

Institute of Parallel and Distributed Systems

University of Stuttgart  
Universitätsstraße 38  
D-70569 Stuttgart

Masterarbeit

# **Hierarchical Adversarial Imitation Learning from Motion Capture Data**

Simon Hagenmayer

<b>Course of Study:</b>	Informatik
<b>Examiner:</b>	Dr. Jim Mainprice
<b>Supervisor:</b>	Philipp Kratzer, M.Sc.
<b>Commenced:</b>	2020-04-01
<b>Completed:</b>	2020-11-27
<b>CR-Classification:</b>	I.2.6



## **Abstract**

A lot of problems in Imitation Learning can be split into multiple low-level tasks that need to be executed in sequence or parallel. Likewise in the area of human motion prediction, the actions of the human are frequently defined by a combination of linked subgoals. However, recent work often focuses on solving the problem statement as a single task or lies its attention on the general accuracy of the predicted motion. In this thesis, a two-step framework is introduced which is able to execute a high-level policy using low-level tasks. We implement different network structures for goal-conditioned human motion prediction by including goal loss regularization and adversarial methods, and demonstrate that they can be combined for long-term planning using Maximum Entropy Inverse Reinforcement Learning on a simplified model of the environment. The framework is tested on a simple pick and place task in simulation, using recorded demonstrations of multiple humans executing the higher-level task. Results show, that the trained networks are capable of reaching subgoals with great accuracy and can be used in combination with the presented Inverse Reinforcement Learning algorithm to learn the long-term objective.



## Kurzfassung

Viele der Probleme des Imitation Learnings können in mehrere kleine Aufgaben aufgeteilt werden, die sequentiell oder parallel ausgeführt werden müssen. Ebenso sind im Bereich der Human Motion Prediction die Aktionen des Menschen oft als eine Kombination aus verknüpften Teilzielen definiert. Jedoch fokussieren sich aktuelle Arbeiten oft darauf, die Problemstellung als einzelne Aufgabe anzugehen, oder legen ihre Aufmerksamkeit auf die allgemeine Genauigkeit der prognostizierten Bewegung. In dieser Thesis stellen wir ein Zwei-Schritte System vor, das in der Lage ist eine high-level Strategie durch Benutzung von low-level Aufgaben auszuführen. Wir implementieren verschiedene Netzwerkstrukturen für ziel-konditionierte Human Motion Prediction durch Einschluss von Goal Loss Regularisierung und adversarial Methoden, und zeigen dass sie für langfristige Planung mittels Maximum Entropy Inverse Reinforcement Learning auf einem vereinfachten Modell der Umgebung verwendet werden können. Das System wird auf einer einfachen Pick and Place Aufgabe getestet unter Verwendung von aufgezeichneter Demonstrationen verschiedener Personen beim Ausführen der higher-level Aufgabe. Die Ergebnisse zeigen, dass die trainierten Netzwerke in der Lage sind, Teilziele mit großer Genauigkeit zu erreichen und in Kombination mit dem präsentierten Inverse Reinforcement Learning Algorithmus benutzt werden können, um das langfristige Ziel zu erreichen.



# Contents

<b>1</b>	<b>Introduction</b>	<b>17</b>
<b>2</b>	<b>Related Work</b>	<b>19</b>
2.1	Adversarial Learning . . . . .	19
2.2	Adversarial Imitation Learning . . . . .	20
2.3	RNN based Human Motion Prediction . . . . .	20
2.4	Hierarchical Imitation Learning . . . . .	21
2.5	Goal-Conditioned Learning . . . . .	22
<b>3</b>	<b>Background</b>	<b>23</b>
3.1	Position-Velocity Recurrent Encoder-Decoder . . . . .	23
3.2	Generative Adversarial Networks . . . . .	24
3.3	Reinforcement Learning . . . . .	27
3.4	Inverse Reinforcement Learning . . . . .	29
3.5	Hierarchical Reinforcement Learning . . . . .	30
<b>4</b>	<b>Methods</b>	<b>33</b>
4.1	Vanilla VRED Implementation . . . . .	33
4.2	Goal-Conditioned VRED . . . . .	34
4.3	Goal-Conditioned VRED with Goal Loss . . . . .	36
4.4	Adversarial VRED . . . . .	37
4.5	Symbolic State Representation . . . . .	39
4.6	Maximum Entropy IRL Framework . . . . .	40
4.7	Heuristics for Trajectory Visualization . . . . .	41
<b>5</b>	<b>Experiments</b>	<b>45</b>
5.1	Dataset . . . . .	45
5.2	Training/Implementation Details . . . . .	46
5.3	Results/Evaluation . . . . .	47
<b>6</b>	<b>Future Work / Conclusion</b>	<b>57</b>
6.1	Conclusion . . . . .	58
	<b>Bibliography</b>	<b>59</b>
<b>A</b>	<b>Experiments on the Taxi Environment</b>	<b>65</b>
<b>B</b>	<b>Additional Examples for the Maximum Entropy IRL</b>	<b>67</b>





# List of Figures

1.1	RL in Games . . . . .	17
3.1	Vanilla VRED Structure . . . . .	23
3.2	GAN Structure . . . . .	25
3.3	Option Framework . . . . .	31
4.1	Trajectory and Goal Condition Visualization . . . . .	34
4.2	Goal-Conditioned VRED Structure . . . . .	35
4.3	Adversarial VRED Structure . . . . .	37
4.4	WalkToX Heuristic Example . . . . .	42
4.5	Place Heuristic Example . . . . .	42
5.1	Hierarchical Task Setup . . . . .	45
5.2	Goal-Conditioned VRED on the Pick and Place Dataset . . . . .	48
5.3	Goal-Conditioned VRED on the Walking Dataset . . . . .	49
5.4	Goal-Conditioned VRED with Goal Loss on the Pick and Place Dataset . . . . .	50
5.5	Goal-Conditioned VRED with Goal Loss on the Walking Dataset . . . . .	50
5.6	Learned Policy Prediction Examples . . . . .	53
5.7	Human Full Trajectory Prediction . . . . .	55
A.1	Taxi Environment AIRL with Different Experts . . . . .	65
A.2	Taxi Environment MaxQ and Q-Learning . . . . .	66



# List of Tables

4.1	MaxEnt IRL Perfect Imitation Example . . . . .	40
5.1	Qualitative Comparison for the Walking Dataset . . . . .	51
5.2	Qualitative Comparison for the Pick and Place Dataset . . . . .	52
5.3	MaxEnt IRL Perfect Imitation Example . . . . .	53
5.4	Learned Policy Full Trajectory Example . . . . .	54
A.1	Taxi Environment MaxEnt IRL . . . . .	65
B.1	MaxEnt IRL Permutation Example . . . . .	67
B.2	MaxEnt IRL Variation Example . . . . .	68



# List of Algorithms

1	Feature matching IRL pseudocode. . . . .	29
2	Tabular MaxEnt IRL pseudocode. . . . .	40



# List of Abbreviations

**GAN** Generative Adversarial Network. 23

**HRL** Hierarchical Reinforcement Learning. 23

**IRL** Inverse Reinforcement Learning. 18

**MaxEnt IRL** Maximum Entropy Inverse Reinforcement Learning. 18

**MDP** Markov Decision Process. 27

**RL** Reinforcement Learning. 18

**RNN** Recurrent Neural Network. 18

**SDF** Signed Distance Field. 41

**VRED** Position-Velocity Recurrent Encoder Decoder. 18



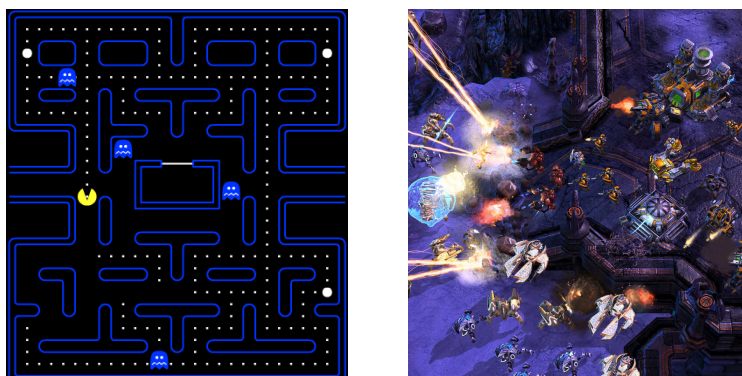


# 1 Introduction

In recent years, the area of Reinforcement Learning, where an agent is interacting with an environment by choosing actions and receiving rewards, made considerable progress. An easy example for RL is the arcade game Pac-Man (see Figure 1.1 left), where the goal is to pick up as much snacks as possible while being chased by hostile ghosts. Solution approaches were pushed with the introduction and extension of neural networks, that are able to move from discrete state and action spaces for the agent to complex continuous environments using Deep Learning. OpenAI DeepMinds AlphaStar [1] (see Figure 1.1 right), an agent for the online game StarCraft II, was already able to beat 99.8% of the whole player base of the game in 2019.

However, when the rewards given to the agent are very sparse or ill-defined, the RL problem can be hard to solve. A help would be the possibility to learn from an expert, that is already able to achieve a certain task. Learning from such demonstrations is done in Imitation Learning. Inverse Reinforcement Learning tries to extract the rewards from the environment by discovering which states and actions benefit the agent by reaching its objective. This way, small changes in the environment or the rewards can also be easier compensated for.

Learning to predict human motion is another active area of research. The goal is to reason from the given past behavior of the human and thereby predict its future trajectory. Example cases could be eating a sandwich, working or walking around. The problem statement is complicated, as it contains temporal information like the current timestep, but also positional and rotational information like the present joint states of the human.



**Figure 1.1:** Examples of RL learning used in Games: Pac-Man (left) and StarCraft II (right)

Combining the full human motion prediction problem with the IRL problem statement is a hard task to solve, as not only one works on a complex state and action space, but also at best tries to synthesize a humanlike trajectory as end result.

Human Motion Prediction in recent works is usually tackled by some kind of Recurrent Neural Network (RNN) [2, 3, 4]. The reason for that is, that they can encode temporal informations in their hidden state. Generally, methods use a state-to-sequence [5] or sequence-to-sequence [2] approach, where one or more human poses are taken as input, and the output is the prediction of its trajectory for a small timestep. Thereby, GRU cells are often favored over LSTM cells, as they require less computational power with similar results. In recent years, short-term predictions became more and more accurate. However, long-term behavior ( $> 500\text{ms}$ ) of the human is still very difficult to predict, especially if the network was trained on multiple distinct tasks like walking, reading a book and dish washing. That is why it is still an active area of research. In our work, we incorporate the concept of RNN but extend the input given to our networks by a goal condition.

When it comes to Inverse Reinforcement Learning (IRL), the most important task is to solve the ambiguity of the reward function. A common approach to do this is by choosing a reward function, where the learned policy exhibits the maximum amount of entropy [6]. State-of-the-art approaches extend this structure by including adversarial training [7], or even a hierarchical framework into the IRL process [8]. These frameworks then often contain a model-free Reinforcement Learning (RL) algorithm for solving its inner loop instead of the easier model-based RL methods. Nevertheless, inflating IRL algorithms often becomes computationally expensive, so one needs to balance the approach to the complexity of the task. In our case, we simplify our environment to a symbolic representation, so that we can use tabular Maximum Entropy Inverse Reinforcement Learning (MaxEnt IRL) for solving our task.

In our work, we approach the issue in two steps: We first tackle goal-conditioned short-term prediction by training multiple Recurrent Neural Networks on a walking and a pick and place dataset. Additionally, the idea of adversarial training is investigated. Secondly, we solve an IRL algorithm on a symbolic representation of the environment, and map the learned policy to comprehensible goal inputs for our networks.

In Chapter 2 we present related work to our topic, including recent achievements in the area of human motion prediction and Inverse Reinforcement Learning. Chapter 3 explains background knowledge like the Position-Velocity Recurrent Encoder Decoder (VRED) network as well as standard methods and theory used in the area of RL and Imitation Learning. Our experiment setup, execution and results are presented in Chapter 5. Finally, 6 provides an outlook for possible future work and improvements. It also draws a conclusion and summarizes the contributions and outcomes of the thesis.

## 2 Related Work

In this chapter, we present state-of-the-art works in the field of Adversarial Learning, Adversarial Imitation Learning, Human Motion Prediction and Goal-Conditioned Learning. Furthermore, we compare their properties to our framework, and explain how they differentiate from ours.

### 2.1 Adversarial Learning

Adversarial training or learning is a concept which gained popularity especially in the area of image generation and synthesis. Hence, in recent years, various variants of the original Generative Adversarial Network (GAN) [9] have emerged [10]: The conditional GAN (cGAN) is an extension where both the generator and discriminator are with respect to provided extra information [11]. Samples can then be generated e.g. conditioned on class labels of a multi-class classification task. Nowozin et al. [12] show that the GAN is special case of the family of  $f$ -GANs, that try to minimize the  $f$ -divergence between the generated and target probability distribution. Instead of trying to find an equilibrium of the two player zero-sum game, the Loss Sensitive-GAN [13] tries to learn a loss function, where the loss of real samples is smaller than that of fake samples by a certain margin.

Multiple works also focus on reducing the obstacles when training adversarial neural networks, which were still present in the original GAN. In their paper, Arjovsky et al. [14] introduce the Wasserstein-GAN, a novel approach that modifies the loss function used for the generator and the critic of the network. This way, it reduces commonly occurring GAN problems like mode collapse and improves training stability. Gulrajani et al. [15] extend this model by swapping the weight clipping suggested in the Wasserstein-GAN paper with a gradient penalty. Adversarial networks trained with this loss tend to learn more complex and meaningful functions, while also removing the sensitive weight clipping hyperparameter from the training procedure. Furthermore, various variants of WGAN or WGAN-GP can be found: The Banach Wasserstein GAN [16] replaces the  $l^2$  norm of the gradient penalty with a dual norm and generalizes the WGAN theory to Banach spaces. Zhou et al. show that for many distance metrics the gradient of the discriminative function does not help to minimize the divergence between generated and real data distribution, resulting in the proposed Lipschitz GAN [17].

In their work, Zhou et al. [18] investigate the effect of enforcing the Lipschitz continuity of neural networks. The authors also introduce a regularisation technique for computing an upper bound of the Lipschitz constant, usable for different feed-forward neural network

structures. Alta et al. [19] analyze different gradient penalty measures and compare their training stability. In comparison to our approach, these works focus mainly on improving adversarial training in general, rather than including it into RL or human motion prediction frameworks.

### 2.2 Adversarial Imitation Learning

With the success of adversarial networks, that try to minimize the divergence between two probability distributions, it makes sense to try to incorporate adversarial learning into the problem statement of imitation learning. Ho et al. propose GAIL ([20]), which draws a connection between GANs and IRL frameworks. A similar approach that shows the equivalence between GAN and an algorithm that performs MaxEnt IRL is GAN-Guided Cost Learning (GAN-GCL) [21]. In contrast to GAIL, where the only return is the learned policy, GAN-GCL allows to also recover the underlying reward structure. Fu et al. introduce AIRL ([7]) and the notion of disentangled rewards, i.e. that they are robust to changes in the environment. Results show that AIRL is on par or outperforming both the previous methods in traditional imitation learning setups as well as transfer learning. In their paper “Wasserstein Adversarial Imitation Learning” [22], Xiao et al. present a new IRL framework, which makes use of Kantorovich potentials as a reward function rather than hand-engineered rewards. In several control tasks, it achieves state-of-the-art results and realistic behaviour even with small numbers of expert demonstrations. The Option-Inverse Reinforcement Learning (oIRL) method [8] embeds the option-critic framework into the AIRL algorithm. They investigate transfer learning with options and are able to learn disentangled rewards, that are easily transferable to new environments. The presented works show promising results for varying environments. However, due to them being model-free IRL methods, an immense amount of sampling and computational power is needed. Our method tackles this issue for the motion prediction problem by splitting it into two easier solvable subproblems: short-term human motion prediction ( $\leq 1s$ ) and IRL for a simplified environment.

### 2.3 RNN based Human Motion Prediction

Human Motion Prediction is the task of forecasting the future trajectory of a human given its former behaviour. As this issue relates to a sequence-to-sequence problem, Recurrent Neural Networks are the commonly used approach in state-of-the-art research [23, 24, 25]. Wang et al. introduce the Variational Recurrent Encoder-Decoder network (VRED) [2], a sequence-to-sequence model using a quaternion loss implementation. Experiments on two important human motion prediction benchmarks show that it outperforms similar methods in both short- and long-term prediction. More recent approaches are opting for an adversarial training framework: The Social GAN [5] tries to predict 2D trajectories for multiple humans at the same time. In the process, each person is encoded as a single LSTM

network, and a state-of-the-art pooling layer is used to aggregate the information of the LSTMs. The BiHMP-GAN [3] is an extension of the HP-GAN [26], a probabilistic generative model used for 3D human motion prediction. In their work “Adversarial Geometry Aware Human Motion Prediction”, Wang et al. [4] implement two different discriminators to distinguish human poses from generated ones. Their proposed “fidelity” and “continuity” discriminators are responsible for analysing smoothness and progressiveness of the given trajectories respectively. In their work, Wang et al. [27] combine a pretrained RNN with adversarial training. Using a refiner layer and a regularized adversarial loss, they try to optimize the generated RNN trajectory. These works differ from our proposed approach in their objective, which is plain sequence-to-sequence learning with no long-term planning, while our networks incorporate the notion of goals and intentions, which can be used for the preceding.

## 2.4 Hierarchical Imitation Learning

Hierarchical Imitation Learning tries to frame long-horizon imitation learning problems in a hierarchical structure. In recent years, most developed approaches make use of the option framework: In their work, Fox et al. [28] introduce the DDO algorithm, that directly extracts parameterized options from demonstration data. Furthermore, it is capable of doing this in a multi-level hierarchical way using recursion. Directed-Info GAIL [29] is an approach that builds a Dynamic Bayesian Network from expert demonstrations and uses maximization of directed information for solving it. In their work, they show that although using a different theoretic perspective, it can be seen as an option framework model. The OptionGAN [30] is an extension of the options framework, which combines options with generative adversarial inverse reinforcement learning. This makes it similar to the already mentioned oIRL approach, however without returning disentangled rewards and therefore being inferior in transfer tasks.

Other hierarchical approaches include using a mixture of policies [31], where each policy tries to optimize an own reward function taken from unstructured and unlabeled demonstrations of multiple experts. This way it is able to learn the notion of intention and to perform variable tasks based on the given intention input. CompILE [32] learns a latent code for segments from sequential data using auto-encoders. A hierarchical policy can then be used to select actions in the latent space, which are mapped to low-level primitive actions by the learned decoders. In their work, Gupta et al. [5] introduce Relay Policy Learning, which combines IRL and RL in a multi-step algorithm. They show that the model is able to learn a single policy which is capable of reaching multiple concatenated subgoals. In his thesis, Prakash [33] presents a hierarchical structured policy, which is able to solve long-horizon tasks like setting up a table or clearing it. In contrast to the proposed methods, rather than using an explicit hierarchical layout, we learn on a symbolic state representation where our learned goal-conditioned networks can be seen similar to options executing a lower-level policy. In contrast to our work, these approaches usually train their low-level policies directly in the IRL process, instead of splitting the task into two parts.

### 2.5 Goal-Conditioned Learning

A typical RL framework tries to learn a single task by optimizing an underlying reward function. Nevertheless, we often want a policy to learn multiple tasks. This can be done by conditioning the decision in each state not only on the current observation, but also on another input: a goal. An important goal-conditioned learning approach is presented in the “Hindsight Experience Replay” paper from Andrychowicz et al. [34]. In their work, the authors show a technique that improves the sample efficiency for goal-conditioned learning algorithms and is able to learn from sparse rewards.

Multiple approaches try to incorporate the goal condition into some kind of value function: Already in 1993, Kaelbling et al. [35] introduced dynamic goal learning (DG-Learning), which is similar to Q-Learning. Instead of calculating state-action values though, its goal is to compute a policy which minimizes the expected number of timesteps to arrive at a certain goal. The more recent work “Temporal Difference Models: Model-free Deep RL for Model-Based Control” [36] presents a goal-conditioned value function, that exploits the connection of model-free and model-based RL. This way, it increases the sample efficiency for model-free RL methods and provides superior training performance over other state-of-the-art model-free approaches. The idea of Universal value function approximators [37] is to approximate the optimal value function by a single powerful approximator, which generalizes well over both: the potentially large state and the goal space. Example usages for such a model are for transfer learning or also as features to represent the state.

The structure of state-of-the-art methods also pivots on the available expert data. Nair et al. [38] provide an approach that is able to execute goal-conditioned RL without seeing any demonstration data. Training is done solely using a Variational Autoencoder to self-specify goals and a Q-function approximator to learn a corresponding policy in the encoded latent space. Lynch et al. [39] try to derive an agent policy from unlabeled play data. By minimizing the KL-divergence between latent plan proposals and their ground truth, the model is able to generalize to multiple different manipulation tasks via a self-supervised algorithm. Another work that tangents goal-conditioned learning is from Kratzer et al. [40]. While the focus of the work lies in human motion prediction, the authors add the possibility to incorporate external controls during online execution including a goal constraint that tries to navigate the human towards a certain objective. In comparison to the presented methods that mostly try to fit goal generalization directly into a RL framework or value function, our proposition is a two-step approach which uses a combination of goal-conditioned supervised learning and IRL.

### 3 Background

This chapter is split up into five different parts: We first introduce the Position-Velocity Recurrent Encoder-Decoder network, as well as the theory behind Generative Adversarial Networks (GANs), which both are used for our goal-conditioned network implementation. Afterwards, we explain the basics of RL, IRL and Hierarchical Reinforcement Learning (HRL).

#### 3.1 Position-Velocity Recurrent Encoder-Decoder

The Position-Velocity Recurrent Encoder Decoder (VRED) [2] is a sequence-to-sequence model based on two RNNs and an improvement of previous works [24, 23] for both short term and long-term predictions. The encoder RNN takes as input a sequence of feature vectors  $X = \{x_1, x_2, \dots, x_n\}$ , that are encoded into a hidden state  $h_n$ . The decoder RNN then decodes the hidden state into a new output sequence  $\tilde{X} = \{\tilde{x}_{n+1}, \tilde{x}_{n+2}, \dots, \tilde{x}_{n+m}\}$ , using its last prediction  $\tilde{x}_{t-1}$  and  $h_{t-1}$  as new input  $x_t$  in each iteration. The length of the input and the prediction are variable.

The VRED distinguishes itself from previous methods [41, 42, 43, 44] by using three types of input data: the human poses and their velocities, as well as a position embedding, which is used to encode temporal information. This also sets it apart from RED, which had problems with inaccurate and unnatural looking poses when doing long-term predictions for longer than 1.5 seconds. In contrast to that, poses predicted by the VRED look natural up to 4 seconds and do not drift away that fast [2].

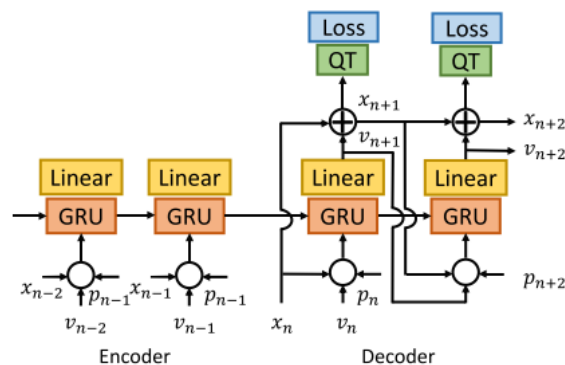


Figure 3.1: Structure of the VRED, taken from Wang et al. [2]

The structure of the network can be seen in Figure 3.1. The RNN layers are using GRU cells instead of LSTM cells [45] due to their superiority, i.e. them being relative computationally inexpensive [46] while showing comparable results. There are usually three different methods to represent human poses: euler angles, exponential maps [47] and quaternions. As euler angles and exponential maps suffer from singularity issues like gimbal lock - which results in a loss of degree of freedom - it is favorable to use quaternions for the pose representation. However, directly transforming the input to quaternions would imply the need of preservation of the unit length of the quaternions in the predictions.

To bypass these additional computational costs, the VRED uses as input  $x_t$  exponential maps like similar approaches [41]. In addition to that, velocity inputs  $v_t$  are calculated by forward differences on the input data. A skip connection ensures, that the GRU cells just learn velocities. Furthermore, a position embedding  $p_t$  is added, consisting of a combination of sine and cosine frequencies to encode temporal position information. It motivates the model to behave with respect to its current time frame. The final prediction  $\tilde{x}_t$  for timestep  $t$  is then computed as the sum of the last state and the output of the RNN cell. In the self-conditioning loop of the network (right part Figure 3.1), no external input data is used. Instead, the last velocity prediction and calculated pose are directly fed back into the model for the next prediction.

In the end, the pose is converted to the quaternion space by the introduced Quaternion Transformation (QT) layer for the loss computation. In this way, the VRED uses a loss function in the unit quaternion space, trying to minimize the differences between the ground truth data and the predicted data. In the need of a robust loss function w.r.t outliers and keeping the unit length of quaternions, they use the mean absolute error loss

$$L = \frac{1}{m} \sum_{t=1}^m \|g(x_{n+t}) - g(\tilde{x}_{n+t})\|_1 \quad (3.1)$$

where  $g$  is the Quaternion Transformation (QT),  $x_{n+t}$  is the observed pose and  $\tilde{x}_{n+t}$  the predicted pose at timestep  $n + t$ .

Tests on the Human 3.6 dataset [48] and the CMU dataset <sup>1</sup> showed that the VRED was superior to similar methods like the Quaternet [25] or the Conv Seq2Seq2 [43] for both short-term prediction (<500ms) as well as long-term prediction (>500ms).

## 3.2 Generative Adversarial Networks

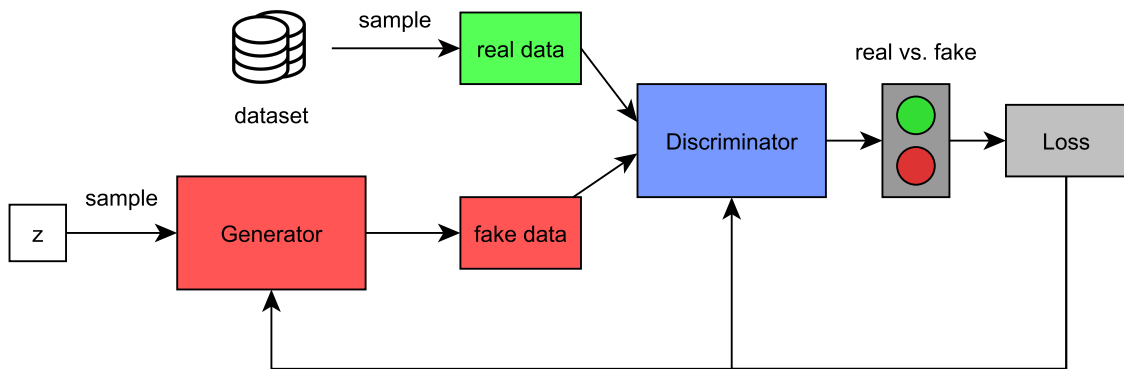
Generative Modeling is an unsupervised learning method that learns patterns and structures of input data, so that the model is able to generate new samples which could likely be drawn from the original dataset. Generative Adversarial Networks estimate these generative models via an adversarial process, which is represented by two neural networks that are competing against each other [9].

---

<sup>1</sup>Available at <http://mocap.cs.cmu.edu/>



A conceptual architecture of a GAN can be seen in Figure 3.2. The generator  $G$  in the adversarial modeling framework is a neural network, often modeled as multilayer perceptron or CNN, that maps a prior of latent input noise  $p_z(z)$  to a data space  $G(z; \theta_g)$ , parameterized by some weights  $\theta$ . The discriminator network  $D(x; \theta_d)$  then tries to discriminate samples drawn from the generator with random samples from the ground truth data  $x_{gt}$ .  $D(x)$  thereby determines the probability of sample  $x$  being from the original data rather than from the generator distribution. In an alternating fashion, the generator and discriminator are then trained to achieve their objective: The discriminator tries to maximize the probability of assigning the correct labels to fake samples (from  $G$ ) and ground truth samples. Respectively,  $G$  is trained to maximize the probability of  $D$  assigning a ground truth label to its generated fake samples.



**Figure 3.2:** General architecture of a Generative Adversarial Network

In summary,  $D$  and  $G$  play the following two-player minimax game with the value function  $V(G, D)$ :

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{x_{gt}}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log 1 - D(G(z))] \quad (3.2)$$

Advantages of GANs include the trainability using only backpropagation and dropout methods, as well as not being dependent on Markov chains or approximate inference to find the generator distribution  $p_g(x)$  [9]. However, training GANs is still hard, as  $G$  and  $D$  must be synchronized well, and no explicit representation of  $p_g(x)$  is given.

### 3.2.1 Wasserstein GAN

Like mentioned before, learning Generative Adversarial models is often difficult, due to the fact that training is done using gradient descent methods. These usually focus on trying to find a local minimum of a loss function, rather than a Nash equilibrium of a minimax game. Problems that may arise include [49, 50]:

- **Instability:** The model weights oscillate or diverge and destabilize
- **Mode Collapse:** The generator collapses and produces limited variation in its samples.

- Diminished Gradient: The discriminator learns too well, which lets the generator gradient vanish, resulting in no learning process
- Sensitive to Hyperparameter Selection: Wrong selection can directly lead to instability issues.

A starting point for improvement of GAN training is the objective function. In contrast to other network architectures, the GAN objective function already offers flexibility, like using the Jensen-Shannon divergence [9] or f-divergences [12]. Arjovsky et al. [14] motivate to use the Wasserstein or Earth Mover (EM) distance, due to its promising gradient properties and superior loss function convergence compared to similar objective functions:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y) \sim \gamma} [\|x - y\|] \quad (3.3)$$

where  $\Pi(\mathbb{P}_r, \mathbb{P}_g)$  is the set of all joint distributions  $\gamma(x, y)$  with marginals  $\mathbb{P}_r$  and  $\mathbb{P}_g$ . In general, the distance function can be seen as the minimum amount of “weight” or “mass”, that must be transported to turn  $\mathbb{P}_r$  into  $\mathbb{P}_g$ .

While equation 3.3 can not be computed directly, it can be approximated using the Kantorovich-Rubinstein duality

$$W(\mathbb{P}_r, \mathbb{P}_g) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim \mathbb{P}_r} [f(x)] - \mathbb{E}_{x \sim \mathbb{P}_g} [f(x)] \quad (3.4)$$

where  $\|f\|_L \leq 1$  denotes all the 1-Lipschitz functions. To maximize this equation while adhering to the 1-Lipschitz property of the solution, Arjovsky et al. introduce weight clipping. This clamps the parameter weights of the discriminator (or in this case the critic, as it is not trained to classify) to a small interval after the gradient step.

Similar to the GAN objective, the WGAN objective can also be represented as a minimax game:

$$\min_G \max_{\|f\|_L \leq 1} \mathbb{E}_{x \sim \mathbb{P}_r} [D(x)] - \mathbb{E}_{\tilde{x} \sim \mathbb{P}_g} [D(\tilde{x})] \quad (3.5)$$

The advantages of the Wasserstein GAN over the traditional GAN are twofold: While the original GAN loss function does not correlate with the quality of produced samples, the Wasserstein Loss does. Moreover, in contrast to GANs, the Wasserstein GAN exhibits improved stability and a minor dependency on the balance of the models during training. This could be seen in an example experiment, where traditional GANs experienced mode collapse while the WGAN did not.

#### 3.2.2 Wasserstein GAN with Gradient Penalty

Already in their work, Arjovsky et al. describe weight clipping as a “clearly terrible way” [14, p. 7] to enforce the Lipschitz constraint for the Wasserstein Loss. They mentioned experimenting with other variants like projecting the weights onto a sphere, but stucked to the original formulation as it already gave a good performance. This imperfection was picked up and analysed by Gulrajani et al. [15], which found two important flaws of the weight clipping approach:

- **Capacity underuse:** Using weight clipping biases the critic of the network towards learning much simpler functions and therefore not being able to represent or capture higher moments in the data distribution. The authors showed this by training on several toy distributions like the Swiss Roll dataset. During training, they kept the generator distribution really close to the ground truth and just trained the critic which in the end resulted in too simplistic data approximations.
- **Exploding and vanishing gradients:** The authors found out that the critic is prone to divergence if the clipping parameter  $c$  is not carefully chosen. An experiment on the Swiss Roll dataset resulted in exploding or vanishing gradients for the critic. Only in a certain value region, the network was able to converge and stabilize during training. The reason for that is that weight clipping pushes weights towards the extremes of the clipping range.

To solve these issues, Arjovsky et al. introduced a soft constraint into the Wasserstein Loss, using a gradient penalty to enforce the Lipschitz constraint:

$$L = \underbrace{\mathbb{E}_{\tilde{x} \sim P_g}[D(\tilde{x})] - \mathbb{E}_{x \sim P_r}[D(x)]}_{\text{Wasserstein Loss}} + \underbrace{\lambda \mathbb{E}_{\hat{x} \sim P_{\hat{x}}}[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2]}_{\text{Gradient Penalty}} \quad (3.6)$$

where  $P_{\hat{x}}$  is sampling uniformly between  $x$  and  $\tilde{x}$ . The reasoning behind the form of the gradient penalty is described in detail in their paper. Experiments show that the WGAN-GP outperforms weight clipping both with respect to training speed and sample quality, and further improves training stability of GANs.

### 3.3 Reinforcement Learning

A Markov Decision Process can be defined as tuple  $(S, A, P, R, \gamma, T, S_0)$ , where

- $S$  is a finite set of states
- $A$  is a finite set of actions
- $P(s'|s, a)$  is a state transition probability distribution over next states given the current state  $s$  and a chosen action  $a$
- $R(s, a)$  is a reward function, which returns the reward one gets when taking action  $a$  in state  $s$
- $\gamma \in [0, 1)$  is a discount factor
- $T$  is the final time step
- $S_0$  is a finite set of start states

It satisfies the markov property, i.e. that future states  $s_{t+1}, s_{t+2}, \dots$  of the process depend only on the present state  $s_t$  it is currently in. If  $T < \infty$ , the Markov Decision Process (MDP) is called a finite-horizon MDP, otherwise it is called an infinite-horizon MDP.

In Reinforcement Learning [51], an agent is interacting with a given environment (for example an MDP). In each timestep, its policy  $\pi : S \rightarrow A$  determines thereby, which action is taken next. Afterwards, the agent receives a reward  $r(s, a)$  and lands in the next state  $s'$ , which is sampled from the state transition probability distribution  $P(s'|s, a)$ . Calculating the discounted sum of the rewards is then called return:  $G_t = \sum_{i=t}^{\infty} \gamma^{i-t} r_i$ . The goal of the agent is to maximize its expected return over the trajectory:  $\mathbb{E}_{s_0}[G_0|s_0]$ . Each state-action pair  $s_t, a_t$  also has an expected return, the value function:  $Q^\pi(s_t, a_t) = \mathbb{E}[G_t|s_t, a_t]$ .

An optimal policy  $\pi^*$  is a policy, where  $Q^{\pi^*}(s, a) \geq Q^\pi(s, a)$  for every  $s \in S, a \in A$ , i.e. that the expected return for each state-action pair is at least as high as for any other policy. The same Q-values are shared across all optimal policies of an environment and are called the optimal Q-function, which can be calculated as

$$Q^*(s_t, a_t) = r_t + \mathbb{E}_{\rho_{\pi^*}} \left[ \sum_{l=1}^{\infty} \gamma^l r_{t+l} \right] \quad (3.7)$$

where  $\rho_{\pi^*}$  means that we are following policy  $\pi^*$ .

The learning objective of traditional RL can in the end be summarized as

$$\pi^* = \arg \max_{\pi} \sum_{t=0}^{\infty} \mathbb{E}_{\rho_{\pi}} [\gamma^t r_t] \quad (3.8)$$

#### 3.3.1 Maximum Entropy Reinforcement Learning

The notion of Maximum Entropy Reinforcement Learning will be useful in Section 4.6 when introducing our algorithm used for IRL. MaxEnt RL is an extension to the traditional RL framework based on a simple principle, proposed already in 1991 by Williams et al. [52]. Instead of trying to find Q-values that only maximize the expected return, we also want the resulting policy to have a high entropy

$$\mathcal{H}(\pi) = - \sum_{\pi_a \in \pi} \pi_a \ln \pi_a \quad (3.9)$$

with  $\pi_a \in \pi$  being a short notation for all the different action probabilities of the policy  $\pi$ .

In general words, this means, that we chose a policy, that is less deterministic or more uncertain about which actions to pick next. Our optimal value function then changes to

$$Q_{\text{soft}}^*(s_t, a_t) = r(s_t, a_t) + \mathbb{E}_{\rho_{\pi^*}} \left[ \sum_{l=1}^{\infty} \gamma^l (r_{t+l} + \tau \mathcal{H}(\pi^*)) \right] \quad (3.10)$$

where  $\tau$  is a temperature parameter that defines the weight of the entropy condition.

The learning objective of MaxEnt RL can finally be summarized as

$$\pi_{\text{MaxEnt RL}}^* = \arg \max_{\pi} \sum_{t=0}^{\infty} \rho_{\pi} [\gamma^t (r_t + \tau \mathcal{H}(\pi))] \quad (3.11)$$

### 3.3.2 SMDP

To include the possibility of temporal abstraction into an MDP, one can make use of Semi-Markov Decision Processes (SMDPs). A discrete-time SMDP is a simple generalization of the MDP. In contrast to traditional MDPs, actions taken in an SMDP can take a variable amount of time. This way, taking an action  $a$  in a state  $s$  can be seen as executing a subroutine which takes one or more time steps to complete. If we change the state transition probability distribution and the reward function to map to the joint distribution of  $s'$  and  $N$ , the Bellman equation for a Semi-Markov Decision Process and a fixed policy is given as

$$V^\pi(s) = \sum_{s',N} P(s', N|s, \pi(s)) [R(s', N|s, \pi(s)) + \gamma^N V^\pi(s')] \quad (3.12)$$

Hierarchical frameworks later explained in this chapter make use of SMDPs, by representing intermediate hierarchy layers as subroutines.

## 3.4 Inverse Reinforcement Learning

---

**Algorithm 1:** Feature matching IRL pseudocode.

---

**Input:** Expert trajectories  $\mathcal{D} = \{\tau_i\}_{i=1}^N$   
Initialize the reward function  $r_\omega$  and policy  $\pi_\theta$   
Calculate expert visitation frequency  $\mu_{\mathcal{D}}$   
**repeat**  
    Evaluate the state-action visitation frequency  $\mu_L$  of the current policy  $\pi_\theta$   
    Evaluate the objective function  $\mathcal{L}$  and its derivative  $\nabla_\omega \mathcal{L}$  by comparing  $\mu_L$  and  $\mu_{\mathcal{D}}$   
    Update the reward function parameter  $\omega$   
    Update the policy parameter  $\theta$  with a RL method  
**until**  
**return** optimized policy  $\pi_\theta$  and reward function  $r_\omega$

---

Traditional RL tries to learn an optimal policy of an environment, given feedback in form of rewards. Nonetheless, not always is a reward function present, and hand-crafted reward functions could bias the behaviour of the agent. This is especially true when the policy is based on demonstrated behaviour. To overcome this problem, Inverse Reinforcement Learning [53] first learns the underlying reward structure given expert demonstration, and then computes an optimal policy on top of it.

A common approach for solving IRL is state-action visitation frequency matching. In this case, we want the state-visitation frequency of the learned policy to be the same as for the expert:

$$\mathbb{E} \left[ \sum_{t=0}^T \gamma^t \phi(s_t) \middle| \pi_L \right] = \mathbb{E} \left[ \sum_{t=0}^T \gamma^t \phi(s_t) \middle| \pi_E \right] \quad (3.13)$$

where  $\phi(s_t)$  are the features at timestep  $t$ ,  $\gamma$  is the discount,  $\pi_L, \pi_E$  are the learner and expert policies respectively, and the reward function is given by  $r(x) = \omega^T \phi(s)$ .

A pseudocode for state-action visitation frequency matching is presented in Algorithm 1: Given trajectories of an expert, we update our reward function iteratively by comparing the state-action visitation frequency of the current learner policy and demonstrator. Afterwards, traditionally a model-based RL algorithm is used to compute the optimal policy for the updated reward grid. The process is repeated until the learner policy converges and the final policy is returned.

Instead of model-based RL algorithms for computing the policy of the reward grid, also model-free methods can be used. Examples approaches are GAIL [20], AIRL [7] or oIRL [8] which were presented in Chapter 2. However, due to the increased operational time of model-free methods, this adds a lot of overhead to the algorithm.

#### 3.4.1 Maximum Entropy Inverse Reinforcement Learning

An issue of the IRL problem statement is that it is ill-posed, as a policy can be optimal for multiple different reward functions. Consider the example, where all rewards are 0. Then every policy you choose is an optimal policy of the environment. Maximum Entropy Inverse Reinforcement Learning [6] is an extension of feature expectation matching, that tackles the issue by choosing a policy that maximizes the entropy over a trajectory, similar to MaxEnt RL:

$$H(p(\tau)) = \sum_{\tau} p(\tau) \ln \frac{1}{p(\tau)} \quad (3.14)$$

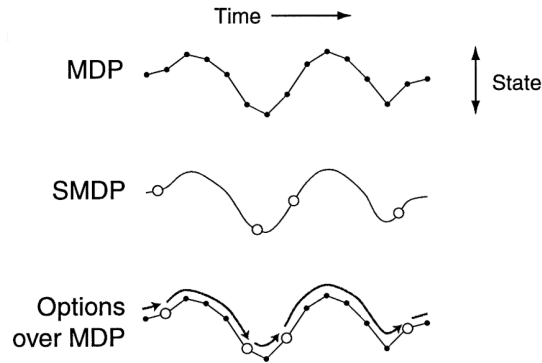
Additionally, the following constraints hold

$$\begin{aligned} \mathbb{E}_{\pi_L}[\phi(\tau)] &= \mathbb{E}_{\pi_E}[\phi(\tau)] \\ \sum_{\tau} p(\tau) &= 1, \forall \tau : p(\tau) > 0 \end{aligned} \quad (3.15)$$

where the upper part tries to match the features of the learner with the experts, and the lower part is the necessary condition for a probability distribution over trajectories.

### 3.5 Hierarchical Reinforcement Learning

The support of principles like state abstraction and temporal abstraction are a desired property of algorithms that try to solve long-term tasks in complex environments. State abstraction can help to narrow down the state space to a reasonable degree, while temporal abstraction can help with issues that occur for long-term objectives like the problem of sparse rewards. Methods can be based roughly on three different basic approaches.



**Figure 3.3:** Connection between an MDP, SMDP and the option framework.

The Option Framework [51] is a hierarchical framework, which uses temporal abstraction for learning. A high level selection policy  $\pi_\Omega$  is trained to select subpolicies  $\pi_\omega$ , called intra-options. Each option itself is represented as a tuple  $(I_\omega, \pi_\omega, \beta_\omega)$ , where  $\pi_\omega$  is the option policy,  $I_\omega \in S$  is the set of start states and  $\beta_\omega : S \rightarrow [0, 1]$  is the termination function. An option executes until the termination function tells it to terminate in a state. After that, the selection policy selects a new option to run. Like visible in Figure 3.3, the selection policy can be interpreted as interacting with an SMDP, where the intra-option policies determine the amount of time a higher-level action takes to be executed.

The MaxQ framework [54] can be used to compute and execute a hierarchical structured policy on an MDP. Therefore, a task hierarchy is defined, which follows five different rules, and the MaxQ Value Function Decomposition is used to learn the optimal value function of the task. In the course of this, the MaxQ decomposition first splits a long-term task into several subtasks, that each contain their different start and terminal state set. The subtasks can be seen as options with a termination probability of 0 when  $s \notin T$  and 1 when  $s \in T$ . However, the hierarchy can be of arbitrary depth, and subtasks in the middle layer do not execute primitive actions, but rather schedule subtasks on their level below. In the end, the learned policy can be executed using a stack to track the currently arising hierarchy. The MaxQ framework thereby allows support of three desirable principles: Temporal abstraction, state abstraction and subtask sharing.

Hierarchical Abstract Machines (HAMs) are a framework introduced by Parr et al. [55] in 1998. HAMs consist of multiple machines, which themselves have states of four different types: Action states, that execute an action in the environment; Call states, that can call a different machine to be executed as subroutine; Choice states, which stochastically select a new state for the machine; and a Stop action, which interrupts the machine and gives control to its caller. A HAM is then defined as an initial start machine, which can execute and call other machines on its purpose of achieving a desired objective. However, as creating a HAM is often laborious and the execution is hardly generalizable for different tasks or environments, this approach is not as popular as the previously mentioned.





## 4 Methods

In this chapter, we present the four different variants of the VRED that we use for short-term human motion prediction. Afterwards, we explain the structure of our IRL algorithm. In the end, we present the heuristics that map from an abstract representation of the state back to inputs that are comprehensible for the neural networks.

### 4.1 Vanilla VRED Implementation

Our basic network structure builds on the Position-Velocity Recurrent Encoder-Decoder network presented in Section 3.1. The state for each frame is thereby modeled as a 66 dimensional vector, including the base position and rotation of the human, as well as 60 joint angles given as exponential maps. These are then converted to quaternions in the loss function.

Similar to [40] we do make the following adjustments to the model:

- We do not use the base position as input to the VRED, to make the model independent of the world coordinates.
- For data augmentation and to avoid overfitting, we randomized the base rotation (and later also the goal input) when training the model.
- We do not only use a quaternion loss for the network but split the loss into a base position loss and quaternion loss for the joint angles.

The angle loss  $L_{\text{angles}}$  is the mean of a combination of multiple quaternion losses, which we mimiced from Kratzer et al. [40]. The position loss and total loss are given as:

$$\begin{aligned} L_{\text{base}} &= \sum \|p' - p\|^2 \\ L &= L_{\text{base}} + L_{\text{angles}} \end{aligned} \tag{4.1}$$

However, in contrast to [40] our VRED model consists of only 200 hidden units per GRU cells. Moreover, we just chose 2 GRU cells for our prediction and the source length as well as the prediction length are set to constant to 30 frames, which equals one second. We found that using an Adam Optimizer with a learning rate of 0.001 and a batch size of 8 works best for training.

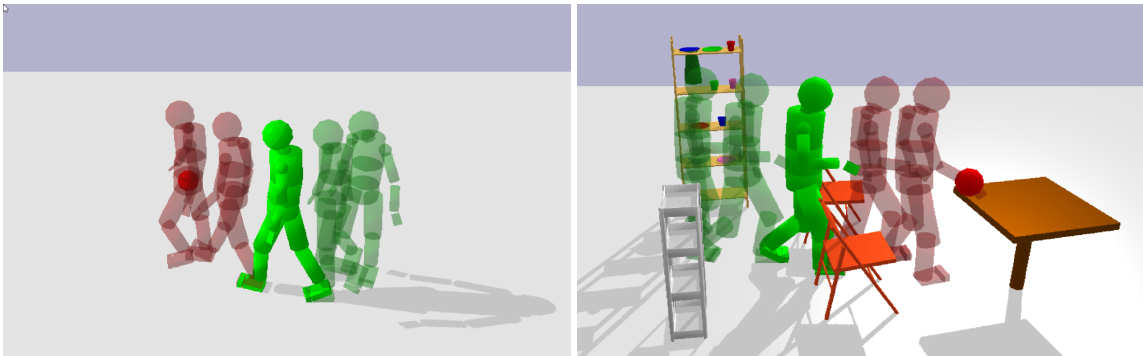
## 4.2 Goal-Conditioned VRED

To be able to use the VRED as a subpolicy in a RL or IRL framework, we do not only need a sequence-to-sequence mapping, but also need it to be goal-conditioned. We do this by adding a three-dimensional position  $g \in \mathbb{R}^3$  to the input of the network. This way, after training, different trajectories can be generated by varying the goal input manually. Stringing together multiple subgoals can be used for sequential long-term motion predictions of the human.

However, the best goal description is often depending on the task the human wants to achieve. Grasping or placing an object should probably have the end effector position of the hand or wrist as goal, while for walking, the base translation of the human should make more sense. Therefore, we used two different goal descriptions for both the available tasks. These were utilized respectively in their own distinct dataset (see details chapter 5.1), one collected from walking demonstrations, and one from pick and place segments.

Given the input human trajectory  $X = \{x_1, x_2, \dots, x_n\}$  until time  $n$  as well as the ground truth trajectory  $Y = \{x_{n+1}, x_{n+2}, \dots, x_{n+m}\}$  from timestep  $n+1$  to  $n+m$ , we define the *base goal*  $g_{base} = \phi_{base}(x_{n+m})$  as the position of the base joint for the last frame of the ground truth. Similar to that, we define the *wrist goal*  $g_{wrist} = \phi_{wrist}(x_{n+m})$  as the position of the right wrist joint for the last frame of the ground truth, where  $\phi_{wrist}$  is the forward kinematic map of the wrist end effector. We use the base goal description for the walking dataset and the wrist goal description for the pick and place dataset.

An example trajectory for both datasets can be seen in Figure 4.1. Five different frames of the original 60 frames are rendered for visibility. The input  $X$  is given as transparent green, with the last input frame showing as opaque pose. The ground truth is presented as transparent red. The goal position is drawn as small red sphere.



**Figure 4.1:** Goal setup for the walking task (left) and the hierarchical task (right)

Figure 4.2 shows the modified network structure after adding the goal to the input. Using a goal input in addition to the joint positions and velocities is a similar way of using a position embedding like in the original VRED paper [2]. It contributes to differentiating similar poses with a notion of time, which results in more discriminative predictions. An

example could be that when the human is already near the goal, it will try to decrease its joint velocities. In contrast to that it will probably react differently when being still far away from it in a similar pose. This way, the human does not only know where to go but also how long he probably needs to go there.

As we want the goal to be relative to the human and independent of world coordinates, some more steps are needed for implementation. Given the calculated absolute goal  $g_{abs} = g_{base}$  or  $g_{abs} = g_{wrist}$  for a trajectory, we calculate for all input frames  $x_t \in X$  the relative position  $g_t$  by using the inverse transform of the human at that time frame  $H_t^{-1}$ . This way, we exclude any type of absolute position information and have it transformed relative to the human:

$$g_t = H_t^{-1} \mathbf{g}_{abs} \tag{4.2}$$

Here,  $\mathbf{g}_{abs}$  is the homogeneous coordinate representation of  $g_{abs}$ .

In the decoder part of the network, we need to recalculate the relative goal in each time step (see T operator in Figure 4.2). This is done in the same way as in Equation 4.2 using the new calculated inverse human transform of the current time step

$$g_{m+t+1} = H_{n+t}^{-1} \mathbf{g}_{abs} \tag{4.3}$$

where  $H_{n+t}$  is the inverse human transform at time step  $n + t$ . We then plug in the relative goal position as input for the next timestep, together with the pose and its velocities.

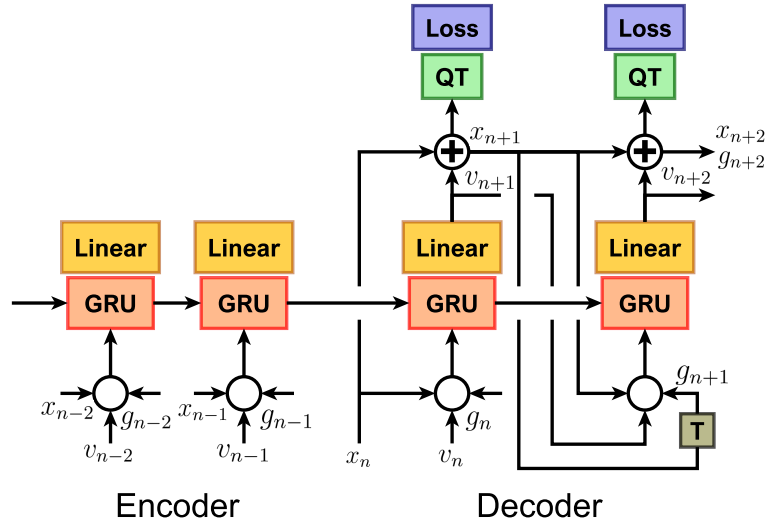


Figure 4.2: Structure of the VRED with the goal input added.

### 4.2.1 Modifying the Goal Input

To analyze the generalizability of the goal input of the network, we added the possibility to modify it before feeding it to the model. For that, we implemented three different methods that we experimented with:

- *Adding Gaussian noise*: Here, we use the original goal, but add some Gaussian noise with mean 0 and variable standard deviation to it. For simplicity, we keep the z-axis of the goal fixed.
- *Taking another random relative goal*: Instead of using the original goal of the sample, we choose a random relative goal from a different sample of the data and use it for the prediction.
- *Taking a random goal*: This variant takes a completely random position as goal, sampled from a Gaussian distribution with the mean being the human base position and a varying standard deviation. Similar to the first method we keep the z value constant at 1.

### 4.2.2 Segmentation

We also experimented with segmenting the recorded trajectories into pick and place trajectories. This way, we could train a VRED that only uses goals that relate to object positions or their target destinations. For that, we used already generated segmentation files from a previous work [56]. The data is then split, so that the start of each extracted trajectory is exactly  $m + n$  frames before the object is grasped or placed onto a surface. Just like for the other VREDs we used 30 frames for the input and 30 frames for the prediction.

## 4.3 Goal-Conditioned VRED with Goal Loss

Using a relative goal as additional input to the network helps to discover the intention behind the action of the human. Still, the network is not penalized for not being able to get to the goal, so that until now it just serves as extra information. In addition to that, the network has no information what the goal input represents.

To increase the probability of reaching the intended goal and to minimize the final distance to it, we add another term to our loss function which regularizes the squared loss of the difference between the absolute ground truth goal position and the reached absolute goal position in the last prediction frame. This means that for the *base loss*, it compares the base position of the human in the last prediction frame and the base position of the human in the last frame of the ground truth. Similar is the calculation for the wrist loss using the wrist positions, however, instead of directly reading off the values from the pose vectors, we need to first calculate the wrist positions of both using forward kinematics.

$$\begin{aligned} L_{goal}^{base} &= \|g_{base} - \phi_{base}(\tilde{x}_{n+m})\|^2 \\ L_{goal}^{wrist} &= \|g_{wrist} - \phi_{wrist}(\tilde{x}_{n+m})\|^2 \end{aligned} \tag{4.4}$$

where  $g$  is the goal,  $\phi_{base}$  or  $\phi_{wrist}$  are the forward kinematic maps as defined in Section 4.2 and  $\tilde{x}_{n+m}$  is the last prediction of the network. The linked loss function for the VRED

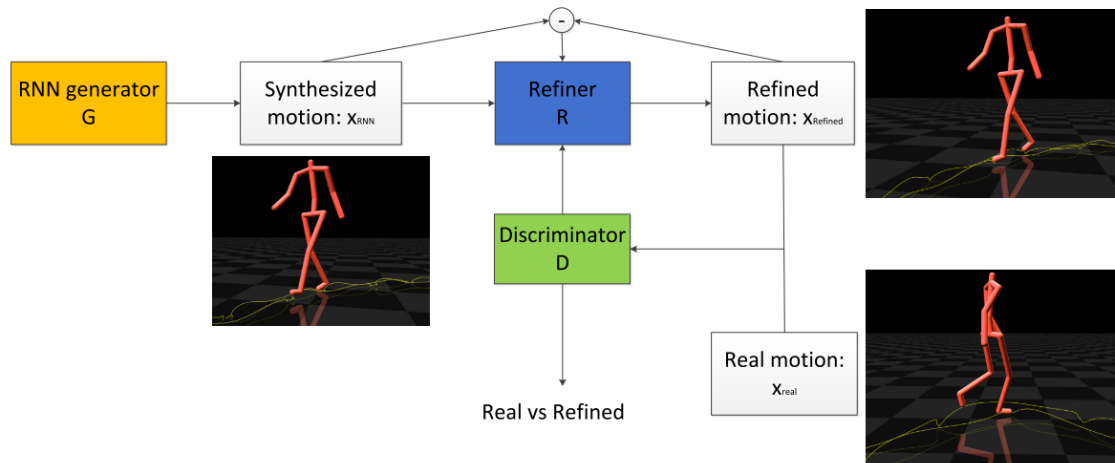
is then given as the combination of Equation 4.4 and one of 4.1, where for the walking dataset we use  $L_{goal} = L_{goal}^{base}$  and for the pick and place dataset we use  $L_{goal} = L_{goal}^{wrist}$ :

$$\begin{aligned} L_{walk} &= L_{base} + L_{angles} + \alpha L_{goal}^{base} \\ L_{pick\&place} &= L_{base} + L_{angles} + \alpha L_{goal}^{wrist} \end{aligned} \quad (4.5)$$

The hyperparameter  $\alpha$  is used to balance the weight of the regularization that is induced by the goal loss.

## 4.4 Adversarial VRED

Generative Adversarial Nets are often used in image generation when it comes to creating real or human looking faces or pictures [57, 58]. In contrast to similar approaches they achieve high fidelity and realism in their generated images. As one of the desired objectives of human motion prediction is to maximize the closeness to reality of the predicted poses, we wanted to extend our VRED with an adversarial structure. We tried multiple compositions and types of adversarial networks, a standard GAN [9] and a WGAN-GP [14], but finally decided for a network similar to [27]. The line of thought being that it can already use a trained Recurrent Neural Network, which reduces some of the training difficulties of GANs [49, 50].



**Figure 4.3:** Structure of the Adversarial VRED network (taken from Wang et al. [27]).

Figure 4.3 shows the pipeline of the adversarial system. At the start, fake predictions are generated from samples of the dataset using the pretrained VRED. The synthesized motion from the generator is then fed into the refiner network, that outputs a modified version of it. In the final step, the critic tries to distinguish between the refined motion and sampled data from the real dataset.

The generator network is a pretrained VRED for the walking or pick and place dataset, depending on the task. Its weights are frozen during training, so that the refiner is solely responsible for adjusting the synthesized motion. After encoding the input sequence, we additionally add some Gaussian noise with standard deviation  $\sigma = 0.05$  to the hidden state of the VRED to generate variations in the output. We made sure that the noise did not distort the trajectories too much by visualizing different values of  $\sigma$ , moreover, the same method is also used in the work of Barsoum et al. [26].

For adversarial neural networks, the discriminator structure is often based on the generator (see [27, 4] as example), as it balances the advantages of both models and reduces the chance of one model overwhelming its counterpart. For that reason, we chose to shape the discriminator similar to the VRED. As the hidden state of it encodes the input into a meaningful latent representation, we can try to use this compact information to distinguish between real and fake sample. We do this by adding a final dense layer to the critic, which transforms the hidden state to a single logit, the critic score.

The refiner is built more simple. It consists of a dense layer with 128 units, followed by an LSTM layer with 128 units to encode temporal information. In the end, another dense layer with linear activation is added to normalize the output dimension to the number of joints of the human. We use a skip connection from the input of the refiner to its output, so that the refiner network encodes just the desired change in human motion.

In the original paper [27], the discriminator loss  $L_D$  is defined as

$$L_D = -\log(1 - D(R(x_{RNN}))) - \log D(x_{real}) \quad (4.6)$$

where  $D$  is the discriminator,  $R$  the refiner,  $x_{RNN}$  the synthesized motion from the RNN generator and  $x_{real}$  the real data samples. Wang et al. thereby use the following refiner loss  $L_R$ :

$$L_R = -\log(D(R(x_{RNN}))) + \beta \|\text{root}(x_{RNN}) - \text{root}(R(x_{RNN}))\|^2 \quad (4.7)$$

Here,  $\text{root}(x)$  is the root position of the pose  $x$ , which resembles our base position  $\phi_{base}(x)$ . The second term is an  $l^2$  regularization term that tries to keep the base position of the human close to its refined motion in each frame, weighted by the parameter  $\beta$ . In this way, the refined motion does not stray away too much from the synthesized one.

However, as previously mentioned, traditional GANs have a lot of disadvantages to the WGAN-GP when it comes to training stability. Therefore, we wanted to incorporate a Wasserstein Loss to the network. Using equation x and y, we then get for the critic loss  $L_D$ :

$$L_D = D(x_{RNN}) - D(x_{real}) + \lambda \mathbb{E}_{\tilde{x} \sim \mathbb{P}_{\tilde{x}}} [(\|\nabla_{\tilde{x}} D(\tilde{x})\|_2 - 1)^2] \quad (4.8)$$

where  $\tilde{x}$  interpolates between  $x_{RNN}$  and  $x_{real}$ , similar as explained in Section 3.2.2, and  $\lambda$  is the hyperparameter of the gradient penalty. We get for the refiner loss  $L_R$ :

$$L_R = -D(x_{RNN}) + \beta \|\text{root}(x_{RNN}) - \text{root}(R(x_{RNN}))\|^2 \quad (4.9)$$

## 4.5 Symbolic State Representation

We not solely wanted to test the trained models on different datasets, but also combine them for solving a higher-level task. We decided to learn a policy that is able to solve to setup the table for one person in a simple environment. Demonstrations for this task were already available in a previously collected dataset. Nevertheless, the full state-action space of the task is really complex. Therefore, we simplified the space to a symbolic representation, to be able to use a tabular variant of MaxEnt IRL to retrieve our policy.

The environment of the dataset consists of three different stationary objects and 10 items (4 cups, 4 plates, 1 jug, 1 bowl) that can be moved around the room. More information about it can be found in Section 5.1. We use a discrete symbolic state representation by saving the combinations of items and stationary objects, as well as the current position of the human. The full state is then given as

(# cupsAtTable, # cupsAtWhiteShelf, # cupsAtBeigeShelf,  
# platesAtTable, # platesAtWhiteShelf,  
# jugBowlAtTable, # jugBowlAtWhiteShelf, # jugBowlAtBeigeShelf, humanPos)

where # *jugBowlAtTable* means the amount of jugs and bowls currently at the table (in this case 0, 1, or 2). When the human is carrying an item, the item does not count for any location. The human positions are organized into an area at the table (0), the white shelf (1), the beige shelf (2), or at the start of the demonstration it can also be away from all the former (3). The total amount of state combinations possible with deduction of invalid states are 15300.

Similarly to the state space we simplified the continuous action space of the demonstrations by choosing seven simple actions. By that we ended up with the following discrete action space:

(*walkToTable*, *walkToWShelf*, *walkToBShelf*, *graspCup*, *graspPlate*, *graspJugBowl*, *Place*)

We distinguish between picking up certain item types while having just a single action for placing. The reason is, that we can directly extract which object we are holding by having a look at our state space, and therefore do not need an individual place action for each item type. Choosing *Place* as action when not holding an object, or a *Grasp* action when holding an object will not change the state. Similarly, selecting a walking action when already at a certain location reroutes to the same state.

A simple example for a demonstration of the dataset can be seen in Table 4.1. The end state is reached when exactly one item of each item type is present at the table and the human has no object in its hand.

	Trajectory
Start State	[2, 1, 1, 2, 0, 2, 1, 1, 0, 0]
Actions	Pick Up Plate Go to Beige Shelf Place Go to Table Pick Up Cup Go to Beige Shelf Place
End State	[1, 1, 2, 1, 0, 3, 1, 1, 0, 2]

**Table 4.1:** Example for a perfect imitation using the learned policy from our MaxEnt IRL algorithm

## 4.6 Maximum Entropy IRL Framework

---

**Algorithm 2:** Tabular MaxEnt IRL pseudocode.

---

**Input:** Expert trajectories  $\mathcal{D} = \{(s_i, a_i)\}_{i=1}^N$   
Initialize the reward function  $r_\omega$  and Q-values  $Q_{r_\omega}^*$   
Calculate expert visitation frequency  $\mu_{\mathcal{D}}$   
**repeat**  
    Perform tabular soft Q-Iteration to get optimal Q-values  $Q_{r_\omega}^*$  of the current reward function  $r_\omega$   
    Compute learner visitation frequency  $\mu_L$  from  $Q_{r_\omega}^*$   
    Evaluate the objective function  $\mathcal{L}$  and its derivative  $\nabla_\omega \mathcal{L}$  by comparing  $\mu_L$  and  $\mu_{\mathcal{D}}$   
    Update the reward function parameter  $\omega$   
**until**  
**return** optimized Q-values  $Q_{r_\omega}^*$  and reward function  $r_\omega$

---

To learn a reward function and policy from the given demonstrations, we use Tabular MaxEnt IRL on our symbolic environment representation. A pseudocode for it is given in Algorithm 2: Similarly to feature matching (see 3.3.1), we extract the reward function by learning from the comparison of the visitation frequency of the expert and the current learner in each iteration step.

After initialising the reward function  $r_\omega$  and calculating the expert visitation frequency  $\mu_{\mathcal{D}}$ , soft Q-Iteration is used to calculate optimal Q-values  $Q_{r_\omega}^*$  w.r.t. the current reward grid. Thereby we use the previous Q-values as warmstart. Soft Q-Iteration is the equivalent to standard Q-Iteration for Soft RL. Afterwards, the learner visitation frequency  $\mu_L$  is computed by Expected Edge Frequency Calculation, like described in the MaxEnt IRL paper [6]. In the end we compare  $\mu_L$  with  $\mu_{\mathcal{D}}$  and update our reward function.



To calculate the learner visitation frequency and extract the final behavior of the agent, we need to compute a policy from given Q-values that follows the maximum entropy principle. For that, we use the following equation, which can be derived from Soft RL principles:

$$\pi_{\text{MaxEnt}}^*(a_t|s_t) = \exp\left(\frac{1}{\tau}(Q_{r_\omega}^*(s_t, a_t) - V_{r_\omega}^*(s_t))\right) \quad (4.10)$$

where the optimal value function can be computed by

$$V_{r_\omega}^* = \tau \log \sum_{a'} \exp\left(\frac{1}{\tau} Q_{r_\theta}^*(s, a')\right) \quad (4.11)$$

In the equation,  $\tau$  is the hyperparameter of the entropy of the Soft RL objective (see Section 3.3.1), and  $a'$  are all the possible actions the agent can take in state  $s$ .

## 4.7 Heuristics for Trajectory Visualization

The MaxEnt IRL framework from the previous section gives us a policy, that we can use on the simplified task space to solve the objective of setting up the table for one person. However, the actions used in the simplistic MDP (*WalkToX*, *PickUpX*, *Place*) can not be used as direct model input for the VRED. To transfer the action space to the three-dimensional goal space for the VREDs, we make use of multiple heuristics. Furthermore, we always track the pose of the human, as well as current positions of the objects in the environment.

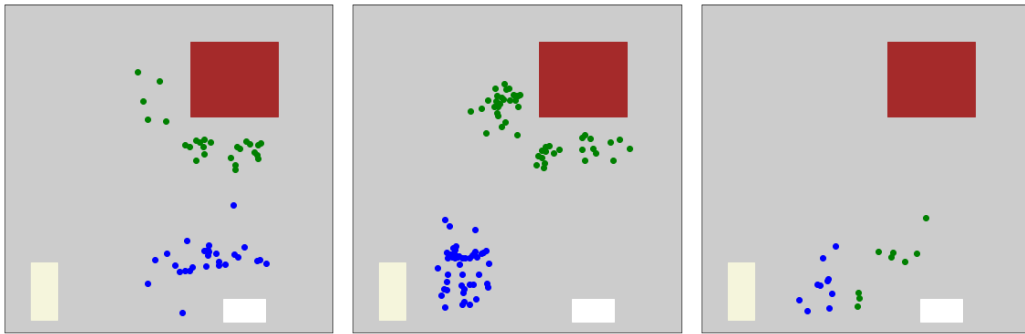
### 4.7.1 WalkToX

The trajectory of the walking motion of the human to a certain location is always depending on the first and last position of the human. Walking from the white shelf to the table, for example, often ends in a different end position than walking from the beige shelf to the table. To incorporate this into our visualization, we analyze the different start and end positions of the human for the six possible transitions 4.4. The shelves are marked as white and beige rectangles, while the table is presented in red. As example, the blue points in the left figure describe the positions the human ended up at when moving from the table to the white shelf.

When choosing an action like “WalkToTable”, the heuristic then randomly picks a point of the desired set as goal for the walking network. Furthermore, it includes an intermediate point between the current position of the human and the goal, so that the human has more time to walk to the destination.

### 4.7.2 PickUpX

For grasping an object when standing near a stationary object, we first collect the position of all the objects on the surface. Afterwards we chose and return the position of the closest object that matches the desired item type (e.g. the closest cup).

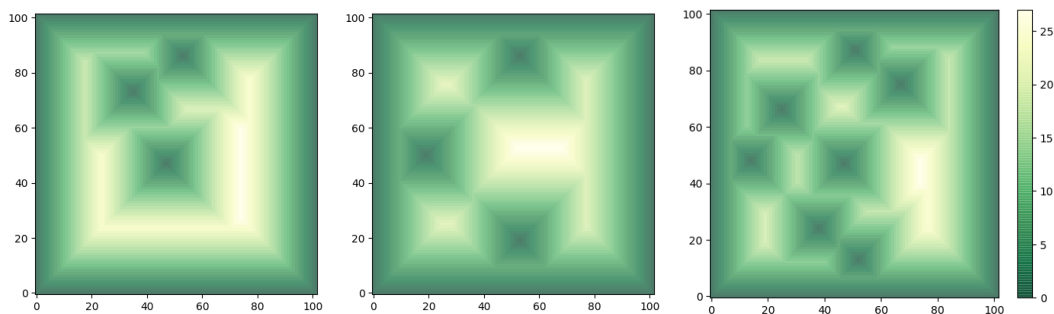


**Figure 4.4:** Scatter plots for the different human start and end positions when walking from one stationary object to another

### 4.7.3 Place

Placing an object should find a location on the stationary object that is far away from other items on the surface, as well as from the border of the table. To realize this, we calculate a Signed Distance Field (SDF) for each surface, that is updated when an object is placed or taken from it. For simplicity, we do not distinguish between sizes of objects (cups and plates) and grow the SDF by setting the distance to horizontal, vertical and diagonal neighbors to 1.

When a new position for placing has to be retrieved, we first collect all SDFs of the current location, for example both the surfaces of the white shelf. Afterwards, we take the maximum value of the computed SDF grids, and transform it to 3D coordinates. For the table, we adapt the heuristic by a small amount: As plates and cups are usually placed in certain locations when setting up a table, we just chose one of four defined positions for placing (or less if an object is already there). The SDF is then solely used for the bowl and the jug, as analysis showed that they take on relatively random table positions throughout the dataset. Three SDF examples can be seen in Figure 4.5.



**Figure 4.5:** Different examples of calculated SDFs of the demonstration data for the table.

The first plot shows a situation, where the table is already set up for a single person (one cup, plate and jug on the table). In the second plot, just three plates are located at the

table. Finally, in the third plot the SDF presents an example where the table is set up for three persons.



## 5 Experiments

In this chapter, we present our performed experiments. First, a qualitative analysis is done, that compares example trajectories of the human when adding the goal condition and goal loss to the VRED network. Afterwards, multiple quantitative measures of the different network structures are compared. Furthermore, we investigate the performance of our IRL algorithm on the symbolic state representation of the task. Finally, we show the results of combining both datasets to a human trajectory prediction.

### 5.1 Dataset

We trained our different VRED models on a custom dataset of the university [56]. Data generation was performed on a 4x4 meter area. The dataset includes 90 minutes of walking distributed over multiple recording sessions, where the human is tracked moving at a constant pace through the room. The other part of the dataset consists of pick and place trajectories, where the subject is interacting with the environment to solve a higher-level task [59]. The objective is achieved by grasping and placing items from or onto stationary objects using the right arm.

The setup of the environment for the picking and placing tasks can be seen in Figure 5.1. It consists of two shelves and a table, that can be used for placing objects onto. Furthermore two chairs are located near the center of the room to provide collision possibilities. A total of ten items can be picked up and placed, namely four cups, four plates, a white bowl and a green jug.

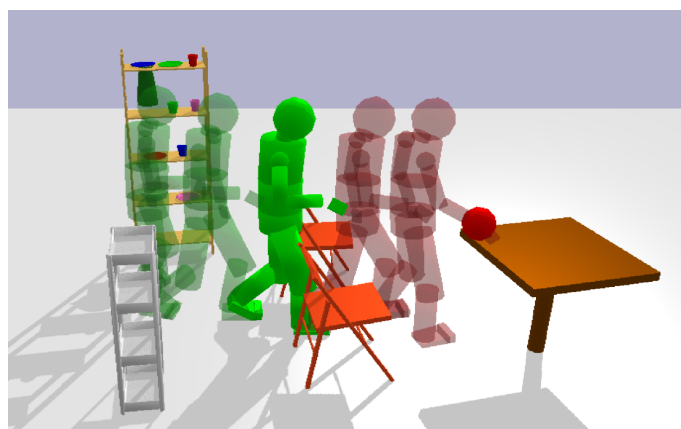


Figure 5.1: Setup for the hierarchical task

With this setup, 16 different tasks were performed, including setting up the table for one or more persons, clearing the table or placing objects by color/type on one of the stationary objects. Each of the tasks was then repeated up to 25 times in arbitrary order to generate a large dataset. The data recorded by the motion capture suits was transformed into a 66-dimensional human state representation. We further used downsampling of factor four to reduce the initial frame rate to 30 fps.

### 5.2 Training/Implementation Details

Training was done on the internal server of the university of Stuttgart, as well as partially on the own machine of the author. GPU training was supported but did not give a significant performance boost, which is why most models were trained on CPU. Tensorflow was used as backend together with the integrated Keras interface.

#### 5.2.1 Models

The main models trained on the dataset were the original VRED model, the VRED model with goal input, the VRED model with goal input and goal loss, and the adversarial network with the VRED as generator. All of the previously mentioned were trained for 300 epochs on the walking dataset and the pick and place dataset. The structure of the VREDs consisted of 2 hidden layers with 200 hidden states. For regularization of the goal (see 4.4) loss, we chose  $\alpha = 1$ , as it resulted in a good balance between minimizing the pose difference and the precision of the endeffector position of the last frame. The gradient penalty parameter for the Adversarial VRED was set to  $\lambda = 10$ , as proposed in the WGAN-GP paper [15] and the regularization of the model weights was kept at  $\beta = 1$  (see 4.9). Furthermore, we trained a VRED using the segmentation technique mentioned in Section 4.2.2, however due to the limited size of available grasp and place trajectories and its poor performance, we do not include it in the comparison Section (5.3). Also the Adversarial VRED did not improve the human trajectories in a visible qualitative way, however, we still add its result in the quantitative comparison.

#### 5.2.2 Baselines

To compare our models trained on our custom dataset, we made use of two commonly used baselines for human motion prediction [24], as well as a custom designed baseline for the base position loss:

- The *Zero Velocity* baseline takes the last step of the input sequence and uses it as prediction throughout the output.

- The *Moving Average* baseline is similar to the zero velocity baseline but takes the mean pose of the last  $n$  input frames, where  $n$  is the so called window parameter. In our evaluation we tried a Moving Average of  $n = 2$ , and  $n = 4$ .

To be able to evaluate our positional loss, we add another straightforward designed custom baseline. Similar to the *Moving Average* baseline, it takes the last  $n$  input frames, and computes the mean velocity of the human base joint. It then propagates this velocity through the prediction frames to get a final estimation of the base position. For simplicity, we call it here the *Constant Velocity* baseline. Note that it does not contain any additional pose information.

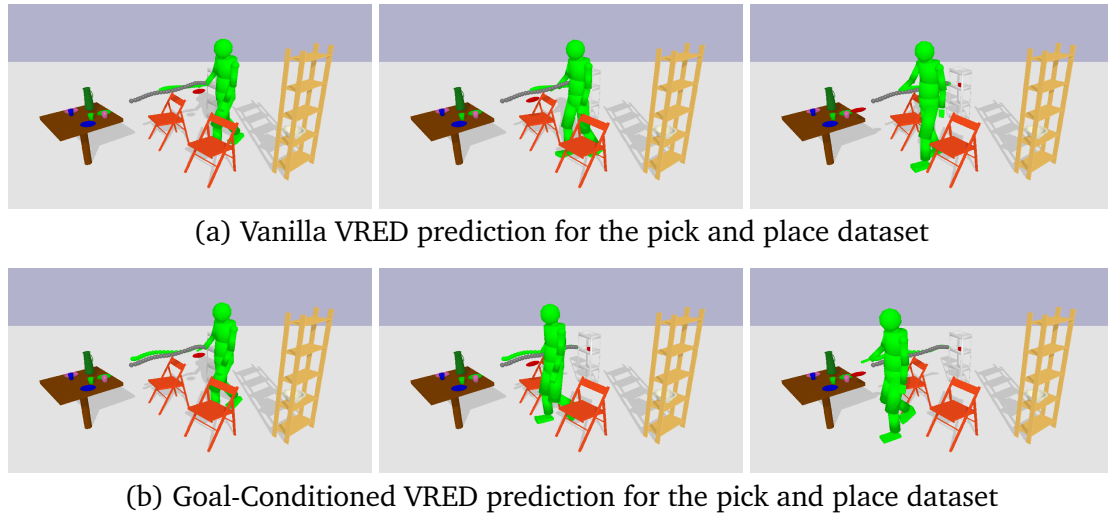
As the base goal and wrist goal could be calculated exactly without any more information using forward kinematics, we do not compare the goal loss and goal distance in the last prediction frame to any baseline.

## 5.3 Results/Evaluation

We first analyze the advantages of adding a goal input to the VRED model, and then compare it with the included goal regularization loss. In Subsection 5.3.3 we do a quantitative comparison of the different models, and lastly examine our hierarchical IRL framework.

### 5.3.1 Goal-Conditioned VRED

We compared the original VRED implementation with the VRED conditioned on a goal input on both the walking dataset, as well as the pick and place dataset. For that, we tried to chose example trajectories that do not favor any of the presented networks, but represent a qualitative average of the models. Two example predictions for the pick and place dataset can be seen in Figure 5.2. The ground truth endeffector position is drawn as a grey trajectory of spheres ranging from the first to the last frame of the ground truth. Respectively, the model predictions and their endeffector trajectories are presented in green. Visualized is only the prediction, not the input data and the prediction length is set to 30 frames or 1 second. On the top, you can see the motion predicted from the Vanilla VRED, on the bottom, the output of the Goal-Conditioned VRED is shown.

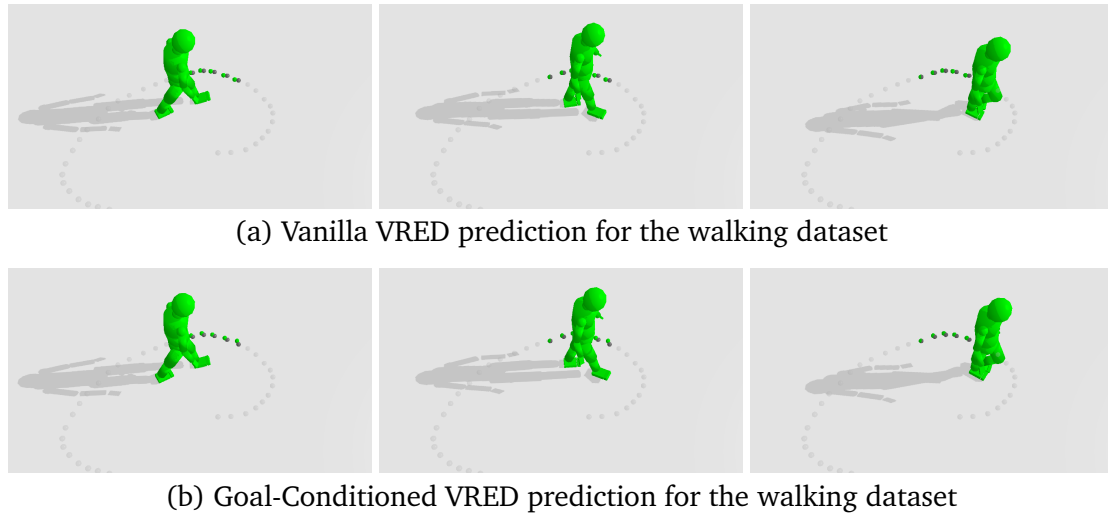


**Figure 5.2:** Difference between the Vanilla VRED and Goal-Conditioned VRED prediction on the pick and place dataset.

In the example, one can recognize that the original VRED has problems to anticipate the intention of the human. The target goal can just be guessed by deduction of the input positions and velocities and therefore is far worse than with goal input. Furthermore, the predicted trajectory ends at nearly half way of the ground truth with a large distance to the desired endeffector position. In contrast to that, adding a goal condition to the network increases the accuracy of the prediction by a large amount. The trajectory lengths are now similar, although the endeffector position of the last frame is still visibly distant to the ground truth.

A similar comparison can be made for the walking dataset. In Figure 5.3, we show an extract of the test set for the walking data. In the example, the human was told to move around the area in an arbitrary fashion, taking turns while keeping a steady speed. As the prediction differences are harder to see we visualize a top-down view of the human and just plot each fifth endeffector position. The history and future of the trajectory can be seen as nearly transparent grey dots. The ground truth endeffector trajectory is again shown in opaque grey, while the prediction and its human pose is presented in green.





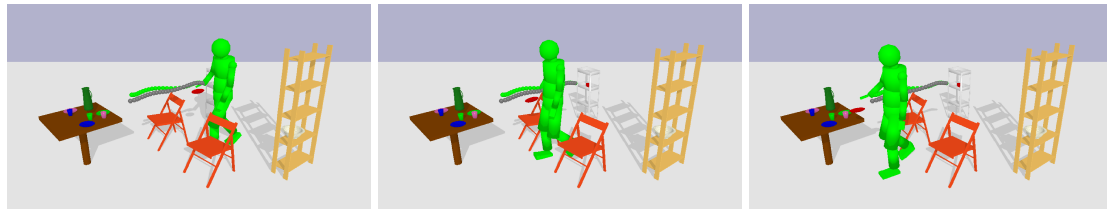
**Figure 5.3:** Difference between the Vanilla VRED and Goal-Conditioned VRED prediction on the pick and place dataset.

Also in the walking example, the prediction of the VRED is not able to reach the goal position completely, and seems to move too slow. Nevertheless it is considerably better than for the pick and place dataset, where the prediction ends in the middle of the ground truth trajectory. When adding the goal condition to the VRED the improvement is hardly visible and its better quality can just be recognized in the quantitative comparison (see 5.3.3). The predicted goal is now closer to the ground truth, but exhibits more bias sideways.

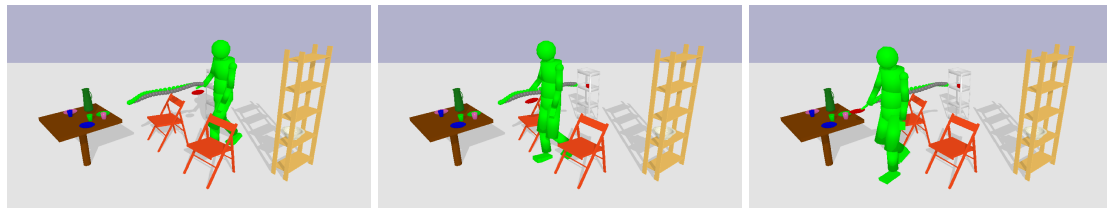
### 5.3.2 Goal-Conditioned VRED with Goal Loss

As discovered in the last subsection, adding the goal condition to the VRED is not sufficient enough to reach a certain endeffector position with a promising precision. The reason for that is that the network is not yet getting penalized for missing the goal in the prediction.

To further improve the predicted trajectory we added the goal loss to the network as described in Section 4.3. Figure 5.4 shows on top the Goal-Conditioned VRED prediction again, where the endeffector is not perfectly able to predict the last output frame. Adding the goal loss to the network loss function when training improves this significantly (5.4b). The added loss regularizes the VRED to minimize its distance of the last frame to the ground truth and therefore predicts a precise end position for the endeffector. The quantitative evaluation emphasizes that.



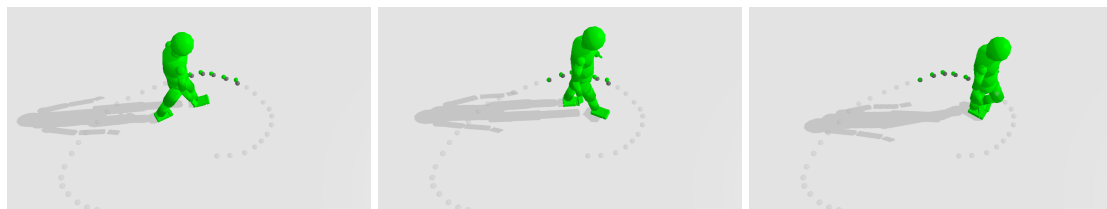
(a) Goal-Conditioned VRED prediction for the pick and place dataset



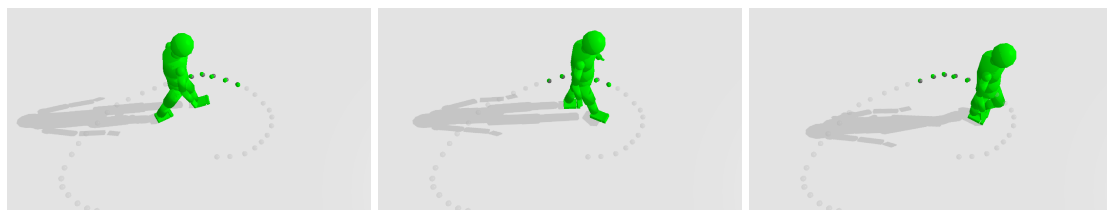
(b) Goal-Conditioned VRED + Goal Loss prediction for the pick and place dataset

**Figure 5.4:** Difference between the Goal-Conditioned VRED prediction without and with goal loss on the pick and place dataset.

Keeping our previous outcomes in mind, the improvement of the endeffector trajectory should also reflect in the walking dataset example. In fact, visualizing the trajectory of the previous example with and without goal loss yields a similar image: The last predicted endeffector position is really close to the ground truth and surpasses the output of the Goal-Conditioned VRED in precision by a large margin.



Goal-Conditioned VRED prediction for the walking dataset



Goal-Conditioned VRED + Goal Loss prediction for the walking dataset

**Figure 5.5:** Difference between the Goal-Conditioned VRED prediction without and with goal loss on the walking dataset.

### 5.3.3 Comparison

For the quantitative comparison of our models, we used the long-term error, which was also compared in the original VRED paper [2], as well as the individual terms of our loss functions:

- Base position loss  $L_{base}$ , as defined in Section 4.1
- Angle loss  $L_{angles}$ , as defined in Section 4.1
- Goal loss  $L_{goal}$ , as defined in Section 4.3

Table 5.1 compares the distinct models on the walking dataset. As expected, all models, including the VRED outperform the baselines by a significant margin. Altogether, the Goal-Conditioned VRED with Goal Loss performed best, being superior to the other networks in all categories except the angle loss, where the Goal-Conditioned VRED is slightly better. A reasonable explanation for that is that the inclusion of the goal loss into the loss function decreases the weight of the angle loss. The higher our hyperparameter  $\alpha$ , the worse this should get. Furthermore, the angular pose predictions of the VRED do not benefit as much from a goal input than the prediction of the base position.

Another salient point is the decrease in position loss and goal loss error when including the goal loss, dropping from  $3.51 \cdot 10^{-3}$  to  $0.80 \cdot 10^{-3}$ , and from  $7.61 \cdot 10^{-3}$  to  $0.76 \cdot 10^{-3}$  respectively. Unfortunately, the Adversarial VRED does not perform as well as hoped, and, while still outperforming the other networks, is worse throughout the field than the Goal-Conditioned VRED with Goal Loss.

Error	Long-term	Position loss	Angle loss	Goal loss
Zero-Vel	$10.24 \cdot 10^{-2}$	-	$20.86 \cdot 10^{-2}$	-
Moving Avg	$10.13 \cdot 10^{-2}$	-	$20.86 \cdot 10^{-2}$	-
Const-Vel	-	$16.86 \cdot 10^{-3}$	-	-
VRED	$5.49 \cdot 10^{-2}$	$3.66 \cdot 10^{-3}$	$2.35 \cdot 10^{-2}$	$19.12 \cdot 10^{-3}$
VRED + GC (ours)	$4.39 \cdot 10^{-2}$	$3.51 \cdot 10^{-3}$	<b><math>1.94 \cdot 10^{-2}</math></b>	$7.61 \cdot 10^{-3}$
VRED + GC + GL (ours)	<b><math>3.69 \cdot 10^{-2}</math></b>	<b><math>0.80 \cdot 10^{-3}</math></b>	$1.97 \cdot 10^{-2}$	<b><math>0.76 \cdot 10^{-3}</math></b>
Adversarial VRED	$4.32 \cdot 10^{-2}$	$2.26 \cdot 10^{-3}$	$2.57 \cdot 10^{-2}$	$2.13 \cdot 10^{-3}$

**Table 5.1:** Multiple errors compared on the walking dataset.

For the pick and place dataset, a similar image presents itself 5.2. In general, one can recognize that the pick and place dataset is harder for the prediction task than the walking dataset. This does not only intuitively make sense, but is also visible through the increased errors across the board. Especially the position and goal loss are reaching high values with no kind of goal condition given. Alike the previous table, the Goal-Conditioned VRED with Goal Loss performs best, except for the goal loss. Here, the Adversarial VRED achieves a higher accuracy, while being subpar for all other measures. What stands out is the decrease in the position loss from the VRED to the Goal-Conditioned VRED. In contrast to the walking dataset, where adding the goal to the input did not improve the position loss

significantly, the position loss more than halves for the pick and place dataset. This can be explained by the unpredictability of the pick and place trajectories. While for walking, the base end position should be somewhere in the walking direction given the current velocity, the base end position for a pick or place prediction can be really distinct (e.g. behind the human). A related progress can also be seen for the goal loss.

Error	Long-term	Position loss	Angle loss	Goal loss
Zero-Vel	$13.44 \cdot 10^{-2}$	-	$24.97 \cdot 10^{-2}$	-
Moving Avg	$13.39 \cdot 10^{-2}$	-	$24.97 \cdot 10^{-2}$	-
Const-Vel	-	$23.34 \cdot 10^{-3}$	-	-
VRED	$10.14 \cdot 10^{-2}$	$12.56 \cdot 10^{-3}$	$6.21 \cdot 10^{-2}$	$71.23 \cdot 10^{-3}$
VRED + GC (ours)	$9.62 \cdot 10^{-2}$	$4.59 \cdot 10^{-3}$	$5.99 \cdot 10^{-2}$	$17.74 \cdot 10^{-3}$
VRED + GC + GL (ours)	<b><math>7.99 \cdot 10^{-2}</math></b>	<b><math>3.84 \cdot 10^{-3}</math></b>	<b><math>5.06 \cdot 10^{-2}</math></b>	$2.60 \cdot 10^{-3}$
Adversarial VRED	$8.88 \cdot 10^{-2}$	$4.16 \cdot 10^{-3}$	$5.81 \cdot 10^{-2}$	<b><math>2.10 \cdot 10^{-3}</math></b>

**Table 5.2:** Multiple errors compared on the pick and place dataset.

### 5.3.4 Maximum Entropy IRL

To evaluate the performance of our simple MaxEnt IRL algorithm, we implemented leave-one-out cross validation. This way, in each iteration a new IRL model was trained on 24 of the 25 available demonstrations for setting up a table for a single person. After training, we examined if the left out test demonstration could be recovered from the learned policy. The evaluation shows, that in 80% of the cases, the agent is able to successfully complete the task in the same steps as desired. If we just count permutations of the test demonstration as positives, this value drops to 52%. Finally, the percentage of cases where the learned policy was able to perfectly mimic the expert lies by 16%.

In our opinion, the low number of identical recoveries of the demonstrated behaviors has different reasons. At first, we use a simplistic discretized state space, which means that higher-level notions of the full-state trajectories can get lost during conversion. Secondly, six different experts demonstrated the data, that all could have different behaviors of solving the task. Finally, as we do not use features in our MaxEnt IRL algorithm and just have 25 demonstrations to work with, it is hard for the algorithm to solve the still large state space of size 22500.

There exist three kinds of possibilities that can be responsible for a trajectory that is not consistent with the test data:

- The learned policy successfully sets up the table but in a different order (permutation)
- The learned policy successfully sets up the table but in a different fashion (variation)
- The learned policy is unsuccessful in setting up the table and alternates between one or more states (failure)

Permutations can occur for example if the behavior of the demonstrators vary. Consider the case where you have to put a plate on the table and move a cup away from the table to finish the task. Person A could prefer first clearing the objects from the table and then setting it up, while person B could behave the opposite way. Then, if person B is in our test set, the learned policy would still imitate person A and create a permutation. Failures can occur if the executed greedy policy either i) picks an action which maps back to the current state or ii) picks multiple actions in sequence that end in a loop.

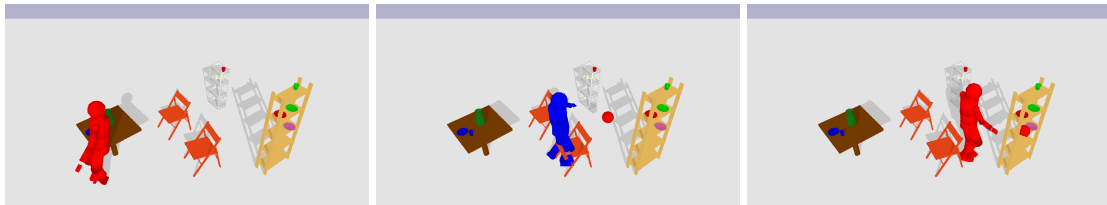
An easy example of a perfect imitation can be seen in Figure 5.4. The presented trajectory is with four actions the smallest trajectory in the demonstrated dataset to complete the task of setting up the table for a single person. The only thing the human has to do is move a cup from the white shelf to the table. More supplementing material and cases can be found in appendix B.

	Ground Truth	Learned Policy
Start State	[0, 4, 0, 1, 0, 3, 1, 0, 1, 2]	[0, 4, 0, 1, 0, 3, 1, 0, 1, 2]
Actions	Go to White Shelf Pick Up Cup Go to Table Place	Go to White Shelf Pick Up Cup Go to Table Place
End State	[1, 3, 0, 1, 0, 3, 1, 0, 1, 0]	[1, 3, 0, 1, 0, 3, 1, 0, 1, 0]

**Table 5.3:** Example for a perfect imitation using the learned policy from our MaxEnt IRL algorithm

### 5.3.5 Mapping Learned Policy to VREDs

The final step of our work was to combine the learned policy from the MaxEnt IRL algorithm on the symbolic state representation of the environment with our goal-conditioned neural networks for walking and manipulating objects. In Figure 5.6 you can see three example frames of the agent during execution, representing a grasping action (left), a walking action (middle) and a placing action (right) respectively. The human is thereby blue when the VRED trained on the walking dataset is used for the prediction. Respectively, a prediction made by the pick and place VRED is visualized in red.



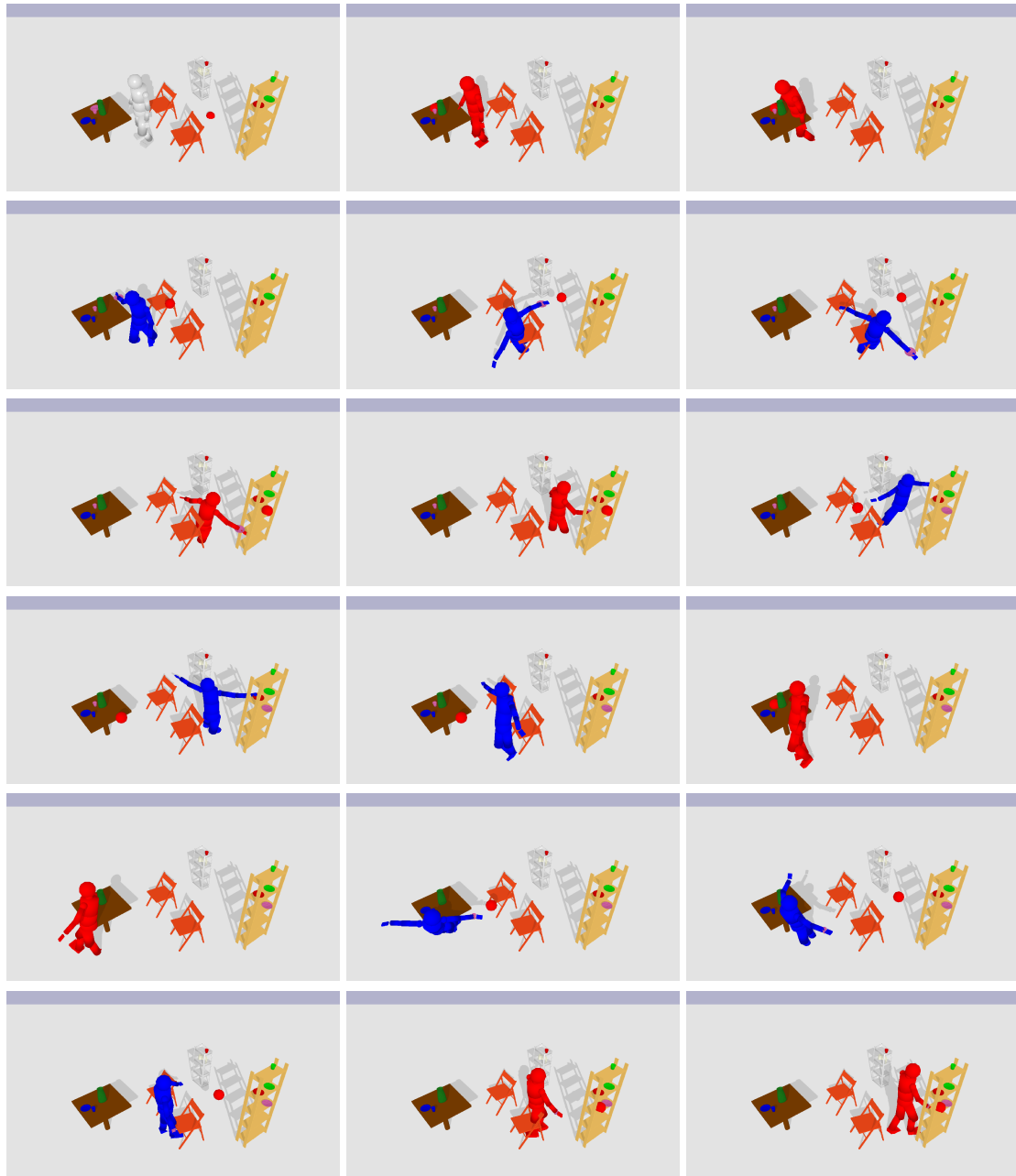
**Figure 5.6:** Examples of predictions when mapping the learned policy of the IRL algorithm to our VRED networks.

We show a full example run in Figure 5.7. The trajectory consists of 11 subgoals, which results in a total length of 330 frames for the prediction. Visible is each 18th frame of the trajectory, starting from frame 18 (top left) to frame 324 (bottom right). Thereby, the human is solving the task of setting up the table for a single person in the following fashion

	Policy
Start State	[2, 1, 1, 2, 0, 2, 1, 1, 0, 0]
Actions	Pick Up Plate Go to Beige Shelf Place Go to Table Pick Up Cup Go to Beige Shelf Place
End State	[1, 1, 2, 1, 0, 3, 1, 1, 0, 2]

**Table 5.4:** Executed policy for the full trajectory prediction of the human.

The learned policy from the tabular MaxEnt IRL algorithm is predicting the discrete actions, afterwards they are mapped by the heuristics from Chapter 4.7 to three-dimensional goal inputs for the networks. The results show, that the VREDs are able to make a trajectory prediction without drifting away. However, especially the walking VRED has difficulties in the prediction when it has to turn around. Predictions made in these scenarios look really unnatural, as the original model was solely trained on data without abrupt changes in directions. Training a new model for turning on the pick and place dataset could possibly fix this issue. Furthermore, as we did not include any type of collision yet, the human often bumps into the stationary objects when traversing the trajectory. A collision model could help to penalize these kind of behaviors. Still, the model is able to solve the task using the pretrained goal-conditioned VREDs in a simplified fashion.



**Figure 5.7:** Human full trajectory prediction by mapping the learned policy of the IRL algorithm to the trained walking and pick and place networks.

### 5.3.6 Discussion

In this chapter we presented our experiment setups and results. We want to quickly summarize our respective outcomes.

The qualitative comparison of Sections 5.3.1 and 5.3.2 show the differences of the predicted endeffector trajectory when adding a goal condition to the network input, as well as including a goal loss to the model loss function. The experiments expound, that the addition of the goal condition manages to produce trajectories that end closer to the last position and goal of the ground truth. Furthermore, including the goal loss further improves the accuracy of the last prediction frame. General limitations of the networks occur for the foot predictions. While for the walking dataset the walking motion is appropriately learned, the networks trained on the pick and place dataset have problems to imitate it, especially when turning around or making smaller movements.

A quantitative comparison of different tested measures was conducted in Section 5.3.3. Results show, that the walking dataset is easier to learn from, as the motions are less arbitrary, and therefore more predictable. Moreover, the Goal-Conditioned VRED with Goal Loss performed best for both the walking dataset and the pick and place dataset. Unfortunately, the adversarial framework did score worse to the former, and would need a closer examination for improvements. Further limitations can be seen for the angular loss, which was hard to improve, and did even slightly increase for the Goal-Conditioned VRED with Goal Loss.

The investigation of the tabular MaxEnt IRL algorithm showed that the learned policy was able to solve the task in 80% of the cases. However, a perfect imitation was achieved solely in 16% of the test runs of the cross validation. As previously mentioned in Section 5.3.4, the algorithm is limited by our symbolic state and action representation and not yet including any kind of features. Moreover, the different experts used for the demonstrations further complicate the prediction.

Finally, the VREDs trained on the walking and pick and place dataset were able to combine the goal inputs from the heuristics into a human trajectory (see Section 5.3.5). Nevertheless, especially the walking VRED has issues predicting a meaningful trajectory, as behaviors like turning and holding an object are not part of the dataset and were therefore not explicitly trained. Additionally, the human is still colliding with stationary objects in the scene, as no collision model is implemented.



## 6 Future Work / Conclusion

As this work combines multiple different approaches and sub-areas of RL, there are a fair amount of regions follow-up work could expand on.

The recorded walking dataset could be extended with different gaits like running or sprinting, which would open up the possibility of learning an agent that would adapt its behavior for example based on the distance to the next subgoal. For the pick and place dataset, more tasks and demonstrations could be added. Another interesting idea would be to include failed demonstrations, which is an active area of research and would create an additional challenge for the IRL algorithm.

Currently, our trained VREDs are always predicting the next second trajectory for the human. As subtasks are usually never of same lengths, a network that could predict trajectories of arbitrary length would be able to use more than a subset of possible subgoals and thereby increase temporal abstraction. Furthermore, the subpar results of our adversarial network can be examined more closely. Also, adding a collision or affordance model like recent work [56] to the heuristics is an option. Training a VRED with base goal for walking on the pick and place dataset could improve the quality of the trajectory prediction. Another interesting approach would be to condition the networks on environment dynamics similar to Peng et al. [60]. Models could then be trained in different simulated environments, so that in the end, a transfer to the real world would be imaginable.

Ultimately, we see the largest possible extension in the choice of the IRL algorithm. In our work, we use a discrete state representation of the environment to be able to use tabular MaxEnt IRL. Future Work could extend this to include model-free RL like Soft Q-Learning instead of Soft Q-Iteration. This way, although also increasing computational effort, one could support continuous state and action spaces. Furthermore, additions like adversarial training [7], hierarchies [8] and goal-conditioned learning [34] directly into the IRL framework could improve its generalizability, as well as the capability of retrieving meaningful subgoals.

In total, the idea would be to have a potent IRL method which is able to directly map from the continuous state-action space of the dataset to suitable subgoals for lower-level network predictions like our VREDs.

## 6.1 Conclusion

In this thesis, we present a two-step framework that combines goal-conditioned supervised learning with IRL from labeled, demonstrated data. Our method trains two types of networks, one for walking within the environment and one for picking and placing items from stationary objects. Furthermore we embed the idea of goal distance regularization, as well as the concept of adversarial training into the model structure. We then implement a tabular MaxEnt IRL algorithm to learn a reward function on the symbolic state and action space of the original task. Finally, the learned policy and its actions are remapped to the continuous state and action space of our trained networks for a simulated long-term prediction.

While the adversarial implementation of the VRED does not improve the predictions of the former trained networks, the goal-conditioned VRED with goal loss reaches desired subgoals with a high accuracy for both the walking dataset, as well as the pick and place dataset. The tabular MaxEnt IRL algorithm does reasonably well imitating the demonstrator policy, but can be further extended with features or a more complex environment representation to increase the generalizability of the learned reward function. Finally, the synthesized motions of the heuristics look promising for the pick and place parts. However the walking VRED struggles to predict meaningful poses, due to the pick and place dataset being too distinct from the dataset the model was trained on.

With more computational power and elaborated algorithms, as well as the progresses made in Deep Learning, Imitation Learning became a powerful alternative to Reinforcement Learning. The ability to imitate humans by learning from expert demonstrations opens up a variety of fields where agents would struggle without additional help. Therefore, it is exciting to see, how far this active area of research can be expanded in the next years and where it is headed.

## Bibliography

- [1] O. Vinyals, I. Babuschkin, W.M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D.H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J.P. Agapiou, M. Jaderberg, A.S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T.L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, D. Silver. “Grandmaster level in StarCraft II using multi-agent reinforcement learning.” *Nature*. Vol. 575. Springer US, 2019, pp. 350–354 (cit. on p. 17).
- [2] H. Wang, J. Feng. “VRED: A Position-Velocity Recurrent Encoder-Decoder for Human Motion Prediction.” 2019. arXiv: [1906.06514](https://arxiv.org/abs/1906.06514) (cit. on pp. 18, 20, 23, 34, 51).
- [3] J.N. Kundu, M. Gor, R.V. Babu. “BiHMP-GAN: Bidirectional 3D Human Motion Prediction GAN.” *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 8553–8560. arXiv: [1812.02591](https://arxiv.org/abs/1812.02591) (cit. on pp. 18, 21).
- [4] L.Y. Gui, Y.X. Wang, X. Liang, J.M. Moura. “Adversarial Geometry-Aware Human Motion Prediction.” *Lecture Notes in Computer Science*. Vol. 11208. 2018, pp. 823–842 (cit. on pp. 18, 21, 38).
- [5] A. Gupta, J. Johnson, L. Fei-Fei, S. Savarese, A. Alahi. “Social GAN: Socially Acceptable Trajectories with Generative Adversarial Networks.” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2018, pp. 2255–2264. arXiv: [1803.10892](https://arxiv.org/abs/1803.10892) (cit. on pp. 18, 20, 21).
- [6] B.D. Ziebart, A. Maas, J. Bagnell, A.K. Dey. “Maximum Entropy Inverse Reinforcement Learning.” *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*. 2008, pp. 1433–1438 (cit. on pp. 18, 30, 40).
- [7] J. Fu, K. Luo, S. Levine. “Learning robust rewards with adversarial inverse reinforcement learning.” *6th International Conference on Learning Representations*. 2018, pp. 1–15. arXiv: [1710.11248](https://arxiv.org/abs/1710.11248) (cit. on pp. 18, 20, 30, 57, 65).
- [8] D. Venuto, J. Chakravorty, L. Boussioux, J. Wang, G. McCracken, D. Precup. “oIRL: Robust Adversarial Inverse Reinforcement Learning with Temporally Extended Actions.” 2020. arXiv: [2002.09043](https://arxiv.org/abs/2002.09043) (cit. on pp. 18, 20, 30, 57).
- [9] I.J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio. “Generative adversarial nets.” *Advances in Neural Information Processing Systems*. Vol. 3. January. 2014, pp. 2672–2680. arXiv: [1406.2661](https://arxiv.org/abs/1406.2661) (cit. on pp. 19, 24–26, 37).

## Bibliography

---

- [10] J. Gui, Z. Sun, Y. Wen, D. Tao, J. Ye. “A Review on Generative Adversarial Networks: Algorithms, Theory, and Applications.” Vol. 14. 2020, pp. 1–28. arXiv: [2001.06937](#) (cit. on p. 19).
- [11] M. Mirza, S. Osindero. “Conditional Generative Adversarial Nets.” *The Computing Research Repository*. 2014, pp. 1–7. arXiv: [1411.1784v1](#) (cit. on p. 19).
- [12] S. Nowozin, B. Cseke, R. Tomioka. “f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization.” *Advances in Neural Information Processing Systems*. 2016, pp. 271–279. arXiv: [1606.00709v1](#) (cit. on pp. 19, 26).
- [13] G.-j. Qi. “Loss-Sensitive Generative Adversarial Networks on Lipschitz.” *International Journal of Computer Vision*. 2019, pp. 1–23. arXiv: [1701.06264v6](#) (cit. on p. 19).
- [14] M. Arjovsky, S. Chintala, L. Bottou. “Wasserstein GAN.” 2017. arXiv: [1701.07875](#) (cit. on pp. 19, 26, 37).
- [15] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, A. Courville. “Improved training of wasserstein GANs.” *Advances in Neural Information Processing Systems*. Vol. December. 2017, pp. 5768–5778. arXiv: [1704.00028](#) (cit. on pp. 19, 26, 46).
- [16] J. Adler, S. Lunz. “Banach Wasserstein GaN.” *Advances in Neural Information Processing Systems*. Vol. Decem. 2018, pp. 6754–6763. arXiv: [1806.06621](#) (cit. on p. 19).
- [17] Z. Zhou, J. Liang, Y. Song, L. Yu, H. Wang, W. Zhang, Y. Yu, Z. Zhang. “Lipschitz generative adversarial nets.” *36th International Conference on Machine Learning*. Vol. June. 2019, pp. 13073–13082. arXiv: [1902.05687](#) (cit. on p. 19).
- [18] H. Gouk, E. Frank, B. Pfahringer, M. Cree. “Regularisation of Neural Networks by Enforcing Lipschitz Continuity.” 2018, pp. 1–30. arXiv: [1804.04368](#) (cit. on p. 19).
- [19] V. M. Alta, B. Rouge. “Local Stability and Performance of Simple Gradient Penalty  $\mu$ -Wasserstein GAN.” February. 2001, pp. 78–81 (cit. on p. 20).
- [20] J. Ho, S. Ermon. “Generative adversarial imitation learning.” *Advances in Neural Information Processing Systems*. 2016, pp. 4572–4580. arXiv: [1606.03476](#) (cit. on pp. 20, 30).
- [21] C. Finn, P. Christiano, P. Abbeel, S. Levine. “A Connection between Generative Adversarial Networks, Inverse Reinforcement Learning, and Energy-Based Models.” 2016. arXiv: [1611.03852](#) (cit. on p. 20).
- [22] H. Xiao, M. Herman, J. Wagner, S. Ziesche, J. Etesami, T. H. Linh. “Wasserstein Adversarial Imitation Learning.” 2019. arXiv: [1906.08113](#) (cit. on p. 20).
- [23] K. Fragkiadaki, S. Levine, P. Felsen, J. Malik. “Recurrent network models for human dynamics.” *Proceedings of the IEEE International Conference on Computer Vision*. Vol. International Conference on Computer Vision. 2015, pp. 4346–4354. arXiv: [1508.00271](#) (cit. on pp. 20, 23).
- [24] J. Martinez, M. J. Black, J. Romero. “On human motion prediction using recurrent neural networks.” *Proceedings of the 30th IEEE Conference on Computer Vision and Pattern Recognition*. Vol. January. 2017, pp. 4674–4683. arXiv: [1705.02445](#) (cit. on pp. 20, 23, 46).

- 
- [25] D. Pavllo, D. Grangier, M. Auli. “QuaterNet: A quaternion-based recurrent model for human motion.” *British Machine Vision Conference*. 2018. arXiv: [1805.06485](#) (cit. on pp. 20, 24).
- [26] E. Barsoum, J. Kender, Z. Liu. “HP-GAN: Probabilistic 3D human motion prediction via GAN.” *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. Vol. June. 2018, pp. 1499–1508. arXiv: [1711.09561](#) (cit. on pp. 21, 38).
- [27] Z. Wang, J. Chai, S. Xia. “Combining Recurrent Neural Networks and Adversarial Training for Human Motion Synthesis and Control.” *IEEE Transactions on Visualization and Computer Graphics*. 2019, pp. 1–1. arXiv: [1806.08666](#) (cit. on pp. 21, 37, 38).
- [28] R. Fox, U. C. Berkeley, U. C. Berkeley, K. Goldberg, U. C. Berkeley. “Multi-Level Discovery of Deep Options.” 2017. arXiv: [1703.08294v2](#) (cit. on p. 21).
- [29] A. Sharma, M. Sharma, N. Rhinehart, K. M. Kitani. “Directed-Info GAIL: Learning Hierarchical Policies from Unsegmented Demonstrations Using Directed Information.” *International Conference on Learning Representations*. 2018, pp. 1–15. arXiv: [1810.01266v2](#) (cit. on p. 21).
- [30] P. Henderson, W. D. Chang, P. L. Bacon, D. Meger, J. Pineau, D. Precup. “OptionGAN: Learning joint reward-policy options using generative adversarial inverse reinforcement learning.” *32nd AAAI Conference on Artificial Intelligence*. 2018, pp. 3199–3206. arXiv: [1709.06683](#) (cit. on p. 21).
- [31] K. Hausman, Y. Chebotar, S. Schaal, G. Sukhatme, J. J. Lim. “Multi-Modal Imitation Learning from Unstructured Demonstrations using Generative Adversarial Nets.” *Advances in Neural Information Processing Systems*. 2017. arXiv: [1705.10479v2](#) (cit. on p. 21).
- [32] T. Kipf, Y. Li, H. Dai, V. Zambaldi. “CompILE : Compositional Imitation Learning and Execution.” *International Conference on Machine Learning*. 2019 (cit. on p. 21).
- [33] R. Prakash. “Hierarchical Inverse Reinforcement Learning using Motion Capture Data.” MA thesis. University of Stuttgart, 2019 (cit. on p. 21).
- [34] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, W. Zaremba. “Hindsight experience replay.” *Advances in Neural Information Processing Systems*. Vol. December. 2017, pp. 5049–5059. arXiv: [1707.01495](#) (cit. on pp. 22, 57).
- [35] L. P. Kaelbling. “Learning to Achieve Goals.” *International Joint Conference on Artificial Intelligence*. 1993, pp. 1094–1098 (cit. on p. 22).
- [36] V. Pong, S. Gu, M. Dalal, S. Levine. “Temporal Difference Models: Model-Free Deep RL for Model-Based Control.” *6th International Conference on Learning Representations*. 2018, pp. 1–14 (cit. on p. 22).
- [37] T. Schaul, D. Horgan, K. Gregor, D. Silver. “Universal Value Function Approximators.” *Proceedings of the 32nd International Conference on Machine Learning*. 2015, pp. 1312–1320. arXiv: [1806.06161](#) (cit. on p. 22).

## Bibliography

---

- [38] A. Nair, V. Pong, M. Dalal, S. Bahl, S. Lin, S. Levine. “Visual Reinforcement Learning with Imagined Goals.” *Advances in Neural Information Processing Systems*. 2018 (cit. on p. 22).
- [39] C. Lynch, M. Khansari, T. Xiao, V. Kumar, J. Tompson, S. Levine, P. Sermanet. “Learning latent plans from play.” *3rd Conference on Robot Learning*. 2020, pp. 1113–1132. arXiv: [1903.01973](https://arxiv.org/abs/1903.01973) (cit. on p. 22).
- [40] P. Kratzer, M. Toussaint, J. Mainprice. “Prediction of Human Full-Body Movements with Motion Optimization and Recurrent Neural Networks.” *IEEE International Conference on Robotics and Automation*. 2020. arXiv: [1910.01843](https://arxiv.org/abs/1910.01843) (cit. on pp. 22, 33).
- [41] H. kuang Chiu, E. Adeli, B. Wang, D. A. Huang, J. C. Niebles. “Action-agnostic human pose forecasting.” *Proceedings of the IEEE Winter Conference on Applications of Computer Vision*. 2019, pp. 1423–1432. arXiv: [1810.09676v1](https://arxiv.org/abs/1810.09676v1) (cit. on pp. 23, 24).
- [42] P. Ghosh, J. Song, E. Aksan, O. Hilliges. “Learning human motion models for long-Term predictions.” *Proceedings of the International Conference on 3D Vision*. 2018, pp. 458–466. arXiv: [1704.02827v2](https://arxiv.org/abs/1704.02827v2) (cit. on p. 23).
- [43] C. Li, Z. Zhang, W. S. Lee, G. H. Lee. “Convolutional Sequence to Sequence Model for Human Dynamics.” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2018, pp. 5226–5234. arXiv: [1805.00655](https://arxiv.org/abs/1805.00655) (cit. on pp. 23, 24).
- [44] Y. Tang, L. Ma, W. Liu, W. S. Zheng. “Long-term human motion prediction by modeling motion context and enhancing motion dynamic.” *International Joint Conference on Artificial Intelligence*. Vol. July. 2018, pp. 935–941 (cit. on p. 23).
- [45] S. Hochreiter, J. Schmidhuber. “Long Short-Term Memory.” *Neural Computation*. Vol. 9. 1997, pp. 1735–1780 (cit. on p. 24).
- [46] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio. “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation.” 2014. arXiv: [1406.1078](https://arxiv.org/abs/1406.1078) (cit. on p. 24).
- [47] F. S. Grassia. “Practical Parameterization of Rotations Using the Exponential Map.” *Journal of Graphics Tools*. Vol. 3.3. 1998, pp. 1–13 (cit. on p. 24).
- [48] C. Ionescu, D. Papava, V. Olaru, C. Sminchisescu. “Human3.6M: Large scale datasets and predictive methods for 3D human sensing in natural environments.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 36. IEEE, 2014, pp. 1325–1339 (cit. on p. 24).
- [49] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen. “Improved techniques for training GANs.” *Advances in Neural Information Processing Systems*. 2016, pp. 2234–2242. arXiv: [1606.03498](https://arxiv.org/abs/1606.03498) (cit. on pp. 25, 37).
- [50] M. Lucic, K. Kurach, M. Michalski, O. Bousquet, S. Gelly. “Are GANs created equal? A large-scale study.” *Advances in Neural Information Processing Systems*. 2017 (cit. on pp. 25, 37).

- 
- [51] R. S. Sutton, D. Precup, S. Singh. “Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning.” *Artificial Intelligence*. Vol. 112. 1999, pp. 181–211. arXiv: [1011.1669v3](#) (cit. on pp. 28, 31).
- [52] R. J. Williams, J. Peng. “Function Optimization using Connectionist Reinforcement Learning Algorithms.” *Connection Science*. Vol. 3. 1991, pp. 241–268 (cit. on p. 28).
- [53] S Russell. “Learning agents for uncertain environments (extended abstract).” *Conference on Computational Learning Theory (COLT)*. 1998, pp. 1–3 (cit. on p. 29).
- [54] T. G. Dietterich. “Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition.” *Journal of Artificial Intelligence Research*. Vol. 13. 2000, pp. 227–303. arXiv: [9905014 \[cs\]](#) (cit. on pp. 31, 66).
- [55] R. Parr, S. Russell. “Reinforcement learning with hierarchies of machines.” *Advances in Neural Information Processing Systems*. 1998, pp. 1043–1049 (cit. on p. 31).
- [56] P. Kratzer, N. B. Midlagajni, M. Toussaint, J. Mainprice. “Anticipating Human Intention for Full-Body Motion Prediction in Object Grasping and Placing Tasks.” *29th IEEE International Conference on Robot and Human Interactive Communication*. 2020, pp. 1157–1163. arXiv: [2007.10038](#) (cit. on pp. 36, 45, 57).
- [57] T. Karras, S. Laine, T. Aila. “A style-based generator architecture for generative adversarial networks.” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vol. June. 2019, pp. 4396–4405. arXiv: [1812.04948](#) (cit. on p. 37).
- [58] A. Brock, J. Donahue, K. Simonyan. “Large scale GAN training for high fidelity natural image synthesis.” *7th International Conference on Learning Representations*. 2019, pp. 1–35. arXiv: [1809.11096v2](#) (cit. on p. 37).
- [59] P. Kratzer, S. Bihlmaier, N. B. Midlagajni, R. Prakash, M. Toussaint, J. Mainprice. “MoGaze : A Dataset of Full-Body Motions that Includes Workspace Geometry and Eye-Gaze.” *IEEE Robotics and Automation Letters*. 2020, pp. 1–7. arXiv: [2011.11552v1](#) (cit. on p. 45).
- [60] X. B. Peng, E. Coumans, T. Zhang, T.-W. Lee, J. Tan, S. Levine. “Learning Agile Robotic Locomotion Skills by Imitating Animals.” 2020. arXiv: [2004.00784](#) (cit. on p. 57).

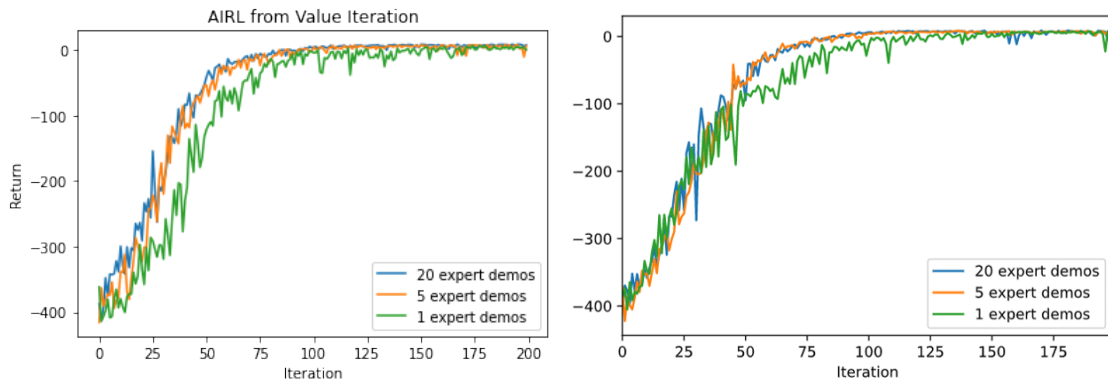
All links were last followed on November 27, 2020.





# A Experiments on the Taxi Environment

At the start of the thesis, to get into the topic, we compared various algorithms on the Taxi environment from the OpenAI Gym <sup>1</sup>. One of the algorithms we analyzed was AIRL [7], where we wanted to experiment with different amount of expert trajectories and expert sources. The plots show, that AIRL is able to achieve good returns in the environment also with a small amount of information.



**Figure A.1:** Results of the AIRL algorithm on the Taxi environment using different types of experts and amounts of expert demonstrations.

Similar to the AIRL trial, we investigated the effect of the number of available demonstrations on the MaxEnt IRL algorithm. In table A.1 we plot the average return for different amounts, as well as the q-values for a sample state of the environment.

Expert	Mean Policy Return	Down	Up	Right	Left	Pick	Drop
1 Demo	-494.77	[0.089	0.255	0.266	0.093	0.148	0.148
10 Demos	3.234	[0.000	0.416	0.522	0.001	0.031	0.031]
100 Demos	7.982	[0.021	0.952	0.023	0.000	0.002	0.002]
1.000 Demos	7.922	[0.000	0.823	0.168	0.000	0.005	0.005]
10.000 Demos	8.158	[0.000	0.873	0.117	0.000	0.005	0.005]
Value Iteration	8.585	[0.000	1.000	0.000	0.000	0.000	0.000]

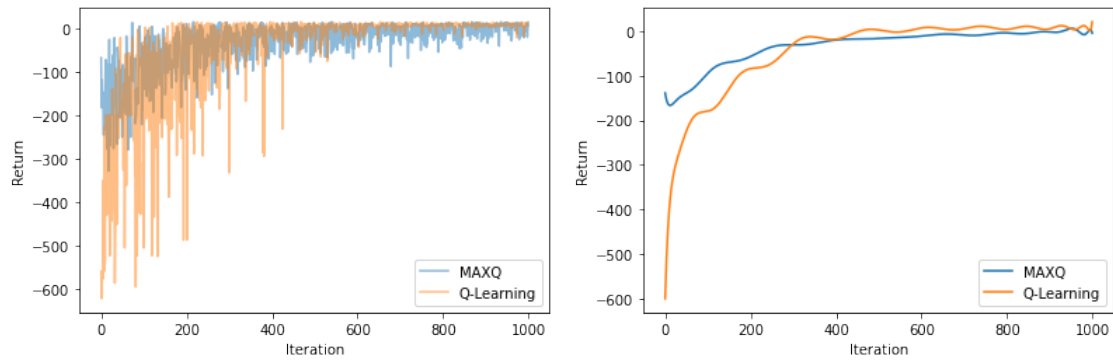
**Table A.1:** Comparison of different amounts of expert demos for the MaxEnt IRL algorithm on the taxi environment.

<sup>1</sup><https://gym.openai.com/>

## A Experiments on the Taxi Environment

---

Lastly, we also compared the MaxQ algorithm [54] with standard Q-Learning. In the left part of Figure A.2 you can see the results after 1000 iterations, and on the right a fitted polynomial is shown to reduce the clutter.



**Figure A.2:** Results of using MaxQ and Q-Learning on the Taxi Environment for multiple granularities (left, middle) and a polynomial approximation (right)

## B Additional Examples for the Maximum Entropy IRL

In this appendix, we present two additional trajectory samples that occurred during our cross-validation. An example of a permutation can be seen in Table B.1. While the demonstrator first carries the cup from the beige shelf to the table, the learned policy swaps the actions but still arrives in the same end state.

	Ground Truth	Learned Policy
Start State	[0, 2, 2, 0, 0, 4, 1, 0, 1, 2]	[0, 2, 2, 0, 0, 4, 1, 0, 1, 2]
Actions	Pick Up Plate Go to Table Place Go to Beige Shelf Pick Up Cup Go to Table Place	Pick Up Cup Go to Table Place Go to Beige Shelf Pick Up Plate Go to Table Place
End State	[1, 2, 1, 1, 0, 3, 1, 0, 1, 2]	[1, 2, 1, 1, 0, 3, 1, 0, 1, 2]

**Table B.1:** Example for a permutation case using the learned policy from the MaxEnt IRL algorithm

In the second example, we visualize a trajectory that needs the same amount of steps to set up the table like the ground truth test demonstration, but chooses different actions for reaching the objective B.2 and stops in a different end state.

	Ground Truth	Learned Policy
Start State	[2, 0, 2, 0, 0, 4, 1, 1, 0, 3]	[2, 0, 2, 0, 0, 4, 1, 1, 0, 3]
Actions	Go to Beige Shelf Pick Up Plate Go to Table Place Pick Up Cup Go to White Shelf Place	Go to Table Pick Up Cup Go to Beige Shelf Place Pick Up Plate Go to Table Place
End State	[1, 1, 2, 1, 0, 3, 1, 1, 0, 1]	[1, 0, 3, 1, 0, 3, 1, 1, 0, 0]

**Table B.2:** Example for a variation case using the learned policy from the MaxEnt IRL algorithm

## **Declaration**

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature