UNIVERSITY OF AMSTERDAM

UvA-DARE (Digital Academic Repository)

Detaching the strings

Practical algorithms for Learning from Demonstration

Shiarlis, K.C.

Publication date
2019

Document Version
Final published version

License
Other

Link to publication

Citation for published version (APA):
Shiarlis, K. C. (2019). *Detaching the strings: Practical algorithms for Learning from Demonstration*. [Thesis, fully internal, Universiteit van Amsterdam].

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

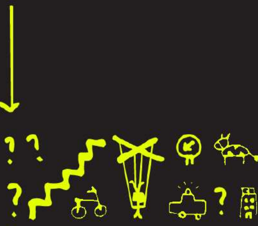
Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: https://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.

UvA-DARE is a service provided by the library of the University of Amsterdam (https://dare.uva.nl)

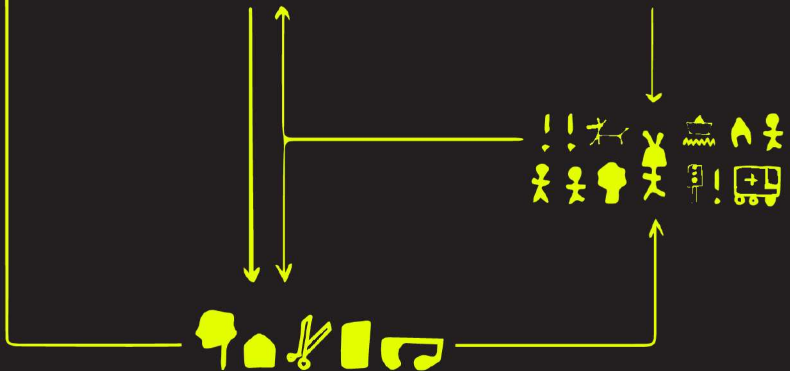
Download date:09 Mar 2023

DETACHING THE STRINGS:

</ PRACTICAL ALGORITHMS FOR LEARNING FROM DEMONSTRATION />



$$P(\tau \; a_p \;\; s_p) = \sum_{z \in Z_{\tau,v}} P(\; | s_p) \prod_{t=1}^{T} \pi_{\theta_{z_t}}(a_t | s_t).$$



KYRIAKOS C. SHIARLIS

Detaching the strings: Practical algorithms for Learning from Demonstration.

Kyriakos Christoforos Shiarlis

Detaching the strings: Practical algorithms for Learning from Demonstration.

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. ir. K.I.J. Maex
ten overstaan van een door het College voor Promoties ingestelde
commissie, in het openbaar te verdedigen in
de Agnietenkapel
op donderdag 5 september 2019, te 14:00 uur

door

Kyriakos Christoforos Shiarlis

geboren te Egkomi

Promotiecommissie

Promotores:

	Prof. dr. M. Welling	Universiteit van Amsterdam
	Prof. dr. S. A. Whiteson	University of Oxford

Overige leden:

	Prof. dr. A. Nowé	Vrije Universiteit Brussel
	Prof. dr. C. G. M. Snoek	Universiteit van Amsterdam
	Prof. dr. T. Gevers	Universiteit van Amsterdam
	Dr H.C van Hoof	Universiteit van Amsterdam
	Dr. F. A. Oliehoek	TU Delft

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

The research was supported by the European Research Council FP7 program, under project number 611153.

Copyright © 2019 Kyriacos Shiarlis, Amsterdam, The Netherlands
Cover by Efi Zeniou
ISBN: 978-94-6375-512-2

Acknowledgements.

First and foremost I would like to thank *Joao Messias*. His tremendous patience, wide-spanning knowledge, inspiring dilligence and all-round competence, have educated me in ways that trancent the technical details and have forged me as a scientist and problem solver. He is the best teacher I ever had and I sincerely hope the best for him and his wonderful family. Equally influential in this journey was my supervisor *Shimon Whiteson*. Amongst many things, Shimon taught (or rather forced) me to explain complex ideas succintly, and with a balanced supervisory style taught me how to work independently and argue critically. Whenever I argue an idea, I always try to simulate his reaction in my head.

In these four years I was also blessed with great collaborators. From the UPO group in Seville, Luis Merino, Noe-Perez Higuieras, Rafael Ramon-Vigo, Ignacio Perez-Hurtado, and Fernando Caballero, with which endless hours were spent hacking away on the TERESA robot in various locations around Europe. From the Robotics Institute in Oxford, Markus Wulfweier, Sasha Salter and Ingmar Posner with which I was very lucky to collaborate at the end of my PhD. Thank you Ingmar for giving me this opportunity. Finally all the bright people in the Whiteson Research Lab in Oxford which welcomed me during this last year.

None of this would have been possible without continuous support from my family, Antonis, Varvara and Anna-Maria. Their love, care and percistence in good education are the very foundation, not only of this work, but of everything I have done and will do.

Despite the workload, my PhD experience was enriched by strong friendships, which I consider as precious and educational as the degree itself. Jorn, Mandy, Peter, Nicos, Iris, Nayia, Nicolas, Amanda, Vitaly, Jonas, Joost and Luisa thank you of filling my days in Amsterdam and Oxford with adventure, music, laughter and love. I could not ask for more unique, inspiring and caring people around me. The same applies to all my friends in Cyprus who have accompanied me throught the years. You know who you are.

Finally, this work was fully funded by the E.U FP7 program TERESA. I am grateful for this opportunity and hopeful that such initialives will continue to grow, as they are essential for the education of young european citizens in all fields.

Contents

1 Introduction **1**
1.1 Focus and Aims 5
1.2 Main Contributions 7
1.3 Thesis Structure 9

2 Background **11**
2.1 Agents, Policies and Demonstrations 11
2.2 Behavioral Cloning 14
2.3 Dataset Augmentation 15
2.4 Inverse Reinforcement Learning 17
2.5 Tasks 20
2.6 Summary 21

3 Inverse Reinforcement Learning from Failure **23**
3.1 Maximum causal entropy IRL 24
3.2 Method 26
3.3 Experiments 31
3.3.1 Simulated Navigation Domain 32
3.3.2 Simulated Factory Domain 35
3.3.3 Real Navigation Domain 38
3.4 Conclusions and Future Work 40

4 Rapidly Exploring Learning Trees **41**
4.1 Related Work 42
4.2 Path Planning 43
4.3 IRL for Path Planning 45
4.4 Method 46
4.4.1 Approximate Maximum Margin Planning 47
4.4.2 Rapidly Exploring Learning Trees 47

4.5 Experiments 49
4.5.1 Simulated Experiments 51
4.5.2 Real Robot Experiments 55
4.6 Discussion 57
4.7 Conclusion 58

5 Deep Behavioral Social Cloning 59
5.1 Introduction 59
5.2 Problem Statement 61
5.2.1 Group Interaction Task 61
5.2.2 Following Task 62
5.2.3 Data and Experiments 62
5.3 Method 64
5.3.1 Inputs and Convolutional Layers 64
5.3.2 Dynamics and Long Short-Term Memory 66
5.3.3 Outputs and Overall Architecture 66
5.3.4 Preventing Covariate Shift 67
5.4 Evaluation and Results 67
5.4.1 Mean Squared Error 68
5.4.2 Output Distribution 68
5.4.3 Human Evaluation 69
5.5 Discussion 71
5.6 Conclusions and future work 72

6 Temporal Alignment for Control 73
6.1 Related Work 75
6.2 Preliminaries 76
6.2.1 Modular Learning from Demonstration 76
6.2.2 Sequence Alignment 78
6.3 Methods 79
6.3.1 Naively Adapted CTC 79
6.3.2 Temporal Alignment for Control (TACO) 80
6.4 Experiments 82
6.4.1 Nav-World Domain 84
6.4.2 Craft Domain 85
6.4.3 Dial Domain 86
6.5 Discussion 89
6.6 Conclusion 90

7 Conclusions **91**

A CTC Probabilistic Sub-Policy Training **93**

Bibliography **95**

Samenvatting **103**

Summary **105**

Chapter 1

Introduction

One of the ultimate goals of artificial intelligence is to build *agents* that can act autonomously in different *environments* and perform useful *tasks*. In a physical form, agents constitute the decision making machinery of robots and the tasks include self-driving vehicles, logistics, small scale manufacturing, agriculture, museum guidance and elderly companionship. In virtual environments agents come in the form of video game AI, chat bots, virtual assistants and recommender systems. Given the potential economic an societal benefits of such an advanced state of automation, the recent resurgence in autonomous agent research comes as no great surprise.

While the applications for autonomous agents vary greatly, the main challenges are common. They are, environment complexity, task complexity and human interaction. Environment complexity refers to the fact that the environment in which the agent operates is unstructured and unpredictable. Take for example a robot navigating between humans. Since it is hard to predict where people will move next, it becomes much more difficult for the agent to make long term navigation plans. Task complexity refers to the the difficulty of the task to be performed. An example for this is dexterous manipulation, where even with perfect sensing and a deterministic environment, defining a behavior for a complex manipulation task can be very challenging. Finally we have human interaction. In this case the task itself is very hard to define because human preferences are so hard to encode numerically in a computer. Depending on the degree to which the above challenges are present in our application, there are three main paths one may take to obtain the required agent.

The first option is to define the required behavior by hand. I.e., provide a fixed set condition-action pairs to account for every possible configuration of the environment. Figure 1.1 shows a schematic of how this is achieved. This approach enjoys robustness and interpretability. For each state we know exactly what action will be taken and we

know why it was taken. Hand definition however can quickly become very tedious from the perspective of the programmer as all conditions need to be thought and accounted for. In addition the programmer needs to iterate the defined policy many times depending on whether the desired behavior is achieved with respect to the environment. This approach is ideal for large scale repetitive tasks in highly predictable environments such as car manufacturing plants. Also, while impractical for complex environments, hand definition of behaviors is the predominant choice for obtaining AI-like behavior in computer games.

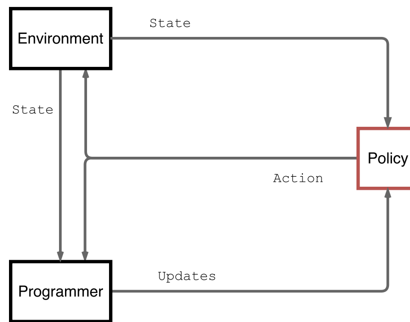


Figure 1.1: Hand definition outline: The programmer defines the policy. If a different policy is required it has to be explicitly programmed.

In some cases we know that the objective of a behavior, but defining the exact policy to achieve that objective can be avoided. In these cases we can define the objective as a *cost* or *reward* function. These are functions that give a single scalar quantity for each situation the agent finds itself, which typically denotes how favorable it is to be in that situation. We can then employ methods that automatically find a policy that minimizes the amount of cost the agent accumulates while executing it. Such methods are much more flexible than hand defining the behavior as they can effectively deal with unknown scenarios based on the basic principle of cost. For example, once we have a basic algorithm for navigating a map we can apply it to many instances without the need for someone to hand-define the behavior for every new map. If the environment dynamics are known and are not very complex one may employ principles from optimal control or decision theoretic planning [13]. Indeed, such methods are quite popular among robotics and gaming communities as they are very effective for navigation and path planning.

If the environment or task is too large and complex, AI tends to employ Machine Learning (ML). A common ML framework in autonomous agent literature is Reinforcement Learning (RL) [112] (Figure 1.2). The main principle of RL is learning by trial and error. The agent attempts different actions in different situations (states) while receiving

feedback from the environment in the form of rewards or costs. The objective of RL is to maximize the sum of rewards accumulated over many time-steps. RL has shown promising results in mastering simple video games [78], extremely complex board games such as GO [109] as well as robotic tasks using only image observations [70]. As shown in Figure 1.2 in the optimal control and RL frameworks the programmer is responsible mainly for defining the cost/reward function while the planning or learning module is responsible for figuring out the optimal policy.

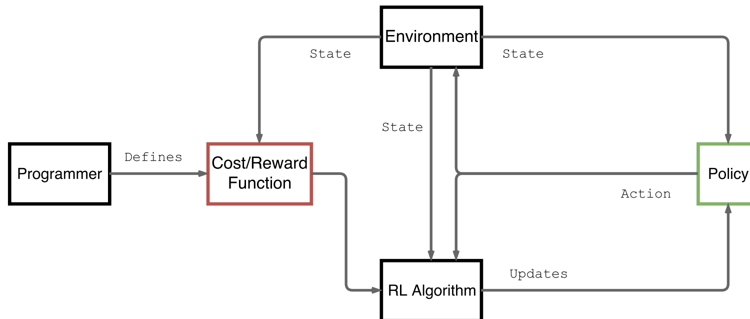


Figure 1.2: RL/optimal control outline: The programmer defines a cost function which is used by a generic algorithm to find the optimal policy. Different cost functions result in different policies, making the framework very flexible.

It could also be the case however that defining such an objective is very challenging as well. This is especially the case when agents interact with humans. Take as an example a chat bot. How does one code the purpose of a conversation with a human? Another example is navigation through human environments. How does one encode how a robot should navigate through humans? More specifically, how does one encode the value of a specific situation into a single cost value for these applications. Attempting to do so would probably require many iterations of defining the cost and learning the policy that minimizes it. This is not only tedious but also dangerous, since badly defined, complex cost functions can result in unexpected behavior when optimized. This pathology is relatively common in RL and OC literature, where a policy learns to exploit mistakes in the cost function resulting in behavior very far from the one intended [48]. At this point could even be favorable to resort to hard-coding the required behavior. This itself would not be preferred, as the lack of flexibility of the hard-coding framework would hinder the adaptability of the system to different users with varying preferences or result in unnatural, mechanical behavior .

In these cases one may employ Learning from Demonstration (LfD) [6], which is the main subject of this thesis. LfD, considers the programming of agents by observing the decisions of other agents, such as humans. Successful learning thus results in the

agent performing the actions that the human would perform in situations not seen in the data. A schematic of LfD is shown in Figure 1.3. We can see that in this case the only thing that needs to be provided by the programmer are the demonstrations and the LfD algorithm is responsible for defining the desired policy. Successful LfD avoids the two drawbacks from previous methods. There is no need to define a complex behavior by hand since the required policy is in the observed decisions. Furthermore promising frameworks within LfD such as *inverse reinforcement learning* (IRL) [80] and *inverse optimal control* (IOC) [50] have as a primary objective the extraction of reward/cost functions from demonstrations. Programming a system by LfD does not require detailed knowledge of the underlying system. The only requirement is to be able to control the agent. This makes LfD an ideal agent programming interface for non-experts. These benefits have established LfD research as an ever growing field in AI, with many successes in manipulation [30], social robotics [14], game AI [42] and sports prediction [67].

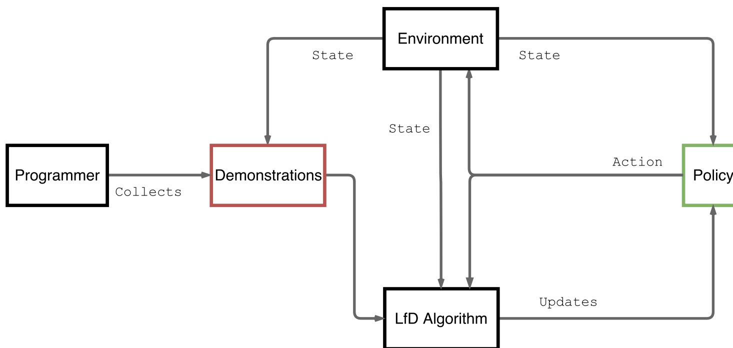


Figure 1.3: LfD outline. The programmer provides a set of demonstrations to an LfD algorithm which is responsible for finding the appropriate policy through learning. Different data result in different policies, making the framework very flexible.

Despite their promising properties LfD methods are not perfect. For example, as learning depends on demonstrations the nature and quality of these are key. In addition methods like IRL can be very computationally expensive for use even in simple setups. Finally, most existing methods focus on extracting a single policy from the demonstrated data and cannot account for the presence of various tasks within a single demonstration, hence relying on additional human labeling effort. This thesis focuses on developing practical LfD methods to overcome some of the above limitations. A large part of it considers robots in social environments and hence concerned with aspects such as social navigation and group interaction. However, a significant portion of the contributions have wider applications.

1.1 Focus and Aims

The focus of this thesis is better demonstrated through a practical LfD pipeline for a social robot. Let us assume we would like to train a robot to move around in a room populated with people, in order to reach a specific group and start conversing with them. At some point this conversation group starts moving and the robot should follow them to a new location where a new conversation starts.

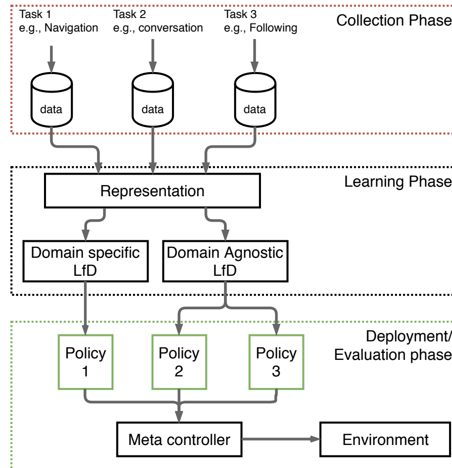


Figure 1.4: LfD pipeline. The process begins with obtaining useful data for each of the tasks we would like our system to be able to solve. Once a suitable state representation is chosen for each task we can either use domain specific or domain agnostic LfD algorithms to obtain a policy. The policies for each task are combined using a meta controller which invokes the learned policies in the environment.

Figure 1.4 depicts an LfD pipeline which proceeds in a similar fashion to many in Machine Learning. In the collection phase we collect or obtain data for the separate tasks that we would like to learn. In the example above we have three tasks. Navigation, group conversation and group following. Each task requires different policies. For example while navigating the robot should avoid people in a socially normative way, in contrast while following we would like it to stay at a safe distance behind the group.

Once the data is collected we enter the learning phase. The first step here is to decide on what state representation should be used for each task. A state representation typically encodes what aspects of the environment we are interested in. This may take the form of either a fixed hard-coded feature representation or some kind of learned feature extraction mechanism such as a neural network. Learning policies can be done by means of either specialized or generic LfD algorithms. For example navigation is a widely studied problem in robotics so one might like to leverage the large body of

knowledge around the subject. Learning in this case outputs the required policies for each task.

Once policies are learned, we enter the deployment phase where we can invoke each one in sequence when required. This invocation typically happens by means of a meta-controller which can be either learned or defined by hand. The details of this meta-controller are however beyond the scope of this thesis.

The questions asked in this thesis can be posed with respect to this pipeline as follows:

1. During data collection some of the demonstrations will be of bad quality. For example we might collide with someone by mistake. Can we leverage these demonstrations to learn what *not* to do?

2. When learning how to navigate, could we learn the robotic planner? I.e., can we learn the cost/reward functions typically employed in robotic planners? Can we include kinematic constraints usually encountered in robotics to improve the learned behavior?

3. During data collection and deployment, the robot's sensors will be noisy. How can we ensure learning is robust to this noise?

4. During interaction, a varying number of people will be around the robot. How do we deal with varying length inputs in the specific application?

5. Instead of collecting separate demonstrations for each task, could we collect demonstrations with mixtures of tasks, e.g., have one demonstration with both navigation and group interaction behaviors in it? How can we efficiently and robustly find out what task occurs when, without needing to laboriously segment label the whole dataset?

From the above we can see that this thesis tackles various problems in LfD. The overarching theme however is an attempt to improve the applicability of the framework to real robot environments through either general, domain agnostic algorithms (questions 1 and 5) or through improved modeling of the specific application environment (questions 2-4).

1.2 Main Contributions

In response to the questions posed in Section 1.1 this thesis makes the following contributions.

Inverse Reinforcement Learning from Failure (IRLF)
IRL provides a principled framework for discovering goals from observed behavior and allows autonomous agents to solve complex tasks from successful demonstrations. However, in many settings, e.g., when a human learns the task by trial and error, failed demonstrations are also readily available. In addition, in some tasks, purposely generating failed demonstrations may be easier than generating successful ones. Since existing IRL methods cannot make use of failed demonstrations, in this chapter we propose inverse reinforcement learning from failure (IRLF) which exploits both successful and failed demonstrations. Starting from the state-of-the-art maximum causal entropy IRL method, we propose a new constrained optimization formulation that accommodates both types of demonstration. We then derive update rules for learning reward functions and policies. Experiments on both simulated and real-robot data demonstrate that IRLF converges faster and generalizes better than maximum causal entropy IRL, especially when few successful demonstrations are available. The work in this chapter appears in the following publication:

Kyriacos Shiarlis, Joao Messias, and Shimon Whiteson. ”Inverse reinforcement learning from failure.” *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems. International Foundation for Autonomous Agents and Multiagent Systems, 2016.* [105]

Rapidly exploring learning trees (RLT*)
IRL can be used for path planning, enabling robots to learn cost functions for difficult tasks from demonstration, instead of hard-coding them. However, IRL methods face practical limitations that stem from the need to repeat costly planning procedures. In this chapter, we propose Rapidly Exploring Learning Trees (RLT), which learns the cost functions of Optimal Rapidly Exploring Random Trees (RRT) from demonstration, thereby making inverse learning methods applicable to more complex path planning tasks. Our approach extends Maximum Margin Planning to work with RRT cost functions. Furthermore, we propose a caching scheme that greatly reduces the computational cost of this approach. Experimental results on simulated and real-robot data from a social navigation scenario show that RLT achieves better performance at lower computational cost than existing methods. We also successfully deploy control

policies learned with RLT on a real telepresence robot. This chapter is based on:

Kyriacos Shiarlis, Joao Messias, and Shimon Whiteson. ”Rapidly exploring learning trees.” *Robotics and Automation (ICRA), 2017 IEEE International Conference on. IEEE, 2017.* [107]

Acquiring Social Interaction Behaviours for Telepresence Robots via Deep Learning from Demonstration (DBSoC)

As robots begin to inhabit public and social spaces, it is increasingly important to ensure that they behave in a socially appropriate way. However, manually coding social behaviors is prohibitively difficult since social norms are hard to quantify. Therefore, LfD, is a powerful tool for helping robots acquire social intelligence. In this chapter, we propose a deep learning approach to learning social behaviors from demonstration. We use an image based representation of the robots social state in order to cope with varying amounts of people around the robot. Furthermore, our architectural choices allow the robot to effectively deal with sensor imperfections during learning and deployment. We apply this method to two challenging social tasks for a semi-autonomous telepresence robot. Our results show that our approach outperforms gradient boosting regression and performs well against a hard-coded controller. Furthermore, ablation experiments confirm that each element of our method is essential to its success. This work in appeared in the following publication:

Kyriacos Shiarlis, Joao Messias, and Shimon Whiteson. ”Acquiring social interaction behaviours for telepresence robots via deep learning from demonstration.” *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on. IEEE, 2017.* [106]

Learning Task Decomposition via Temporal Alignment for Control (TACO)

Many advanced LfD methods consider the decomposition of complex, real-world tasks into simpler sub-tasks. By reusing the corresponding sub-policies within and between tasks, they provide training data for each policy from different high-level tasks and compose them to perform novel ones. Existing approaches to modular LfD focus either on learning a single high-level task or depend on domain knowledge and temporal segmentation. In contrast, this chapter proposes a weakly supervised, domain-agnostic approach based on task sketches, which include only the sequence of sub-tasks performed in each demonstration. Our approach simultaneously aligns the sketches with the observed demonstrations and learns the required sub-policies. This improves generalization in comparison to separate optimization procedures. We evaluate the approach on multiple

domains, including a simulated 3D robot arm control task using purely image-based observations. The results show that our approach performs commensurately with fully supervised approaches, while requiring significantly less annotation effort. This chapter is based on the following publication:

Kyriacos Shiarlis, Markus Wulfmeier, Sasha Salter, Shimon Whiteson, and Ingmar Posner. ”TACO: Learning Task Decomposition via Temporal Alignment for Control.” *International Conference on Machine Learning. 2018.* [108]

1.3 Thesis Structure

In this Chapter we motivated LfD, discussed some of the issues that are usually encountered when applying the framework in the real world and discussed the research questions that this thesis tackles. We also provided more detail as to how the individual chapters tackle the aforementioned questions. Chapter 2 serves as both a conceptual and technical introduction to the methods employed and extended by this thesis. The ideas introduced are common to all the succeeding chapters. When this is not the case however, i.e., if a chapter utilizes specific methods and ideas, these are discussed in the first sections of that chapter. Chapter 3 describes Inverse Reinforcement Learning from Failure (IRLF) a method that allows agents what *not* to do from demonstrations. Chapter 4 describes the Rapidly Exploring Learning Tree (RLT*) algorithm. This is an algorithm that allows learning cost functions for sample based planners typically used in robotic applications such as navigation and manipulation. Chapter 5 describes an application of LfD and deep learning to social robotics. This chapter is a nice example of the practical considerations required for real world applications and how LfD can be used to program different behaviors very easily and with minimal technical knowledge. Chapter 6 describes Temporal Alignment for Control (TACO). We consider cases where demonstrations contain execution of various sub-tasks and are accompanied by a weak supervisory signal, a sketch, describing what these sub-tasks are. TACO simultaneously aligns the demonstrations with the sketch and learns the required policies. Chapter 7 concludes the thesis with general comments about the algorithms and ideas discussed. It also provides an overview of the frontiers and interesting emerging ideas in LfD. Since this thesis is concerned with various problem settings for LfD, the related work for each chapter can be quite distinct. An overview of the existing work for each problem considered can be found in the respective chapter.

Chapter 2

Background

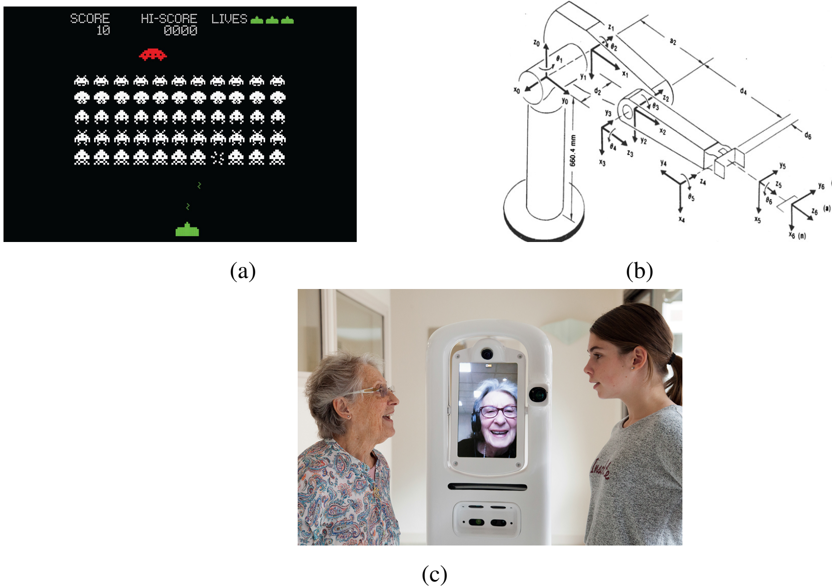
In this chapter we introduce the basic principles of Learning from Demonstration (LfD). We focus primarily on the methods employed and extended by this thesis. Specifically we consider a state-action abstraction of the demonstration [84], as opposed to a trajectory based abstraction [88] [47]. A more detailed overview of the LfD field can be found in survey papers by Argall et al. [6], and Billard et al. [9].

The three main frameworks we consider are behavioral cloning (BC), dataset aggregation (DA) and inverse reinforcement learning (IRL). Behavioral cloning considers LfD as a supervised learning problem. This treatment causes several problems that DA algorithms attempt to alleviate by teaching the agent to recover from errors. IRL poses the problem from the perspective of reinforcement learning (RL). We project BC, DA and IRL on a common framework and discuss their advantages and disadvantages, as well as how they are employed in this work.

2.1 Agents, Policies and Demonstrations

The environment in which an agent is acting is typically modeled through a *Markov decision process* (MDP), a discrete-time process wherein an agent's actions may stochastically influence its environment. In an MDP, at step t, the system (which includes the agent and its environment) is known to be in a *state* $s_t \in \mathcal{S}$; the agent selects an action $a_t \in \mathcal{A}$ and is awarded a real-valued *reward*; and the system jumps to state s_{t+1} with probability $P(s_{t+1}|s_t, a_t)$. Formally, an MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$, where $\mathcal{S}$ and $\mathcal{A}$ are sets of states and actions respectively, both of which can be discrete or continuous. $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is a transition function such that $\mathcal{T}(s, a, s') = P(s'|s, a)$, and $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function. The MDP framework is extremely powerful because it is general. Figure 2.1 shows examples of environments typically modeled

as MDPs. The states, actions and transitions for each of the depicted domains are completely different. For example in an Atari game, like Space Invaders, states are represented as pixels and actions as discrete controller decisions. When controlling a robot arm (Figure 2.1b) one may model the state as the joint angles of the manipulator or their position in 3D space, both of which are continuous vectors. In a social interaction scenario (Figure 2.1c) the state can be the position and pose of people around the robot. In such a domain actions can be both continuous, e.g., velocity, or discrete, e.g., calling somebody. Rewards in MDPs can be equally diverse. For a game a reward is the score of the game whereas for a manipulator it can be the precision and smoothness of its actions or the distance from a goal configuration.



(a) (b)

(c)

Figure 2.1: Different state representations. (a) The state for an Atari game can be the pixels in the screen. (b) For a manipulator the state includes continuous quantities of the joint angle and positions. (c) Social robot states typically encode the positions of people around the robot.

An agent acting in such an environment does so through a *policy* $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ with $\pi(s, a) = P(a \,|\, s)$, i.e., the probability of taking an action in a certain state. Finding a "good" policy is typically the purpose of solving the MDP and the definition of "good" depends on the setting we are considering. If a reward function is provided or can be designed, then we seek a policy that maximizes the expected sum of rewards over a fixed number of decisions T. This expectation can be expressed as the *value* of

policy π:

$$V^{\pi}(s) := \mathbb{E}\{\sum_{t=1}^{T} R(s_t, a_t) \mid s_1 = s\}. \quad (2.1)$$

Here the expectation in taken with respect to all the possible paths of length T, induced by $\mathcal{T}(s, a, s')$ and π both of which are typically stochastic. In the special case where both are deterministic, there is only a single path that that results from a policy at that state and the expectation collapses to a single point.

Planning in an MDP attempts to find a policy that maximizes the value for each possible state. This is typically requires a step of *policy evaluation*, where the value function of the current policy is determined, followed by a step of *policy improvement* where the policy is changed to increase the value of the required states. Due to the Markovian and sequential nature of the MDP model, planning can be performed using dynamic programming techniques.

If the number of states in the MDP is large, e.g., if the state space is continuous, or the transition function is unknown, planning cannot be easily employed to solve the MDP. In these cases we resort to Reinforcement Learning (RL). Instead of enumerating all possible states and paths to compute the value function and subsequently optimize it, RL learns the optimal policy by collecting data from its interaction with the environment.

There are two main approaches to RL, on-policy methods and off-policy methods. On-policy methods, which include the majority of policy gradient [121] and actor-critic algorithms, learn value of the policy being used and update it until it is optimal [122]. Off-policy methods such as Q-learning, can learn the optimal policy using data samples that may originate from any policy, e.g., a random policy. RL is a very general and powerful tool for decision making and a large body of work has resulted in many successes in the field. Despite significant advancements, RL is still very hard to perform in the real world and especially continuous environments. In such cases most of the success has been concentrated in related methods which make heavy use of physical models and more often than not, tend to fall under the field of optimal control [126] and trajectory optimization [97]. Despite their differences, all of the above methods have one common assumption, the presence of a reward function.

LfD considers the case where a $R(s, a)$ is not present (or it is difficult to design). This results in an MDP/R defined as $\langle \mathcal{S}, \mathcal{A}, T \rangle$. In place of a reward function we receive a dataset of N demonstration trajectories $\mathcal{D} = \{\rho_1, \rho_2, ...\rho_N\}$. Each trajectory $\rho_i = \langle (s_1, a_1), (s_2, a_2), \ldots, (s_{T_\rho}, a_{T_\rho}) \rangle$ is a state-action sequence of length T_ρ generated by a demonstrator or *expert* which we denote with the symbol E. The name expert here refers an implicit maximization of some latent personal reward function. The goal of LfD is to find a policy that matches the behavior seen in the demonstrations and generalizes well to unseen states. A helpful measure that allows us to quantify how

good a policy is in terms of LfD is the *state occupancy* measure of a policy. Let $d_t^\pi(s)$ be the probability that the agent is at state s at time t induced by following policy π and the system dynamics. This implies the presence of an average state occupancy distribution over T time-steps,

$$d^\pi(s) = \frac{1}{T}\sum_{t=1}^{t=T} d_t^\pi(s). \tag{2.2}$$

The states and actions in the dataset D can thus be seen as samples from this distribution and LfD as an attempt to match this distribution. Different families of LfD algorithms attempt to do so in different ways.

2.2 Behavioral Cloning

BC algorithms approach LfD as a supervised learning problem, by optimizing a policy π to maximize the likelihood of the actions in the training dataset. Let $\pi_\theta(a|s)$ be the probability of taking action a in state s as modeled by a policy π_θ parameterized by θ. BC performs the following optimization:

$$\theta^* = \operatorname*{argmax}_\theta \mathbb{E}_{\rho\sim\mathcal{D}}[\sum_{t=1}^{T} \log \pi_\theta(a_t|s_t)]. \tag{2.3}$$

In other words BC attempts to perform similar actions as those in the demonstrations conditioned on the states observed. The hope is that given enough data, the learning algorithm would be able to generalize by performing the correct action given new situations. The above formulation is very simple to implement and benefits from the fact that supervised learning is a widely studied field with many options of representing and optimizing the policy at hand.

However, along with its benefits BC inherits all the assumptions from supervised learning, including the fact that the samples are *independent* and *identically distributed*. This is of course is not true in the decision making setting as states are connected together through the policy and the system dynamics. Agents typically take decisions in a sequential manner over many time-steps which in turn means that past actions affect the future states that the agent will encounter. The possible non-deterministic nature of the environment as well as small errors in the actions means that the agent may quickly find itself in a different state distribution as the one encountered in the data. A classic example of this failure case is the 'racetrack' problem. In this setting, an agent is trained on data of an expert driving around a track. As errors in the learned policy begin to compound the agent finds itself very close to the edge of the track. Since the agent

has no notion of crashing and its catastrophic effects, it hits the wall of the track. In the literature this pathology is referred to as the *compounding errors* or *covariate shift* problem.

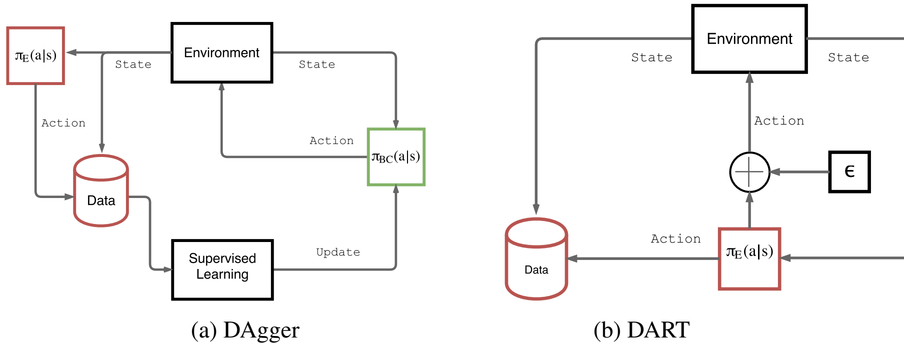
Another way to look at this problem is in terms of $d^t_\pi(s)$. BC expects to observe the state distribution encountered in the data. Instead the learned policy induces a time varying distribution which does not guarantee that the agent will find itself in the next state in the data for which it has learned a good policy. This results in the learned policy potentially occupying a completely different average distribution $d_\pi(s)$. This pathology is further aggravated by the tendency of BC to overfit to the data. In fact cite [99] have quantified the degree of this drift. Under specific circumstances and assuming a (hidden) expert reward $R^E(s,a)$ the difference of the value of the expert policy V^{π^*} that of the learned policy $V^{\pi_{bc}}$ grows quadratically in time. For more quantification of covariate shift see [100], [64].

Another potential drawback of BC that sources from supervised learning is the mode averaging problem. This problem is especially serious in the case of continuous action spaces, and sources from the fact that demonstrations might contain different actions for the exact same state. This means that we might be trying to model a multimodal distribution with a unimodal one which would lead the agent in learning a completely wrong action at the specific state. Even if we model the policy as a multimodal distribution, in continuous action spaces, decisions are taken quickly and have a small effect on the states for a single time-step. The overall effect of sampling from this multimodal policy would thus be very similar to that of a single mode. In this way, the mode averaging problem further aggravates the covariate shift problem from above.

In this work we employ BC for its simplicity. To alleviate the covariate shift problem we employ a set of methods for dataset augmentation.

2.3 Dataset Augmentation

Dataset augmentation (DA) techniques take into account the problems encountered during deployment of policies generated by BC. This is typically done by collecting diverse data, with greater coverage of the state distribution in order to teach the BC agent to recover from errors. In this thesis we employ two methods for DA, *dataset aggregation* (DAgger) and *disturbances for augmenting robot trajectories* (DART). DAgger considers an interactive scenario where the most recent π_{BC} is deployed on the environment. At every time-step the agent samples from a binomial distribution with parameter p to decide whether to query an expert or not. If the expert is queried s/he provides the correct action for the current state and the dataset is augmented.



(a) DAgger (b) DART

Figure 2.2: Dataset augmentation methods. (a) DAgger augments the dataset by querying an expert for the right action in states that the learner encounters during deployment. (b) DART collects a dataset with greater coverage by adding noise to the actions executed by the expert (or the transitions in the environment). Both methods aim to teach the agents how to recover from errors in the decision making process.

Alternatively one may deploy π_{bc}, collect execution trajectories and label each state encountered in these trajectories with the correct actions in a post-hoc manner. The effect of DAgger is that the agent is allowed to drift from the trajectory it should have taken and into the new distribution at which point the correct responses to the encountered states are recorded. This provides new data where they are really needed. By doing so DAgger not only informs the agent on how to recover from errors, but prevents further errors and distributional change. We use a variant of the method in Chapter 5 in order to augment the dataset of a social robot interacting with groups of people. While this is in principle a simple and effective algorithm, using a human teacher who is able to intervene with the correct action at each time-step can be quite hard to perform for real world agents. Furthermore in a lot of LfD applications such as computer games, we cannot always query someone for interactive feedback to augment the dataset. This would be the case for example if the data was collected by professional online players. In many cases, it is therefore better to take into account the distributional change during data collection.

DART is an alternative method for DA that is in principle easier and safer to implement than DAgger [64]. The key feature of DART is to inject noise in the environment while data is being collected. This can be done easily by adding random noise to the actions taken by the demonstrator. This in turn brings the demonstrator to states that would not otherwise be encountered. This means that the agent is essence collecting actions that allow it to recover from small errors during deployment. DART can also be trained interactively through iterations of learning and data collection. By doing so one can quantify the amount of covariate shift and adjust the amount

of noise injected in the demonstrations. Figure 2.2b details DART. Note that while noise is injected in the actions, this is done so to bring about different transitions in the environment and broaden the state coverage of the data. The action that is actually recorded at each time-step is however noiseless. DART is effective and easy to implement. However, the more principled interactive setting of the algorithm can be quite tedious for real world scenarios. We use DART in Chapter 6 in order to perform robust imitation on a 7 degree of freedom manipulator.

DA is an effective framework for dealing with covariate shift but as we have seen, most of the times this requires the algorithms to be interactive. This is because one needs to asses the amount of covariate shift that takes place after learning and provide the right augmented dataset. In addition DA methods are not very useful if the dataset has already been collected and cannot be augmented. This is the case for example in demonstrations coming from computer games. In such cases a more appropriate way to achieve robust imitation learning is Inverse Reinforcement Learning.

2.4 Inverse Reinforcement Learning

Instead of treating the LfD problem as in a supervised learning fashion, IRL attempts to the recover the reward function missing from the MDP/R from the dataset $\mathcal{D}$. The problem was first formulated as such in [102] although [50] posed the problem in an optimal control setting, by asking 'When is an observed system optimal?'. IRL is an extremely interesting task for autonomous agent research for two reasons. First, recovering the reward function of agents acting in time can be useful in predicting what they will do in the future. This has applications in areas such as robotics, computer games, advertising and recommender systems where it is also known as *preference elicitation* [101]. For the purposes of this thesis however IRL is interesting because it allows us to recover, from the dataset D, the reward function we are missing from the MDP/R. Optimizing that reward function we can then find a policy π that mimics the demonstrated policy π_E.

Though initially unknown, the reward function can be represented as a linear combination of K *feature functions*, $\phi_k(s, a)$:

$$R(s, a) = \sum_{k=1}^{K} w_k \phi_k(s, a), \tag{2.4}$$

where $w = [w_1 \, w_2 \, \dots \, w_k]^T$ is the weight vector that IRL aims to learn and $\phi_k(s, a)$ are usually pre-defined.

Since w is independent of states and actions, (2.1) and (2.4) imply a parametric

form of the value function:

$$V^{\pi}(s) = \sum_{k=1}^{K} w_k \left(E\{\sum_{t=1}^{T} \phi_k(s_t, a_t) \,|\, s_1 = s\} \right) \tag{2.5}$$

$$=: \sum_{k=1}^{K} w_k \mu_k^{\pi,1:T}|_{s_1=s}, \tag{2.6}$$

where $\mu_k^{\pi,1:T}|_{s_1=s}$, the *feature expectation*, is the expected accumulation of instances of feature ϕ_k between steps 1 and T under policy π given that $s_1 = s$. The step indices are omitted for simplicity when this expectation is over the full horizon $\{1, \ldots, T\}$.

Given $\mathcal{D}$, the learner can compute the *empirical feature expectation*, the average accumulated instances of each feature in $\mathcal{D}$:

$$\widetilde{\mu}_k^{\mathcal{D}} := \frac{1}{N} \sum_{\tau \in \mathcal{D}} \sum_{t=1}^{T_\rho} \phi_k(s_t^\tau, a_t^\tau). \tag{2.7}$$

Note that $\widetilde{\mu}_k^{\mathcal{D}}$ is state independent and implicitly estimates an expectation across the expert's initial state. To fairly compare the feature expectation of some policy π to $\widetilde{\mu}_k^{\mathcal{D}}$, we obtain an analogously state-independent measure of π's feature expectation by marginalizing out s_1:

$$\mu_k^{\pi}|_{\mathcal{D}} := \sum_{s \in \mathcal{S}} P_{\mathcal{D}}(s_1 = s) \mu_k^{\pi}|_{s_1=s}, \tag{2.8}$$

where $P_{\mathcal{D}}(s_1 = s) = N_1(s)/N$ is the maximum likelihood estimate of the expert's initial state distribution, given that $N_1(s)$ is the number of trajectories with $s_1 = s$. An interesting and enlightening property of the feature expectation is that assuming that the state-action space is finite and that the features are delta functions centered at each state action pair, they reduce to the unnormalized state occupancy measure d^T. Typically however state action spaces can be very large and complex, so one would attempt to engineer features that allow a general reward function to be learned. For example, if we have an agent navigating to a goal we may use the distance to the goal as this feature, instead of using a feature for each position in the room.

The learner aims to find w such that the value of the demonstrated trajectories is higher than the value of any policy within a set Π. This can be seen as trying to find the saddle point of the following optimization problem:

$$\max_{w} \min_{\pi \in \Pi} \quad w^T(\widetilde{\mu}^{\mathcal{D}} - \mu^{\pi}|_{\mathcal{D}}). \tag{2.9}$$

This objective implicitly encodes the fact that we are seeking a reward function that

makes the value of the demonstrations greater or equal than any optimal policy.

$$V^D(s) \geq V^\pi(s) \quad \forall \pi \in \Pi \quad \forall s \in (s) \tag{2.10}$$

It turns out that the solution to IRL is a weight vector that minimizes distance between the vectors $\widetilde{\mu}^\mathcal{D} = [\widetilde{\mu}_1^\mathcal{D} \dots \widetilde{\mu}_k^\mathcal{D}]^T$ and $\mu^{\pi^*}|_\mathcal{D} = [\mu_1^{\pi^*}|_\mathcal{D} \dots \mu_k^{\pi^*}|_\mathcal{D}]$ according to some metric. IRL thus seeks to match the occupancy measure of the policy with the one observed in the data. This is very different objective from BC where we simply match the actions seen in the data. To achieve this IRL methods are typically iterative. An initial guess for w is fed to a planning or reinforcement learning algorithm, which produces a policy π^* that is optimal given w. This is the inner minimization in equation (2.9). Using π^*, trajectories are generated from the same initial states as in $\mathcal{D}$ to compute $\mu^{\pi^*}|_\mathcal{D}$, which is then compared to $\widetilde{\mu}^\mathcal{D}$. The weights are then updated to maximize the distance between the two values, which performs the outer maximization of (2.9). A trivial way of obtaining the weight update is gradient descent:

$$w = w + \alpha(\widetilde{\mu}^\mathcal{D} - \mu^{\pi^*}|_\mathcal{D}) \tag{2.11}$$

The process (Figure 2.3) repeats until convergence, at which point we hopefully recover the reward function and match the distribution in the data. It turns out however that a naive implementation of the above optimization is unlikely to be successful, as there are many reward functions (weights) that can achieve the required constraint. Different algorithms place different additional constraints and regularization in order to improve performance and make IRL appropriate for different applications [98],[128]. In this thesis we extend two existing IRL algorithms, namely maximum causal entropy IRL and maximum margin planning. We consider the specific details of these algorithms in their respective chapters.

IRL does not suffer from covariate shift. Intuitively IRL tries to place high rewards at the state-action pairs observed by the data and low rewards everywhere else. This means that policies that result in trajectories that drift away from the observed data will be discarded during the inner optimization loop that results in π^*. The capability of IRL to find a policy that matches the data distribution relies on a model of the environment, during planning, or interaction with the environment itself, during RL. As mentioned earlier both planning and RL are potentially expensive and non trivial to perform in real world environments. This has limited the real world applicability of IRL. Despite this, it remains a highly active field of research with constant improvements to existing methods. A significant assumption that one may lift is that of the reward function being a linear function of hand-crafted features. In modern methods, reward functions can be represented using neural networks [123], Gaussian processes [69] and other non-linear

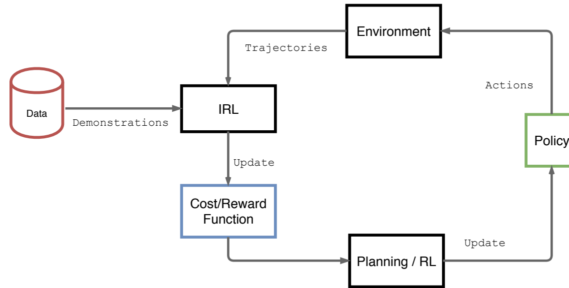


Figure 2.3: Inverse Reinforcement Learning: A reward function is input to a planning or RL module which produces a policy. This policy is deployed in the environment producing some trajectories. These are compared with the demonstrations from the data. This comparison updates the reward function in an attempt to make the generated trajectories similar to the demonstrated ones.

methods [95]. There is also a very interesting connection between IRL and generative adversarial networks, [35], [43], [27] that has recently allowed learning to match the occupancy distribution without a need for an explicit reward function.

While in this thesis we only consider linear reward functions, such representational extensions to the reward function are applicable to the introduced algorithms.

2.5 Tasks

Autonomous agents are designed to perform specific *tasks*. For Space Invaders, the task is to kill all the aliens in the screen while avoiding being hit. For a manipulator a task can be to pick an object and place it in a goal location. For a social robot, to follow somebody, or to navigate through a crowd. There are two main reasons to adopt a task based perspective in autonomous agent research, *multi-task learning* [16] and *hierarchical learning* [113]. Multi-task learning also related to meta-learning [115] or transfer learning [92], aims to design systems, (algorithms, representations, pipelines), that somehow readily generalize to new similar tasks, rather than similar instantiations of the same task. For example an algorithm that learns to pick and place a light cube, should be able to quickly learn to pick up a heavier cube, without needing to learn the whole task from scratch. The key idea is to design the training procedure or state representation of the agent such that such generalization can take place. A trivial way to do this is to use the policy learned from one task to initialize a policy for another, hoping that this would speed up learning. More complicated training schemes under the umbrella of meta-learning, consider a training procedure consisting of many tasks and combining the training information provided during learning, to learn policies that can

explicitly adapt quickly to a new unseen tasks [29].

Hierarchical learning procedures on the other hand consider the composition of sub-tasks and the learning of simple sub-policies. These policies can then be combined by a meta-controller to achieve a larger more complex task. Consider as an example a tower stacking task for a robotic manipulator. One may attempt to learn how to perform the whole task from scratch, but it is obvious that it can be broken down into multiple pick-and-place sub-tasks with only slight variations. The meta-controller in this case would have the task of executing each pick and place sub-policy in sequence, with possibly the additional responsibilities of deciding when each one is finished and what the input arguments to the next one should be. Hierarchical learning frameworks include *options* [113] in RL and symbolic planning [58] in robotics. In Chapter 6 we develop a method for hierarchical LfD that learns to break down demonstrations into sub-tasks and learn the required sub-policies using weak supervision. This enables the agent to perform zero-shot learning from demonstration, i.e., perform new unseen tasks without additional data or training.

2.6 Summary

In this chapter we introduced three main families of LfD algorithms employed and extended in this thesis. These are, behavioral cloning, dataset aggregation and inverse reinforcement learning. Each method makes different assumptions about the data, and the environment in which the agent acts. In turn these assumptions bring about different benefits and drawbacks for each. In the chapters that follow build upon these principles to introduce practical algorithms and deploy them in practical real world settings.

Chapter 3

Inverse Reinforcement Learning from Failure

In the previous chapter we introduced inverse reinforcement learning (IRL) [80], where an agent aims to learn a policy for acting in an environment for which the reward function is not available, but successful demonstrations provided by an *expert* are given instead. An IRL algorithm tries to find a reward function that leads the agent to behave similarly to the expert while generalizing well to situations for which expert data is not available.

Numerous IRL methods have been proposed. In [98] a structured prediction formulation using maximum margin is developed and applied to mobile robotics. In [94], a Bayesian formulation is proposed along with approximations necessary for computational tractability. Other work considers nonlinear representations of the reward function using Gaussian processes [69] or decision trees [95]. IRL has also been applied to a wide range of applications, from autonomous driving [1, 62] to socially appropriate navigation [41, 119] and training parsers for natural language processing [79].

Existing IRL algorithms learn only from successful demonstrations, i.e., from data gathered by an expert performing the task well. This is consistent with the main motivation of IRL since it allows learning in tasks where the reward cannot be trivially hard-coded. For example, the reward function that allows an agent to perform complicated manoeuvres while flying a helicopter cannot be trivially determined, but example demonstrations are easy to obtain from an expert.

In many realistic scenarios, failed demonstrations are also readily available. Consider for example tasks such as driving a car. Since humans also learn this task by trial and error, demonstrations of both successful and failed behavior are available.

Although hand-coding a reward function for this task would be infeasible, labeling each trial as successful or failed is straightforward. In addition, in many tasks, it may be easier to demonstrate failure than success. If expert demonstrations are scarce but a safe simulator is available, a non-expert can often easily demonstrate multiple failure modes, yielding data that complements the scarce successful demonstrations. This idea is explored in [111], where a robot learns to avoid people using simulations of failed avoidance. Finally, failed demonstrations may be used to explore the state-action space, an idea previously leveraged in other methods for learning from demonstration [38].

In existing IRL methods, failed demonstrations have been treated as noise [125] and filtered out in order to improve robustness. However, such methods do not actually use such demonstrations for learning.

In this chapter, we introduce *inverse reinforcement learning from failure* (IRLF), which is to our knowledge the first IRL algorithm that can learn from both successful and failed demonstrations. In doing so, we address a key difficulty in IRL: the problem is typically under-constrained since many reward functions are consistent with the expert's behavior. By exploiting failed demonstrations, our method reduces this ambiguity, resulting in faster and better learning.

To derive IRLF, we start from the state-of-the-art *maximum causal entropy IRL* [127, 128] method, which is also related to [8]. We propose a new constrained optimization formulation that accommodates both successful and failed demonstrations. We then derive update rules for learning policies and reward functions.

We evaluate our algorithm on the task of social navigation for a mobile robot, using both simulated and real-robot data, as well as the *Factory* problem, a well known decision making benchmark. On the simulated scenarios, our results demonstrate that IRLF generalizes better than maximum causal entropy IRL when successful demonstrations are scarce, with little additional computational cost. On the real-robot data, IRLF also outperforms this baseline.

3.1 Maximum causal entropy IRL

In the previous chapter we formulated the main idea behind IRL, which, in the linear case, aims to find a weight vector w that represents the reward function used to generate the observed demonstrations. Finding w can be formulated as an optimization problem that requires the expert trajectories to have maximal value [1] or that do so by some margin [98]. Alternatively one may employ Bayesian inference [22, 94] to compute a posterior distribution over the weights. In this chapter, we concentrate on methods that learn maximum-entropy policies [127, 128]. A maximum-entropy approach is attractive because it is probabilistic and thus robust to noise and randomness in the actions of the

expert, and results in convex optimization problems. The methods work by solving the following constrained optimization problem:

$$\text{find:} \quad \max_{\pi} H(\mathcal{A}^h || \mathcal{S}^h) \tag{3.1}$$

$$\text{subject to:} \quad \widetilde{\mu}_k^{\mathcal{D}} = \mu_k^{\pi}|_{\mathcal{D}} \quad \forall k \tag{3.2}$$

$$\text{and:} \quad \sum_{a \in \mathcal{A}} \pi(s,a) = 1 \quad \forall s \in \mathcal{S} \tag{3.3}$$

$$\text{and:} \quad \pi(s,a) \geq 0 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}, \tag{3.4}$$

where $H(\mathcal{A}^h || \mathcal{S}^h)$, the *causal entropy*, is the conditional entropy of the action sequence $\mathcal{A}^h$, causally conditioned on the state sequence $\mathcal{S}^h$:

$$H(\mathcal{A}^h || \mathcal{S}^h) = -\sum_{t=1}^{h} \sum_{\substack{s_{1:t} \in \mathcal{S}^t \\ a_{1:t} \in \mathcal{A}^t}} P(a_{1:t}, s_{1:t}) \log(P(a_t | s_t)), \tag{3.5}$$

where:

$$\begin{aligned} P(a_{1:t}, s_{1:t}) &= P(s_{1:t-1}, a_{1:t-1}) P(s_t | s_{t-1}, a_{t-1}) P(a_t | s_t) \\ &= P(s_{1:t-1}, a_{1:t-1}) T(s_{t-1}, a_{t-1}, s_t) \pi(s_t, a_t). \end{aligned}$$

The constraints require that π consists of proper probability distributions that match the empirical feature expectations. The optimization problem is solved using the method of Lagrange multipliers. First, the equality constraint (3.2) is relaxed, yielding a Lagrangian:

$$\mathcal{L}(\pi, w) \coloneqq H(\mathcal{A}^h || \mathcal{S}^h) + \sum_{k=1}^{K} w_k (\mu_k^{\pi}|_{\mathcal{D}} - \widetilde{\mu}_k^{\mathcal{D}}), \tag{3.6}$$

and in turn a less constrained optimization problem:

$$\text{find:} \quad \min_{w} \left\{ \max_{\pi} \left(\mathcal{L}(\pi, w) \right) \right\} \tag{3.7}$$

$$\text{subject to:} \quad \sum_{a \in \mathcal{A}} \pi(s,a) = 1 \quad \forall s \in \mathcal{S} \tag{3.8}$$

$$\text{and:} \quad \pi(s,a) \geq 0 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}, \tag{3.9}$$

Differentiating $\mathcal{L}(\pi, w)$ with respect to π at time t results in [127, p. 186]:

$$\begin{aligned}
&\nabla_{\pi(s_t,a_t)}\mathcal{L}(\pi, w) = P(s_{1:t}, a_{1:t-1})\Bigg(- \log(\pi(a_t, s_t)) + \\
&H(\mathcal{A}^{t+1:h}||\mathcal{S}^{t+1:h}) + \sum_{k=1}^{K} w_k E\left\{\sum_{\tau=1}^{h} \phi_k(s_t, a_t)|s_{1:t}, a_{1:t-1}\right\}\Bigg).
\end{aligned} \tag{3.10}$$

Setting the gradient to zero and solving for π gives:

$$\pi(s_t, a_t) \propto \exp\left(H(\mathcal{A}^{t:h}||\mathcal{S}^{t:h}) + \sum_{k=1}^{K} w_k \mu_k^{\pi,t:h}|_{s_t,a_t}\right). \tag{3.11}$$

Due to (2.6), the rightmost term in (3.11) is a measure of value for a given state and action, parameterised by w. Ziebart [127] showed that solving (3.11) while satisfying (3.8) and (3.9) amounts to solving a soft Bellman equation:

$$\begin{aligned}
&Q_w(s, a)^{soft} = \sum_{k=1}^{K} w_k\phi_k(s, a) + \sum_{s'} T(s, a, s')V_w(s'), \\
&V_w(s)^{soft} = \log \sum_{a} exp(Q_w(s, a)), \\
&\pi(s, a) = \exp(Q_w(s, a) - V_w(s)).
\end{aligned} \tag{3.12}$$

Once π is computed for a given set of weights, w is updated via gradient descent, by noting that:

$$\nabla_w \mathcal{L}(\pi, w) = \mu^\pi|_\mathcal{D} - \widetilde{\mu}^\mathcal{D}, \tag{3.13}$$

where $\mu^\pi|_\mathcal{D}$ is computed by rolling out policy π from the initial state distribution $P_\mathcal{D}(s_1)$ over h steps, and taking an expectation over the features accumulated at each step. The optimization process terminates once the weight vector converges to the optimal solution.

3.2 Method

In this section, we propose *inverse reinforcement learning from failure* (IRLF), a novel IRL algorithm that exploits failed demonstrations. IRLF is applicable in settings where the agent has access to a dataset $\mathcal{F}$ of failed demonstrations, in addition to a dataset $\mathcal{D}$ of successful ones.

A critical challenge in developing such a method is deciding how to interpret failed

trajectories. While successful trajectories may not be optimal, it is clear that the agent should try to imitate them, which in our case means matching feature expectations. By contrast, a failed trajectory is more ambiguous because it is unclear what about it is wrong: was the entire trajectory incorrect, or was it almost correct but wrong with respect to one particular feature? A key characteristic of the method we propose is that it works well in both these cases.

The first step is to formulate a new constrained optimization problem analogous to (3.1)–(3.4). In addition to requiring that the feature expectations of the learned policy match the empirical expectations of $\mathcal{D}$, we now also want those feature expectations to be *dissimilar* to the empirical expectations of $\mathcal{F}$. Although successful and failed demonstrations are semantic opposites, incorporating the latter into IRL proves to be nontrivial.

A straightforward approach is to add inequality constraints to the optimization problem:

$$|\widetilde{\mu}_k^{\mathcal{F}} - \mu_k^{\pi}|_{\mathcal{F}}| > \alpha_k \quad \forall k, \tag{3.14}$$

where $\widetilde{\mu}_k^{\mathcal{F}}$ is the empirical expectation of feature k according to $\mathcal{F}$, computed analogously to (2.7), and α_k is a variable to be maximized as part of the optimization objective, which becomes:

$$\max_{\pi,\alpha} H(\mathcal{A}^h||\mathcal{S}^h) + \sum_{k=1}^{K} \alpha_k. \tag{3.15}$$

However, this formulation is problematic because (3.14) is a non-linear constraint. An attempt to linearize this constraint quickly becomes involved and much harder to interpret.

Another way of penalizing similarity to the failed demonstrations is to remove the extra constraint in (3.14) and instead add to the objective the inner product of a parameter vector $\theta = [\theta_1 \sim \ldots \sim \theta_k]^T$ and the difference in feature expectations, yielding the following objective function:

$$\max_{\pi,\theta} H(\mathcal{A}^h||\mathcal{S}^h) + \sum_{k=1}^{K} \theta_k(\mu_k^{\pi}|_{\mathcal{F}} - \widetilde{\mu}_k^{\mathcal{F}}). \tag{3.16}$$

However, this approach is also problematic because it complicates the maximization of the Lagrangian. We now need to find a critical point with respect to *both* π and θ, which is analytically involved and numerically expensive, since it requires calculating $\mu^{\pi}|_{\mathcal{F}}$ each time that π is updated via the soft Bellman backup in (3.12).

To avoid these difficulties, we propose a different formulation. The main idea is to

create new equality constraints of the form $\mu_k^\pi|_\mathcal{F} - \widetilde{\mu}_k^\mathcal{F} = z_k$ with $z_k \in \mathbb{R}$, and introduce the z_k variables as terms in the optimization objective. The complete constrained optimization problem is thus:

$$\max_{\pi,\theta,z} \quad H(\mathcal{A}^h||\mathcal{S}^h) + \sum_{k=1}^{K} \theta_k z_k - \frac{\lambda}{2}||\theta||^2 \tag{3.17}$$

$$\text{subject to:} \quad \mu_k^\pi|_\mathcal{D} = \widetilde{\mu}_k^\mathcal{D} \quad \forall k \tag{3.18}$$

$$\text{and:} \quad \mu_k^\pi|_\mathcal{F} - \widetilde{\mu}_k^\mathcal{F} = z_k \quad \forall k \tag{3.19}$$

$$\text{and:} \quad \sum_{a\in\mathcal{A}} \pi(s,a) = 1 \quad \forall s \in \mathcal{S} \tag{3.20}$$

$$\text{and:} \quad \pi(s,a) \geq 0 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}, \tag{3.21}$$

where λ is a constant. Intuitively, the first term in the objective seeks to maximize π's causal entropy, while the second term balances this against maximizing dissimilarity between π's feature expectations and the empirical expectations in $\mathcal{F}$; the third term regularizes to discourage large values of θ. The main advantage of this formulation is that π and θ become decoupled for a given z, making maximization of the Lagrangian feasible while achieving our the goal of (3.16). Specifically, the new Lagrangian is:

$$\begin{aligned}\mathcal{L}(\pi,\theta,z,w^\mathcal{D},w^\mathcal{F}) = H(\mathcal{A}^h||\mathcal{S}^h) - \frac{\lambda}{2}||\theta||^2 + \sum_{k=1}^{K}\theta_k z_k + \\ \sum_{k=1}^{K} w_k^\mathcal{D}(\mu_k^\pi|_\mathcal{D} - \widetilde{\mu}_k^\mathcal{D}) + \sum_{k=1}^{K} w_k^\mathcal{F}(\mu_k^\pi - \widetilde{\mu}_k^\mathcal{F}|_\mathcal{F} - z_k).\end{aligned} \tag{3.22}$$

The less constrained optimization problem of (3.7)–(3.9) remains unchanged except that maximization of the Lagrangian is now with respect to π, θ, and z. Next, we differentiate the new Lagrangian with respect to the primal variables, beginning with θ and z:

$$\nabla_{\theta_k}\mathcal{L}(\pi,\theta,z,w^\mathcal{D},w^\mathcal{F}) = z_k - \lambda\theta_k, \tag{3.23}$$

$$\nabla_{z_k}\mathcal{L}(\pi,\theta,z,w^\mathcal{D},w^\mathcal{F}) = \theta_k - w_k^\mathcal{F}. \tag{3.24}$$

Setting both derivatives to zero yields:

$$z_k = \lambda w_k^\mathcal{F} \sim \text{and} \sim \theta_k = w_k^\mathcal{F}. \tag{3.25}$$

Plugging this back into the Lagrangian gives:

$$\begin{aligned}
&\max_{z,\theta} \mathcal{L}(\pi, \theta, z, w^{\mathcal{D}}, w^{\mathcal{F}}) =: \mathcal{L}_{z,\theta}(\pi, w^{\mathcal{D}}, w^{\mathcal{F}}) = \\
&H(\mathcal{A}^h || \mathcal{S}^h) - \frac{\lambda}{2} ||w^{\mathcal{F}}||^2 + \\
&\sum_{k=1}^{K} w_k^{\mathcal{F}} (\mu_k^{\pi}|_{\mathcal{F}} - \widetilde{\mu}_k^{\mathcal{F}} - \lambda w_k^{\mathcal{F}}) + \sum_{k=1}^{K} w_k^{\mathcal{D}} (\mu_k^{\pi}|_{\mathcal{D}} - \widetilde{\mu}_k^{\mathcal{D}}).
\end{aligned} \tag{3.26}$$

Finally, we differentiate $\mathcal{L}_{z,\theta}$ with respect to π:

$$\begin{aligned}
&\nabla_{\pi(s_t,a_t)} \mathcal{L}_{z,\theta}(\pi, w^{\mathcal{D}}, w^{\mathcal{F}}) = \\
&P(s_{1:t}, a_{1:t-1}) \big(-\log(\pi(a_t, s_t)) + H(\mathcal{A}^{t+1:h} || \mathcal{S}^{t+1:h}) \\
&+ \textstyle\sum_{k=1}^{K} (w_k^{\mathcal{D}} + w_k^{\mathcal{F}}) E\big\{ \textstyle\sum_{\tau=1}^{h} \phi_k(s_t, a_t) | s_{1:t}, a_{1:t-1} \big\} \big)
\end{aligned} \tag{3.27}$$

$$\pi(s_t, a_t) \propto \exp\left(H(\mathcal{A}^{t:h} || \mathcal{S}^{t:h}) + \sum_{k=1}^{K} (w_k^{\mathcal{D}} + w_k^{\mathcal{F}}) \mu_k^{\pi,t:h}|_{s_t,a_t} \right). \tag{3.28}$$

Intuitively, (3.28) implies that the value of π now depends on *both* Lagrangian multipliers $w^{\mathcal{D}}$ and $w^{\mathcal{F}}$. We can maximize with respect to π using a soft backup method analogous to (3.12) with the crucial difference that the reward function is now $\sum_{k=1}^{K} (w_k^{\mathcal{D}} + w_k^{\mathcal{F}}) \phi_k(s, a)$. Using the resulting policy π^*, we can define the dual objective:

$$\begin{aligned}
L^*(w^{\mathcal{D}}, w^{\mathcal{F}}) &:= \max_{\pi,\theta,z} \big(\mathcal{L}(\pi, \theta, z, w^{\mathcal{D}}, w^{\mathcal{F}}) \big) \\
&= H^{\pi^*}(\mathcal{A}^h || \mathcal{S}^h) - \frac{\lambda}{2} ||w^{\mathcal{F}}||^2 + \\
\sum_{k=1}^{K} w_k^{\mathcal{D}} (\mu_k^{\pi^*}|_{\mathcal{D}} - \widetilde{\mu}_k^{\mathcal{D}}) &+ \sum_{k=1}^{K} w_k^{\mathcal{F}} (\mu_k^{\pi^*}|_{\mathcal{F}} - \widetilde{\mu}_k^{\mathcal{F}} - \lambda w_k^{\mathcal{F}}).
\end{aligned} \tag{3.29}$$

Finally, to solve the dual, i.e., minimize L^*, we differentiate it with respect to $w^{\mathcal{D}}$ and $w^{\mathcal{F}}$:

$$\nabla_{w_k^{\mathcal{D}}} L^*(w^{\mathcal{D}}, w^{\mathcal{F}}) = \mu_k^{\pi^*}|_{\mathcal{D}} - \widetilde{\mu}_k^{\mathcal{D}}, \tag{3.30}$$

$$\nabla_{w_k^{\mathcal{F}}} L^*(w^{\mathcal{D}}, w^{\mathcal{F}}) = \mu_k^{\pi^*}|_{\mathcal{F}} - \widetilde{\mu}_k^{\mathcal{F}} - \lambda w_k^{\mathcal{F}}. \tag{3.31}$$

Setting (3.31) to zero yields:

$$w_k^{\mathcal{F}} = \frac{\mu_k^{\pi^*}|_{\mathcal{F}} - \widetilde{\mu}_k^{\mathcal{F}}}{\lambda}. \tag{3.32}$$

(3.30) implies that $w_k^{\mathcal{D}}$ can be updated using gradient descent, since the value of $\mu_k^{\pi^*}|_{\mathcal{D}}$ will change in the next iteration. The minimizing solution for $w_k^{\mathcal{F}}$ is found analytically. (3.32) shows how IRLF pushes the agent away from the failed demonstrations. If $\mu_k^{\pi^*}|_{\mathcal{F}} - \widetilde{\mu}_k^{\mathcal{F}}$ is positive, then $w_k^{\mathcal{F}}$ will also be positive. Since $w^{\mathcal{F}}$ is part of the reward function, on the next iteration, this will encourage a new π^* that increases $\mu_k^{\pi^*}|_{\mathcal{F}} - \widetilde{\mu}_k^{\mathcal{F}}$ further. The reverse occurs when $w^{\mathcal{F}}$ is negative.

A key characteristic of IRLF is that it also handles cases where the failed trajectories are only 'partial' failures, i.e., when the failed trajectories are similar to the successful ones with respect to some features and dissimilar with respect to others. For example, suppose there are only two features and $\widetilde{\mu}_1^{\mathcal{D}}$ is similar to $\widetilde{\mu}_1^{\mathcal{F}}$ but $\widetilde{\mu}_2^{\mathcal{D}}$ is dissimilar to $\widetilde{\mu}_2^{\mathcal{F}}$. It might seem that the updates to $w_k^{\mathcal{D}}$ and $w_k^{\mathcal{F}}$ with respect to the first feature would cancel each other out, thereby preventing IRLF from successfully imitating the successful trajectories with respect to that feature. However, that is not the case.

As IRLF iterates, the gradient descent update on $w_1^{\mathcal{D}}$ will bring the model and the data closer with respect to the first feature. Due to the similarity between $\widetilde{\mu}_1^{\mathcal{D}}$ and $\widetilde{\mu}_1^{\mathcal{F}}$, the analytical update on $w_1^{\mathcal{F}}$ will be smaller. Therefore, the influence of the failed demonstrations with respect to that feature will be small. The opposite will happen with respect to the second feature. Since, $\widetilde{\mu}_2^{\mathcal{D}}$ and $\widetilde{\mu}_2^{\mathcal{F}}$ are dissimilar, the update $w_2^{\mathcal{F}}$ will have an increasingly greater influence on the overall reward associated with that feature. Thus, with respect to the second feature, IRLF will be pulled towards the successful trajectories and pushed away from the failed ones. At the same time, this will not prevent IRLF from being pulled towards the successful trajectories with respect to the first feature. Our experiments in the next section empirically validate this characteristic of IRLF.

Another convenient characteristic of IRLF is that it does not require the feature sets for successful and failed demonstrations that make up $\mu^{\pi^*}|_{\mathcal{F}}$ and $\mu^{\pi^*}|_{\mathcal{D}}$ to be the same. A designer is therefore free to represent failed demonstrations with a different feature set that is more informative than the one used for successful demonstrations. Exploiting this property in practice is an interesting direction for future work.

However, a potential problem with IRLF is that, because (3.30) is updated incrementally while (3.32) is solved analytically, performance may oscillate, as small changes in w_k^s may be accompanied by large changes in w_k^f . This is especially true because the feature expectations of π are influenced by the updates on both $w^{\mathcal{D}}$ and $w^{\mathcal{F}}$. Fortunately, we can make IRLF more stable by annealing λ across iterations. Specifically, we start with a large value of λ, meaning that failed demonstrations are essentially ignored, and decrease it by a factor of α_λ on each iteration until a user-specified floor λ_{min} is reached. Intuitively, IRLF begins by ignoring the failed demonstrations and gradually takes them into account more and more.

Algorithm 1 describes IRLF procedurally. First, we compute the empirical feature expectations (lines 1-2) for both datasets $\mathcal{D}$ and $\mathcal{F}$ using (2.7). The initial state distributions are also computed (lines 3-4) by taking the normalized counts of the initial states in the two datasets. Then, we initialize the reward function (lines 5-8) used to find a policy (line 9). Using this policy, the model feature expectations are calculated as described by [129]. Next, we perform the updates following (3.30) and (3.32) (lines 12-13). Finally, we incrementally reduce λ (lines 14-15) to improve stability.

Algorithm 1 IRLF($\mathcal{S}, \mathcal{A}, T, \phi, \mathcal{D}, \mathcal{F}, \alpha, \alpha_\lambda, \lambda, \lambda_{min}$)

1: $\widetilde{\mu}^{\mathcal{D}} \leftarrow$ `empiricalFE`($\mathcal{D}$) {using (2.7)}
2: $\widetilde{\mu}^{\mathcal{F}} \leftarrow$ `empiricalFE`($\mathcal{F}$)
3: $P_{\mathcal{D}}^{s_1} \leftarrow$ `initialStateDistribution`($\mathcal{D}$)
4: $P_{\mathcal{F}}^{s_1} \leftarrow$ `initialStateDistribution`($\mathcal{F}$)
5: $w_k^{\mathcal{F}} \leftarrow 0 \quad \forall k \in \{1, \ldots, K\}$
6: Initialize $w^{\mathcal{D}}$ randomly
7: **repeat**
8: $\quad R(s,a) \leftarrow (w^{\mathcal{D}} + w^{\mathcal{F}})^T \phi(s,a) \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$
9: $\quad \pi \leftarrow$ `softPlan`($\mathcal{S}, \mathcal{A}, T, R$) {using (3.12)}
10: $\quad \mu^\pi|_{\mathcal{D}} =$ `calculateFE`($\pi, T, P_{\mathcal{D}}^{s_1}$)
11: $\quad \mu^\pi|_{\mathcal{F}} =$ `calculateFE`($\pi, T, P_{\mathcal{F}}^{s_1}$)
12: $\quad w^{\mathcal{D}} \leftarrow w^{\mathcal{D}} - \alpha(\mu^\pi|_{\mathcal{D}} - \widetilde{\mu}^{\mathcal{D}})$
13: $\quad w^{\mathcal{F}} \leftarrow \frac{(\mu^\pi|_{\mathcal{F}} - \widetilde{\mu}^{\mathcal{F}})}{\lambda}$
14: $\quad$ **if** $\lambda > \lambda_{min}$ **then**
15: $\quad\quad \lambda \leftarrow \alpha_\lambda \lambda$
16: **until** convergence
17: **return** R, π

3.3 Experiments

To evaluate IRLF, we consider three domains. Two of these concern the task of navigating a mobile robot in a social environment, a key challenge problem in social robotics [85]. Because social rules are difficult to quantify, IRL is well suited to this task [41, 119]. We first consider learning in a simulated navigation domain and then repeat the process using data gathered from experiments on a real telepresence robot.[1]

Our third domain is the *Factory* problem, a well known decision-theoretic benchmark domain [20]. We compare IRLF to the original *maximum casual entropy IRL* [128], which we henceforth refer to simply as IRL.

[1]Code to replicate the experiments in Sections 4.1 and 4.2 can be found at : `https://github.com/KyriacosShiarli/IRLF_aamas2016_Replicate.git`.

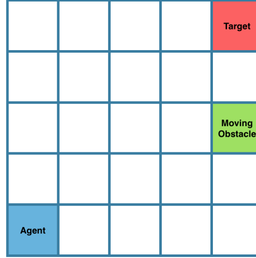
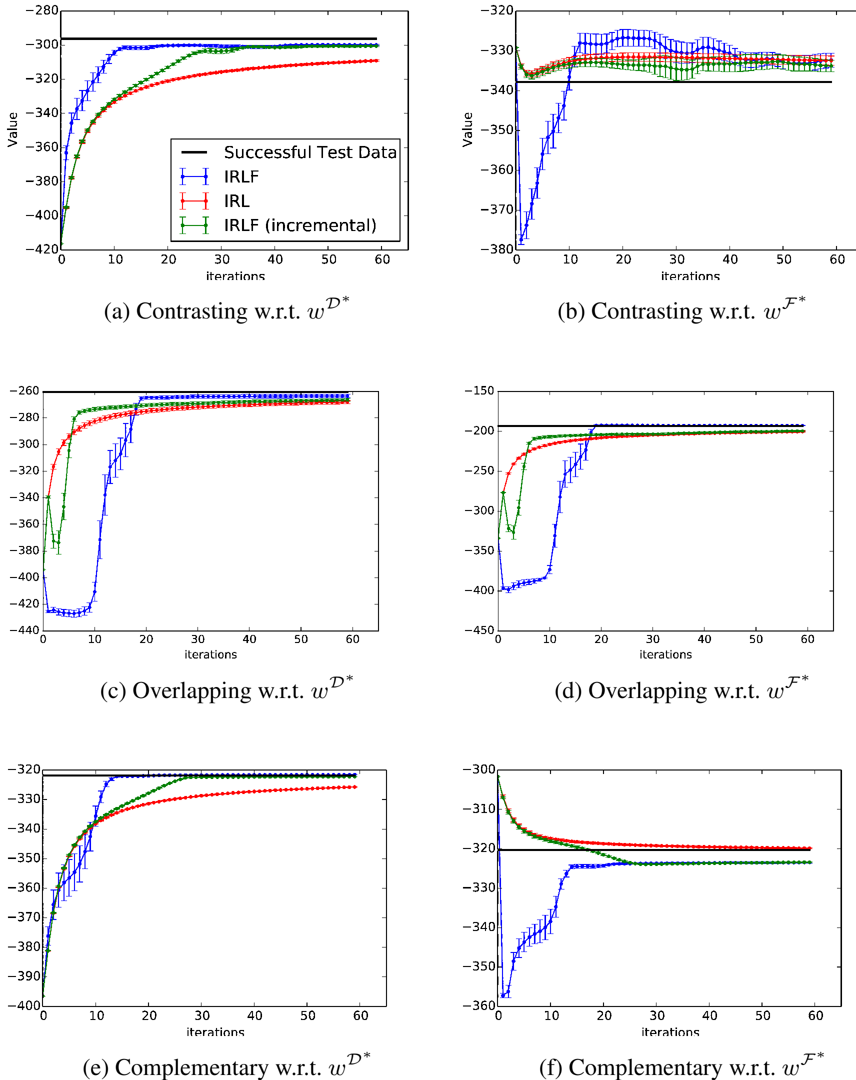


Figure 3.1: Simulated social navigation task.

3.3.1 Simulated Navigation Domain

First, we consider a simulated navigation domain, shown in Figure 3.1, in which a robot (blue) navigates its environment to reach a target (red) while avoiding a moving obstacle (green). The state space contains all possible combined (x, y) positions of the obstacle and the robot as well as five possible orientations for the obstacle, with one of them being stopped. The action space is $\mathcal{A} = \{up, down, left, right, stay\}$. Binary state features are computed by discretizing the displacement between the (x, y) coordinates of the robot to both the obstacle and the target into five possible values for each of the four dimensions, yielding a complete feature vector $\phi(s) \in \{0, 1\}^{20}$ for any $s \in \mathcal{S}$. The transition model for this world is deterministic. The action chosen by the robot results in the agent moving to that direction with probability 1. The moving obstacle maintains its initial orientation throughout the episode. When the obstacle moves beyond the edge of the grid, it reappears on the opposite side, while the robot remains in the same cell until a valid action is chosen.

We first manually define two ground truth reward functions, $R^{\mathcal{D}^*} = w^{\mathcal{D}^*}\phi(s, a)$, $R^{\mathcal{F}^*} = w^{\mathcal{F}^*}\phi(s, a)$, described further below. Then, we sample initial test states from a uniform distribution over the state space, over which we define $P^{s_1}_{test}$, for all experimental runs. These initial conditions, along with the optimal maximum-entropy policies for $R^{\mathcal{D}^*}$ and $R^{\mathcal{F}^*}$, allow us to compute feature expectations $\widetilde{\mu}^{\mathcal{D}_{test}}$ and $\widetilde{\mu}^{\mathcal{F}_{test}}$ respectively. For each experimental run, we then sample a set of initial training states, over which we define $P^{s_1}_{train}$, and generate the respective feature expectations $\widetilde{\mu}^{\mathcal{D}_{train}}$, $\widetilde{\mu}^{\mathcal{F}_{train}}$. These feature expectations are used to learn reward functions and their corresponding policies using each algorithm under evaluation. Each learned policy π^* is then executed from initial states sampled from $P^{s_1}_{test}$ to determine $\mu^{\pi^*}|_{test}$, the feature expectations for the policy at those initial conditions. Finally, we compute the values of each policy, at those initial conditions, with respect to the two reward functions, $V^{\pi^*}_{\mathcal{D},test} = (w^{\mathcal{D}^*})^T \mu^{\pi^*}|_{test}$, $V^{\pi^*}_{\mathcal{F},test} = (w^{\mathcal{F}^*})^T \mu^{\pi^*}|_{test}$. A good algorithm will yield a high $V^{\pi^*}_{\mathcal{D},test}$ and a low $V^{\pi^*}_{\mathcal{F},test}$.



(a) Contrasting w.r.t. $w^{\mathcal{D}^*}$

(b) Contrasting w.r.t. $w^{\mathcal{F}^*}$

(c) Overlapping w.r.t. $w^{\mathcal{D}^*}$

(d) Overlapping w.r.t. $w^{\mathcal{F}^*}$

(e) Complementary w.r.t. $w^{\mathcal{D}^*}$

(f) Complementary w.r.t. $w^{\mathcal{F}^*}$

Figure 3.2: Value over 60 iterations, for 20 runs in contrasting, complementary and overlapping simulated domains w.r.t. $w^{\mathcal{D}^*}$ and $w^{\mathcal{F}^*}$.

Within this domain, we consider three scenarios that differ in how $R^{\mathcal{D}^*}$ and $R^{\mathcal{F}^*}$ are defined. In the *contrasting scenario*, $w^{\mathcal{D}^*}$ rewards reaching the target and avoiding the obstacle, while $w^{\mathcal{F}^*}$ rewards being in the same cell as the obstacle. This scenario examines the value of completely failed demonstrations when the successful demonstrations already show the complete desired behavior.

In the *overlapping scenario*, $w^{\mathcal{D}^*}$ is as before but $w^{\mathcal{F}^*}$ rewards not only colliding

with the obstacle, but also reaching the target. This scenario examines the value of failed demonstrations when they are similar in some respects to the successful demonstrations.

In the *complementary scenario*, $w^{\mathcal{D}^*}$ rewards only reaching the target, while $w^{\mathcal{F}^*}$ only rewards hitting the obstacle. This scenario examines the value of failed demonstrations when the successful demonstrations do not fully disambiguate the desired behavior.

Figures 3.2a and 3.2b compare the performance of IRLF, with and without incremental updates to λ, to that of IRL in the contrasting scenario. Both versions of IRLF successfully utilize failed demonstrations to learn better and faster than IRL in terms of $V^{\pi^*}_{\mathcal{D},test}$. They achieve similar performance to IRL in terms of $V^{\pi^*}_{\mathcal{F},test}$.

Figures 3.2c and 3.2d show results for the overlapping scenario. Even in this challenging setting, IRLF learns better than IRL, demonstrating the resilience of our method to the fact that some successful and failed trajectories might be similar and showing that our method can exploit the additional data found in failed trajectories without negative side effects.

Finally, Figures 3.2e and 3.2f show similar results for the complementary scenario. The IRLF methods again perform better in terms of $V^{\pi^*}_{\mathcal{D},test}$ but now also perform better in terms of $V^{\pi^*}_{\mathcal{F},test}$. In fact, they outperform even the successful data in terms of $V^{\pi^*}_{\mathcal{F},test}$, a consequence of the complementary nature of the reward functions.

IRLF's performance with respect to $V^{\pi^*}_{\mathcal{F},test}$ (Figure 3.2 bottom row) illustrates how the algorithm gives priority to successful demonstrations when necessary. For the contrasting scenario, the probability of approaching the obstacle is already low for $w^{\mathcal{D}^*}$; therefore, IRL and IRLF behave similarly with respect to $w^{\mathcal{F}^*}$. In the overlapping scenario, IRLF gives priority to reaching the target quickly, since this matches the behavior in the successful demonstrations (even though it is discouraged from doing so by the failed demonstrations). Doing so means accumulating more value in terms of $V^{\pi^*}_{\mathcal{F},test}$. In the complementary scenario, since it is possible to satisfy both objectives simultaneously (reaching the target and avoiding obstacles), IRLF finds a reward function that performs well with respect to the successful demonstrations while at the same time having a lower value with respect to the failed demonstrations.

In all scenarios, IRLF is more stable with incremental updates than without. In the contrasting and complementary scenarios, IRLF with incremental updates learns more slowly, while in the overlapping scenario it learns faster initially but plateaus slightly lower.

In the experiments above, all methods received successful and failed demonstrations from five initial states. Hence, the results do not address how the number of successful demonstrations given to the two learners affects their performance on the test set. In the complementary scenario, failed demonstrations are obviously important regardless

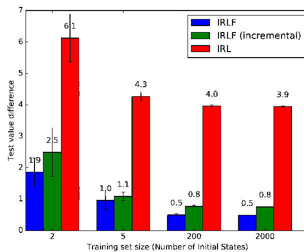


Figure 3.3: Performance of all three methods in the contrasting scenario for different numbers of initial states.

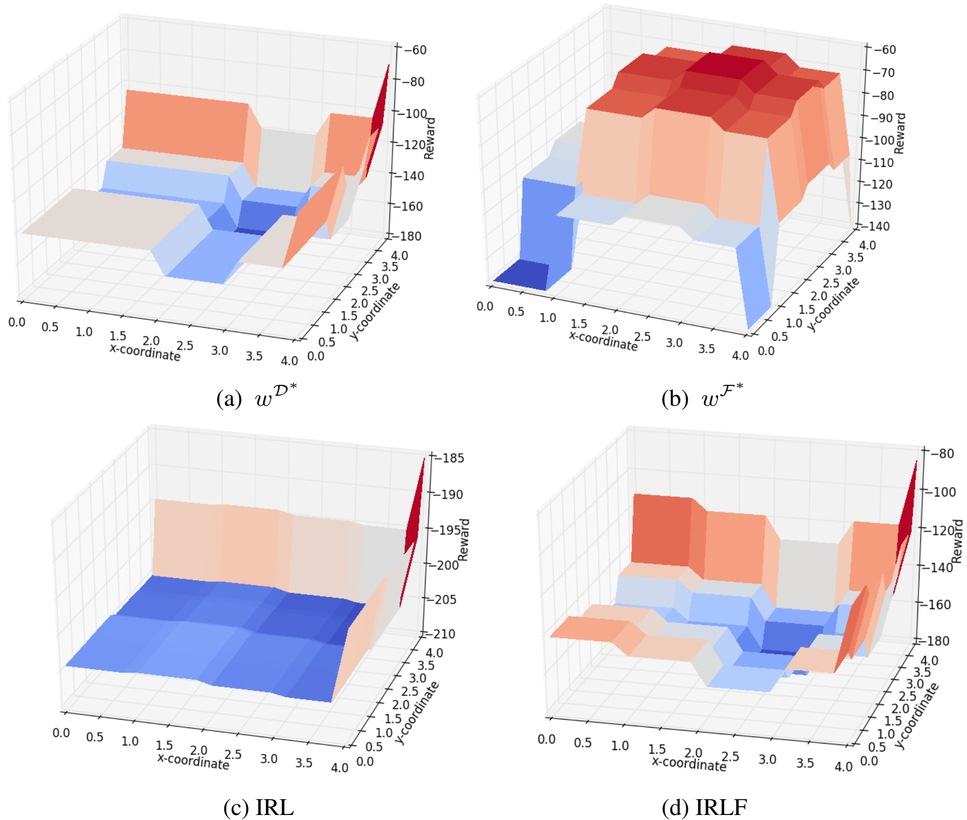
of how many successful demonstrations are available. In the other scenarios, however, it is not clear whether failed demonstrations are still useful even when successful demonstrations are abundant.

To test this, we repeated our experiments in the contrasting scenario but with 5, 50, 200 and 2000 initial states for successful demonstrations, while keeping the failed demonstrations to 5. For each algorithm, we plot $(w^{\mathcal{D}^*})^T\widetilde{\mu}^{\mathcal{D}_{test}}$ - $V^{\pi^*}_{\mathcal{D},test}$ after 60 iterations of the algorithm. The smaller this value, the closer the learner comes to replicating the expert on the test set of initial states. The results, shown in Figure 3.3, demonstrate that IRLF maintains its superiority over IRL even if the number of initial states for which demonstrations are given rises significantly.

To shed more light on IRLF's behavior, Figure 3.4 plots the original and learned reward functions for IRL and IRLF for the contrasting scenario. In the original reward function $w^{\mathcal{D}^*}$ (Figure 3.4a), the obstacle is in cell [2,2] and the goal in cell [4,4], which explains the dips and spikes in those locations. The reward function learned by IRL (Figure 3.4c) is flat in the area of the obstacle. However, IRLF, by employing $w^{\mathcal{F}^*}$ (Figure 3.4b) learns a reward function (3.4d) that properly assigns a low reward to the obstacle and its surroundings. The reward function resulting from IRLF can therefore generalize better to unseen initial conditions and environments.

3.3.2 Simulated Factory Domain

Our second simulated domain is the *Factory* benchmark problem proposed by Dearden & Boutilier [20]. In this domain, an autonomous agent is tasked with building an object according to specifications, which involves some sequence of shaping, painting, cleaning and assembly operations on its parts. Each individual operation costs some time and may have a probabilistic outcome, possibly resulting in unwanted side effects on the condition of each of the parts. Furthermore, there are precedence conditions on some of these operations – for instance, parts must be shaped before they can be



(a) $w^{\mathcal{D}^*}$ (b) $w^{\mathcal{F}^*}$

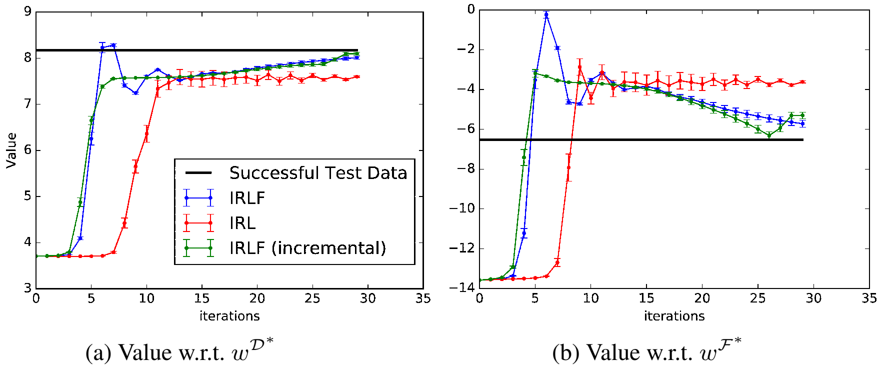
(c) IRL (d) IRLF

Figure 3.4: The original reward and learned reward functions for the two algorithms. Obstacle is static at [2,2]

assembled. The goal of this domain is to produce the object in the most cost-efficient way.

While this problem has been previously tackled from the perspective of off-line decision-theoretic planning, it is a suitable domain to demonstrate the applicability of learning from demonstration, and in particular of IRLF. In a real manufacturing problem, human demonstrations on how to properly execute the manufacturing task, as well as failed demonstrations, in which the resulting object did not meet the quality specifications, would be readily available. Furthermore, consider a situation where the conditions in the factory problem may be subject to changes (for example, in the shapes or sizes of the parts) which may affect the optimal manufacturing policy. Learning a reward function, as opposed to learning a policy from demonstration, would make it easier for an autonomous agent to adapt to these changes.

Our instantiation of the Factory problem follows Dearden & Boutilier's description



(a) Value w.r.t. $w^{\mathcal{D}^*}$

(b) Value w.r.t. $w^{\mathcal{F}^*}$

Figure 3.5: Value over 30 iterations, for 15 runs of learning in the Factory domain.

Feature Index i	Proposition	$w_i^{\mathcal{D}^*}$	$w_i^{\mathcal{F}^*}$
1	AClean	0.1	1
2	BClean	0.1	-0.1
3	APainted	0.2	0.4
4	BPainted	0.2	0
5	Joined	0.4	-2

Table 3.1: Reward functions for the Factory domain.

[20] (*c.f.* Section A.2). It has 512 states, factored into 9 binary variables, and 10 actions. The expert demonstrations were drawn from the optimal policy for that domain according to its original additive rewards ($w_i^{\mathcal{D}^*}$, Table 3.1), while the failed demonstrations were drawn from a policy derived from a different reward function ($w_i^{\mathcal{F}^*}$, Table 3.1). Note that we consider only 5 features, corresponding to those state factors that have non-zero rewards. Each demonstration starts from random initial conditions and the evaluation proceeds in an identical manner to Section 3.3.1.

Figure 3.5 shows the results, which are consistent with our previous observations. Figure 3.5a shows that the reward function learned using IRLF is capable of yielding a policy that achieves a higher $V^{\pi^*}_{\mathcal{D},test}$ than IRL, and very close to that of the original policy. Furthermore, Figure 3.5b shows that IRLF is capable of achieving lower $V^{\pi^*}_{\mathcal{F},test}$ than IRL. These results not only demonstrate the stability and reliability of IRLF, but further confirm the usefulness of failed demonstrations in inverse reinforcement learning.

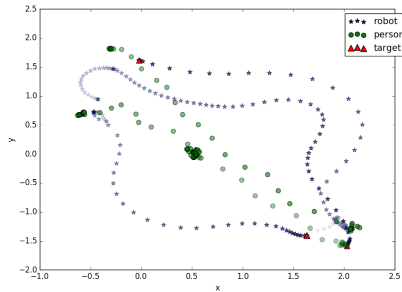


Figure 3.6: A sample of trajectories from the real navigation domain data set. Darker dots occur later in time.

3.3.3 Real Navigation Domain

We also evaluate the performance of IRLF on a social navigation dataset gathered using is a commercially available telepresence robot that has been augmented with extra sensors and processors. The collected dataset consists of 47 trajectories of an expert navigating the robot towards an interaction target while avoiding a moving or standing person in a socially appropriate way. The positions of the people and the robot were accurately recorded using a motion capture system covering an area of $5m \times 5m$. A sample of the collected trajectories are shown in Figure 3.6.

For safety reasons, the failed dataset $\mathcal{F}$, which consists of 59 trajectories, was collected by driving the robot in a simulator in a way that intentionally violates social norms, e.g., running into, following, or standing next to people.

To model this domain as an MDP, we define a state space S consisting of discretized positions for both the obstacle and the robot, as well as the discretized orientation of the obstacle, yielding a total of $837,808$ states. The action space consists of eight possible directions for robot movement, and an action that stops the robot. Using $\mathcal{D}$, we learned a factored stochastic transition function T. Furthermore, we defined a feature set $\phi(s,a)$ of 1401 binary features, encoding the relative position of the robot from both the target and the obstacle, as well as their relative orientations.

Applying IRL and IRLF to this domain is straightforward. However, measuring performance is not, because we do not have access to the ground truth reward function needed to compute a policy's value. In previous work on IRL, researchers have taken multiple approaches to evaluation. Some have considered simulated domains [69, 101, 114] as in our results above, where a ground truth reward function is available. Others have considered real-world data but relied on qualitative evaluation [98], domain-specific performance measures [79] or ad-hoc performance measures [119].

Since none of these approaches is entirely satisfactory, we adopt a different approach. First, we apply IRL to both $\mathcal{D}$ and $\mathcal{F}$ and derive reward functions, $R^{\mathcal{D}}$ and $R^{\mathcal{F}}$ and their respective weights $w^{\mathcal{D}^*}$ and $w^{\mathcal{F}^*}$, which we thereafter treat as ground truth. Because MDPs are generative models, we can use these two separate reward functions to generate data while preserving access to the ground-truth reward functions that are so essential to evaluation. We are therefore able to conduct an evaluation analogous to the done in the simulated navigation domain, with the key difference that the $w^{\mathcal{D}^*}$ and $w^{\mathcal{F}^*}$ are learned from real data.

Figure 3.7, which shows example trajectories generated by these reward functions, confirms that $w^{\mathcal{D}^*}$ produces human-like trajectories, while $w^{\mathcal{F}^*}$ generates inappropriate behavior. These two reward functions can therefore be considered to be two separate agents whose demonstrations IRLF uses in order to learn a single reward function in a principled manner.

Figure 3.8 shows the results of our experiments on the real data, averaged across 6 independent runs. These results demonstrate that both versions of IRLF substantially outperform IRL with respect to $w^{\mathcal{D}^*}$. The performance of all methods is similar with respect to $w^{\mathcal{F}^*}$, which suggests that, in this domain, imitating good behavior is sufficient to score very low with respect to $w^{\mathcal{F}^*}$. IRLF without incremental updates to λ clearly oscillates, as expected, but still converges, while IRLF with incremental updates is stable and achieves a higher value than IRL. Overall, the consistency of these results with those of the simulated domain provides further confirmation that failed demonstrations are useful for IRL, allowing it to generalize better to new initial conditions. Furthermore, IRLF is a useful method for exploiting this data.

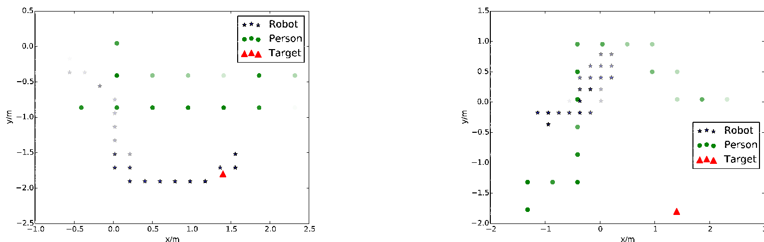
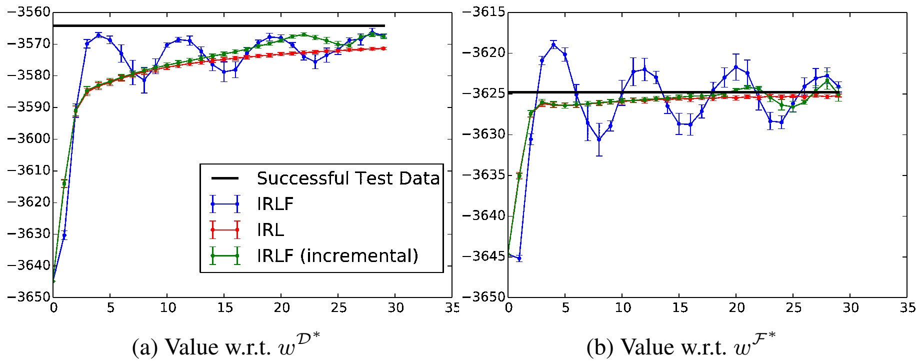


Figure 3.7: Sample trajectories from $w^{\mathcal{D}^*}$ (top) and $w^{\mathcal{F}^*}$ (bottom). Darker dots occur later in time.



(a) Value w.r.t. $w^{\mathcal{D}^*}$

(b) Value w.r.t. $w^{\mathcal{F}^*}$

Figure 3.8: Value over 30 iterations, for 6 runs on real data.

3.4 Conclusions and Future Work

This chapter introduced IRLF, the first inverse reinforcement learning algorithm that can learn from successful as well as failed demonstrations simultaneously. Starting from the maximum causal entropy IRL method, we proposed a new constrained optimization formulation that accommodates both types of demonstrations and derived update rules for learning reward functions and policies. Our simulated navigation experiments investigated the properties of IRLF in cases where the successful and failed demonstrations are either contrasting, overlapping, or complementary. The results in the overlapping scenario show that IRLF works even if the failed demonstrations are similar to the successful ones. We also presented results on a simulated factory domain as well as on real robot data from a social navigation task. These results clearly suggest that IRLF consistently learns faster and generalizes better than the original IRL algorithm in consideration. In future work, we aim to investigate whether other IRL methods can also be extended to exploit failed demonstrations. In addition, we plan to conduct more experiments with real robots, including deploying an IRLF-learned policy on a robot.

Chapter 4

Rapidly Exploring Learning Trees

In the previous chapters we introduced the main principles behind inverse reinforcement learning (IRL), motivated its use in robotic environments and discussed its benefits against competing LfD methods such as behavioral cloning (BC). The power of IRL is rooted in its iterative procedure, which in the inner loop solves the forward planning problem, i.e., for the current cost function it plans or learns an optimal policy. This allows us to collect statistics about the policy induced by this cost function and update it by comparing them with the corresponding statistics in the data.

However, the forward planning problem becomes increasingly harder to perform as the size of the state space increases. For continuous domains such as robotics, solving the planning problem exactly is typically intractable, necessitating a coarse discretization that yields poor performance. Another alternative is to perform reinforcement learning (RL) in the inner loop which trades off planning for environment interaction. However RL is also impractical for non-simulated robots both in terms of time as well as safety. In robotics, planning is often done using sample-based, single-query motion planners such as Rapidly Exploring Random Trees (RRTs) [66] and their variants [51]. Such methods can cope with high-dimensional continuous domains, as well as the presence of obstacles in the environment and motion constraints on the robot.

In this chapter, we first propose Approximate Maximum Margin Planning (AMMP), a variant of Maximum Margin Planning (MMP) [98] that does not assume an optimal planner. Doing so allows us to use sample-based planning algorithms, such as RRT* in the inner loop of IRL. Second, we propose a caching scheme that both improves performance and reduces the computational cost when RRT* is used within AMMP. The

resulting algorithm, which we call Rapidly Exploring Learning Trees (RLT*), allows computationally efficient IRL in high-dimensional, continuous domains with obstacles and motion constraints.

We evaluate RLT* on real and simulated data from a social navigation scenario. The results demonstrate that, in the absence of obstacles and motion constraints, RLT* performs better and is faster than both MMP and RLT* without caching. Furthermore, we show that RLT*, unlike MMP, can learn cost functions in robotic tasks with obstacles and motion constraints. Finally, we deploy our method on a real telepresence robot using data from human demonstrations.

4.1 Related Work

Substantial research has applied IRL to robotics [2, 41, 119]. A big challenge in doing so is solving the forward planning problem under the current cost function at every iteration. The planning problem is especially intractable in robotics because the state-action spaces encountered are often continuous and high dimensional.

One approach is to discretize the state-action spaces and solve the resulting Markov decision process (MDP). However, in high-dimensional tasks, fine discretization renders planning intractable and coarse discretization yields poor performance.

Some methods that use discretization formulate the problem such that only an open-loop path (rather than a closed-loop policy) is required. Then, given deterministic dynamics, planning can be performed using A* [98], allowing realistic problems to be tackled. We take a similar approach but, by planning using RRT*, we substantially enlarge the class of tasks that can be tackled.

Another approach is to use hybrid planners. Inverse Optimal Heuristic Control [96] models the long-term goals of the agent as a coarsely discretized MDP, to ensure tractability, while using supervised learning to determine the local policy of the agent at each state. Graph-Based IOC [15] uses discrete optimal control on a coarse graph and the actual path is executed using local trajectory optimization techniques such as CHOMP [97], which would otherwise suffer from local minima. However, these methods employ complex and domain-specific planning formulations that are not suitable for all robotics tasks. By contrast, our approach employs a widely used planner, making it versatile and easy to implement.

Another recent method is Adaptive State Graphs [83], which builds a state-action controller graph before doing any learning. This controller graph is akin to *options* in semi-Markov decision processes, and allows for a more flexible representation of the state-action space. However, the controller used to learn the underlying cost function is different from the one used to execute the robot's behavior. This can have adverse

effects since IRL assumes that the demonstration paths came from the same planner used during learning. Instead of building a controller graph first and then using a different controller to optimize trajectories, RLT* builds a controller tree on the fly and uses the same planner for learning and execution.

All of the above methods require a model of the dynamics of the system. Another paradigm, that of model-free IRL, avoids the need to explicitly plan, employing sampling instead. As a result it tends to scale better to high-dimensional state-action spaces and does not require a model of the system dynamics, [12, 49]. Model free IRL samples trajectories, usually starting from the initial conditions observed in the data. The probability of these trajectories is weighted according to the current cost function, which is then updated to make trajectories closer to the data more likely. A big challenge is to find an appropriate importance distribution at each iteration. [28] considers guiding the importance distribution by simultaneously learning a good policy under the current cost function and using it as an adaptive importance sampler. However, since it uses a modified version of Linear Quadratic Regulator control the derived policies are only locally optimal and can produce highly suboptimal paths or even fail in the presence of obstacles. By contrast, RRTs cope naturally with obstacles, as long as they are static and mapped, and is also asymptotically optimal. Other model free IRL methods include generative adversarial imitation learning [43] and its variants [71] [32]. While these methods are extremely general and can in theory be applied to any environment, they have only been employed to solve high dimensional control tasks in simulation. In addition because these methods do not typically extract a reward function it is very hard to transfer a learned model from simulation to the real world. Our method, while less general, elegantly handles problems in robotic planning domains and can transfer the real reward function to the real world.

4.2 Path Planning

Path planning occurs in a space $\mathcal{S}$ of possible robot configurations. A configuration $s \in \mathcal{S}$ is usually continuous, and often represents spatial quantities such as position and orientation. A path planner seeks an obstacle-free path $\rho_{o,g} = (s_1, s_2, s_3 \ldots, s_{l_\rho})$ of length l_ρ, from an initial configuration $o = s_1$ to a goal configuration $g = s_{l_\rho}$. When the initial and goal configurations are implied, we refer to a path as ρ.

As there could be several paths to the goal, planners often employ a *cost functional*, $C(\rho)$, which typically sums the costs between two subsequent configurations $c(s_i, s_j)$:

$$C(\rho) = \sum_{i=1}^{l_\rho - 1} c(s_i, s_{i+1}). \tag{4.1}$$

This cost functional is similar to the one used in optimal control and analogous to the return used in reinforcement learning. Given the cost functional, the path planner seeks an optimal path ρ^*, which satisfies,

$$\rho^*_{o,g} = \underset{\rho_{o,g} \in \mathcal{P}_{o,g}}{\operatorname{argmin}} C(\rho), \tag{4.2}$$

where $\mathcal{P}_{o,g}$ is the set of possible paths such that $s_1 = o, s_{l_\rho} = g$. Many path planning algorithms discretize $\mathcal{S}$ and use graph search algorithms like A* to find the optimal path. Under mild assumptions, these approaches are guaranteed to find the best path on the graph, therefore solving (4.2) for a subset $\tilde{\mathcal{P}}_{o,g} \subset \mathcal{P}_{o,g}$, whose size depends on the discretization. However, such methods scale poorly in the size of $\mathcal{S}$. Furthermore, such algorithms ignore motion constraints as they assume all nodes in the graph can be reached by their neighbors in exactly the same way. This assumption does not hold, e.g., for non-holonomic robots, where the orientation at each node places constraints on how it can be reached. In this constrained space, it is less straightforward to define actions, neighboring nodes, or an admissible heuristic, which is necessary for optimality. These drawbacks motivate *sample-based* path planning algorithms such as RRT* [51]. Instead of building a graph and then searching it, RRT* builds a tree on the fly and keeps track of the current best path. The algorithm consists of two interleaved steps.

The first step is *sampling*. A random point s_{rand} is sampled from the configuration space. Next, the closest point $s_{closest}$, already in the existing vertex set V is determined and a new point s_{new} is created by *steering* from $s_{closest}$ to s_{rand}. In a Euclidean space, steering between two points means simply connecting them with a straight line. However, if orientations and motion constraints are used, then steering becomes more complex. Finally, the sampling step determines the points, S_{near}, within a given radius of s_{new}.

The second step is *rewiring*, which determines which points in S_{near} we should connect to s_{new}, i.e., it determines which path to s_{new} results in a lowest cost path. Finally, we repeat this step for the parents of S_{near}. Thus, the tree is rewired locally around the new point, such that lower global cost paths arise.

Alternating between these two steps for a given time budget T solves (4.2) for a subset $\tilde{\mathcal{P}}_{o,g}$ that is determined by the randomly sampled points. As $T \to \infty$, RRT* minimizes over the entire $\mathcal{P}_{o,g}$, i.e., it is asymptotically optimal in time [51]. By contrast, A* is asymptotically optimal in resolution.

4.3 IRL for Path Planning

In path planning, we are given a cost function and must find a (near) optimal path to the goal. In the inverse problem, we are given example paths and must find the cost function for which these paths are (near) optimal. The example paths comprise a dataset $\mathcal{D} = (\rho^1_{o_1,g_1}, \rho^2_{o_2,g_2} ... \rho^D_{o_D,g_D})$ where $\rho^i_{o_i,g_i}$ is an example path with initial and final configurations o_i and g_i. We assume the unknown cost function is of the form,

$$c(s_i, s_j) = \mathbf{w}^T \mathbf{f}(s_i, s_j), \tag{4.3}$$

where $\mathbf{f}(s_i, s_j)$ is a K-dimensional vector of features that encode different aspects of the configuration pair and $\mathbf{w}$ is a vector of unknown weights to be learned. Since $\mathbf{w}$ is independent of the configuration, we can express the total cost of the path in a parametric form:

$$C(\rho) = \mathbf{w}^T \sum_{i=0}^{l_\rho - 1} \mathbf{f}(s_i, s_{i+1}) := \mathbf{w}^T \mathbf{F}(\rho), \tag{4.4}$$

where $\mathbf{F}(\rho)$ is the *feature sum* of the path.

While many formulations of the inverse problem exist, the general idea is to find a weight vector that assigns less cost to the example paths than all other possible paths with the same initial and goal configuration. This can be formalized by a set of inequality constraints:

$$C(\rho^i_{o_i,g_i}) \leq C(\rho) \quad \forall \rho \in \mathcal{P}_{o_i,g_i} \quad \forall i. \tag{4.5}$$

The constraint is an inequality because $\mathcal{P}_{o_i,g_i}$ contains only paths available to the planner and thus may not include the example path $\rho^i_{o_i,g_i}$. $\mathcal{P}_{o_i,g_i}$ can be large but if we have an optimization procedure that solves (4.2), it is enough to satisfy,

$$C(\rho^i_{o_i,g_i}) \leq \min_{\rho \in \mathcal{P}_{o_i,g_i}} C(\rho) \quad \forall i. \tag{4.6}$$

Maximum Margin Planning (MMP) [98] introduces a margin function $L_i(\rho)$ that decreases the cost of the proposed path ρ if it is dissimilar to $\rho^i_{o_i,g_i}$. For example, $L_i(\rho)$ could be -1 times the number of configurations in the demonstration path not visited by ρ. As in support vector machines, requiring the model to fit the data well even in the presence of a margin improves generalization. Furthermore, the margin helps address the ill-posed nature of IRL, i.e., many cost functions are consistent with the

demonstrated behavior. Formally, MMP solves the following optimization problem:

$$\underset{\mathbf{w},\tau}{\operatorname{argmin}} \frac{1}{2}||\mathbf{w}||^2 + \frac{\lambda}{D}\sum_i \tau_i \tag{4.7}$$

$$\text{s.t.} \quad C(\rho^i_{o_i,g_i}) - \tau_i \leq \min_{\rho \in \mathcal{P}_{o_i,g_i}} C(\rho) + L_i(\rho) \quad \forall i, \tag{4.8}$$

where τ_i are slacks that can be used to relax the constraints. Rearranging the inequality in terms of the slacks yields:

$$C(\rho^i_{o_i,g_i}) - \min_{\rho \in \mathcal{P}_{o_i,g_i}} C(\rho) + L_i(\rho) \leq \tau_i \quad \forall i. \tag{4.9}$$

Consequently, the $\mathbf{w}$ minimizing:

$$\frac{1}{2}||\mathbf{w}||^2 + \frac{\lambda}{D}\sum_i \left(C(\rho^i_{o_i,g_i}) - \min_{\rho \in \mathcal{P}_{o_i,g_i}} \left(C(\rho) + L_i(\rho)\right)\right) \tag{4.10}$$

also minimizes (4.7), i.e., the slacks are tight. The minimum can be found by computing a subgradient and performing gradient descent on the above objective:

$$\nabla_{\mathbf{w}} = \mathbf{w} + \frac{\lambda}{D}\sum_{i=0}^{D} F(\rho^i_{o_i,g_i}) - F(\tilde{\rho}^*_{o_i,g_i}), \tag{4.11}$$

where,

$$\tilde{\rho}^*_{o_i,g_i} = \underset{\rho \in \mathcal{P}_{o_i,g_i}}{\operatorname{argmin}} C(\rho) + L_i(\rho). \tag{4.12}$$

The inverse problem can therefore be seen as an iterative procedure that first solves (4.12) in the inner loop while keeping the weights constant. Given that solution, it then updates the weights using (4.11) in the outer loop. The weights at convergence represent the cost function that is used to plan future behavior. In [98], A* search was used for planning in the inner loop, assuming that the domain contained acyclic positive costs. In this chapter, we make the same assumptions but develop methods that use RRT* for planning.

4.4 Method

In this section, we propose Rapidly Exploring Learning Trees (RLT*). We first propose a generic extension to MMP that we call Approximate Maximum Margin Planning. We then show how an implementation of this approach with an RRT* planner and a novel caching scheme yields RLT*.

4.4.1 Approximate Maximum Margin Planning

Section 4.3 shows how the multiple constraints of (4.5) can be reduced to the single constraint of (4.6) for each demonstration. However, this reduction requires an optimal planner to perform the minimization in (4.5), which is impractical in many robot applications. Suppose instead that, as in RRT*, we have a mechanism for sampling different paths from $\mathcal{P}_{o,g}$ along with their respective costs and that, for a given finite time budget T, this path sampler samples a subset $\tilde{\mathcal{P}}_{o,g} \subset \mathcal{P}_{o,g}$. Then, we can modify (4.6) to demand that our cost function satisfies,

$$C(\rho^i_{o_i,g_i}) \leq \min_{\rho_{o_i,g_i} \in \tilde{\mathcal{P}}_{o_i,g_i}} C(\rho) \quad \forall i. \tag{4.13}$$

As T increases, lower cost paths are sampled, making this inequality harder to satisfy. Assuming $\tilde{\mathcal{P}}_{o_i,g_i}$ is constant, we can rewrite (4.10) as:

$$\frac{1}{2}||\mathbf{w}||^2 + \frac{\lambda}{D} \sum_i \big(C(\rho^i_{o_i,g_i}) - \min_{\rho \in \tilde{\mathcal{P}}_{o_i,g_i}} \big(C(\rho) + L_i(\rho)\big)\big). \tag{4.14}$$

This gives rise to an approach we call Approximate Maximum Margin Planning (AMMP). It is similar to MMP with the crucial difference that the planning step is executed by a sample-based planner and not a deterministic one, like A*. An important consequence is that $\tilde{\mathcal{P}}_{o_i,g_i}$ changes every time we invoke the sample-based planner. Thus, AMMP can be thought of as sampling the *constraints* that we want our cost function to satisfy. The main advantage of AMMP is that it is not bound by the restrictive assumptions of A*, such as a discrete state-action space and no motion constraints.

However, an ineffective sampler could yield poor gradient estimates and thus poor solutions. In fact, Ratliff et al. [97] argue that, for this reason, sample-based planners like RRT are not suited to learning cost functions from demonstration, as the paths sampled by RRT are heavily biased during the sampling process and thus highly suboptimal. However, asymptotically optimal sample-based planners like RRT* are better able to plan low-cost paths [51] and thus well suited for use within AMMP.

4.4.2 Rapidly Exploring Learning Trees

As suggested above, a simple way implement AMMP is to use RRT* as the sample-based planner. RRT* can sample low-cost paths, allowing AMMP to learn a good cost function. Furthermore, RRT* can cope with large and continuous state-action spaces with motion constraints. However, the result is a computationally expensive algorithm that calls the planner $I \times |D|$ times over I iterations, given a dataset of size $|\mathcal{D}|$. Furthermore, sampling a separate set of points at every iteration could produce noisy gradients that

negatively affect convergence. In this section, we propose Rapidly Exploring Learning Trees (RLT*), which implements AMMP with an RRT* planner using a novel caching scheme to ensure both computational efficiency and more consistent gradients.

A key observation is that, in RRT*, only the rewiring step depends on the cost function. The sampling step is independent of it and, especially when motion constraints are present, contains the most computationally expensive operations. These are: 1) fitting and querying a data structure to find the nearest neighbor and the radius neighbors to a newly sampled point, 2) applying the steer function to all neighbours to and from the newly sampled point, and 3) checking for obstacles in these paths. By contrast, the only potentially expensive operation in the rewiring step is querying the cost of an edge.

Therefore, a key idea behind RLT* is to perform the sampling step just once for each data point and cache it for reuse throughout learning. Then, at each iteration, only the rewiring step needs to be repeated as the cost function changes.

Algorithm 2 describes the caching step, which takes as input p, the number of points to randomly sample from free space; s_{init}, the initial point; and η, the steer step size. For each randomly sampled point s_{rand}, we find the nearest neighbor, $s_{nearest}$, from the set of points in the vertex set V. We then create a new configuration point s_{new} by steering from $s_{nearest}$ to s_{rand}. Next, we query the radius neighbors, S_{near}, of s_{new} at a radius determined by $\min\{\gamma_{RRT^*}(\frac{\log(|V|)}{|V|})^{\frac{1}{d}}, \eta\}$. Here, d is the dimensionality of S, and γ_{RRT^*} is a constant based on the volume of free space (see [51]). Next, we determine which points in S_{near} can safely reach s_{new} through the chosen `Steer` function (lines 13-17). These forward paths are stored in $Paths_{fwd}$. We then perform the same procedure but this time checking the paths from s_{new} to S_{near} and store them in the set $Paths_{bwd}$ (lines 18-22). This algorithm turns the sampling process of RRT* into a preprocessing step. Consequently, the expensive `Nearest`, `Near`, `Steer` and `Safe` procedures only need to be repeated $|\mathcal{D}|$ times instead of $I \times |\mathcal{D}|$ times.

The output of Algorithm 2 is input to Algorithm 3, which resembles the rewiring procedure in RRT* [51] and returns a low-cost path to the goal. However, unlike RRT* rewiring, the vertices of the tree and their neighbors at each iteration are already known and contained within the point cache. This speeds computation while keeping consistency between the planners used during learning and final execution. As learning proceeds and the cost function changes, so does the wiring of this tree; however, the points involved do not change.

Algorithm 4 describes Rapidly Exploring Learning Trees (RLT*), which uses Algorithms 2 and 3. First, we initialize the weights, either randomly or using a cost function that simply favors shortest paths. Then, for each data point ρ_i, we calculate feature sums and run `cacheRRT`. The main learning loop involves cycling through all data points and finding the best path under a loss-augmented cost function. The feature sums of

Algorithm 2 `cacheRRT`(n,s_{init},η)

1: $P \leftarrow \emptyset$
2: $V \leftarrow s_{init}$
3: **for** $i = 0 \ldots n$ **do**
4: $s_{rand} \leftarrow SampleFree_i$
5: $s_{nearest} \leftarrow \texttt{Nearest}(V, s_{rand})$
6: $s_{new} \leftarrow \texttt{Get}(s_{nearest}, s_{rand})$
7: $S_{near} \leftarrow \texttt{Near}(V, s_{new}, \min\{\gamma_{RRT^*}(\frac{\log(|V|)}{|V|})^{\frac{1}{d}}, \eta\})$
8: $Paths_{fwd} \leftarrow \emptyset$
9: $Paths_{bwd} \leftarrow \emptyset$
10: $S_{fwd} \leftarrow \emptyset$
11: $S_{bwd} \leftarrow \emptyset$
12: **for** $s_{near} \ldots S_{near}$ **do**
13: $path_{fwd} = \texttt{Steer}(s_{near}, s_{new})$
14: **if** $\texttt{Safe}(path_{fwd})$ **then**
15: $Paths_{fwd} \leftarrow Paths_{fwd} \cup path_{fwd}$
16: $S_{fwd} \leftarrow S_{fwd} \cup s_{near}$
17: $path_{bwd} = \texttt{Steer}(s_{new}, s_{near})$
18: **if** $\texttt{Safe}(path_{bwd})$ **then**
19: $Paths_{bwd} \leftarrow Paths_{bwd} \cup path_{bwd}$
20: $S_{bwd} \leftarrow S_{bwd} \cup s_{near}$
21: $V \leftarrow V \cup s_{new}$
22: $P \leftarrow P \cup \{s_{nearest}, s_{new}, S_{fwd}, S_{bwd}, Paths_{fwd}, Paths_{bwd}\}$
23: **return** P

this path are calculated and subsequently the difference with the demonstrated feature sums is computed. At the end of each iteration, an average gradient is calculated and the cost function is updated. At convergence, the learned weights are returned.

In addition to saving computation time, the use of caching encourages consistency between the gradients computed in each iteration, as they are estimated from the same sampled points. The effect on the gradients, which resembles that of momentum [91], can improve performance, as our results in the next section show.

For RRT*, the dependence of $\tilde{\mathcal{P}}_{o_i,g_i}$ on the time budget T is hard to quantify since it depends on the size and nature of S as well as the cost function we are using, which also changes with every iteration. For this reason, we resort to an experimental assessment of the ability of RRT* to sample the right constraints at every iteration of RLT* and hence effectively learn a cost function from demonstration.

4.5 Experiments

We evaluate RLT* by comparing it to MMP, implemented using an A* planner, and RLT*-NC, an ablated version of RLT* that does not use caching. We consider three experimental settings: 1) a simulated holonomic robot, 2) a simulated non-holonomic robot, and 3) a real telepresence robot.

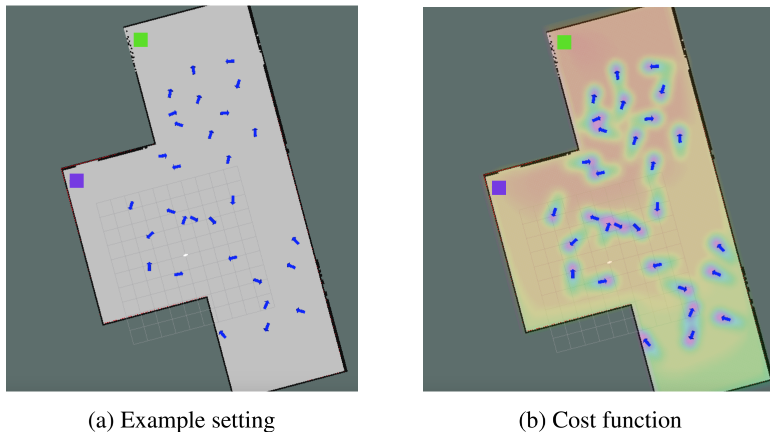
Our experiments take place in the context of socially intelligent navigation. IRL has been widely used in this setting [41, 83, 119] because it is usually infeasible to hard-code the cost functions that a planner should use in complex social situations.

Algorithm 3 $\texttt{planCachedRRT}^*(P, s_{init}, c())$

1: $E \leftarrow \emptyset$
2: $V \leftarrow s_{init}$
3: **for** $i = 0 \ldots |P|$ **do**
4: $\quad (s_{nearest}, s_{new}) \leftarrow P_i$
5: $\quad (S_{fwd}, S_{bwd}) \leftarrow P_i$
6: $\quad (Paths_{bwd}, Paths_{fwd}) \leftarrow P_i$
7: $\quad V \leftarrow V \cup s_{new}$
8: $\quad s_{min} \leftarrow s_{nearest}$
9: $\quad c_{min} \leftarrow \texttt{Cost}(s_{nearest}) + c(path_{s_{nearest}, s_{new}})$
10: $\quad$ **for** $j = 0 \ldots |S_{fwd}|$ **do**
11: $\quad\quad s_{fwd} = S^j_{fwd}$
12: $\quad\quad path_{fwd} = Paths^j_{fwd}$
13: $\quad\quad C_{near} \leftarrow \texttt{Cost}(s_{fwd}) + c(path_{fwd})$
14: $\quad\quad$ **if** $C_{near} < c_{new}$ **then**
15: $\quad\quad\quad s_{min} \leftarrow s_{fwd}; c_{min} \leftarrow c_{near}$
16: $\quad E \leftarrow E \cup \{(s_{min}, s_{new})\}$
17: $\quad$ **for** $j = 0 \ldots |S_{bwd}|$ **do**
18: $\quad\quad s_{bwd} = S^j_{bwd}$
19: $\quad\quad path_{bwd} = Paths^j_{bwd}$
20: $\quad\quad C_{new} \leftarrow \texttt{Cost}(s_{new}) + c(path_{bwd})$
21: $\quad\quad$ **if** $C_{new} < \texttt{Cost}(s_{near})$ **then**
22: $\quad\quad\quad s_{parent} \leftarrow \texttt{Parent}(s_{bwd})$
23: $\quad\quad\quad E \leftarrow E \setminus (s_{parent}, s_{bwd}) \cup (s_{new}, s_{bwd})$
24: $\rho_{min} \leftarrow \texttt{minCostPath}(V, E, c())$
25: **return** ρ_{min}

Algorithm 4 $\texttt{RLT}^*(\mathcal{D}, p, \eta, \lambda, \delta)$

1: $\mathbf{w} \leftarrow \texttt{initialiseWeights}$
2: $\tilde{\mathbf{F}} \leftarrow \emptyset$
3: $R \leftarrow \emptyset$
4: **for** ρ^i in $\mathcal{D}$ **do**
5: $\quad \tilde{F}_{\rho^i} \leftarrow \texttt{FeatureSums}(\rho^i)$
6: $\quad \tilde{\mathbf{F}} \leftarrow \tilde{\mathbf{F}} \cup \tilde{F}_{\rho^i}$
7: $\quad r_i \leftarrow \texttt{cacheRRT}(p, s^{\rho^i}_{init}, \eta)$
8: $\quad R \leftarrow R \cup r_i$
9: **repeat**
10: $\quad \nabla_{\mathbf{w}} \leftarrow 0$
11: $\quad$ **for** ρ^i in $\mathcal{D}$ **do**
12: $\quad\quad c() \leftarrow \texttt{getCostmap}(\mathbf{w}) + L(\rho^i)$
13: $\quad\quad r_i \leftarrow R\{i\}\,;\, \tilde{F}_i \leftarrow \tilde{\mathbf{F}}\{i\}$
14: $\quad\quad \rho \leftarrow \texttt{planCachedRRT}^*(r_i, x^i_{init}, c())$
15: $\quad\quad F_i \leftarrow \texttt{FeatureSums}(\rho)$
16: $\quad\quad \nabla_{\mathbf{w}} \leftarrow \nabla_{\mathbf{w}} + \tilde{F}_i - F_i$
17: $\quad \nabla_{\mathbf{w}} \leftarrow \mathbf{w} + \frac{\lambda}{|\mathcal{D}|} \nabla_{\mathbf{w}}$
18: $\quad \mathbf{w} \leftarrow \mathbf{w} - \delta \nabla_{\mathbf{w}}$
19: **until** convergence
20: **return** $\mathbf{w}$



(a) Example setting (b) Cost function

Figure 4.1: (a) A randomised instance of the social navigation task. Arrows denote the position and orientation of people in the scene. The robot is represented by the magenta box and the goal location is represented by the green box. (b) The corresponding cost function. Red denotes *low* cost, while purple denotes *high* cost.

Having the ability to quickly and effectively learn social navigation cost functions from demonstration would be a major asset for robots that operate in crowded environments such as airports [118], museums [116] and care centers [104].

4.5.1 Simulated Experiments

We first consider randomly generated social environments in simulation, such as the one shown in Figure 4.1a. Each arrow in the figure represents a person's position and orientation. The robot is given the task of navigating from one point in the room to another. While it is aware of the orientation and position of different people, it has no idea how to trade off reaching the target quickly with avoiding people and obstacles, i.e., the cost function is unknown. Instead, the robot is given a dataset of demonstrations $\mathcal{D}$. Each demonstration ρ_i is a set of configurations $s = (x, y)$ representing positions of the robot in the configuration space and each demonstration takes place for a different random configuration of the social environment, i.e., the people are at different positions and orientations every time. The task is to use $\mathcal{D}$ to extract a cost function that enables socially intelligent behavior.

The features we use can be divided into three categories. The first category encodes proxemics to the people present in the scene, i.e., the social features. Within this category we consider two variations, for reasons explained in the following section.

- **Social feature set 1 (S1)**: Three isotropic Gaussian functions with different means, centered in front, behind, and on the person.

- **Social feature set 2 (S2)**: Three field-of-view features. The features have a value of 1 if the robot is within a certain distance and angle from the person.

The second category of features encodes the distance from the target location using linear, exponential, and logarithmic functions. The third category encodes the obstacle cost using a stable function of the reciprocal of the distance from the nearest obstacle. Figure 4.1b shows an example cost function over the whole configuration space for the configuration in Figure 4.1a. We use different functions for human and target proximity, to allow for more degrees of freedom when modeling the underlying cost function. Sufficient regularization ensures that that the model does not overfit.

Evaluation

To evaluate our algorithms, we generate a dataset $\mathcal{D}$ by planning near-optimal paths from initial configurations s_o to goal configurations s_g under a ground-truth cost function $c_{gt}()$ derived from ground-truth weights $\mathbf{w}_{gt}$ and features $\mathbf{F}_{gt}$. A fully optimal path can only be derived asymptotically in terms of either time for RRT*, or resolution for A*. In practice, however, we found that planning for 100 seconds using RRT* achieves a path that is nearly optimal; running longer leads to negligible changes in path cost. The resulting ground truth dataset enables a quantitative empirical evaluation, which is otherwise problematic in IRL [105, 119]. For each path ρ generated by the learning algorithm, we know its cost under the ground-truth cost function and features is simply $\mathbf{w}_{gt}^T\mathbf{F}_{gt}(\rho)$. Furthermore, we can compute the *cost difference* between the learned path and the demonstrated path with respect to ground truth:

$$Q(\rho, \rho_i, \mathbf{w}_{gt}) = \mathbf{w}_{gt}^T(\mathbf{F}_{gt}(\rho) - \mathbf{F}_{gt}(\rho_i)), \tag{4.15}$$

which is our primary performance metric. Note that, if the demonstration path ρ_i is optimal under $\mathbf{w}_{gt}$, then $Q(\rho, \rho_i, \mathbf{w}_{gt}) \geq 0$. For our holonomic simulated experiments, we consider two learning scenarios.

1. **Unknown weights**: only $\mathbf{w}_{gt}$ is unknown. The demonstrations and the learning algorithm share social feature set S1.

2. **Unknown weights and features**: $\mathbf{w}_{gt}$ and $\mathbf{F}_{gt}$ are unknown. S2 is used to generate the demonstrations and S1 is used for learning.

The first scenario evaluates each algorithm's ability to learn good cost functions when provided only with limited demonstrations of the task. The second scenario introduces a feature discrepancy to better simulate real-world settings, since it is unlikely that the features we define will exactly match those implicitly used by the human demonstrator.

We also document the total learning time for K iterations for the algorithms under comparison. All algorithms were implemented in Python, share similar functions, and were not optimized for speed apart from the caching scheme in RLT*. Finally, we perform a qualitative evaluation by visually comparing the learned cost functions for each algorithm and the paths they generate against ground truth.

Holonomic Robot Results

Our dataset $\mathcal{D}$ consists of 20 trajectories from random social social situations. We split $\mathcal{D}$ into $\mathcal{D}_{train}$ and $\mathcal{D}_{test}$, each with 10 trajectories. After training on $\mathcal{D}_{train}$, the cost difference of a cost function is evaluated on D_{test} using (4.15). The process is repeated 7 times for the same dataset but with different random compositions of $\mathcal{D}_{train}$ and $\mathcal{D}_{test}$. All learning algorithms are initialized using the same cost function that only favors shortest paths.

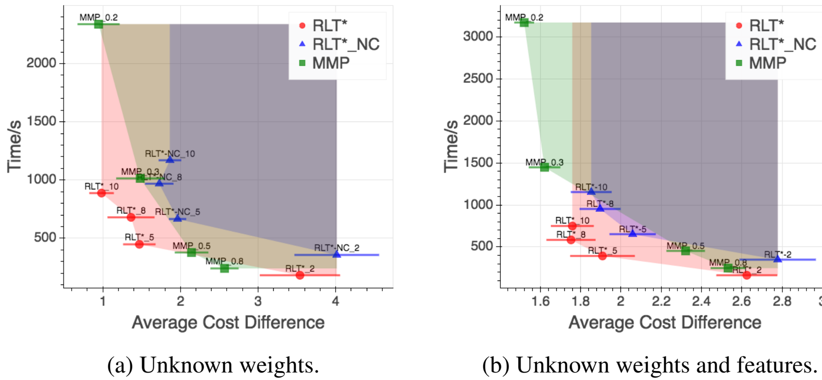
As mentioned earlier, planning time and grid resolution affect the performance of RRT* and A^*, respectively. To make a fair comparison, we vary these two quantities for each algorithm and plot cost difference against learning time at each setting. We can then identify which algorithms at which settings comprise the Pareto front, i.e., are undominated with respect to cost difference and learning time.

Figures 4.2a and 4.2b show the results for the two scenarios described earlier. MMP_X (green) refers to MMP with X meters of grid resolution while RLT_X (red) and RLT*-NC_X (blue) refer to RLT and RLT without caching, respectively, with X seconds of planning. The shading represents an interpolation of the performance between settings for each method. In this way an area of single color illustrates hypothesized domination of that method over another, given the same amount of time. Since lower is better for both cost difference and learning time, the closer a point is to the bottom left corner, the better.

RLT* (red) comprises a large majority of the Pareto front, demonstrating good performance and generalization in reasonable time. However, the performance of RLT* and RLT*-NC significantly degrades at very low planning times (2 seconds) because AMMP cannot sample good enough paths to compute a useful gradient.

Note that caching not only speeds learning in RLT*, it also modestly reduces cost differences. This suggests that the caching scheme introduces extra robustness and generalisation capabilities within the algorithm. As mentioned in Section 4.4.2, we hypothesise that the caching scheme improves learning by making the gradients smoother and more consistent, as with momentum. To confirm this, we plot the inner product between successive gradients during learning in Figure 4.3. The plot confirms that subsequent gradients in RLT* are more similar.

Finally, note that MMP's learning time scales exponentially with the size of the



(a) Unknown weights. (b) Unknown weights and features.

Figure 4.2: Learning time vs. cost difference on the holonomic robot for MMP (green), RLT*-NC (blue), and RLT (red) at different planning fidelities. On both axes *lower is better*.

grid. This, however, is not solely due to the graph getting larger but also because A* search scales poorly with the complexity of the cost function itself. Since it relies on an admissible heuristic that for complex cost functions is no longer tight, A* must expand many more states. RLT* is less susceptible to such problems.

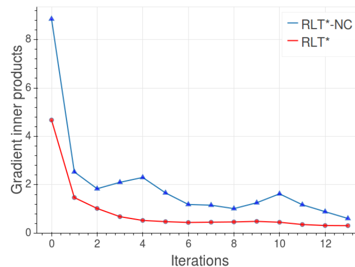


Figure 4.3: Inner product of successive gradients during learning. The smoother curve for RLT* suggests that fixing the sampled points during learning makes the gradients more consistent.

Non-Holonomic Robot Results

As mentioned earlier, a potential advantage of RLT* is that it can efficiently handle learning in the presence of motion constraints. In this section, we consider a robot planning in a three-dimensional space, (x, y, θ), representing the position and orientation

	RLT*	RLT-NC
Average Cost Difference	0.1	2.31
Learning time (s)	1320	18530

Table 4.1: Non-holonomic robot results.

of the robot. The robot's motion is further subject to the following motion constraints:

$$\dot{x} = v \times \cos(\theta), \tag{4.16}$$

$$\dot{y} = v \times \sin(\theta), \tag{4.17}$$

$$\dot{\theta} = \omega, \tag{4.18}$$

where v, ω are the linear and angular velocities respectively. To meet these constraints, we use the POSQ `Steer` function [86]. Since this approach gives a local closed-loop policy between two vertices of the tree, it is more robust to noise and uncertainty in the motion than an open loop trajectory. Furthermore, it has been shown to produce smooth paths between feasible configurations [86]. Evaluation is done in the same way as in the previous section, except that we compare RLT* only to RLT*-NC and not MMP, as the latter cannot handle motion constraints. RLT*-NC was given 100 seconds to plan, in which case about 3000 configurations were sampled, and RLT*'s cache was set to this size.

Table 4.1 shows that RLT* is an order of magnitude faster than RLT*-NC, while achieving a lower cost difference. The kinematic constraints contribute to this speedup since they make the `Steer` and `Safe` procedures, which are cached, more expensive. As in the holonomic case, RLT* resulted in smoother learning, confirming the results of Figure 4.3.

Figure 4.4 shows an instance of the types of paths generated by our method when compared to ground truth. This specific example was drawn from the validation set and is thus not a case of overfitting.

4.5.2 Real Robot Experiments

In this section, we apply RLT* to real, human demonstrations using a telepresence robot, shown in Figure 5.2a, in a social navigation scenario. Furthermore, we deploy the learned cost function on the actual robot.

The experiments take place in a simplified version of the social scenarios we have seen in the previous section. There are two people in the scene at different positions and orientations. A human demonstrator is asked to execute paths for different initial and final conditions across the room. The task is similar to the one used in [83] and

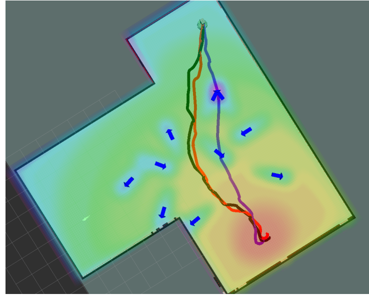


Figure 4.4: Qualitative evaluation of simulated results. The color map denotes the learned cost function. Red denotes *(*low) cost and green *high* cost. The learned path (red) is quite similar to the demonstrated path (black) with a clear improvement over the path before learning (purple).

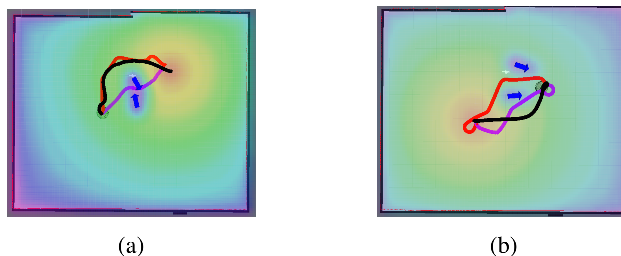
Figure 4.5: The telepresence robot used in our experiments.

the purpose is to find cost functions that account for potential relationships between people depending on their orientation with respect to each other. For example, if people are facing each other, they are likely engaged in conversation or a similar activity (e.g., taking a photograph) that should not be interrupted. By contrast, if they are looking away from each other and there is enough distance between them then, it might be better to pass between them if doing so yields a shorter path.

To collect data for learning and validation, we use an off-the-shelf telepresence system augmented with several sensors that allow localization and perception [104]. We use an Optitrack motion capture system to accurately collect ground truth data of both people and robot positions. RLT* learns a cost function from this data using the social feature set S1 and the rest of the features described earlier.

Since quantitative evaluation is difficult using real data, as no ground truth is avail-

able, we perform a qualitative evaluation instead. Figure 4.6 shows some representative cases that arose during learning. Figure 4.6a shows a case where RLT* produced paths (red) that are quite similar to the demonstrated ones (black). Figure 4.6b shows an instance instances where the learned paths are reasonable even though they are not similar to the demonstrated paths.



(a) (b)

Figure 4.6: Real demonstrated paths (black), learned paths (red), and shortest paths (purple). In (a), the learned path is similar to the demonstrated path; in (b) it is different but reasonable. The paths are laid over the learned cost function.

Finally, we successfully deployed the learned cost function, also shown in Figure 4.6, on a real telepresence robot. A video demonstrating this deployment can be found the supplementary videos accompanying this thesis [1]. Even though our planner outputs a full policy in terms of angular and linear velocities, to follow the prescribed path we used an elastic bands local planner in order to deal with dynamic changes in the environment.

4.6 Discussion

In order to perform IRL in path planning domains, one must somehow enumerate all the other paths that the expert could have taken and place the lowest cost on the path taken. This chapter argues that RRT* is an effective way of sampling high cost paths and can therefore be used within the IRL process. Immediate directions for future work would include methods for improving both the inner planning loop, and the outer cost learning loop, in the IRL framework.

Sample based planners, of which RRT* is an instantiation of, is an active research field in the robotics community. Other methods consider smarter sampling schemes [33] different data structures, and sometimes even employ graph based planners to inform the sampling process [87]. As long as some guarantee on the optimality of these improved methods can be guaranteed, the idea of AMMP can be applied with potentially better results than the ones reported in this paper.

[1] https://surfdrive.surf.nl/files/index.php/s/oYgoyimLFzIJe8I

Some recent work [90] considers the paths sampled by RRT* as a way to compute the normalizing constant required by alternative IRL methods such as path integral IRL [4]. This also shows competitive and often better results than AMMP because AMMP computes the gradient based on the features of a single path, which can induce high variance estimates of the gradient.

The cost learning capability of our method could further be improved by lifting the assumption of a linear feature parametrization of the cost. This can be effectively done using for example neural networks [28, 43, 123]. Learning complex non-linear cost functions for sample based planners is certainly a promising direction for future research.

A limiting assumption in the methods presented in this chapter is that of stationarity. In all the considered scenarios the people present in the scene are static, which relatively unrealistic. A clear way to remove this assumption is to execute the planned paths during learning in a dynamic environment and allowing the RRT* to re-plan the path. This of course would demand other, perhaps less limiting, assumptions to be made based on the local optimality of paths, as in [68].

Finally, while RRT* can be applied to mobile robot navigation, it is particularly useful in domains with many degrees of freedom such as manipulation. It is uncertain how RLT* will fair in these situations because the search tree is usually not able to cover the whole space of possible solutions. State of the art sampling based planners such as Batch Informed Trees [34] are however faster and produce significantly better results than RRT*. Learning costs for complex manipulation tasks from demonstration is a popular problem for robotics and machine learning research, making the methods developed in this chapter applicable to much more than social navigation.

4.7 Conclusion

In this chapter, we propose Rapidly Exploring Learning Trees (RLT*), which learns the cost functions of Rapidly Exploring Random Trees (RRT) from demonstration, thereby making inverse learning methods applicable to more complex tasks. Our approach extends the Maximum Margin Planning to work with RRT* cost functions. Furthermore, it uses a caching scheme that greatly reduces the computational cost and improves performance. Our results in simulated social navigation scenarios show that RLT* achieves better performance at lower computational cost, even when there is a discrepancy between the features used for demonstration and learning. Furthermore, our results show that RLT* can handle more complex configuration spaces with motion constraints. Finally, we use RLT* to learn a cost function using data from real demonstrations on a telepresence robot and successfully deployed that cost function back on the robot.

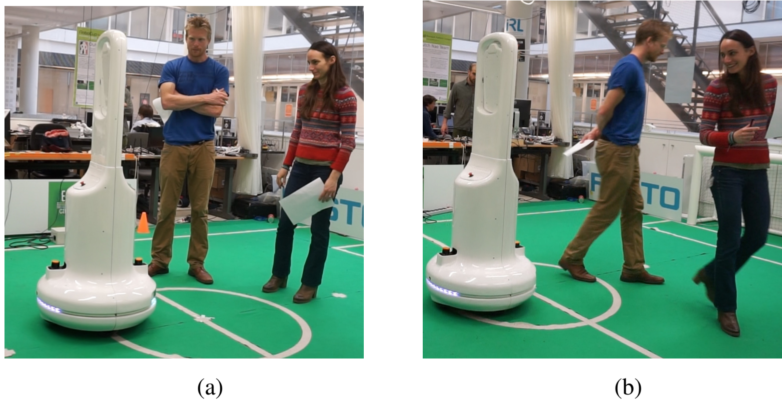
Chapter 5

Deep Behavioral Social Cloning

5.1 Introduction

In recent years, robots have been migrating out of conventional, constrained, industrial environments and into more social and less structured ones such as museums [116], airports [118], restaurants [93] and care centers [104]. This shift poses countless challenges for robotics, from hardware to high-level behavior design. A particularly vexing challenge is to imbue robots with sufficient social intelligence, i.e., to ensure that they respect social norms when interacting with humans or completing tasks in their presence. For example, a robot facing a group of people must maintain an appropriate distance, orientation, level of eye contact, etc.

The previous chapter explored the possibility of learning social cost functions for robotic navigation by combining robotic based planners with inverse reinforcement learning (IRL). There are however other tasks that a social robot might need to execute such as interacting with and following groups of people. Such tasks are typically simpler than navigation in the sense that they can be performed with little planning, i.e., the robot does not need to look into possible future situations that a current action may bring about. However the robot still needs to make a well educated decision about how it should respond to a certain social situation. This means that the learned control policies should be robust to inevitable errors in the robot's perception system. For example, a social robot must be able to detect and localize people in its environment. If errors in such perception occur while demonstration data is being collected, then the LfD algorithm may misinterpret the demonstration, e.g., by inferring that the robot was avoiding a person when actually no person was present. Furthermore, if perception errors occur when the learned policy is deployed, then the robot may behave incorrectly [19].



(a) (b)

Figure 5.1: The two social interaction tasks we consider in this chapter. (a) The group interaction task: the robot should reconfigure itself to facilitate conversation in a group. (b) The following task: the robot should follow a group of people. These pictures were taken during a human evaluation session where DBSoC was compared to different baselines.

A second challenge is to cope with the dynamic size of the perceptual input. A robot that interacts with a group of people must be able to cope with groups of different sizes, and even people coming and going within a single conversation. This is a poor match for most LfD methods, which assume observations in the form of fixed-length feature vectors.

Third, in order to be useful for non-experts, the algorithm used must be sufficiently generic. In other words, it should be able to learn a wide set of tasks with a single algorithm, by varying only the data.

In this chapter, we propose deep behavioral social cloning (DBSoC), a new LfD architecture that overcomes these challenges. Behavioral cloning (BC) is an approach to LfD in which a control policy is directly inferred using supervised learning, i.e., the demonstrated actions are treated as labels for the corresponding observations, and a mapping from observations to actions is learned. By leveraging the power of deep learning, DBSoC makes behavioral cloning practical for social robotics. In particular, DBSoC 'paints' the output of the robot's person detection and localization system onto a 2D 'image', which is then fed to a convolutional neural network. In this way, DBSoC can cope elegantly with varying numbers of people, as well as automatically discover useful features for selecting actions. In addition, DBSoC employs long short-term memory (LSTM) [44] network layers, allowing it to condition action selection on its history of observations and thus filter out transient perceptual errors.

We apply DBSoC to learn two social behaviors for a semi-autonomous non-holonomic

telepresence robot. In the first task, the robot must continually adjust its position and orientation while interacting with a group of people of dynamic size and position. In the second task, the robot must follow two people as their relative positions change.

We evaluate DBSoC on these two tasks by deploying the learned policies on a real telepresence robot and having it interact with actual human subjects. Our quantitative and qualitative results show that DBSoC performs well against a naive controller and outperforms a *gradient boosting regression* baseline. Furthermore, ablation experiments confirm that each element of DBSoC is essential to its success.

5.2 Problem Statement

In this chapter, we consider two social tasks for a telepresence robot. Such a robot acts as an avatar for a remotely present *pilot* controlling the machine. While telepresence robots are typically manually controlled, we aim to automate low-level control so that the pilot need only give high-level commands telling the robot, e.g., to follow a particular person or support a conversation with appropriate body poses. Because telepresence robots are inherently social, and especially since they represent their pilots as avatars, it is essential that any learned behavior be socially appropriate. In this section, we describe the two particular telepresence tasks considered in this paper, and the demonstration data we collected. For both tasks, we assume the robot is equipped with an imperfect person detection and localization system. We also assume it is operating in an unstructured environment containing a dynamic number of people, some of whom may not be involved in the pilot's conversation.

5.2.1 Group Interaction Task

In the first task, the robot must maintain a socially appropriate position and orientation as the location, position, and size of the group with which the pilot is interacting changes. To do so, the robot must distinguish people who are part of the conversation from those who are merely in the scene. Though this task, shown in Figure 5.1a, looks deceptively simple, manually coding such behavior, or even deciding on what features such behavior should condition, is quite difficult in practice, making it an ideal application for LfD.

Previous work has considered what effect robot repositioning following a change in a group's size and shape has on the people in that group [63, 120]. Here, we consider how to automate such repositioning using LfD. Other work considers detecting groups of people in crowded environments [65] and then interacting with them. In our approach, groups are not explicitly detected, but the learned control policy may implicitly do so when deciding how to act.

5.2.2 Following Task

In the second task, the interaction target(s) are moving and the robot should follow them. The robot should be able to deal with variability in the formation of the people it is following. Furthermore, it should be robust to false positive detections and be able to retain a following behavior despite detecting other people, who are in the scene but not relevant to the task. This task is shown in Figure 5.1b. Following behaviors can be challenging because the robot's sensors often suffer from reliability issues as velocities increase [55].

Previous work used an extended social force model enriched with a goal prediction module to allow a robot to walk side by side with a person [26]. The system tests different social force parameters and allows the user to give feedback about the experience. Based on this feedback, the robot learns to adapt itself to the specific person. Our work investigates the possibility of bypassing such models and simply learning the required behavior robustly through data.

Knox et al. [54] train a robot to perform several tasks that are similar to ours, such as maintaining a conversational distance and "magnetic control". However, training is done using human generated reward and reinforcement learning through the TAMER framework [53]. Furthermore, in contrast to our work, their approach can handle only a single interaction target and assumes perfect sensing.

More generally, in social robotics settings, LfD has been used to learn high level behaviors [74, 75] and cost functions for social path planning through the use of *inverse reinforcement learning* (IRL) [1], [41]. In this work, we show how low level reactive social policies can be learned from demonstration. Outside of social settings, LfD is most commonly used for complex manipulation tasks [6]. Deep learning has also been used in an end-to-end fashion to perform behavioral cloning in self-driving cars [11]. However, such an approach is highly intensive in terms of both data and computation. By contrast, we rely for people detection on a sensor fusion pipeline that uses both the RGBD camera and the laser rangefinders of the robot to enable $360°$ tracking. Consequently, we require only a modest amount of demonstration data (approximately one hour), perform all processing on the robot, and cope successfully with the robot's limited RGBD field of view.

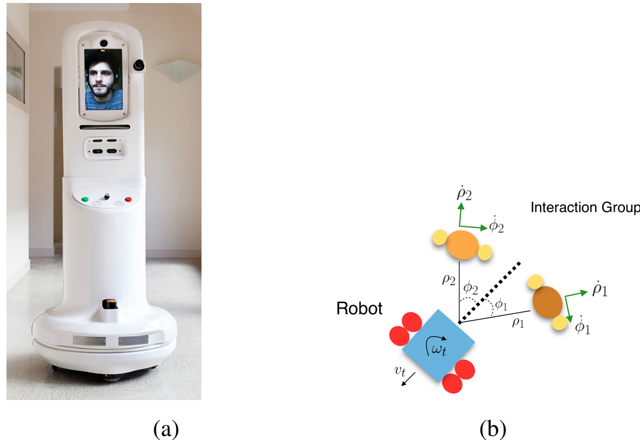
5.2.3 Data and Experiments

To collect the data required for learning, we performed one set of experiments for each of the tasks described above. Every experimental session involved two experimenters, one or more volunteers from the University of Amsterdam, and a TERESA[1] intelligent

[1] http://www.teresaproject.eu.

telepresence system, shown in Figure 5.2a. One experimenter acted as the pilot while the other acted as an interaction target. The interaction target also instructed the volunteers to join or leave the interaction and change the shape of the interaction group (triggering an appropriate reaction from the pilot) in order to generate sufficient variability in the data. For the follow task, the two experimenters walked with the robot, making sure that at least one of them was tracked at all times, in order to prevent false data from being recorded.

During these interactions, we recorded the positions and linear velocities of all people detected and tracked by the robot. We also recorded a binary label, indicating the primary interaction target for the robot, i.e., the most important person in the interaction, as indicated by the pilot. The observation o_t of K people tracked at time t, where the first person is the primary interaction target, is therefore defined as, $o_t = \{(\rho_1, \phi_1, \dot{\rho}_1, \dot{\phi}_1, 1), ...(\rho, \phi_{K_t}, \dot{\rho}_{K_t}, \dot{\phi}_{K_t}, 0)\}_t$, where ρ_k and ϕ_k are the distance and angle from the robot respectively and $\dot{\rho}_k$ and $\dot{\phi}_k$ represent the associated velocities (see Figure 5.2b). We also recorded the linear and angular velocity commands given by the pilot, $a_t = (v_t, \omega_t)$, where a_t denotes the action at time t. Recording took place at 10Hz, which is the control rate the robot uses. Data was gathered in multiple different environments, each containing different numbers of people and different sources of false positives for people detection. We collected in total 43512 and 27311 data points for the following task and group interaction tasks, respectively. This amounts to approximately two hours of data.



(a) (b)

Figure 5.2: (a) The TERESA telepresence robot used in our experiments; (b) the data collected for a single frame (t) during our experiments.

5.3 Method

In this section, we describe deep behavioral social cloning (DBSoC), the learning architecture used to learn a mapping from robot observations to actions for the tasks described above. We first describe our behavioral cloning approach and motivate the use of neural networks in this application. We then build the network architecture piece by piece by looking at different aspects of the data such as input representations, temporal aspects, and output mappings.

Behavioral cloning is a simple approach to LfD that treats the problem as one of supervised learning. The (sequences of) observations are viewed as inputs s_t and actions a_t as labels. Then, any supervised learning paradigm (regression, classification) and algorithm (decision trees, Gaussian processes, neural networks) [10] can be used to learn a mapping from s_t to a_t, which then constitutes a control policy. As mentioned in Section 5.2 an alternative to BC is IRL [1], which extracts a cost function from the demonstrations that is subsequently used to plan (or learn [12]) a policy. IRL however, usually requires either a model of the social environment, which is hard to obtain, or many hours of interaction, which is time inefficient. Behavioral cloning is simpler, more practical, and does not require access to a model or the ability to try out candidate policies during learning.

DBSoC uses behavioral cloning with *neural networks* (NNs). Using NNs has two key advantages in this setting. First, given the need for a generic algorithm that can tackle multiple tasks, NNs can learn features, e.g., that can detect if someone is part of a group, that can be reused across related tasks. Second, NNs are modular, allowing us to swap in different modules depending on the high-level application. For example, we describe below how we use recurrent neural networks to deal with noisy perceptual input.

5.3.1 Inputs and Convolutional Layers

An important characteristic of our observation vector o_t is that its length depends on the number of people around the robot, which varies over time. This is problematic for many learning algorithms, which expect a fixed-length input vector. One way to alleviate this problem would be to fix K to some reasonable constant. If the number of people in the data is more than K, only the K closest are kept. If the number is smaller than K, then only the first K positions of the vector are filled and the rest set to zeros. However, this implies that one or more people are in the same location as the robot, since $\rho, \theta = (0, 0)$, which could mislead the learning algorithm. Another option would be to add a binary feature to our observation vector to indicate whether the detection is

a valid one, yielding, e.g.:

$$o_t = \{(\rho_1, \phi_1, \dot{\rho}_1, \dot{\phi}_1, 1, 1), ...(\rho_{K_t}, \phi_{K_t}, \dot{\rho}_{K_t}, \dot{\phi}_{K_t}, 0, 0)\}_t \tag{5.1}$$

While this representation accurately describes the observation, it is not convenient for learning. Forcing the algorithm to learn to ignore observations about people whose last bit is zero is an unnecessary additional complication. In addition, the representation ignores crucial symmetries, e.g., rearranging the order of the people in the vector yields a completely different input that describes exactly the same state.

To avoid these difficulties, we instead imprint the information in the observation onto a 2D image. In particular, we fix a maximum distance x_{max}, y_{max} from the robot and 'paint' all the people around the robot within that area on a ternary image, i.e., each pixel can have one of three values: $0 =$ no person is present, $1 =$ a person is present, $2 =$ a primary interaction target is present. We also denote the velocities of people as lines extending from the detections in the direction of the detected velocity and with a length corresponding to its magnitude. Figure 5.3 shows an example of such an image. It is straightforward to augment this representation with, e.g., people's orientations or the positions of non-human obstacles, should they be detectable.

A potential downside of this approach, however, is a great increase in the dimensionality of our representation. We tackle this problem by applying *convolutional neural networks* (CNNs) to this input representation. CNNs perform well on spatially structured images. Although the dimensionality of the input is large, the number of possible images is limited. CNNs have also proven effective at automatically extracting high-level features from the raw pixel data. This in turn means that they would be better suited in building an implicit representation of what a group is. Our results in the next section show that in practice this representation together with CNNs is not only more convenient, but also improves learning and the stability of the resulting behavior.

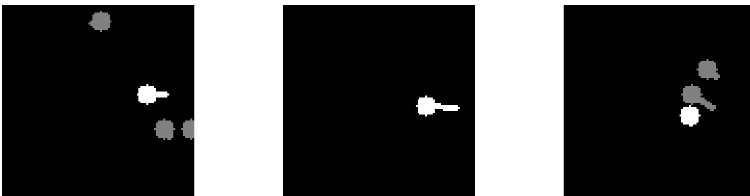


Figure 5.3: Example images representing the state of the robot. Circles denote people, lines denote velocity, and the white circle represents the primary interaction target.

5.3.2 Dynamics and Long Short-Term Memory

Using a single observation as input to a convolutional neural network that outputs a control signal is problematic for three reasons:

1. A single observation instance o_t does not contain higher order information such as acceleration.

2. The people detection module is imperfect and generates some false positives. Conditioning only on the most recent observation makes it impossible to filter such false positives out.

3. Velocity estimates, while helpful, are also inaccurate and noisy, and cannot be improved using only the most recent observation.

These problems can be avoided by using *recurrent neural networks* (RNNs), which implicitly allow the network to condition on its entire history of inputs. *Long short-term memory* [44] is a standard way of implementing recurrence in NNs. Our architecture uses LSTMs to deal with the time-dependent nature of the problem and achieve robustness in the face of uncertain detections.

5.3.3 Outputs and Overall Architecture

After processing by the recurrent layers, the architecture then outputs two real continuous values representing the linear and angular velocities $(v_{pred}, \omega_{pred})$ of the robot. An alternative would be to discretize the outputs and treat the prediction as a classification problem. While doing so has some benefits, e.g., allowing multi-modal output distributions, we chose regression for two reasons. First, since the right discretization is problem specific, a regression approach is more general. Second, continuous outputs can generate smoother behavior.

To train our network, we use backpropagation with the ADAM [52] optimizer to minimize the *mean squared error* (MSE) between the network's prediction and the data.

Figure 5.4 shows the complete resulting architecture. We employ three convolutional layers consisting of 10 3×3 filters at each layer. We use ReLU nonlinearities followed by max-pooling of size 2. The output from the last convolutional layer is flattened and processed through two LSTM layers of size 200 and 100 respectively. Finally, a densely connected layer is used to output the two required control values. Because the robot selects action s_t at 10Hz, the trained network must be able to forward propagate at that rate. We found that our network could do so comfortably on an Intel i7 processor.

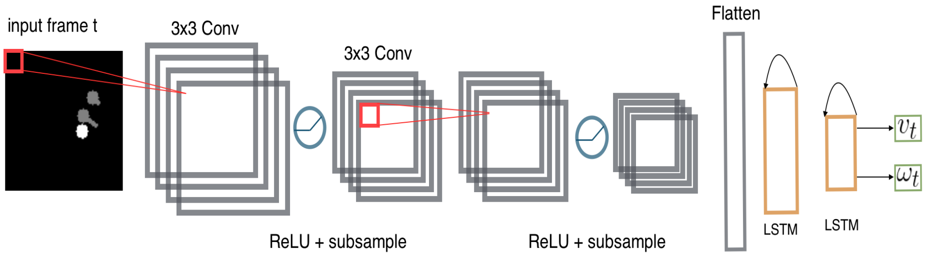


Figure 5.4: The complete DBSoC architecture. The image representing the state passes through two convolution-nonlinearity-subsample layers before being fed through two LSTM layers that output the control signals $(v_{pred}, \omega_{pred})$.

5.3.4 Preventing Covariate Shift

A well known failure mode of BC is that of covariate shift. This arises because, while the policy is trained on a dataset that is assumed to be stationary, it is deployed in an inherently non-stationary environment. A policy trained using BC thus does not recover easily from states not seen in the training data and may drift. This phenomenon motivates methods such as DAGGER [100], where the expert policy is queried for the right action at different instances of learning. Although in our setting DAGGER would be hard to apply, we found it was essential to train the robot in a similar manner. Specifically, we performed training sessions iteratively. After a partial data collection, the robot was trained and the policy deployed. During deployment, data of correcting actions was collected whenever the robot behaved inappropriately. The dataset was augmented with this data and the process was repeated.

5.4 Evaluation and Results

We evaluate DBSoC by comparing it against three other learning baselines. The first is an off-the-shelf regression method. Due to the mixed types in the inputs, we use *gradient boosting regression* (GBR), which is considered one of the best off-the-shelf algorithms for nonlinear regression. We use this baseline to show that, for traditional regression methods to work well, task-specific feature engineering would need to be performed. Inputs to this baseline are the concatenated observation vectors for K people around the robot (as described in Section 5.3.1) for T time-steps. In this case, $K = 4$ and $T = 5$ so the input vector representations is $K \times T \times 6 = 120$. We did not use $T = 20$ as DBSoC does because the explosion in input dimensionality caused a degradation in performance. The second baseline is a deep architecture that omits the convolutional layers and only employs the LSTM layers. The input representation is a

vector concatenating the features of $K = 4$ people around the robot. The third baseline is an architecture with no memory that uses only the convolutional layers.

When evaluating these algorithms, we are concerned not only with minimizing error on the data but also with how they perform when placed on the robot and tested with humans. Therefore, our evaluation relies on a mixture of quantitative and qualitative metrics.

5.4.1 Mean Squared Error

Since the algorithms are trained to minimize MSE, the first performance measure we consider is MSE on both the training set and a test set not used during learning. In our case, the test set contained of 20% of the data for each task, which amounts to 8702 and 5462 data points for the follow and group interaction tasks, respectively. Table 5.1 shows the performance of each method on both the training and test sets. We can see that DBSoC outperforms all baselines on the test set, followed by the LSTM method (where no image representation is used). GBR is the worst performer by a significant margin.

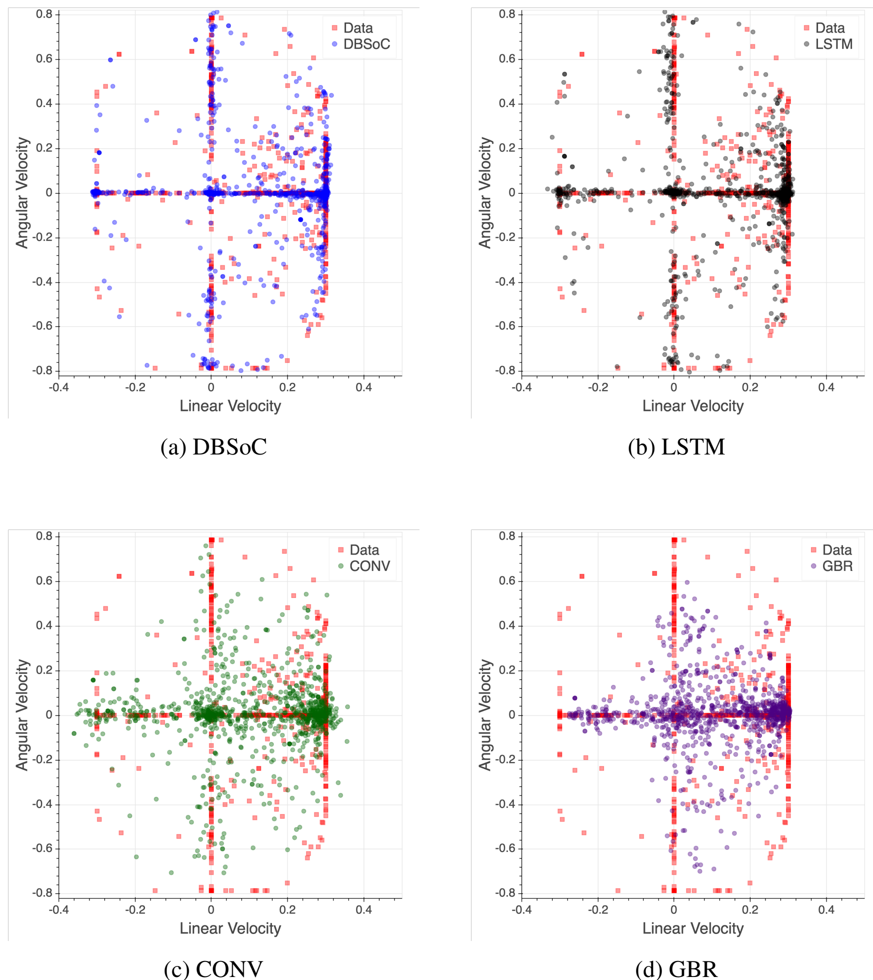
		GBR	LSTM	CONV	DBSoC
Group	Train	0.0232	0.0098	0.0087	**0.0018**
	Test	0.0264	0.0120	0.015	**0.0031**
Follow	Train	0.0265	0.00334	0.0079	**0.0030**
	Test	0.0323	0.00450	0.0250	**0.0036**

Table 5.1: Train and test MSE for DBSoC and the baselines.

5.4.2 Output Distribution

A more qualitative yet insightful means of evaluation can be achieved by examining the distribution of the outputs of each learned policy, conditioned on the inputs. That is, using a portion of the test data, we plot the predicted linear and angular velocities $(v_{pred}, \omega_{pred})$ and compare them with the actual values encountered in the data (v_{dem}, ω_{dem}). This gives an indication of how the output distribution of the predictor matches that of the human demonstrator. This can be more insightful than a single metric value because it sheds some light on the expected behavior of the robot. Figure 5.5 overlays (v_{dem}, ω_{dem}) (blue) with $(v_{pred}, \omega_{pred})$ for our method and the baselines for the follow task. From this figure it is clear that GBR captures the demonstrated distribution quite poorly. Its behavior is expected to not use large angular velocities (y-axis) and will thus not be very responsive. The CONV baseline also seems unable to capture the distribution, as we see many points that are scattered and far away from

any data point, so we can expect it to behave poorly too. The LSTM policy on the other hand seems to have similar capabilities to DBSoC, which clearly does a good job at matching the demonstrator's outputs.



(a) DBSoC (b) LSTM

(c) CONV (d) GBR

Figure 5.5: Output distributions of the follow data (red) and each of the four learned policies.

5.4.3 Human Evaluation

Since the goal of DBSoC is to learn social behavior, it is essential to evaluate it using human subjects. Furthermore, the policy can only be properly assessed by deploying it on an actual robot. A key reason for this is that behavioral cloning treats learning

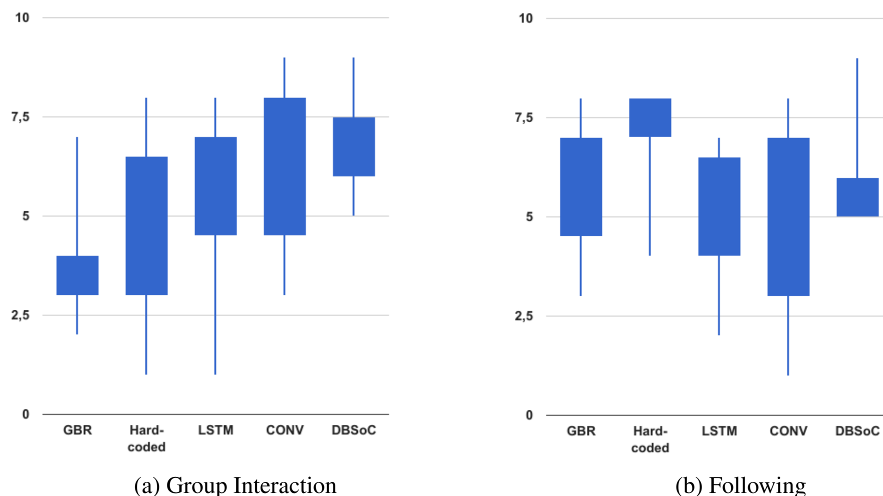
as a supervised problem, but the learned policy is ultimately deployed in a sequential decision making setting. It is thus crucial to actually deploy the policy and verify that the learned actions do not carry the robot to undesirable states. In addition, the robot is a complex dynamical system and we thus expect delays or even failure to achieve commanded reference velocities, depending on the low level control hardware. Small errors from the correct output are thus even more likely to result in an undesired next state, resulting in more covariate shift.

Our human evaluation consisted of 5 groups and a total of 15 subjects from the University of Amsterdam interacting with the learned policies as well as a hard-coded controller. This hard-coded behavior interprets the person closest to the robot as the sole interaction target and attempts to keep a constant distance (ρ) of $1.3m$ from them, in accordance with standard proxemics [39], and an orientation (ϕ) of $0°$. The subjects were first briefed about the purpose of each behavior they were about to evaluate (i.e., follow or group conversation). Following the briefing was a calibration step where the subjects were shown human-level performance in each of the tasks as well as various policies at random. This allowed the subjects to get a feel for what to expect from the robot. After calibration, the subjects were given approximately two minutes to interact with each policy for each of the tasks. The policies were given anonymously and in a different order each time. The subjects were then given the opportunity to rate the social performance of each policy with a number from 1 to 10, with 10 being human-level performance. Examples of the tasks and policies in such an evaluation session can be seen in the associated video.[2]

A summary of the scores for each policy is shown as box plots in Figure 5.6. The results show that different methods perform quite differently depending on the task. For example, the hard-coded controller is perceived to perform well for the follow behavior—due to the smoothness of its control—whereas in the group interaction task that controller performs poorly. A common comment from subjects was that the hard-coded controller only concentrated on one person and did not adjust to the group. A perhaps more surprising result in that humans perceive GBR to be a decent policy on average for the following tasks. These ratings are quite noisy, however, because the robot's behavior varied a lot depending on the situations that arose during interaction. DBSoC performs quite well during all evaluations. Remarkably, it never received a score lower than 5, whereas most of the other methods did. Furthermore, when performance is averaged across tasks, it performed the best of all methods, showing greater generality than the baselines. However, it does not perform quite as well as the hard-coded controller on the follow task. This was mainly because control was less smooth for DBSoC, as it was for all the learning baselines.

[2] http://bit.ly/2vqo9Vv

A more comprehensive evaluation using many more subjects and a more detailed questionnaire would be necessary to reach definitive conclusions. Nonetheless, these results provide some confirmation of the generality and effectiveness of DBSoC. Furthermore, the fact that the learned policies were positively evaluated by subjects who were allowed to interact freely with the robot (rather than having to follow a rigid script), shows that DBSoC has learned useful policies that generalize beyond the data on which they were trained.



(a) Group Interaction (b) Following

Figure 5.6: Box plots summarising human evaluation scores for all baselines on the two tasks.

5.5 Discussion

This chapter describes a full LfD pipeline that allows learning of two social tasks for a telepresence robot. The end result is a system that allows for continuous, and interactive data collection and learning of the required tasks. More importantly the finished pipeline requires minimal technical interaction in order to collect data learn a new behavior. This serves as a very good example of the benefits of LfD as a programming interface for non-experts. It further demonstrates how domain specific considerations, such as the choice of input representation can greatly improve the performance and enable generalization. Finally it demonstrates the great challenges involved in evaluating systems that operate in social environments. More notably, while our architecture did a great job in the evaluation metrics, when it came to real world evaluation and deployment the results exhibited a lot more variance, as different people had different ideas on how a robot

should behave in a given situation. For example a lot of people would rate idle policies quite well as they did not like the robot taking too much of an initiative on how to behave. Fortunately if behaviors are easy to program using LfD a user can define the behavior they want by simply collecting the appropriate data, of perhaps choose from a library of pre-learned policies.

The approach described here also suffers from some limitations. First of all the egocentric view of the robot prevents it from being aware of its own velocity. Feeding previous velocities as part of the input however resulted in unwanted feedback loops. Future work is required to establish an input representation that is appropriate of informing the robot of its own kinematic state. In addition, our input representation only contains the relative positions of people around the robot. This means that the robot is not aware of obstacles around it and relies on external hard coded systems for safety. The maximum range around the robot is also a design decision that might be inappropriate for some cases. A straightforward way is augmenting the symbolic input with another channel that includes the obstacles around the robot as detected by the lasers on the TERESA robot. This applies to any other important information about the surroundings of the robot that we might want to include. Again this representation would still be much more efficient than using pure visual input as it requires less data, more readily generalizes across different rooms, leverages the powerful person detection systems that exist, and fuses information from various sensory modalities.

In addition to alleviating the above concerns, future directions of the above research include the use of reinforcement and inverse reinforcement learning along with behavioral cloning in order to further improve the quality as well as the range of behaviors that can be learned.

5.6 Conclusions and future work

In this chapter, we introduced DBSoC, a generic neural network architecture that allows learning from demonstration for social robotic tasks. We considered two social tasks: group interaction and following. Our architecture was able to learn both tasks well without any programming, demonstrating its generality. Its performance was confirmed quantitatively and also from human evaluation sessions where the learned policies were deployed on a real telepresence robot.

Chapter 6

Temporal Alignment for Control

In the previous chapter we trained a deep neural network via behavioral cloning (BC) in order to perform two social interaction tasks. Our learning pipeline consisted of collecting separate datasets for the two tasks and then learning separate policies for them. During testing we also evaluated the two learned policies separately. While this is a feasible option for the tasks considered, it would be much more natural to provide demonstrations for a full interaction, i.e., a demonstration trajectory could contain both examples of group interaction and following. Simple LfD algorithms like BC however cannot distinguish between different policies present within a demonstration and, without any modification, would learn one joint policy for both tasks. The resulting policy is not only more complex and harder to learn but also less reusable. One would not be able to isolate the group interaction behavior from following. This would prevent us from using it in cases where the latter is not required, or combining it with other policies that the agent has learned. Ideally we would like a method that given demonstrations that contain multiple policies, is able to discover this decomposition and learn separate sub-policies. *Modular LfD* addresses this challenge by modeling a demonstrated task as a composition of sub-tasks for which reusable sub-policies (modules) are learned. These sub-policies are commonly easier to learn and can be composed in different ways to execute new tasks, enabling zero-shot imitation.

One approach to modular LfD is to provide the learner with additional information about the demonstrations. This can come in many forms, e.g., manual segmentation of demonstrations into sub-tasks [45], interactive feedback during learning [82], or prior knowledge about the task. Such domain knowledge may come in the form of motion primitives, hard-coded parametric motion models [24, 103] and problem-

specific state modeling [3, 57]. An additional benefit is that since these methods ground demonstrations on human-defined sub-tasks, the learned policies are interpretable. They are, however, labor intensive, perform sub-optimally if the underlying assumptions are incorrect, and require domain knowledge.

At the other extreme are unsupervised methods for modular LfD [31, 40, 59, 61], which model reusable sub-policies as latent variables [60], selected by a high level meta-controller. Such approaches can be applied to arbitrary domains and do not require additional supervision. However, the lack of constraints means that the sub-policies can be hard to learn and may not be interpretable. This lack of interpretability, together with the need for a task-specific meta-controller, limits the opportunity to compose the sub-policies in different ways to execute new tasks.

In this chapter, we consider a general, weakly supervised, modular LfD setting where demonstrations are augmented only with a *task sketch*. This sketch describes the sequence of *sub-tasks* that occur within the demonstration, without additional information on their alignment (see Figure 6.1).

Drawing inspiration from speech recognition [37] as well as modular reinforcement learning [5], we introduce *temporal alignment for control* (TACO), an efficient, domain agnostic algorithm that learns modular and grounded policies from high level task descriptions, while remaining weakly supervised.

Instead of considering the alignment of a demonstration with the task sketch and the learning of associated sub-policies as two separate processes, in TACO the imitation learning stage affects the alignment and vice-versa via a recursive forward-backward procedure that maximizes the joint likelihood of the observed sketch and the observed action sequence given the states. TACO learns one sub-policy for each sub-task present in the data and extends each sub-policy's action space to enable self-termination. At test time, the agent is presented with new, potentially unseen, and often longer sketches, which it executes by composing the required sub-policies. In addition to making complex tasks easier to learn, this approach thus enables zero-shot imitation based on sketch provision.

We evaluate the performance of TACO on four domains of varying complexity. First, we consider two toy domains, a 2D navigation task and the Craft domain proposed by Andreas et al. [5]. Furthermore, we consider a complex domain based on controlling a simulated robot arm to use a number pad and extend the task to use only image-based observations. We demonstrate that, in all domains, policies trained using TACO are capable of matching the performance of policies trained using a fully supervised method, where the segmentation of the demonstration is provided, at a small fraction of the labeling cost.

6.1 Related Work

Learning from demonstration encompasses a wide range of techniques that focus on learning to solve tasks based on expert demonstrations [6]. The fields of modular and hierarchical LfD aim to extract reusable policy primitives from complex demonstrations. This allows the reuse of such sub-policies between tasks, and enables their combination to generate more complex controllers.

In robotics, sub-policies are also modeled as *motion primitives* [103] which build the foundation for various works on modular LfD e.g. [61, 76, 81, 89]. Most notable is recent work based on *skill trees* [57] and semantically grounded finite representations [82]. These consider separate segmentation of the trajectories and fitting of primitives, decoupling the two parts of the optimization problem. Our work addresses segmenting the demonstrations and learning policies in one combined process. In addition all of these methods consider learning policies that are only appropriate in the context of robotics. In contrast, we consider learning general domain agnostic policies represented as neural networks.

Interleaving the two processes been shown to provide better segmentations and policies in recent work by Lioutikov et al. [72]. The approach considers learning via policy embeddings in task space using methods from probabilistic motion primitives [88]. This puts a restriction in the types of tasks and observations that the method can handle. The method developed in this chapter on the other hand, can be applied to a much wider range of observation spaces and tasks, as long as they can be modeled via a differentiable function approximator.

Recent work on hierarchical LfD transfers concepts from the *options* framework [113], which models low-level policies as actions of a meta controller, to LfD [31, 40, 59]. Generally, options serve as tools for dividing a complex task into multiple sub-policies specialized for regions in the state space. TACO differs from option discovery frameworks [31, 59] during both training and inference time by replacing the functionality of the meta-controller with weak supervision in the form of a sequence of symbols. This approach allows us to compute semantically grounded sub-policies. Weak supervision further constrains the learned policies to follow the description in the task sketch. This constraint prevents high-frequency switching between policies common with options [59] as well as the potential collapse of the meta-controller to apply only a single option. Furthermore, by applying task sketches at test time, we enable the composition of sub-policies in previously unseen and potentially longer sequences for zero-shot imitation.

Another branch of hierarchical LfD consists of *neural programmer interpreters* such as *neural task programming* (NTP) [124], which has shown success in robotics tasks.

In NTP, the learner receives a demonstration and the corresponding program calls. At test time, a new demonstration is given with the aim of executing the most appropriate piece of code to imitate it. At the lowest level, NTP thus relies on the existence of the programs to be called. Complimentary to this direction, our approach learns these programs from demonstration.

Another common approach to learning complex policies from demonstration provides the learned policy with additional context about the task to be executed. This context can come in the form of a whole demonstration [23, 30], images [21], or even natural language [17], [77]. Natural language approaches are the ones most related to our work as they attempt to ground demonstrations with instructions. Since these methods condition the policy on an embedding of the instructions given, they tend to be more flexible however they do not recover interpretable modular policies as in our case.

Recent work in *modular reinforcement learning* (RL) [5], introduces the notion of sketches as additional information representing the decomposition of tasks. Similar to our work, Andreas et al. [5] assume that complex tasks can be broken down into subtasks. Our approach exploits a similar modular structure but uses an LfD formulation that addresses the problem of aligning sequences of different lengths.

A common approach to sequence alignment in speech recognition is *connectionist temporal classification* (CTC) [37] which is also used in action recognition [46], handwriting recognition [73] and lip-reading [7]. Previous extensions of CTC have been proposed to increase its flexibility by reducing the assumptions underlying the framework [36], exploiting structure in the input space [46], and addressing structured outputs [56]. In this paper, we extend CTC to combine maximum likelihood objectives for sequence alignment and behavioral cloning.

6.2 Preliminaries

In this section, we introduce the required concepts and methods for the derivation of TACO.

6.2.1 Modular Learning from Demonstration

The standard formulation of BC as introduced in Chapter 2 learns a single policy from the demonstrated data. This has two important limitations. The first is modularity: the demonstrated behaviour can have a hierarchical structure that decomposes into modules, or sub-policies. The second is reuse: the modules can be composed in various ways to perform different tasks. Modular LfD is a simple way of introducing modularity and reuse into LfD. A schematic is shown in Figure 6.1. To make policies more reusable, it

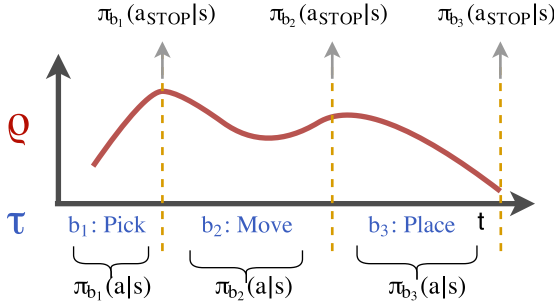


Figure 6.1: Problem setting: The trajectory ρ (red) is augmented by a task sketch τ (blue). The two sequences operate at different timescales. The whole trajectory is aligned (manually or automatically) and segmented into three parts. From this alignment three separate policies are learned. The unobserved a_{STOP} action for each policy is inferred to occur at the point where the policies switch from one to the other.

assumes that a task can be solved by multiple sub-policies, each of which operates in an augmented action space $\mathcal{A}^+$ that includes a STOP action (i.e., $\mathcal{A}^+ := \mathcal{A} \cup a_{STOP}$) that is not observed in the demonstration. It also assumes that more than one sub-policy may be present within a demonstration. In our setting, extra information is provided in the form of a *task sketch* $\tau = (b_1, b_2, \ldots, b_L)$, with $L \leq T$, and $b_l \in \mathcal{B}$, where $\mathcal{B} = \{1, 2, \ldots, K\}$, is a dictionary of sub-tasks. The sketch indicates which sub-tasks are active during a trajectory.

Although the a_{STOP} action is never observed, it can be inferred from the data. If the demonstration contains a simple task then a_{STOP} is called only at the end of the demonstration. If $L = T$, then we know which policy from $\mathcal{B}$ is active at each time-step. i.e., a_{STOP} for each policy is called as soon as the active policy changes within the demonstration. If all extra information is available, we can perform behavioural cloning, with two differences. First, maximising the likelihood takes place assuming an action-augmented policy $\pi(a^+|s)$. Second, we learn $|\mathcal{B}| = K$ modular policies π_{θ_k} from K datasets $\mathcal{D}_k$ containing trajectories ρ_k as segmented based on τ:

$$\theta^*_{k=1,\ldots,K} = \underset{\theta_k}{\operatorname{argmax}} \, \mathbb{E}_{\rho_k}[\sum_{t=1}^{T_\rho} \log \pi_{\theta_k}(a^+_t|s_t)]. \tag{6.1}$$

However, the fully supervised approach is labour intensive as each trajectory must be manually segmented into sub-policies. In this paper, we consider cases where $L << T$ and τ contains only the sequence of active sub-tasks in the order they occur, without duplicates. Inferring when a_{STOP} occurs is therefore more challenging as τ and ρ operate at different timescales and must first be aligned.

6.2.2 Sequence Alignment

Since $L << T$, we cannot independently maximise the likelihood of active sub-tasks in τ for every time step in ρ. The problem of aligning sequences of different lengths, a common challenge in speech and action recognition, can be addressed via *connectionist temporal classification* (CTC) [37]. Here, we review a variant of CTC adapted to the notation introduced so far, which does not include gaps between the predictions, based on the assumption that sub-tasks occur in sequence without pause.

A *path* $\zeta = (\zeta_1, \zeta_2, ..., \zeta_T)$ is a sequence of sub-tasks of the same length as the input sequence ρ, describing the active sub-task ζ_t from the dictionary $\mathcal{B}$ at every time-step. The set of all possible paths $\mathcal{Z}_{T,\tau}$ for a task sketch τ is the set of paths of length T that are equal to τ after removing all adjacent duplicates. For example, after removing adjacent duplicates, the path $\zeta = (b_1, b_1, b_2, b_3, b_3, b_3)$ equals the sketch $\tau = (b_1, b_2, b_3)$. The CTC objective maximises the probability of the sequence τ given the input sequence ρ:

$$\psi^* = \underset{\psi}{\text{argmax}}\, \mathbb{E}_{(\rho,\tau)}[p_\psi(\tau|\rho)] \tag{6.2}$$

$$= \underset{\psi}{\text{argmax}}\, \mathbb{E}_{(\rho,\tau)}\Big[\sum_{\zeta \in \mathcal{Z}_{T,\tau}} p_\psi(\zeta|\rho)\Big] \tag{6.3}$$

$$= \underset{\psi}{\text{argmax}}\, \mathbb{E}_{(\rho,\tau)}\Big[\sum_{\zeta \in \mathcal{Z}_{T,\tau}} \prod_{t=1}^{T} p_\psi(\zeta_t|\rho)\Big] \tag{6.4}$$

where $p_\psi(\zeta_t|\rho)$ is commonly represented by a neural network parameterised by ψ that outputs the probability of each sub-task in $\mathcal{B}$. While naively computing (6.2) is infeasible for longer sequences, dynamic programming provides a tractable solution. Let $\mathcal{Z}_{t,\tau_{1:l}}$ be the set that includes paths $\zeta_{1:t}$ of length t corresponding to tasks sketches $\tau_{1:l}$ of length l, and $\alpha_t(l) = \sum_{\zeta_{1:t} \in \mathcal{Z}_{t,\tau_{1:l}}} p(\zeta|\rho)$ be the probability of being in task b_l at time-step t in the graph in Figure 6.2a. The probability of a task sketch given the input sequence $p(\tau|\rho)$ is equal to $\alpha_T(L)$.

We can recursively compute $\alpha_t(l)$ based on $\alpha_{t-1}(l)$, $\alpha_{t-1}(l-1)$, and the probability of the current sub-task $p(\zeta_t|\rho_t)$. As τ begins with a specific sub-task, the initial α's are deterministic and the probability of starting in the corresponding policy b_1 at time-step $t = 1$ is 1. Figure 6.2a depicts the recursive computation of the forward terms which is

mathematically summarised as:

$$\alpha_t(l) = p(b_l|\rho_t)[\alpha_{t-1}(l-1) + \alpha_{t-1}(l))], \tag{6.5}$$

$$\alpha_1(l) = \begin{cases} 1, & \text{if } l = 1, \\ 0, & \text{otherwise.} \end{cases} \tag{6.6}$$

Based on the recursive computation of the CTC objective in (6.5) and any automatic differentiation framework, we can optimise our model. For the manual derivation of the gradients and CTC backwards variables, please see the work of Graves et al. [37].

6.3 Methods

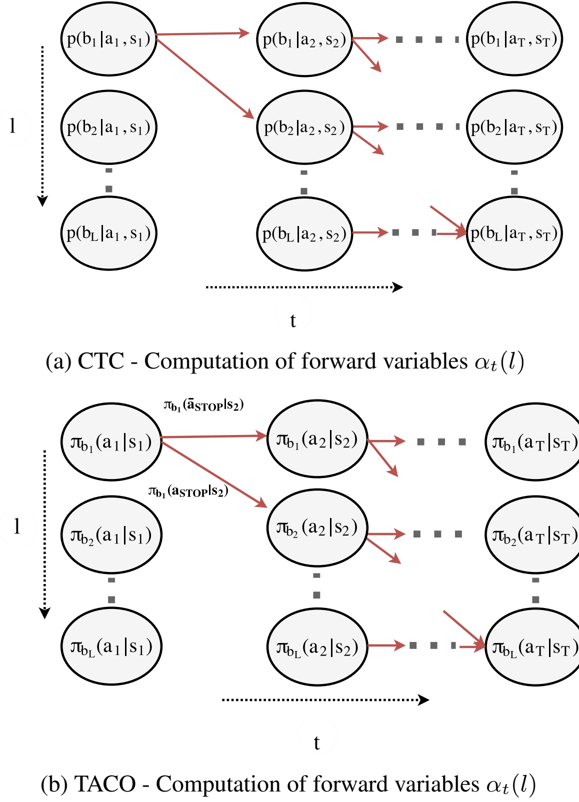
This section describes two ways to apply insights from CTC to address modular LfD. We first describe a naive adaptation which performs modular LfD with arbitrary differentiable architectures and discuss its drawbacks. We then introduce TACO, which simultaneously aligns the trajectory ρ and task sequence τ and learns the underlying policies.

6.3.1 Naively Adapted CTC

A naive modular LfD algorithm can first align the state-action trajectories with the prescribed sketches via CTC to derive the required datasets $\mathcal{D}_k$ and then learn K modular policies with behavioral cloning as in (6.1), an approach we denote as CTC-BC.

However, this approach fundamentally differs from the regular application of CTC. At inference time, CTC is typically given a new trajectory ρ and computes the most likely sketch assignment τ. In contrast, we use CTC to align ρ with τ and subsequently train the sub-policies via BC while discarding the CTC based controller. This alignment for BC is derived via maximization over $\alpha_t(l)$, as computed in (6.5), at each time-step, leading to the most likely path through the sequence. Following the determination of sketch-trajectory alignment, the sub-policies are optimized following (6.1).

While this approach only trains sub-policies via a single alignment based on the $\arg\max$ of the forward variables α for every time-step in (6.5), we can account for the probabilistic assignment of active sub-policies by utilizing a weighting based on the forward variables. However, as no aligned targets exist for the stop actions, they have to be derived based on the relations of the normalized α's between consecutive time-steps. A derivation of this α-weighted version of CTC-BC can be found in Appendix A. In our experiments, however, we consider the direct, single alignment resulting from taking



(a) CTC - Computation of forward variables $\alpha_t(l)$

(b) TACO - Computation of forward variables $\alpha_t(l)$

Figure 6.2: Visualization of the forward recursion for all paths ζ corresponding to the sketch τ. The horizontal axis denotes time, while the vertical denotes the task sequence. While a node at l, t in CTC is only weighted by the meta-controller probability $p(b_l|s_t, a_t)$, nodes and edges are weighted in TACO respectively via the sub-policies $\pi_l(a_t|s_t)$ and $\pi_l(a_{stop}|s_t)$

the argmax as well as the full optimization of the joint probabilities via TACO.

A crucial drawback of CTC-BC is independent computation of the optimization for alignment and for imitation. The alignment affects the policy optimization as it is performed in a later step but not vice-versa. The next section addresses this shortcoming.

6.3.2 Temporal Alignment for Control (TACO)

Aligning the sequences ρ and τ via CTC treats the index for active sub-policies as pure symbols and fails to exploit the fact that subsequently we need to optimize policies to map states to actions via BC. Consequently, what CTC determines to be a good alignment might result in a badly conditioned optimization problem for the sub-policies,

converging to local minima and thus demonstrate degraded performance once deployed.

In this section, we propose *Temporal Alignment for Control* (TACO), in which the alignment is influenced by the performance of the sub-policies and addressed within a single optimization procedure. Concretely, instead of maximizing Equation (6.2) followed by Equation (6.1), we seek to maximize the joint log likelihood of the task sequence and the actions conditioned on the states:

$$p(\tau, \mathbf{a}_\rho | \mathbf{s}_\rho) = \sum_{\zeta \in \mathcal{Z}_{T,\tau}} p(\zeta | \mathbf{s}_\rho) \prod_{t=1}^{T} \pi_{\theta_{\zeta_t}}(a_t | s_t). \tag{6.7}$$

Where $p(\zeta|\mathbf{s}_\rho)$ is the product of the stop, a_{stop}, and non-stop, $\bar{a}_{stop}$, probabilities associated with any given path. Note that the first term in the equation above is similar to the term in (6.4) but now depends only on states. Every possible alignment ζ dictates which data within the sequence ρ is associated with which sub-policy π_θ, which is the second term in (6.7) and corresponds to the BC objective. Maximizing thus performs simultaneous alignment of τ and ρ and learns the associated policies for each sub-task.

As with CTC, the sketch length is expected to be shorter than the trajectory length, $L \ll T$, which for longer trajectories renders the computation of all paths ζ in $\mathcal{Z}_{T,\tau}$ intractable. However, the summation over paths can be performed via dynamic programming with a forward-backward procedure similar to that of CTC. Using consistent notation, the likelihood of a being at sub-task l at time t can be formulated in terms of forward variables:

$$\alpha_t(l) := \sum_{\zeta_{1:t} \in \mathcal{Z}_{t,\tau_{1:l}}} p(\zeta | \mathbf{s}_\rho) \prod_{t'=1}^{t} \pi_{\theta_{\zeta_{t'}}}(a_{t'} | s_{t'}). \tag{6.8}$$

Here $\tau_{1:l}$ denotes the part of the sub-task sequence until l.

Even though not explicitly modeling the probability of a certain sub-task given the states and actions ($p(b|s,a)$), extending the policies with the stop action a_{STOP} enables switching from one sub-task in the sketch to the next. This in turn allows to compute the probability of being in a certain sub-task b_l of the sketch, at time t. At time $t = 1$ we know that $\zeta_1 = \tau_1$, i.e., we always necessarily begin with the first sub-policy described in the sketch.

$$\alpha_1(l) = \begin{cases} \pi_{\theta b_1}(a_1 | s_1), & \text{if } l = 1, \\ 0, & \text{otherwise.} \end{cases} \tag{6.9}$$

Subsequently, a certain sub-policy can only be reached via staying in the same sub-policy or stopping from the previous policy in the sketch using the a_{STOP} action,

i.e.,

$$\alpha_t(l) = \pi_{\theta b_l}(a_t|s_t)\big[\alpha_{t-1}(l-1)\pi_{\theta b_{l-1}}(a_{STOP}|s_t) \\ +\alpha_{t-1}(l)(1-\pi_{\theta b_l}(a_{STOP}|s_t))\big]. \tag{6.10}$$

Performing this recursion until T yields the forward variables. A visualization of the recursion is shown in Figure 6.2b. The forward variables at the end of this recursion determine the likelihood in (6.7):

$$\alpha_T(L) = p(\tau, \mathbf{a}_\rho|\mathbf{s}_\rho). \tag{6.11}$$

Since the computation is fully differentiable, the backward variables and subsequently the gradient of the likelihood with respect to the parameters θ_i for each policy can be computed efficiently by any auto-diff framework. However, as the forward recursion can lead to underflow, we employ the forward variable normalization technique from Graves et al. [37].

Intuitively, the probability for the stop actions of a policy at each time-step determines the weighting of data-points (state-action pairs) for the all sub-policies. If a sub-policy assigns low probability a specific data-point, e.g., if at similar states it has been optimized to fit different actions, the optimization will increase the probability of the preceding and succeeding policy in the sketch for that data-point, effectively influencing the probabilistic alignment.

6.4 Experiments

We evaluate TACO across four different domains with different continuous and discrete states and actions including image-based control of a 3D robot arm. Our experiments aim to answer the following questions.

- How does TACO perform across a range of different tasks? Can it be successfully applied to a range of architectures and input-output representations?
- How does TACO perform with respect to zero-shot imitation on sequences not included in the training set and task sketches of different length?
- How does TACO perform in relation to baselines including CTC-BC (Section 6.3.1) and the fully supervised approach with all trajectories segmented into sub-tasks?
- How does the dataset size influence the relative performance of TACO in comparison with our baselines?

The latter questions are investigated by introducing a set of baselines based on Sections 6.2.1 and 6.3.1.

- GT-BC, which uses ground-truth, segmented demonstrations and performs direct maximum likelihood training to optimize the sub-policies (Equation 6.1).
- CTC-BC, which uses both bidirectional *gated recurrent units* [18] [CTC-BC(GRU)] or *multi-layer perceptrons* [CTC-BC(MLP)] for the CTC controller. In both cases, we apply MLPs for the sub-policies.

For each domain we consider three evaluation metrics in order to compare the different methods. The main evaluation metric is the *task accuracy*, i.e. the ratio of full tasks completed over the total attempted. A task is considered succesful if all the prescribed sub-tasks are completed in the given sequence. Each domain has different success criteria for its subtasks which are described in their respective sections. To provide more detailed insight *sub-task accuracy* is also reported. This is the ratio of successful subtasks over the total attempted. Finally we report *sequence alignment accuracy* for each method, to evaluate the influence of integrating the sub-policy and alignment optimization. This measured as the % of agreement between the ground truth sub-task sequence of length T and the most likely sequence predicted by the methods. We compute the alignment accuracy for 100 hold-out test sequences, which come from the same distribution as the training data. The results across all the experiment domains are stated in Table 6.2. For BC, we pass the learned policy through TACO alignment without learning. That is, we perform the forward variable computation on the already learned policy $\pi(a^+|s)$ and take the argmax over the forward variables to discover the most likely active subtask at every timestep.

For task accuracy we primarily focus on testing in a zero-shot setting: the task sequences prescribed at test time are not in the training data and in are longer than the training sketches. Note that in the non-zero-shot settings, while the tasks to be completed have been seen, the world parameters such as feature positions are randomized. Finally, in all our evaluations we vary the size of the training set to investigate how performance varies with respect to available data.

The policies for all tasks are modeled as MLPs unless images are used. In this case, convolutional layers, shared between all sub-policies, are used to extract state features. The same applies for the architectures utilized to align the sequences in the case of CTC based methods. In continuous domains, action probabilities are modeled as $a \sim \mathcal{N}(\mu, \sigma = 1)$ where μ is the output of the MLP. In discrete domains, action probabilities are modeled using categorical distributions. We found that having separate networks for the domain actions and the a_{STOP} action, leads to better performance, especially for the Dial domain. See Table 6.1 for implementation details of the architectures used

in each experiment where *Core* represents the optional architecture that learns a feature representation of the input space before feeding it into the stop and action policies.

As mentioned in Chapter 2, one drawback of BC is its susceptibility to *covariate shift*, which occurs when small errors during testing cause the agent to drift away from states it encountered during training, yielding poor performance. In the following experiments we use DART [64], which introduces noise in the data collection process, allowing the agent to learn actions that can recover from errors.

Experiment	State Dim	Core	Stop Policy	Action Policy	Output Dim
Nav-World	8	-	FC [100]	FC [100]	2
Craft	1076	-	FC [400,100]	FC [400,100]	7
Dial	39	-	FC [300,200,100]	FC [300,200,100]	9
Dial (Images)	[112,112,3]	conv[10,5,3] kernel[5,5,3] stride[2,2,1]	FC [400,300]	FC [400,300]	9

Table 6.1: Specification of the architectures used in each experiment. For experiment *Dial (Images)* conv[], kernel[] and stride[] represent the number of channels, dimension of square kernels and 2D strides per convolutional layer respectively

6.4.1 Nav-World Domain

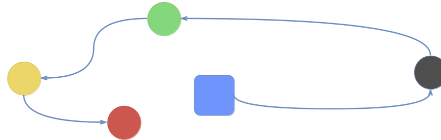
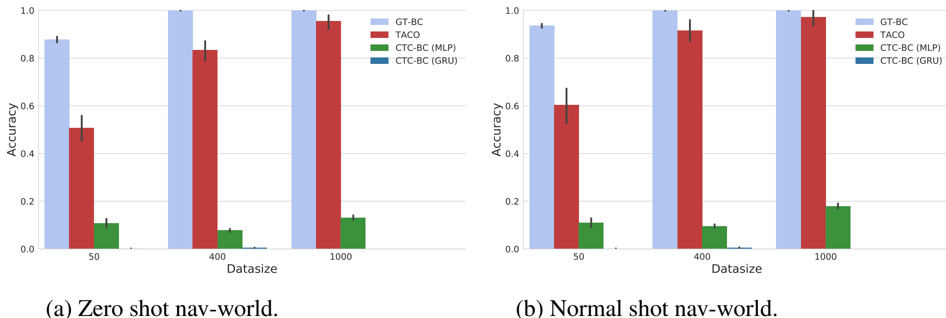


Figure 6.3: The Nav-World. The agent (Blue) receives a route as a task sketch. In this case. $\tau =$ (Black, Green, Yellow, Red)

We present the Nav-World domain, depicted in Figure 6.3, as a simple 2D navigation task. The agent (Blue) operates in a 8-dimensional continuous state space with four destination points (Green, Red, Yellow, Black). The state space represents the (x, y) distance from each of the destination points. The action space is 2-dimensional and represents a velocity (v_x, v_y). At training time, the agent is presented with state-action trajectories ρ from a controller that visits L destinations in a certain sequence given by the task sketch, e.g. $\tau =$ (Black, Green, Yellow, Red). At the end of learning, the agent outputs four sub-policies $\pi(a^+|s)$ for reaching each destination. At test time, it is given a sketch τ_{test} of length L_{test} that details the sequence of destinations. The task is considered successful, if the agent visits all destinations in the correct order. During demonstrations and testing, the agent's location and the destination points are sampled from a Gaussian distribution centred at predefined locations for each point.



(a) Zero shot nav-world.

(b) Normal shot nav-world.

Figure 6.4: Nav-World results: Mean accuracy over 100 agents on 100 test tasks. Task length at test time is $L_{test} = 4$ for the zero shot 6.4a and $L = 3$ for the normal shot 6.4b. TACO (red) approaches the performance of a fully supervised sequence (grey) given enough data.

In this domain, the dataset sizes are 50, 400, and 1000 demonstrations. The task length at training time is $L = 3$. We report the task success rate for unseen, longer tasks of length $L_{test} = 4$ (zero-shot setting). 100 agents are trained for each task and each algorithm, with the evaluation based on 100 testing tasks.

As displayed in the results in Figure 6.4, CTC-BC performs poorly in both settings, with the MLP architecture performing slightly better. However, CTC(MLP) provides accurate alignment often reaching 90% overlap between the predicted sub-task sequence and ground truth (Table 6.2). Fitting on a smaller dataset however strongly reduces the quality of the policies. As the states are similar before and after a policy switches but the actions are different (different policies), small mistakes in alignment cause multi-modality in the data distribution. This causes relatively low performance for BC with a mean squared error (MSE) objective. In contrast, TACO avoids this problem by probabilistically weighting all policies when fitting to each time-step. Furthermore, TACO achieves much better alignment accuracy, because of its inherent inductive bias for control. These two factors bring about performance that approaches that of GT-BC for larger datasets, a consistent trend in all our experiments.

6.4.2 Craft Domain

Andreas et al. [5] introduced the Craft domain to demonstrate the value of weak supervision using policy sketches in the RL setting. In this domain, an agent is given hierarchical tasks and a sketch description of that task. The binary state space has 1076 dimensions and the action space enables discrete motions as well as actions to pick up and use objects. A typical example task is $\tau_{\text{make planks}} = (\text{get(wood)}, \text{use(workbench)})$. The tasks vary from $L = 2$ to $L = 4$. The demonstrations are provided from a trained

RL agent, which obtains near optimal performance. Performance is measured using the reward function defined in the original domain. We train agents for all baselines and deploy them on randomly sampled tasks from the same distribution seen during training.

Figure 6.5 shows the results, which are similar to the outcome in Nav-World. CTC-BC fails to obtain substantial reward. However, unlike in Nav-World, CTC does not achieve good alignments with either architecture, as the abstract, binary state-action space makes it harder to detect distribution changes. Instead of concatenating state-action pairs, TACO learns a mapping from one to the other, making it easier to tell when the sub-policy changes. This in turn allows TACO to match or even surpass BC with sufficient data.

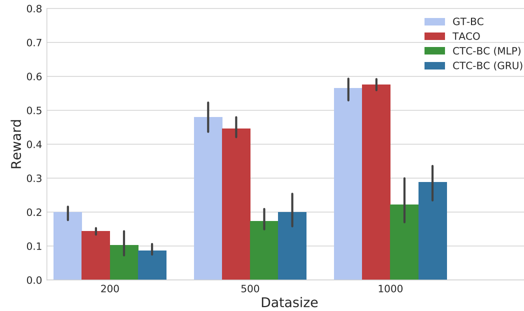


Figure 6.5: Craft results: mean reward over 2000 tasks. The tasks are the same as during training but the environment instantiation is different. The RL agent used to derive the policies achieves a mean reward of 0.9. TACO (red) approaches the performance of a fully supervised method (grey) given enough data.

6.4.3 Dial Domain

Even though the Craft domain is quite challenging, the lengths of the demonstrations and the resulting episodes are quite short, $T \leq 20$, yielding a potentially easier alignment task as T is close to L. The final two experiments take place in a more realistic robotic manipulation domain. In the Dial Domain, a JACO 9 DoF manipulator simulated in MuJoCo [117] interacts with a large dial-pad, as shown in Figure 6.6. The demonstrations contain state-action trajectories that describe the process by which a PIN is pressed, e.g. $\tau = (0, 5, 1, 6)$. A task is considered successful if all the digits in the PIN are pressed in order. Demonstrations in this domain come from a PID controller that can move to predefined joint angles. We sub-sample the data from the simulator by 20, resulting in trajectories $T \approx 200$ for $L = 4$. We consider two variants of the Dial Domain, one with joint-angle based states and the other with images. The action space represents the torques for each joint of the JACO arm in both cases.

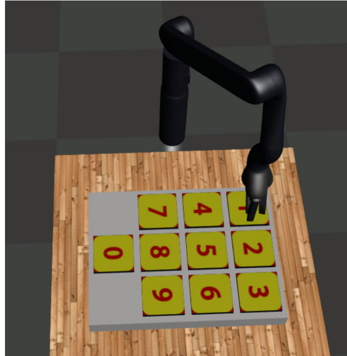


Figure 6.6: Dial Domain: A 9 DoF JACO arm must dial a PIN of arbitrary length.

Joint Space Dial Domain

In the first variant, the state state is manually constructed and 39-dimensional, containing the joint angles and the distance from each digit on the dial pad in three dimensions. If the locations of the numbers in the dial-pad remains the same, however, the problem can be solved only using joint angles. For this reason, during each demonstration we randomly swap the location of the numbers. We test the policies on the standard number formation displayed in Figure 6.6, which is never observed during training.

Figure 6.7 shows the results, which follow the same trend as in the other domains. CTC-BC fails to complete any tasks, although CTC-BC(MLP) aligns the sequences with 90% accuracy (Table 6.2). TACO achieves superior performance to GT-BC for all data sizes, which is due to regularising effects of TACO's optimisation procedure as discussed in Section 6.5. The resulting alignment is lower than that of TACO, which suggests that GT-BC is more prone to overfitting. Furthermore this difference in performace may result from the fact that the labels associated with stopping an active policy are much more sparse than the ones commanding the policy to remain active. Specifically for a sequence of T=200 and L=4 there are 4 instances of stopping and 196 datapoints of not stopping. This creates a large label inbalance in the dataset. Because TACO views the whole procedure as a unified optimisation problem it does not suffer from this drawback.

To evaluate our methods in more complex zero-shot scenarios, we also measure the task accuracy over 100 tasks as L_{test} is increased, as shown in Figure 6.8. As expected, TACO's performance falls with increasing task length as the chances of failing at a single sub-task increases. The accuracy however decreases at significantly lower rate than the baseline's performance.

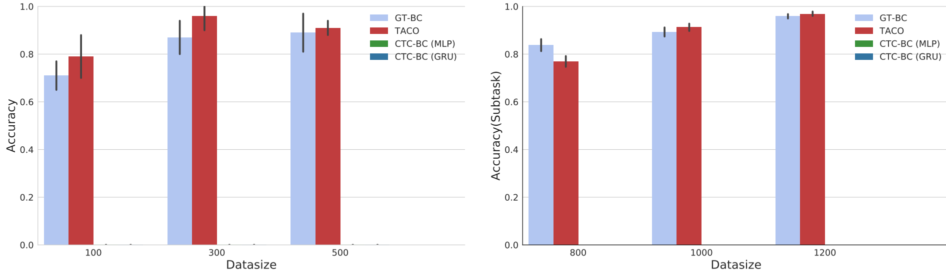


Figure 6.7: Joint Space Dial results: Task accuracy over 100 tasks of $L_{test} = 5$. During training $L = 4$. Evaluation is performed in an unseen configuration of the dial pad. Bars for CTC based methods do not appear as they did not finish any tasks.

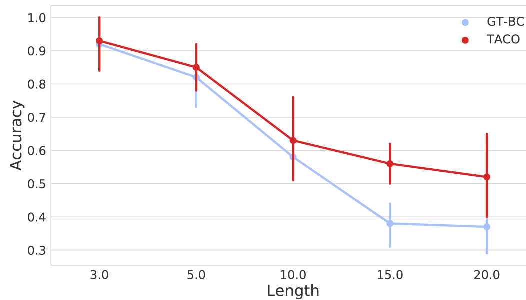
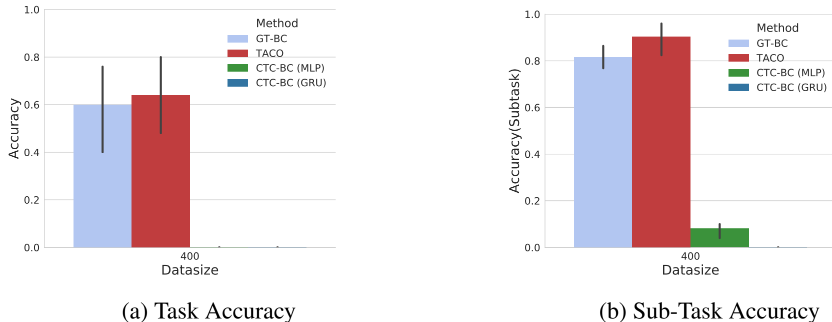


Figure 6.8: Joint Space Dial: Task accuracy for increasing values of L_{test}. The accuracy is measured over 100 runs at each configuration for the TACO and GT-BC.

Image Space Dial Domain

The image-based variant of the Dial Domain considers the same task as above, but with an image-based state representation. We use RGB images of size 112×112, which are passed through a simple convolutional architecture before splitting into individual policies (Table 6.1). In this case, we do not randomise the digit positions but discard joint angles from the agent's state space, as those angles would allow an agent to derive an optimal policy without utilising the image.

Figure 6.9 details task and sub-task accuracy for this domain. We can see that TACO performs well in comparison with the baselines, despite the increased difficulty in the state representation. Apart from shared convolutional network used as a feature extractor, no algorithmic modifications were required to achieve this performance.



(a) Task Accuracy

(b) Sub-Task Accuracy

Figure 6.9: Image-Space Dial results: Accuracy over 100 test tasks similar to the joint-space experiments, CTC-BC is unable to perform full task sequences and only solves a minor percentage of the sub-tasks, while both GT-BC and TACO are able to complete most sub-tasks. TACO completes the most full tasks.

	Domain			
Algorithm	Nav-World	Craft	Dial	Dial (Image)
TACO	**95.3**	95.6	**98.9**	**99.0**
CTC-BC (MLP)	89.0	41.4	31.4	84.6
CTC-BC (GRU)	80.0	57.1	28.6	48.8
GT-BC (aligned with TACO)	94.6	**99.4**	98.7	98.2

Table 6.2: Alignment accuracy of each algorithm for all experiments. TACO always outperforms CTC emphasising the importance of maximising the joint likelihood of sequences and actions for alignment accuracy.

6.5 Discussion

TACO requires only weak supervision for modular LfD while providing performance commensurate with fully supervised approaches on a range of tasks including zero-shot imitation scenarios. On more challenging domains, TACO demonstrates performance superior to that of the fully supervised approach.

We argue that TACOs strength lies in optimizing sub-policies over a distribution instead of a point estimate of the alignment. As each sub-policy is exposed to more data, this has a regularization effect since policies are optimized to additionally benefit the alignment. To test this hypothesis we evaluate trained GT-BC policies for alignment on unseen sequences by applying Equations (6.9) and (6.10) with GT-BC trained sub-policies. We find that GT-BC results in less accurate alignment than policies trained using TACO. The integration of optimization for sequence alignment as well as imitation learning into a single objective additionally achieves superior performance over CTC in aligning sequences. This suggests that the policy component step of TACO positively influences the alignment.

Due to the interdependence between both optimization processes, TACO relies on sufficient exploration of the different alignment paths (in Figure 6.2b). Failing to do so will cause early collapse of the probability distribution to a single path. We found this to be especially the case in the longer sequences encountered in the Dial domain. To mitigate this problem we use dropout [110] on the a_{STOP} actions. Dropout allows TACO to randomly sample different alignment paths during learning which in turn expose the policies to different actions in the data. We found this to be essential for the success of TACO on longer tasks. Dropout is applied with a gradually decaying drop probability, usually starting from $p_{drop} = 0.3$ and ending at $p_{drop} = 0.$ at the end of training.

TACO has been effectively applied to the presented tasks and significantly reduces supervision efforts. However, the given sketches are highly structured and dissimilar to natural human communication. An interesting avenue for future work is the combination of the increased modularity of TACO with more flexible architectures that can handle natural language [77], [17]. One way to do this would be to first ground natural language to general symbolic representations and then use TACO to learn the required policies. This would combine the flexibility of language with the interpretability and robustness of TACO, as demonstrated in this chapter.

Finally, future work aims at applications in more complex hierarchical tasks and on real robots. This is especially interesting for repetitive tasks such as pick and place, or even robotic surgical procedures [25]. In these scenarios we have exactly the problem of knowing what tasks are constrained within the demonstrations but manually labeling where each one occurs is tedious especially when the number of demonstrations is large.

6.6 Conclusion

This chapter introduces TACO, a novel method to address modular learning from demonstration by incorporating weakly supervised information in the form of a task sketch, that provides a high-level description of sub-tasks in a demonstration trajectory. We evaluated TACO in four different domains consisting of continuous and discrete action and state spaces, including a domain with purely visual observations. With limited supervision, TACO performs commensurate to a fully supervised approach while significantly outperforming the straightforward adaptation of CTC for modular LfD in both control and alignment.

Chapter 7

Conclusions

This thesis introduced a wide range of LfD algorithms spanning from theoretical and generic (Chapters 3 and 6) to more those applicable specifically to robotics (Chapters 4 and5). Each chapter tackles a different problem along the LfD pipeline in order to improve the robustness, data efficiency and computational complexity of either BC or IRL.

Chapter 1 introduced inverse reinforcement learning from failure (IRLF). An algorithm that can leverage failed executions of a task in order to learn what *not* do. We have shown that IRLF is capable of learning reward functions that are closer to ground truth with less data, than if only successful executions of a task are provided. We further showed that IRLF is stable to overlaps between the successful demonstrations and failed demonstrations. In the past two years there have been major advancements in model-free IRL with successes in simulated high dimensional continuous control. It would be very interesting to see how a model free variant of IRLF will fair in such domains.

Chapter 2 introduced rapidly exploring learning trees, an algorithm that learns sample based planner cost functions from demonstration. We have shown that RLT* is capable or learning cost functions close to the ground truth with very little data, surpassing learning algorithms for graph based planners such as MMP. We have also shown that RLT* can learn under motion constraints, making it valuable for robotic applications such as manipulation and driving, where sample based planners are widely employed. The algorithm has not been tested in such domains however, this is an interesting avenue for future work as the IRL methods currently used in manipulation are locally optimal and do not readily take into account obstacle avoidance.

In Chapter 3 we developed DBSoC a deep neural network architecture that is capable of learning social interaction policies from demonstration. DBSoC owes its success to

a smart input representation and the use of LSTMs. These two components allow the robot to learn robust policies and to take into account an arbitrary amount of people around the robot. Furthermore, the same architecture was used to learn two different tasks, with the only thing varying being the data. This fulfills one of the main promises of LfD, being able to program robots with little knowledge of the underlying technical details. The learned policies were deployed on a real telepresence robot. The DBSoC representation and architecture is very versatile and besides the possibility of exploring more social tasks one could apply it to other tasks involving human interactions such as pedestrian movement prediction.

Finally we introduced TACO, an algorithm that decomposes a demonstrated trajectory into sub-tasks according to a task sketch and learns the underlying sub-policies. The strength of TACO lies in the fact that the alignment step, between trajectory and sketch, is performed simultaneously to policy learning. This allows the two procedures to influence each other providing improved performance in both. TACO has been shown to perform as well as a fully supervised approach, where the alignment of trajectory and sketch is provided, in a number of domains with different state and action spaces, including image inputs. Furthermore TACO greatly outperforms a procedure that performs the alignment and policy learning separately. In industry there are a large amount of complex, hierarchical yet repetitive procedures. Such domains and the data they produce are ideal for methods like TACO. This is because while we know what each demonstration contains in terms of sub-policies, we do not know when each one occurs, and it would be very tedious to manually label the segmentation. Applying TACO in a real world task is thus an exciting and feasible avenue for future work.

In general, LfD is an ever growing field of artificial intelligence which will, in our opinion, play an important role in the deployment of autonomous agents in real life situations. This is because demonstrations are a rich data source for a hard problem, sequential decision making, and because alternative methods for solving it such as RL are computationally expensive and potentially unsafe to perform. Indeed RL practitioners are already leveraging LfD techniques to reduce the massive search space of certain domains. In terms of industrial and commercial applications we expect LfD to become more prominent in domains where demonstrations are readily available and do not need to be collected. This includes video games, medium scale industrial processes, self driving vehicles and pedestrian movement prediction and simulation.

Appendix A

CTC Probabilistic Sub-Policy Training

While the naive extension of CTC (Chapter 4.1) trains sub-policies based on the a single alignment by taking the argmax of the forward variables $\alpha_t(l)$ for every time-step, this paragraph addresses the possible probabilistic assignment of active sub-policies.

To obtain the probabilistic weighting based optimisation objective in Equation (A.1), we first define the probability distribution $p_t(l)$ at time-step t over all sub-policies l in a sketch based on the normalised CTC forward variables in Equation (A.2).

However, as the ground-truth targets in the trajectory ρ only exist for the regular actions, we first compute the stop action probability targets based on the CTC forward variables.

$$\theta^*_{k=1,..,K} = \underset{\theta_k}{\mathrm{argmax}}\, \mathbb{E}_{\rho_k}[\sum_{t=1}^{T}\sum_{l=1}^{L} \log p_t(l)\pi_{\theta_k}(a_t^+|s_t)] \tag{A.1}$$

$$\text{where} \;\; p_t(l) = \frac{\alpha_t(l)}{\sum_{l_i=1}^{L}\alpha_t(l_i)}. \tag{A.2}$$

To determine probabilistic targets for the stop actions, we associate the edges in Figure A.1 between nodes of the same or subsequent sub-policies respectively with $\bar{a}_{stop}$ and a_{stop}. For $l = 1$, nodes depend only on a single relevant edge from the same sub-policy at the previous time-step, while for all nodes with $l > 1$ we take into account

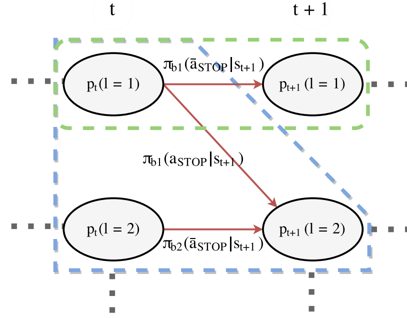


Figure A.1: CTC-based computation of stop action targets. The green and blue areas respectively depict the relations for $l = 1$ in Equation A.3 and $l > 1$ in Equation A.4

edges from the same sub-policy and the previous sub-policy at the previous time-step.

$$p_{t+1}(1) = p_t(1) \cdot \pi_1(\bar{a}_{stop}|s_{t+1})] \tag{A.3}$$

$$p_{t+1}(l+1) = p_t(l+1) \cdot \pi_{l+1}(\bar{a}_{stop}|s_{t+1}) + p_t(l) \cdot \pi_l(a_{stop}|s_{t+1}) \tag{A.4}$$

$$\pi_l(a_{stop}|s_t) = 1 - \pi_l(\bar{a}_{stop}|s_t) \tag{A.5}$$

Based on Equations (A.3), (A.4) and (A.5), we can derive the targets in Equation (A.6) for $t = 1, ..., T-1$. Starting with the targets for $l = 1$ we can compute the targets for $l+1$ based on the targets for l and the forward variables $\alpha_t(l)$.

$$\pi_l(\bar{a}_{stop}|s_{t+1}) \begin{cases} \frac{p_t(l)}{p_{t+1}(l)}, & \text{if } l = 1, \\ \frac{p_{t+1}(l+1)}{p_t(l+1)} \\ \sim\sim - \frac{p_t(l)\cdot(1-\pi_l(\bar{a}_{stop}|s_{t+1}))}{p_t(l+1)} & \text{if } l > 1. \end{cases} \tag{A.6}$$

Bibliography

[1] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first International Conference on Machine Learning (ICML)*, page 1. ACM, 2004. (Cited on pages 23, 24, 62, and 64.)

[2] P. Abbeel, D. Dolgov, A. Y. Ng, and S. Thrun. Apprenticeship learning for motion planning with application to parking lot navigation. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 1083–1090. IEEE, 2008. (Cited on page 42.)

[3] N. Abdo, H. Kretzschmar, L. Spinello, and C. Stachniss. Learning manipulation actions from a few demonstrations. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 1268–1275. IEEE, 2013. (Cited on page 74.)

[4] N. Aghasadeghi and T. Bretl. Maximum entropy inverse reinforcement learning in continuous state spaces with path integrals. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 1561–1566. IEEE, 2011. (Cited on page 58.)

[5] J. Andreas, D. Klein, and S. Levine. Modular multitask reinforcement learning with policy sketches. In *International Conference on Machine Learning*, pages 166–175, 2017. (Cited on pages 74, 76, and 85.)

[6] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009. (Cited on pages 3, 11, 62, and 75.)

[7] Y. M. Assael, B. Shillingford, S. Whiteson, and N. de Freitas. Lipnet: Sentence-level lipreading. *arXiv preprint arXiv:1611.01599*, 2016. (Cited on page 76.)

[8] M. Babes, V. Marivate, K. Subramanian, and M. L. Littman. Apprenticeship learning about multiple intentions. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 897–904, 2011. (Cited on page 24.)

[9] A. Billard, S. Calinon, R. Dillmann, and S. Schaal. Robot programming by demonstration. In *Springer handbook of robotics*, pages 1371–1394. Springer, 2008. (Cited on page 11.)

[10] C. M. Bishop. Pattern recognition. *Machine Learning*, 2006. (Cited on page 64.)

[11] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. *arXiv:1604.07316*, 2016. (Cited on page 62.)

[12] A. Boularias, J. Kober, and J. Peters. Relative entropy inverse reinforcement learning. In *AISTATS*, pages 182–189, 2011. (Cited on pages 43 and 64.)

[13] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11(1):94, 1999. (Cited on page 2.)

[14] C. Breazeal and B. Scassellati. Robots that imitate humans. *Trends in cognitive sciences*, 6(11): 481–487, 2002. (Cited on page 4.)

[15] A. Byravan, M. Monfort, B. Ziebart, B. Boots, and D. Fox. Graph-based inverse optimal control for robot manipulation. In *Proceedings of the International Joint Conference on Artificial Intelligence*,

2015. (Cited on page 42.)

[16] R. Caruana. Multitask learning. In *Learning to learn*, pages 95–133. Springer, 1998. (Cited on page 20.)

[17] D. S. Chaplot, K. M. Sathyendra, R. K. Pasumarthi, D. Rajagopal, and R. Salakhutdinov. Gated-attention architectures for task-oriented language grounding. *arXiv preprint arXiv:1706.07230*, 2017. (Cited on pages 76 and 90.)

[18] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014. (Cited on page 83.)

[19] K. Dautenhahn and C. L. Nehaniv. *Imitation in animals and artifacts*. MIT Press Cambridge, MA, 2002. (Cited on page 59.)

[20] R. Dearden and C. Boutilier. Abstraction and approximate decision-theoretic planning. *Artificial Intelligence*, 89(1):219–283, 1997. (Cited on pages 31, 35, and 37.)

[21] M. L. e. a. Deepak Pathak. Zero-shot visual imitation. *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=BkisuzWRW`. (Cited on page 76.)

[22] C. Dimitrakakis and C. A. Rothkopf. Bayesian multitask inverse reinforcement learning. In *Recent Advances in Reinforcement Learning*, pages 273–284. Springer, 2012. (Cited on page 24.)

[23] Y. Duan, M. Andrychowicz, B. Stadie, J. Ho, J. Schneider, I. Sutskever, P. Abbeel, and W. Zaremba. One-shot imitation learning. *arXiv preprint arXiv:1703.07326*, 2017. (Cited on page 76.)

[24] S. Ekvall and D. Kragic. Learning task models from multiple human demonstrations. In *Robot and Human Interactive Communication, 2006. ROMAN 2006. The 15th IEEE International Symposium on*, pages 358–363. IEEE, 2006. (Cited on page 73.)

[25] M. J. Fard, S. Ameri, R. B. Chinnam, A. K. Pandya, M. D. Klein, and R. D. Ellis. Machine learning approach for skill evaluation in robotic-assisted surgery. *arXiv preprint arXiv:1611.05136*, 2016. (Cited on page 90.)

[26] G. Ferrer, A. G. Zulueta, F. H. Cotarelo, and A. Sanfeliu. Robot social-aware navigation framework to accompany people walking side-by-side. *Autonomous Robots*, pages 1–19, 2016. (Cited on page 62.)

[27] C. Finn, P. Christiano, P. Abbeel, and S. Levine. A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. *arXiv preprint arXiv:1611.03852*, 2016. (Cited on page 20.)

[28] C. Finn, S. Levine, and P. Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. *arXiv preprint arXiv:1603.00448*, 2016. (Cited on pages 43 and 58.)

[29] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017. (Cited on page 21.)

[30] C. Finn, T. Yu, T. Zhang, P. Abbeel, and S. Levine. One-shot visual imitation learning via meta-learning. *arXiv preprint arXiv:1709.04905*, 2017. (Cited on pages 4 and 76.)

[31] R. Fox, S. Krishnan, I. Stoica, and K. Goldberg. Multi-level discovery of deep options. *arXiv preprint arXiv:1703.08294*, 2017. (Cited on pages 74 and 75.)

[32] J. Fu, K. Luo, and S. Levine. Learning robust rewards with adversarial inverse reinforcement learning. *arXiv preprint arXiv:1710.11248*, 2017. (Cited on page 43.)

[33] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. *arXiv preprint arXiv:1404.2334*, 2014. (Cited on page 57.)

[34] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *ICRA 2015*, pages 3067–3074. IEEE, 2015. (Cited on page 58.)

[35] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and

Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014. (Cited on page 20.)

[36] A. Graves. Sequence transduction with recurrent neural networks. *arXiv preprint arXiv:1211.3711*, 2012. (Cited on page 76.)

[37] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376. ACM, 2006. (Cited on pages 74, 76, 78, 79, and 82.)

[38] D. H. Grollman and A. Billard. Donut as i do: Learning from failed demonstrations. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3804–3809. IEEE, 2011. (Cited on page 24.)

[39] E. T. Hall. The hidden dimension. 1966. (Cited on page 70.)

[40] P. Henderson, W.-D. Chang, P.-L. Bacon, D. Meger, J. Pineau, and D. Precup. OptionGAN: Learning Joint Reward-Policy Options using Generative Adversarial Inverse Reinforcement Learning. *ArXiv e-prints*, Sept. 2017. (Cited on pages 74 and 75.)

[41] P. Henry, C. Vollmer, B. Ferris, and D. Fox. Learning to navigate through crowded environments. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, pages 981–986. IEEE, 2010. (Cited on pages 23, 31, 42, 49, and 62.)

[42] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, G. Dulac-Arnold, et al. Deep q-learning from demonstrations. In *Proceedings of the Conference on Artificial Intelligence (AAAI)*, 2018. (Cited on page 4.)

[43] J. Ho and S. Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pages 4565–4573, 2016. (Cited on pages 20, 43, and 58.)

[44] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. (Cited on pages 60 and 66.)

[45] G. E. Hovland, P. Sikka, and B. J. McCarragher. Skill acquisition from human demonstration using a hidden markov model. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, volume 3, pages 2706–2711. Ieee, 1996. (Cited on page 73.)

[46] D.-A. Huang, L. Fei-Fei, and J. C. Niebles. Connectionist temporal modeling for weakly supervised action labeling. In *European Conference on Computer Vision*, pages 137–153. Springer, 2016. (Cited on page 76.)

[47] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–373, 2013. (Cited on page 11.)

[48] A. Irpan. Deep reinforcement learning doesn't work yet. *https://www.alexirpan.com/2018/02/14/rl-hard.html*, 2018. (Cited on page 3.)

[49] M. Kalakrishnan, P. Pastor, L. Righetti, and S. Schaal. Learning objective functions for manipulation. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 1331–1336. IEEE, 2013. (Cited on page 43.)

[50] R. E. Kalman. When is a linear control system optimal? *Journal of Basic Engineering*, 86(1):51–60, 1964. (Cited on pages 4 and 17.)

[51] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011. (Cited on pages 41, 44, 47, and 48.)

[52] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. (Cited on page 66.)

[53] W. B. Knox and P. Stone. Interactively shaping agents via human reinforcement: The TAMER framework. In *K-CAP*. ACM, 2009. (Cited on page 62.)

[54] W. B. Knox, P. Stone, and C. Breazeal. Training a robot via human feedback: A case study. In *ICSR*,

pages 460–470. Springer, 2013. (Cited on page 62.)

[55] M. Kobilarov, G. Sukhatme, Hyams, et al. People tracking and following with mobile robot using an omnidirectional camera and a laser. In *ICRA*, 2006. (Cited on page 62.)

[56] L. Kong, C. Dyer, and N. A. Smith. Segmental recurrent neural networks. *arXiv preprint arXiv:1511.06018*, 2015. (Cited on page 76.)

[57] G. Konidaris, S. Kuindersma, R. Grupen, and A. Barto. Robot learning from demonstration by constructing skill trees. *The International Journal of Robotics Research*, 31(3):360–375, 2012. doi: 10.1177/0278364911428653. URL `https://doi.org/10.1177/0278364911428653`. (Cited on pages 74 and 75.)

[58] G. Konidaris, L. Kaelbling, and T. Lozano-Perez. Constructing symbolic representations for high-level planning. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014. (Cited on page 21.)

[59] S. Krishnan, R. Fox, I. Stoica, and K. Goldberg. Ddco: Discovery of deep continuous options for robot learning from demonstrations. In *Conference on Robot Learning*, pages 418–437, 2017. (Cited on pages 74 and 75.)

[60] O. Kroemer, H. Van Hoof, G. Neumann, and J. Peters. Learning to predict phases of manipulation tasks as hidden states. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4009–4014. IEEE, 2014. (Cited on page 74.)

[61] O. Kroemer, C. Daniel, G. Neumann, H. Van Hoof, and J. Peters. Towards learning hierarchical skills for multi-phase manipulation tasks. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1503–1510. IEEE, 2015. (Cited on pages 74 and 75.)

[62] M. Kuderer, S. Gulati, and W. Burgard. Learning driving styles for autonomous vehicles from demonstration. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA), Seattle, USA*, volume 134, 2015. (Cited on page 23.)

[63] H. Kuzuoka, Y. Suzuki, J. Yamashita, and K. Yamazaki. Reconfiguring spatial formation arrangement by robot body orientation. In *International conference on Human-robot interaction*, 2010. (Cited on page 61.)

[64] M. Laskey, J. Lee, R. Fox, A. Dragan, and K. Goldberg. Dart: Noise injection for robust imitation learning. In *Conference on Robot Learning*, pages 143–156, 2017. (Cited on pages 15, 16, and 84.)

[65] B. Lau, K. O. Arras, and W. Burgard. Multi-model hypothesis group tracking and group size estimation. *International Journal of Social Robotics*, 2(1):19–30, 2010. (Cited on page 61.)

[66] S. M. LaValle. Rapidly-exploring random trees a new tool for path planning. 1998. (Cited on page 41.)

[67] H. M. Le, Y. Yue, and P. Carr. Coordinated multi-agent imitation learning. *arXiv:1703.03121*, 2017. (Cited on page 4.)

[68] S. Levine and V. Koltun. Continuous inverse optimal control with locally optimal examples. *arXiv:1206.4617*, 2012. (Cited on page 58.)

[69] S. Levine, Z. Popovic, and V. Koltun. Nonlinear inverse reinforcement learning with gaussian processes. In *Advances in Neural Information Processing Systems*, pages 19–27, 2011. (Cited on pages 19, 23, and 38.)

[70] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016. (Cited on page 3.)

[71] Y. Li, J. Song, and S. Ermon. Infogail: Interpretable imitation learning from visual demonstrations. In *Advances in Neural Information Processing Systems*, pages 3815–3825, 2017. (Cited on page 43.)

[72] R. Lioutikov, G. Neumann, G. Maeda, and J. Peters. Learning movement primitive libraries through probabilistic segmentation. *The International Journal of Robotics Research*, 36(8):879–894, 2017. (Cited on page 75.)

[73] M. Liwicki, A. Graves, S. Fernàndez, H. Bunke, and J. Schmidhuber. A novel approach to on-line handwriting recognition based on bidirectional long short-term memory networks. In *Proceedings of the 9th International Conference on Document Analysis and Recognition, ICDAR 2007*, 2007. (Cited

on page 76.)

[74] A. Lockerd and C. Breazeal. Tutelage and socially guided robot learning. In *Intelligent Robots and Systems, 2004.(IROS 2004).*, 2004. (Cited on page 62.)

[75] W.-Y. G. Louie and G. Nejat. A learning from demonstration system architecture for robots learning social group recreational activities. In *IROS, 2016 IEEE/RSJ International Conference on*. IEEE. (Cited on page 62.)

[76] S. Manschitz, J. Kober, M. Gienger, and J. Peters. Learning to sequence movement primitives from demonstrations. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 4414–4421. IEEE, 2014. (Cited on page 75.)

[77] H. Mei, M. Bansal, and M. R. Walter. Listen, attend, and walk: Neural mapping of navigational instructions to action sequences. In *AAAI*, volume 1, page 2, 2016. (Cited on pages 76 and 90.)

[78] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015. (Cited on page 3.)

[79] G. Neu and C. Szepesvári. Training parsers by inverse reinforcement learning. *Machine learning*, 77 (2-3):303–337, 2009. (Cited on pages 23 and 38.)

[80] A. Y. Ng, S. J. Russell, et al. Algorithms for inverse reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 663–670, 2000. (Cited on pages 4 and 23.)

[81] S. Niekum, S. Osentoski, G. Konidaris, and A. G. Barto. Learning and generalization of complex tasks from unstructured demonstrations. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5239–5246, 2012. (Cited on page 75.)

[82] S. Niekum, S. Osentoski, G. Konidaris, S. Chitta, B. Marthi, and A. G. Barto. Learning grounded finite-state representations from unstructured demonstrations. *The International Journal of Robotics Research*, 34(2):131–157, 2015. (Cited on pages 73 and 75.)

[83] B. Okal and K. O. Arras. Learning socially normative robot navigation behaviors with bayesian inverse reinforcement learning. (Cited on pages 42, 49, and 55.)

[84] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, J. Peters, et al. An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, 7(1-2):1–179, 2018. (Cited on page 11.)

[85] E. Pacchierotti, H. I. Christensen, and P. Jensfelt. Embodied social interaction for service robots in hallway environments. In *Field and Service Robotics*, pages 293–304. Springer, 2006. (Cited on page 31.)

[86] L. Palmieri and K. O. Arras. A novel rrt extend function for efficient and smooth mobile robot motion planning. In *IROS 2014*, pages 205–211. IEEE, 2014. (Cited on page 55.)

[87] L. Palmieri, S. Koenig, and K. O. Arras. Rrt-based nonholonomic motion planning using any-angle path biasing. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 2775–2781. IEEE, 2016. (Cited on page 57.)

[88] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann. Probabilistic movement primitives. In *Advances in neural information processing systems*, pages 2616–2624, 2013. (Cited on pages 11 and 75.)

[89] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal. Learning and generalization of motor skills by learning from demonstration. In *2009 IEEE International Conference on Robotics and Automation*, pages 763–768, May 2009. doi: 10.1109/ROBOT.2009.5152385. (Cited on page 75.)

[90] N. Pérez-Higueras, F. Caballero, and L. Merino. Teaching robot navigation behaviors to optimal rrt planners. *International Journal of Social Robotics*, 10(2):235–249, 2018. (Cited on page 58.)

[91] B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964. (Cited on page 49.)

[92] L. Y. Pratt. Discriminability-based transfer between neural networks. In *Advances in neural information*

processing systems, pages 204–211, 1993. (Cited on page 20.)

[93] Y. Qing-xiao, Y. Can, F. Zhuang, and Z. Yan-zheng. Research of the localization of restaurant service robot. *International Journal of Advanced Robotic Systems*, 7(3):18, 2010. (Cited on page 59.)

[94] D. Ramachandran and E. Amir. Bayesian inverse reinforcement learning. *Urbana*, 2007. (Cited on pages 23 and 24.)

[95] N. Ratliff, D. Bradley, J. A. Bagnell, and J. Chestnutt. Boosting structured prediction for imitation learning. *Robotics Institute*, page 54, 2007. (Cited on pages 20 and 23.)

[96] N. Ratliff, B. Ziebart, K. Peterson, J. A. Bagnell, M. Hebert, A. K. Dey, and S. Srinivasa. Inverse optimal heuristic control for imitation learning. AISTATS, 2009. (Cited on page 42.)

[97] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 489–494. IEEE, 2009. (Cited on pages 13, 42, and 47.)

[98] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich. Maximum margin planning. In *Proceedings of the 23rd international conference on Machine learning*, pages 729–736. ACM, 2006. (Cited on pages 19, 23, 24, 38, 41, 42, 45, and 46.)

[99] S. Ross and D. Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668, 2010. (Cited on page 15.)

[100] S. Ross, G. J. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics*, pages 627–635, 2011. (Cited on pages 15 and 67.)

[101] C. A. Rothkopf and C. Dimitrakakis. Preference elicitation and inverse reinforcement learning. In *Machine Learning and Knowledge Discovery in Databases*, pages 34–48. Springer, 2011. (Cited on pages 17 and 38.)

[102] S. Russell. Learning agents for uncertain environments. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 101–103. ACM, 1998. (Cited on page 17.)

[103] S. Schaal. Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. In *Adaptive motion of animals and machines*, pages 261–280. (Cited on pages 73 and 75.)

[104] K. Shiarlis, J. Messias, M. Someren, S. Whiteson, J. Kim, J. Vroon, G. Englebienne, K. Truong, V. Evers, N. Pérez-Higueras, et al. TERESA: a socially intelligent semi-autonomous telepresence system. 2015. (Cited on pages 51, 56, and 59.)

[105] K. Shiarlis, J. Messias, and S. Whiteson. Inverse reinforcement learning from failure. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 1060–1068. International Foundation for Autonomous Agents and Multiagent Systems, 2016. (Cited on pages 7 and 52.)

[106] K. Shiarlis, J. Messias, and S. Whiteson. Acquiring social interaction behaviours for telepresence robots via deep learning from demonstration. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 37–42. IEEE, 2017. (Cited on page 8.)

[107] K. Shiarlis, J. Messias, and S. Whiteson. Rapidly exploring learning trees. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 1541–1548. IEEE, 2017. (Cited on page 8.)

[108] K. Shiarlis, M. Wulfmeier, S. Salter, S. Whiteson, and I. Posner. Taco: Learning task decomposition via temporal alignment for control. *International Conference on Machine Learning. 2018.*, 2018. (Cited on page 9.)

[109] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016. (Cited on page 3.)

[110] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):

1929–1958, 2014. (Cited on page 90.)

[111] C. Sungjoon, K. Eunwoo, L. Kyungjae, and O. Songhwai. Leveraged non-stationary gaussian process regression for autonomous robot navigation. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, 2015. (Cited on page 24.)

[112] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998. (Cited on page 2.)

[113] R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999. (Cited on pages 20, 21, and 75.)

[114] U. Syed and R. E. Schapire. A game-theoretic approach to apprenticeship learning. In *Advances in neural information processing systems*, pages 1449–1456, 2007. (Cited on page 38.)

[115] S. Thrun and L. Pratt. *Learning to learn*. Springer Science & Business Media, 2012. (Cited on page 20.)

[116] S. Thrun, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, D. Hähnel, C. Rosenberg, N. Roy, J. Schulte, et al. Minerva: A second-generation museum tour-guide robot. In *Robotics and automation, 1999. Proceedings. 1999 IEEE international conference on*, volume 3. IEEE, 1999. (Cited on pages 51 and 59.)

[117] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012. (Cited on page 86.)

[118] R. Triebel, K. Arras, R. Alami, L. Beyer, S. Breuers, R. Chatila, M. Chetouani, D. Cremers, V. Evers, M. Fiore, et al. Spencer: A socially aware service robot for passenger guidance and help in busy airports. 2015. (Cited on pages 51 and 59.)

[119] D. Vasquez, B. Okal, and K. O. Arras. Inverse reinforcement learning algorithms and features for robot navigation in crowds: an experimental comparison. In *International Conference on Intelligent Robots and Systems*, pages 1341–1346. IEEE, 2014. (Cited on pages 23, 31, 38, 42, 49, and 52.)

[120] J. Vroon, M. Joosse, M. Lohse, J. Kolkmeier, J. Kim, K. Truong, G. Englebienne, D. Heylen, and V. Evers. Dynamics of social positioning patterns in group-robot interactions. In *RO-MAN*, 2015. (Cited on page 61.)

[121] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992. (Cited on page 13.)

[122] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, pages 5–32. Springer, 1992. (Cited on page 13.)

[123] M. Wulfmeier, P. Ondruska, and I. Posner. Maximum entropy deep inverse reinforcement learning. *arXiv:1507.04888*, 2015. (Cited on pages 19 and 58.)

[124] D. Xu, S. Nair, Y. Zhu, J. Gao, A. Garg, L. Fei-Fei, and S. Savarese. Neural task programming: Learning to generalize across hierarchical tasks. *arXiv:1710.01813*, 2017. (Cited on page 75.)

[125] J. Zheng, S. Liu, and L. M.-S. Ni. Robust bayesian inverse reinforcement learning with sparse behavior noise. In *Proceedings of the National Conference on Artificial Intelligence, 28th AAAI Quebec City, Canada*, page 2198, 2014. (Cited on page 24.)

[126] K. Zhou, J. C. Doyle, K. Glover, et al. *Robust and optimal control*, volume 40. Prentice hall New Jersey, 1996. (Cited on page 13.)

[127] B. D. Ziebart. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. PhD thesis, Carnegie Mellon University, 2010. (Cited on pages 24 and 26.)

[128] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, pages 1433–1438, 2008. (Cited on pages 19, 24, and 31.)

[129] B. D. Ziebart, J. A. Bagnell, and A. K. Dey. The principle of maximum causal entropy for estimating interacting processes. *Information Theory, IEEE Transactions on*, 59(4):1966–1980, 2013. (Cited on page 31.)

Samenvatting

Een groot gedeelte van hedendaagse kunstmatige intelligentie onderzoek focust op het ontwikkelen van agents, zoals robots, video spel karakters en virtuele assistenten, die veilig en autonoom kunnen handelen voor lange tijdsperiodes. Een significant obstakel in dit streven is het moeten definiren van het gewenste gedrag, ofwel door handmatig programmeren dan wel door de definitie van kost functies. Leren van Demonstratie (LvD) probeert dit obstakel uit de weg te gaan door agents direct gedrag policies aan te leren door middel van het observeren van acties van andere agents, zoals mensen. Als zodanig, dient het als een bijzonder krachtige programmeer interface voor robots en virtuele agents in zaken waar het definiren van verplicht gedrag handmatig niet triviaal is. Desondanks vereisen LvD methodes alsnog de implementatie van een machine learning pipeline om het success te verzekeren. Dit bevat het verzamelen van de benodigde data, het bouwen van de juist staat representaties, het leren van verschillende taken en het inzetten van de agent in haar omgeving. Deze scriptie maakt praktische en theoretische contributies gericht op verschillende niveaus van de LvD pipeline, met een focus op het simpel maken om LvD methodes uit te voeren en in te zetten voor toepassingsmogelijkheden zoals sociale robotica en robot manipulatie. Hoofdstuk 3 beschrijft Inverse Reinforcement Learning from Failure (IRLF), welke de situatie bekijkt waar mislukte demonstraties ook worden meegenomen in de demonstratie dataset en een methode ontwikkelt die agents laat leren wat niet te doen van demonstraties. De methode wordt toegepast op een verscheidenheid aan benchmark problemen zowel als robot data. Wij laten zien dat mislukte demonstraties inderdaad kunnen leiden tot betere representaties van de beloningsfunctie die de de data onderbouwt, resulterend in veiliger gedrag. Hoofdstuk 4 introduceert Rapidly Exploring Learning Trees (RLT*). Dit is een algoritme dat het mogelijk maakt om kost functies van sample gebaseerde planners te leren, typisch gebruikt in robot toepassing zoals navigatie en manipulatie. Wij passen RLT* toe op een sociaal navigatie domein die haar capaciteiten demonstreert om sneller te leren en beter werkt dan de niet sample gebaseerde tegenhanger, zowel als leren in domeinen met kinematische beperkingen Hoofdstuk 5 beschrijft een toepassing van LvD

en deep learning op sociale robotica. Wij laten zien dat moderne leer algoritmes, zoals convolutionele en recurrente netwerken gebruikt kunnen worden om problemen aan te pakken die opkomen als sociale robots interacteren met mensen, zoals variabele input representatie groottes en sensor ruis. Dit hoofdstuk is een mooi voorbeeld van praktische overwegingen die nodig zijn voor toepassingen in de echte wereld en hoe LvD gebruikt kan worden om verschillende gedragsvormen makkelijk en met minimale technische kennis te programmeren. Hoofdstuk 6 beschrijft Temporal Alignment for Control (TACO). We kijken naar situaties waar demonstraties executies van verschillende sub-taken bevatten en vergezeld zijn door een zwak supervisory signaal, een schets, die beschrijft wat deze sub-taken zijn. TACO gelijktijdig aligns de demonstratie met de schets en leert de benodigde policies om taken te kunnen uitvoeren met een grotere complexiteit. Wij evalueren onze methode op een verscheidenheid aan domeinen, inclusief een beeld gebaseerde manipulatie taak die laat zien dat TACO even goed werkt als volledig supervised methodes met maar een fractie van de annotatie inspanning.

Summary

A large portion of today's artificial intelligence research focuses on developing agents, such as robots, video game characters and virtual assistants, that can act safely and autonomously over long periods of time. A significant obstacle in this endeavour is having to define the desired behavior, either through manual programming or through the definition of cost functions. Learning from Demonstration (LfD) seeks to circumvent this obstacle by allowing agents to learn a behavior by observing the actions of other agents, such as humans. As such, it serves as a particularly powerful programming interface for robots and virtual agents in cases where defining the required behavior by hand is non-trivial. LfD methods however, still require the implementation of a full machine learning pipeline to ensure their success. This includes collecting the required data, engineering the right state representations, learning different tasks and deploying the agent in its environment. This thesis makes practical and theoretical contributions aimed at different levels of the LfD pipeline, with a focus on making LfD methods easy to perform and deploy in a range of applications including social robotics and robotic manipulation.

Chapter 3 describes Inverse Reinforcement Learning from Failure (IRLF). It considers the case where failed demonstrations are also included within our demonstration dataset and develops a method that allows agents what *not* to do from demonstrations. The method is applied to a variety of benchmark problems as well as real robot data. We show that indeed failed demonstrations can be leveraged to learn a better representation of the reward function underpinning the data resulting in safer behavior.

Chapter 4 introduces Rapidly Exploring Learning Trees (RLT*). This is an algorithm that allows learning cost functions for sample based planners typically used in robotic applications such as navigation and manipulation. We apply RLT* to a social navigation domain, demonstrating its capability to learn faster and better than its non-sampling based counterpart, as well as learn in domains with kinematic constraints.

Chapter 5 describes an application of LfD and deep learning to social robotics. We show that modern learning tools such as convolutional and recurrent networks can be

employed to tackle problems that arise when social robots interact with people, such as variable input representation sizes and sensor noise. This chapter is a nice example of the practical considerations required for real world applications and how LfD can be used to program different behaviors very easily and with minimal technical knowledge.

Chapter 6 describes Temporal Alignment for Control (TACO). We consider cases where demonstrations contain execution of various sub-tasks and are accompanied by a weak supervisory signal, a sketch, describing what these sub-tasks are. TACO simultaneously aligns the demonstrations with the sketch and learns the required policies. At test time we are then able to compose the learned policies together to execute tasks of much larger complexity. We evaluate our method or a range of domains including and image-based manipulation task showing that TACO can perform equally well to fully supervised methods with only a fraction of the annotation effort.