# Inverse Reinforcement Learning from Observation for Locomotion on the Unitree A1 Robot

**Inverse Reinforcement Learning anhand von Beobachtungen zum Erlernen des Gehverhaltens des Unitree A1 Roboters**
Bachelor thesis by Tim Althaus
Date of submission: April 13, 2023

1. Review: Prof. Ph.D Jan Peters
2. Review: Ph.D Davide Tateo
3. Review: M.Sc Firas Al-Hafez
Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT

## Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 und § 23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Tim Althaus, die vorliegende Bachelorarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß § 23 Abs. 7 APB überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 13. April 2023

Tim Althaus

# Abstract

Quadrupeds are capable of performing complex tasks in challenging environments. Many of the state-of-the-art approaches are based on Model Predictive Control to navigate through these environments. Model Predictive Control predicts the robot's behavior at each time step. This prediction requires large computational resources during execution and a high level of expert knowledge to design the model. These requirements can lead to a slow reaction time, less computational resources for other tasks, and a vulnerability to model errors. This thesis uses Imitation Learning to generate behavior that is comparable to the results of Model Predictive Control. More precisely, it uses Inverse Reinforcement Learning with a model of the Unitree A1 to imitate an expert. It produces a policy that can walk in eight different directions, independent of the robot's yaw. The result is a fast and computationally inexpensive agent with comparable results to the expert. The model requires no expert knowledge. Additionally, the thesis prepares for further experiments with more complex tasks.
This work implements a model of the Unitree A1 with all necessary properties for Inverse Reinforcement Learning. It embeds this model in the learning environment with the required algorithms. To generate expert data, it adapts an existing Model Predictive Control model. For the final goal, it compares several different training settings and properties. The end result is an agent that is capable of walking forward, backward, sideways, and diagonally, regardless of the robot's yaw.

# Zusammenfassung

Vierbeinige Roboter sind in der Lage komplexe Aufgaben in anspruchsvollen Umgebungen auszuführen. Viele aktuelle Ansätze basieren auf Model Predictive Control um durch diese Umgebungen zu navigieren. Model Predictive Control sag das Verhalten des Roboter in jedem Zeitschritt voraus. Diese Vorhersage erfordert einen hohen Rechenaufwand während der Laufzeit und viel Expertenwissen um das Model zu designen. Diese Anforderungen können zu einer langsamen Reaktionszeit, weniger Rechenkapzität für andere Aufgaben und zu einer Anfälligkeit für Modellfehler führen. Diese Thesis nutzt Imitation Learning um Verhalten zu erzeugen, das mit den Ergebnissen von Model Predictive Control vergleichbar ist. Genauer gesagt wird Inverse Reinforcement Learning verwendet um mit einem Modell des Unitree A1 einen Experten zu imitieren. Dadurch wird ein Verhalten erzeugt, das unabhängig von der Rotation um die vertikale Achse in acht verschiedenen Richtungen laufen kann. Das Ergebnis ist ein reaktionsschneller und recheneffizienter Agent mit vergleichbaren Ergebnissen zu dem Experten. Das Modell benötigt kein Expertenwissen. Darüber hinaus bereitet die Thesis den Weg für weitere Experimente mit komplexeren Aufgaben vor.
Diese Arbeit implementiert ein Modell des Unitree A1 mit allen notwendigen Eigenschaften für das Inverse Reinforcement Learning. Sie bettet dieses Modell in die Lernumgebung mit den entsprechenden Algorithmen ein. Zur Erzeugung der Expertendemonstrationen wird ein bestehendes Model Predictive Control Modell adaptiert. Für das Endziel vergleicht sie verschiedene Trainingseinstellungen und -eigenschaften. Das Endergebnis ist ein Agent, der fähig ist vorwärts, rückwärts, seitwärts und diagonal zu laufen unabhängig von der Rotation um die vertikale Achse.

# Contents

# Figures and Tables

## List of Figures

# Abbreviations, Symbols and Operators

## List of Abbrevations

| Notation | Description |
|---|---|
| AIL | Adversarial Imitation Learning |
| GAIL | Generative Adversarial Imitation Learning |
| GAILfO | Generative Adversarial Imitation from Observation |
| GAN | Generative Adversarial Networks |
| IL | Imitation Learning |
| IQLearn | Inverse soft-Q Learning for Imitation |
| IRL | Inverse Reinforcement Learning |
| LS-IQ | Least Squares Inverse Q-Learning |
| MDP | Markov Decision Process |
| MDP\R | Markov Decision Process without reward function |
| MPC | Model Predictive Control |
| RL | Reinforcement Learning |
| SQIL | Soft Q Imitation Learning |
| TRPO | Trust Region Policy Optimization |

| | |
|---|---|
| VAIL | Variational Adversarial Imitation Learning |
| VAILfO | Variational Adversarial Imitation Learning from Observation |
| VDB | Variational Discriminator Bottleneck |

## List of Symbols

| Notation | Description |
|---|---|
| $\mathcal{D}$ | Decoder used in VAIL for the VDB |
| $\mathrm{D}$ | Discriminator that discriminates between the expert data and the generated data in GAN, GAIL and VAIL |
| $\mathcal{E}$ | Encoder used in VAIL for the VDB |
| $\mathrm{E}$ | Expert agent from which we have our expert data |
| $\mathrm{G}$ | Generator that generates the new data in GAN |
| $\mathrm{Hz}$ | Mathematical unit describing the number of repeated operations per second in a periodical signal |
| $\mathrm{I_C}$ | Information constraint used in the VDB |
| $\chi$ | The prior distribution we want for the encoded samples in VAIL |
| $\mathrm{S_0}$ | Initial state distribution for states $\mathrm{S}$ of an MDP |
| $\gamma$ | Discount factor for an MDP |
| $\mu$ | Mean of encoder $\mathcal{E}$ distribution used in VAIL |
| $\eta$ | Weight for $\mathrm{H}$, prevent RL and IRL to overestimate the importance of $\mathrm{H}$ |
| $\beta$ | Lagrange multiplier introduced for VAIL to maintain the info constraint $\mathrm{I_C}$ |
| $\alpha$ | Learning rate of an RL problem. It weights the the importance of the improvement steps |

| | |
|---|---|
| $\alpha_D$ | Learning rate of the discriminator in GAIL and VAIL. It weights the the importance of the improvement steps on the discriminator $\mathrm{D}$ |
| $\alpha_\pi$ | Learning rate of the policy in GAIL and VAIL. It weights the the importance of the improvement steps on the policy |
| $\boldsymbol{\omega}$ | Weights Vector of a parameterized Discriminator $\mathrm{D}$ |
| $\boldsymbol{\theta}$ | Weights Vector for a parameterized policy |
| | |
| $\mathcal{A}$ | Set of actions in an MDP |
| $\Pi$ | Set of policies to solve an MDP |
| $\mathrm{S}_A$ | Space of positions of the abduction joint of the unitree A1: $[-0.802851, 0.802851]$ |
| $\mathcal{G}$ | Space of goal states in an IRL environment to distinguish between different desired goals in training. In our case: $[-1, 1] \times [-1, 1] \times [-0.6, 0.6]$ |
| $\mathrm{S}_H$ | Space of positions of the hip joint of the unitree A1: $[-1.0472, 1.0472]$ |
| $\mathrm{S}_K$ | Space of positions of the knee joint of the unitree A1: $[-2.69653, 2.69653]$ |
| $\mathrm{S}_P$ | Space of the trunk positions of the unitree A1: $\mathbb{R}^3$ |
| $\mathrm{S}_R$ | Space of the trunk rotations of the unitree A1: $(-\pi, \pi]^3$, where $\pi$ denotes the ratio of a circle's circumference |
| $\mathrm{S}_V$ | Space of the observed velocities of the unitree A1. It contains the change ratio of the 4 legs with 3 joints each and the position and rotation of the trunk : $\mathbb{R}^{16}$ |
| $\mathrm{S}$ | Set of states in an MDP |

## List of Functions

| Notation | Description | Definition Space |
|---|---|---|
| $\mathrm{J}$ | Expected discounted reward | $\mathrm{S} \to \mathbb{R}$ |
| $\mathrm{H}$ | Entropy of a policy | $\mathbb{R}^{\mathrm{S} \times \mathcal{A}} \to \mathbb{R}$ |
| $\mathrm{R}$ | Metric to evaluate agents. It is the accumulated difference between the desired velocity and the actual velocity per step. See Equation (5.1) for more details | $\mathrm{S} \to \mathbb{R}$ |
| $\rho$ | Occupancy measure, can be understood as the distribution of state-action pairs that an agent encounters | $\mathrm{S} \times \mathcal{A} \to \mathbb{R}$ |
| $\rho_{\pi}^{\mathrm{S}}$ | Occupancy measure for state-transitions, can be understood as the distribution of state transitions pairs that an agent encounters | $\mathrm{S} \times \mathrm{S} \to \mathbb{R}$ |
| $\pi$ | Policy of an MDP determines the probability of an action depending on a state | $\mathrm{S} \times \mathcal{A} \to [0, 1]$ |
| $\pi_{\mathrm{E}}$ | Expert policy of an MDP, the perfect behavior we want to imitate | $\mathrm{S} \times \mathcal{A} \to [0, 1]$ |
| $\pi_{\boldsymbol{\theta}}$ | Policy of an MDP parameterized with $\boldsymbol{\theta}$ | $\mathrm{S} \times \mathcal{A} \to [0, 1]$ |
| $\psi$ | Adaptive regularizer that penalizes reward functions introduced in GAIL | $\mathbb{R}^{\mathrm{S} \times \mathcal{A}} \to \mathbb{R}$ |
| $\psi^{\mathrm{S}}$ | Adaptive regularizer that penalizes reward functions introduced in GAIL | $\mathbb{R}^{\mathrm{S} \times \mathrm{S}} \to \mathbb{R}$ |
| $\mathrm{r}$ | Reward function of an MDP | $\mathrm{S} \times \mathcal{A} \to \mathbb{R}$ |
| $\mathrm{r}_{\mathrm{E}}$ | The reward function that explains the expert's behavior in IRL. | $\mathrm{S} \times \mathcal{A} \to \mathbb{R}$ |
| $\mathrm{r}^{\mathrm{S}}$ | Reward function of an MDP for states only, rewards transitions | $\mathrm{S} \times \mathrm{S} \to \mathbb{R}$ |
| $\mathrm{P}$ | Transition probability to get from a state with an action to a next state | $\mathrm{S} \times \mathcal{A} \times \mathrm{S} \to \mathbb{R}$ |

# 1. Introduction

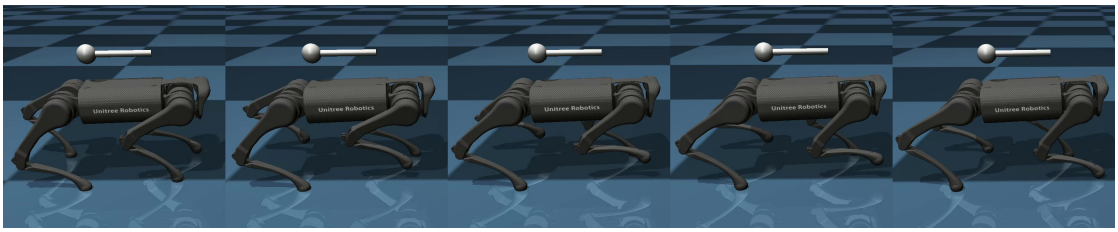


Figure 1.1.: A picture series of the final agent walking in a straight line.

Inspired by their animal counterparts, robotic quadrupeds can perform complex tasks such as various running gaits [5], stair climbing [8], and jumping over obstacles [15]. These abilities are of great interest for navigating through challenging environments. For example, robotic quadrupeds can be used in environments that are inaccessible to humans and to challenging for robots with wheels. Many of the current approaches are based on Model Predictive Control (MPC) systems [7][19][29]. These models predict a tree of possible trajectories per step, starting in the current state. Then they select the first action from the best trajectory and recalculate all trajectories for the new state again [2]. Since this leads to an exponential growth in the length of the predicted trajectories, MPC models require a large amount of computational resources while performing. Additionally, designing internal predictive models requires a high degree of expertise. If the model is inaccurate, the robot's performance will suffer significantly due to incorrect predictions. The first step towards reducing computational resources and towards a model-free approach are neural networks. They are computational inexpensive during the runtime. A common approach to train these neural networks is Reinforcement Learning (RL)[25]. Many successes have shown its suitability for various challenging tasks [12][14][18]. However, RL has one disadvantage for training complex tasks with multiple desiderata like quadruped walking. It is difficult to define a suitable goal description that balances the different requirements [1]. This often leads to fine tuning this goal description until the resulting behavior is the desired behavior. That is why we use Inverse Reinforcement Learning (IRL) to imitate the

behavior of an expert. As demonstrations we can use any data from similar locomotion, for example from real animals with four legs [21]. In this work we use data generated by an adapted MPC model. IRL is model-free, computationally efficient to run and generates an entire behavior based on some expert data [13]. The result is an agent that can react fast to uncertain situations, while reducing the computational costs during execution and keeping the required expert knowledge about the model low.

## 1.1. Goal of the Thesis

This thesis is about applied Inverse Reinforcement Learning from observation on the Unitree A1 robot [24]. The final goal is to obtain a trained agent that can produce gaits comparable to an expert. It should be able to walk in eight different directions, each with a different gait, regardless of the trunk's yaw. This goal is divided into multiple subgoals: First, expert demonstrations that can be used for the training are needed. We generate the data with a data generation pipeline that uses an adapted MPC model. Second, a model of the Unitree A1 robot must be built and embedded in the environment with the necessary algorithms. Third, agents that control the model need to be trained with IRL. They should be able to produce a periodic and stable gait that behaves similarly to the expert. The training starts with a straight forward walk and increases the complexity step-wise by adding more walking directions until the final goal is reached. An example of the desired agent walking on a straight line is shown in Figure 1.1.

We will start in Chapter 2 with a short overview of related work. Then we continue in Chapter 3 with the necessary foundations on which we build later. We explain the algorithms and the structure of a basic IRL environment. Chapter 4 describes the methodology to achieve the goal. More precisely, it explains how the expert data is generated, how the Unitree A1 model is designed, and how the learning environment is constructed. In Chapter 5, we discuss different learning experiments, their settings, and their results. We start with justified design choices about the following experiments and continue with different complex tasks until we achieve the desired behavior. We conclude the results of this thesis in Chapter 6 and provide an outlook on possible future work.

# 2. Related Work

This chapter introduces different approaches that are related to our work. They deal with the same learning problem as we do: imitating an expert's behavior. We explain these approaches and compare their benefits. Section 2.1 introduces a fundamental idea to solve the Inverse Reinforcement Learning problem. Section 2.2 deals with the state-of-the-art approach Adversarial Imitation Learning (AIL) and mentions work that is based on this approach. The next Section 2.3 comprises Imitation Learning (IL) with Sparse Rewards. The last Section 2.4 introduces two state-of-the-art algorithms to solve IRL problems with implicit rewards.

## 2.1. Apprenticeship Learning via Inverse Reinforcement Learning

The first paper by Abbeel and Ng [1] realizes Apprenticeship Learning via Inverse Reinforcement Learning. They prove that they can use the so called feature expectations to compare the expert's performance with the learned performance. So the goal is to generate a policy with a feature expectation close to the expert's feature expectation. This approach alternates between an reward optimization step and a whole RL computation to learn the corresponding policy. Their approach assumes that the reward function of the expert is linear in the features, so the algorithm can not learn a behavior with a nonlinear reward function.

## 2.2. Adversarial Imitation Learning

Another approach for IRL that is worth mentioning is Adversarial Imitation Learning (AIL). It trains two different operators: the first operator tries to imitate an expert's behavior and the second operator tries to distinguish between the expert data and the data of

the first operator [13]. Many state-of-the-art algorithms are following this idea. One algorithm we use for training, Generative Adversarial Imitation Learning (GAIL), uses Generative Adversarial Networks (GAN) to train the two operators [13]. Since it is often very easy for the second operator to distinguish between the data, Variational Adversarial Imitation Learning (VAIL) by Peng et al. implements an information bottleneck to restrict the attention of the second operator to the most important features [22]. That leads to better improvement steps of the first operator. Kostrikov et al. propose an algorithm, called Discriminator-Actor-Critic, that prevents a reward bias of other approaches in the second operator and reduces environmental interaction through an off-policy variant of AIL [17]. Adversarial Motion Priors introduced by Peng et al. combine AIL with a high-level task objective. A task objective defines the goal and short unstructured motion clips are used to learn how to reach that goal [20]. Chapter 3 explains GAIL and VAIL in more detail.

## 2.3. SQIL: Imitation Learning via Reinforcement Learning with Sparse Rewards

Reddy et al. [23] presenting another approach to IRL. Their algorithm, Soft Q Imitation Learning (SQIL), motivates the agent to return to the demonstrated data. To achieve this behavior, they apply RL with a reward function based on the expert data. The reward function assigns a constant reward of 1 for matching the expert action in the demonstrated state. All other state-action pairs receive a reward of 0. Because they use the off-policy RL algorithm soft-Q for learning, the expert data can be kept in the learning buffer and does not need to be reachable for the agent under the current policy. The algorithm is applicable to high-dimensional problems with a continuous state space and stochastic unknown dynamics. According to the authors, it achieves comparable results to GAIL [13] while the implementation is less complex.

## 2.4. Inverse Reinforcement Learning with Implicit Rewards

Many state-of-the-art algorithms formulate an implicit reward instead of using an explicit reward function to train the agent [9][11][16]. The two approaches we mention here use the Q-function to express this implicit reward. Similar to Apprenticeship Learning in Section 2.1, they transform the IRL problem into a occupancy measure matching problem.

**Inverse soft-Q Learning for Imitation** The first approach by Garg et al. [9] learns the desired policy directly instead of learning a reward that indicates the desired policy. To do this reformulation, it directly recovers the expert's Q-function. This Q-function evaluates each state-action pair in terms of its expected discounted reward and can represent the reward function and the policy. The authors prove a one-to-one relation between the Q-function and the reward function. Thus, the targeted optimum is the same as in standard approaches such as GAIL [13] and VAIL [22], but they use a simpler way to reach it. The algorithm by Garg et al., called Inverse soft-Q Learning for Imitation (IQLearn), uses a modified version of a soft actor-critic update rule to learn the Q-function. According to their benchmarks, IQLearn achieves a high positive correlation between the learned and the true reward function, uses only few expert samples for training, and can scale to complex behaviors. One disadvantage of IQLearn is that the behaviors are not generalizable to other environments, because the recovered reward depends on the environment's dynamics.

**LS-IQ: Implicit Reward Regularization for Inverse Reinforcement Learning** The last algorithm by Al-Hafez et al. [11] combines ideas from IQLearn[9], GAIL[13], and SQIL[23]. They discover properties of the IQLearn regularizer and use these properties to eliminate sources of instability, by limiting the rewards and the Q-function targets. The resulting objective can be seen as a least-squares Bellman error minimization. The authors also prove a close similarity between this objective and the SQIL approach. The new approach they propose is called Least Squares Inverse Q-Learning (LS-IQ). It avoids the survival bias of IQLearn by actively assigning a reward depending on the reward bounds and the discount factor. Their approach results in more stable training with less variance in the Q-function estimation. The authors also introduce a version for training from states only. It predicts the actions based on an Inverse-Dynamics Model and shows comparable performance to the state-action training.

# 3. Preliminaries

In the following, we introduce the basic concepts on which we will build later. We start in Section 3.1 with a description of the learning environment. The we continue with an explanation of the idea behind Reinforcement Learning and IRL in Sections 3.2 and 3.3. In the last Sections 3.4 and 3.5 with an explanation of the concrete learning algorithms we use for training.

## 3.1. Markov Decision Process

A Markov Decision Process (MDP) describes the learning environment for our learning algorithms and for the performing agents [25]. The environment is fully observable. An MDP is a tuple $(\mathrm{S}, \mathcal{A}, \mathrm{P}, \mathrm{r}, \gamma, \mathrm{S}_0)$ consisting of the state space $\mathrm{S}$, the action space $\mathcal{A}$, a state transition function $\mathrm{P} : \mathrm{S} \times \mathcal{A} \times \mathrm{S} \to [0, 1]$, a reward function $\mathrm{r} : \mathrm{S} \times \mathcal{A} \to \mathbb{R}$, a discount factor $\gamma \in [0, 1)$, and an initial state distribution $\mathrm{S}_0 : \mathrm{S} \to \mathbb{R}^+$, where $\mathbb{R}^+$ contains all positive elements in $\mathbb{R}$. $\mathrm{S}$ contains all possible states that the agent can observe. MDPs are fully observable, which means the observed state completely describes the process of the environment. The agent can perform all actions $a \in \mathcal{A}$ and depending on the current state $s$ and the performed action $a$ it reaches the next state $s'$ with probability $\mathrm{P}(s' \mid s, a)$. The reward function $\mathrm{r}(s, a)$ gives away rewards for the agent. The discount factor $\gamma$ determines how important the future rewards are for the agent in the current state. A reward i steps in the future is only worth $\gamma^{i-1}$ times the original value. The agent starts with the probability $\mathrm{S}_0(s)$ in state $s$.

## 3.2. Reinforcement Learning

The task of Reinforcement Learning (RL) [25] is to find a policy $\pi : \mathrm{S} \times \mathcal{A} \rightarrow \mathbb{R}^{+}$ that maximizes the expected reward in the MDP. An agent following $\pi$ selects action $a$ in state $s$ corresponding to the probability density $\pi(s, a)$. To do so, the agent can interact with the environment with the actions $\mathcal{A}$ to optimize the expected reward via trial and error. Figure 3.1 shows the interaction between an agent and the MDP environment. So the learning algorithms do not need training data, it generates the data itself. Another characteristic of RL are delayed rewards. In contrast to other learning concepts like supervised learning, the agent only receives feedback through the rewards. The agent does not know whether the decision was the best one. The reward function weights transitions with a reward to describe the desired behavior, but it should not explain how to reach the behavior. This reward definition can be a problem in RL: If the task that should be solved gets more complex, it gets more difficult to define a suitable reward function for multiple desiderata. For example for robot walking there are multiple desiderata, such as keeping a velocity in the desired direction, walking in a periodical gait, preventing unnecessary movements of the motors, not falling et cetera. These complex goal behavior often leads to reward function fine tuning until the agent generates the wanted behavior [1]. This problem is addressed later in Section 3.3.

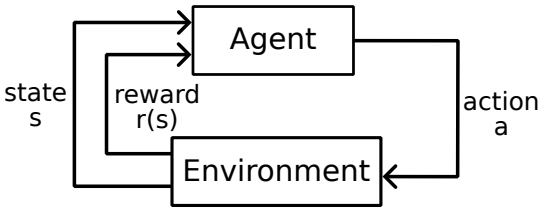



Figure 3.1.: Interaction between an RL environment and an agent. The agent observes the state and receives a reward. Depending on this observation, it chooses an action to interact with the environment. This environment is an MDP.

The maximum entropy Reinforcement Learning problem is to maximize the expected discounted reward under the discounted causal entropy $\mathrm{H}(\pi) = \mathbb{E}_{\pi}[-\log \pi(a \mid s)]$ corresponding to the policy $\pi$. It is defined as

$$\mathrm{J}(s) = \mathbb{E}_{\pi}[\mathrm{r}(s, a)] = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^{t} \mathrm{r}(s_t, a_t)\right] . \tag{3.1}$$

The desired policy is element of the policy space $\pi \in \Pi$. To get a policy that maximizes J from Equation (3.1), the following equation needs to be optimized [13]

$$RL(\mathrm{r}) = \arg\max_{\pi \in \Pi} \eta \mathrm{H}(\pi) + \mathbb{E}_{\pi}[\mathrm{r}(s, a)] , \tag{3.2}$$

while $\eta$ is a constant factor used in practice to prevent the Reinforcement Learning algorithms to overestimate the importance of the entropy. To solve the RL problem there are three main approaches: First, action-value methods are based on the Q-function $Q(s, a)$. The function evaluates each state $s$ and action $a$ with its expected discounted reward starting in state $s$ with action $a$ and following $\pi$ after that. Second, policy gradient methods try to optimize the policy function based on gradients. Third, actor-critic methods as a combination of these two approaches. This is the approach we employ in this thesis. Those methods use a Q-function as well as policy gradients. We utilize parameterized policies $\pi_{\boldsymbol{\theta}}$ for infinite state and action spaces. The RL algorithm tries to find a policy function with the maximum expected reward by finding the appropriate parameters $\boldsymbol{\theta}$. Additionally, the occupancy measure [13] $\rho_\pi : \mathrm{S} \times \mathcal{A} \to \mathbb{R}$ corresponding to a policy $\pi$ can be used. The measure is closely related to the distribution of the state-action pairs that occur when following policy $\pi$. It is defined by

$$\rho_\pi(s, a) = \pi(a \mid s) \sum_{t=0}^{\infty} \gamma^t \mathrm{P}(s_t = s \mid \pi) \ .$$

There is a one-to-one relationship between the occupancy measure $\rho_\pi$ and its corresponding policy $\pi$

$$\pi_\rho(a \mid s) = \rho(s, a) / \sum_{a'} \rho(s, a') \ .$$

So $\rho_\pi$ can be used to make statements about the policy and vice versa.
In the terms of RL, a trajectory is a sequence of successive states the agent receives when following $\pi$ and the corresponding rewards $\mathrm{r}_i$ in time step $i$. An important parameter to fine tune for the RL training is the learning rate $\alpha$. It determines how fast the agent learns, so how big the correction step based on the new knowledge is. With a too large learning rate the agent will overestimate the importance of the new data and the learning will probably not converge. If the learning rate is too slow the training takes longer than necessary.

## 3.3. Maximum Entropy Inverse Reinforcement Learning

Inverse Reinforcement Learning addresses the problem to design the reward function for complex tasks. It is a part of Imitation Learning, which tries to imitate a behavior from an expert agent $\pi_\mathrm{E}$. IRL assumes that the expert's behavior $\pi_\mathrm{E}$ tries to optimize an unknown reward function $\mathrm{r}_\mathrm{E}$. So basic IRL algorithms alternate between an estimation

improvement of the reward function and a calculation of its policy. The goal is to produce a policy that is at least as good as $\pi_{\mathrm{E}}$ on the expert's reward function $\mathrm{r}_{\mathrm{E}}$. Therefore IRL has a slightly different environment than RL: an MDP\R $(\mathrm{S}, \mathcal{A}, \mathrm{P}, \gamma, \mathrm{S}_0)$ denotes an MDP without a reward function.
The first IRL principle we employ later is from Ziebert et al.[30]. It uses the principle of Maximum Entropy for a probabilistic approach to Inverse Reinforcement Learning. One behavior can be described by multiple reward functions. To resolve this ambiguity Ziebert et al. use the principle of Maximum Entropy: they maximize the policy's entropy $\mathrm{H}(\pi)$ so they get the state distribution that has the single preference to match the occupancy measure of the expert. To find the reward function that has the smallest difference from the expected discounted reward of the expert, the IRL problem with Maximum Entropy is defined as [13]

$$IRL(\pi_{\mathrm{E}}) = \arg \min_{\mathrm{r} \in \mathbb{R}^{\mathrm{S} \times \mathcal{A}}} \left( \max_{\pi \in \Pi} \eta \mathrm{H}(\pi) \mathbb{E}_{\pi}[\mathrm{r}(s,a)] \right) - \mathbb{E}_{\pi_{\mathrm{E}}}[\mathrm{r}(s,a)] \; . \tag{3.3}$$

After the expert's reward function is calculated, an RL step (3.2) is needed to get the corresponding policy. So the desired policy is calculated by $RL \circ IRL(\pi_{\mathrm{E}})$.

## 3.4. Generative Adversarial Imitation Learning

The first learning algorithm, called Generative Adversarial Imitation Learning (GAIL), by Jonathan Ho and Stefano Ermon [13] connects Imitation Learning and Generative Adversarial Networks. We use this algorithms for the IRL training. It is model-free, sample efficient, and usable for learning complex behavior as well as for large, high-dimensional environments. The algorithm allows all possible reward functions $\mathrm{r} \in \mathbb{R}^{\mathrm{S} \times \mathcal{A}}$. Instead of learning the policy for the learned reward function until convergence per step, the approach by Jonathan Ho and Stefano Ermon alternates between one reward improvement step and one policy improvement step.
To do so, GAIL uses the concept of Generative Adversarial Networks [10]. It generates a new data distribution as close as possible to a given desired data distribution. That makes it usable for many kinds of model and optimization problems. The idea is to train a generator $\mathrm{G}$ to produce a new data distribution imitating the expert's distribution. At the same time it trains a discriminator $\mathrm{D}$ to differentiate between the expert's data distribution and the data distribution created by $\mathrm{G}$.
To prevent the agent from overfitting due to the large reward function space, the authors

employ an adaptive regularizer $\psi$ on the Maximum Entropy objective (3.3). So the new objective regularized with $\psi$ for IRL is

$$IRL_\psi(\pi_\mathrm{E}) = \arg \min_{\mathrm{r} \in \mathbb{R}^{\mathrm{S}\times\mathcal{A}}} \psi(\mathrm{r}) + \left( \max_{\pi \in \Pi} \eta \mathrm{H}(\pi) + \mathbb{E}_\pi[\mathrm{r}(s,a)] \right) - \mathbb{E}_{\pi_\mathrm{E}}[\mathrm{r}(s,a)] \ .$$

The one-to-one relation between a policy $\pi$ and the occupancy measure $\rho_\pi$ shown in Equation (3.1) allows to use the occupancy measure $\rho_\pi$ analogous to the data distribution in Generative Adversarial Networks, which the generator G tries to imitate. Thereby the Inverse Reinforcement Learning problem is transformed to an occupancy measure matching problem.
The new regularizer has to be suitable for large environments and achieve high accuracy for occupancy matching. To make these properties possible, $\psi$ adapts to the expert data

$$\psi_{GA}(\mathrm{r}) = \begin{cases} \mathbb{E}_{\pi_\mathrm{E}}[g(-\mathrm{r}(s,a))] & \text{if } \mathrm{r} > 0 \\ +\infty & \text{otherwise} \end{cases} \text{ with } g(x) = \begin{cases} -x - \log(1-e^x) & \text{if } x < 0 \\ +\infty & \text{otherwise} \end{cases} .$$

This definition little penalizes reward functions that assign high rewards to the expert's data and strongly penalizes reward functions that assign low rewards to the expert's data. Furthermore, it allows any reward function r as long as it is positive everywhere. With the convex conjugate $\psi^*_{GA}(\rho_\pi - \rho_{\pi_\mathrm{E}})$, the goal can be redefined as

$$RL \circ IRL_\psi(\pi_\mathrm{E}) = \arg\min_\pi \psi^*_{GA}(\rho_\pi - \rho_{\pi_\mathrm{E}}) \ .$$

With discriminator $\mathrm{D} : \mathrm{S} \times \mathcal{A} \to (0,1)$, the policies causal entropy H weighted by $\eta$ as policy regularizer, they get a new expression used for the optimization of D and $\pi$

$$\mathbb{E}_{\pi_\mathrm{E}}[-\log(\mathrm{D}(s,a))] + \mathbb{E}_\pi[-\log(1-\mathrm{D}(s,a))] + \eta \mathrm{H}(\pi) \ . \tag{3.4}$$

Since there are two objectives to improve, two learning rates needs to be fine-tuned. $\alpha_\pi$ weights the gradient improvement step on the policy and $\alpha_D$ weights the improvement step on the discriminator. GAIL uses a parameterized policy $\pi_{\boldsymbol{\theta}}$ with weights $\boldsymbol{\theta}$ and a parameterized discriminator $\mathrm{D}_{\boldsymbol{\omega}}$ with weights $\boldsymbol{\omega}$. These parameters ensure the policy and the discriminator are suitable for large environments. The authors alternate between an Adam gradient step on $\boldsymbol{\omega}$ to decrease the objective in Equation (3.4) and a Trust Region Policy Optimization (TRPO) step on $\boldsymbol{\theta}$ to increase Equation (3.4) with the reward function $\mathrm{r} = -\log(1-\mathrm{D}(s,a))$. The TRPO is a gradient-based RL method that additionally avoids too large policy improvement steps that could be caused by noise in the policy gradient. The result is the following specific algorithm. Notice that the trajectories $\tau$ approximate

the occupancy measure $\rho$ based on monte-carlo-rollouts because in practice, the exact $\rho$ are not given:

**input** : Expert trajectories $\tau_{\mathrm{E}}$, initial policy and discriminator parameter $\boldsymbol{\theta}_0, \boldsymbol{\omega}_0$

**for** *i = 0,1,2,...* **do**

Sample trajectories $\tau_i$ corresponding to $\pi_{\boldsymbol{\theta}_i}$

Update parameter $\boldsymbol{\omega}_i$ from D to $\boldsymbol{\omega}_{i+1}$ with gradient

$$\hat{\mathbb{E}}_{\tau_{\mathrm{E}}}[\nabla_{\boldsymbol{\omega}} - \log(\mathrm{D}_{\boldsymbol{\omega}}(s,a))] + \hat{\mathbb{E}}_{\tau_i}[\nabla_{\boldsymbol{\omega}} - \log(1 - \mathrm{D}_{\boldsymbol{\omega}}(s,a))] \; .$$

Take a policy step from $\boldsymbol{\theta}_i$ to $\boldsymbol{\theta}_{i+1}$, using the TRPO rule with reward function $-\log(1 - \mathrm{D}_{\boldsymbol{\omega}_{i+1}}(s,a))$. Specifically, take a KL-constrained natural gradient step with

$$\hat{\mathbb{E}}_{\tau_i}[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a \mid s) Q(s,a)] + \eta \nabla_{\boldsymbol{\theta}} \mathrm{H}(\pi_{\boldsymbol{\theta}}),$$
$$\text{where } Q(\bar{s},\bar{a}) = \hat{\mathbb{E}}_{\tau_i}[-\log(1 - \mathrm{D}_{\boldsymbol{\omega}_{i+1}}(s,a)) \mid s_0 = \bar{s}, a_0 = \bar{a}] \; .$$

**Algorithm 1:** Generative Adversarial Imitation Learning by Jonathan Ho and Stefano Ermon [13]. It alternates between an Adam gradient step on $\boldsymbol{\omega}$ and a TRPO step on $\boldsymbol{\theta}$.

Torabi et al. [27] additionally introduce an only-state version of GAIL called Generative Adversarial Imitation from Observation (GAILfO). This algorithm uses only the expert's states for training. In contrast to this version, the state-action algorithm above trains with states and actions. The goal is not to imitate exactly the same policy $\pi_{\mathrm{E}}$ and actions, but to imitate the expert's effect on the environment. This property makes it especially suitable for training with expert data generated by another model that may react differently to the same actions. They define a new state-transition reward $\mathrm{r}^{\mathrm{S}} : \mathrm{S} \times \mathrm{S} \to \mathbb{R}$, which rewards state-transitions based on how close they are to the expert's state-transition distribution. To do so they define the state-transition occupancy measure $\rho^{\mathrm{S}}_{\pi} : \mathrm{S} \times \mathrm{S} \to \mathbb{R}$

$$\rho^{\mathrm{S}}_{\pi}(s_i, s_j) = \sum_a \mathrm{P}(s_j \mid s_i, a) \pi(a \mid s_i) \sum_{t=0}^{\infty} \gamma^t \mathrm{P}(s_t = s_i \mid \pi) \; .$$

It is closely related to the distribution of state-transitions that occur when following policy $\pi$. Additionally, they introduce a slightly new regularizer adapted to the state-transition

reward

$$\psi^{\mathrm{S}}{}_{GA}(\mathrm{r}^{\mathrm{S}}) = \begin{cases} \mathbb{E}_{\pi_{\mathrm{E}}}[g(-\mathrm{r}^{\mathrm{S}}(s,s'))] & \text{if } \mathrm{r}^{\mathrm{S}} > 0 \\ +\infty & \text{otherwise} \end{cases} \text{ with } g(x) = \begin{cases} -x - \log(1-e^{x}) & \text{if } x < 0 \\ +\infty & \text{otherwise} \end{cases}.$$

And like before with the convex conjugate of this regularizer and a discriminator $\mathrm{D} : \mathrm{S} \times \mathrm{S} \to (0,1)$ over states, there is a new expression to optimize

$$\mathbb{E}_{\pi_{\mathrm{E}}}[-\log(\mathrm{D}(s,s'))] + \mathbb{E}_{\pi}[-\log(1-\mathrm{D}(s,s'))] + \eta \mathrm{H}(\pi)\ . \tag{3.5}$$

The discriminator tries to minimize the objective in Equation (3.5) and the TRPO step on $\boldsymbol{\theta}$ tries to maximize it. The algorithm is pretty similar to the pseudo-code 1 for GAIL, except that the discriminator is adapted.

## 3.5. Variational Adversarial Imitation Learning

One remaining challenge for adversarial learning remains to balance the generator and the discriminator. The training can be unstable because the discriminator can easily distinguish between the expert's and the generator's data. If that is the case the discriminator produces uninformative gradients for improvement. As a stabilization method, Peng et al. [22] propose to use an information bottleneck before the input of the discriminator. So the discriminator gets less information and has to concentrate on the most important differences resulting in more informative gradients. This information bottleneck is called Variational Discriminator Bottleneck (VDB). An Imitation Learning algorithm based on GAIL extended with the VDB is called Variational Adversarial Imitation Learning (VAIL). The following section explains the Variational Discriminator Bottleneck and how it is integrated into GAIL.

The discriminator is built of the encoder $\mathcal{E}$, the prior distribution $\chi$, the info-constraint $\mathrm{I}_{\mathrm{C}}$ and the decoder $\mathcal{D}$. An overview of the structure of the discriminator is given in Figure 3.2. The Encoder $\mathcal{E}$ maps the sample $(s,a) \in \mathrm{S} \times \mathcal{A}$ to a stochastic encoding $z \sim \mathcal{E}(z \mid s,a)$. The prior distribution $\chi$ maps the encoding $z$ to the wanted shape. Then a constraint $\mathrm{I}_{\mathrm{C}}$ on the mutual information between the encoding and the original features is applied. The larger $\mathrm{I}_{\mathrm{C}}$ is the more information the constraint lets through. The decoder $\mathcal{D}$ gets the stochastic encoding after the information constraint and trains to distinguish between the expert's data and the policy's data. This Variational Discriminator Bottleneck formulates a

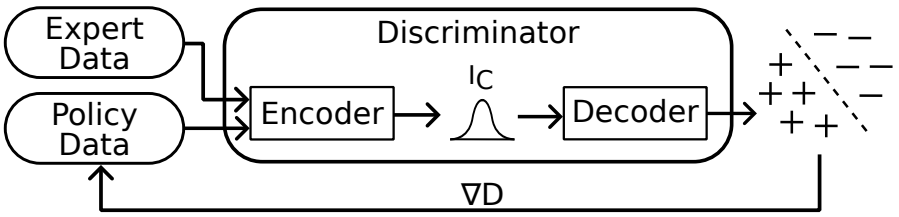



Figure 3.2.: The structure of the Variational Discriminator Bottleneck. The encoder maps samples from the expert and the trained policy. The information bottleneck reduces the information for the decoder, and the decoder tries to distinguish between the expert's data and the agent's data. The gradient of the discriminator is used to improve the agent. This figure is based on a figure of Peng et al. [22].

new objective for VAIL to optimize

$$\min_{\mathcal{D},\mathcal{E}} \max_{\beta \geq 0} \mathbb{E}_{\pi_\mathrm{E}} \left[ \mathbb{E}_{z \sim \mathcal{E}(z|s,a)} [-\log(\mathcal{D}(z))] \right] + \mathbb{E}_{\pi} \left[ \mathbb{E}_{z \sim \mathcal{E}(z|s,a)} [-\log(1 - \mathcal{D}(z))] \right] + \eta \mathrm{H}(\pi) + \beta \left( \mathbb{E}_{\tilde{\pi}} [\mathrm{KL}[\mathcal{E}(z \mid s,a) \mid\mid \chi(z)]] - \mathrm{I_C} \right) \ ,$$

with $\tilde{\pi} = 0.5\pi_\mathrm{E} + 0.5\pi$ representing a mixture of the learned policy and the expert policy and KL being the Kullback-Leiber divergence. That is the objective from GAIL (3.5) extended with the encoder $\mathcal{E}$ and the additional condition that $\mathbb{E}_{\tilde{\pi}}[\mathrm{KL}[\mathcal{E}(z \mid s,a) \mid\mid \chi(z)]] \leq \mathrm{I_C}$. To maintain this condition the authors use the Lagrange multiplier $\beta$. This multiplier must be adaptively updated to maintain the specific constraint $\mathrm{I_C}$. Therefore they apply a dual gradient descent

$$\beta \leftarrow \max \left(0, \beta + \alpha_\beta \left( \mathbb{E}_{\tilde{\pi}} [\mathrm{KL}[\mathcal{E}(z \mid s,a) \mid\mid \chi(z)]] - \mathrm{I_C} \right) \right) \ , \tag{3.6}$$

with $\alpha_\beta$ as the step size for the dual variable in the dual gradient descent. So after the gradient improvement step on $\mathcal{D}$, $\beta$ adapts with Equation (3.6). For the policy update, they take $\mathrm{r} = -\log(1 - \mathcal{D}(\mu(s,a)))$ as a reward while $\mu$ is the mean from the encoder distribution. That satisfies a simplified objective. It approximates the expectation over $z$ with $\mu$, which is sufficient for this task.
In practice, the encoder $\mathcal{E}(z \mid s,a) = \mathcal{N}(\mu(s,a), \sum_{\mathcal{E}}(s,a))$ is modeled as a Gaussian distribution with mean $\mu$ and diagonal covariance matrix $\sum_{\mathcal{E}}(s,a)$. The encoder $\mathcal{D}(z) = \sigma(w_\mathcal{D}^T + b_\mathcal{D})$ is realized by a sigmoid function with a linear input depending on weights $w_\mathcal{D}$ and $b_\mathcal{D}$. The prior distribution $\chi(z) = \mathcal{N}(0, I)$ is a standard Gaussian.
As for GAIL in Section 3.4, there is also a version of VAIL that trains with only states: Variational Adversarial Imitation Learning from Observation (VAILfO). The only difference is the encoder $\mathcal{E}$ maps samples $(s, s') \in \mathrm{S} \times \mathrm{S}$ to the stochastic encoding $z \sim \mathcal{E}(z \mid s, s')$.

# 4. Methodology

For Inverse Reinforcement Learning, we need to design the agent model, generate expert data, and embed it in the learning environment with the learning algorithms. How these components collaborate is visualized in Figure 4.1. We already explained the algorithms in the Preliminaries Section 3. In this chapter, we present the remaining requirements. We start with a specification of the Unitree A1 model and its state and action spaces in Section 4.1. To generate the demonstrations we use a Data Generation Pipeline and adapt a Model Predictive Control model. How we adapted it, what data we collect, and how this data is prepared for training is explained in Section 4.2. In the last Section 4.3, we describe the Inverse Reinforcement Learning framework, the model's environment and the physics simulator.

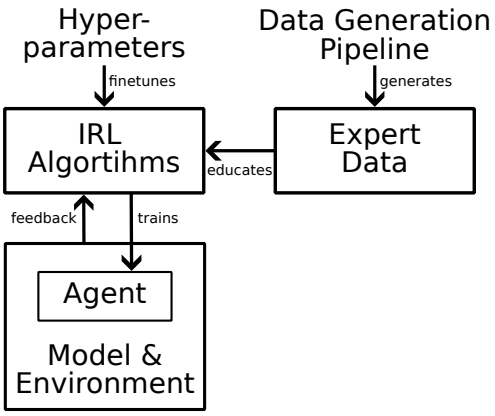



Figure 4.1.: The collaboration of the components of IRL. The IRL algorithms use expert data to train an agent that interacts through the model with the environment which gives feedback in form of observed states to the algorithms.

## 4.1. Specification of the Unitree A1

We use the MuJoCo [26] model of the Unitree A1 by Deepmind [6] as a foundation and adapt it to our needs. The quadruped model has 4 legs, each with 3 joints. These joints can

be divided into 3 different types that are shown in Figure 4.2: First, the abduction, which moves the whole leg laterally to the trunk with a range of $\mathrm{S}_A = [-0.802851, 0.802851]$. Then there is the hip, which moves the upper part of the leg forward and backward with a range of $\mathrm{S}_H = [-1.0472, 1.0472]$. And the last type is the knee that moves the lower part of the leg back and forth with a range of $\mathrm{S}_K = [-2.69653, 2.69653]$. In addition, we can observe the 3-dimensional position of the trunk. For training, we omit the x- and y-coordinates of the robot, to avoid unnecessary complex states and to allow the agent to train the gait independently of the x and y position. The range for the height is $\mathrm{S}_P = \mathbb{R}$. We observe the rotation of the trunk in the 3-dimensions with the range $\mathrm{S}_R = (-\pi, \pi]^3$, where $\pi$ denotes the ratio of a circle's circumference. Also important for the states are the velocities of the features just mentioned. Notice, the x and y velocities are important features to describe the state and that they are independent of the x and y position. Our velocity state space contains the change ratio of all previous features, including the x and y velocities of the trunk. The range is defined by $\mathrm{S}_V = \mathbb{R}^{16}$. So we can describe the current state $s \in \mathrm{S}$ of the robot, while the state space $\mathrm{S}$ is defined by the height and rotations of the trunk, the positions of the 12 joints, and the corresponding change rates of the previous features

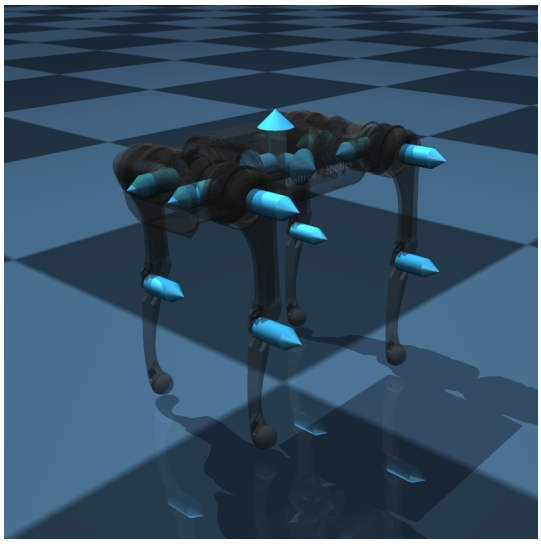


Figure 4.2.: The joints in the Unitree A1 model. In addition to the 3 joints per leg, this image shows the joints needed to observe the position of the trunk.

$$\mathrm{S} = \mathrm{S}_P \times \mathrm{S}_R \times (\mathrm{S}_A \times \mathrm{S}_H \times \mathrm{S}_K)^4 \times \mathrm{S}_V \ . \tag{4.1}$$

These 36 features describe our state space of the MDP\R mentioned in Chapter 3.3.
To interact with the environment the robot can move the 12 joints with actions $a \in \mathcal{A}$. We clip the actions for torque control as well as for position control to the range $[-1, 1]$ and scale the actions to the joint ranges. This way we can avoid having different action ranges per joint. Therefore we define our action spaces in both cases with

$$\mathcal{A} = [-1, 1]^{12} \ . \tag{4.2}$$

Additionally, we use a goal state space $\mathcal{G}$ to distinguish between different desired goals in training. Each separate goal has different values in the goal state so that the agent can differentiate between them and learn different behaviors for different goals. We distinguish between different direction angles in which the robot is able to walk and additionally store the desired velocity per direction. The maximum speed of the robot is $0.6$. To avoid a gap between the direction angles near to $\pi$ and $-\pi$, we represent the direction angle as the cosine and the sine of the angle. So we define the goal state space as

$$\mathcal{G} = [-1, 1] \times [-1, 1] \times [-0.6, 0.6] \,, \tag{4.3}$$

to define the desire behavior. The model is currently implemented in this Git repository[1], but it will be merged with the mushroom library [4] in the near future.

## 4.2. Data Generation Pipeline

As mentioned in Section 3.3, IRL needs demonstrations to learn from. So we need a dataset containing states in the state space $\mathrm{S}$ (4.1) as well as torque and position actions in the action space $\mathcal{A}$ (4.2) and goals in the goal space $\mathcal{G}$ (4.3). Therefore, we have a Data Generation Pipeline of four steps: We use an adapted MPC model to generate the gait, we add some noise to the actions to make the states more diverse, we scale the data to the desired control frequency, and we augment the dataset with different yaws. This pipeline is shown in Figure 4.3.

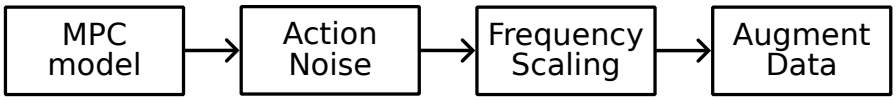



Figure 4.3.: The Data Generation Pipeline to create the expert demonstrations.

To generate the data, we use an already implemented Model Predictive Control model by Yuxiang Yang [28]. It is able to control a model of the Unitree A1 by desired velocity vectors and to store the desired states and actions. The MPC model runs in the pybullet [3] physics engine. That will be important later, because we use another physics simulator. Thus the model to create the data behaves different too. However, the model in its original form had some properties that could harm our training: First, the robot could not walk in

[1] `https://github.com/robfiras/mushroom-rl/tree/tim`

a straight line. The MPC model interprets the desired velocity vectors from the robot's point of view. In addition, there are small deviations between the desired velocity vector and the actual velocity vector. So if the model gets a constant desired velocity vector, the deviations will accumulate and the robot will walk in a big circle. This behavior results in too less similar state examples to learn the desired behavior and it is not the behavior the agents should imitate. Second, the model walks at different velocities in different directions. So the MPC model has a higher velocity when walking to the left than its velocity when walking to the right, despite the same desired absolute velocity.

This thesis implements a new controller on top of the existing one. This controller solves both of the problems. It takes a desired velocity vector and interprets it in the overall coordinate system. It also ensures that the robot's velocity is the desired velocity. We use the desired velocity vector to calculate a desired position. This position is defined by the x and y coordinates and the rotation of the trunk. In every time step, we obtain the next desired position by adding the velocity vector to the previous desired position, starting from the robot's initial position. Then we use three PID-controllers to calculate the velocities needed to reach this position. Thus, we are aiming for a straight line beginning at the initial position in the direction of the velocity vector. Additionally, our desired position depends only on the time and the maximum velocity, so for different directions, we want to be at the same distance. This approach also guarantees the same velocities although the MPC model interprets the commands differently. This process of calculating the next desired position is also visualized in Figure 4.4. For our demonstration data, we store the torque and position control actions (4.2), and all of the above mentioned features in the state space (4.1) at each time step. Additionally, we store the cosine and the sinus of the desired velocity direction angle and the absolute velocity of the robot for the goal states in (4.3).

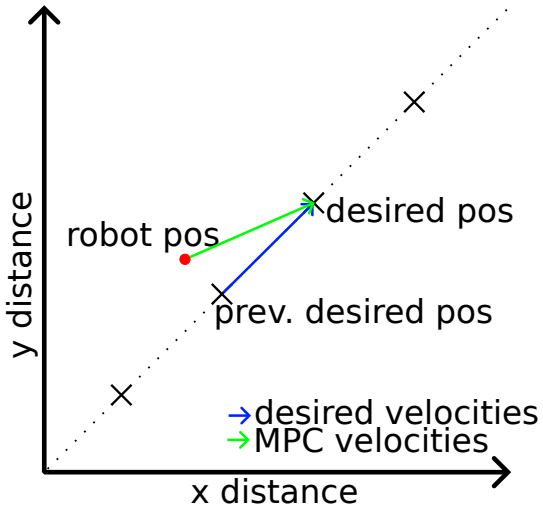



Figure 4.4.: The adaption on the MPC model. The required velocities are calculated with the previous desired position, the desired velocity and the robot's position.

To generate a more diverse dataset and to cover as many states as possible, we add some noise to the actions. So we make sure that the expert data is more widely distributed.

The noise is in the order of magnitude 1e-1 so that the MPC model is still able to build a stable gait. We always generate one trajectory without noise and 2 with different random noise per walking direction.
The next step is the interpolation of the dataset. The MPC model collects the data at a 500 $\mathrm{Hz}$ control frequency. We train our model with a control frequency of 100 $\mathrm{Hz}$. So we need to interpolate the states and actions from 500 $\mathrm{Hz}$ to 100 $\mathrm{Hz}$ with a third order spline interpolation.
We generate trajectories for 8 different directions with 50.000 data points per trajectory and 3 different noises per direction. Additionally we start in each trajectory in a default position. So we are able to learn the initialization of the gait as well.
The adapted MPC model is implemented at the Git repository [2] with the generated datasets we use for training.

## 4.3. The Inverse Reinforcement Learning Framework

To complete the description of our MDP\R environment, we still need to define the transition function $\mathrm{P}$, the discount factor $\gamma$, and the initial state distribution $\mathrm{S}_0$. We use the MuJoCo [26] physics simulator, so our $\mathrm{P}$ is defined by its physics. We use a discount factor of $\gamma = 0.99$. As an initial state, we draw a random state from a random trajectory of the expert data. To be sure that there are enough expert states left after this state, we draw the sample from the first $0.45$ part of the trajectory. The MuJoCo environment runs at a frequency of 1000 Hz, so it simulates the physics in 1000 steps per second. To simulate the ground contact, the environment simulates and observes the collision forces between the four feet and the ground.
We use the mushroom library [4] as our learning environment. It provides an interface to the MuJoCo simulator and the algorithmic environment to implement the described model of the Unitree A1. We use different models for torque control and position control. The model that can be controlled with torques has a 34 Newton meter action range. The position control model has a force range of -20 to 20 and uses a proportional gain of 100. The remaining gains are determined by MuJoCo. We create the Unitree A1 model with an arrow above its trunk to visualize the goal direction.
To reduce the complexity of the imitation training, we define absorbing states. In our case, the state is absorbing if the robot has fallen. We use the roll and pitch rotation of the trunk as well as the height of the trunk to define these absorbing states. The roll range

[2] `https://github.com/tja72/locomotion_simulation`

of the trunk in the demonstration data is approximately from $-11$ to $11$ degrees. When the trunk's roll is above $16$ degrees, the state is absorbing. The maximum observed pitch of the trunk in the expert data is about $7.6$ degrees. In this case we allow a $2.4$ degrees buffer before the state is interpreted as fallen. The minimum height in the data is $-0.197$. As soon as the height goes below $-0.24$, the state is absorbing. This limits the possible training space to this range.

# 5. Experiments and Results

In this chapter, we explain how we train the agents. First, we have to decide on the type of training, then we increase the complexity of the training task step-wise until the task is the final goal task. For the design decision experiments in the first two sections, the task is to walk a straight line. We compare agents trained with position control and torque control in Section 5.1. We weigh between the only-states algorithms and the state-action algorithms in Section 5.2. With these two comparisons, we decide on a setting for the rest of the experiments. In the last Section 5.3, we increase the complexity of the task to be learned. We start with walking in a straight line. After we get acceptable results for this task, the agents learn to walk in 8 different directions. While the agent's yaw remains the same, it walks forward, backward, sideways and diagonal. The last task is to walk in these 8 directions regardless of the trunk's yaw. We compare the results of all tasks and algorithms. In the end, we show that one agent fulfills the final goal of this thesis. The actual launcher files for the experiments are implemented here [1].

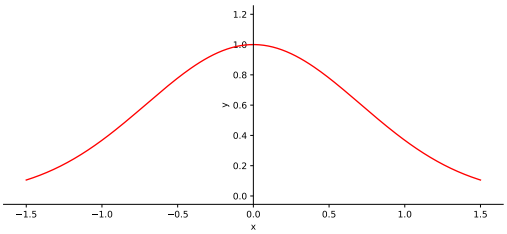



Figure 5.1.: The reward function depending on the error x.

We evaluate the learned agents by initializing a random position of the expert data and observing their behavior for 1000 steps. Additionally, we use a metric reward R during the training to roughly filter out bad agents. The difference between the desired velocity $v^*$ and the actual velocity $v_s$ per step tells us whether the robot is moving in the right direction. To have a reward range of (0,1] per step we weight the difference as follows: $r(v^*, v_s) = e^{-|v^* - v_s|^2}$, while $e$ denotes Euler's number. Figure 5.1

[1] https://github.com/tja72/IRL_unitreeA1

visualizes how this function weights the errors. We use the accumulated reward over 1000 steps to compare the agents

$$\mathrm{R} = \sum_{i=0}^{1000} r(s_i) \ , \tag{5.1}$$

for states $s_i$ observed while evaluating the agent. The plots in the following sections visualize the mean and the confidence interval of $\mathrm{R}$ of 5 different agents that are trained with the same settings, but with different seeds of randomness. The reward of an agent in an epoch is the mean reward of 25 trajectories performed by that agent with a maximum length of 1000. Note that the discounted reward $\mathrm{J}$ is a worse metric because it forgets past behavior. Importantly, the metric we use is not the true reward that we want to learn. So if $\mathrm{R}$ is high, it does not mean that the evaluated agent is similar to the expert. It is possible that the agent is moving at the right speed in the desired direction, but has a bad gait. For example, it could be doing mini-steps or walking on its knees. But we want to learn the gait from the expert data, so these results are not desired. Therefore we use $\mathrm{R}$ only as a metric, not as a reward for Reinforcement Learning. The mean length of the trajectories generated by the 5 agents are for clarity in the Appendix A.

We use a second metric to visualize the quality of the agent. Since the first metric $\mathrm{R}$ (5.1) tells us if the robot is moving in the right direction with the desired velocity, the second metric addresses the quality of the gait. The height of the feet visualizes the movement, the periodicity, and the stability of the gait. To get comparable results, this metric is always evaluated for walking in a straight line from the starting position. So it also includes the gait initialization. Since we use a different model and a different physics simulator, the expert's foot height in Figure 5.2 is not reachable for a stable agent in our environment. For example, we use more elastic feet, so the default position without movement is lower than for the MPC model. The red line visualizes this default position for our model. The image of the expert's

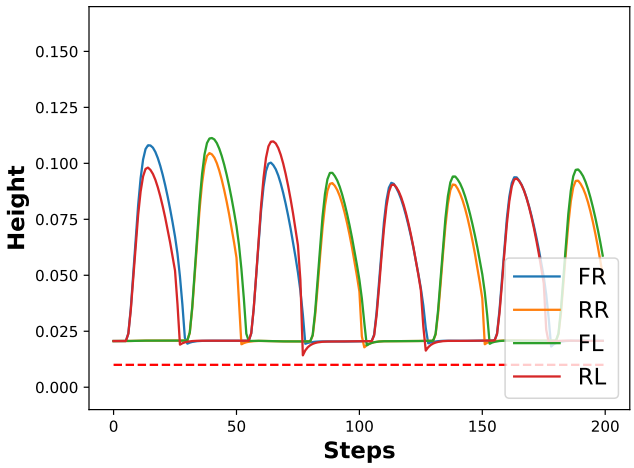



Figure 5.2.: The height of the expert's feet. This graph is not reachable by a stable agent in our environment.

foot height is an orientation of what to expected. The legend describes the position of the foot, the first letter stands for front (F) or rear (R), and the second letter stands for right (R) or left (L). We refer to a good gait or agent if the agent is able to produce a stable, periodic gait that gets a high reward and is similar to the expert. A bad agent or gait is an agent that produces a gait with a low reward and not periodic movement like double steps or small jumps.
We fine-tuned the hyperparameters for GAIL, VAIL, GAILfO, and VAILfO. For all experiments, we evaluate 5 different seeds with 300 epochs each. One epoch contains 100.000 policy improvement steps and each improvement step is performed with 1.000 states. For every 3 policy improvement steps, we improve the discriminator. We store and compare the best agents according to our metric (5.1) for every 50 epoch intervals. The learned agent is evaluated with the average reward over 25 runs. The discount factor is $\gamma = 0.99$. GAIL and GAILfO use a policy learning rate of $\alpha_\pi = 1e-4$ and a discriminator learning rate of $\alpha_D = 5e-5$. Both of them also use an entropy weight for the discriminator entropy with a value of $1e-3$. VAIL and VAILfO use a policy learning rate of $\alpha_\pi = 1e-4$ and a discriminator learning rate of $\alpha_D = 5e-5$. The information constraint is $\mathrm{I_C} = 1$. We only evaluate the stochastic policies, because the graphs and agents are better than for the deterministic policies.

## 5.1. Torque versus Position Control

Since we have two different control methods, we want to decide on one of them for the rest of the training. In this section, we compare the results of GAILfO and VAILfO with torque and position control. The next Section 5.2 shows that the state-only training is more stable which is why we only use these algorithms for comparison here. The goal is to walk in a straight line. The experimental parameters are described in the chapter's introduction.

The cumulative reward of all 4 experiments is shown in Figure 5.3. We can see that VAILfO learns quicker to reach a high reward than GAILfO. That was to be expected since VAILfO is forced by the info constraint $\mathrm{I_C}$ to focus on the most important features. So it learns quicker to imitate a similar behavior and to move in the right direction. Both position control experiments reach a lower maximum reward than the other two torque control experiments. Figure A.1 shows that VAILfO with position control has trajectories with the same length as torque control. That means position control is worse in walking with the desired speed in the desired direction than torque control. GAILfO with position

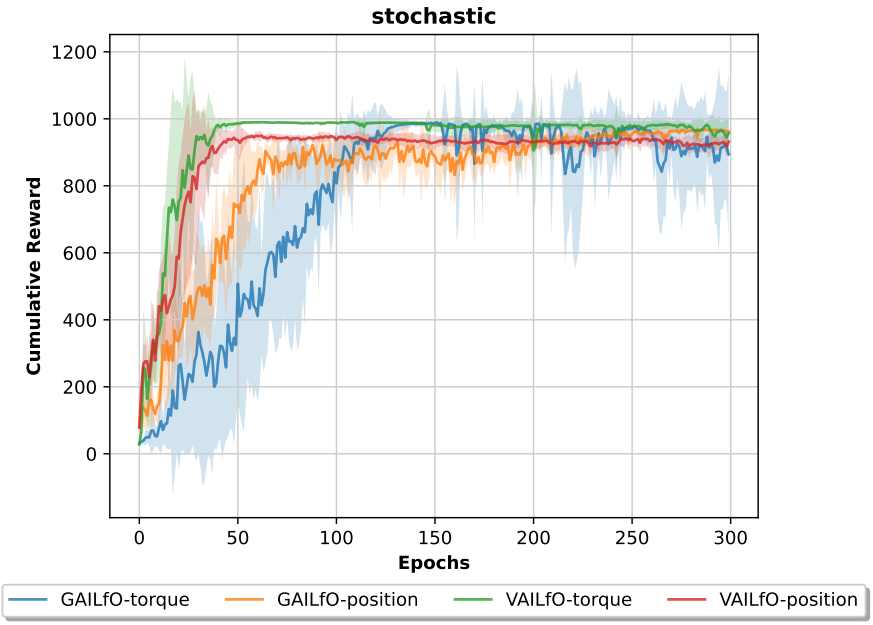
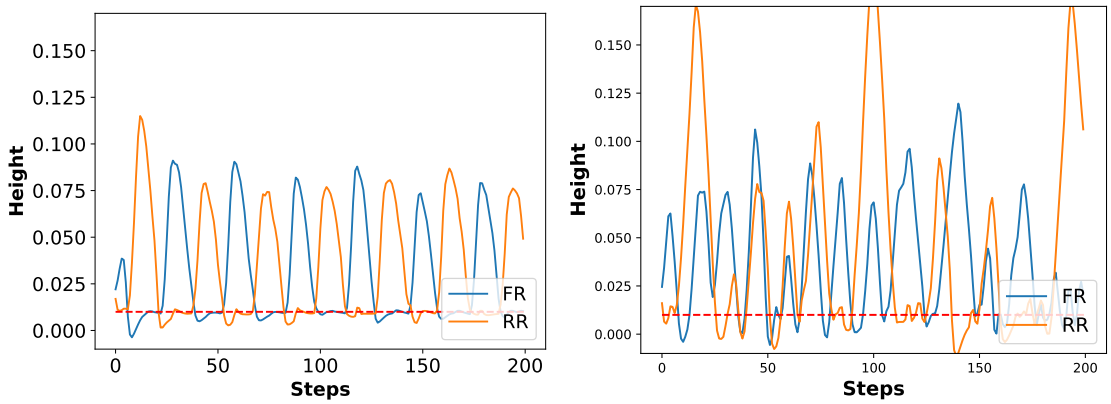



Figure 5.3.: The mean accumulated reward for GAILfO and VAILfO with torque and position control over 300 epochs.

control needs much more time to reach a the maximum length than torque control. So it learns a behavior that tends to fall in the first 200 epochs. The agents with torque control generate good gaits for GAILfO and VAILfO. In contrast, the agents with torque control are very unstable. They lift their feet for an expert like gait, but every movement is shaky and insecure. The feet vibrate with each movement. The agents can move in the right direction at the right speed, but their feet shake so much that they fall in some trajectories. VAILfO learns quicker not to fall but still has vibrating feet. Figure 5.4 visualizes the effect by showing the height of the right feet. The good agent with torque control on the left has a periodical foot movement and does not lift its feet above the default height if not for a step. The position control agent on the right has nearly no periodical movement and lifts multiple times both feet above the ground. That is not the behavior we desire.
So the torque control model provides more stable and better results. We use this kind of control method for the rest of the experiments.




(a) An GAILfO agent at epoch 163 trained with torque control.

(b) An GAILfO agent at epoch 152 trained with position control.

Figure 5.4.: Comparison of the feet heights of a torque control agent and a position control agent. The red line denotes the default height of our model. To maintain clarity only the right feet are shown.

## 5.2. Only-States versus State-Action Training

The previous Section 5.1 reasons for our decision for the torque control model. In this section, we compare the training with only states of GAILfO and VAILfO with the training with states and actions of GAIL and VAIL. We highlight at this point again that the expert's physics engine and the expert's model are different from our setting. So the exact states are probably not reachable for the same agent in different environments and the actions can cause different reactions.

Figure 5.5 shows the accumulated reward R for both learning algorithms with state-only training and state-action training. As in the previous section, VAIL and VAILfO learns quicker than GAIL and GAILfO. All agents reach the maximum cumulative reward. Both of the state-action experiments reach the maximum reward quicker than the state-only agents. State-only training has to learn without any foreknowledge which actions cause which reaction. In contrast, state-actions training can use the expert's actions as an orientation to find the right actions. That's probably why the state-action training reaches the maximum reward more quicker. There are no significant differences between the mean accumulated reward and the mean length of trajectories from the learned agents in Figure A.2. So the lower reward is caused by behavior that tends to fall. As the previous section mentioned, the state-only agents for straight walking show results similar to the expert

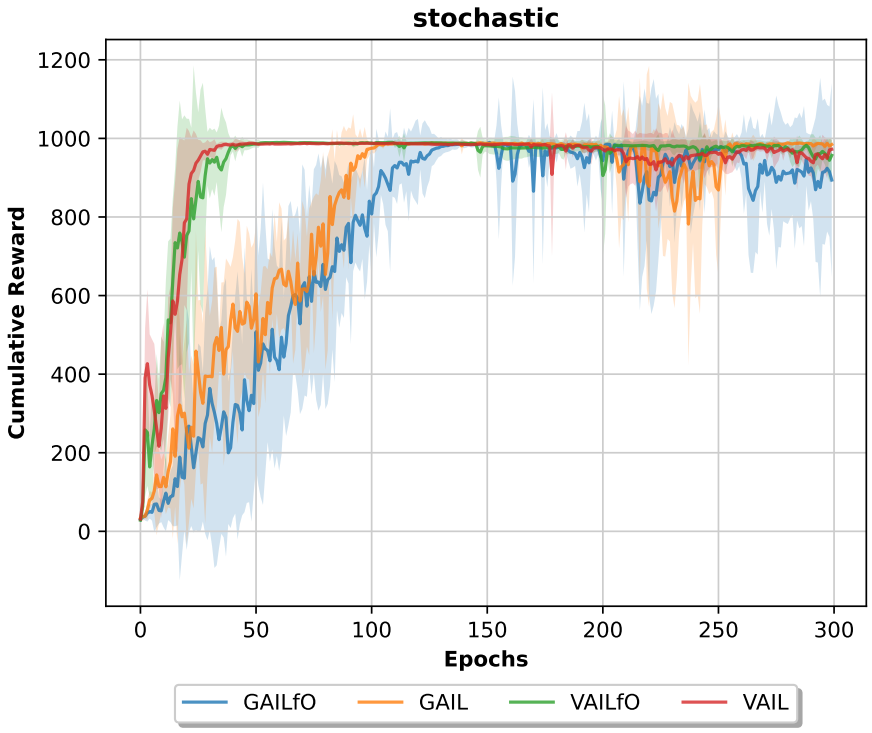
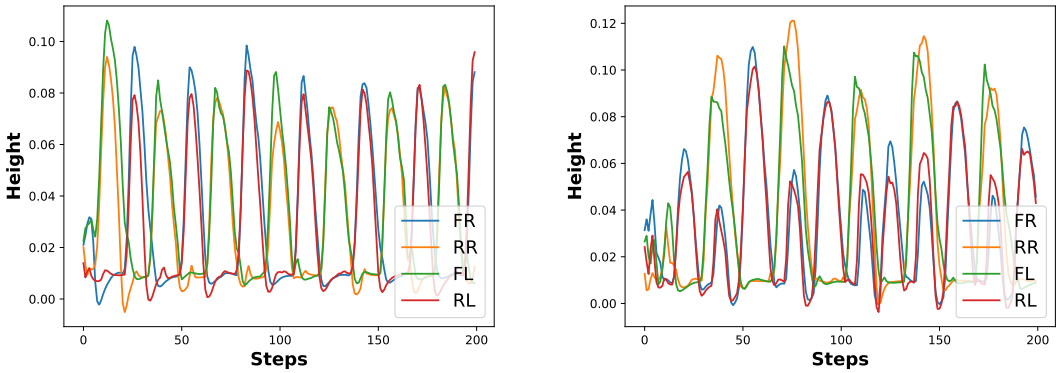



Figure 5.5.: The mean accumulated reward for GAILfO, VAILfO, GAIL, and VAIL over 300 epochs.

for GAILfO and VAILfO. In the contrast, most of the GAIL agents make little jumps after every step. One of these jumps is visualized in Figures 5.6. The worse agent from GAIL has multiple times all feet above the ground, while GAILfO producs periodical and stable movement with one foot at the ground all the time. Additionally, the gait initialization of GAIL is bad, so the robot falls very often. The algorithms also want to match the actions, not only the states. Since the actions are not usable in our environment, it seems that causes wrong behavior. VAIL generates similar good agents to VAILfO. We assume VAIL learns to concentrate on the differences between the states. That would result in training similar to state-only training.
We want to compare different learning algorithms in the other tasks. Since the GAILfO has a better gait than GAIL and VAILfO and VAIL agents have a similar quality, we decide on state-only, torque control training.




(a) The feet heights of a GAILfO agent. (b) The feet heights of a GAIL agent.

Figure 5.6.: The height of the four feet of the robot over 200 simulation steps. On the left is a good agent. On the right is an agent that makes little jumps. At steps 60, 100, 140, and 175 no feet of the GAIL agent touches the ground at a height of 0.01. We leave out the red line to highlight the jumps of the bad agent

## 5.3. Different Tasks

After the explanation of why we choose state-only, torque control training, we want to train an agent that can perform the final goal behavior. It should be able to walk in 8 different directions, regardless of the yaw of the trunk. To prevent mistakes in the model, fine-tune the parameters and control the complexity of the task, we divide the goal task into multiple smaller ones. The first task is to walk a straight line. The second task is to walk in 8 different directions with always the same yaw rotation. So it learns different gaits for walking forward, backward, sideways, and diagonally. The last task is to train an agent that is capable of walking in 8 directions, regardless of the trunk's yaw.

### 5.3.1. Walking in a Straight Line

The last two sections already compared the torque control, state-only agents to the other approaches. This section analyzes the results of GAILfO and VAILfO as well as the quality differences between multiple epochs.

According to the accumulated reward in Figure 5.7, GAILfO learns to move in the right direction with the right speed from epoch 100. Between the learning epochs 100 and

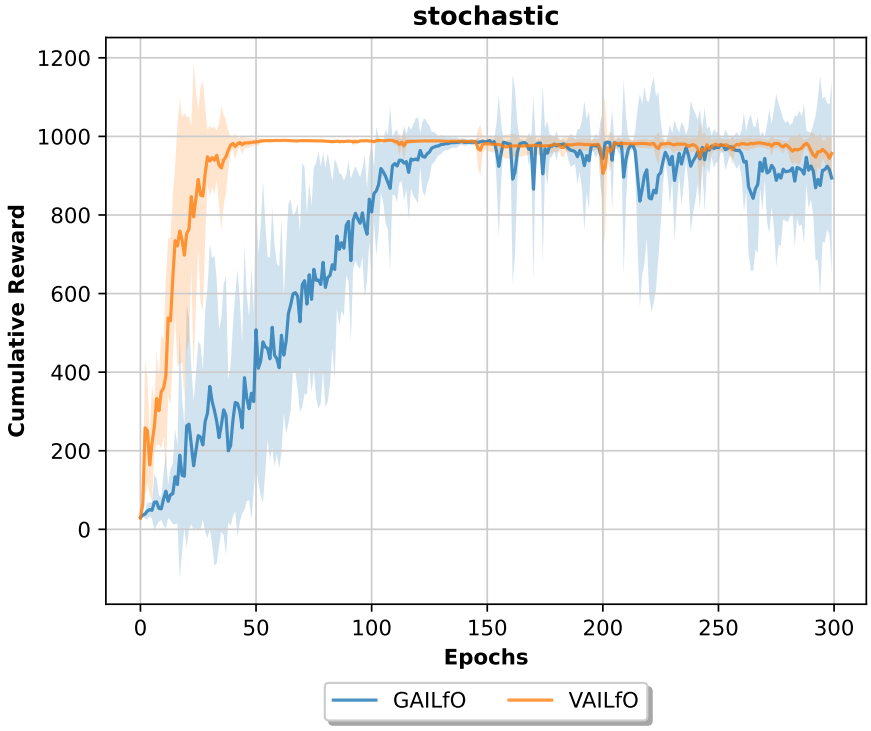
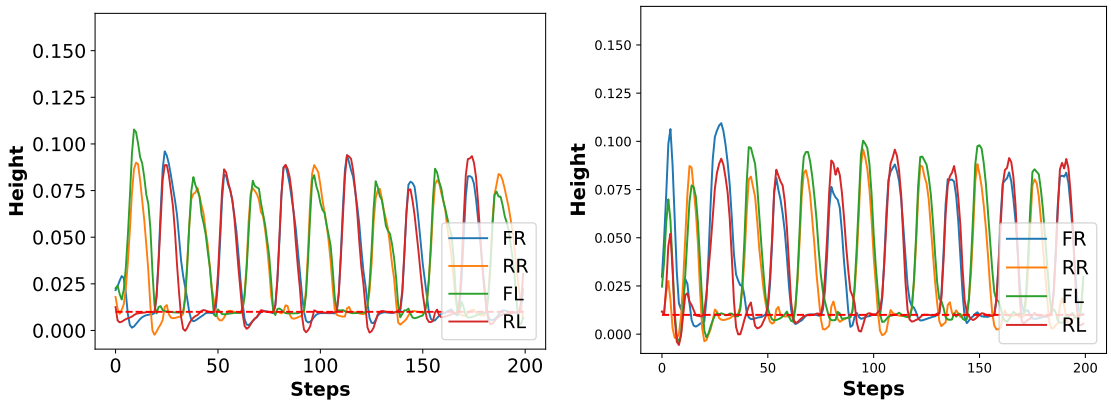



Figure 5.7.: The mean accumulated reward for GAILfO and VAILfO over 300 epochs. The goal is to walk in a straight line.

200, the accumulated reward is mostly at a high level, although the reward is not stable. From epoch 200 to the end of the training, the agents get a less constant reward. Figure A.3 helps to understand why these agents get a lower reward. Their average length per trajectory when evaluating the learned agents is not constant at the maximum of 1000. That means the agents learn a behavior that is more likely to fall. When testing the agents with the highest reward in these last 100 epochs, we observe a higher rate of agents that tend to make double steps. A double step describes a ground contact where the foot bounces off the ground and makes a mini step. That may help the agents to prevent behavior that causes them to fall in the previous epochs. The agents between epoch 100 and epoch 250 from GAILfO are good. Most of them generate a periodic gait without double steps and with large enough steps. Some of them are not as good as others but there is no recognizable pattern in the quality of the agents. Figure 5.8 compares two different agents. 5.8a shows the feet height of a GAILfO agent at epoch 163 and 5.8b




(a) GAILfO agent at epoch 163.

(b) GAILfO agent at epoch 208.

Figure 5.8.: The height of the four feet of the robot over 200 simulation steps. On the left is a good agent. On the right is an agent that makes double steps. The bad agent exceeds the default height of the ground multiple times while these feet should stay on the ground.

shows the same of an agent at epoch 208. In the latter graph, the feet are more often above the ground while they should stay on the ground. That's the behavior we name double steps. At this level of complexity, these double steps are small and only slightly noticeable. But we can also see in Figure 5.8a that GAILfO produces a very nice, periodical gait with a good initialization.

The agents from VAILfO receive a mostly constant high reward between epochs 40 and 275. Although the length of the trajectories stays the same, the reward gets lower after epoch 275. That means the agents in this range do not walk with the desired speed or they do not walk in the desired direction. The agents between epochs 50 and 150 generate the most periodic and stable gaits. The agents from epochs 150 to 250 are also good but tend to make smaller steps than the previous agents. After epoch 250, more agents tend to make double steps or they make smaller steps than necessary. The agents before epoch 50 are more likely to vibrate during their steps and some of them do not generate a gait at all.

In summary, the GAILfO and the VAILfO agents are equally good. They generate a stable gait with periodic movement. Both algorithms tend to generate less stable agents at the end of the training, even though VAILfO gets still a high reward.

### 5.3.2. Walking in 8 Different Directions

For the next experiments, we increase the number of possible goal states. The task is to learn 8 different gaits in different directions. The robot's desired yaw stays the same during this task. That means we want to learn a gait for walking forward, backward, sideways, and diagonally. So every direction has a different goal state with different desired angles and corresponding maximum velocities. This new task increases the complexity of the behavior to be learned.

The cumulative reward in Figure 5.9 shows that GAILfO generates agents with a good

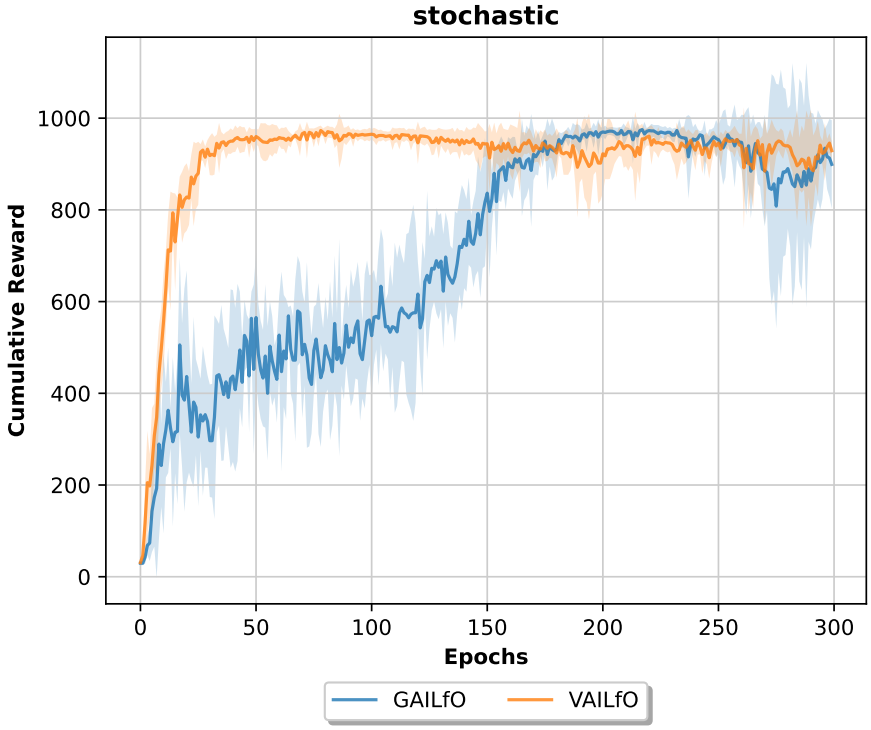
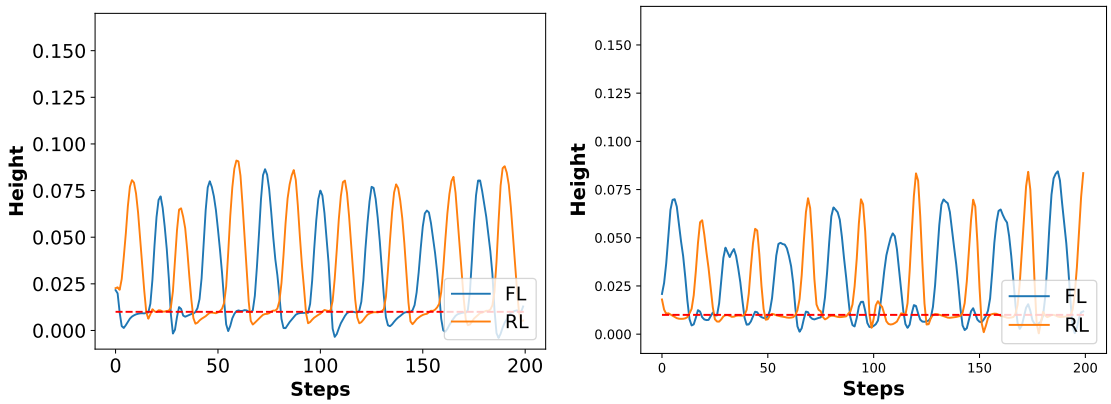



Figure 5.9.: The mean accumulated reward for GAILfO and VAILfO over 300 epochs. The goal is to walk in 8 different directions.

reward at epoch 175 and above. The agents get the highest reward between epochs 175 and 250. After epoch 250 the reward decreases and has a larger confidence interval. The length in Figure A.4 shows the same properties. That means the reward decreases because the agents learn a behavior that is more likely to fall. Overall the maximum reward is

lower than in the previous experiments. Since the complexity of the problem is increased, that was to be expected. The length in this range is at the maximum, so the lower reward has to be caused by slower agents or agents that do not walk perfectly in the desired directions. In comparison to the previous experiment, GAILfO learns slower to reach a high reward. The evaluation of the learned agents reflects these results. The best agents are between epochs 150 and 250. After that, most of the agents unlearn their gait. The agent limps, moves not periodically, or makes double steps. Overall the agents in the epoch range 150 to 250 are worse than before. More agents learn to make double steps or have vibrating feet during their movement. But some of these agents are still good. As in the experiment before they can produce a stable and periodical gait in all 8 directions. According to Figure 5.9, the VAILfO agents reach their average maximum reward between epochs 50 to 150. After that the reward has a bigger confidence interval and the average reward decreases. The length in Figure A.4 stays at most of the time at its maximum and it is more unstable than in the previous experiment. That means the lower and more unstable reward after epoch 150 is caused by a learned behavior that moves slower than desired or in the wrong direction. In comparison to the last experiment, the reward and the length are more unstable. As for GAILfO, VAILfO reaches a lower maximum reward in comparison to walking in a straight line. The agents from VAILfO make in the first 100 epochs very small steps and vibrates to the goal. That causes a high reward but is not our desired behavior. After that, it starts to learn the expert's behavior and makes bigger steps. But 3 out of 5 agents are still insecure and make a lot of double steps. They move slower in the right direction, which results in a lower reward. But there are also some good agents. After epoch 250 the agents produce undesired, not periodical gaits with little jumps and double steps. They move way slower in the desired directions, and sometimes they prefer to stay at a place instead of risking to fall. In Figure 5.10, a bad agent at epoch 292 with double steps is compared with a good agent with a nice gait at epoch 121. Figure 5.10b shows that the feet of the worse agent exceed the default height every time they touch the ground before they go down to the default height again. The behavior of the bad seeds could be explained by the survival bias. Algorithms like GAILfO and VAILfO prefer an agent that stays alive over one that imitates the expert's behavior but falls. So these mini steps help the agents not to fall, but they do not produce the desired behavior.

The best agents of both algorithms are comparable again. But overall GAILfO generates more agents that are closer to the expert's behavior than VAILfO. Both algorithms reach a little lower reward but since the complexity increased that is acceptable.




(a) VAILfO agent at epoch 121.

(b) VAILfO agent at epoch 292.

Figure 5.10.: The height of the two left feet of the robot over 200 simulation steps. On the left is a good agent. On the right is an agent that makes double steps. The bad agent exceeds the default height of the ground multiple times while these feet should stay on the ground.

### 5.3.3. Walking in 8 Different Directions regardless of the Yaw

We continue to increase the complexity of the desired behavior by adding trajectories with random yaw. So the agents should be able to perform gaits for 8 different directions, independent of the robot's yaw. In practice, we add a random yaw rotation in the discriminator improvement step to the expert data and initialize the agent with a random yaw, instead of producing a larger dataset.

Figure 5.11 shows a good reward for GAILfO between epochs 190 and 265. Before and after these epochs the reward decreases. Figure A.5 shows a very similar graph for the length of the trajectories. That indicates the lower reward is caused by behavior that tends to fall. The graphs indicate a similar behavior to the previous experiments, except that the agents learn a little slower and have a slightly lower maximum reward. Additionally, the reward decreases stronger than before after episode 265. The agents between epochs 150 and 200 make very small steps and the feet vibrate during every movement. All of the agents remove most of the vibration till epoch 260, but some agents still tend to make some undesired behavior like double steps or a small vibrating. The best and most stable agents are in epochs 200 to 265. Most of them generate a stable, periodical walk in all directions, regardless of the robot's yaw. Since some agents in this area also tend to make very small steps, the reward is lower than in the previous experiment.

VAILfO does not get any good cumulative reward according to Figure 5.11. The reward

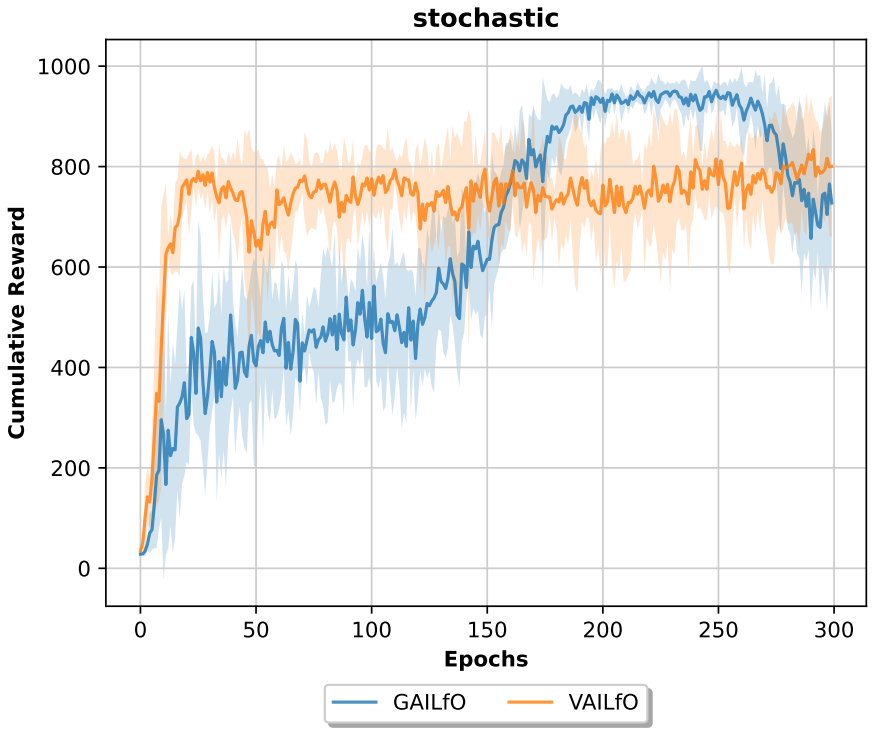



Figure 5.11.: The mean accumulated reward for GAILfO and VAILfO over 300 epochs. The goal is to walk in 8 different directions, independent of the yaw.

has a maximum of 800. The length of the trajectories in Figure A.5 has its maximum at the beginning, but quickly decreases and is very unstable after that. Till epoch 50, all agents are very insecure. They do not move in any direction and only vibrate with their feet. That behavior explains the low reward and the high length. The longer the agents train, the smaller the vibration in their feet becomes. In the range of 100 to 200 epochs, all agents have less vibrating but still do not move in the right direction. They only lift their feet but put them down again immediately. It looks like they are over careful. After epoch 250 some agents start to develop a recognizable gait. They make very very small steps in the right direction, but they sometimes get stuck in a position or fall. Since not all agents learn this behavior and these agents fall sometimes the average reward does not increase.
Because the default parameters do not deliver good results, we also try other combinations. Figure 5.12 shows that none of them reaches a good result. We use a higher and a

lower info constraint of the VDB. Because the complexity and the dataset size of the task increased, it may be possible that the discriminator has too much or too less information to learn from. Additionally, we compared training with different learning rates: all of them result in a similar low reward. We used an increased and decreased discriminator learning rate because the learning speed of the discriminator is critical to get good gradients to improve the policy. Also, a lower policy learning rate with a lower discriminator learning rate does not result in a good reward.

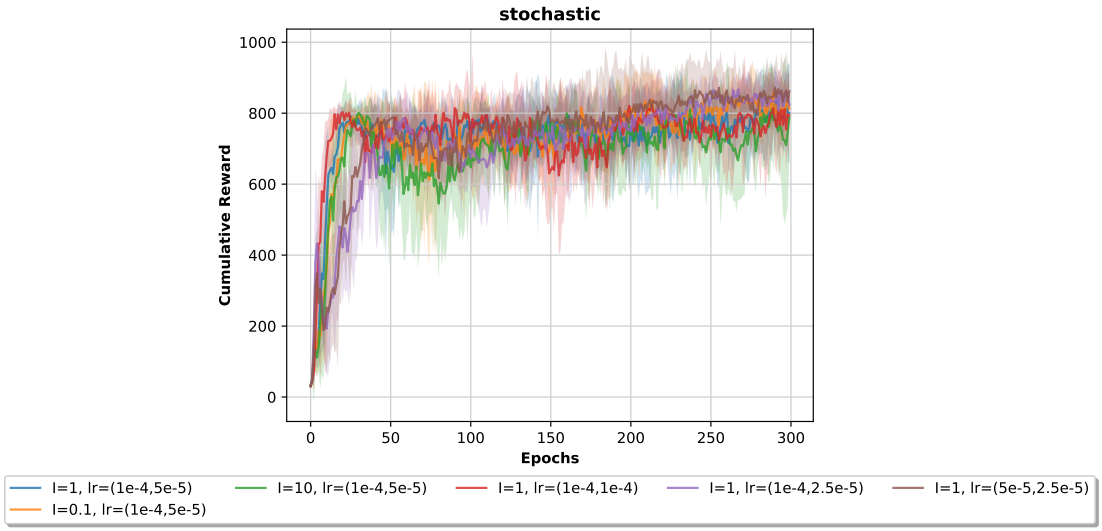



Figure 5.12.: The mean accumulated reward for a variation of VAILfO parameters. The goal is to walk in 8 different directions, independent of the yaw. The I in the legend describes the information constraint $\mathrm{I_C}$, the second letter the learning rates of the policy $\alpha_\pi$, and the discriminator $\alpha_D$

In summary, both algorithms get a lower reward than in the previous experiments. But in contrast to VAILfO, GAILfO is able to train some good and stable agents that can perform the desired task with a periodical gait.

The best agent that learned the desired behavior is from GAILfO at epoch 259. The agent has a mean reward of 969 over 50 runs in different directions with different yaws. The corresponding mean trajectory length is 1000. Figure 5.13 shows a periodic movement of the feet for walking in a straight line. But the agent makes some slight double steps that

are barely noticeable. We propose an idea how to make the training less complex and probably improve the gait in the future work Section 6. Figure 5.14 is a series of images of the agent walking to the right. This agent fulfills the goal of this thesis with a stable, periodic gait.

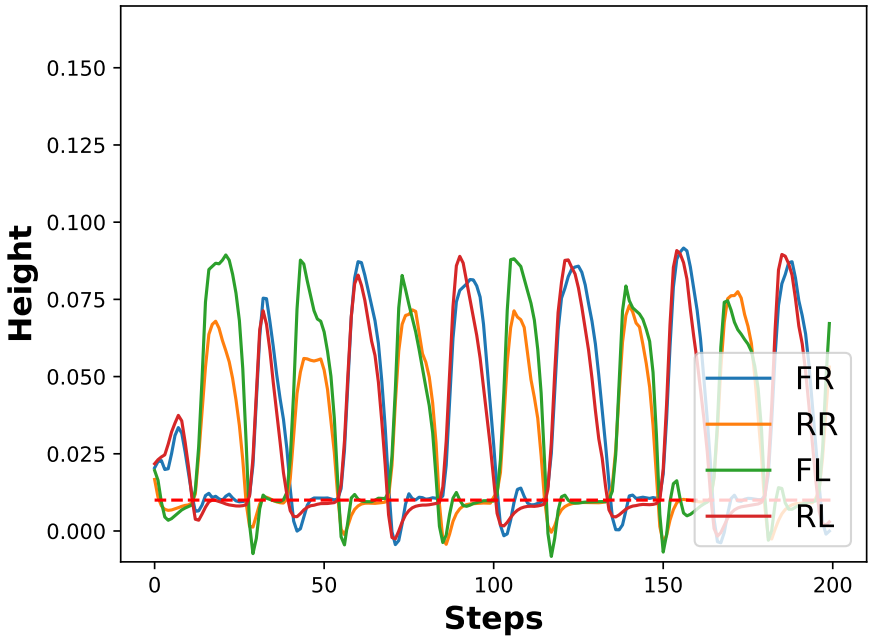



Figure 5.13.: The feet heights for the best agent. The agent walks in a straight line and initializes the gait.

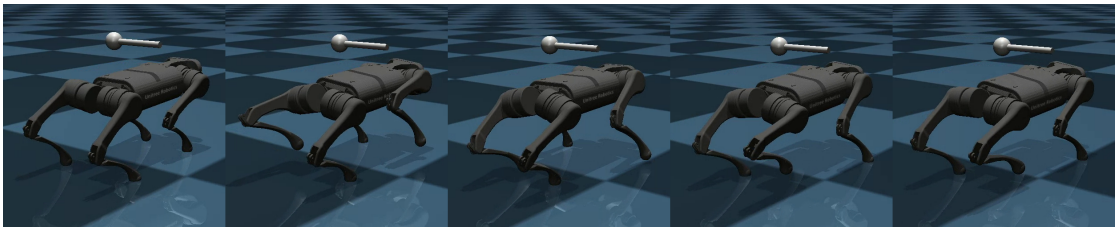


Figure 5.14.: A picture series of the final agent walking to the right.

# 6. Conclusion and Outlook

In this thesis, we describe how to imitate locomotion on the Unitree A1 robot using Inverse Reinforcement Learning.
We implement two models of the Unitree A1 in the mushroom environment. One model uses torque control and the other model uses position control. We define the possible states and actions and use a goal space to distinguish between different desired behaviors. Our Data Generation Pipeline uses an adapted Model Predictive Control model to produce the demonstration data. It generates trajectories with different noise, scales them to the desired control frequency, and augments them with different yaws of the trunk.
We define the Markov Decision Process without reward function as the learning environment. For the specification of the absorbing states, we examine the expert data and add a buffer to the roll, pitch, and height of the maxima.
We use the algorithms Generative Adversarial Imitation Learning and Variational Adversarial Imitation Learning for training. After the comparison of different control methods and different learning sets, we decide to train the torque control model with states only. With this setting, we increase the complexity of the tasks step-wise: We start with learning to walk straight forward, continue with walking in 8 different directions, and end up with walking in 8 different directions independent of the robot's yaw.
We show that we obtain an agent with GAIL that can perform this tasks with a periodical and stable gait similar to the expert.

In future work, we want to further fine-tune the hyperparameters of VAIL. In addition, we will remove the robot's yaw from the observation space to decrease the complexity of the problem to obtain similar or better results. Then, we will increase the complexity of the task further and fine tune the experiment settings until a good agent is retrieved. At the moment the robot can walk in straight lines in 8 directions with a constant speed. The first step will be to vary the desired velocity. Then we can add a new goal state, the desired yaw velocity of the robot. So the task will be to learn to walk curves. The last step will be to combine all the previous different gaits to switch between the goals.

Thus, instead of pursuing only one goal per trajectory, we want to change it during the execution. Additionally, the resulting agents can be tested on the real Unitree A1 robot once the torque interface is implemented.

# Acknowledgments

I am very grateful for the help and support of my advisors Firas Al-Hafez and Davide Tateo. They accompanied me through this thesis, answered patient to all my questions and gave helpful advise. I am thankful for the opportunity to work on this topic offered by Firas and Davide in collaboration with Jan Peters. I could learn a lot about Machine Learning and Robotics. I wish them all the best in the future.
The computations were performed on the Lichtenberg Cluster of the TU Darmstadt.

# Bibliography


[1] Pieter Abbeel and Andrew Y. Ng. "Apprenticeship Learning via Inverse Reinforcement Learning". In: *International Conference on Machine Learning (ICML)*. Association for Computing Machinery, 2004. URL: https://doi.org/10.1145/1015330.1015430.

[2] Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer science & business media, 2013. URL: https://books.google.de/books?hl=de&lr=&id=tXZDAAAAQBAJ&oi=fnd&pg=PA13&dq=Model+predictive+control&ots=L3QnmwxVjV&sig=Z9i9Cdx38cJRuo0h58V9zjkD5d8#v=onepage&q=Model%5C%20predictive%5C%20control&f=false.

[3] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. 2016–2021. URL: http://pybullet.org.

[4] Carlo D'Eramo et al. "MushroomRL: Simplifying Reinforcement Learning Research". In: *Journal of Machine Learning Research* 22 (2021). URL: http://jmlr.org/papers/v22/18-056.html.

[5] Avik De and Daniel E Koditschek. "Vertical hopper compositions for preflexive and feedback-stabilized quadrupedal bounding, pacing, pronking, and trotting". In: *The International Journal of Robotics Research* 37 (2018). URL: https://journals.sagepub.com/doi/pdf/10.1177/0278364918779874.

[6] Deepmind. *Unitree A1 Description (MJCF)*. 2022. URL: https://github.com/deepmind/mujoco_menagerie/tree/main/unitree_a1.

[7] Yanran Ding et al. "Representation-Free Model Predictive Control for Dynamic Motions in Quadrupeds". In: *IEEE Transactions on Robotics* 37.4 (2021). URL: https://ieeexplore.ieee.org/abstract/document/9321699.

[8] Peter Fankhauser et al. "Robust Rough-Terrain Locomotion with a Quadrupedal Robot". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018. URL: https://ieeexplore.ieee.org/abstract/document/8460731.

[9] Divyansh Garg et al. “IQ-Learn: Inverse soft-Q Learning for Imitation”. In: *Advances in Neural Information Processing Systems*. Vol. 34. Curran Associates, Inc., 2021. URL: https://proceedings.neurips.cc/paper/2021/file/210f760a89db30aa72ca258a3483cc7f-Paper.pdf.

[10] Ian Goodfellow et al. “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems*. Vol. 27. Curran Associates, Inc., 2014. URL: https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

[11] Firas Al-Hafez et al. “LS-IQ: Implicit Reward Regularization for Inverse Reinforcement Learning”. In: *International Conference of Learning Representations (ICLR)*. Vol. 11. 2023. URL: https://openreview.net/pdf?id=o3Q4m8jg4BR.

[12] Nicolas Heess et al. *Emergence of Locomotion Behaviours in Rich Environments*. 2017. URL: https://arxiv.org/abs/1707.02286.

[13] Jonathan Ho and Stefano Ermon. “Generative Adversarial Imitation Learning”. In: *Advances in Neural Information Processing Systems*. Vol. 29. Curran Associates, Inc., 2016. URL: https://proceedings.neurips.cc/paper/2016/file/cc7e2b878868cbae992d1fb743995d8f-Paper.pdf.

[14] Max Jaderberg et al. *Reinforcement Learning with Unsupervised Auxiliary Tasks*. 2016. URL: https://arxiv.org/abs/1611.05397.

[15] “Jumping over obstacles with MIT Cheetah 2”. In: *Robotics and Autonomous Systems* 136 (2021). URL: https://www.sciencedirect.com/science/article/pii/S0921889020305431.

[16] Ilya Kostrikov, Ofir Nachum, and Jonathan Tompson. *Imitation Learning via Off-Policy Distribution Matching*. 2019. URL: https://arxiv.org/abs/1912.05032.

[17] Ilya Kostrikov et al. *Discriminator-Actor-Critic: Addressing Sample Inefficiency and Reward Bias in Adversarial Imitation Learning*. 2018. URL: https://arxiv.org/abs/1809.02925.

[18] Volodymyr Mnih et al. “Asynchronous Methods for Deep Reinforcement Learning”. In: *Proceedings of The 33rd International Conference on Machine Learning*. Vol. 48. PMLR, 2016. URL: https://proceedings.mlr.press/v48/mniha16.html.

[19] Michael Neunert et al. “Whole-Body Nonlinear Model Predictive Control Through Contacts for Quadrupeds”. In: *IEEE Robotics and Automation Letters* 3 (2018). URL: https://ieeexplore.ieee.org/abstract/document/8276298.

[20] Xue Bin Peng et al. “Amp: Adversarial motion priors for stylized physics-based character control”. In: *ACM Transactions on Graphics (TOG)* 40 (2021). URL: https://dl.acm.org/doi/abs/10.1145/3450626.3459670.

[21] Xue Bin Peng et al. “Learning agile robotic locomotion skills by imitating animals”. In: *arXiv preprint arXiv:2004.00784* (2020). URL: https://arxiv.org/pdf/2004.00784.pdf.

[22] Xue Bin Peng et al. “Variational discriminator bottleneck: Improving imitation learning, inverse rl, and gans by constraining information flow”. In: *arXiv preprint arXiv:1810.00821* (2018). URL: https://arxiv.org/abs/1810.00821.

[23] Siddharth Reddy, Anca D Dragan, and Sergey Levine. “Sqil: Imitation learning via reinforcement learning with sparse rewards”. In: *arXiv preprint arXiv:1905.11108* (2019). URL: https://arxiv.org/abs/1905.11108.

[24] Unitree Robotics. *Unitree Robotics, A1*. 2021. URL: https://www.unitree.com/products/a1/.

[25] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. MIT press, 2018. URL: https://ebookcentral.proquest.com/lib/ulbdarmstadt/detail.action?docID=6260249.

[26] Emanuel Todorov, Tom Erez, and Yuval Tassa. “MuJoCo: A physics engine for model-based control”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012. URL: https://ieeexplore.ieee.org/document/6386109.

[27] Faraz Torabi, Garrett Warnell, and Peter Stone. *Generative Adversarial Imitation from Observation*. 2018. URL: https://arxiv.org/abs/1807.06158.

[28] Yuxiang Yang. *Python Environments for Unitree A1 Robot*. 2021. URL: https://github.com/yxyang/locomotion_simulation.

[29] Yuxiang Yang et al. “Fast and Efficient Locomotion via Learned Gait Transitions”. In: *Proceedings of the 5th Conference on Robot Learning*. Vol. 164. PMLR, 2022. URL: https://proceedings.mlr.press/v164/yang22d.html.

[30] Brian D Ziebart et al. “Maximum entropy inverse reinforcement learning.” In: *Association for the Advancement of Artificial Intelligence (Aaai)*. Vol. 8. 2008. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=3789e642100687714c0196771710ee35094209e2.

# A. Graphs of the Mean Length of the Experiments

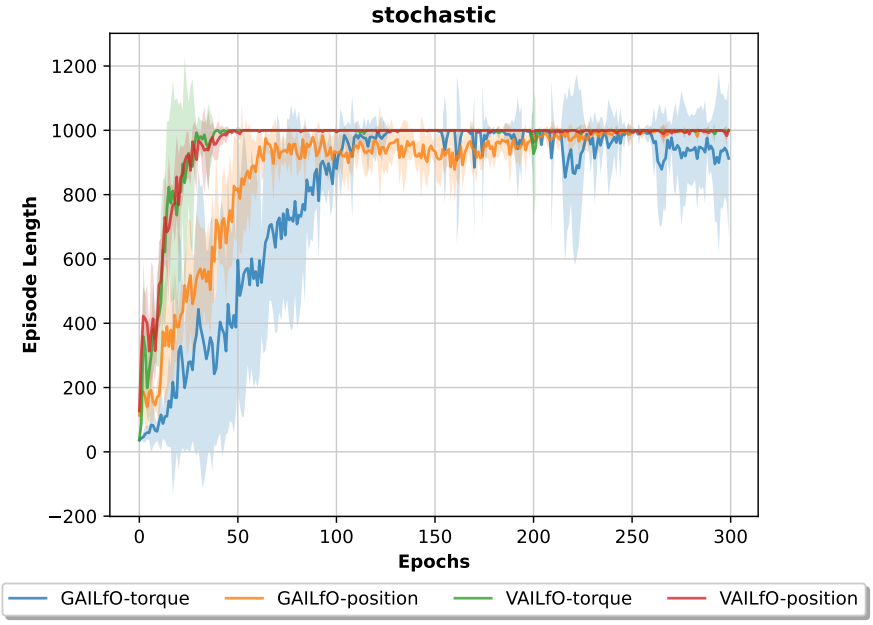



Figure A.1.: The mean length of trajectories from the agents. The agents are trained with GAILfO and VAILfO with torque and position control over 300 epochs.

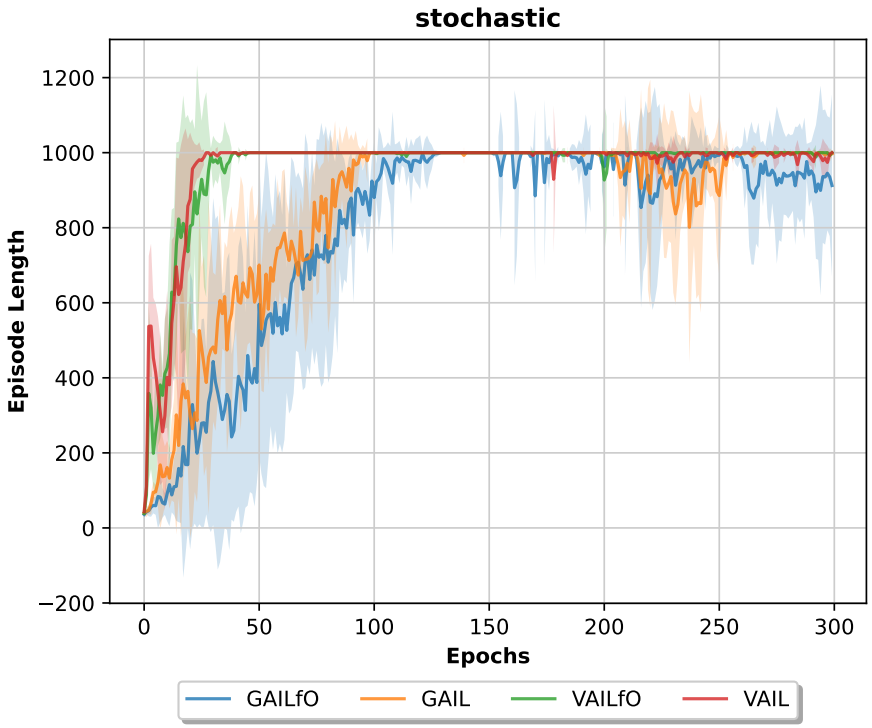



Figure A.2.: The mean length of trajectories from agents from GAILfO, VAILfO, GAIL, and VAIL over 300 epochs.

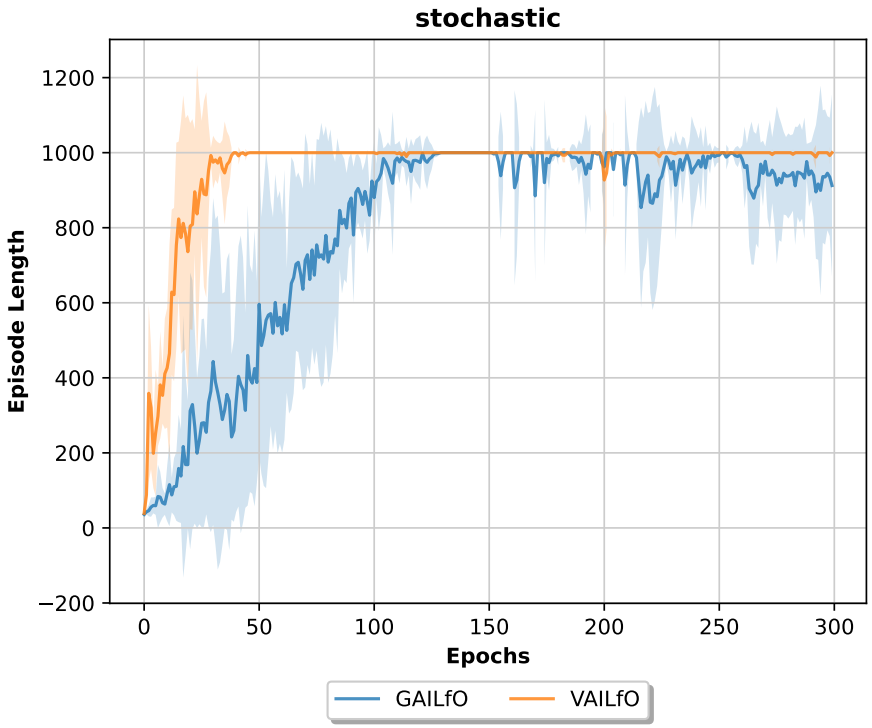



Figure A.3.: The mean length of trajectories of agents learned with GAILfO and VAILfO. The goal is to walk in a straight line.

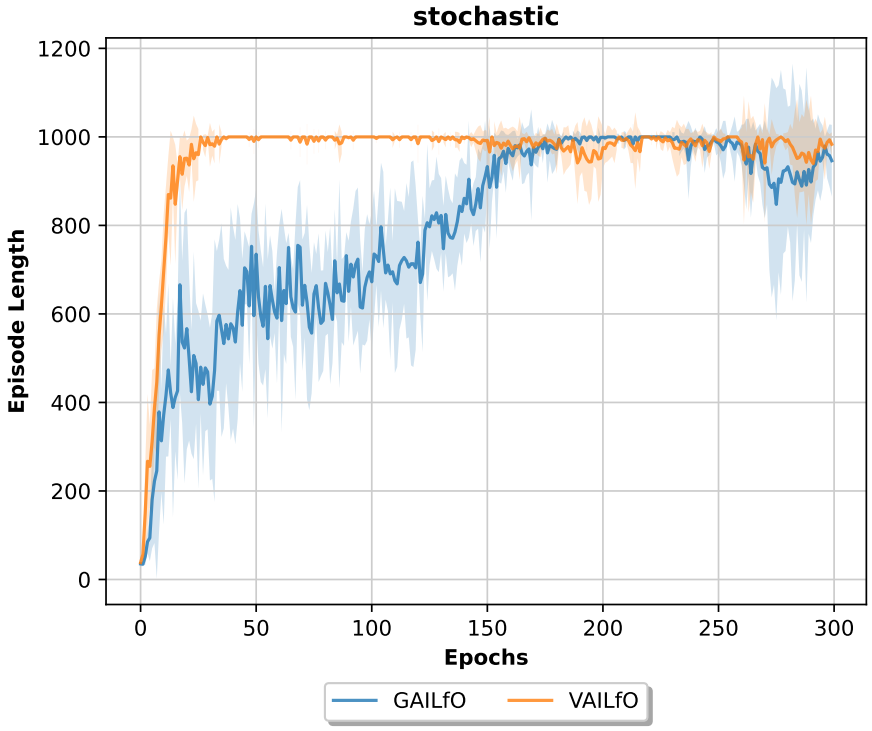



Figure A.4.: The mean length of trajectories of agents learned with GAILfO and VAILfO. The goal is to walk in 8 directions.

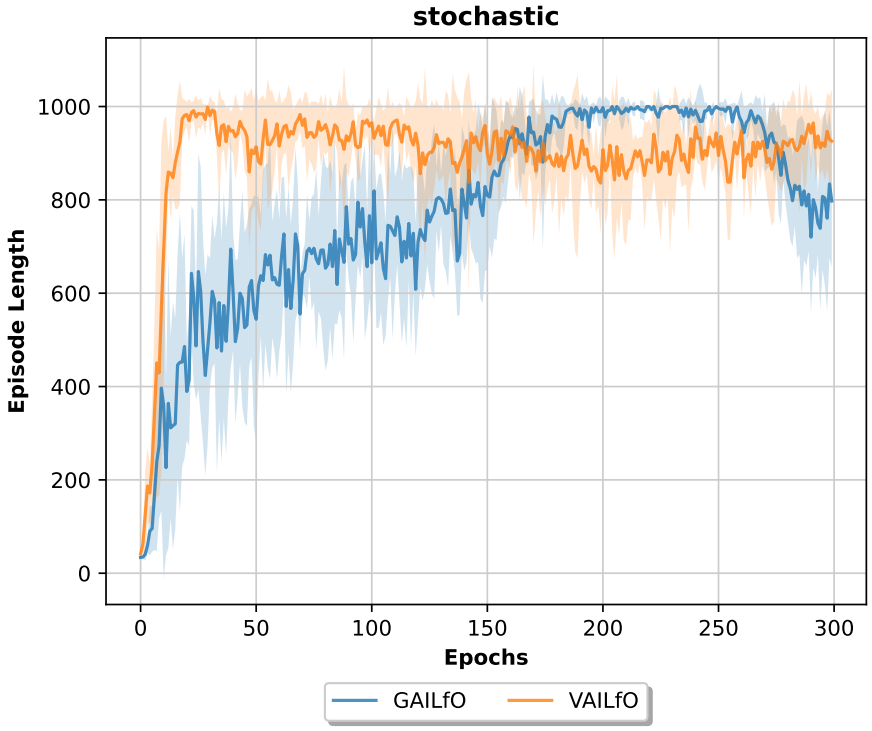



Figure A.5.: The mean length of trajectories of agents learned with GAILfO and VAILfO. The goal is to walk in 8 directions, independent of the yaw.