually executing the control action. Therefore it is necessary to learn the dynamics model so that one can use standard optimal control algorithms (like LQR, MPC, model-based reinforcement learning, etc.) to carry out the desired task.

In the field of robotics, a model describes the kinematics and dynamics capabilities of the robot and how it interacts with external objects. Robots rely on sensor and motor functions to perform a wide range of tasks. However, programming an autonomous robot to perform a task, by designing controllers for each task is often time-consuming and infeasible to program for all possible scenarios. The ability to predict the consequence of actions has several uses in robotics, such as learning controllers for executing a particular task or imitating the actions of

humans or other robots. Another added advantage to learning a model is that the model can also be used to improve the robot's performance and to adapt it to a specific environment.

Learning models is important in many decision-making approaches that require understanding the evolution of the dynamics of the environment. Section 2-1 gives details about Markov decision process, a sequential decision method that is most commonly used in Robot Learning (Reinforcement Learning, and Imitation Learning).

A transition function describes how the system evolves from one state to another as a result of the previous action. A dynamical model of a system can be written as a state transition function f whose inputs are the current state the system is in s_t, and the input action given to the system a_t, while the output is the next state s_{t+1}, which the system ends up in after executing the action, mathematically it can be written as :

$$s_{t+1} = f(s_t, a_t) \tag{2-1}$$

where f represents the state transition function.

Based on the argument of the function, we can classify the models as forward dynamics models, inverse dynamics models and multi-step prediction models. Section 2-2 provides more descriptions in detail. These transition functions can be deterministic or stochastic in nature, and modelled with different kinds of function approximators, section 2-3 provides more description in detail. Since model learning requires ground truth data for training the models, section 2-4 provides more information on the different data collection techniques.

2-1 Markov decision process

A model of a robot executing a task describes interactions of a robot with its surrounding environment into a useful framework that can be used to predict possible future states from the current state. One such framework is Markov Decision Process (MDP) [Puterman, 2014]. It is commonly used in machine learning for modeling decision making in a sequential environment. Many problems in the field of robotics regarding planning and control can be modelled as an MDP.

The setting in which a robot executes its task is called an **environment**. An environment can be represented by a finite set of **states** that can fully describe the configuration of all entities. The control system or the decision maker which can sense the state and execute an action to change the environment is called the **agent**. **Actions** are used to control/change the system state in order to achieve a goal. At time step t, the agent senses the environment's state, $s_t \subseteq \mathcal{S}$, (where $\mathcal{S}$ is the set of possible states), and executes an action, $a_t \subseteq \mathcal{A}$, (where $\mathcal{A}$ is the set of actions), which results in the agent reaching a new state s_{t+1} at time $t+1$. For example, a robotic arm interacting with an external object, a complete set of robot actuator angles and angular velocities along with the position and velocities of the object form a complete state s_t. The set of all torques the actuators can execute to change its state forms the action of the MDP a_t.

The dynamics of the environment are represented by the transition functions. They can be deterministic or stochastic in nature. In deterministic models, the next predicted state will always be the same for a given state and action. For stochastic models, the next state is

defined by a probability distribution Pr over all possible future states after executing action a_t in state s_t.

All states of an MDP follow the Markovian property, it means the next state of the system does not depend on the previous action or the history of previous states, but only depends on the current state and action. Mathematically, the environment's dynamics can be specified by a probability distribution function :

$$p(s_{t+1}|s_t, a_t) = Pr(s_{t+1}|s_t, a_t) = Pr(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, ...) \tag{2-2}$$

The environment gives feedback in terms of a special signal called **reward** r, when the agent carries out a transition from the current state s_t to the next state s_{t+1}. The agent's goal is to maximize the cumulative reward it receives in the long run. The objective of the task is defined by the reward function.

A **policy** π, the decision making part of the agent, is a function that maps from states to actions. Policies can be stochastic or deterministic. For a stochastic policy, the output would be a probability distribution over actions. For a deterministic policy, the policy directly gives the action to be taken.

For an agent to maximise its cumulative reward, the agent must have an estimate of how much reward is expected in future states under the current policy. This is obtained from a **value function** $V(s)$, which represents how good a state is for an agent to be in. It is equal to the expected total reward for an agent starting from state s_t. The value function depends on the policy by which the agent picks actions to perform. Knowing the value of each state, the agent can figure out what is the best action to be carried out.

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s,r} p\left(s_{t+1}, r \mid s, a\right)\left[r + \gamma v_\pi\left(s_{t+1}\right)\right], \quad \text{for all } s \in \mathcal{S} \tag{2-3}$$

where γ is discount factor that represents the difference in importance between present and future reward. More information on MDPs can be got from Sutton and Barto [2018]

2-2 Types of models

We can classify models based on what information the model predicts as forward dynamics models, inverse dynamics models and multi-step models (Fig. 5-1)

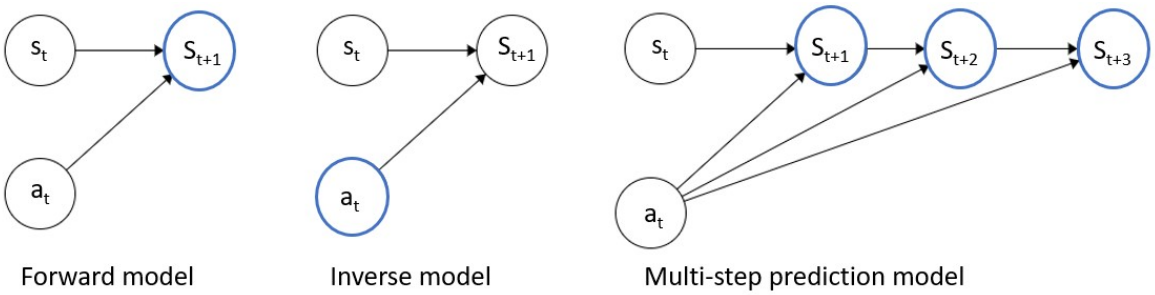


Figure 2-1: Types of dynamics models

Forward models predict the next state s_{t+1} of a dynamic system given the current action a_t and current state s_t. The forward model describes the mapping $(s_t, a_t) \rightarrow s_{k+1}$. If the mapping of the state action pairs are unique, then learning can be done using standard regression techniques (such as gaussian processes regression or neural networks). For stochastic systems, the forward models describe the conditional probability of the next possible state given the current state and action.

Forward models can be used for finding a solution for optimal control problems. The most commonly used method is model predictive control (MPC) [Kerrigan and Maciejowski, 2002]. These controllers find the optimal action to be executed in each state to minimizing a cost function, by predicting the future states using the forward dynamics model over a prediction horizon N. Further application of forward models is in model-based reinforcement learning which again relates to the problem of learning an optimal policy. Abbeel et al. [2007] was able to successfully stabilize a helicopter in an inverted flight using the learnt forward dynamic model.

Inverse models are used to predict the required action a_t to move the systems from the current state s_t to a desired future state s_{t+1}. If we know the current state and the desired or expected future state we can use the inverse model to infer the necessary action, i.e., the relation $(s_t, s_{t+1}) \rightarrow a_t$. Craig [2009] used an inverse dynamics model to predict the torques required to move the robot along the desired joint-space trajectory. If the mapping $(s_t, s_{t+1} \rightarrow a_t)$ is unique, a model can be learnt with standard regression techniques, if the mapping is not unique, (say for robotic arms with kinematic redundancy), it can be solved by introducing additional constraints such as minimizing energy to move from current state to the next state. Additionally, a combination of forward and inverse models can be used to resolve the non-uniqueness of the inverse model. These models are called mixed models and are not widely used in traditional control applications, but they've shown promising results in biologically inspired robot, due to the presence of kinematic redundancies [Salaün et al., 2010], [Ting et al., 2009].

Multi-step models Both forward and inverse dynamic models discussed above made 1-step predictions of the next state. We can make multi-step predictions by repeatedly feeding the prediction back into the learned forward dynamics model. Using feeding predicted states to a model can result in accumulation of errors. These accumulating errors may cause the model to diverge from the true dynamics. To overcome the error accumulation, there are two main approaches i) different loss functions and ii) separate dynamics functions for 1,2..n-step predictions. In the first approach, multi-step prediction losses are added to the loss function used for training the model [Chiappa et al., 2017; Hafner et al., 2019; Ke et al., 2019]. These models make 1-step predictions, but during training they are unrolled for n steps and trained on a loss with the ground truth n-step observation. The second solution is to learn a specific dynamics model for every n-step prediction [Asadi et al., 2018].

2-3 Different function approximators

For a discrete system or a discretized version of a continuous system, a discrete MDP can be used to represent the transition function. Each transition can be input as a separate entry in a table. Tabular models were popular in initial model-based and model-free RL algorithms

[Sutton, 1991]. However, they do not scale to high-dimensional problems, as the size of the required table scales exponentially. For continuous or high dimensional state spaces, function approximation methods are used to represent transition functions.

If the model dynamics can be derived by following the first principle methods, then the transition function can be obtained by integrating the differential equations stemming from Newton Euler methods, which lead to the most realistic models [Ross and Bagnell, 2012; Colomé et al., 2015]. One disadvantage of physics based models is that they require many constants like the coefficient of friction and spring constants to be known, to build an accurate model of the system. For complex systems, where modeling from first principle methods is not possible, transition function approximations are learnt from data. One method to learn a function approximation is to perform regression analysis on the data, and some examples are elaborated below

Locally Weighted Linear Regression (LWLR) is a non-parametric model that fits regression models on the training data. The model's coefficients are estimated using the ordinary least square estimator, modified by a kernel that provides a measurement of the similarity between new input and learned data. LWLR can be used for learning the models directly from sensor data without any knowledge about the dynamics of the robot. [Kim et al., 2004].

Dynamic Bayesian networks (DBN) extend standard Bayesian networks with the concept of time, which allows us to model time series data like dynamic models. Learning the local structure of the DBN corresponds to learning of MDPs' transition probabilities by exploiting conditional independence relations that exist between state features at time t and t+1. A variant of logistic regression is used for training of the DBN on the collected data [Hester and Stone, 2011].

Decision Trees can also be used to model deterministic models. They use a divide and conquer approach in the feature space to predict the output or the next state of the system. They are used because of their ability to generalize the learned model well [Hester et al., 2010], [Hester et al., 2012].

Models can also be built using **random forest**, which is a collection of decision trees [Hester and Stone, 2010]. Each decision tree is trained on only a subset of the agent's experiences. The final prediction of the next state/action is the average of the predictions of each of the trees. The use of multiple sub-models could be used for estimating the uncertainty of the model

Rather than providing a single 'best-fit' to the observed data, **Gaussian Processes (GP)** take a Bayesian approach and provide a complete posterior distribution over states and actions to the next possible states. The dynamics can be modelled as $GP(m(x), k(x, x'))$, where $x = [s_t, a_t]$, $m(x)$ is the mean function and $k(x, x')$ the covariance function of the Gaussian process (GP). The mean of the GP is assumed to be zero, i.e., no prior knowledge, and the covariance function $k(x, x')$ is usually a general kernel such as a squared exponential.

Gaussian mixture models (GMM) can also be used to fit unknown model dynamics, and such models are used in robots [Gribovskaya et al., 2011], [Calinon and Billard, 2009]. The model is trained on a tuple of the current state-action pair and the next combination (s_t, a_t, s_{t+1}), to obtain a gaussian distribution $p(s_t, a_t, s_{t+1})$.

In recent years, there has been much interest in using **(deep) neural networks (DNN)** approximation for dynamics function approximation [Christiano et al., 2016; Polydoros and Nalpantidis, 2016; Nagabandi et al., 2020]. DNNs have millions of parameters allowing them to model complex functions such as nonlinear dynamics. They learn compact representations of state from high-dimensional, multimodal sensor data commonly found in robotic systems [Böhmer et al., 2015], and unlike many machine learning methods, they do not require hand-engineer feature vectors from sensor data. This nonlinear regression of NN provides the functionality that is needed for operating dynamical systems in continuous spaces [Lewis et al., 2020; Müller et al., 1995].

DNNs are well suited for use with robots because they are flexible and can be used in structures that other machine learning models cannot support. The simplest of the structures is a feed-forward neural network which acts as function approximators/ transition function. The input to the neural network is the current state and the current action, and output is the next state. These models are trained to approximate the mappings represented in a training set of pair-wise examples in a supervised manner. An optimization method is applied to minimize the prediction loss. For such problems, loss is typically measured with sum-squared error, $\sum_i^n (y_i - \hat{y}_i)^2$. The most popular optimization method for neural networks is stochastic gradient descent, but improved methods such as RMSProp and Adam have recently gained popularity.

Another neural network structure that is used for modeling dynamics is Recurrent neural networks(RNNs) specialize in dynamics and temporal predictions. It is trained with an approach called 'backpropagation through time' [Werbos, 1990],44]. Many advances, such as 'long short-term memory units,' have made recurrent neural networks much stronger [Hochreiter].

RNN excel at learning to anticipate complex dynamics as the recurrent connections give the model a form of "memory" that is used for learning a relation between the current and the previous states. This knowledge of state enables them to model the effects of time in a changing environment. Several researchers have used recurrent networks to learn model dynamics directly from full observations. Lenz et al. [2015] modeled robotic food cutting with a knife. This includes difficult-to-model effects such as friction, deformation, and hysteresis. Food-knife surface contact changes through the cut, and so do the material properties of the food. Data obtained while operating under fixed-trajectory stiffness control was used to train the RNN on the system dynamics, and the resulting model was used to implement a model predictive control algorithm.

Artificial neural network can handle non-linearities and uncertainty on its inputs. Making them ideal for modelling novel dynamics like those needed to solve problems such as grasping new objects, traveling over surfaces with unknown or uncertain properties, managing interactions between a new tool and/or environment, or adapting to degradation and/or failure of robot subsystems. The architecture of neural networks allow for learning of models with high degrees of freedom. Kumar et al. [2019] carried out a comparative study of different neural network architectures (MLPs, FNN, RNN, LSTMs) for learning forward dynamics.

2-4 Data collection methods

Since model learning is essentially a supervised learning problem [Jordan and Rumelhart, 1992], we need ground truth data for learning a model. In order to learn an accurate model for applications in robotics, the sampled data must be representative of a large region of the state space. The data collected should be processed to ensure there are no redundant and irrelevant data, as spurious data can cause bias and affects the accuracy of the model. Data is collected in the following methodologies as cited in Hong et al. [2020]:

Random exploration: The samples required for training the model are got by the agent executing random action. It is the most commonly used method for learning inverse dynamics models due to its simplicity in implementation [Pathak et al., 2018; Nair et al., 2017; Agrawal et al., 2016]. In model learning RL, for an agent to execute its optimal policy, its state space and action space must be explored. The agent follows an epsilon greedy policy, the agent takes action using the greedy policy (policy that maximises the immediate reward) with a probability of $(1-\epsilon)$ and a random action with a probability of ϵ. This approach ensures all the action space is explored. The random transitions observed are recorded, and appended to the training data. Given the additional training data, the dynamics model is re-trained, which improves the dynamics model, as well as, the policy being executed. [Deisenroth and Rasmussen, 2011; Gal et al., 2016]

Demonstration: The dynamics model is learnt directly with expert demonstrations. The sequence of state-action pairs, recorded over time t during task execution by the expert are called Trajectories, $\tau = (s_0, a_0), ...(s_t, a_t)$. Since expert demonstrations comprise successful trajectories, the recorded state action pairs are sufficient to learn a dynamics model to be able to replicate the demonstrated task [Bain and Sammut, 1995; Abbeel and Ng, 2004; Nair et al., 2017]. However, learning only from successful demonstration results in the co-variate shift problem, as the model may not have learned different transition which it has not encountered during the demonstration.

Curiosity: Curiosity can be used to explore the environment and discover goal states, but also as a way of learning new states which might come handy for pursuing actions in the future. This method incentivizes an agent to collect samples that lead to large errors of its forward dynamics model [Pathak et al., 2017, 2018; Hong et al., 2020] .

2-5 Region of validity

Dynamics models can be classified as global models or local models based on the region of state space where they are valid.

Global: These models approximate the dynamics over the entire state space. This is the main approach of most model learning methods. It can be challenging to generalize well over the entire state space, but it is the main way to store all information from previous observations. In order to learn an accurate model for a large state space more data is required which may be expensive or unsafe.

Local: The other approach is to only locally approximate the dynamics and each time discard the local model after planning over it. This approach is especially popular in the

control community, where they frequently fit local linear approximations of the dynamics around some current state. A local model restricts the input domain in which the model should be valid, and is also fitted to a restricted set of data. Because fitting data on smaller state spaces is more computational efficient, such techniques have become widespread in model learning for robotics [Atkeson et al., 1997; Bagnell and Schneider, 2001; Levine and Abbeel, 2014; Tevatia and Schaal, 2008]. A benefit of local models is that we may use a more restricted function approximation class (like linear), and potentially have less instability compared to global approximation [Moerland et al., 2020]. On the downside, we continuously have to estimate new models and do not continue to learn from all collected data (since it is infeasible to store all previous data points).

2-6 State abstraction

Since model learning is a supervised learning algorithm, it suffers from the curse of dimensionality. These high-dimensional observations present a challenge for perception, learning and planning. While deep reinforcement learning algorithms have shown that it is possible to learn controllers directly from large dimensional observations (like raw images) [Mnih et al., 2015], however, reinforcement learning (or other control algorithms) can take advantage of low dimensional and informative representations, instead of raw data, to solve tasks more efficiently [Munk et al., 2016]. Especially, in the field of robotics, which relys heavily on sensors for estimating the states of the system for control tasks, finding and defining interesting states (or features) for control tasks usually requires a considerable amount of manual engineering. It is therefore of interest to learn these features with as little supervision as possible. State Representation learning algorithms are a form of feature learning, where the pertinent information from high-dimensional observations is extracted using machine learning instead of human intuition.

The objective of state representation learning (SRL) algorithms is to learn a low dimensional representation of a high dimensional state space while encoding information to carry out a given task while discarding the many irrelevant aspects of the original data. The low dimension representation must capture the variations in high dimensional states caused by the agent's actions. These low level or latent spaces can help improve performance and speed in policy learning algorithms such as reinforcement learning. For a robot, sensors like cameras act as high dimensional input, while the task could be expressed in a low dimensional state space like the cartesian coordinates of the end effector.

The main advantage of carrying out state representation learning is the ability to carry out computation and planning at latent spaces. Separation of representation learning and policy learning is a way to lighten the complete process. As described in a few of the reviewed papers [Van Hoof et al., 2016; Munk et al., 2016; Lesort et al., 2018], this approach is used to make policy learning faster and lighter in computation.

SRL can also be used in a transfer learning setting by taking advantage of a state space learned on a given task to rapidly learn a related task. This is for example the case in [Jonschkowski and Brock, 2015] where a state space related to a robot position is learned in a given navigation task.

Early methods of learning a lower dimensional state space involved using dimensionality

reduction algorithms like principal component analysis (PCA). PCA is a linear transformation able to compress and decompress observations with minimal reconstruction error. Using PCA, Karakovskiy and Togelius [2012] was able to reduce images to 4-dimensional state spaces, which reduced the time to converge to optimal policy in simulations such as mountain car. Due to non linearities when compressing high dimensional data to a lower dimensional latent state space, deep neural networks work well for state representation learning [Oh et al., 2015; Watter et al., 2015; Chiappa et al., 2017]. Commonly used neural network architectures for SRL are auto-encoders. They learn to reproduce the input under constraints on their internal representations such as dimensionality constraints in a middle layer. The architecture of an autoencoder can be split into i) an encoding function $z_t = f_{enc}(s_t)$, which maps the observation to a latent representation z_t, ii) a decoder function $s_{t+1} = f_{dec}(z_{t+1})$, which maps the latent state back to the next state prediction. The latent state space z has a lower dimensionality compared to the observed state s.

Models based on auto-encoders can not only reconstruct the high dimensional state spaces, they can also be used to learn the system dynamics as proposed in Goroshin et al. [2015]. The auto-encoder model can learn the forward dynamics by firstly making an encoding from the original space to a latent state space to obtain z_t and then reconstruct the next state $\hat{s}_{t+1}$. The transition function is trained by minimising the error computed by comparing the estimated next state $\hat{s}_{t+1}$ with the value from the next observation s_{t+1} at the next time step. However, we must ensure that the predicted next latent state lives in the same state space as the encoded current latent state, else the model will deviate from the truth. This is done by ensuring that the learnt latent dynamics function follows linear dynamics [Watter et al., 2015] Karl et al. [2016].

Similarly, an inverse dynamics model can also be learnt using the autoencoder framework. The model encodes the states s_t and s_{t+1} to latent states z_t and z_{t+1}, then the model predicting the action $\hat{a}_t$ that can produce the transition from z_t to z_{t+1}. An example using inverse models to learn state representations is the Intrinsic Curiosity Module(ICM) [Pathak et al., 2017]. Learning an inverse model is more sample efficient compared to learning a forward model, since actions space is a much lower dimension compared to state spaceLesort et al. [2018].

Another method for state representation learning is with the Generative Adversarial Network (GAN) framework. Chen et al. [2016] proposed InfoGAN that achieves the disentanglement of latent variables on 3D poses of objects. State representation learning in continuous state-action spaces using very high-dimensional observations remains a key challenge in developing fully autonomous systems [Lesort et al., 2018].

2-7 Conclusion

Traditionally roboticist relied on manually generated physics based models for control and planning, but future autonomous robots need to be able to automatically learn models that are based on information that is extracted from the data collected by the robot. Model learning can be a useful alternative to manual pre-programming, as the model is estimated directly from measured data. With model learning, unknown non linearities can be directly taken into account, while they are neglected by the standard physics-based modeling techniques and by hand-crafted models. Model learning has been shown to be an efficient tool in a variety of scenarios, such as robot manipulation, autonomous navigation or robot locomotion.

The two main problems faced for model learning for robotics are : high dimensional data and small data sets. Firstly model learning algorithms have to deal with massive amounts of data, such as in learning dynamics from image data. The algorithms need to be efficient in terms of computation without sacrificing the learning accuracy. Secondly, as the data generation may be too tedious and expensive, we need algorithms that allow us to learn and improve the model in the presence of sparse data, by incorporating prior knowledge or using active learning. Standard model learning approaches, such as Gaussian process regression, scale cubically in the number of training data, preventing a straightforward usage in robotics. In recent years, there have been serious efforts to speed up model learning algorithms by using deep neural networks. These DNN are able to work with sparse and large data sets to learn a function approximation of the system's transition function.

In the field of imitation learning explained in the following section, a policy is learnt from trajectories of optimal states and actions. However, in real-life demonstrations, the action information may be missing in some cases and only state trajectories are available. The demonstration provided could have a viewpoint mismatch or embodiment mismatch (the kinematic model of the demonstrator may be different from that of the robot). To solve these problems it is necessary to learn the model of the system.

Chapter 3

Imitation Learning

A more intuitive way for an agent to learn control is to learn from demonstrations of the desired behavior. Imitation Learning (IL) is a machine learning technique where a robot (or an agent) learns a policy to execute a task by observing demonstrations performed by a human or an expert [Schaal, 1997; Argall et al., 2009]. Imitation learning was developed in the field of robotics to address difficulties programming autonomous robots using standard methods. Primarily due to the inability of humans to pre-program a robot for every possible scenario. Secondarily to allow people who do not have the specific set of skills and knowledge to program robots to modify a robot's behavior.

By observing demonstrations executed by an expert, the agent can reduce the search for a possible solution, by either starting the search from the observed good solution, or by pruning parts of state spaces which produce a bad solution [Billard and Grollman, 2012]. It is considered to be a key technology for applications such as autonomous vehicles [Sammut et al., 1992; Pan et al., 2017], manufacturing [Jha et al., 2017], elder care [Bemelmans et al., 2012] and the service industry [Saunders et al., 2006; Nicolescu and Mataric, 2003].

Section 3-1 provides preliminary information about IL is provided, followed by details about traditional IL techniques and highlights two of the prominent IL methods, namely, Behavioral Cloning (BC) and Inverse Reinforcement Learning (IRL).

Section 3-3 provides detailed methods that use inputs from a demonstrator during learning phase to improve the agent's performance i.e. Interactive Imitation Learning.

Section 3-4 provides details on Imitation from Observation, IL techniques where policy is learnt from state only trajectories.

Section 3-5 delves into Imitation learning in latent space and the types of methods used.

Section 3-6 summarizes the chapter and offers a hypothesis on an Interactive Imitation Learning (IL) method that learns a policy in state space invariant on the environment dynamics.

3-1 Preliminaries

The behavior of the expert demonstrator (or the learner itself) can be observed as a trajectory $\tau = [(s_0, a_0), ..., (s_n, a_n)]$, which is a sequence of state/observation, action pairs. In imitation learning, a dataset of demonstrations $\mathcal{D} = (\tau_i, r_i)_{i=1}^{N}$ that consists of pairs of trajectories τ and optionally reward signals r are collected. Using the collected trajectories, the imitating learning problem can be reframed as an optimization problem

$$\pi^* = arg\ min\ D(q(\phi), p(\phi)) \tag{3-1}$$

where the agent learns a policy π^* such that it minimises the difference between executed sequence of states and the demonstrated trajectory [Osa et al., 2018], where ϕ represents state (or observation) - action pairs, $q(\phi)$ is the distribution of the features induced by the experts' policy, $p(\phi)$ is the distribution of the features induced by the learner, and $D(q, p)$ is a similarity measure between q and p.[Osa et al., 2018]

Imitation learning techniques can be characterized based on the modes of interaction, which enable a robot to learn (1)through doing (Learning from Demonstration (LfD)), (2) through observation (Learning from Observation (LfO)), and from critique.

Learning from Demonstration (LfD), popularised by Atkeson and Schaal [1997], explores techniques for learning a task policy from examples provided by a human teacher, who demonstrate how to perform the desired task. These demonstrations are provided as a sequence of the state and the control input. Teleoperation provides the most direct method for accessing state and action from demonstrations [Chernova and Veloso, 2009]. During teleoperation, the robot is operated by the teacher while recording from its sensors. Demonstrations recorded through human teleoperation have been used in a variety of applications, including flying a robotic helicopter [Abbeel et al., 2007], robotic arm assembly tasks [Chen and Zelinsky, 2003] and obstacle avoidance and navigation Min et al. [2005].

In some situations, the teacher performs the task demonstration using their own body or with another robot embodiment instead of controlling the robot directly, as a result, the agent has access to the robot's state observed by its sensors. Here, the trajectory is given as a sequence of the state of the system $\tau = [s_0, ..., s_n]$. These IL algorithms are called **Learning from Observations (LfO)** [Liu et al., 2017]. IfO algorithms are discussed in detailed in section 3-4.

3-1-1 Model-Free and Model-Based Imitation Learning

Imitation learning algorithms can be classified into model-free and model-based algorithms. Model-free imitation learning methods learn a policy that replicates the behavior demonstrated by experts without learning a model of the system. Behavioral cloning (BC) is one such IL algorithm that learns a direct mapping from states to actions. Therefore, there is no need to estimate the system dynamics in model-free imitation learning method. The model dynamics is encoded into the policies learned by model-free methods. Such algorithms can be easily applied to motion planning for fully actuated robotic systems when expert demonstrations are available.

Model-based imitation learning methods learn a policy that reproduces the demonstrated behavior by using a transition model, like a forward dynamics model of the system. If the transition model is given, then a policy can be derived from simple planning algorithms. When no transition model is given, then the model can be learned by sampling the environment, and be used with planning to update the policy.

For high dimensional problems, the complexity of learning the transition model, followed by a planning algorithm is smaller than the complexity of learning the policy model directly and is more sample efficient. In model-free learning, a sample is used once to optimize the policy, and then discarded, in model-based learning the sample is used to learn a transition model, which can then be used many times in planning to optimize the policy. The sample is used more efficiently.

Inverse reinforcement learning (IRL) is one such IL algorithm that attempts to recover a reward function from the demonstration. Using the dynamic model makes inverse reinforcement learning easier since the learner's performance can be predicted when the system dynamics is known. IRL has focused on learning a policy that needs to be iteratively evaluated in a given system. A model-based approach is suitable for such applications, hence many model-based methods have been developed for IRL.

3-2 Sequential Imitation Learning

Traditional methods of imitation learning can be classified as sequential imitation learning. These methods learn to imitate the task in two phases: 1) demonstration recording, 2) policy derivation. Depending on the learning strategy and information available, two of the most prominent sequential IL approaches are, Behavioral Cloning (BC) and Inverse Reinforcement Learning (IRL).

3-2-1 Behavioral Cloning

One way to learn a policy that reproduces the demonstrated behavior is to capture the expert's cognitive skill into a program. The data set of demonstrated trajectories with state-action pairs $\mathrm{D} = (s_t, a_t)$ is used to build a regression model with the current state as input and action to be executed to reproduce the expert's trajectory as an output. Typical standard supervised learning techniques used to learn a policy directly map from the state or the observation to the control input. This method is often referred to as **Behavioral Cloning (BC)** [Bain and Sammut, 1995; Ross et al., 2011].

Algorithm 1 Behavioral cloning

Require: a set of trajectories demonstrated by the expert $\mathcal{D}$
1: Initialize π_θ with objective loss function $\mathcal{L}$
2: Optimize $\mathcal{L}$ w.r.t. the policy parameter θ using $\mathcal{D}$
3: return optimized policy parameters θ

Algorithm 1 gives a description of the procedure of BC methods. The first step of BC is to record a set of expert demonstrations $\mathcal{D}$ which are usually given as a set of trajectories.

Thereafter, we need to select a policy representation π_θ appropriate for a given application. In addition, we need to select an objective function $\mathcal{L}$ that represents the similarity between the demonstrated behaviors and the learner's policy. Commonly used loss functions are l_2 or quadratic loss functions. The parameters θ are then optimized using the collected data set of demonstrations.

When using supervised learning, an appropriate regression method must be chosen, based on the complexity of the model. Simple models can be trained using linear regression techniques [Atkeson et al., 1997; Atkeson and Schaal, 1997]. Complex models use different architectures of neural networks to handle highly nonlinear mappings [Wen et al., 2015; Baram et al., 2017; Nair et al., 2017]. Various function approximators described in section 5-1 are used for behavior cloning. However, training such complex models require a large amount of training data [Bojarski et al., 2016].

Model-free BC methods do not learn system dynamics, they learn a mapping directly from the state to the action. Direct learning means that the learning algorithm does not iterate between trajectory and behavior generation. However, model-free methods are hard to apply to underactuated systems, since without a model predicting desired behavior is hard. Model learning BC methods on the other hand do not suffer from this issue, since they utilise the transition model of the system [Torabi et al., 2018a; Van Den Berg et al., 2010]. But learning the transition model of high dimensional problems requires high capacity networks that require many samples for training to achieve high generalization while preventing overfitting, potentially undoing the sample efficiency gains of model-based methods.

Model learning/based BC methods are used in systems where there exists an embodiment mismatch between that of the demonstrator and the learner. This mismatch is known as the **correspondence problem** in imitation learning [Billard et al., 2008]. The demonstrated trajectory needs to be modified to adapt the demonstrated trajectories to follow the constraints and dynamics of the learner.

The supervised learning algorithms used in BC can suffer a domain shift between the training experience and the online behavior. This domain shift is called **covariate shift** [Osa et al., 2018]. If the system encounters a new state that it has never seen during training, the model would not know the appropriate action to execute. From the regression model, the agent approximates the closest state. Sequential estimation of actions can lead to compounding errors leading to derivation from the desired trajectory. This would be dangerous for safety-critical applications like autonomous driving [Codevilla et al., 2019]. One solution to fix the covariate shift is to carryout on-policy data collection. In DAGGER [Ross et al., 2011], data is collected from the current robot policy and the policy is updated on the aggregate data set collected. Bojarski et al. [2016] and Laskey et al. [2017b] learn a stabilizing policy where the policy mitigates the trajectory drift issue by learning a corrective action simultaneously.

3-2-2 Inverse Reinforcement Learning

Inverse reinforcement learning (IRL) [Ng et al., 2000], is an imitation learning algorithm where the learner tries to recover/learn a reward function from a policy (or demonstrations of a policy). From the obtained reward signal, a policy can be obtained so as to maximize

the expected return following reinforcement learning. Such a policy can be expressed as

$$\pi = arg\ max\ J(\hat{\pi}) \tag{3-2}$$

where J($\hat{\pi}$) is the expectation of the accumulated reward given the policy. The reward function needs to be shaped from expert demonstrations. Alternatively, this can also be thought of as learning a cost function that produces the same trajectory as the demonstrator.

Algorithm 2 Inverse Reinforcement Learning

Require: a set of trajectories demonstrated by the expert $\mathcal{D}$
1: Initilize $V(s) = 0$, for all $s \in \mathcal{S}^+$
2: **repeat**
3: learn reward $r_\theta(s_t, a_t)$
4: Using the learnt reward function r_θ,learn policy using RL;
5: Compare π with π^*(experts' policy);
6: **until** π is satisfactory

Algorithm 2 gives a description of the procedure of IRL methods. IRL involves two steps: (i) We start with a set of expert's demonstrations (we assume these are optimal) and then we try to estimate the parameterized reward function, that would cause the expert's behaviour/policy. (ii) Using the estimated reward function, we try to find the optimal policy using standard reinforcement learning methods. Finally, policy improvement can be carried out by repeating the previous steps until the learnt policy π is satisfactory.

Similar to behavioral cloning methods, IRL methods can be categorized into two categories: model-based and model-free methods. Model-based IRL methods use the knowledge of the system dynamics to evaluate the model-based RL on the learnt reward function and policy. These methods use the model to estimate the agent policy's trajectory distribution [Abbeel and Ng, 2004; Levine et al., 2011]. Model-free IRL methods do not require prior knowledge of the system dynamics. Model-free IRL methods employ sampling techniques to evaluate and update the learned reward function and policy. These algorithms avoid explicit learning of system dynamics, but need many trajectories to estimate the trajectory distribution, which can be time-consuming and computationally expensive [Ho and Ermon, 2016].

IRL methods work on the assumption that learning a reward function is statistically faster than learning a policy π directly [Arora and Doshi, 2018]. However, reward estimation/RL needs to be done repeatedly for policy optimization for every reward function derived, which can be costly from a time and safety perspective. [Torabi et al., 2019b]. Another limitation of IRL is finding a unique reward function for a given demonstration. For most observations of behavior there are many fitting reward functions. Additional objective functions which employ the principle of maximum entropy can be used to resolve the ambiguity in choosing a distribution over decisions which result in obtaining a unique solution in IRL [Ziebart et al., 2008]. Finally, IRL algorithms assume that the observed behavior is optimal. Demonstrations provided by humans are sub-optimal, therefore the learnt policy will always be sub-optimal, despite carrying out RL repeatedly for policy improvement.

3-3 Interactive imitation learning

A common drawback of agents trained using IL is that they suffer from compounding error problem [Bagnell, 2015]. Due to small errors accumulating, the agents will inevitably reach unknown states that were not explored in the demonstration [Osa et al., 2018]. Most supervised learning IL methods require a lot of training data to learn the transition model and the imitation policy, but the demonstrator may not be able to cover all possible situations, or demonstrations may be highly expensive, therefore there is a need for alternative methods that provide the necessary information to the agent efficiently. One way to do this is to use the provided feedback during training of the algorithm to guide the agent towards desired behaviour. This can be considered Interactive Machine Learning (IML) [Chernova and Thomaz, 2014; Amershi et al., 2014]. Using interactive machine learning, the model gets updated immediately in an incremental manner in response to teacher's feedback. This allows the teacher to interactively examine the impact of their actions and adapt subsequent inputs to obtain the desired behavior. As a result of these alternating policy execution and policy updation cycles, even teachers with little or no machine-learning expertise can adjust the behavior of agent through low data trials and experiments. Based on the nature of feedback provided by an expert, interactive learning can be classified into the following - (i) evaluative feedback, (ii) corrective feedback [Mourad et al., 2020; Zhang et al., 2019b].

In evaluative feedback, the human provides evaluative critiques to indicate the merit of the performed action. This kind of feedback is used when the teacher is not a expert at performing the task or is too difficult for humans to provide corrective action, but can assess the agent's state or action. Corrective action feedback is used where the human teacher cannot provide an optimal action, which true in most realistic cases.

3-3-1 Evaluative feedback

The simplest form of evaluative feedback is a scalar value given by a human in real time indicating how desirable an observed state or action is while watching the agent performing the task. This approach greatly reduces the needed human effort. One of the main challenges in this approach is use human feedback effectively since such interpretation determines how the feedback is used to improve the policy in the MDP framework. The evaluative feedback provided can be either over entire trajectory executed by learners [Akrour et al., 2014; Jain et al., 2013] or can be provided for each action executed in order to nudge the agent in the right direction [Knox and Stone, 2009].

One such method is TAMER (Training an agent manually via evaluative reinforcement) [Knox and Stone, 2008], where the human feedback is learnt as the value function or a human-specified reward function (H). In any given state, the agent's goal is to choose the action that will receive the most reward from the human. To do this, the agent chooses the actions that will maximise the expected reward. After learning an accurate model of the human's reward (H), the agent can continue to perform the task in the absence of the human, choosing actions that maximize the received reward as if the human were providing them. Once H is trained, the human need not provide input continuously, feedback is provided only when the agent reaches an unknown state. The TAMER framework was recently extended to deep TAMER [Warnell et al., 2018]. Deep TAMER uses a neural network to represent and learn an estimate of H, via supervised learning.

Another form of evaluative feedback is preference. Many RL tasks may be too complex for humans to provide demonstrations or these could be control tasks with many degrees of freedom, like MuJoCo Ant [Todorov et al., 2012]. It is, therefore, harder for humans to control or to tell whether a particular state-action pair is good or not. However, it may be possible to set a ranking between multiple behaviors. This ranking or preference is used as feedback to modify the policy. Busa-Fekete et al. [2013] used preference to learn policies directly, Akrour et al. [2014] learnt a reward function using preference. The preference framework can be used to evaluate states, actions or a segment of trajectory. Christiano et al. [2017] used a pair of agent trajectories of 1-2 seconds long are simultaneously presented to human trainers to query for their preference. Accurate selection or ranking of preferences ensures policy improvement. But if all the presented trajectories are sub optimal, the resultant policy also will be sub optimal. Because of this, the algorithm will query the user a large number of times before it converges to an optimal policy.

3-3-2 Corrective feedback

Corrective feedback is given to steer the agent to the preferred state or action. The assumption is made such that the expert knows what is the correct input to be given to the system. One of the commonly used methods for corrective feedback is DAGGER (Data Aggregation Approach) [Ross et al., 2011]. In DAGGER, an expert simultaneously provides corrective actions while the agent is executing in order to correct the agent's behavior. These action labels are appended to the current state of the agent. It solves the covariate shift problem faced by BC algorithm by collecting a dataset at each iteration under the current policy and trains the next policy under the aggregate of all collected datasets. This is considered to be an on-policy approach to imitation learning [Xiong et al., 2019].

In the first iteration, the policy replicates the actions of the demonstrations, which results in policy π_0^L (where L represents the policy of the learning agent). The data set of states visited and actions executed are recorded when following the previous policy. The policy is now trained on an aggregated dataset containing the new data (newly visited states and actions) and the previously collected data. The improved policy π_i^L is now used for mimicking the demonstrated trajectory (i represents the iteration). The general policy used by DAGGER algorithm can be written as $\pi_i^L = \beta_i \pi^E + (1 - \beta_i)\hat{\pi}_i^L$, a stochastic mixing of expert policy π^E and learnt policy π_i^L is used to collect the next data set, where $\beta \epsilon [0 - 1]$ is a gating function , is used to execute the expert's action with probability β and the agents action with probability $(1 - \beta)$.

The executed policy is always a mixture of the learnt policy and the expert's policy, the expert does not know if the input correction is sufficient (in case of conflicting control efforts from the agent and the expert). This can cause the agent to learn behaviors that are significantly different from the expert behavior [Kim et al., 1992]. To overcome these limitations, in HG-DAGGER [Kelly et al., 2019], the current policy is executed out until the expert observes that the agent has entered an unsafe region of the state space. The expert takes full control of the agent's actions and guides the system back to a safe and stable region of the state space. The recovery actions provided by the expert are collected and added to $\mathcal{D}$, during which the human expert has uninterrupted control of the system. This way the expert can observe the execution of the corrective action given to the agent immediately.

HG-DAgger outperformed policies trained with DAgger and behavioral cloning in terms of sample efficiency, training stability in the simulated driving task.

Also, DAgger queries the expert for every instance of time for the expert action which is expensive. To minimize the number of queries required, Zhang and Cho [2016] and Laskey et al. [2017a] the expert only gives corrective actions when there is a significant discrepancy between the executed trajectory of the agent and the demonstration.

Chernova and Veloso [2009] proposed a method that learns a policy by requesting additional demonstrations based on the confidence metric when the current policy is executed in a given state. The agent learns a model from a finite set of demonstrations by using classifiers that return selection confidence. When the confidence is lower than a threshold, additional expert demonstrations are requested.

For simple systems the expert may be able to estimate the magnitude of action but it may be difficult to estimate it for continuous systems. For such systems the corrective feedback provided can be used to modify or adjust the current behaviour of the agent. Argall [2009] defined advice-operators which are a finite predefined list of corrections, which are used for modifying low-level continuous valued motion control actions. These operators tell modification that has to be carried out on the current action. This modified action and state is used for retaining the model. The addition of state-action pair generated from the human feedback can address the LfD limitation of dataset sparsity for continuous action spaces.

Celemin and Ruiz-del Solar [2019] proposed COACH (COrrective Advice Communicated by Humans) which uses a sequence of binary signals as operators to modify the magnitude of the action to the learning agent. In COACH, the expert provides a sequence of feedback in terms of a binary signal $h = \{-1, 0, +1\}$, which represents an increase/decrease in the value of an action. The corrective feedback h, modifies the desired action as shown

$$a_t^{des} = a_t + h.e, \text{ where } e \text{ is step size} \tag{3-3}$$

The agent's policy is then trained with the current state and the desired action a_t^{des}. If a feedback of "+1" is received while executing the policy, that the executing policy must increase its magnitude of action to return to desired trajectory, a feedback of "-1" is received, then the executing policy must decrease the magnitude of action to return to the desired trajectory, 0 indicates no corrective feedback to be given to the system. The COACH algorithm is more intuitive since the algorithm executes the corrected action immediately and the user is able to see the effect of the correction. The method has been used to successfully learn tasks such as a cart pole, ball-dribbling (with a humanoid robot) and learning to balance on a bicycle. The results of their experiment show that an agent can learn much faster with lesser data using the corrective advice from a non expert operator's intuition (just based on the trend of the correction).

For higher dimensional systems where neural network is used to represent a policy, D-COACH algorithm [Pérez-Dattari et al., 2019] can be used. D-COACH combines the human feedback mechanism of COACH with a deep neural network using a replay buffer. On each execution of the policy, the corrected state-action pair are stored in the replay buffer. The neural network is trained again on the state action pairs stored in the buffer.

Feedback is usually used to provide correction in state or action. Most algorithms discussed above provide corrections by a human teacher in action spaces where the corrective

actions are discrete. But it is unintuitive for humans to know what the corrective action is for providing feedback to the agent (say corrective joint torques in a multi DOF robotic arm). Whereas it is more intuitive for humans to provide corrective feedback in state space (provide feedback in the workspace than in the configuration space i.e.joint torques for a multi DOF robotic). In TIPS (Teaching Imitative Policies in State space), Jauhri et al. [2020] uses human feedback in state-space during the execution of the current policy to teach a policy. The corrective feedback, $h = \{-1, 0, +1\}$, is used to indicate the desired state as shown

$$s_{t+1}^{des} = s_t + h.e, \text{ where } e \text{ is step size} \tag{3-4}$$

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

Initial Model-Learning Phase

Generate N_e experience samples $s_t, a_t{}_1^{N_e}$ by executing a random/exploration policy π_e
Append samples to experience buffer E
Learn forward dynamics model $f\theta$ using inputs $s_t, a_t{}_1^{N_e}$ and targets $s_{t+1}{}_1^{N_e}$

Teaching Phase:

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**
      Compute desired state $s_{t+1}^{des} = s_t + h_t.e$
      Compute action $a_{t+1}^{des} = \underset{a}{\operatorname{argmin}} \left\| f_\theta(s_t, a_t) - s_{t+1}^{des} \right\|$
      Append (s_t, a_t^{des}) to demonstrations 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}
    else
      ## No feedback
      Execute action $a_t = \pi_\phi(s_t)$, reach state s_{t+1}
    end if
    Append (s_t, a_t, s_{t+1}) to experience buffer E
    if $mod(t, T)$ **then**
      Update policy π_ϕ using batch sampled from demonstration buffer D
    end if
    Update learnt FDM f_θ using samples from experience buffer E
  end for
end for

In this approach, an inverse dynamics model (IDM) which is learnt previously, translates the corrections in the state space to corrections in the action space for the visited states that receive feedback from the user. The desired state could be in the partial state dimension or could be infeasible, therefore an indirect inverse dynamics model is used. From the current state using a forward dynamics model all possible next states are sampled and the action is chosen such that the next feasible state s_{t+1} is closest to the desired state s_{t+1}^{des}. The agent's policy is then trained with the current state s_t and the desired action a_t^{des} got from the indirect IDM.

TIPS outperforms other IL techniques like BC and GAIL in OpenAI environments like CartPole, Reacher and LunarLanderContinuous. This showed the advantage of TIPS as an interactive learning technique suitable for scenarios where demonstrators are non-experts. When comparing interactive learning techniques in state space (TIPS) and action space (D-COACH), for the Reacher task and cart-pole task, there is a significant reduction in demonstrator task load and frustration with state space demonstration. This also shows the state-space feedback mechanism can lead to a significant reduction in demonstrator effort.

A major limitation of TIPS is learning an accurate inverse dynamics model to compute actions. This can be challenging for environments with high dimensional state and can require a large number of interactions with the environment. Another limitation is action selection, TIPS identifies the correct action by querying the FDM will all possible actions before executing the in the agent's action-space. This would not scale well to high-dimensional or continuous action spaces.

3-4 Imitation Learning from Observation

Conventionally algorithms developed for solving IL problems require the demonstration information to include the expert's states (e.g.robot joint angles), and its actions (e.g., robot torque commands). However, these algorithms don't exploit existing resources such as videos of humans performing the task. Thus there is a need for methods that learn by utilizing demonstrations that have state only trajectories. Thus imitation learning algorithms, where an agent learns how to perform the tasks from state-only demonstrations given by an expert are called **Imitation learning from Observation (IfO)**. Compared to the typical IL paradigm described earlier, IfO is more similar to the way animals and humans learn a task by observing other executing the task (state/observation only), without getting any information about which muscles to move exactly (action).

Challenges faced in LfO algorithms are viewpoint difference and embodiment mismatch [Torabi et al., 2019b]. Since the IfO algorithms learn only from state only demonstrations, they are sensitive to the perception of the state from agent's and teacher's point of view. For robots sensors can be placed on joints to read the states [Ijspeert et al., 2002], for human teachers motion capture technology can be used to observe states [Field et al., 2009] and for sources like videos, computer vision with deep learning techniques can be used to extract features that are used in IfO algorithms. Embodiment mismatch arises when the kinematic model of the expert and the learning agent are not the same, then the agent cannot learn the policy directly from the expert. One solution is to encode the states invariant of the embodiment of the expert or the learner. Gupta et al. [2017] used an auto-encoder to learn correspondence between the embodiments in a supervised manner.

3-4-1 Model based and Model free IfO algorithms

IfO algorithms can be split into two general groups: (i) model-free algorithms,(ii) model-based algorithms based on whether the algorithm learns a model of the system using observations or not.

Model-free IfO algorithms learn an imitation policy by using inverse reinforcement learning techniques. A technique called reward shaping is used, where a reward function is manually designed, such that it rewards the agent if it executes actions that result in producing the expert state trajectories. One such reward function could be the Euclidean distance between the actual next state reached and the next state predicted by the sequence model. This reward function is used by Kimura et al. [2018] where a predictor is trained to predict the next state of the agent given the current state. Finally, the imitation policy is learnt via regular RL methods. Aytar et al. [2018] applied the same idea of closeness between the imitator's embedded states and demonstrator's states for video based on the demonstration. Gupta et al. [2017] showed that a similar reward function can be used even if the experts and the agents have an embodiment mismatch. The reward function is defined as the Euclidean distance of the expert and imitator state features in the invariant space at each time step (invariant to the embodiment of the demonstrator and the learning agent). Liu et al. [2018] showed that a similar reward function would work even if there exists a viewpoint mismatch. This is done by learning a translation model through which observations of a task from one point of view can be converted to the other. Like most model-free algorithms these methods need many demonstrations to learn the translation model.

Adversarial approaches to IfO are inspired by the generative adversarial imitation learning (GAIL) algorithm. Unlike GAIL which uses generative adversarial networks (GANs) Ho and Ermon [2016] to bring the distribution of state and action pairs of the agent and the demonstrator closer together, GAILfO (GAIL from Observations) [Torabi et al., 2018b], uses GANs to learn a distribution state transition (instead only states) of the expert and that of the imitating agent. The discriminator's output is used as a reward function to train the imitation policy using standard RL techniques. IRL-based algorithms need a large number of interactions with the environment during the learning phase to construct a reward signal, therefore these methods have primarily been applied in simulated robots and systems. Torabi et al. [2019a] modified GAILfO to be more sample efficient by using linear quadratic regulators (LQR) for the policy training step, in order to be able to execute it directly on physical robots. Stadie et al. [2017] also try to solve the viewpoint difference problem. In their method, the network is then trained in such a way that the demonstrations fed to the discriminator are invariant to viewpoint differences.

In Model-based IfO, a model of the system is either known or learnt during the learning process or by pre-training. The model can be written as a mapping state/observation transition (s_t, s_{t+1}) to action a_t, commonly known as inverse dynamics model. Since the desired state trajectory is provided by the demonstration, the appropriate action can be computed through the model. If the model of the system is available, then the action can be inferred easily. The policy can be learnt from simple IL methods like BC etc. If a model is not available, the model can be learnt using standard regression techniques (explained in chapter 1).

Hanna and Stone [2017] and Nair et al. [2017] learn an inverse dynamics model at a pixel-level using supervised learning. The agent collects ground truth data (o_t, o_{t+1}, a_t) using an exploratory policy which is then fed to a recurrent neural network to learn an inverse dynamics model. The agent is able to replicate the task of rope manipulation using only video demonstrations of the task being performed.

Behavioral Cloning from Observation (BCO) [Torabi et al., 2018a], learns a similar map-

ping but it learns a model of the environment rather than that of the agent. BCO takes this one step further by improving the inverse dynamics model learnt iteratively. A BC agent is trained on the states (got from demonstration), and action (inferred from the model). The agent then executes its current policy and the actual states visited are saved for policy improvement. The algorithm then carries out BC on the saved states and the actions (inferred from the IDM) to improve the imitation policy. The state transitions visited are used for IDM improvement. BCO shows promising results for simulated OpenAI Gym tasks such as 'Reacher' and 'Ant'. The method converges to a good model and policy even in high dimensional state spaces[Torabi et al., 2018a]. However, BCO suffers from the same problems as Behavioral Cloning (BC) such as distribution-mismatch between the states observed by the agent and the demonstrations. Guo et al. [2019] improved BCO by utilizing the reward for policy improvement.

In model based IL algorithms, the agent still has to interact with the environment a large number of times to learn an inverse dynamics model. To acquire samples(s_t, s_{t+1}, a_t), the agent may use a random or an exploratory policy or a predefined trajectory. For these models to be accurate and useful, the agent needs to cover a large distribution of the state space. Executing such random transitions to reach these states could damage the robot or be unsafe.

3-5 Imitation learning in latent space

Recently, a significant shift towards the use of latent dynamics models has emerged in model-based reinforcement learning [Hafner et al. [2019], Watter et al. [2015], Kaiser et al. [2019], Gelada et al. [2019], Zhang et al. [2019a]]. Learning the control policies in the latent dynamics space results in better generalization when compared to learning the policy in high dimensional state space [Hafner et al., 2019]. Recent research has learned latent dynamics [Watter et al., 2015], trajectories [Co-Reyes et al., 2018], plans [Lynch et al., 2020], policies [Edwards et al., 2019], and skills for reinforcement learning [Hausman et al., 2018]. Some of the most common approaches to learning latent space representations in an unsupervised fashion are latent variable models such as Variational Autoencoders (VAEs) [Kingma and Welling, 2013; Rezende et al., 2014] or encoder-decoder based Generative Adversarial Networks (GANs) [Goodfellow et al., 2014; Dumoulin et al., 2016].

One such method is 'Embed to Control' (E2c),[Watter et al., 2015], where latent dynamic model is learnt from raw images to control non-linear dynamic systems. E2C utilises a variational autoencoder, that learns to generate image trajectories from a latent space, where dynamics in the latent space are constrained to be locally linear. Here, the planning for optimal action is carried out entirely in the latent space without access to any observations except for the depiction of the current state. In comparison when the separately trained autoencoders are used for planning, the models failed to discover the underlying structure of the state space including the latent dynamics constraints in the E2c models ensures that the learnt latent space approaches an optimal planar embedding.

Using the E2c model, a visual version of the classic inverted pendulum swing-up task and balancing a cart-pole system was evaluated. The results show that E2C can find embeddings on which control can be performed with ease, and can perform as close to that achievable

by optimal control on the real system model. E2C models provide an accurate long-term prediction by accumulating latent and real trajectory costs to quantify whether the "imagined" trajectory reflects reality. Using global or local E2C models show that trajectories planned in latent space are as good as trajectories planned on the real state space. Jaques et al. [2020] used a VAE to learn a latent dynamics model which is linear in nature. This learnt dynamics model makes it easier to carry out proportional control from high dimensional state spaces like images. NewtonianVAE simplifies imitation learning, by framing imitation learning as a goal inference problem under a switching proportional control law, where a specific proportional controller in different regions of the latent space would move the agent proportionally to match the trajectory.

Losey et al. [2020] used an autoencoder to learn a consistent and controllable latent action, which provides a low dimensional embedding for controlling high-DoF robot actions. These latent actions capture the most important aspects of high-DoF actions. The latent actions for a task-specific training data are learnt in an online manner. The robot has access to multiple demonstrations in the form of a dataset of state-action pairs, and the user then inputs latent actions in the form of a low DoF joystick input while the task is being executed following the expert policy. This is done to map the learned latent dimensions with the joystick DoF. When the user interacts with the robot, their inputs are treated as z, and the robot utilizes its decoder $\phi(a|z)$ to reconstruct high-DoF actions. The algorithm was evaluated on a 7 DoF robot where the end effector follows 4 different trajectories. Users teleoperating the cVAE robot reached their preferred goal more accurately than shared autonomy baselines while requiring less time, effort, and movement. Users controlling the robot arm with low-DoF latent actions completed the tasks more quickly and with less overall effort.

One limitation of this method is that the user has to manually map the latent action while robot is following executing a demonstration, this may be expensive and the user input may wary between demonstration. Another limitation faced is, if the robot reached configurations, where the user had not provided demonstrations, the latent action executed was erratic.

Lynch et al. [2020] proposed a way to scale up skill learning from teleoperated play data called learning from play, where play data is an unbounded sequence of states and actions corresponding to voluntary, repeated, non-stereotyped object interaction between an agent and its environment. The aim is to learn general-purpose policies that can flexibly accomplish a wide range of user-specified tasks, using data that is not task-specific and is easy to collect. Here a task is no longer discrete but continuous indexed by the pair (current state s_c, goal state s_g). Learning in this setting can be formalized as the search for a goal-conditioned policy $\pi(a|s_c, s_g)$ [Kaelbling, 1993]. To learn control from play, Play-LMP simultaneously learns a latent plan representation from play data and a goal-conditioned control policy at test time to achieve specific goals. The algorithm learns to organize play behaviors in a latent space in a self-supervised method, then reuse them at test time to achieve specific goals. The points in the space correspond to behaviors recognized during play that got the agent from some initial state to some final state. A policy conditioned on the current state s_t and the goal state s_g, and a latent plan z, is trained to reconstruct the actions the agent took to reach the goal state from the initial state, as described by inferred plan z. Lynch et al. [2020] experiments show that Play-LMP algorithm, despite not being trained on task-specific data was able to learn up to 18 user-specified manipulation tasks, compared to a collection of single trained BC models. The models trained on play data are far more robust to perturbation than models trained using BC methods, and exhibit natural failure recovery despite not being trained explicitly

to do so.

One approach for reducing the number of interactions with the environment is to learn a self-supervised model which can selectively choose action sequences from the exploratory data to take the agent from the current state to the desired state. This function that labels the exploratory data is called goal-conditioned skill policy (GSP) [Pathak et al., 2018]. Given desired state only trajectory from a demonstration, the GSP can infer the action or the sequence of actions needed to reach the goal from the current state, thereby imitate the task step-by-step. The method penalizes actions that produce states other than the states predicted by the forward dynamics model f. This way multiple actions can be learnt which produce the same resultant state.

Algorithm 4 Imitating Latent Policies from Observation

Learning Latent FDM and Latent Policy

1: **for** $k = 0, 1, 2...\#Epochs$ **do**
2: **for** $i = 0, 1, 2...N-1$ **do**
3: Train latent dynamics parameters
$\theta \leftarrow \theta - \nabla_\theta \min z \left\| G\theta \left(E_{p\theta} \left(s_i \right), z \right) - s_{i+1} \right\|_2^2$
4: Train latent policy parameters
$\omega \leftarrow \omega - \nabla_\omega \| \sum_z \pi_\omega \left(z \mid s_i \right) G_\theta \left(E_{p\theta} \left(s_i \right), z \right) - s^*_{i+1} \|_2^2$
5: **end for**
6: **end for**

Action remapping:

1: Observe state s_0
2: **for** $t = 0, 1, 2, ...\#Interactions$ **do**
3: Choose latent action $z_t \leftarrow \underset{z}{\text{argmax}} \pi_\omega(z|E_{a\zeta}(s_t))$
4: Take - greedy action $a_t \leftarrow \underset{a}{\text{argmax}} \pi_\zeta(a|z_t, E_{a\zeta}(s_t))$
5: Observe state s_{t+1}
6: Infer closest latent action
$z_t = \underset{a}{\text{argmin}} \left\| E_{p\theta} \left(s_{t+1} \right) - E_{p\theta} \left(G_\theta \left(E_{p\theta} \left(s_t \right), z \right) \right) \right\|_2$
7: Train action remapping parameters
$\zeta \leftarrow \zeta + \Delta_\zeta \; log \frac{\pi_\zeta(a_t|z_t, E_{a\zeta}(s_t))}{\sum \pi_\zeta(a_t|z_t, E_{a\zeta}}$
8: **end for**

Another method to reduce interaction with the environment is to learn a task only from observations independent of the dynamics of the agent. This is done by carrying out learning an imitation policy using latent actions (ILPO) [Edwards et al., 2019]. Using the demonstrated data the agent learns a latent forward dynamics model $f(s_{t+1}|E_p(s_t), z)$ where z is a latent action that causes a transition from s_t to s_{t+1}, and E_p is the state encoding. The forward dynamics model f has to learn these transitions in terms of states of the world after taking a latent action labels z, as ground true action is not yet known. The agent replicates the demonstrated trajectory by learning a latent policy $\pi_\omega(z|E_p s_t)$, which represents the latent action z that needs to be executed in state s_t. The latent forward model f and the latent policy π are learnt offline with only the demonstrated observations and independent of the agent interacting with the environment. The agent then interacts with the environment

to learn an action remapping $\pi_\omega(a|z, E_p s_t)$, which is a mapping from latent action z_t to true actions a_t in a supervised manner. The action remapping depends on the current state s_t because latent action learnt is not necessarily invariant across state space. In order to imitate an expert's trajectory, the agent must find the latent cause that will result in the transition the expert intended, $z^* = argmax_z \pi_\omega(z|s_t)$, followed by finding the true action that can result in this transition, $a^* = argmax_a \pi_\xi(a|z, s_t)$. The agent can then follow this policy to replicate the expert's behavior without any expert actions being provided explicitly.

ILPO was proven to be more sample efficient compared to BCO when tested in classic control task like cartpole, acrobot, and mountain car, since BCO requires interacting with the environment for learning an inverse dynamics model and then executing the updated policy for collecting data (for behavioral cloning for policy improvement).

3-6 Conclusions

In this chapter, we discussed the latest methods and algorithms for Imitation Learning (IL). We briefly elaborated on the Imitation from Observation (IfO) and methods to learn tasks using state-space information. Interactive algorithms show that human feedback given while learning the task can improve agent learning rate and learn a more robust policy.

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

- Among interaction-based learning approaches, 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] to guide agents while learning policy. Providing feedback to the agent in the action-space (Eg. joint torques for a robotic arm) is often not intuitive for the demonstrator. The user requires significant prior knowledge of the dynamics of the agent. TIPS [Jauhri et al., 2020] showed that providing corrections in state action space is more intuitive for the expert. This state-space feedback mechanism also leads to a significant reduction in demonstrator task load.

- To solve the control problem of learning using state-space information, many Imitation from Observation (IfO) methods use or learn a dynamics model of the system, either an Inverse Dynamics Model (IDM) [Torabi et al., 2018a; Nair et al., 2017] or a Forward Dynamics Model (FDM)[Edwards et al., 2019]. Using these dynamic model, the action need to produce the necessary state-to-state transition can be inferred. Using the state and the inferred action, a regression policy is learnt in state-action space. In such approaches, if the dynamics of the environment changes, the learnt imitation policy may not be useful and the policy may need to be re-learnt. The agent may need additional effort from the demonstrator to train the policy in the new environment.

- In ILPO, [Edwards et al., 2019], the imitation policy is learnt in state latent action space. The method characterizes the causal effects of latent actions on observations while simultaneously predicting their likelihood by learning a latent policy and a latent forward dynamics model. The model remaps the latent action to real action by leveraging a small number of environment interactions to determine a mapping between the latent and real-world actions. In the new environment, latent policy learnt in the

original environment can be reused, only latent and real-world actions remapping has to be learnt for imitating the observed behavior.

- Hence it's useful to learn an imitation policy in state space interactively like in TIPS [Jauhri et al., 2020] or from state only demonstrations like in ILPO [Edwards et al., 2019], such that the learnt policy can be reused even if the dynamics of the environment changes. To use the learnt state-space policy, like TIPS, an Indirect Inverse Dynamics Model (IIDM) can be learnt to compute the action need to produce the state to state transitions. If the dynamics of the environment changes, the agent can reuse the imitation policy by relearning the dynamics model without any additional expert actions or demonstration given.

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

Chapter 4

Task-Space Policy Learning

As mentioned in the last chapter, there is a need for learning a state space policy (either interactively or from offline data) that can be reused in a new environment where the agent's dynamics are different from the original environment. To use the learnt state-space policy, the agent can compute the necessary action needed to produce the state to state transitions. If the dynamics of the environment changes, the agent would be able to execute the task in the new environment without any additional demonstrations, by reusing the state space policy (previously learnt in the original environment) and the dynamics model of the agent in the new environment.

This chapter details our proposed method **Task-Space Policy Learning (TaSPL)**, where we learn a generalized imitation policy in state-space, independent of the model dynamics which is used for predicting the next state the agent has to reach to replicate the task. We propose 2 methods, one, an offline policy learning algorithm, which learns the imitative policy from state only demonstrations; two, an interactive model in which a human demonstrator can interactively teach the agent an imitative policy by providing feedback in state space. We presume environments with continuous state spaces ($s \epsilon S$) with unknown dynamics and discrete action spaces. The algorithms could be applied to continuous action spaces, but we chose to prove our method in the discrete action spaces.

Section 4-1 provides a detailed explanation of the learning framework of both algorithms. In TaSPL, a policy is learnt in state space, either from data collected from sequence of state only demonstration or from a human demonstrator modifying the current state.

Section 4-2 provides an explanation on the indirect inverse dynamics mechanism that is used in TaSPL. This is used to compute the action that is required to produce the necessary state transition from the current state to the predicted next state.

Section 4-3 outlines the algorithms in full.

Section 4-4 discusses the advantages and disadvantages of TaSPL.

4-1 Learning Framework

The general idea of Next state predictor is that the agent learns a policy in state space, such that given the current state (s_t), the policy can be used to predict the next ideal state (s_t^{des}) the agent has to be in to successfully carry out task. This can be done either in an **offline manner** using state only demonstration data or in an **interactive manner** where the human demonstrator provides feedback during execution time to shape the policy.

In the **offline method**, from a set of expert demonstrations described through noisy state observations $s^{1 \dots s^{Ne} \epsilon D}$. we learn to predict a policy in state space. As such, noise is necessary for ensuring that there as little covariate shift in data used to learn the policy from.

In the **interactive method**, the human feedback is then used to update the agent's policy on-line i.e. during the execution of the current policy. The human feedback is used to modify the current state using a binary feedback mechanism, similar to that used in TIPS[Jauhri et al., 2020] (inspired by COACH [Celemin and Ruiz-del Solar, 2019], where the human feedback is used to modify the policy). The expert modifies an observable state to imply a change in the state similar to the modification in state carried out in TIPS [Jauhri et al., 2020].

$$s_{t+1}^{des} = s_t + h_t.e \tag{4-1}$$

where, h_t is the binary human feedback (where $h_t \, \epsilon \, \{+1, 0, -1\}$) and e is a constant change between the current state and the next desired state. The performance of TIPS [Jauhri et al., 2020] shows that the demonstrator is able to teach an optimal policy by just suggesting modifications in an observable partial state. The performance of COACH [Celemin and Ruiz-del Solar, 2019] and D-COACH [Pérez-Dattari et al., 2019], show the efficacy of the binary feedback mechanism as compared to providing an exact value of the desired action. Furthermore, the binary feedback mechanism is simpler for the demonstrator. Even if the computed next state (s_{t+1}^{des}) using binary feedback is slightly larger or smaller than the true next state(s_{t+1}), Celemin and Ruiz-del Solar [2019] and Pérez-Dattari et al. [2019] show that it is sufficient to show the direction of change.

A policy is learnt using a feed-forward artificial neural network. The neural networks are trained by carrying out supervised learning on the collected data, taking the current state s_t as input and the next state s_{t+1} as target. The training is done in two steps, immediately with the current state-next state sample, and again in a batch process periodically in every T time-steps.

Once the desired next state(s_{t+1}^{des}), is known (from either the offline method or online method), we can compute the action required a_{des} for this state transition from the current state s_t to the next state s_{t+1}, we use the indirect inverse dynamics model (IIDM) as proposed in TIPS [Jauhri et al., 2020].

The computed action a_{des} by the IIDM model can be immediately executed by the agent. This makes it easy for the human demonstrator to observe the effect of the feedback that they have just provided. The immediate execution of the action also helps in the shaping of the policy, the human feedback provided helps to train the model faster, as the agent does not have to explore undesirable states (unlike reinforcement learning).

The advantage of learning the policy in state space is that, should the dynamics of the environment change, the agent would only need to relearn the forward dynamics model used in IIDM. Using the previously learnt policy, the next desired state can be predicted and using the new forward dynamics model learnt, the agent can compute the desired action a_{des} to produce the necessary transition from the current state(s_t) to next state(s_{t+1}). The implication of this, is that the agent will be able to execute the task and reuse the policy learnt even if the dynamics changes, the agent simply has to sample all possible actions from the IIDM for every time step in real-time, making the algorithm unsuitable for high dimensional action spaces. Consequently, the performance of the agent in the new environment is dependent on the policy learnt in the original environment and the quality of the FDM learnt in the new environment.

4-2 Computing Actions via Indirect Inverse Dynamics

For computing the action (a_t) that leads to the desired state transitions from ($s_t \rightarrow s_{t+1}^{des}$), the an Inverse Dynamics Model (IDM) can be used, like in BCO [Torabi et al., 2018a]. However, using IDM would not work for the following reasons [Jauhri et al., 2020] :

- There may exist some state transitions $s_t \rightarrow s_{t+1}$ that have multiple valid actions a_t, then an IDM learnt with multiple solutions to the regression function $f_\theta(s_t, s_{t+1}) = a_t$ would produce invalid results.
- There may exist kinematic redundancy in the agent or conditions in the environment such that there may exist multiple actions that produce the same ($s_t \rightarrow s_{t+1}^{des}$) transition. In such cases, the IDM regression model fails to predict the right solution.
- The state transition ($s_t \rightarrow s_{t+1}^{des}$) suggested by the demonstrator may not be possible for the agent to execute. Here the IDM methods would not provide a valid solution hence the method fails.
- In the interactive algorithm, the demonstrator provides feedback in the partial state space, modifying only a few perceivable states, which can lead to ambiguity regarding the desired state transition in the remaining dimensions.

To avoid these challenges, we propose to use the indirect inverse dynamics mechanism proposed in TIPS [Jauhri et al., 2020]. This method involves sampling possible actions ($a \epsilon A$) and using a learnt forward dynamics model (f) to predict the next states ($s_{t+1} = f(s_t, a_t)$) for these actions. The action that results in a subsequent state that is closest to the desired state in the full state dimensions is chosen. Mathematically it is written as:

$$a_t^{des} = \underset{a}{\operatorname{argmin}} \left\| f_\theta(s_t, a_t) - s_{t+1}^{des} \right\| \tag{4-2}$$

where $a \epsilon A$.

In discrete action spaces, the IIDM queries all possible actions and in continuous action spaces, the actions are uniformly sampled from the action-space A. This indirect inverse dynamics formulation avoids the infeasible transition problem by choosing the action that

brings the agent closest to the desired next state, regardless of whether the desired state transition is feasible. If the policy is learnt in partial state-space i.e. $o_{t+1} = \pi(o_t)$ where $o \subseteq s$, then the computation of the action can be done in partial space, the equation 4-2 can be modified as follows.

$$a_t^{des} = \underset{a}{\operatorname{argmin}} \left\| w.s_{t+1}^{des} - f_\theta(s_t, a_t) \right\| \tag{4-3}$$

where w is a weight matrix that is used for mapping from a smaller observation state (o_t) space to the full state space (s_t). However, computing action in partial space may pose the challenge in which there multiple viable actions can result in the desired state transition. One method to solve the ambiguity in action selection is to select actions that result in the least predicted change in the full state i.e. minimizes $\|s_{t+1} - s_t\|$. Another method would be to add a small weight in the weight matrix w on an unobserved state, so that action selected moves the agent to a more stable state.

The advantage of using such a method is that the action to be executed is computed using the FDM model learnt in that environment, dependent on the state to state transitions, making it an ideal method for computing actions from a policy learnt in state space. The success of the policy depends on the accuracy of the forward dynamics model learnt in that environment.

4-2-1 Dynamic Model Learning

A forward dynamics model f is learnt from state, action, state triplets (s_t, a_t, s_{t+1}), using a feed-forward artificial neural network trained in a supervised-learning fashion. The collection of (s_t, a_t, s_{t+1}) triplets and the training of the neural network is carried out in two different ways, (1) following a random policy (π_{rand}) to collect (s_t, a_t, s_{t+1}) triplets and training the neural network prior to evaluating the state space policy (π_s), (2) the (s_t, a_t, s_{t+1}) triplets are collected whilst following the learnt state space policy (π_s) and the FDM model is trained in an online manner.

Dynamics model learning for Offline TaSPL

In the offline TaSPL, the FDM is learnt in an online manner whilst interacting with the environment following the state space policy (π_s). In state (s_t), the agent predicts the next desired state s_{t+1}^{des}, the agent then queries the untrained IIDM model f_θ for the desired action (a_t^{des}). On executing the computed action, the agent reaches s_{t+1}. The current state, the executed action, the visited next state are saved in the experience buffer E, the FDM model f_θ is trained in an online manner. The state-transitions and actions taken by the agent are continuously added to the experience buffer E and the FDM is updated periodically every T_{update} time-steps. The update is done using all prior experience i.e. batches sampled from the experience buffer E. The online model learning not only learns the dynamics in state-action spaces that are newly visited by the agent as it executes its policy, also reduces the need for learning state transitions never taken by the agent to successfully complete the task.

Dynamics model learning for the Interactive TaSPL

In the Interactive algorithm, prior to the expert providing demonstrations to the agent, the agent interacts with the environment to learn the dynamics by executing a random policy π_{rand}. The (s_t, a_t, s_{t+1}) visited are saved on an experience buffer E which is used to train the FDM model f. A learnt FDM is necessary since the teacher provides feedback in state space by suggesting a desired next state, the agent has to compute the necessary action needed to be executed to produce the necessary state transition. The policy is trained in an online manner, while executing the current policy, if an untrained FDM model is used, the agent can execute an action which would produce state transition different from what the teacher intended, this, in turn, would make the teaching difficult for a human teacher and increase the demonstrator effort significantly. Optionally, the initial FDM could then further be tuned using the state visited during the teaching phase.

4-3 Algorithms in Full

4-3-1 Offline TaSPL

The full procedure for Offline TaSPL can be seen in Algorithm 5.

In the first phase, state-next state (s_t, s_{t+1}) samples are collected into a demonstration buffer D, following an expert demonstration either got from a human or an RL policy or using a closed-loop controller (line 4). But in order to learn a robust policy and to reduce covariate shift in the data used for training the policy, the agent/teacher injects noise in the form of random exploration by executing random action 20% of time (line 8). We inject noise into the supervisor's policy while demonstrating, similar to DART [Laskey et al., 2017b], this forces the supervisor to demonstrate how to recover from errors. This is done in order to collect trajectories from unstable states to that simulate the errors that the agent may make over time. The policy π_s, represented by a feed-forward neural network is trained on the samples collected in the experience buffer E, in a supervised learning fashion (line 12).

In the second phase, The agent uses the learnt policy π_s to predict the next desired state the agent should ideally be in (line 3). The action necessary for this state transition is computed using the IIDM explained in section 4-2 (line 4). While executing the action computed by the IIDM, the state (s_t, s_{t+1}) visited and the action executed a_t are saved on an experience buffer E which is used to train the FDM $f\theta$ in an online fashion while interacting with the environment (line 6-3).

In case the dynamics of the environment changes changes, the agent has to only execute the second online model learning phase, i.e. to relearn or fine-tune the original FDM f by retraining the neural network using the state, action, state triplets (s_t, a_t, s_{t+1}) collected in the new network. The agent would be able to perform the task using the originally learnt policy π_s and the new FDM f_ϕ.

4-3-2 Interactive TaSPL

The full procedure for interactive TaSPL can be seen in Algorithm 6.

Algorithm 5 Offline TaSPL

State-Space Policy learning

Required : Expert policy π_e

Initialize : Demonstration buffer $D = []$, Agent state space policy π_s

for $i = 0, 1, 2, 3...N_e$ **do**
  Visit state s_t
  if $random(0, 1) \leq 0.8$ **then**
    Execute expert policy $a_t = \pi_e(s_t)$
    Observe State s_{t+1}
    Append s_t, s_{t+1} to Demonstration buffer D
  else
    Execute random action a_t
    Observe State s_{t+1}
  end if
end for
Learn state space policy model π_s using inputs $\{s_t\}_1^{Ne}$ and targets $\{s_{i+1}\}_1^{Ne}$

Online forward dynamics model learning:

Initialize: Forward-dynamics model f_θ, Experience buffer $E = []$

for $t = 0, 1, 2, ...\#Interactions$ **do**
  Visit state s_t
  Use policy to predict next state, $s_{t+1} = \pi_s(s_t)$
  Compute action $a_t^{des} = \underset{a}{\operatorname{argmin}} \left\| f_\theta(s_t, a) - s_{t+1}^{des} \right\|$
  Execute action $a_t = a_t^{des}$, reach state s_{t+1}
  Append (s_t, a_t, s_{t+1}) to experience buffer E
  if $mod(t, T)$ **then**
    Update policy f_θ using batch sampled from demonstration buffer E
  end if
end for

In the initial offline model learning phase, a forward dynamics model f_θ of the agent is learnt using a feed forward neural network. The network is trained using samples generated by executing a random exploration policy π_r (line 3).

In the demonstration phase, when considered necessary the human demonstrator suggests the next desired state s_{t+1} the agent should be in by advising relative changes to the observable states of the current vector i.e. $s_t \rightarrow s_{t+1}^{des}$ (line 6). The agent then computes the action required for this state transition by using the indirect inverse dynamics model as detailed in 4-2 (line 13). The computed action a_t^{des} is also executed immediately (line 14). The imitation policy is updated using the (s_t, s_t^{des}) pair. The (s_t, s_t^{des}) pair is appended to the demonstration buffer D. The policy π_s is retrained periodically in batch training using samples stored in the demonstration buffer D (line 15).

Similar to the offline algorithm, if the dynamics of the environment changes changes, the agent has to only know the dynamics model of the agent in the new environment to execute the task by reusing the policy π_s learnt in the original environment. This knowledge of the dynamics of the environment can either be provided as a known model, or the model can be learnt from interactions with the environment. These interactions can be obtained either by following a random/exploratory policy (learning the FDM in an offline manner) or following the current state space policy (learning the FDM in an online manner). If the new environment is similar to the original environment the FDM was trained in, the agent can fine tune the original FDM f_θ by retraining the neural network using the state, action, state triplets (s_t, a_t, s_{t+1}) collected in the new setting. If the environment dynamics of the new environment is very different from that of the original environment, the agent could learn the FDM from scratch.

Algorithm 6 Interactive TaSPL

Initial Model-Learning Phase

Generate N_e experience samples $s_t, {a_t}_1^{N_e}$ by executing a random/exploration policy π_r
Append samples to experience buffer E
Learn forward dynamics model f_θ using inputs $s_t, {a_t}_1^{N_e}$ and targets ${s_{t+1}}_1^{N_e}$

Teaching Phase:

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**
      Compute desired state $s_{t+1}^{des} = s_t + h_t.e$
      Append (s_t, s_{t+1}^{des}) to demonstrations buffer D
      Update policy π_s using pair (s_t, s_{t+1}^{des}) and using batch sampled from D
    else
      ## No feedback
      Use policy to predict next state, $s_{t+1} = \pi_s(s_t)$
    end if
    Compute action $a_t^{des} = \underset{a}{\operatorname{argmin}} \left\| f_\theta(s_t, a) - s_{t+1}^{des} \right\|$
    Execute action $a_t = a_t^{des}$, reach state s_{t+1}
    Append (s_t, a_t, s_{t+1}) to experience buffer E
    if $mod(t, T)$ **then**
      Update policy π_ϕ using batch sampled from demonstration buffer D
    end if
    (Optional) Update learnt FDM f_θ using samples from experience buffer E
  end for
end for

4-4 Discussion

In this chapter, we present our proposed method of "Task-Space Policy Learning" from data or from human corrective feedback to teach the agent to carry out an imitation task.

We explained the learning framework used for learning a trajectory and how learning the trajectory allows the agent to execute the task even if the dynamics of the environment changes changes. We showcase two methods that allow the agent to learn trajectory either from state only demonstration or from corrective feedback in state spaces inspired by TIPS [Jauhri et al., 2020]. Learning the policy in state space allows the agent to execute the task in the new environment by reusing the policy from the old environment and knowledge of the agent's dynamics model in the new environment. The dynamics model can be either provided or can be learnt from interaction with the environment.

However, there are some considerations that can limit the performance of the proposed algorithm. Firstly, the algorithm queries the dynamics model in each step for computing the action that produces the desired state transition. The performance of our algorithm is dependent on the quality of the learnt state space policy and the dynamics model. Learning a dynamics model through random exploration can be difficult in many environments and typically requires a significant number of environment interactions. The proposed algorithm computes action by sampling random actions in a given state, followed by execution of the chosen action. Secondly, the proposed algorithm suffers the limitations of IIDM outlined in TIPSJauhri et al. [2020]. Computing actions from IIDM would not be scalable in high dimensional continuous action spaces, such computation would require more computational resources and affecting the real time execution.

One solution to avoid querying the FDM in each step is to learn state action policy in the new environment, without the need for additional human teacher interventions. The state space policy learnt in the original environment can be used to train or retrain a state-action policy in the new environment. The above proposed method was implemented as a modification to TIPS [Jauhri et al., 2020], where the state action policy learnt in the original environment is retrained using the state space policy learnt in the original environment. The state space policy is learnt simultaneously in an online manner while the demonstrator trains the state action policy. In the new environment, no additional demonstrative effort is required for training the state action policy. The advantage of such a method is that the agent can be trained in a human intuitive environment, but the agent is able to perform the same task in a new environment where the dynamics of the environment may not be intuitive. However, the quality of the new state action policy learnt is dependent on the quality of the state space policy (π_s) and the forward dynamics model f_θ learnt. The full procedure for the modification to TIPS can be seen in Algorithm 7. Algorithm 8 outlines the procedure to be followed to retrain a state action policy in the new environment.

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

Algorithm 7 Modified TIPS Algorithm

Initial Model-Learning Phase

Generate N_e experience samples s_t, a_t, s_{t+1} by executing a random/exploration policy π_r
Append samples to experience buffer E
Learn forward dynamics model f_θ using inputs $\{s_t, a_t\}_1^{Ne}$ and targets $\{s_{t+1}\}_1^{Ne}$

Teaching Phase:

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**
      Compute desired partial state $s_{t+1}^{des} = s_t + h_t.e$
      Compute action $a_t^{des} = \underset{a}{\operatorname{argmin}} \left\| f_\theta(s_t, a_t) - s_{t+1}^{des} \right\|$
      Compute next state $s_{t+1} = f_\theta(s_t, a_t^{des})$
      Append (s_t, a_t^{des}) to demonstrations buffer D
      Append (s_t, s_{t+1}) to stats space demonstrations buffer D_s
      Update policy π_ϕ using pair (s_t, a_t^{des}) and using batch sampled from D
      Update policy π_s using pair (s_t, s_{t+1}^{des}) and using batch sampled from D_s
      Execute action $a_t = a_t^{des}$, reach state s_{t+1}
    else
      ## No feedback
      Execute action $a_t = \pi_\phi(s_t)$, reach state s_{t+1}
    end if
    Append (s_t, a_t, s_{t+1}) to experience buffer E
    if $mod(t, T_{Update})$ **then**
      Update policy π_ϕ using batch sampled from demonstration buffer D
      Update policy π_s using batch sampled from demonstration buffer D_s
    end if
    (Optional):Update learnt FDM f_θ using samples from experience buffer E
  end for
end for

Algorithm 8 Algorithm to retrain state-action policy in new environment

Initial Model-Learning Phase

Generate N_e experience samples $s_t, a_t{}_1^{N_e}$ by executing a random/exploration policy π_r
Append samples to experience buffer E
Learn forward dynamics model f_θ using inputs $s_t, a_t{}_1^{N_e}$ and targets $s_{t+1}{}_1^{N_e}$

Retraining Phase:

for *episodes* **do**
  for $t = 0, 1, 2, ...T$ **do**
    Visit state s_t
    Use state space policy to predict next state, $s_{t+1} = \pi_s(s_t)$
    Compute action $a_{t+1}^{des} = \underset{a}{\operatorname{argmin}} \left\| f_\theta(s_t, a_t) - s_{t+1}^{des} \right\|$
    Append (s_t, a_t^{des}) to demonstrations buffer D
    Update policy π_ϕ using pair (s_t, a_t^{des}) and using batch sampled from D
    Execute action $a_t = \pi_\phi(s_t)$, reach state s_{t+1}
    if $mod(t, T_{Upate})$ **then**
      Update policy π_ϕ using batch sampled from demonstration buffer D
    end if
    (Optional:) Update learnt FDM f_θ using samples from experience buffer E
  end for
end for

Chapter 5

Experimental Setting

In this section, we provide information about the experimental setting for the evaluation of the proposed method: Task Space Policy Learning (TaSPL). We evaluate and compare TaSPL to other techniques based on two main criteria: (i) the task performance of the trained agent over time and (ii) the ability of the agent to learn generalization of the policy to environments with different dynamics. The performance of the TaSPL agent is compared with other agents trained using standard Imitation Learning (IL) techniques.

Offline TaSPL (algorithm 5), where the agent learns a policy from state only demonstrations is compared with BCO [Torabi et al., 2018a] and ILPO [Edwards et al., 2019]. Interactive TaSPL (algorithm 6), where the agent learns a policy interactively from corrections in state space is compared with TIPS [Jauhri et al., 2020] and its modified version proposed in section 4-4.

Exhaustive experiments for evaluation and comparison of agents trained with TaSPL were carried out with simulated environments from OpenAI gym. Additionally, the proposed method was validated in two different manipulation tasks with a real robot arm KUKA iiwa.

Section 5-1, provides details of the implementation of the algorithm. Section 5-2 describes the different environments where the algorithm was evaluated. It also describes the modification done to the environment, to modify the dynamics in order to test the learned policies when the environment of the policy changes. Section 5-3, 5-4 provides details of the experimental setup used and provides a list of the algorithms that are used for comparing the performance of the two algorithms.

5-1 Implementation of the algorithms

The state space policy (π_s) and forward dynamics model(f) are implemented as simple feed-forward artificial neural networks. Training of the neural networks is done in a standard supervised learning fashion. TensorFlow2 python library is used to implement and train the neural networks. 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 5-1.

Table 5-1: Parameter settings in the implementation of TaSPL for different evaluation tasks

Parameter	Environments				
	CartPole	MountainCar	Pendulum	Acrobot	LunarLander
No. of exploration samples (N_e)	500	1000	500	500	5000
Periodic policy update interval (T_{update})	10	10	10	10	10
FDM Network (f_θ)					
Network layer sizes	16,16	16,16	32,32	32,32	64,64
Learning rate	0.005	0.0025	0.0025	0.0025	0.0025
Batch size	16	32	32	32	32
State Space Policy Network (π_s)					
Network layer sizes	32, 32	32, 32	32, 32	128, 128	128, 128
Learning rate	0.005	0.005	0.005	0.005	0.005
Batch size	32	64	64	64	64

For training the policy and the forward dynamics model neural network in the original environment, the neural nets are initialized with random small weights. The models are trained in a supervised learning manner following the procedure outlined in algorithm 5 and 6. In the new environment with new dynamics, the policy from the original environment can be reused, without additional training which would not be possible by other methods. The dynamics model neural networks can be either retained from random small weights or from weights tuned from the original environment. For learning the dynamic model, we use a random exploration policy (π_r) and the number of initial exploration samples (Ne) varies with the size of state-action space and complexity of the task to be learnt. The parameters for each of the tasks used for evaluation can be seen in Table 5-1.

5-2 Evaluation Domain - OpenAI Gym

The OpenAI gym [Brockman et al., 2016] is an open source toolkit used primarily for reinforcement learning research. It contains a collection of simulated benchmark environments used for the purpose of evaluating learning agents. We use a few such environments for the evaluation of agents trained via TaSPL, namely: CartPole, MountainCar, Pendulum, Acrobot and LunarLander (depicted in Figure 4.1).

The reward function from each OpenAI gym environment has no influence on the learning process of TaSPL, hence it can be used as a performance index of the policy learnt by the agent.

In order to evaluate the learnt state space policy, physical properties like the mass, length, magnitude and direction of force were modified from the original OpenAI Gym environment to create new environments where the dynamics of the environment are different from the original environment. In each modified environment, the agent was allowed to learn a dynamics model from interaction with the environment. The policy from the original environment and the FDM learnt in the new environment is used for evaluating the task in the new environment.

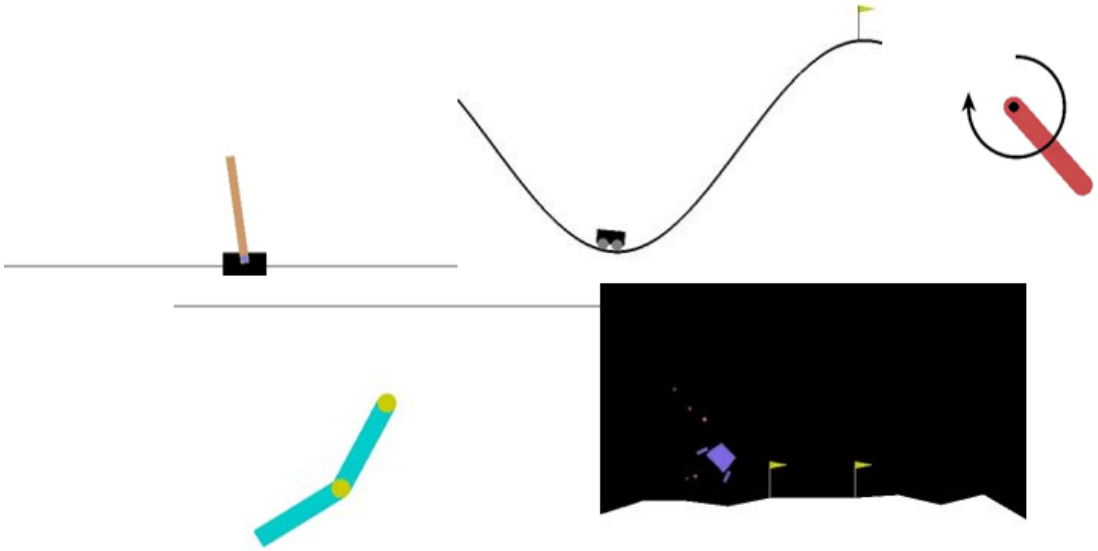


Figure 5-1: Screenshots of environments (clockwise) the CartPole, MountainCar, Pendulum, LunarLander and Acrobot from OpenAI GymBrockman et al. [2016]

5-2-1 CartPole - Original environment

This is a classic control problem that involves balancing a pole attached by an un-actuated joint to a cart moving on a frictionless plane [Barto et al., 1983]. The objective of the task is to prevent the pole from falling over by changing the cart's position. The cart is controlled by applying a force to it. For every time step the pole is upright a reward of +1 is provided. If the pole is more than 40 degrees from the vertical or if the cart moves more than 2.4 units from the center, the task is considered as a fail. The maximum length of an episode is 500 time-steps

5-2-2 Cart-pole - Modified environment

Physical properties like the mass of the pole ($m_p = 1$), mass of the cart ($m_c = 1$), length of pole($l = 1$), magnitude and direction of force($f = 1$) were modified from the original CartPole environment in order to test the learnt policy in situations with different dynamics.

- In CartPoleEnv_mph, the mass of the pole was halved, $m_p = 0.5$
- In CartPoleEnv_mpd, the mass of the pole was double, $m_p = 2$
- In CartPoleEnv_mch, the mass of the cart was halved, $m_c = 0.5$
- In CartPoleEnv_mcd, the mass of the cart was double, $m_c = 2$
- In CartPoleEnv_lh, the length of the pole was halved, $l = 0.5$
- In CartPoleEnv_ld, the length of the pole was double, $l = 2$
- In CartPoleEnv_fh, the force applied on the cart was halved, $f = 0.5$

- In CartPoleEnv_fd, the force applied on the cart was double, $f = 2$
- In CartPoleEnv_fn, the force applied on the cart was negative, $f = -1$

5-2-3 MountainCar - Original environment

This is a classic control problem that involves an under powered car on a one-dimensional track, positioned between two mountains. The objective of the task is to drive the car to the top of the mountain on the right. The car's engine being under powered is not powerful enough to drive up the mountain in a single pass. The agent has to build up enough momentum by driving back and forth in the valley between the two mountains, to be able to drive up the mountain can reach the desired position[Moore, 1990]. For every time step the car hasn't reached the desired state, a reward of -1 is provided. If the car takes more than 200 time steps, the task is considered to be failed.

5-2-4 MountainCar - Modified environment

The direction of the force($f = 1$) was modified from the original MountainCar environment in order to test the learnt policy in situations with different dynamics.

- In MountainCarEnv_fn, the force applied on the cart was negative, $f = -1$

5-2-5 Pendulum - Original environment

The pendulum swing-up task is a classic control commonly used in optimal control literature. Here, the pendulum starts in a random position and velocity, and the objective is to swing it upright with the least amount of angular velocity. The pendulum is a link actuated at the point of rotation, however, the actuation force is insufficient to rotate the pendulum to the upright position. The agent has to build up enough momentum by swinging the link clockwise and anticlockwise to bring the link to its upright position and maintain an upright position.[Moore, 1990].

5-2-6 Pendulum - Modified environment

Physical properties like the mass of the pole ($m_p = 1$), magnitude and direction of force($f = 1$) were modified from the original Pendulum environment in order to test the learnt policy in situations with different dynamics.

- In PendulumEnv_mph, the mass of the pole was reduced, $m_p = 0.75$
- In PendulumEnv_mpd, the mass of the pole was increased, $m_p = 1.5$
- In PendulumEnv_fh, the torque applied on the link was reduced, $f = 0.75$
- In PendulumEnv_fd, the force applied on the cart was increased, $f = 1.5$
- In PendulumEnv_fn, the force applied on the cart was negative, $f = -1$

5-2-7 Acrobot - Original environment

The Acrobot is a two-link robotic arm in the vertical plane with a single actuator between the links [Murray and Hauser, 1991]. Initially, the links are hanging downwards, the objective of this task is to swing the end of the lower link up to a given height. In order to swing up and balance the entire system, the agent must build momentum in the system by actuating the joint in between to reach the necessary height. For every time step the lower limb hasn't reached the desired height, a reward of -1 is provided. If the agent takes more than 500 time steps the task is considered to be failed.

5-2-8 Acrobot - Modified environment

Physical properties like the mass and its distribution of the two links were varied ($m_1 = 1, m_2 = 1$), magnitude and direction of force($f = 1$) were modified from the original Acrobot environment in order to test the learnt policy in situations with different dynamics.

- In AcrobotEnv_mh, the mass of the links was reduced, $m_1 = 0.75,\ m_2 = 0.75$
- In AcrobotEnv_mv1, the mass of the links was varied, $m_1 = 0.8,\ m_2 = 1.2$
- In AcrobotEnv_mv2, the mass of the links was varied, $m_1 = 1.2,\ m_2 = 0.8$
- In AcrobotEnv_fh, the torque applied on the link was reduced, $f = 0.75$
- In AcrobotEnv_fd, the torque applied on the cart was increased, $f = 1.5$
- In AcrobotEnv_fn, the torque applied on the cart was negative, $f = -1$

5-2-9 LunarLander - Original environment

The objective of the task is to land a spaceship between two flags smoothly. The ship has 3 actuators to control the flight. One actuator points downward and the other 2 points in the left and right direction, these affect the spaceship's vertical and angular velocity. The space-ship starts at the top of the screen with a random angular velocity, the agent needs to navigate the space ship to the landing pad at the bottom. If the space-ship lands with zero velocity between the flags it receives a reward of 140. Firing of the actuators in each step costs -0.3 points each step. The episode ends if the spaceship goes out of the frame or crashes or lands successfully, receiving an additional -100 or +100 respectively.

5-2-10 LunarLander - Modified environment

Magnitude of main engine power($f_{MAIN\_ENGINE\_POWER} = 13$), magnitude and direction of side engine power($f_{SIDE\_ENGINE\_POWER} = 0.6$), were modified from the original LunarLander environment in order to test the learnt policy in situations with different dynamics.

- In LunarLanderEnv_mh, the main engine power was reduced, $f_{MAIN} = 10$

- In LunarLanderEnv_md, the main engine power was increased, $f_{MAIN} = 16$
- In LunarLanderEnv_sh, the side engine power was reduced, $f_{SIDE} = 0.4$
- In LunarLanderEnv_sh, the side engine power was increased, $f_{SIDE} = 0.9$
- In LunarLanderEnv_sh, the side engine power was negative, $f_{SIDE} = -0.6$

5-3 Experiment Setup for Offline Policy learning

The data for training the policy is obtained from state only demonstrations. We recorded a data set of state, next state (s_t, s_{t+1}) pairs from an oracle based on a closed-form policy [Xiao, 2020]. The agent follows the teacher(oracle) 80% of the time, and executes random action 20% of the time, as mentioned in section 4-3-1 inspired by DART. Data is not collected when a random action is executed.

A state space policy π_s is trained on the collected state - next state pairs in an offline manner (without any further interaction with the environment), following the steps outlined in algorithm 5. Once the agent has learnt a policy from the collected data, it learns the forward dynamics model needed for the action computation, in an online manner while executing the learnt state space policy. To exploit the policy learnt in state space, the agent is made to execute its policy in new environments where the dynamics model is different from that of the original environment from wherein the demonstration data was recorded.

We evaluate and compare TaSPL to other techniques based on two main criteria: (i) the task performance of the trained agent over time and (ii) the ability of the agent to be able to execute the task it was trained to do, but in a new environment where the agent dynamics are different from that of the original environment. The performance of the TaSPL agent is compared with other agents trained using standard Imitation Learning (IL) techniques.

- Torabi et al. [2018a] proposed Behavior Cloning from Observation (BCO), a supervised learning method intended to imitate the demonstrator using state-only demonstrations. The agent learns an inverse dynamics model using an exploratory policy, and then uses that model to infer the expert's missing action information. With the states and the inferred actions of the demonstrator, behavioral cloning is used to learn the policy. Since the agent is inferring the action from the IDM model learnt to train a BC policy, the agent is able to adapt the change in the environment dynamics.
- Edwards et al. [2019] proposed Imitating Latent Policies from Observation (ILPO), using the state only demonstration, a latent policy $\pi_z(st)$ and a latent forward model $f(s_t, z)$ is learnt in latent action (z) space. Subsequently, the agent maps the latent actions(z) to actual actions (a) in supervised learning fashion. In a new environment where the model dynamics are different, the agent only has to relearn the mapping from latent actions(z) to actual actions(a) and thus the agent is able to execute the task in the new environment.

All three algorithms tested utilised the same data set of state only demonstrations for learning the imitation policy. Once a policy is learnt in the original environment, the policy is tested in a new environment where the agent's dynamics are different.

5-3-1 Ablation Study

From the discussion in section 4-4, we know that the performance of the agent is dependent on the quality of the learnt state space policy (π_s) and the forward dynamics model (f_θ). In order to evaluate the influence of the FDM learning strategy on the overall performance of the system, an ablation study with varying FDM learning schemes and including the use of the actual FDM model was carried out. This study involves learning both the state space policy and forward dynamics models in the original environments. The independent variable of this study is the strategy to obtain the FDM with the following cases:

1. Using the actual FDM (not learning it).
2. Learning the FDM online, without the initial phase of Algorithm 6.
3. Learning the FDM offline, with the exploratory policy π_{rand} in the initial phase.

5-4 Experiment Setup for Interactive Policy learning

In Interactive TaSPL experiments, human teachers who are non-experts are used to train the agent. But for the purpose of an exhaustive evaluation of the learnt policy, we used oracles based on a closed form policy. This is done so that the experiments are not affected by human factors, (like mental fatigue alertness) and thus providing similar feedback conditions to both algorithms used in the comparison.

The oracle is used as a teacher to provide the desired state-space modification information as a non-expert human teacher would do. We ran 5 sets of experiments with non-expert human participants, the trend in human input provided for training the agent to successfully execute the task was used as a reference for the amount of feedback given by the oracle to mimic human effort put into providing feedback. The number of episodes where oracle provided feedback was provided was progressively reduced until the agent was able to learn the policy with the least amount of feedback. The amount of feedback given per episode by the oracle was determined based on an average input given by a human demonstrator over 5 trials.

We evaluate and compare TaSPL to TIPS based on two main criteria: (i) the task performance of the trained agent over time and (ii) the ability of the agent to execute the task it was trained for, but in a new environment where the agent dynamics are different from that of the original environment. The performance of the TaSPL agent is compared with an agent trained using TIPS policy [Jauhri et al., 2020].

- Jauhri et al. [2020] proposed Teaching Imitative Policy in State space (TIPS), an interactive imitation learning algorithm that uses binary corrective feedback in the state space. The demonstrator suggests the next desired state by modifying it to the current state. The agent then converts this suggest change in state space to a change in action space by using the indirect inverse dynamics model. An FDM model is learnt using an exploratory policy, and then uses that model to infer an action that produces the desired state transition information. With the state and the computed action available, a regular state action policy is learnt in a supervised learning manner.

The demonstration for both algorithms was provided using an oracle which followed a deterministic policy [Xiao, 2020].

5-4-1 Ablation study

In order to evaluate the influence of the FDM learning strategy on the overall performance of the system, two ablation studies varying the FDM learning scheme and including the use of the actual FDM model. The two ablation studies cover all cases- of learning only the state space policy, only the forward dynamics model, and both.

Learning in the original environment

This study involves learning both the state space policy and forward dynamics model in the original environments. The independent variable of this study is the strategy to obtain the FDM with the following cases:

1. Using the actual FDM (not learning it).
2. Learning the FDM online, without the initial phase of Algorithm 6.
3. Learning the FDM offline, with the exploratory policy π_{rand} in the initial phase.

The first case works as the reference for evaluating how is the progress of only learning the state space policy, i.e., it measures how long does it take to shape the state space policy since there is no influence of the low performance of forward dynamics model. The other two cases show actually how is the convergence of learning both models either simultaneously as in the second case, or sequentially as in the third case.

Learning in a modified environment

In this study, a learnt state space policy is reused in a modified environment, without additional training, while learning only the forward dynamics model. Two independent variables are considered in this study, one is the initialization of the FDM which could be either a complete random set of parameters or the FDM parameters obtained while learning the policy in the original environment. The other variable is the sampling/update strategy, which could be either learning offline or online, as in the previous study. It also includes the use of the actual FDM as a reference that works for measuring the performance of the actual state space policy, and in order to know what is the upper bound performance of the study. Therefore, there are four possible cases that were considered in this study, along with the one using the actual FDM, as listed below.

1. Using the actual FDM (not learning it).
2. Learning a new FDM from scratch online, without the initial phase of Algorithm 6.
3. Learning a new FDM from scratch offline, with the exploratory policy π_{rand} in the initial phase.

4. Learning the FDM online, without the initial phase of Algorithm 6, but initializing it with the learnt FDM of the original environment.

5. Learning the FDM offline, with the exploratory policy π_{rand} in the initial phase, but initializing it with the learnt FDM of the original environment.

In the first study, the cases of learning only the state space policy, and learning both together are evaluated, while in the second study the remaining case of learning only the forward dynamics model is evaluated considering the main possible variants.

5-5 Validation

The proposed method was validated in two different manipulation tasks, namely 'car balancing' and 'object pulling' with a real robot arm KUKA iiwa. The tasks were performed using a Cartesian position controller. ROS (Robot Operating System) is used as middleware to interface the robot using the iiwa stack. Optitrack camera system, an optical marker tracking system is used to track the position of the objects.

To teach these tasks, the teacher uses a joystick as an interface, for providing feedback in the form of correction of the current observation. The IIDM takes the desired transition and finds the corresponding action that moves the robot end-effector accordingly.

5-5-1 Car Balance Task

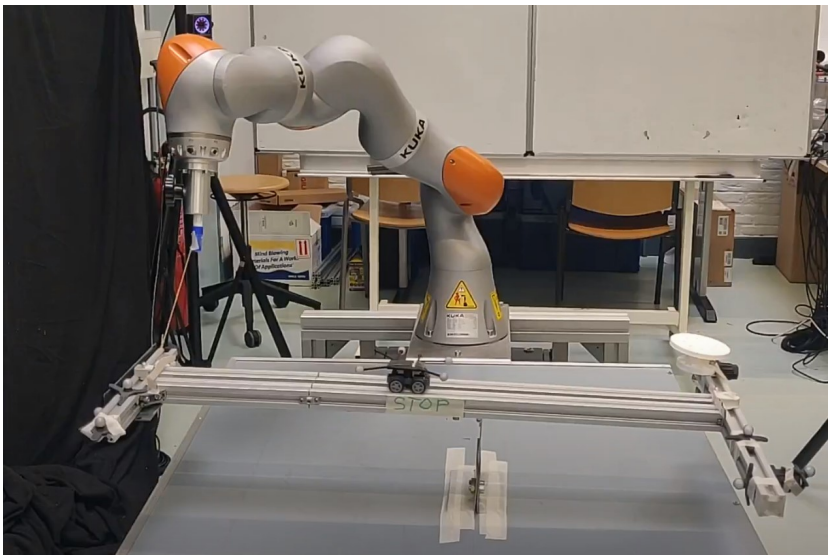


Figure 5-2: The robotic Balance task using a KUKA iiwa 7 robot. The task is to balance the car at the middle of the track.

In this task (fig 5-2), a car is placed on a track which is pivoted at the center, allowing it to tilt around the axis, and moving the car along the track depending on its inclination. The robot tilts the track by pulling a thread attached to one end of it. The objective is to

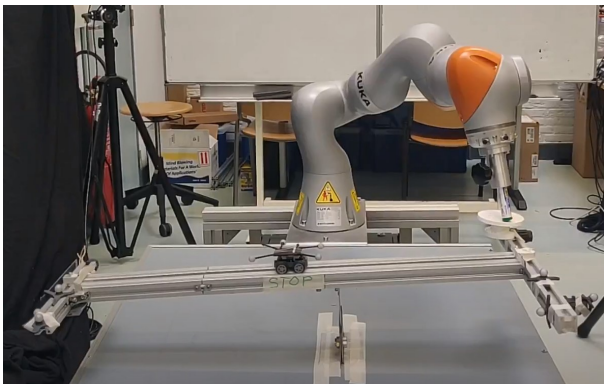


Figure 5-3: Modified environment to test state space policy: Balancing the car by pushing the lever on the opposite side.

place the car at the center of the track. The state space of the environment $s_t = [x\ \dot{x}\ \theta\ \dot{\theta}]$, i.e. the position, and velocity of the car, and the angle, angular velocity of the track. The action space of the robot is limited to discrete actions in the vertical axis.

To teach the task, feedback is provided in the position and velocity of the car. The agent learns the policy in partial state space i.e. $\hat{o_{t+1}} = \pi(o_t)$ where $o_t = [x, \dot{x}]$. The demonstrator provides left/right signals that should accordingly move the end-effector in the z axis. To enable this, a forward dynamics model within the entire state space is learnt. The agent then selects the action that minimises the difference in partial space i.e.

$$a_t^{des} = \underset{a}{\operatorname{argmin}} \left\| w.\hat{o_t} - f_\theta(s_t, a) \right\|, \text{where } w = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{5-1}$$

w is weighting matrix to mask the state the policy is learnt in. This also has the as added advantage of prioritizing having the track leveled and static, i.e., a state that would not move the car to any side, but at the same time a state that could produce a movement to any side with just a small change, thus, it is easy to quickly compensate the car position if required.

To evaluate the task, we define a reward function to be used as a performance metric. The function is inspired by the MountainCar task and consists of a negative reward at every time-step until the car is centered on the track with an almost zero velocity. The learnt observation state space policy π_o is tested in a modified environment, wherein the robot instead of pulling the track from one end, pushes down the other end of the track, as shown in figure 5-3.

This setting changes the way the manipulator interacts with the track, the movements of the robot have an opposite effect on the movement of the track. Another change of the environment was tested with the change of the pivot to a point more distant from the end of the track wherein the robot is attached as shown in figure 5-4.

This requires the robot to perform larger actions.

This experiment is run to validate the application of TaSPL on a robotic task, comparisons are not made with other learning methods.

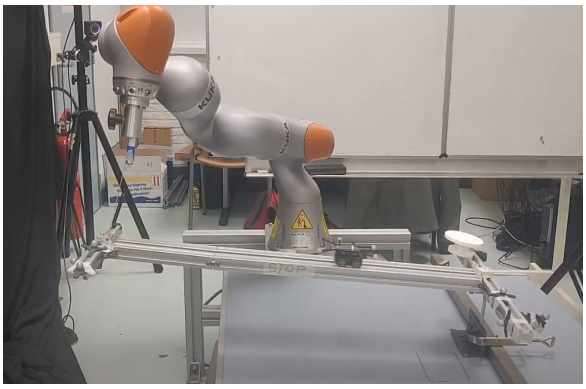


Figure 5-4: Modified environment to test state space policy: The pivot point is moved to one end of the track, larger actions are required to move the car on the track.

5-5-2 Object pulling task

In this task (fig 5-5), a puck is tied to the end-effector by a string. The objective is to move the end effector in such a way that the puck being pulled traces a path around some obstacles. The state space of the environment is $s_t = [x_p \ y_p \ x_r \ y_r]$, i.e., $[x_r \ y_r]$ the coordinates of the puck, and $[x_r \ y_r]$ the coordinates of the robot's end-effector. The action space of the robot is limited to discrete changes in the XY plane but with different magnitudes.

To teach the task, feedback is provided in the position and velocity of the car. The agent learns the policy in partial state space i.e. $\hat{o_{t+1}} = \pi(o_t)$ where $o_t = [x, \dot{x}]$. The demonstrator provides left/right signals that should accordingly move the end-effector in the z axis. To enable this, a forward dynamics model within the entire state space is learnt. The agent then selects the action that minimises the difference in partial space i.e.

$$a_t^{des} = \underset{a}{\operatorname{argmin}} \left\| w.\hat{o_t} - f_\theta(s_t, a) \right\|, \text{where } w = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{5-2}$$

w is weighting matrix to mask the state the policy is learnt in.

To evaluate the performance of the policies in this task, the trajectory of the puck is compared with a reference trajectory of the object going from the starting to the target point, while avoiding collisions with some obstacles. The executed trajectory is compared with the reference trajectory using the Hausdorff distance Huttenlocher et al. [1993].

The learnt state space policy π_o is tested in an environment with different dynamics, by changing the length of the string attaching the end-effector to the puck. With this change, the robot has to perform a different and wider trajectory which is not necessarily proportional to the one executed in the original environment, rather a nonlinear transformation of it.

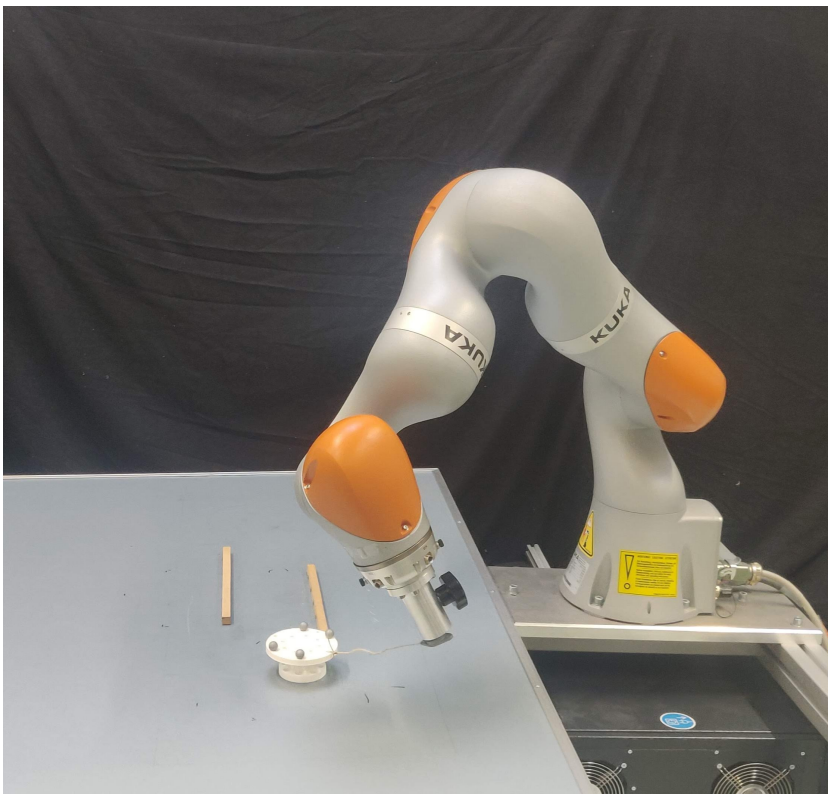


Figure 5-5: Experiment setup for rope drawing task on the KUKA robot. The task is to move the end effector in such a way that the puck traces a path around obstacles

Chapter 6

Results

In this section, we provide the results from our experiments. We show the performance of the agents trained by the two TaSPL algorithms, whilst comparing them against the baseline imitation learning techniques listed in the previous chapter. The results show the advantage provided by the proposed method in learning the policy in state space when the system dynamics changes. We validated TaSPL with real robot settings by computing the performance obtained during the learning process of two manipulation tasks.

The results from offline TaSPL in the OpenAI environments are presented in section 6-1 followed by the results of interactive TaSPL in section 6-2. For each of the proposed algorithm, results of the comparison of TaSPL with the baseline IL algorithm using the simulated environments are presented, followed by the results of the generalization to the change of the dynamics, and then the ablation study regarding the learning of the FDM. Sections 6-1 and 6-2 only show the most relevant results, for all the results in all modified environments check Appendix A

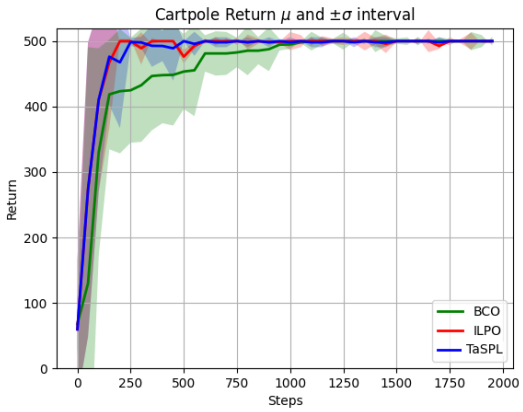
Section 6-3 shows the results of validating the online method with a real robot.

6-1 Offline TaSPL

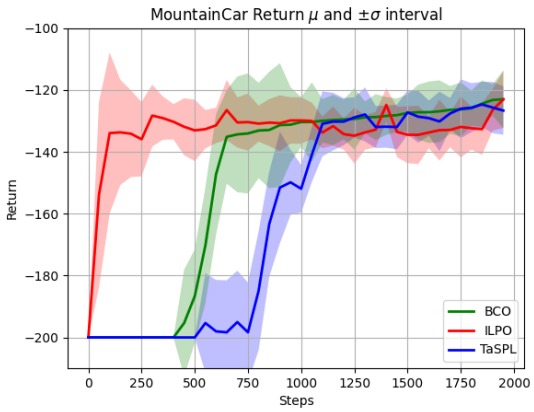
6-1-1 Performance in original environment

Figure 6-1 shows the average return obtained by an agent trained in the original environment using the offline TaSPL for each of the simulation tasks (averaged over 20 experiments) compared against the baselines defined in the previous section IL techniques, namely, ILPO and BCO.

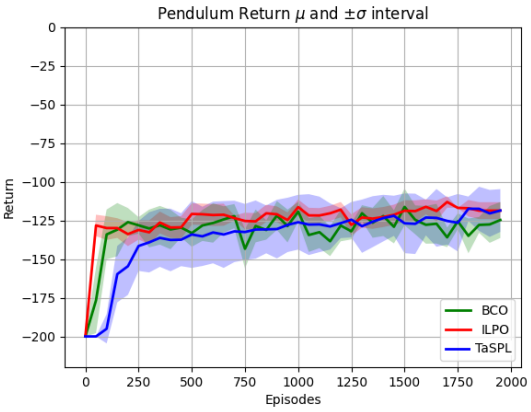
We can observe that the performance of the agent trained by TaSPL is at par with the performance of BCO and ILPO in terms of the cumulative return obtained by the agent at the end of the 2000 steps. Compared to BCO and ILPO, TaSPL takes a few more steps to learn an optimal policy, as the agent has to lean FDM in an online manner. The performance



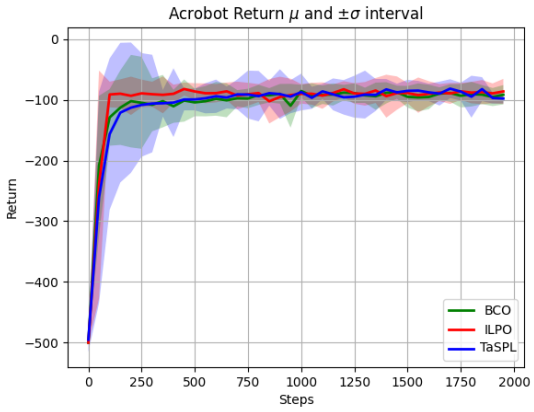
(a) Average return in Cartpole environment



(b) Average return in MountainCar environment



(c) Average return in Pendulum environment



(d) Average return in Acrobot environment

Figure 6-1: Average performance of TaSPL, ILPO and BCO agents over training steps when trained by oracles in the original unmodified environments

of the agent depends on its ability to learn the forward dynamics model in an online manner. This can be seen clearly in the MountainCar environment (fig A-5b), the rewards obtained by the agent following TaSPL are low for nearly 500 steps as the agent is stuck in the valley between the two mountains, before it learned the dynamics of the agent at the mountain tops.

6-1-2 Performance in modified environment

Physical properties (like mass, length, force applied) of the environment were modified such that the dynamics of the agent become different from the original environment. Figure 6-2 shows the average return obtained by an agent in the worst-case scenario where the applied force is negative. By learning the FDM in an online manner the agent is able to adapt to the highly modified environment. The learning progress of the agent trained in TaSPL and ILPO is the same as that of the agent executing the task in the original unmodified environment, as the policy learnt in state space and latent action space. The BCO agent takes longer to learn in the modified environment, as the agent has to carryout behavior cloning on the new state action pairs obtained while following the original policy.

Sections 6-1-2 shows only the worst-case scenario of negative force, for plots of the agents return in other modified environments check Appendix A

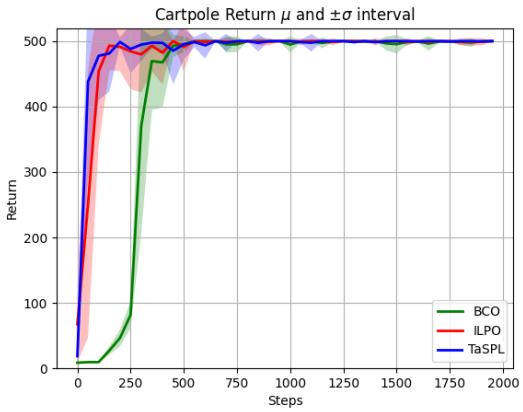
6-1-3 Ablation study

The results of the ablation study to test the efficacy of the state space policy using FDMs learnt from different techniques. The agents used the policy from the original environment but the policy was evaluated in the modified environment (here, force applied is negative). Figure 6-3 shows agents that used the true known FDM were able to execute the task immediately with no further interactions in the modified environment. In the MountainCar environment (Fig 6-3b), the agent was unable to learn an effective forward dynamics model following a random policy due to the nature of the environment. However, if agent combines offline model learning with online model learning, the agent is able to learn the forward dynamics model more effectively following the learnt policy and exploring new state spaces.

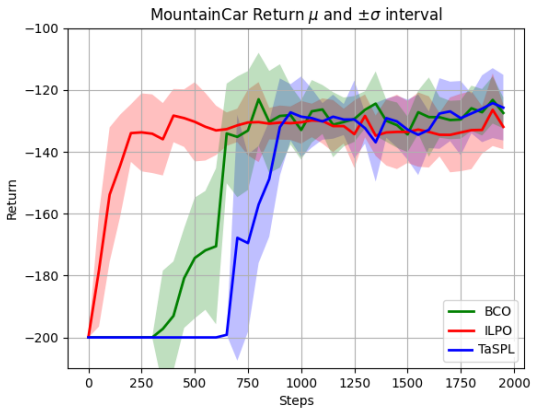
6-2 Interactive TaSPL

The comparisons of teaching a policy interactively with TaSPL and TIPS are plotted in the figures 6-4 to 6-7, wherein the learning curves are showing the return obtained by the current policy throughout the time steps of the learning process. For each problem, it is shown the learning curves in the original environment (in the left), and the curves of learning in one of the multiple tested modified environments (in the right), which is the case of the environment with actions being multiplied by -1, which we call 'Negative Dynamics'.

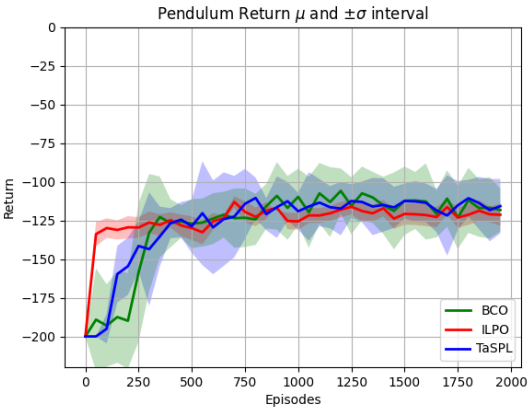
In general, when learning the policies in the original environment, both algorithms show a very similar convergence, and similar human input. The cases of Cartpole, and Lunarlander required slightly more human input with TaSPL. Nevertheless, there is a considerable advantage of TaSPL when the dynamics change and the high level policy is reused, since there is



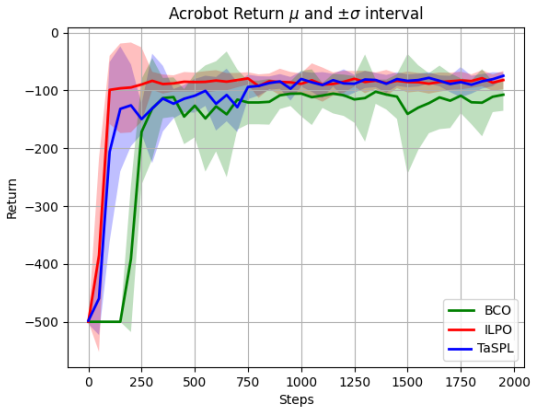
(a) Average return in Cartpole environment



(b) Average return in MountainCar environment

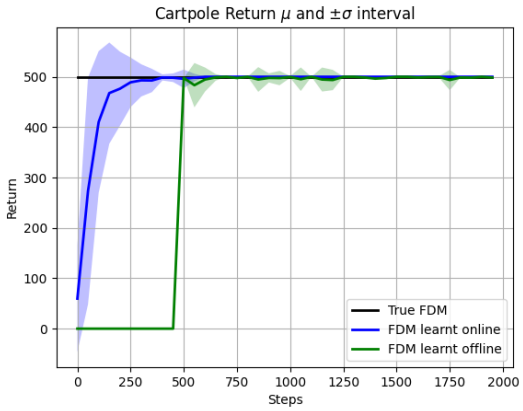


(c) Average return in Pendulum environment

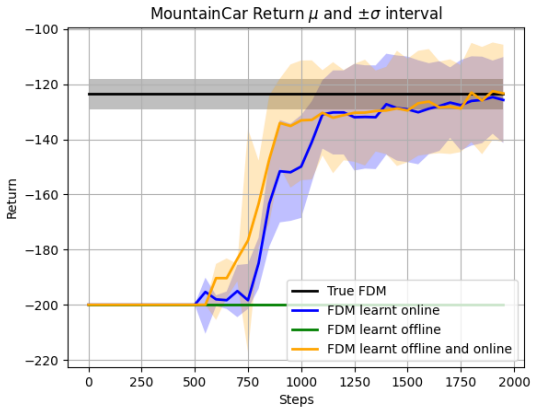


(d) Average return in Acrobot environment

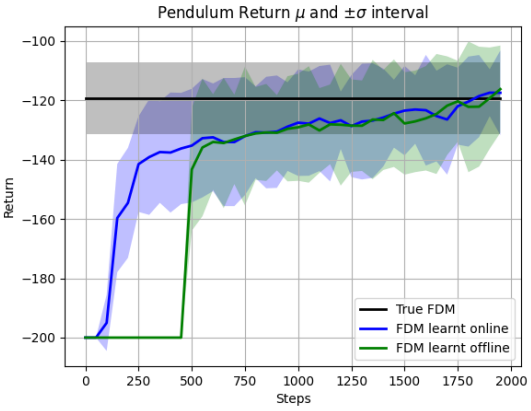
Figure 6-2: Average performance of TaSPL, ILPO and BCO agents in modified environments (force applied is negative) while following the policy learnt in the original environment



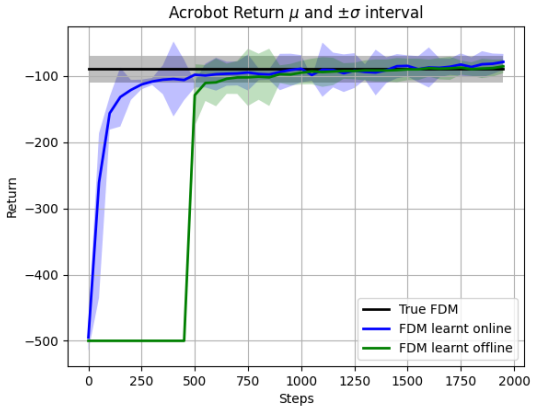
(a) Average return in Cartpole environment



(b) Average return in MountainCar environment



(c) Average return in Pendulum environment



(d) Average return in Acrobot environment

Figure 6-3: Average performance of TaSPL, using True FDM, FDM learnt offline following a random policy and FDM learnt online in modified environments (force applied is negative) while following the policy learnt in the original environment. Returns of the FDM from random policy are offset by the number of training samples taken to train the dynamics model (f_θ)

no need to modify the high level policies with further human interventions, while with TIPS it is still needed the human input as learning in the original environment.

With TaSPL, it is just needed to sample transitions of the environment during 500 time steps with π_{rand} in the initial phase of the algorithm, in order to capture the new FDM behavior. Only in the case of the Mountaincar (figure 6-5), the data sampled with π_{rand} is not representative enough to train a model that describes properly the system, therefore, it is required to keep collecting transition samples while following the high level policy. That is why in this problem, with both algorithms, the performance is the lowest possible during the first episodes of training. Although, in the modified environment, TaSPL started having successful executions in half the steps required by TIPS to do so, and still without additional teacher input.

The results of training in the LunarLander environment showed that neither TaSPL nor TIPS learn an optimal policy to land the spacecraft between the flags, however both obtain policies that are able to softly land it.

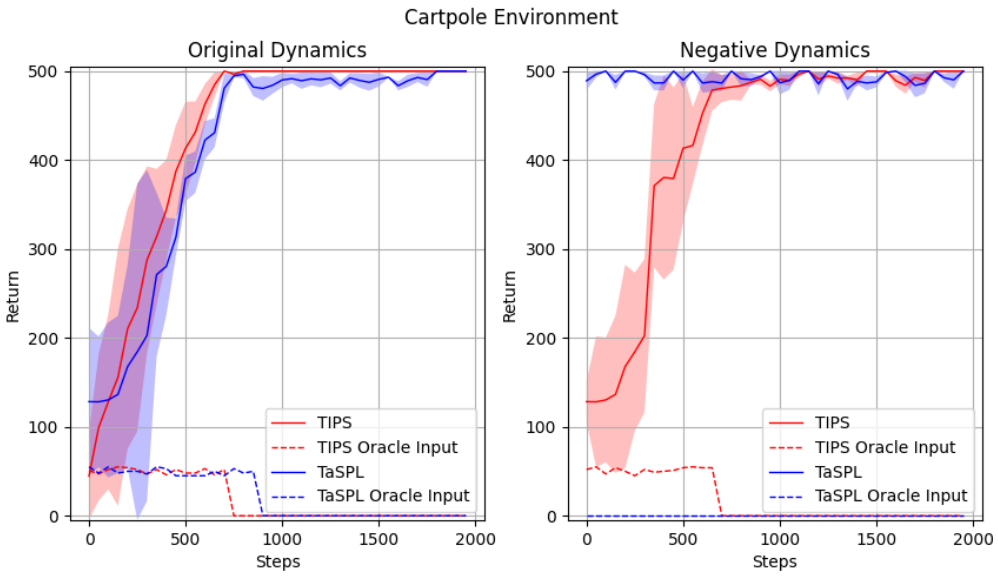


Figure 6-4: Comparison of the average performance of TaSPL and TIPS agents over training steps with the amount of feedback provided by oracles in the original Cartpole environment and the modified environment (where the force is negative)

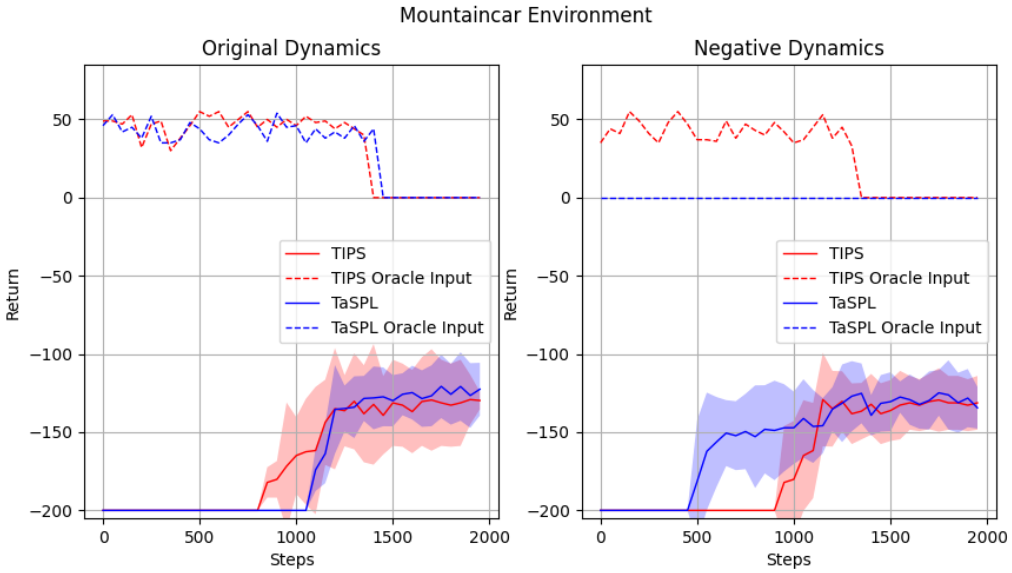


Figure 6-5: Comparison of the average performance of TaSPL and TIPS agents over training steps with the amount of feedback provided by oracles in the original MountainCar environment and the modified environment (where the force is negative)

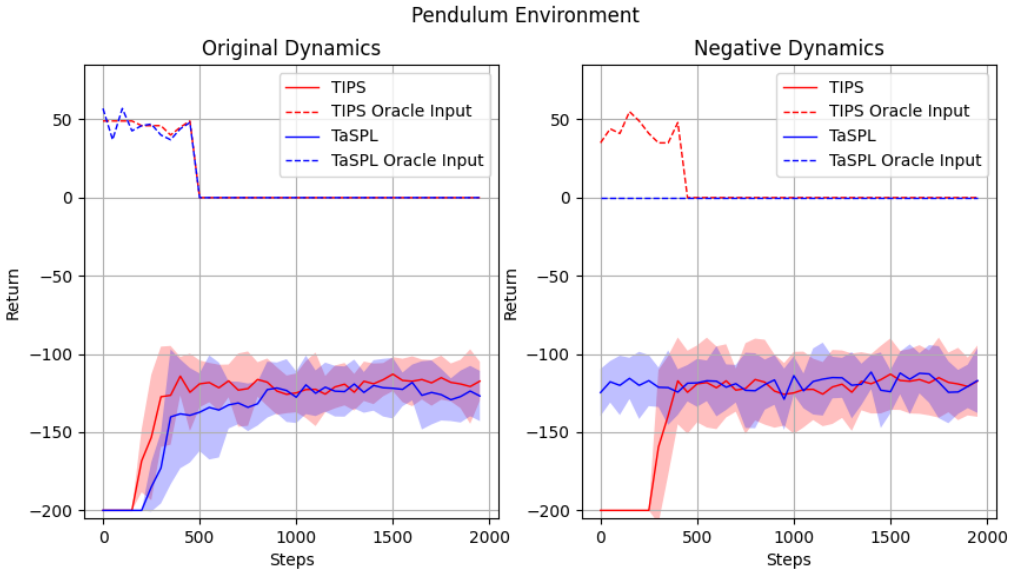


Figure 6-6: Comparison of the average performance of TaSPL and TIPS agents over training steps with the amount of feedback provided by oracles in the original Pendulum environment and the modified environment (where the force is negative)

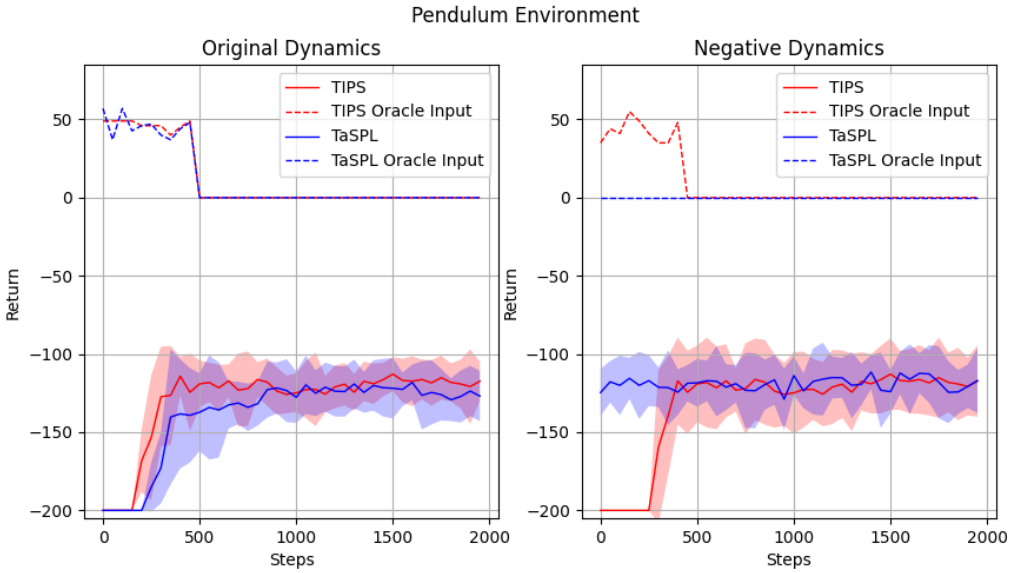


Figure 6-7: Comparison of the average performance of TaSPL and TIPS agents over training steps with the amount of feedback provided by oracles in the original Acrobot environment and the modified environment (where the force is negative)

6-2-1 Ablation study

For these studies, the Lunarlander environment is not considered because in the previous experiments with both evaluated algorithms, the obtained performances were relatively low and the task was partially fulfilled.

Learning in the original environment

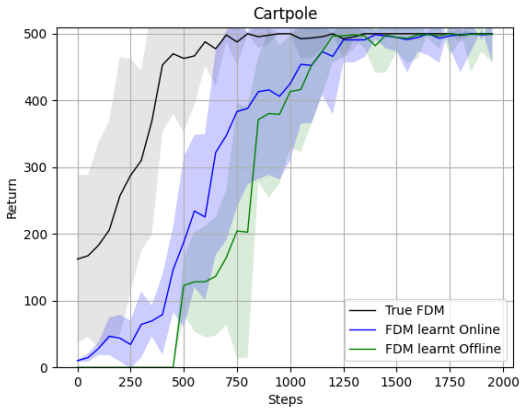
The results of these experiments are depicted in figure 6-8. It could be seen that learning both the dynamics model and the policy makes the process slower than only needing to learn the task space policy. In general for most of the cases, the final performance of the learnt FDMs is at the same level as the actual FDMs, since they allow to follow the desired transitions and obtain a similar average return with respect to the baseline based on the true FDM.

In the Cartpole problem, we could see that not knowing the FDM could take double the required time for convergence, with respect to training based on a known FDM. Learning the FDM online or offline obtains the convergence almost at the same time, although in this problem, the offline approach has the advantage that the teacher had to participate during around 40% less time, because the first 500 time steps are dedicated to the initial model learning phase.

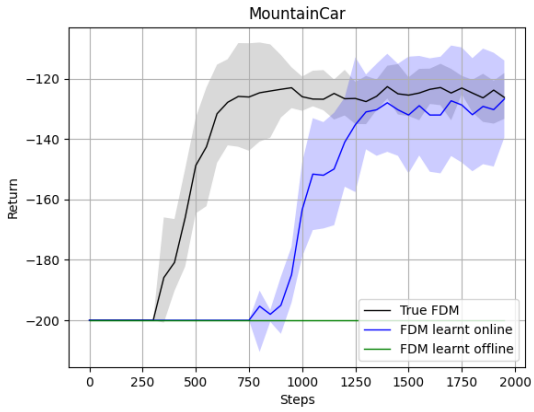
For the problem of the MountainCar, the results are rather different, because learning offline was not successful. This is because for this problem, the exploratory policy π_{ramd} is not enough to obtain representative samples from the entire environment, i.e., the random exploration never makes the car to move out of the bottom of the valley, therefore the learnt model only describes correctly one area of the state space.

The results of the pendulum show a rather fast improvement at the beginning, although the three cases converged almost at the same time. With the offline approach there is an steeper improvement (when the interactive phase starts after the first 500 time steps), however, the online case has a considerable improvement before the 500 steps, which could be convenient in situations wherein a good but suboptimal policy is acceptable to be rolled out.

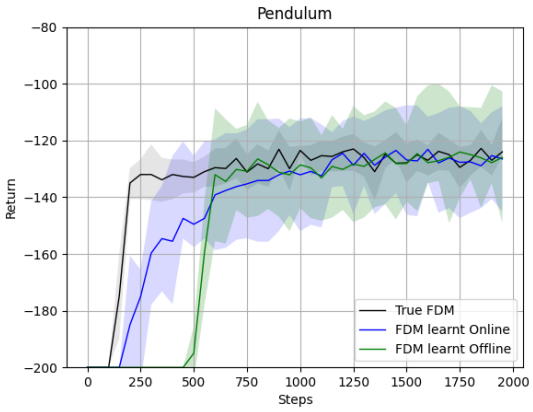
From these results, it is observed that learning each of the models has a similar contribution to the time required for the convergence, and none of them represents a big bottleneck in this regard. Although of course it is always needed the existence of an FDM in order to be able to train a task space policy, whereas the other way around is not always a limitation, i.e., it is not always required a task space policy to be able to train a FDM as in the Cartpole and Pendulum problems, only the task space policy π_o was required for learning the FDM in the MountainCar problem since it helped to obtain better samples of the system that the exploratory policy π_{rand}.



(a) Average return in Cartpole environment



(b) Average return in Mountain Car environment



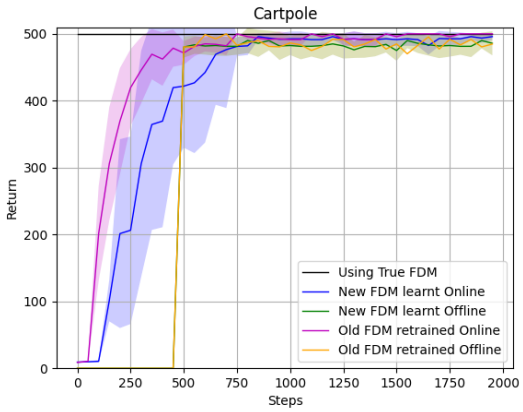
(c) Average return in Pendulum environment

Figure 6-8: Average performance of TaSPL, using True FDM, FDM learnt offline following a random policy and FDM learnt in an online manner

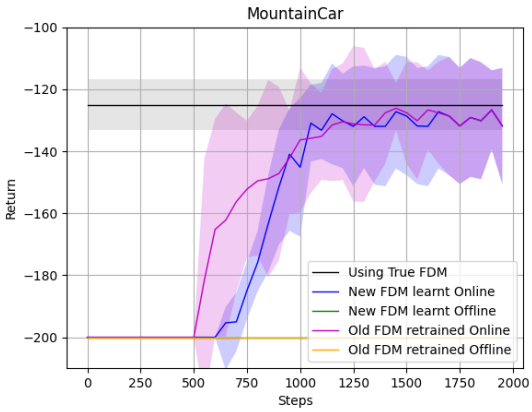
Learning in a modified environment

This study evaluated the remaining case of learning only the FDM. As expected from the observations of the previous study, the results in figures 6-9 showed that it takes less time to only learn the FDM that obtains the desired transitions requested by the task space policy.

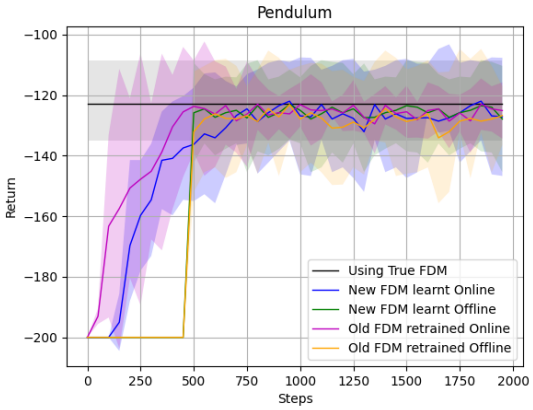
In general, the online learning approach seems to be more beneficial to have relatively good performance with few time steps, and reusing the model of the original dynamics for initializing the new FDM seems to obtain an advantage in the velocity of convergence, since part of the features extraction is reused from the original model and is not fully learnt from scratch again.



(a) Average return in Cartpole environment



(b) Average return in Mountain Car environment



(c) Average return in Pendulum environment

Figure 6-9: Average performance of TaSPL, using FDM learnt from different sources explained in section 5-4-1

6-3 Robotic Task

6-3-1 Car Balancing task

In this section we provide the final results from experiments on the real system (Figure 5-2). Since the objective is to validate the application of TaSPL, comparisons with other learning techniques are not made. To teach the task, the teacher provides corrective feedback in terms of the desired position and velocity of the car. Based on some initial trials, a correction magnitude (i.e. error constant e) of 5 centimeters is found to work well for the task. To limit the angle of tilt in the track, the range of the end effector is limited in the z axis between 15 to 65 cm. To avoid the car rolling off the track, mechanical limits were added on either end of the track.

Each episode is 50 seconds long, after which the robot is reset. The initial position of the car is randomized by making the agent learn a state policy over the entire state space. The agent performance and demonstrator feedback rate over learning episodes can be seen in Figure 6-10. The agent successfully learns to reliably perform the task after 60 episodes of training. The amount of feedback provided by the teacher reduces over time as the agent performs better and only some fine-tuning of the behavior is needed after 50 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 tilt the track to a small angle to get the car moving in the desired direction with a low velocity, and tilt in the opposite direction to stop the car from moving. This behavior is successfully imitated by the agent.

Figure 6-10: Validation task: Learning curves and demonstrator feedback rates for the Balance experiment

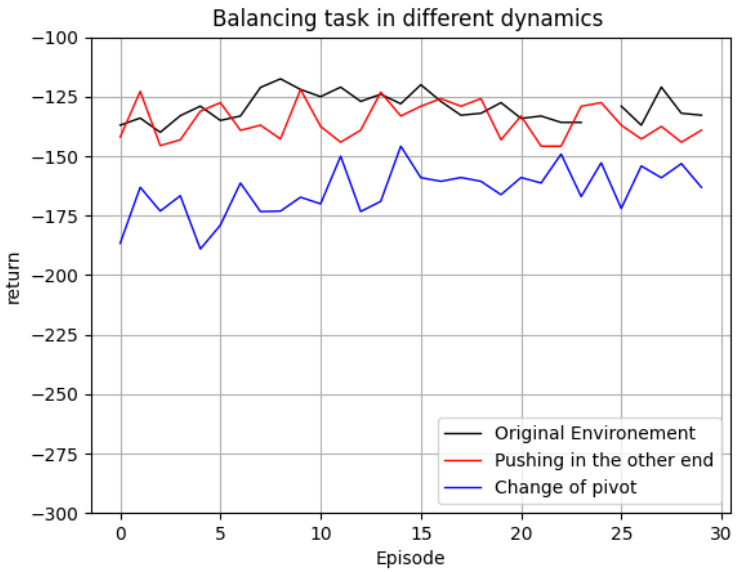


Figure 6-11: Cumulative return obtained in different dynamics setups- Original environment, Pushing in the other end, Change of pivot

When changing the configuration of the environment as described in Section 5-5-1, the learnt policy was successful to fulfill the objective of the task. It was enough the exploratory policy for learning the new dynamics in order to reuse the high level policy. It is possible to see that when the robot has to push (instead of pulling) on the other side of the track, the performance is rather similar to the one of the policies in the original environment. However, when the pivot is moved to the end of the track, the task is also accomplished but with a lower performance, this is due to the fact that in this setup the robot had to execute larger movements in order to obtain the same changes in the inclination of the track, however, the velocity of the robot was limited.

6-3-2 Object pulling task

In this task the high level policy was taught in the setup with a short rope for pulling the object, and later this policy was successfully validated with a longer rope without additional human input. From the feedback given by the teacher, the agent is has learnt to trace a path in the XY plane within 20 episodes (fig 6-11). The agent successfully learns to trace a path with a short string. Further feedback is provided for the agent to learn to avoid the obstacle. The amount of feedback reduces once the agent has learnt the basic path. Additional feedback is given to fine-tuned the behavior to avoid obstacles.

Figure 6-13 shows the path traced by the puck and the end effector when using a short thread and a long thread to attach the puck and the end effector. The path traced by the puck in both the cases is very similar, however the path traced by the end effector is very different due to modified dynamics from a longer rope attaching the two. The black dotted line is an object reference trajectory obtained with teleoperation. In red, it is shown the trajectories

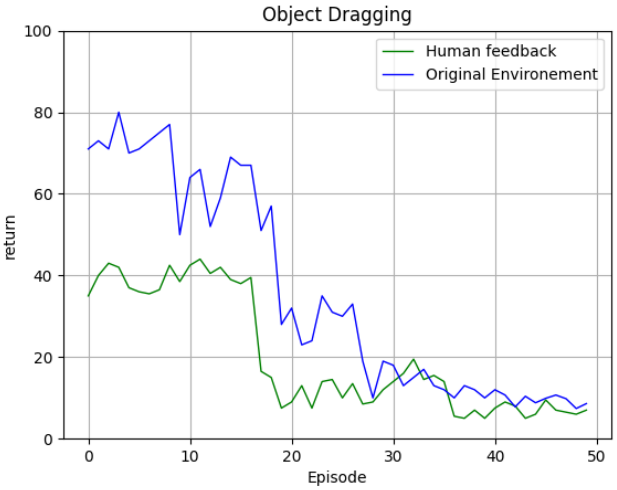


Figure 6-12: Validation task: Learning curves and demonstrator feedback rates for the Object Pulling task

obtained in the original environment, and in blue the ones with the modified dynamics (long rope).

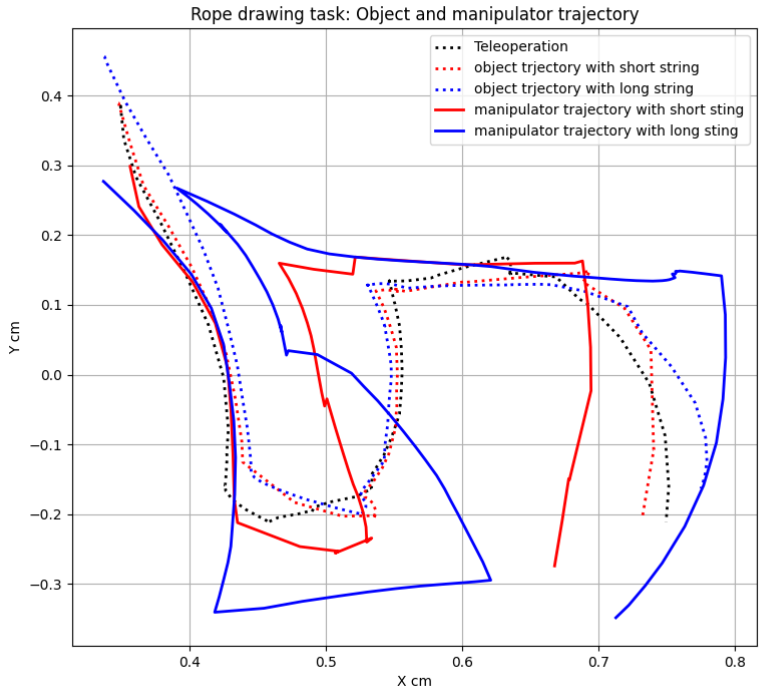


Figure 6-13: Validation task: Path traced by the puck and the end effector when using a short thread and a long thread

6-4 Discussion

In this chapter, we provided the results from our experiments for the evaluation of offline and interactive TaSPL. Through our comparison of Offline TaSPL with ILPO and BCO techniques, we have shown that agents trained using TaSPL achieve similar performance with proven baseline IL techniques in both, the original environment and the modified environment. This illustrates the viability of TaSPL as an Imitation from Observation learning technique suitable for scenarios where the model dynamics changes and state only demonstrative data is available.

Through our comparison of Interactive TaSPL with TIPS, we have shown that we can not only learn a policy from state-space feedback but also a policy independent of the model dynamics. Both these algorithms achieve the same performance in terms of cumulative return over time as well as demonstration effort required to learn the task.

TaSPL learns a generalised state space policy $\pi(s_{t+1}|s_t)$, as opposed to learning a policy in state-action space $\pi(a_t|s_t)$. This state space policy enables the agent to execute the task when the environment dynamics is changed from the original environment, without the need of additional interventions from the human teacher (required in TIPS), therefore, reducing their workload. Thus, the merits of learning a generalised state-space policy are clear.

In the experiment carried out on the KUKA iiwa robot to learn the balancing and object pulling task, we have shown that the agent is able to successfully learn a generalised policy in a reasonable amount of time through state-space feedback (end-effector position). This policy is validated by executing the task in new modified environments.

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

Chapter 7

Conclusion

Imitation Learning (IL) methods are an intuitive approach for programming robot behaviors. For Human teachers, it is much easier to transfer their knowledge through demonstrating what a robot has to do than articulating it algorithmically. Interactive IL or IL with humans in the loop methods have recently become popular in the robot learning community, especially because they overcome the limitations of standard IL approaches. These methods iteratively keep collecting information from the teachers, while rolling out the current learning policy.

Although interactive IL methods have been demonstrated to obtain more robust policies than with standard IL, most of them obtain explicit policies mapping from states to actions, that cannot be reused to fulfill the task objectives when the dynamics of the environment change, e.g. due to wear of the system, change of the manipulator, the tools, or the objects to be manipulated. Therefore, whenever the learned policy is not valid anymore, these methods require again the intervention of the teacher to tune the policy for the new situation, i.e. requiring a higher workload from the teachers who might not be always available.We propose a method titled Task Space Policy Learning (TaSPL), a novel technique that learns a generalised task state space policy $\pi(s_{t+1}|s_t)$, as opposed to learning a policy in state-action space $\pi(a_t|s_t)$.

The method was tested and compared to other imitation learning methods, for various control tasks of the OpenAI Gym toolkit in their original environment. Both TaSPL and TIPS provide a simple binary corrective feedback mechanism (in the form of increase/decrease signals) in state-space through which demonstrators can train agents as it is more intuitive for human teachers to teach in terms of change of state than in terms of exact action to be executed.

This state space policy enables the agent to execute the task when the dynamics of the environment is changed, without the need for additional interventions from the human teacher, therefore, reducing their workload. The obtained task space policies were also tested in the modified environments, showing that this method can be used to learn a generalized policy to obtain imitation policies with the benefits of interactive IL methods, while also being able to generalize that knowledge to several varied conditions unseen during the teacher interventions.

TaSPL needs an accurate forward dynamics model to be able to compute actions needs to execute the desired state transition. TaSPL suffers the same limitations of TIPS i.e. accurate

dynamics model learning and action computation in high dimensional or continuous action spaces. These limitaions can be mitigated if an accurate dynamics model is available or can be learnt offline from data in a supervised learning manner.

However, there are some considerations that can limit the performance of the TaSPL algorithm. If the dynamics of the environment changes to a large extent, the agent might have to follow a different strategy/ trajectory to achieve the objective. The high level policy learnt may not be optimal to execute the task. Additional feedback may be required to make the learnt policy optimal in the new environment. In the relatively small dimensional spaces in our experiments, the action computation is carryout in every step and is inexpensive, however this does, not hold for higher dimensional spaces where a lot of computational power would be required, we could solve this by training a state to action policy simultaneously.

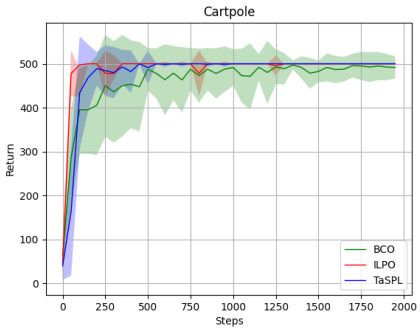
In conclusion, we have successfully demonstrated a method that learns a generalised task state space policy $\pi(s_{t+1}|s_t)$, from interactive corrections in state space or state only demonstrations that can be used to execute the task even when the model dynamics changes without any additional demonstrative effort from the teacher.

Appendix A

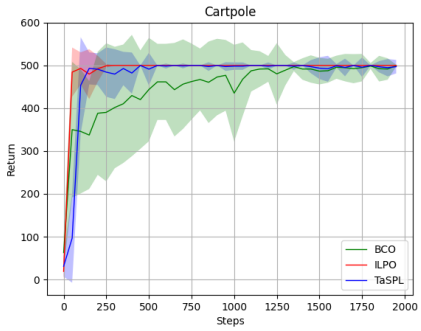
All Plots

A-1 Offline TaSPL

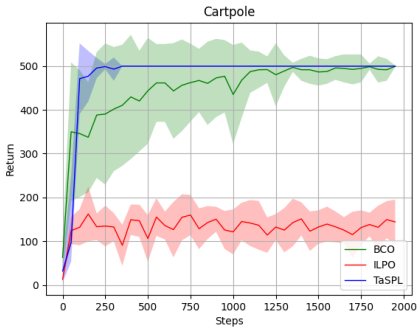
A-1-1 Cartpole Environment Modification



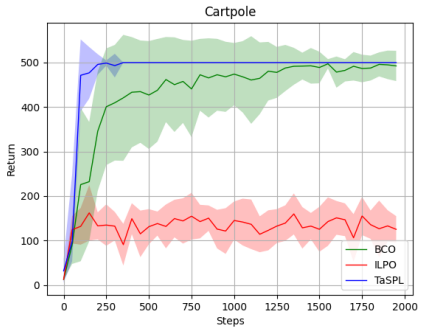
(a) the mass of the pole was halved, $m_p = 0.5$



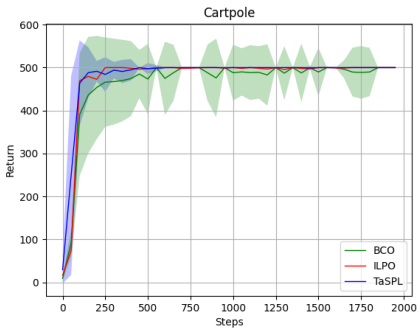
(b) the mass of the pole was double, $m_p = 2$



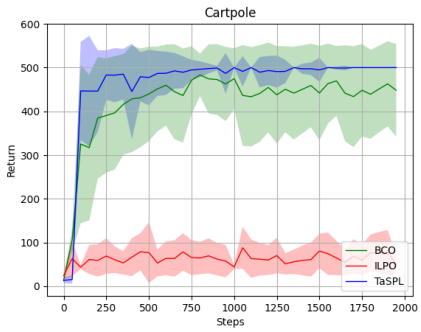
(c) the mass of the cart was halved, $m_c = 0.5$



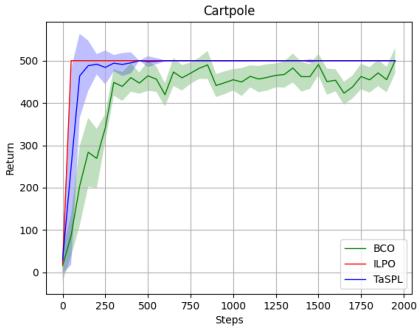
(d) the mass of the cart was double, $m_c = 2$



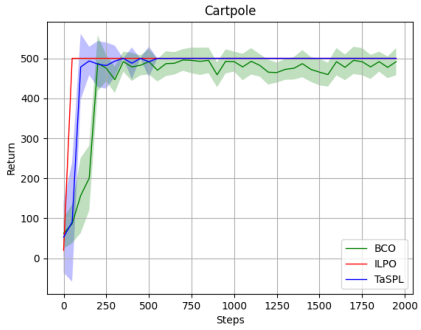
(a) the length of the pole was halved, $l = 0.5$



(b) the length of the pole was double, $l = 2$

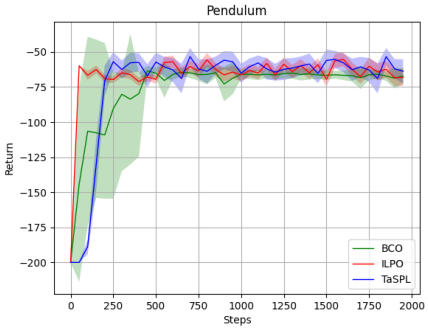


(c) the force applied on the cart was halved, $f = 0.5$

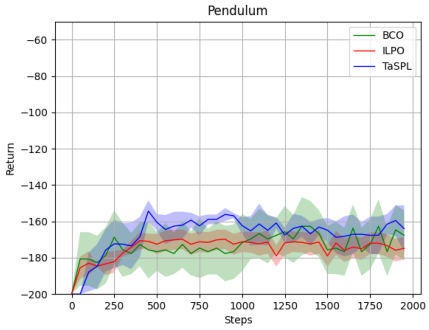


(d) the force applied on the cart was double, $f = 2$

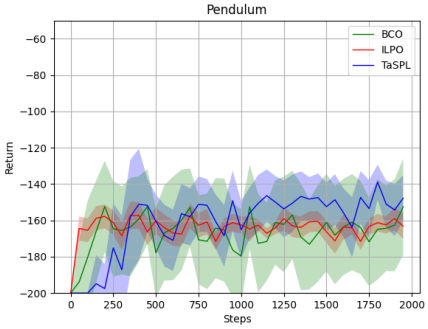
A-1-2 Pendulum Environment Modification



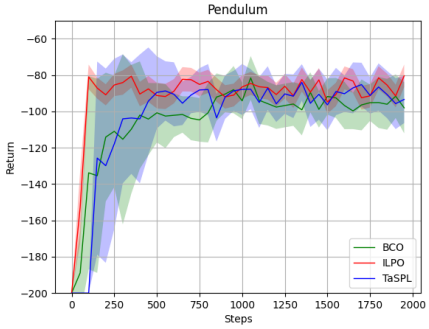
(a) the mass of the pole was reduced, $m_p = 0.75$



(b) the mass of the pole was increased, $m_p = 1.5$

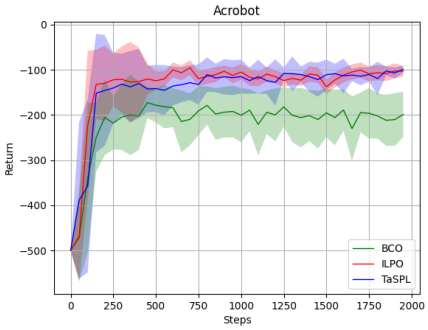


(c) the torque applied on the link was reduced, $f = 0.75$

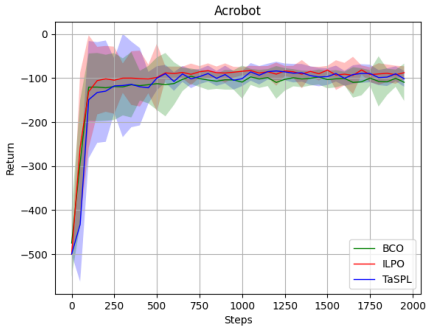


(d) the force applied on the cart was increased, $f = 1.5$

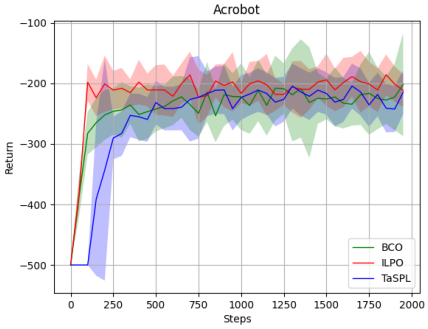
A-1-3 Acrobot Environment Modification



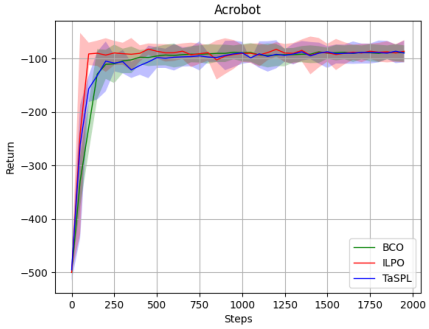
(a) the mass of the links was varied, $m_1 = 0.8,\ m_2 = 1.2$



(b) the mass of the links was varied, $m_1 = 1.2,\ m_2 = 0.8$



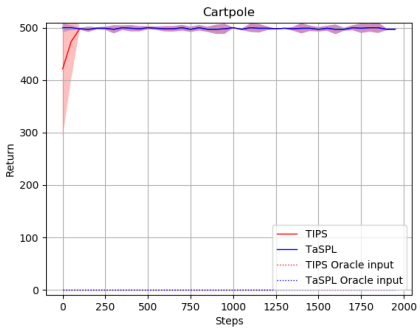
(c) the torque applied on the link was reduced, $f = 0.75$



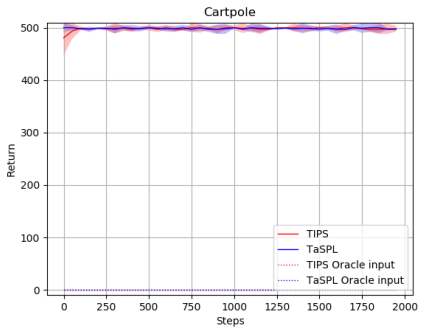
(d) the force applied on the cart was increased, $f = 1.5$

A-2 Interactive TaSPL

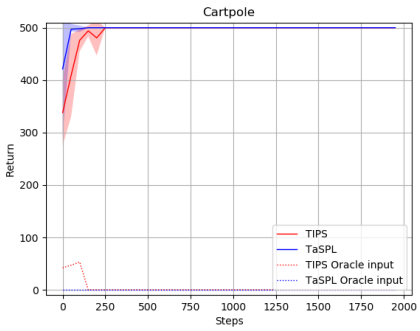
A-2-1 Cartpole Environment Modification



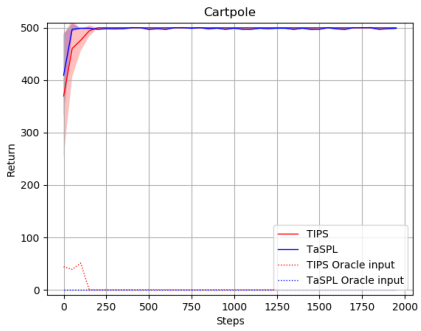
(a) the mass of the pole was halved, $m_p = 0.5$



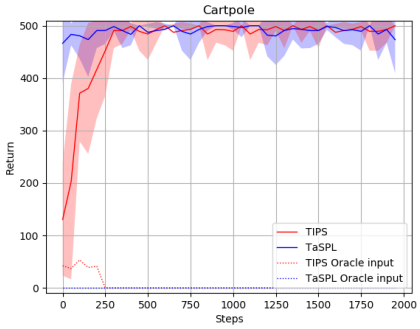
(b) the mass of the pole was double, $m_p = 2$



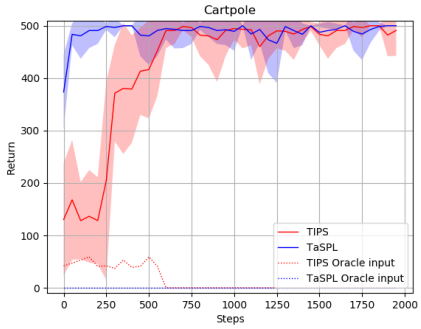
(c) the mass of the cart was halved, $m_c = 0.5$



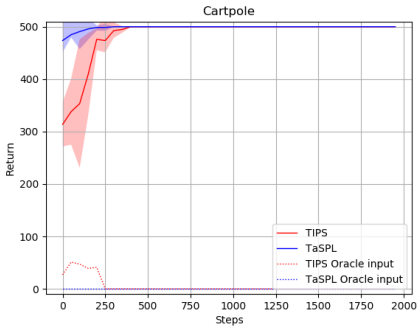
(d) the mass of the cart was double, $m_c = 2$



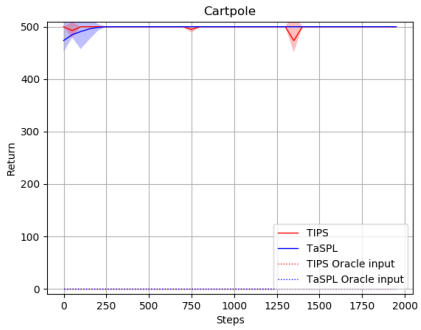
(a) the length of the pole was halved, $l = 0.5$



(b) the length of the pole was double, $l = 2$

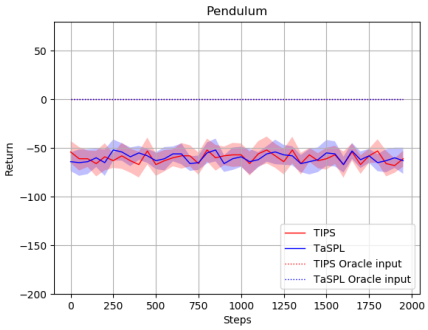


(c) the force applied on the cart was halved, $f = 0.5$

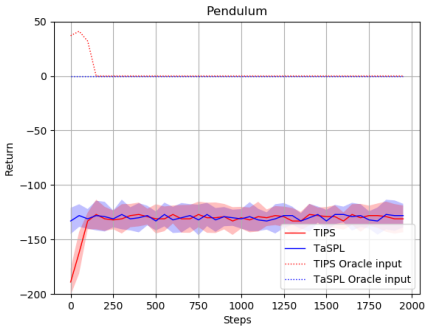


(d) the force applied on the cart was double, $f = 2$

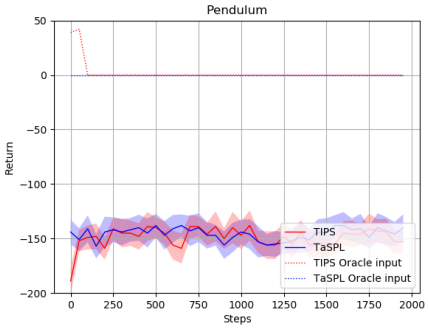
A-2-2 Pendulum Environment Modification



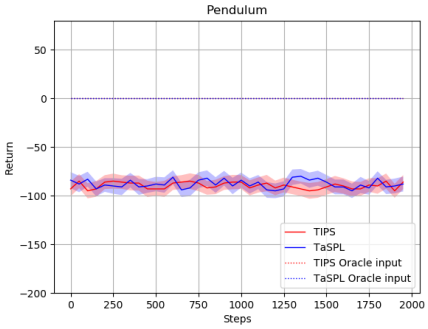
(a) the mass of the pole was reduced, $m_p = 0.75$



(b) the mass of the pole was increased, $m_p = 1.5$

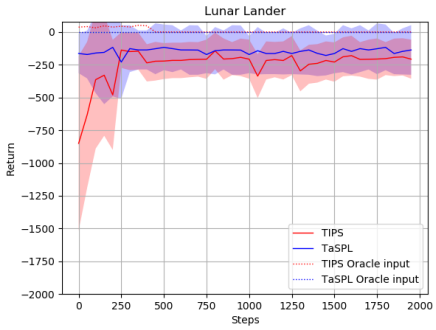


(c) the torque applied on the link was reduced, $f = 0.75$

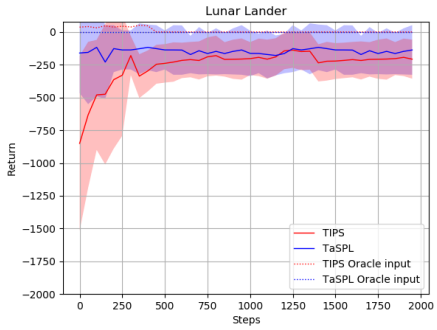


(d) the force applied on the cart was increased, $f = 1.5$

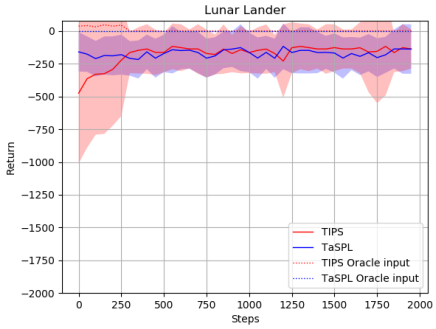
A-2-3 Lunar Lander Environment Modification



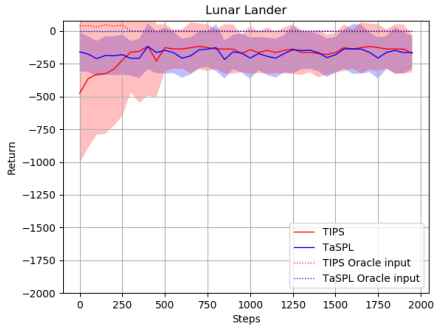
(a) the main engine power in was reduced, $f_{MAIN} = 10$



(b) the main engine power in was increased, $f_{MAIN} = 16$



(c) the side engine power in was reduced, $f_{SIDE} = 0.4$



(d) the side engine power in was increased, $f_{SIDE} = 0.9$

Appendix B

Paper

A research paper presenting the interactive method proposed in this thesis: "Policy Imitation Learning with Online Task-space Interactive Non-expert Guidance (PILOTING)", is provided below.

Learning task space policy from demonstration

Lalith Keerthan[1], Carlos Celemin[1], and Jens Kober[1]

Abstract— **In this paper we propose a method titled "Policy Imitation Learning with Online Task-space Interactive Non-expert Guidance (PILOTING)", a novel technique that uses interactive corrections in the observation space to learn a generalised observation space policy $\pi(o_{t+1}|o_t)$, as opposed to learning a policy in state-action space $\pi(a_t|s_t)$. This observation space policy enables the agent to execute the task when the environment dynamics is changed from the original environment, without the need of additional demonstrative effort from the human teacher. We achieve this by decoupling the objective task into two policies. A high level policy, that describes how the observable states should transition in order to reach the goal of a task, that is learnt through corrective feedback from the teacher. And a low level policy, which is responsible for performing the action that obtains the desired transition. Thus effectively decoupling the task objective from the dynamics of the environment. In case the environment dynamics changes, only the low-level policy has to be relearnt, while the high level policy can be reused.**

The method was tested and compared to other imitation learning methods, for various control tasks of the OpenAI Gym toolkit in their original environment. The obtained policies were also tested in the modified environments, showing that this method can be used to obtain imitation policies with the benefits of interactive IL methods, while also being able to generalize that knowledge to several varied conditions unseen during the teacher interventions. The method was validated in two different tasks with a KUKA iiwa robot manipulator, testing generalization capabilities of the learnt policies.

I. INTRODUCTION

Imitation Learning (IL) methods are an intuitive approach for programming robot behaviors [1], [2]. For Human teachers, it is much easier to transfer their knowledge through demonstrating what a robot has to do than articulating it algorithmically [3]. These machine learning techniques are convenient for programming autonomous robots in order to cope with difficulties like the inability of humans to preprogram a robot for every possible scenario, and/or not having technical skilled users able to modify a complex program of the robot, for instance, in household environments with service robots.

For Learning a policy from a set of recorded demonstrations, there are two main approaches in standard IL methods: i) directly deriving a policy from the data with supervised learning, known as Behavioral Cloning (BC) [4], [5]; ii) using Inverse Reinforcement Learning (IRL) to obtain the objective function of the task which is implicitly described in the demonstrations, then using that learned objective function and Reinforcement Learning (RL) to train an imitation policy [6], [7]. The former family of methods has the advantage of deriving an explicit policy without the additional computationally expensive/data hunger RL process. However, the latter, also known as indirect IL, has the advantage of being able to generalize the obtained knowledge encoded in the objective function, that could be used to learn an imitation policy in environments with different dynamics, or with mismatches between the embodiment of the teacher and the learner, i.e. without the need to record new teacher demonstrations in the changed environment.

Directly deriving a policy after collecting demonstrations has two major limitations: i) the distribution shift, given by the compound error that occurs when the learned policy shifts towards states unseen during the demonstration recording, and it is not able to recover and fulfill the task; ii) the policy performance is at most as good as the demonstrations, therefore, the methods only work for teaching tasks that the teachers are able to perform successfully. Interactive IL or IL with humans in the loop methods have recently become popular in the robot learning community [8], especially because they overcome the mentioned limitations of standard approaches. These methods iteratively keep collecting information from the teachers, while rolling out the current learning policy. These approaches let the teacher to correct the agent when the policy performs wrong actions and/or needs to recover actions to move towards desired states.

Although interactive IL methods have been demonstrated to obtain more robust policies than with standard IL, most of them obtain explicit policies mapping from states to actions, that cannot be reused to fulfill the task objectives when the dynamics of the environment change, e.g. due to wear of the system, change of the manipulator, the tools, or the objects to be manipulated. Therefore, whenever the learned policy is not valid anymore, these methods require again the intervention of the teacher to tune the policy for the new situation, i.e. requiring a higher workload from the teachers who might not be always available.

In this paper, we propose a method titled "Policy Imitation Learning with Online Task-space Interactive Non-expert Guidance (PILOTING)", a novel technique that uses interactive corrections in state space to learn a generalised state space policy $\pi(s_{t+1}|s_t)$, as opposed to learning a policy in state-action space $\pi(a_t|s_t)$. This state space policy enables the agent to execute the task when the environment dynamics is changed from the original environment, without the need of additional interventions from the human teacher, therefore, reducing their workload. The method was tested and compared to other imitation learning methods, for various control tasks of the OpenAI Gym toolkit in their

[1]Lalith Keerthan, Carlos Celemin, and Jens Kober are with the Cognitive Robotics Department, TU Delft, Delft, The Netherlands L.K.SureshKumar@student.tudelft.nl, (c.e.celeminpaez, j.kober)@tudelft.nl

original environment. The obtained policies were also tested in the modified environments, showing that this method can be used to obtain imitation policies with the benefits of interactive IL methods, while also being able to generalize that knowledge to several varied conditions unseen during the teacher interventions. The method was also successfully validated in two different tasks with a KUKA iiwa robot manipulator, testing generalization capabilities of the learnt policies.

II. Related Work

The proposed method that learns policies specified only in the state space is related to both methods of Learning from Observations (LfO), and Interactive IL based on relative corrections.

A. Learning from Observations

These methods are IL techniques dedicated to learn policies when the recorded demonstrations are state-only sequences, instead of state-action pair sequences [9]. This could be required, for instance, when the demonstrations are performed by a different embodiment from the one of the learner. Most of these methods are model-based, since they require to have a Forward Dynamics Model (FDM) or an Inverse Dynamics Model (IDM), in order to find what are the actions that generate the demonstrated state transitions. Behavioral Cloning from Observations [10] trains an IDM that is used to find the actions that generate the demonstrated transitions, such that the state-action pairs are completed and used for training a policy as with BC. If the environment dynamics changes, the learnt policy may not be useful, and the policy may need to be re-learnt.

Learning latent policies has been proposed in order to learn from observations, and to be able to generalize the behavior independently of the available actions. Imitating Latent Policies from Observation (ILPO) [11] is based on learning simultaneously a latent policy and a latent FDM. This generalizable latent policy leverages a small amount of environment interactions for training the mapping from the latent to the real actions in environments with different dynamics.

B. Interactive Imitation Learning with relative corrections

Interactive IL methods include the human in the learning loop with different types of teacher-agent interaction, in which the teacher could provide other kinds of information rather than full demonstrations of the task. Users could teach providing their preferences when comparing different executions of the policy [12], [13], with evaluations (rewards) about the executed actions in the visited states [14], [15], partial demonstrations of the recovery actions [16], relative corrections over the executed actions [17], or relative corrections over the state transitions [18]. In [19] it is shown that when teaching with occasional relative corrections of the executed actions, users can train policies with higher performance and with less required training time than with evaluations (human rewards) [14]. In [20] it is shown that the same approach is more intuitive and could obtain higher performances than using corrective demonstrations like in [16]. However, providing relative corrections in the action domain might not be always intuitive, especially if the actions are in a high dimensional space.

Teaching Imitative Policies in State-space (TIPS) [18] is a method that works under the assumption that the action domain is not always intuitive for the teacher, and therefore, for those cases, it is simpler to advise relative corrections in a domain observable by the teacher as it is the state/observation space, i.e. advising a desired transition, whenever the teacher consider necessary a correction. Results showed that the learning process is faster with corrective feedback in the state space than in the action space, and user studies reported that the workload was lower for the participants when they used TIPS.

Hence, in this work we propose an interactive IL algorithm that provides the same teaching experience from the user's perspective as with TIPS, but learning state space policies that could be generalized to other environments as it is done in ILPO.

III. PILOTING

In order to interactively teach robust and generalizable policies with minimum human intervention, we propose a learning scheme that requires the same kind of teacher interaction as TIPS. However, the feedback is used to shape a policy model that predicts the next desired observation depending on the current one. Such a model decouples the information of the required actions, and specifies a high level policy that describes the objective of the task step by step.

A. Policy Representation

In our approach, the objective of the task, and the dynamics of the system are decoupled by using two policies. A high level policy encodes the objective of the task, and a low level policy is in charge of performing the desired transitions based on a dynamics model. The high level policy π_o is learned from the human input once in the original environment, and a low level policy is trained based on experience sampled from the environment, any time the environment dynamics is changed. Hence, it is possible to generalize the obtained knowledge to other environments without the need of additional human intervention.

1) High Level Policy: We propose to learn policies that are a high level model that describes how the observable states should transition in order to reach the goal of a task. We assume the task to be learnt is defined within a Markov Decision Process (MDP), wherein the state $s \in S$ defines the full situation of the environment. However, the behavior to be trained is described in what we call the observation space or the teacher's observable state space O, where $O \subset S$ and $o \in O$, which is the subset of states in which the user could provide demonstrations of the task, or shape with relative corrections the desired trajectory, e.g. the position of the object to be manipulated could be the observation space, while the full state needed for control includes its velocities,

the positions and velocities of the manipulator, and some other context variables.

The unobserved states $u \in U$ are the additional variables of the state s not included in the observations o, the ones in which the teacher does not provide demonstrations or feedback, therefore, $S = O \times U$ and the current state is $s_t = [o_t\ u_t]$.

The learning behavior is described by the observations space policy π_o which predicts the next desired observation $o^{des}_{t+1} = \pi_o(o_t)$ where the environment should transition in. Therefore, π_o sets the 1-step prediction of the observation as a high level policy that a low level module takes as reference to follow. The way this policy π_o is learnt is explained in Section III-B

2) Low Level Policy: It is based on an IDM that computes the action required for a specific transition from s_t to s_{t+1}, such that $a_t = I(s_t, s_{t+1})$.

We use the scheme of Indirect Inverse Dynamics Model (IIDM) proposed in TIPS. This method involves using a learnt FDM (f_θ) to predict the next states ($s_{t+1} = f_\theta(s_t, a)$) while sampling all the possible actions ($a \in A$). It is chosen the action that predicts the next state that is closest to the desired state. The computation of the IIDM is performed as:

$$a^{des}_t = \underset{a}{\operatorname{argmin}} \left\| s^{des}_{t+1} - f_\theta(s_t, a) \right\| \tag{1}$$

In discrete action spaces, the IIDM queries all possible actions and in continuous action spaces, the actions are uniformly sampled from the action-space A. This indirect inverse dynamics formulation avoids the problem of evaluating infeasible transitions, by choosing the action that brings the agent closest to the desired next state, regardless the desired state transition is feasible or not. It also avoids the typical issues of training inverse models due to the lack of bijection. The success of the policy depends on the accuracy of the forward dynamics model learnt in that environment.

Unlike TIPS, the proposed approach does not search for the actions that produce a desired full state transition, but a desired observation transition. However, the FDM used in the IIDM scheme is specified in the complete state space, then, in order to use (1) the desired next observation o^{des}_{t+1} given by π_o should be mapped into the state space, i.e. a desired unobserved state vector u^{des}_{t+1} should be computed and concatenated to obtain a desired state ($s^{des}_{t+1} = [o^{des}_{t+1}\ u^{des}_{t+1}]$). For the desired transition to o^{des}_{t+1} from o_t, sometimes there could be many (infinite) values in U that could be paired with o_{t+1des}, and still compose a feasible state vector s^{des}_{t+1}. Nevertheless, some of those values in U might involve some costs, or even lead to undesirable states due to inconvenient transitions in the unobserved space, for instance, with the aforementioned manipulator example, the manipulated object may follow the desired trajectory, but the robot may be too close to physical constraints, or may collide with some obstacles.

In order to limit this problem, we propose to implement a null-space filter that conditions the selected actions based on some priors over the unobserved space U. This filter could be composed by a function n that predicts the desired unobserved state based on the current state ($u^{des}_{t+1} = n(s_t)$). Depending on the application, this function could aim to obtain the lowest change in u with $u^{des}_{t+1} = u_t$, e.g. moving the object (object position in o) to the desired state, while trying to move the robot the least (robot state in u); or to try to set a fixed reference for u with $u^{des}_{t+1} = u^{des}_0$, e.g. moving an object while keeping its orientation (object orientation is in u); or just set it completely variable if more priors are available. Additionally, the filter includes an importance weight vector w, that is set in order to prioritize the dimensions that should be tracked with low margin of error by this low level controller, in both the observed and unobserved states. Hence, the IIDM is computed as follows:

$$a^{des}_t = \underset{a}{\operatorname{argmin}} \left\| w \cdot ([\pi_o(o_t)\ n(s_t)] - f_\theta(s_t, a)) \right\| \tag{2}$$

The forward dynamics model f_θ is learnt from state, action, state triplets (s_t, a_t, s_{t+1}), using a feed-forward artificial neural network trained in a supervised-learning fashion. The collection of the triplets in a buffer E and the training of the neural network is carried out in two different ways, i) following a random exploratory policy (π_{rand}) to collect the triplets and training the neural network prior to evaluating the observation space policy (π_o); ii) collecting the triplets whist following the learnt observation space policy (π_o) and training the FDM model f_θ in an online manner.

B. Observation Space Policy Learning

The policy π_o could be trained in two different approaches, either in an offline fashion from a dataset of demonstrated observations as in BCO or ILPO, or interactively with occasional corrections from the teacher on the desired observation transitions, similar to what is done with the teacher corrections in the action domain with COACH [17]. However, in this work we focus on the interactive approach since it could provide higher benefits, as mentioned in Section (I).

The policy π_o is shaped while observing it trying to perform the task, it could be initialized with prior knowledge or from scratch. The human teacher advises relative corrections on the observation space, whenever she or he considers it necessary. The occasional corrections could be advised either in some or all of the dimensions of the observation space. When a correction is advised, the corrective feedback h_t would take one out of three possible values $h_t \in \{+1, 0, -1\}$ for each of the advised observation dimensions, which mean increase, keep, or decrease the current observation magnitude respectively. We keep the assumption that the teacher is a non-expert who does not know exactly how much the observations should be modified as a consequence of the actions of the agent, but she/he could estimate the direction in which the change should be done, then with these vague corrections, the policy is incrementally shaped.

In order to compute the desired correction of the observation, the teacher's feedback h_t is multiplied with the error rate e_o, and added to the current observation. The hyperparameter e_o to be tuned before runtime, is a diagonal

matrix defining the step size of the advised corrections in each dimension of the observation space.

$$\hat{o}_{t+1}^{des} = o_t + h_t.e_o \tag{3}$$

Then, the computed desired transition $\{o_t, \hat{o}_{t+1}^{des}\}$ is appended to the experience replay buffer D, which is used to train the artificial feed-forward neural network that represents the policy π_o, being o_t the inputs, and $\hat{o}_{t+1}^{des}$ the output labels. This high level policy model is updated every T time steps.

C. Policy Imitation Learning with Online Task-space Interactive Non-expert Guidance

The complete proposed learning method called PILOTING is depicted in the Figure 1, where it is shown the agent-environment interaction, and the human teacher is in the loop observing the performance of the agent in order to provide the required corrections. When there is no human correction (shown as $h = 0$), the high level policy π_o computes the desired next observation $o_{t+1}^{des} = \pi_o(o_t)$, that goes as reference to the low level control (bottom left of the diagram), in order to compute the desired action. In any time step the user recommends a correction, the advised next desired observation $\hat{o}_{t+1}^{des}$ is computed with (3), in the bottom right of the diagram, then it is used to update π_o, and finally used as reference for the low level controller that computes the action a_t.

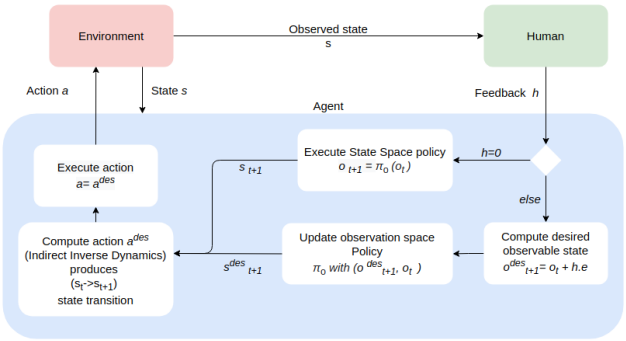


Fig. 1: High-level view of the learning framework of Interactive.

The interactive learning process of PILOTING is described in Algorithm 1, which is composed by two phases. The first phase is intended to train an initial FDM f_θ, as explained in Section III-A.2. The teaching phase is wherein the interactive process takes place, the state of the environment is observed (line 3), the human feedback is received (line 4), in case it is not null (line 5), the high level policy π_o is updated with the given feedback (lines 6-8), otherwise the current high level policy is computed (line 11). The low level policy is computed and executed (lines 13-14), the experience for updating the FDM is stored (line 15), and both models π_o and f_θ are updated (lines 16-20). The online update of f_θ could be omitted in simple environments that could be fully sampled with the random/exploration policy during the initial phase.

Generalizing high level policies to different environments: In order to generalize the learnt policies that encode the teacher's knowledge, into environments unseen during the interactive teaching time, the low level policy should be trained again while collecting transition samples from the new environment. Initializing the f_θ model of the new environment with the parameters of the one of the original environment might help to learn the model faster than learning it from scratch, especially when the change of the dynamics is relatively small. However, when the change is considerable, reusing those parameters as starting point could be also convenient since the features obtained previously within the network could be still useful, regardless the large change of the input-output mappings.

Hence, in order to generalize π_o, the Algorithm 1 could be used without considering human inputs or updates to the π_o model, i.e., just executing the lines 3, 11, 13, 14, 15, 20. Only when the change of the dynamics is too large and relearning the low level policy does not achieve the goal, it is required tuning the high level strategy, which could be done following the full algorithm.

Algorithm 1 PILOTING

Initial Model-Learning Phase

1: Generate N_e experience samples (s_t, a_t, s_{t+1}) by executing a random/exploration policy π_{rand}
2: Append samples to experience buffer E
3: Learn forward dynamics model f_θ using E

Teaching Phase:

1: **for** *episodes* **do**
2: **for** $t = 0, 1, 2, ...T$ **do**
3: Observe state s_t
4: Get human corrective feedback h_t
5: **if** h_t is not 0 **then**
6: Compute desired observation $\hat{o}_{t+1}^{des} = o_t + h_t.e_o$
7: Append $(o_t, \hat{o}_{t+1}^{des})$ to demo buffer D
8: Update policy π_o using pair $(o_t, \hat{o}_{t+1}^{des})$
9: $o_{t+1}^{des} = \hat{o}_{t+1}^{des}$
10: **else**
11: Predict next desired observation $o_{t+1}^{des} = \pi_o(o_t)$
12: **end if**
13: Compute action $a_t^{des} = \underset{a}{\operatorname{argmin}} \left\| w \cdot ([o_{t+1}^{des}\ n(s_t)] - f_\theta(s_t, a)) \right\|$
14: Execute action $a_t = a_t^{des}$, reach state s_{t+1}
15: Append (s_t, a_t, s_{t+1}) to experience buffer E
16: **if** $mod(t, T)$ **then**
17: Update policy π_o using batch sampled from demo buffer D
18: **end if**
19: **end for**
20: Update f_θ using samples from buffer E
21: **end for**

IV. Experimental Setting

The objective of the experiments is to evaluate PILOTING and compare it with the baseline TIPS, which is to the best of our knowledge, the only one method that allows non-expert teachers to train policies interactively with corrections in the state space. The comparisons are mainly focused on: i) convergence and final performance of the agent, and ii) the ability of the agents to adapt to new conditions on the environment without further human teacher's interventions. Since PILOTING is inspired by TIPS and is intended to require the same kind of teacher interaction, a user study is not considered in this experimental procedure, indeed, the main benefits of PILOTING are experienced when the knowledge obtained from the teacher could be used in unseen environments, without actual human interventions. It is not expected to have an improvement on the user experience during the interactive teaching with respect to TIPS.

A. Comparative Study

Exhaustive experiments for evaluation and comparison of agents trained with PILOTING were carried out with simulated environments from OpenAI gym, namely: CartPole, MountainCar, Pendulum and LunarLander. For this simulated problems, we set the observation state equal to the state $o_t = s_t$.

With the purpose of exhaustive evaluations of Piloting, we used oracles based on a closed form policy [21], so that the experiments are not affected by human factors and thus providing similar feedback conditions with both algorithms used in the comparison.

The oracle is used as a teacher who provides the desired observation space correction, as a non-expert human teacher would do. We ran 5 preliminary sets of experiments with non-expert human participants, the average amount of human input provided for training the agent to successfully execute the task was used as reference for the amount of feedback the oracle should give to mimic human effort put into providing feedback. The number of episodes where oracle provided feedback was provided was progressively reduced until the agent was able to learn the policy with the least amount of feedback. The amount of feedback given per episode by the oracle was determined based on average input given by a human demonstrator over 5 trials.

The total reward obtained by the agent during execution is used as a performance metric for analysing the convergence of the algorithms. Additionally, in order to evaluate the generalization capabilities of the learnt policy, physical properties (like mass, length, force applied) of the environments were modified to create new environments with dynamics unseen during the training of the policy π_o. In the environments with new dynamics, the agent could leverage the policy learnt from the original environment, without additional demonstrative effort from the teacher, which is not possible with the scheme of TIPS. We performed experiments that show the additional teacher interaction experience that is required with TIPS, for reaching a similar performance obtained with PILOTING in the new environments. For this, it is shown the percentage of time steps per episode that the teacher provides input with each learning method.

B. Ablation study

In order to evaluate the influence of the FDM learning strategy on the overall performance of the system, we carried out two ablation studies varying the FDM learning scheme and including the use of the actual FDM model. The two ablation studies cover all cases of learning only the high level policy, only the low level policy, and both.

1) Learning in the original environment: This study involves learning both the high and low level policies in the original environments. The independent variable of this study is the strategy to obtain the FDM with the following cases:

(a) Using the actual FDM (not learning it).
(b) Learning the FDM online, without the initial phase of Algorithm 1.
(c) Learning the FDM offline, with the exploratory policy π_{rand} in the initial phase.

The first case works as the reference for evaluating how is the progress of only learning the high level policy, i.e., it measures how long does it take to shape the high level policy since there is no influence of low performance low level policies. The other two cases show actually how is the convergence of learning both models either simultaneously as in the second case, or sequentially as in the third case.

2) Learning in a modified environment: In this study a learnt high level policy is reused in a modified environment, without additional training, while learning only the low level policy. Two independent variables are considered in this study, one is the initialization of the FDM which could be either a complete random set of parameters, or the FDM parameters obtained while learning the policy in the original environment. The other variable is the sampling/update strategy, which could be either learning offline or online, as in the previous study. It is also included the use of the actual FDM as a reference that works for measuring the performance of the actual high level policy, and in order to know what is the upper bound performance of the study. Therefore, there are four possible cases that were considered in this study, along with the one using the actual FDM, as listed below.

(a) Using the actual FDM (not learning it).
(b) Learning a new FDM from scratch online, without the initial phase of Algorithm 1.
(c) Learning a new FDM from scratch offline, with the exploratory policy π_{rand} in the initial phase.
(d) Learning the FDM online, without the initial phase of Algorithm 1, but initializing it with the learnt FDM of the original environment.
(e) Learning the FDM offline, with the exploratory policy π_{rand} in the initial phase, but initializing it with the learnt FDM of the original environment.

In the first study, it is evaluated the cases of learning only the high level policy, and learning both together, while in the second study the remaining case of learning only the low level policy is evaluated considering the main possible variants.

C. Validation on a real robot task

The proposed method was validated in two different manipulation tasks, namely 'car balancing' and 'object pulling' with a real robot arm KUKA iiwa. The tasks were performed using a Cartesian position controller. To teach these tasks, the teacher uses a joystick as interface, for providing feedback in the form of correction of the current observation (movement of the balanced car, and position of the pulled object respectively). The low level policy takes the desired transition and finds the corresponding action that moves the robot end-effector accordingly.

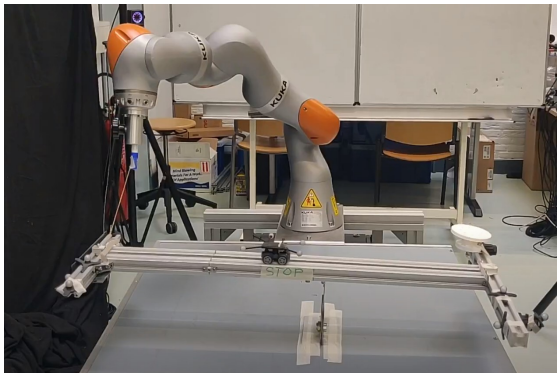


Fig. 2: Experiment setup for Balance task on the KUKA robot. The task is to balance the car at the middle of the track.

1) Car balancing task: In this task (fig 2), a car is placed on a track which is pivoted at the center, allowing it to tilt around the axis, and moving the car along the track depending on its inclination. The robot tilts the track by pulling a thread attached to one end of it. The objective is to place the car at the center of the track. The state space of the environment $s_t = [x\ \dot{x}\ \theta\ \dot{\theta}]$, i.e. the position, and velocity of the car, and the angle, angular velocity of the track. The observation space is $o_t = [x\ \dot{x}]$, thus, the unobserved states are $u_t = [\theta\ \dot{\theta}]$.

For this specific problem, we assumed $u_{t+1}^{des} = n(s_t) = [0\ 0]$, since that is the ideal target unobserved state for the time in which the car is centered, because it prioritizes having the track leveled and static, i.e., a state that would not move the car to any side, but at the same time a state that could produce a movement to any side with just a small change, thus, it is easy to quickly compensate the car position if required. The action space of the robot is limited to discrete actions in the vertical axis.

To evaluate the task, we define a reward function to be used as a performance metric. The function is inspired from the MountainCar task and consists of a negative reward at every time-step until the car is centered on the track with an almost zero velocity. The learnt observation state space policy π_o is tested in a modified environment, wherein the robot instead of pulling the track from one end, it pushes down the other end of the track, as shown in figure 3.

This setting changes the way the manipulator interacts with the track, the movements of the robot have an opposite effect on the movement of the track. Another change of the environment was tested with the change of the pivot to a point more distant from the end of the track wherein the robot is attached as shown in figure 4. This requires the robot to perform larger actions.

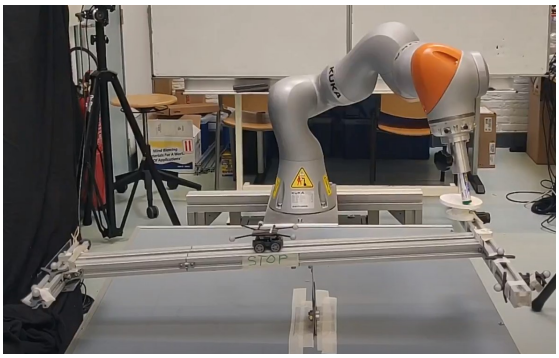


Fig. 3: Modified environment to test state space policy: Balancing the car by pushing the lever on the opposite side.

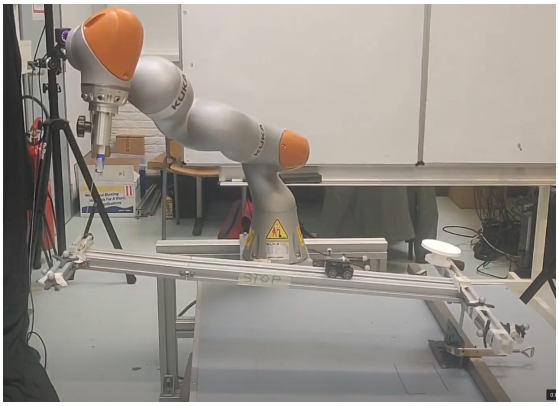


Fig. 4: Modified environment to test state space policy: The pivot point is moved to one end of the track, larger actions are required to move the car on the track.

2) Object pulling task: In this task (fig 5), a puck is tied to the end-effector by a string. The objective is to move the end effector in such a way that the puck being pulled traces a path around some obstacles. The state space of the environment is $s_t = [x_p\ y_p\ x_r\ y_r]$, i.e., $[x_r\ y_r]$ the coordinates of the puck, and $[x_r\ y_r]$ the coordinates of the robots end-effector. The information of s_t could be also represented with the coordinates of the robot being relative to the puck $s_t = [x_p\ y_p\ x_{pr}\ y_{pr}]$, that additionally could be in polar coordinates $s_t = [x_p\ y_p\ d\ \varphi]$, being d and φ the distance between the robot end-effector and the puck, and its angle, respectively. The observation space is $o_t = [x_p\ y_p]$, while the unobserved state space is $u_t = [d\ \varphi]$, this polar representation is convenient for defining the null-space filter, since it is preferred to keep the rope as much stretched as possible, because that is how always a movement of the robot results in a movement of the puck. Thus, it is desired

to maximize d (constrained by the length of the rope l), while the angle φ is not relevant (best prior about φ is to try avoiding large changes following the current value φ_t). Hence, for this task it is assumed $u_{t+1}^{des} = n(s_t) = [l\ \varphi_t]$. The action space of the robot is limited to discrete changes in the XY plane.

To evaluate the performance of the policies in this task, it is compared the trajectory of the puck with a reference trajectory of the object going from the starting to the target point, while avoiding collisions with some obstacles. The executed trajectory is compared with the reference trajectory using the Hausdorff distance [22]. The learnt high level policy π_o is tested in an environment with different dynamics, by changing the length of the string attaching the end-effector to the puck. With this change, the robot has to perform a different and wider trajectory which is not necessarily proportional to the one executed in the original environment, rather a nonlinear transformation of it.

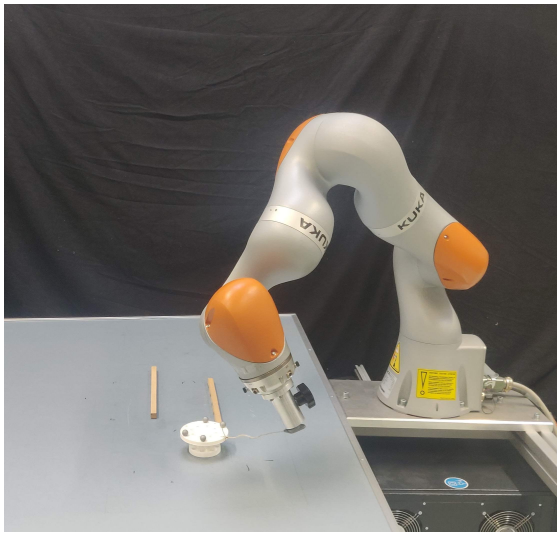


Fig. 5: Experiment setup for rope drawing task on the KUKA robot. The task is to move the end effector in such a way that the puck traces a path around obstacles

V. Results

The carried out experiments lead to the results presented below.

A. Comparative Study

The comparisons of teaching a policy interactively with PILOTING and TIPS are plotted in the figures 6-9, wherein the learning curves are showing the return obtained by the current policy throughout the time steps of the learning process. For each problem, it is shown the learning curves in the original environment (in the left), and the curves of learning in one of the multiple tested modified environments (in the right), which is the case of the environment with actions being multiplied by -1, that we call 'Negative Dynamics'.

In general, when learning the policies in the original environment, both algorithms show a very similar convergence, and similar human input. The cases of Cartpole, and Lunarlander required slightly more human input with PILOTING. Nevertheless, there is a considerable advantage of PILOTING when the dynamics change and the high level policy is reused, since there is no need to modify the high level policies with further human interventions, while with TIPS it is still needed the human input as when learning in the original environment.

With Piloting, it is just needed to sample transitions of the environment during 500 time steps with π_{rand} in the initial phase of the algorithm, in order to capture the new FDM behavior. Only in the case of the Mountaincar (figure 7), the data sampled with π_{rand} is not representative enough to train a model that describes properly the system, therefore, it is required to keep collecting transition samples while following the high level policy. That is why in this problem, with both algorithms, the performance is the lowest possible during the first episodes of training. Although, in the modified environment, PILOTING started having successful executions in half the steps required by TIPS to do so, and still without additional teacher input.

The results of training in the LunarLander environment showed that neither PILOTING nor TIPS learn an optimal policy to land the spacecraft between the flags, however both obtain policies that are able to softly land it.

Table I, gives a summary of the various modifications done to several OpenAI gym environments to test the high-level policy learnt. The table shows the average return obtained by the agents in the modified environment and the number of steps they take to archive the average return. It can be seen that for small changes to the environment dynamics, the agent following Piloting does not require any additional steps to be able to perform the task in the modified environment, where as agent following TIPS requires additional demonstrative feedback to be able to execute the task.

B. Ablation Study

For these studies, the Lunarlander environment is not considered because in the previous experiments with both evaluated algorithms, the obtained performances were relatively low and the task was partially fulfilled.

1) Learning in the original environment: The results of this experiments are depicted in figure 10-12. It could be seen that learning both policies makes the process slower than only needing to learn the high level one. In general for most of the cases, the final performance of the learnt FDMs is in the same level as the actual FDMs, since they allow to follow the desired transitions and obtain a similar average return with respect to the baseline based on the true FDM.

In the Cartpole problem, we could see that not knowing the FDM could take the double of the required time for convergence, with respect to training based on a known FDM. Learning the FDM online or offline obtains the convergence almost at the same time, although in this problem, the offline approach has the advantage that the teacher had to participate

TABLE I: Average return for various changes in the environments

Evironment	Modification to environement	PILOTING		TIPS	
		average return obtained	steps to achieve average return	average return obtained	steps to achieve average return
CartPole-v1	No modification	500	600	500	550
	the mass of the pole was halved, $m_p = 0.5$	497.2	0	500	50
	the mass of the pole was double, $m_p = 2$	500	0	500	50
	the mass of the cart was halved, $m_c = 0.5$	495.8	0	500	150
	the mass of the cart was double, $m_c = 2$	493	50	497	150
	the length of the pole was halved, $l = 0.5$	496	0	500	100
	the length of the pole was double, $l = 2$	476.9	50	500	500
	the force applied on the cart was halved, $f = 0.5$	498.2	50	500	200
	the force applied on the cart was double, $f = 2$	500	0	500	0
	the force applied on the cart was negative, $f = -1$	500	0	500	700
MountainCar-v0	No modification	-124.26	1250	-128.17	1200
	the force applied on the cart was negative, $f = -1$	-129.31	500	-123.02	1150
Pendulum-v0	No modification	-86.63	400	-82.23	350
	the mass of the pole was reduced, $m_p = 0.75$	-61.66	0	-52.25	0
	the mass of the pole was increased, $m_p = 1.5$	-121.88	0	-124.04	100
	the torque applied on the link was reduced, $f = 0.75$	-149.86	0	-152.37	150
	the force applied on the cart was increased, $f = 1.5$	-74.12	0	-79.12	0
	the force applied on the cart was negative, $f = -1$	-85.78	0	-83.77	300
LunarLander-v2	No modification	-164.9	750	-134.69	750
	the main engine power in was reduced, $f_{MAIN} = 10$	-189.31	0	-179.75	250
	the main engine power in was increased, $f_{MAIN} = 16$	-179.45	0	-199.2	0
	the side engine power in was reduced, $f_{SIDE} = 0.4$	-168.21	50	-183.8	150
	the side engine power in was increased, $f_{SIDE} = 0.9$	-169.77	50	-164.77	200
	the side engine power in was negative, $f_{SIDE} = -0.6$	-164.32	0	-154.43	800

during around 40% less time, because the first 500 time steps are dedicated to the initial model learning phase.

For the problem of the MountainCar, the results are rather different, because learning offline was not successful. This is because for this problem, the exploratory policy π_{ramd} is not enough to obtain representative samples from the entire environment, i.e., the random exploration never makes the car to move out of the bottom of the valley, therefore the learnt model only describes correctly one area of the state space.

The results of the pendulum show a rather fast improvement at the beginning, although the three cases converged almost at the same time. With the offline approach there is an steeper improvement (when the interactive phase starts after the first 500 time steps), however, the online case has a considerable improvement before the 500 steps, which could be convenient in situations wherein a good but suboptimal policy is acceptable to be rolled out.

From these results, it is observed that learning each of the models has a similar contribution to the time required for the convergence, and none of them represents a big bottleneck in this regard. Although of course it is always needed the existence of an FDM in order to be able to train a high level policy, whereas the other way around is not always a limitation, i.e., it is not always required a high level policy to be able to train a FDM as in the Cartpole and Pendulum problems, only the high level policy π_o was required for learning the FDM in the MountainCar problem since it helped to obtain better samples of the system that the exploratory policy π_{rand}.

2) Learning in a modified environment: This study evaluated the remaining case of learning only the FDM. As expected from the observations of the previous study, the results in figures 13-15 showed that it takes less time to only learn the FDM that obtains the desired transitions requested by a known high level policy.

In general, the online learning approach seems to be more beneficial to have relatively good performance with few time steps, and reusing the model of the original dynamics for initializing the new FDM seems to obtain an advantage in the velocity of convergence, since part of the features extraction is reused from the original model and is not fully learnt from scratch again.

C. Validation

1) Car balancing task: In the figure 16 it is shown the learning curve of this task along with the percentage of human input. It could be seen that around the episode 20 the policy becomes able to achieve the objective with a low performance, but with continuous improvement. After the episode 26 the amount of feedback given is reduced considerably, however the trend of improvement in the return keeps relatively constant, meaning that there are less corrections which are still meaningful and relevant for the task execution.

When changing the configuration of the environment as described in Section IV-C.1, the learnt policy was successful to fulfill the objective of the task. It was enough the exploratory policy for learning the new dynamics in order to reuse the high level policy. It is possible to see that when the robot has to push (instead of pulling) in the other side of the track, the performance is rather similar to the one of the policy in the original environment. However, when the pivot is moved to the end of the track, the task is also accomplished but with a lower performance, this is due to the fact that in this setup the robot had to execute larger movements in order to obtain the same changes in the inclination of the track, however, the velocity of the robot was limited.

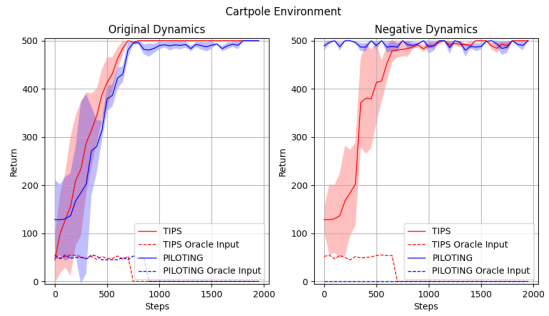


Fig. 6: Average return in Cartpole environment in with original dynamic and modified dynamics (Right)

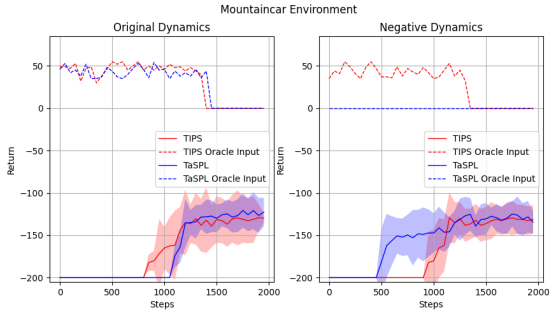


Fig. 7: Average return in MountainCar environment in with original dynamic and modified dynamics (Right)

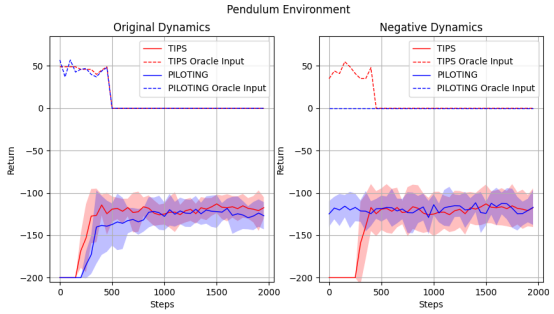


Fig. 8: Average return in Pendulum environment in with original dynamic and modified dynamics (Right)

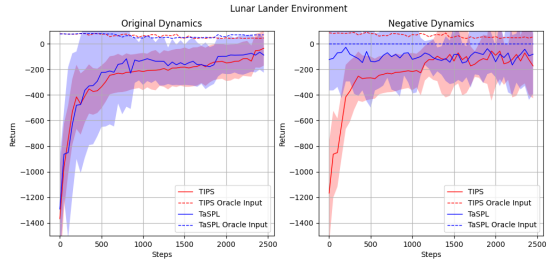


Fig. 9: Average return in LunarLander environment in with original dynamic and modified dynamics (Right)

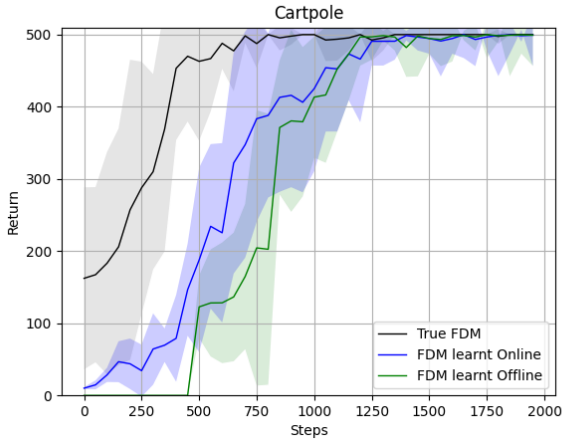


Fig. 10: Average return in Cartpole environment

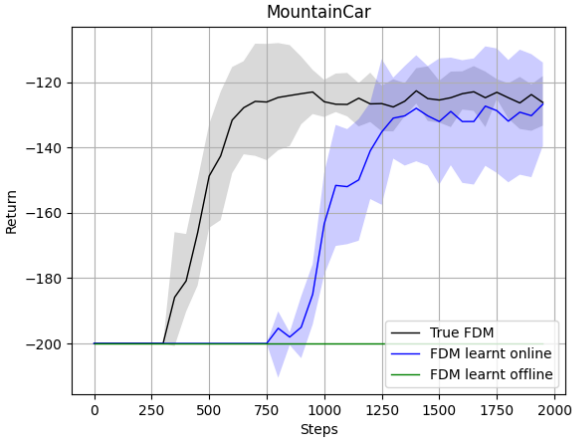


Fig. 11: Average return in Mountain Car environment

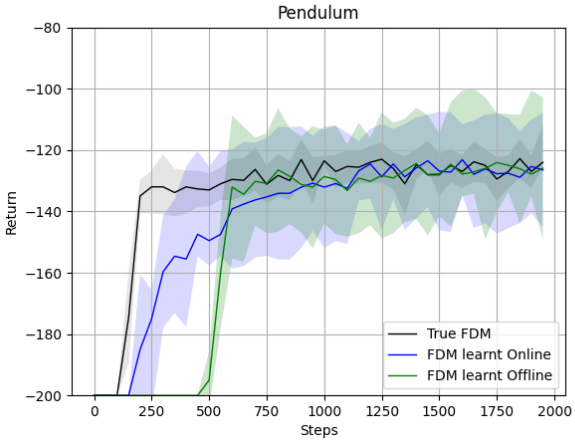


Fig. 12: Average return in Pendulum environment

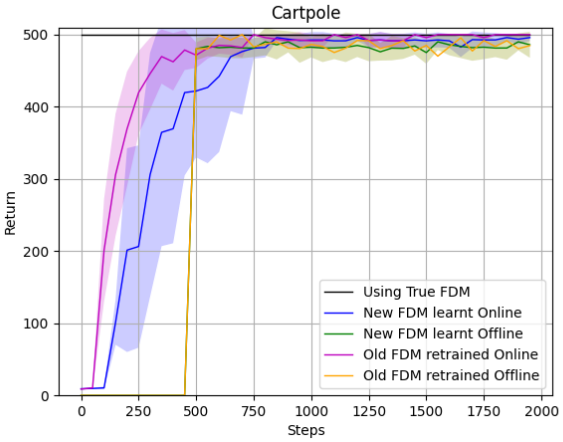


Fig. 13: Average return in Cartpole environment

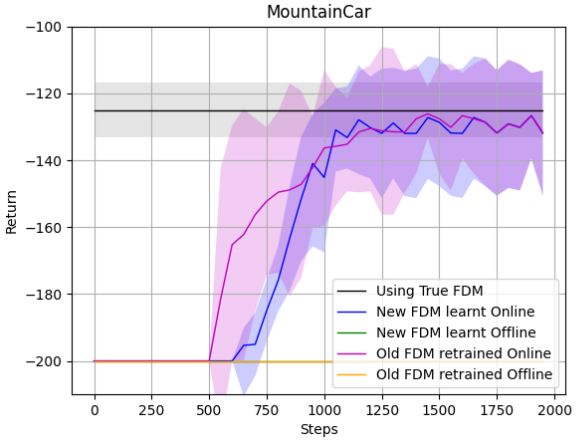


Fig. 14: Average return in MountainCar environment

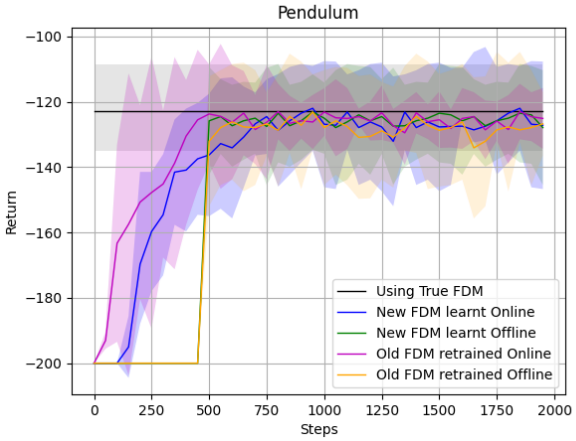


Fig. 15: Average return in Pendulum environment

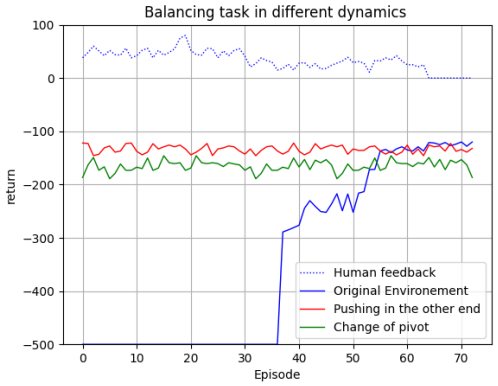


Fig. 16: Validation task: Learning curves and demonstrator feedback rates for the Balance experiment

2) Object pulling task: In this task the high level policy was taught in the setup with a short rope for pulling the object, and later this policy was successfully validated with a longer rope without additional human input. From the feedback given by the teacher, the agent is has learnt to trace a path in the XY plane within 20 episodes (fig 17). The agent successfully learns to trace a path with a short string. Further feedback is provided for the agent to learn to avoid the obstacle. The amount of feedback reduces once the agent has learnt the basic path. Additional feedback is given to fine-tuned the behavior to avoid obstacles.

As it is depicted in the figure 18, with dotted lines it is shown the trajectories followed by the object and with the solid lines the trajectories executed by the the robot end-effector. The black dotted line is an object reference trajectory obtained with teleoperation. In red, it is shown the trajectories obtained in the original environment, and in blue the ones with the modified dynamics (long rope).

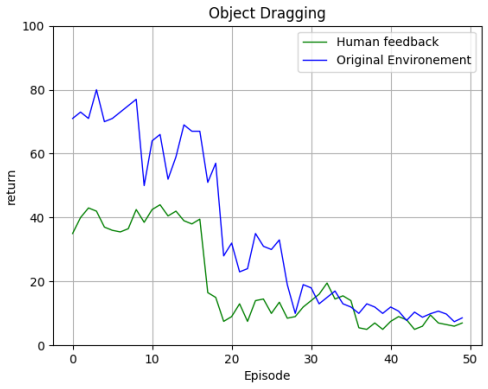


Fig. 17: Validation task: Learning curves and demonstrator feedback rates for the Object Pulling task

The high level policy was successful at executing the task, avoiding the obstacles seen in figure 5, while the low level policy had to execute a rather different path in order to make the object to follow the desired trajectory without collisions.

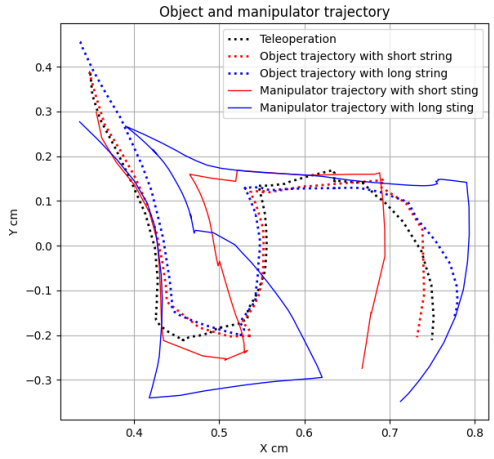


Fig. 18: Validation task: Path traced by the puck and the end effector when using a short thread and a long thread

VI. Conclusion

In this work we have introduced a policy learning method that offers the advantages of interactive imitation learning, which offers: i) a strategy to collect better samples in order to train more robust policies, avoiding the distribution shift issues: ii) a reduction on the effort of the human teacher giving the possibility of correcting only when considered necessary, instead of every time step; iii) another reduction on the teacher load because the interventions are given with corrections on the observed state space that could be simpler than correcting the executed actions, as tested before by TIPS [18]; iv) a decoupling of the dynamics of the environment with the split of the policy learning into the high and low level policies, a strategy that allows to generalize the knowledge transferred by the teacher to unseen conditions of the dynamics of the environment, without additional teacher intervention.

The method was tested exhaustively in simulations, and compared to TIPS, showing that PILOTING could be equivalent to TIPS when the observed space is the full state space. However, the results showed that PILOTING learns general policies that could be applied to varied changes of the environment without any additional guidance of the teacher, because it manages to extract and represent better the relevant knowledge from the teacher interventions. This policy representation characteristic of PILOTING provides an advantage over TIPS, which learns a standard state to action policy that cannot be reused successfully in the new environments.

The ablation studies showed that the proposed policy structure composed of the high and low level policies could work more data efficiently whenever one of them is known, and that in general each of them contributes to around half of the experience load required during learning, i.e., when one of the policies is known, it is only needed half of the time to learn the other one with respect to the situation of learning both policies together. The experiments showed that learning the forward dynamics model online is the alternative that worked for all the situations, since it creates the loop in which the improvement of the policy helps to obtain better transition samples that work for improving the forward dynamics model, which again benefits the policy improvement, as the learning loop present in some of the Model Learning RL methods like PILCO [23].

The validation of PILOTING in the real robot tasks, wherein the change of environment was more evident, showed that the method is successful at learning policies that capture the high level intention of the teacher.

However, there are some considerations that can limit the performance of the proposed algorithm. If the dynamics of the environment changes to a large extent, the agent might have to follow a different strategy/ trajectory to achieve the objective. The high level policy learnt may not be optimal to execute the task. Additional feedback may ne required to make the learnt policy optimal in the new environment. In the relatively small dimensional spaces in our experiments, the action computation is carryout in every step and is inexpensive, however this does, not hold for higher dimensional spaces where a lot of computational power would be required, we could solve this by training a state to action policy simultaneously,

We explained the learning framework used for learning a policy in observation space and how learning the policy in an observation space allows the agent to execute the task even if the environment dynamics changes. Learning the policy in observation space allows the agent to execute the task in the new environment by reusing the policy from the old environment and knowledge of the agent's dynamic model in the new environment. The dynamics model can be either provided or can be learnt from interaction with the environment.

However, there are some considerations that can limit the performance of the proposed algorithm. Firstly, PILOTING queries the dynamics model in each step for computing the action that produces the desired state transition. The performance of our algorithm is dependent on the quality of the learnt state space policy and the dynamics model. Learning a dynamics model through random exploration can be difficult in many environments and typically requires a significant number of environment interactions. The proposed algorithm computes action by sampling random actions in a given state, followed by execution of the chosen action. Secondly the proposed algorithm suffers same the limitations of IIDM outlined in TIPS, computing actions from IIDM would not be scalable in high dimensional continuous action spaces, such computation would require more computational resources and affecting the real time execution.

One solution to avoid the querying the FDM in each step is to learn state action policy in the new environment, without the need for additional human teacher interventions. The state space policy learnt in the original environment can be used to train or retrain a state-action policy in the new environment. The above proposed method was implemented

as modification to TIPS, where the state action policy learnt in the original environment is retrained using the state space policy learnt in the original environment. The state space policy is learnt simultaneously in an online manner while the demonstrator trains the state action policy. In the new environment, no additional demonstrative effort is required for training the state action policy. The advantage of such a method is that the agent can be trained in a human intuitive environment, but the agent is able to perform the same task in a new environment where the dynamics of the environment may not be intuitive. However, the quality of the new state action policy learnt is dependent on the quality of the state space policy (π_o) and the forward dynamics model f_θ learnt.

REFERENCES

[1] S. Schaal *et al.*, "Learning from demonstration," *Advances in neural information processing systems*, pp. 1040–1046, 1997.

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

[3] S. Raza, S. Haider, and M.-A. Williams, "Teaching coordinated strategies to soccer robots via imitation," in *2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 2012, pp. 1434–1439.

[4] D. Michie, M. Bain, and J. Hayes-Miches, "Cognitive models from subcognitive skills," *IEE control engineering series*, vol. 44, pp. 71–99, 1990.

[5] S. Daftry, J. A. Bagnell, and M. Hebert, "Learning transferable policies for monocular reactive mav control," in *International Symposium on Experimental Robotics*. Springer, 2016, pp. 3–11.

[6] A. Y. Ng, S. J. Russell *et al.*, "Algorithms for inverse reinforcement learning." in *Icml*, vol. 1, 2000, p. 2.

[7] S. Arora and P. Doshi, "A survey of inverse reinforcement learning: Challenges, methods and progress," *Artificial Intelligence*, p. 103500, 2021.

[8] Y. Cui, P. Koppol, H. Admoni, S. Niekum, R. Simmons, A. Steinfeld, and T. Fitzgerald, "Understanding the relationship between interactions and outcomes in human-in-the-loop machine learning," in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, Z.-H. Zhou, Ed. International Joint Conferences on Artificial Intelligence Organization, 8 2021, pp. 4382–4391, survey Track.

[9] F. Torabi, G. Warnell, and P. Stone, "Recent advances in imitation learning from observation," *arXiv preprint arXiv:1905.13566*, 2019.

[10] ——, "Behavioral cloning from observation," *arXiv preprint arXiv:1805.01954*, 2018.

[11] A. Edwards, H. Sahni, Y. Schroecker, and C. Isbell, "Imitating latent policies from observation," in *International Conference on Machine Learning*, 2019, pp. 1755–1763.

[12] R. Akrour, M. Schoenauer, M. Sebag, and J.-C. Souplet, "Programming by feedback," in *International Conference on Machine Learning*, vol. 32. JMLR. org, 2014, pp. 1503–1511.

[13] P. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," *arXiv preprint arXiv:1706.03741*, 2017.

[14] 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*, 2009, pp. 9–16.

[15] G. Warnell, N. Waytowich, V. Lawhern, and P. Stone, "Deep tamer: Interactive agent shaping in high-dimensional state spaces," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[16] M. Kelly, C. Sidrane, K. Driggs-Campbell, and M. J. Kochenderfer, "Hg-dagger: Interactive imitation learning with human experts," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8077–8083.

[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)*. IEEE, 2019, pp. 7611–7617.

[18] S. Jauhri, C. E. Celemin, and J. Kober, "Interactive imitation learning in state-space," in *Conference on Robot Learning (CoRL)*, 2020.

[19] C. Celemin and J. Ruiz-del Solar, "An interactive framework for learning continuous actions policies based on corrective feedback," *Journal of Intelligent & Robotic Systems*, vol. 95, no. 1, pp. 77–97, 2019.

[20] R. Perez-Dattari, C. Celemin, G. Franzese, J. Ruiz-del Solar, and J. Kober, "Interactive learning of temporal features for control: Shaping policies and state representations from human feedback," *IEEE Robotics & Automation Magazine*, vol. 27, no. 2, pp. 46–54, 2020.

[21] Z. Xiao, "Solutions to openai gym," https://github.com/ZhiqingXiao/OpenAIGymSolution, 2020.

[22] D. P. Huttenlocher, G. A. Klanderman, and W. J. Rucklidge, "Comparing images using the hausdorff distance," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 15, no. 9, pp. 850–863, 1993.

[23] M. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on machine learning (ICML-11)*. Citeseer, 2011, pp. 465–472.

Bibliography

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

Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in neural information processing systems*, pages 1–8, 2007.

Pulkit Agrawal, Ashvin V Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in neural information processing systems*, pages 5074–5082, 2016.

Riad Akrour, Marc Schoenauer, Michèle Sebag, and Jean-Christophe Souplet. Programming by feedback. In *International Conference on Machine Learning*, volume 32, pages 1503–1511, 2014.

Saleema Amershi, Maya Cakmak, William Bradley Knox, and Todd Kulesza. Power to the people: The role of humans in interactive machine learning. *Ai Magazine*, 35(4):105–120, 2014.

Brenna D Argall. Learning mobile robot motion control from demonstration and corrective feedback. *Diss. University of Southern California*, 2009.

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

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

Kavosh Asadi, Evan Cater, Dipendra Misra, and Michael L Littman. Towards a simple approach to multi-step model-based reinforcement learning. *arXiv preprint arXiv:1811.00128*, 2018.

Christopher G Atkeson and Stefan Schaal. Robot learning from demonstration. In *ICML*, volume 97, pages 12–20. Citeseer, 1997.

Christopher G Atkeson, Andrew W Moore, and Stefan Schaal. Locally weighted learning for control. In *Lazy learning*, pages 75–113. Springer, 1997.

Yusuf Aytar, Tobias Pfaff, David Budden, Thomas Paine, Ziyu Wang, and Nando de Freitas. Playing hard exploration games by watching youtube. In *Advances in neural information processing systems*, pages 2930–2941, 2018.

J Andrew Bagnell and Jeff G Schneider. Autonomous helicopter control using reinforcement learning policy search methods. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, volume 2, pages 1615–1620. IEEE, 2001.

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

Michael Bain and Claude Sammut. A framework for behavioural cloning. In *Machine Intelligence 15*, pages 103–129, 1995.

Nir Baram, Oron Anschel, Itai Caspi, and Shie Mannor. End-to-end differentiable adversarial imitation learning. In *International Conference on Machine Learning*, pages 390–399, 2017.

Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 1983.

Roger Bemelmans, Gert Jan Gelderblom, Pieter Jonker, and Luc De Witte. Socially assistive robots in elderly care: A systematic review into effects and effectiveness. *Journal of the American Medical Directors Association*, 13(2):114–120, 2012.

Aude Billard and Daniel Grollman. *Imitation Learning in Robots*, pages 1494–1496. Springer US, Boston, MA, 2012. ISBN 978-1-4419-1428-6. doi: 10.1007/978-1-4419-1428-6_758. URL https://doi.org/10.1007/978-1-4419-1428-6_758.

Aude Billard, Sylvain Calinon, Ruediger Dillmann, and Stefan Schaal. Survey: Robot programming by demonstration. *Handbook of robotics*, 59(BOOK_CHAP), 2008.

Wendelin Böhmer, Jost Tobias Springenberg, Joschka Boedecker, Martin Riedmiller, and Klaus Obermayer. Autonomous learning of state representations for control: An emerging field aims to autonomously learn state representations for reinforcement learning agents from their real-world sensor observations. *KI-Künstliche Intelligenz*, 29(4):353–362, 2015.

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

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

Róbert Busa-Fekete, Balázs Szörényi, Paul Weng, Weiwei Cheng, and Eyke Hüllermeier. Preference-based evolutionary direct policy search. In *ICRA Workshop on Autonomous Learning*, 2013.

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

Carlos Celemin and Javier 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.

Jason Chen and Alex Zelinsky. Programing by demonstration: Coping with suboptimal teaching actions. *The International Journal of Robotics Research*, 22(5):299–319, 2003.

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *Advances in neural information processing systems*, 29:2172–2180, 2016.

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

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

Silvia Chiappa, Sébastien Racaniere, Daan Wierstra, and Shakir Mohamed. Recurrent environment simulators. *arXiv preprint arXiv:1704.02254*, 2017.

Paul Christiano, Zain Shah, Igor Mordatch, Jonas Schneider, Trevor Blackwell, Joshua Tobin, Pieter Abbeel, and Wojciech Zaremba. Transfer from simulation to real world through learning deep inverse dynamics model. *arXiv preprint arXiv:1610.03518*, 2016.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in neural information processing systems*, pages 4299–4307, 2017.

John D Co-Reyes, YuXuan Liu, Abhishek Gupta, Benjamin Eysenbach, Pieter Abbeel, and Sergey Levine. Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings. *arXiv preprint arXiv:1806.02813*, 2018.

Felipe Codevilla, Eder Santana, Antonio M López, and Adrien Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9329–9338, 2019.

Adria Colomé, Antoni Planells, and Carme Torras. A friction-model-based framework for reinforcement learning of robotic tasks in non-rigid environments. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 5649–5654. IEEE, 2015.

John J Craig. *Introduction to robotics: mechanics and control, 3/E*. Pearson Education India, 2009.

Yuchen Cui, Pallavi Koppol, Henny Admoni, Scott Niekum, Reid Simmons, Aaron Steinfeld, and Tesca Fitzgerald. Understanding the relationship between interactions and outcomes in human-in-the-loop machine learning. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 4382–4391. International Joint Conferences on Artificial Intelligence Organization, 8 2021. Survey Track.

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

Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. Adversarially learned inference. *arXiv preprint arXiv:1606.00704*, 2016.

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

Matthew Field, David Stirling, Fazel Naghdy, and Zengxi Pan. Motion capture in robotics review. In *2009 IEEE International Conference on Control and Automation*, pages 1697–1702. IEEE, 2009.

Yarin Gal, Rowan McAllister, and Carl Edward Rasmussen. Improving pilco with bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop, ICML*, volume 4, page 34, 2016.

Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G Bellemare. Deepmdp: Learning continuous latent space models for representation learning. *arXiv preprint arXiv:1906.02736*, 2019.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27:2672–2680, 2014.

Ross Goroshin, Michael F Mathieu, and Yann LeCun. Learning to linearize under uncertainty. *Advances in neural information processing systems*, 28:1234–1242, 2015.

Elena Gribovskaya, Seyed Mohammad Khansari-Zadeh, and Aude Billard. Learning nonlinear multivariate dynamics of motion in robotic manipulators. *The International Journal of Robotics Research*, 30(1):80–117, 2011.

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

Abhishek Gupta, Coline Devin, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Learning invariant feature spaces to transfer skills with reinforcement learning. *arXiv preprint arXiv:1703.02949*, 2017.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pages 2555–2565. PMLR, 2019.

Josiah P Hanna and Peter Stone. Grounded action transformation for robot learning in simulation. In *AAAI*, pages 3834–3840, 2017.

Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller. Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations*, 2018.

Todd Hester and Peter Stone. Real time targeted exploration in large domains. In *2010 IEEE 9th International Conference on Development and Learning*, pages 191–196. IEEE, 2010.

Todd Hester and Peter Stone. Learning and using models. In Marco Wiering and Martijn van Otterlo, editors, *Reinforcement Learning: State of the Art*. Springer Verlag, Berlin, Germany, 2011.

Todd Hester, Michael Quinlan, and Peter Stone. Generalized model learning for reinforcement learning on a humanoid robot. In *2010 IEEE International Conference on Robotics and Automation*, pages 2369–2374. IEEE, 2010.

Todd Hester, Michael Quinlan, and Peter Stone. Rtmba: A real-time model-based reinforcement learning architecture for robot control. In *2012 IEEE International Conference on Robotics and Automation*, pages 85–90. IEEE, 2012.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in neural information processing systems*, pages 4565–4573, 2016.

Sepp Hochreiter. Ja1 4 rgen schmidhuber (1997).“long short-term memory”. *Neural Computation*, 9(8).

Zhang-Wei Hong, Tsu-Jui Fu, Tzu-Yun Shann, and Chun-Yi Lee. Adversarial active exploration for inverse dynamics model learning. In *Conference on Robot Learning*, pages 552–565, 2020.

Daniel P Huttenlocher, Gregory A. Klanderman, and William J Rucklidge. Comparing images using the hausdorff distance. *IEEE Transactions on pattern analysis and machine intelligence*, 15(9):850–863, 1993.

Auke Jan Ijspeert, Jun Nakanishi, and Stefan Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, volume 2, pages 1398–1403. IEEE, 2002.

Ashesh Jain, Brian Wojcik, Thorsten Joachims, and Ashutosh Saxena. Learning trajectory preferences for manipulators via iterative improvement. *Advances in neural information processing systems*, 26:575–583, 2013.

Miguel Jaques, Michael Burke, and Timothy Hospedales. Newtonianvae: Proportional control and goal identification from pixels via physical latent spaces. *arXiv preprint arXiv:2006.01959*, 2020.

Snehal Jauhri, Carlos Celemin, and Jens Kober. Interactive imitation learning in state-space. *arXiv preprint arXiv:2008.00524*, 2020.

Abhishek Jha, Shital S. Chiddarwar, Rohini Y. Bhute, Veer Alakshendra, Gajanan Nikhade, and Priya M. Khandekar. Imitation learning in industrial robots: A kinematics based trajectory generation framework. In *Proceedings of the Advances in Robotics*, AIR '17, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450352949. doi: 10.1145/3132446.3134879. URL https://doi-org.tudelft.idm.oclc.org/10.1145/3132446.3134879.

Rico Jonschkowski and Oliver Brock. Learning state representations with robotic priors. *Autonomous Robots*, 39(3):407–428, 2015.

Michael I Jordan and David E Rumelhart. Forward models: Supervised learning with a distal teacher. *Cognitive science*, 16(3):307–354, 1992.

Leslie Pack Kaelbling. Learning to achieve goals. In *IJCAI*, pages 1094–1099. Citeseer, 1993.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

Sergey Karakovskiy and Julian Togelius. The mario ai benchmark and competitions. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):55–67, 2012.

Maximilian Karl, Maximilian Soelch, Justin Bayer, and Patrick Van der Smagt. Deep variational bayes filters: Unsupervised learning of state space models from raw data. *arXiv preprint arXiv:1605.06432*, 2016.

Nan Rosemary Ke, Amanpreet Singh, Ahmed Touati, Anirudh Goyal, Yoshua Bengio, Devi Parikh, and Dhruv Batra. Learning dynamics model in reinforcement learning by incorporating the long term future. *arXiv preprint arXiv:1903.01599*, 2019.

Michael Kelly, Chelsea Sidrane, Katherine Driggs-Campbell, and Mykel J Kochenderfer. Hg-dagger: Interactive imitation learning with human experts. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8077–8083. IEEE, 2019.

Eric C Kerrigan and Jan M Maciejowski. Designing model predictive controllers with prioritised constraints and objectives. In *Proceedings. IEEE International Symposium on Computer Aided Control System Design*, pages 33–38. IEEE, 2002.

H Jin Kim, Michael I Jordan, Shankar Sastry, and Andrew Y Ng. Autonomous helicopter flight via reinforcement learning. In *Advances in neural information processing systems*, pages 799–806, 2004.

Won S Kim, Blake Hannaford, and Antal K Bejczy. Force-reflection and shared compliant control in operating telemanipulators with time delay. *Robotics and Automation, IEEE Transactions on*, 8(2):176–185, 1992.

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

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

W Bradley Knox and Peter Stone. Tamer: Training an agent manually via evaluative reinforcement. In *2008 7th IEEE International Conference on Development and Learning*, pages 292–297. IEEE, 2008.

W Bradley Knox and Peter Stone. Interactively shaping agents via human reinforcement: The tamer framework. In *Proceedings of the fifth international conference on Knowledge capture*, pages 9–16, 2009.

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

Rajesh Kumar, Smriti Srivastava, JRP Gupta, and Amit Mohindru. Comparative study of neural networks for dynamic nonlinear systems identification. *Soft Computing*, 23(1): 101–114, 2019.

Michael Laskey, Caleb Chuck, Jonathan Lee, Jeffrey Mahler, Sanjay Krishnan, Kevin Jamieson, Anca Dragan, and Ken Goldberg. Comparing human-centric and robot-centric sampling for robot deep learning from demonstrations. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 358–365. IEEE, 2017a.

Michael Laskey, Jonathan Lee, Roy Fox, Anca Dragan, and Ken Goldberg. Dart: Noise injection for robust imitation learning. In *Conference on robot learning*, pages 143–156. PMLR, 2017b.

Ian Lenz, Ross A Knepper, and Ashutosh Saxena. Deepmpc: Learning deep latent features for model predictive control. In *Robotics: Science and Systems*. Rome, Italy, 2015.

Timothée Lesort, Natalia Díaz-Rodríguez, Jean-Franois Goudou, and David Filliat. State representation learning for control: An overview. *Neural Networks*, 108:379–392, 2018.

Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems*, pages 1071–1079, 2014.

Sergey Levine, Zoran Popovic, and Vladlen Koltun. Nonlinear inverse reinforcement learning with gaussian processes. In *Advances in neural information processing systems*, pages 19–27, 2011.

FW Lewis, Suresh Jagannathan, and Aydin Yesildirak. *Neural network control of robot manipulators and non-linear systems*. CRC press, 2020.

YuXuan Liu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Imitation from observation: Learning to imitate behaviors from raw video via context translation. *arXiv preprint arXiv:1707.03374*, 2017.

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

Dylan P Losey, Krishnan Srinivasan, Ajay Mandlekar, Animesh Garg, and Dorsa Sadigh. Controlling assistive robots with learned latent actions. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 378–384. IEEE, 2020.

Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. In *Conference on Robot Learning*, pages 1113–1132. PMLR, 2020.

Donald Michie, Michael Bain, and J Hayes-Miches. Cognitive models from subcognitive skills. *IEE control engineering series*, 44:71–99, 1990.

Hua-Qing Min, Jin-Hui Zhu, and Xi-Jing Zheng. Obstacle avoidance with multi-objective optimization by pso in dynamic environment. In *2005 international conference on machine learning and cybernetics*, volume 5, pages 2950–2956. IEEE, 2005.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Thomas M Moerland, Joost Broekens, and Catholijn M Jonker. Model-based reinforcement learning: A survey. *arXiv preprint arXiv:2006.16712*, 2020.

Andrew William Moore. Efficient memory-based learning for robot control. 1990.

Nafee Mourad, Ali Ezzeddine, Babak Nadjar Araabi, and Majid Nili Ahmadabadi. Learning from demonstrations and human evaluative feedbacks: Handling sparsity and imperfection using inverse reinforcement learning approach. *Journal of Robotics*, 2020, 2020.

Berndt Müller, Joachim Reinhardt, and Michael T Strickland. Network architecture and generalization. In *Neural Networks*, pages 93–107. Springer, 1995.

Jelle Munk, Jens Kober, and Robert Babuška. Learning state representation for deep actor-critic control. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 4667–4673. IEEE, 2016.

Richard M Murray and John Edmond Hauser. *A case study in approximate linearization: The acrobat example*. Electronics Research Laboratory, College of Engineering, University of . . . , 1991.

Anusha Nagabandi, Kurt Konolige, Sergey Levine, and Vikash Kumar. Deep dynamics models for learning dexterous manipulation. In *Conference on Robot Learning*, pages 1101–1112, 2020.

Ashvin Nair, Dian Chen, Pulkit Agrawal, Phillip Isola, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Combining self-supervised learning and imitation for vision-based rope manipulation. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2146–2153. IEEE, 2017.

Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

Monica N Nicolescu and Maja J Mataric. Natural methods for robot task learning: Instructive demonstrations, generalization and practice. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 241–248, 2003.

Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in atari games. In *Advances in neural information processing systems*, pages 2863–2871, 2015.

Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J Andrew Bagnell, Pieter Abbeel, and Jan Peters. An algorithmic perspective on imitation learning. *arXiv preprint arXiv:1811.06711*, 2018.

Yunpeng Pan, Ching-An Cheng, Kamil Saigol, Keuntaek Lee, Xinyan Yan, Evangelos Theodorou, and Byron Boots. Agile autonomous driving using end-to-end deep imitation learning. *arXiv preprint arXiv:1709.07174*, 2017.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 16–17, 2017.

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

Rodrigo Pérez-Dattari, Carlos Celemin, Javier Ruiz-del Solar, and Jens 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.

Athanasios S Polydoros and Lazaros Nalpantidis. A reservoir computing approach for learning forward dynamics of industrial manipulators. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 612–618. IEEE, 2016.

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

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

Stephane Ross and J Andrew Bagnell. Agnostic system identification for model-based reinforcement learning. *arXiv preprint arXiv:1203.1007*, 2012.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

Yoshiaki Sakagami, Ryujin Watanabe, Chiaki Aoyama, Shinichi Matsunaga, Nobuo Higaki, and Kikuo Fujimura. The intelligent asimo: System overview and integration. In *IEEE/RSJ international conference on intelligent robots and systems*, volume 3, pages 2478–2483. IEEE, 2002.

Camille Salaün, Vincent Padois, and Olivier Sigaud. Learning forward models for the operational space control of redundant robots. In *From motor learning to interaction learning in robots*, pages 169–192. Springer, 2010.

Claude Sammut, Scott Hurst, Dana Kedzier, and Donald Michie. Learning to fly. In *Machine Learning Proceedings 1992*, pages 385–393. Elsevier, 1992.

Joe Saunders, Chrystopher L Nehaniv, and Kerstin Dautenhahn. Teaching robots by moulding behavior and scaffolding the environment. In *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, pages 118–125, 2006.

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

Stefan Schaal. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3(6):233–242, 1999.

Bradly C Stadie, Pieter Abbeel, and Ilya Sutskever. Third-person imitation learning. *arXiv preprint arXiv:1703.01703*, 2017.

Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

G Tevatia and S Schaal. Efficient inverse kinematics algorithms for high-dimensional movement systems. *University of Southern California*, 2008.

Jo-Anne Ting, Mrinal Kalakrishnan, Sethu Vijayakumar, and Stefan Schaal. Bayesian kernel shaping for learning control. In *Advances in neural information processing systems*, pages 1673–1680, 2009.

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

Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954*, 2018a.

Faraz Torabi, Garrett Warnell, and Peter Stone. Generative adversarial imitation from observation. *arXiv preprint arXiv:1807.06158*, 2018b.

Faraz Torabi, Sean Geiger, Garrett Warnell, and Peter Stone. Sample-efficient adversarial imitation learning from observation. *arXiv preprint arXiv:1906.07374*, 2019a.

Faraz Torabi, Garrett Warnell, and Peter Stone. Recent advances in imitation learning from observation. *arXiv preprint arXiv:1905.13566*, 2019b.

Jur Van Den Berg, Stephen Miller, Daniel Duckworth, Humphrey Hu, Andrew Wan, Xiao-Yu Fu, Ken Goldberg, and Pieter Abbeel. Superhuman performance of surgical tasks by robots using iterative learning from human-guided demonstrations. In *2010 IEEE International Conference on Robotics and Automation*, pages 2074–2081. IEEE, 2010.

Herke Van Hoof, Nutan Chen, Maximilian Karl, Patrick van der Smagt, and Jan Peters. Stable reinforcement learning with autoencoders for tactile and visual data. In *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 3928–3934. IEEE, 2016.

Garrett Warnell, Nicholas Waytowich, Vernon Lawhern, and Peter Stone. Deep tamer: Interactive agent shaping in high-dimensional state spaces. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in neural information processing systems*, pages 2746–2754, 2015.

Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Pei-Hao Su, David Vandyke, and Steve Young. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. *arXiv preprint arXiv:1508.01745*, 2015.

Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

Zhiqing Xiao. Solutions to openai gym. https://github.com/ZhiqingXiao/OpenAIGymSolution, 2020.

Bo Xiong, Fangshi Wang, Chao Yu, Fei Qiao, Yi Yang, Qi Wei, and Xin-Jun Liu. Learning safety-aware policy with imitation learning for context-adaptive navigation. 2019.

Jiakai Zhang and Kyunghyun Cho. Query-efficient imitation learning for end-to-end autonomous driving. *arXiv preprint arXiv:1605.06450*, 2016.

Marvin Zhang, Sharad Vikram, Laura Smith, Pieter Abbeel, Matthew Johnson, and Sergey Levine. Solar: Deep structured representations for model-based reinforcement learning. In *International Conference on Machine Learning*, pages 7444–7453. PMLR, 2019a.

Ruohan Zhang, Faraz Torabi, Lin Guan, Dana H Ballard, and Peter Stone. Leveraging human guidance for deep reinforcement learning tasks. *arXiv preprint arXiv:1909.09906*, 2019b.

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