Sample-Efficient Learning-Based Controller For Bipedal Walking In Robotic Systems

Daten-effizienter lernbasierter Regler
für zweibeiniges Laufen in robotischen Systemen

Master thesis by Rustam Galljamov
Date of submission: November 11th, 2020

1. Review: Prof. Jan Peters Ph.D.
2. Review: Prof. Dr. André Seyfarth
3. Review: M. Sc. Boris Belousov
4. Review: Dr. Guoping Zhao

Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

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

Hiermit versichere ich, Rustam Galljamov, die vorliegende Masterarbeit 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, November 11th, 2020

R. Galljamov

Abstract

Bipedal locomotion in robotic systems remains a generally unsolved challenge. With only a few exceptions like the Boston Dynamics' robot Atlas or Digit from Agility Robotics, no humanoid robot is able to dynamically traverse its environment on two legs guaranteeing to keep balance on rough terrain and recover from perturbations. The reason is likely to be the complexity of the walking motion requiring processing high dimensional sensory input and producing synchronized motor commands for multiple joints at high control rates.

Deep Reinforcement Learning (deep RL) was successfully applied to replicate dynamic bipedal locomotion in physics-based simulations on different level of complexities, starting from outputting target angles for PD position controllers in each joint (Peng et al., 2018a) to controlling 284 muscle activations of a full-body musculoskeletal model (Lee et al., 2019). Despite these promising results, deep RL is only slowly finding its way into the robotics community. As one of the main reasons for that, we see the low sample efficiency of this algorithmic approach.

The aim of this work is therefore to improve the sample efficiency of deep RL in the specific case of training bipedal walking in simulation. We follow the imitation learning approach of DeepMimic (Peng et al., 2018a) and use the Proximal Policy Optimization algorithm (Schulman et al., 2017) to achieve stable and visually human-like forward walking in 3D. We further develop a new metric for measuring the sample efficiency of an algorithm in the considered context and show that changing the action space and incorporating the knowledge about the symmetry of the walking gait increase the sample efficiency by up to 53%. The combination of our best approaches reduced the required number of samples to achieve stable walking by 78% corresponding to a resulting wall-clock time of approximately two hours.

Zusammenfassung

Die bipedale Fortbewegung in Robotersystemen bleibt nach wie vor eine allgemein ungelöste Herausforderung. Mit nur wenigen Ausnahmen, wie dem Roboter Atlas von Boston Dynamics oder Digit von Agility Robotics, ist kein humanoider Roboter in der Lage, seine Umgebung dynamisch auf zwei Beinen zu durchqueren, das Gleichgewicht auch auf unebenem Gelände zu halten und sich von äußeren Störungen zu erholen. Der Grund hierfür liegt wahrscheinlich in der hohen Komplexität der Laufbewegung, die die Verarbeitung hochdimensionaler sensorischer Eingaben erfordert sowie die Vorhersage synchronisierter Motorbefehle für mehrere Gelenke.

Deep Reinforcement Learning (Deep RL) wurde erfolgreich angewandt, um die dynamische bipedale Fortbewegung in physikbasierten Simulationen auf verschiedenen Komplexitätsebenen zu replizieren, angefangen von der Ausgabe von Soll-Winkeln für PD-Positionsregler in jedem Gelenk (Peng et al., 2018a) bis hin zur gleichzeitigen Steuerung von 284 Muskelaktivierungen eines detaillierten Menschen-Models (Lee et al., 2019). Trotz dieser vielversprechenden Ergebnisse findet Deep RL nur langsam den Weg in das Feld der Robotik. Als einen der Hauptgründe dafür sehen wir die geringe Daten-Effizienz dieser algorithmischen Ansätze.

Das Ziel dieser Arbeit besteht daher darin, die Daten-Effizienz von Deep RL Algorithmen im speziellen Fall des Trainings vom Laufen auf zwei Beinen in der Simulation zu verbessern. Wir folgen der Methode von DeepMimic (Peng et al., 2018a) und verwenden den Proximal Policy Optimization (PPO) Algorithmus (Schulman et al., 2017), um ein stabiles und visuell menschenähnliches Laufen in 3D zu erreichen. Zudem, entwickeln wir eine neue Metrik zur Messung der Daten-Effizienz von Algorithmen im betrachteten Kontext und zeigen, dass die Vorhersage von Gelenk-Drehmomenten und die Einbeziehung des Wissens über die Symmetrie des Gehens die Daten-Effizienz um bis zu 53% erhöhen. Die Kombination unserer besten Ansätze reduziert die für ein stabiles Gehen erforderliche Anzahl von Proben um 78%, was einer resultierenden Trainings-Zeit von etwa zwei Stunden entspricht.

Contents

1. Motivation

We want robots to do tedious, dirty and dangerous work for us. How great would it be to have robots tidying up our rooms, taking care of garbage collection or entering highly radioactive nuclear reactors? Our world however is shaped for creatures walking on two legs and manipulating the environment using two hands. To fulfill this longstanding dream, we need to build humanoid robots and develop algorithms controlling motors to replicate bipedal locomotion.

Deep Reinforcement Learning (deep RL) has solved bipedal walking in simulated humanoids using motors and muscles (Peng et al., 2018a; Anand et al., 2019; Lee et al., 2019; Yang et al., 2020). Using domain adaptation techniques, policies trained in simulation have been successfully transferred to real robots (Akkaya et al., 2019; Lee et al., 2020; Siekmann et al., 2020). To develop a deep reinforcement learning based controller for bipedal locomotion on a real robots we thus have to follow 3 general steps (Peng et al., 2020):

1. Build a simulation model of your robot.
2. Train your robot to achieve stable robust walking in simulation.
3. Transfer the policy learned in simulation to the real robot (domain adaptation).

These steps might seem easier as they really are but are though a valid recipe to train a robot to walk on two legs. Every sophisticated robot having the physical capabilities to walk on two legs is very likely to have a precise simulation. Domain adaptation techniques make the sim-to-real transfer possible and are already efficient. Peng et al. (2020) adapted their policy from simulation to the real quadrupedal robot using less then 10 minutes of real-world data.

Given these results, why aren't more roboticists working on humanoids using deep RL to train their robots?

We believe, the answer is likely to be the sample inefficiency of deep RL algorithms, making the training in simulation require multiple days to solve a given task. And this only after a set of optimized hyperparameters is provided which in turn requires repeating the training procedure multiple to hundreds of times. By making learning in simulation faster, we hope to lower the entry barrier for roboticists to use deep RL, thus extend their toolbox by another promising tool and speed up the progress in humanoid robots.

2. Introduction

For many decades, science fiction feeds our fantasies of a future with human-like robots being all around us. They are strong, agile and besides taking care of tedious or dangerous tasks can also perform sophisticated parcours and martial arts. In real life, an average humanoid robot on the other side is far away from these scenarios. Most humanoids are struggling at keeping balance on their legs, walk slowly, unnaturally and often fall down. Why is there such a high difference between science fiction and science fact?

The answer to this question lies in the very intelligent design of the human body and muscles and as we believe especially in the complexity of our high level controller: the central nervous system (CNS) consisting of the brain and the spinal cord. Our internal sensors produce high amounts of raw sensory data every few milliseconds. Our CNS processes these in order to produce control signals for up to 300 skeletal muscles (Lee et al., 2019) that move the segments of our body. Even though the robots are inferior to us humans in terms of hardware design, we believe the control plays a crucial role in explaining the difference in walking agility and stability.

Artificial neural networks have proven their ability to automatically extract useful features from high dimensional raw input data (Schmidhuber, 2015; LeCun et al., 2015). Deep Reinforcement Learning (deep RL) successfully used these capabilities for sequential decision making and achieved superhuman-level performance in board and video games (Mnih et al., 2013; Silver et al., 2017). Finally, deep RL found its way in the domain of continuous control (Lillicrap et al., 2015; Duan et al., 2016) and was successfully applied to solve bipedal walking in simulation.

While first attempts resulted in partially idiosyncratic gaits (Lillicrap et al., 2015), the introduction of motion capturing data to the learning process made the motions smooth and human-like (Peng et al., 2018a). Lastly, deep RL has been able to replicate human-like muscle activations in complex musculoskeletal models including up to 300 muscles (Anand et al., 2019; Lee et al., 2019) and definitely confirmed its potential to compete with the human central nervous system.

Deep RL however has a drawback. It requires tens to hundreds of millions data samples to learn walking (Peng et al., 2018a; Lee et al., 2019; Peng et al., 2020). We believe this to be a major entry barrier for roboticists to apply deep RL in their research and therefore want to improve the sample efficiency of the specific case of learning bipedal walking in simulation.

We follow the imitation learning approach called DeepMimic (Peng et al., 2018a), adjust it to the specifics of stable walking on two legs and use it as the baseline for our investigations on sample efficiency improvements. The use of motion capturing data helps achieving a visually human-like appearance of the learned walking gait. By shaping the reward function and stopping the episodes based on early falling detection, we force the agent to focus on keeping balance resulting in faster achievement of stable walking.

This work further evaluates multiple metrics to measure the sample efficiency of an algorithm while considering the stability and human-likeness of the learned walking gait, resulting in a new metric called the summary score. Based on these metrics, we compare multiple action spaces and show that torque control results in

twice as sample-efficient learning compared to outputting target angles for PD position controllers. In case position control is the only option for a robot, we propose an alternative action space definition improving the data efficiency by 41% compared to the baseline. We furthermore show that incorporating prior knowledge about the symmetry of the walking gait in the training process can double the learning efficiency. The work is closed by summarizing the results into practical advice for roboticists interested in applying deep RL.

3. Foundations

In this chapter we present the required preliminaries to understand the work at hand. We start by explaining terms from the field of biomechanics used to describe bipedal walking. Thereafter, an overview of the relevant areas in the field of reinforcement learning is given.

3.1. Biomechanics

Biomechanics study the movement of living beings by utilizing concepts and methods from the science of mechanics (Hatze, 1974). It therefore provides the necessary vocabulary to describe the motion of interest in our work: bipedal walking. Despite focusing on the movement of living beings, biomechanics concepts equally apply to motion in robots and have been often the inspiration for better control algorithms (Popović, 2013; Maldonado et al., 2019; Oehlke et al., 2019). Throughout this work, we refer to multiple biomechanical terms and shortly explain them in this section.

3.1.1. Anatomical Planes

Each movement can be decomposed into displacements in three anatomical planes. The intersection of these planes forms the longitudinal axis that vertically traverses the body from the feet to the head. Joint rotations in each plane are performed around the corresponding normal axes. The following listing describes the anatomical planes (Likens and Stergiou, 2020):

- The **sagittal plane** divides the body vertically into left and right. Forward and backward motions like walking and running are almost fully captured by this plane. Therefore, it is a common simplification to reduce the walking to a two-dimensional movement in the sagittal plane.
- The **frontal plane** intersects the body forming a front and a back part. It is the second most involved plane during walking and captures deviation of the COM in the left and right direction away from the straight line.
- The **transverse plane** separates the human horizontally into a lower and upper body. During walking, there is only a little movement in this plane. However, it plays an important role in detecting falling.

3.1.2. Center of Mass (COM)

Humans are multi-body systems consisting of multiple segments connected by joints. To describe a movement of such a complex system in a 3D space, we need to report the trajectories of spatial positions and orientations of individual segments or the angle and velocity trajectories of each joint. Still, these information only provide us with the movement of the individual parts and not the whole system. To understand the motion of the overall system, we can sum up the positions of individual joints into a single point and track its kinematics. When this point is determined as a weighted sum of segment positions p_i with their corresponding masses m_i as weights, the resulting point is called the *center of mass* or *COM* for short. The following formula illustrates the calculation of the COM position vector (Beatty, 2005):

$$p_{com} = \frac{1}{M} \sum_{i=1}^{n} m_i p_i \ \text{ with } \ M = \sum_{i=1}^{n} m_i$$

Interestingly, any forces acting on the system can be reduced to forces acting on this single point. Therefore, the COM kinematics and kinetics can be used to fully characterize a motion of arbitrarily complex objects and systems (Beatty, 2005). In the specific case of a bipedal walking motion, the COM position and velocity play a crucial role and are parts of most models describing the motion (Kuo, 2007; Lee and Farley, 1998).

3.1.3. Bipedal Walking Gait Cycle

The bipedal walking is a periodic motion consisting of a repetition of gait cycles. The gait cycle describes the time duration between the reoccurrence of the same point in the walking movement (Alamdari and Krovi, 2017). Within the context of this work, we define the gait cycle as the interval between two consecutive touchdowns of the same foot. The touchdown is the moment a foot touches the ground after being in the air. In addition, we refer to a *step cycle* describing a single step or concretely the time between a touchdown with one foot to the touchdown of the other.

The human gait cycle can be subdivided into multiple phases, starting with two to a detailed distinction of 8 phases (Richie Jr, 2020). The simplest definition distinguish between a stance and a swing phase. The stance phase describes the 60% of the gait cycle where the foot is in contact with the ground. The swing phase describes the remaining 40%. The next level of detail can be added by introducing a double stance phase describing the moment where both feet touch the ground (Li and Dai, 2006).

The two main phases can be further subdivided into multiple intervals. The stance phase for example distinguishes between initial contact, loading response, mid-stance, terminal stance and pre-swing (Richie Jr, 2020). For the context of this work it is important to notice the dependency of the gait phases on ground contact information of individual feet as well as the duration of the ground contact illustrated by terms like initial, mid and terminal.

3.2. Deep Reinforcement Learning

This section provides a brief overview over reinforcement learning topics relevant to the work at hand. For details, please refer to the review articles (Kober et al., 2013; Arulkumaran et al., 2017; Li, 2018), a tutorial from OpenAI (Achiam, 2018) as well as the popular book of Sutton and Barto (2018).

Reinforcement learning (RL) aims at solving sequential decision tasks (Sutton and Barto, 2018). In a standard RL setting, an agent interacts with an environment in discrete timesteps (Lillicrap et al., 2019). At each timestep t the agent receives a state $s_t \in \mathcal{S}$ and chooses an action $a_t \in \mathcal{A}$ sampled from its policy $\pi(a_t|s_t)$. The environment then transitions in the next state s_{t+1} according to its transition probability $\mathcal{P}(s_{t+1}|s_t, a_t)$ and outputs a scalar reward signal $r_t = \mathcal{R}(s_t, a_t, s_{t+1})$ (Li, 2018). When the environment satisfies the Markov Property, hence the current state action pair (s_t, a_t) contain all required information to determine the next state s_{t+1}, the RL setting is modeled as a Markov Decision Process or MDP for short (Li, 2018).

A sequence of states and actions generated by following the policy for multiple timesteps is called a trajectory $\tau = (s_0, a_0, s_1, a_1, \ldots)$ with the first state being sampled from an initial probability distribution $s_0 \sim p(s_0)$ (Achiam, 2018). The cumulative sum of rewards collected on the trajectory is called the return $R(\tau)$ and is calculated as follows:

$$R(\tau) = \sum_{t=0}^{T} \gamma^t r_t$$

Here, T denotes the length of an episode in episodic tasks in which case the discounting factor γ is often set to 1. In case of an infinite-horizon task $T = \infty$ and $\gamma \in [0, 1)$. The goal of the agent is to maximize the expected cumulative return by finding the optimal policy π^* (Achiam, 2018):

$$\pi^* = \arg\max_{\pi} \mathop{\mathrm{E}}_{\tau\sim\pi}[R(\tau)]$$

Other important concepts in RL are the value functions. The state value function $V^\pi(s)$ denotes the expected cumulative reward by starting from a state s and following the current policy π. The action value function $Q^\pi(a, s)$ analogously provides the expected reward by taking the specific action a in state s. The values are calculated as following (Achiam, 2018):

$$\begin{aligned} V^\pi(s) &= \mathop{\mathrm{E}}_{\tau\sim\pi}[R(\tau) \mid s\,] \\ Q^\pi(s, a) &= \mathop{\mathrm{E}}_{\tau\sim\pi}[R(\tau) \mid s, a\,] \end{aligned}$$

By preceding the expectations with a max operator over the actions $\max_\pi$ assuming to take the best possible actions in each state instead of sampling actions from the current policy π, we get the optimal state and action value functions $V^*(s)$ and $Q^*(s, a)$ showing the maximum achievable values. By taking the difference between both function, we get the advantage function $A^\pi(a_t|s_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$ which the relative value of action a_t with respect to the average value of all possible actions in the state s_t.

Finally, we arrive at the explanation of the phrase "deep" in deep reinforcement learning (deep RL). Where in small dimensional discrete state spaces the values of states and actions as well as the policy can be represented in tabular forms, targeting at continuous state and action spaces requires a parametrized representation. When choosing deep neural networks to represent the value functions or the policy, we speak of deep RL.

3.2.1. Model-Free vs. Model-Based RL

Targeting at the very broad and general task of optimizing an expected cumulative reward signal, multiple different solution approaches have been proposed by the research community.

An important split in the family of reinforcement learning algorithms is the fact if the agent is using a model of the environment transitions (model-based) or not (model-free). A transition model $p[s_{t+1}, r_t | s_t, a_t]$ predicts the distribution over possible next states and rewards given the current state and action. Some authors treat the reward generating part of the model separately. Having a model of the environment allows to plan the outcome of multiple consecutive actions without the need to perform them in the real environment. The outcomes of the planning can then be incorporated into policy learning (Achiam, 2018). This way, model-based algorithms can significantly reduce the amount of required environment interactions and thus improve the sample efficiency (Kober et al., 2013; Arulkumaran et al., 2017; Li, 2018).

If the model is not provided, the agent has to learn the model purely from interactions with the environment. Even it is possible to learn a transition model from interactions (Åström and Wittenmark, 2013), it is not possible to do it without errors for complex environments like a robot in the real-world (Kober et al., 2013). Inaccurate transition models lead to prediction errors when used for planning. The errors can significantly increase when using the model to predict multiple steps into the future (Asadi et al., 2019). RL agents have been observed to exploit these errors to maximize the return which can also be explained as overfitting to the inaccurate learned model (Kober and Peters, 2010). The performance in the real environment in these cases is poor.

Model-free methods treat the environment as a black-box generating rewards and the next state given state-action pairs. As no planning is involved, the policy is fully optimized based on collected experiences. This fact makes model-free algorithms significantly less sample-efficient compared to model-based approaches. On the other hand, model-free approaches avoid the pitfalls of an inaccurate model, are easier to tune and show a more stable convergence behavior (Achiam, 2018).

3.2.2. Value-Based Methods

Model-free methods can be further distinguished in value-based and policy optimization methods. The final goal is always a policy, that maps the states to the optimal actions. However, there are multiple ways to achieve this behavior. In this section we explain the value-based methods and present their counterpart in the next section.

The first deep RL agent was the Deep Q-Network (Mnih et al., 2013), a value-based algorithm. These methods focus on approximating the optimal action value function with a deep neural network $Q^*_\theta(s, a)$ with θ denoting the network parameters. The action of the agent is then chosen by

$$a_t = a(s_t) = \arg\max_{a_t} Q_\theta(s_t, a_t)$$

An important role in these methods is played by the Bellman Equation, describing the relationship between the optimal Q-values of consecutive state-action pairs:

$$Q^*(s_t, a_t) = \mathop{\mathrm{E}}_{s_{t+1} \sim \mathcal{P}} \left[r(s_t, a_t) + \gamma \max_{a_{t+1}} \left[Q^* \left(s_{t+1}, a_{t+1} \right) \right] \right]$$

Value-based methods use this relationship and minimize the Bellman Error Loss to optimize the parameters of the Q-network (Mnih et al., 2013):

$$L(\theta_t) = \underset{s_t \sim \mathcal{P},\, a_t \sim \pi}{\mathrm{E}} \left[\left(r(s_t, a_t) + \gamma \max_{a_{t+1}} \left[Q_{\theta_{t-1}}(s_{t+1}, a_{t+1}) \right] - Q_{\theta_t}(s_t, a_t) \right)^2 \right]$$

As the Bellman Equation has to hold for every state-action pair sampled from the same environment, value-based methods can use experiences collected by other than the current policy. This scenario is referred to as being off-policy and allows next to using experiences from previous versions of the policy, use experiences from any other policy or expert as long as they were collected in the same environment (Nachum et al., 2017). This makes value-based methods very sample-efficient.

However, these methods have been reported to be much harder to train coming with many scenarios what can go wrong (Tsitsiklis and Van Roy, 1997; Szepesvári, 2009; Achiam, 2018). Multiple improvements to counteract the drawbacks were proposed (Hasselt, 2010; Hessel et al., 2018). The most prominent deep RL algorithms today however either follow the policy gradient approach or use a combination of both (Lillicrap et al., 2015; Fujimoto et al., 2018).

3.2.3. Policy Gradient Methods

Policy gradient methods are motivated by the idea that it might be easier to directly learn a policy instead of learning the values of individual states and actions to make decisions (Simsek et al., 2016). In contrast to value-based approaches these methods explicitly use a parametrized policy and do not rely on the Q-function to select an action. The only requirement for the parametrization of the policy is to be differentiable with respect to its parameters θ for all states and actions. This way, policy-based methods can directly optimize the policy by using the gradient of a performance metric with respect to the policy parameters (Sutton et al., 2000).

Conventionally, performance of a deep RL agent is measured as the expected return over trajectories τ sampled by following the policy, described by the following objective function:

$$J(\pi_\theta) = \underset{\tau \sim \pi_\theta}{\mathrm{E}} [R(\tau)]$$

The policy gradient is derived in (Sutton et al., 2000) and results in the following formula:

$$\nabla_\theta J(\pi_\theta) = \underset{\tau \sim \pi_\theta}{\mathrm{E}} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta (a_t \mid s_t) R(\tau) \right]$$

Having an expectation of the policy gradient, we can estimate it by sampling $|\mathcal{D}|$ trajectories $\{\tau_i\}_{i=1,\dots,N}$ from the environment and calculate the sample mean $\hat{g}$:

$$\hat{g} = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta (a_t \mid s_t) R(\tau)$$

To obtain an unbiased estimate of the policy gradient, it requires on-policy data. This means all trajectories are required to be sampled by following the current policy π_θ. This restriction reduces the sample-efficiency but leads to a significantly better convergence behavior compared to off-policy methods (Nachum et al., 2017; Mousavi et al., 2017).

Policy gradient estimates using the return have a low bias but high variance. To reduce the variance, it is common to replace the return $R(\tau)$ by the advantage function $A^{\pi}(a_t|s_t)$ resulting in the following policy gradient:

$$\nabla_\theta J(\pi_\theta) = \underset{\tau\sim\pi_\theta}{\mathrm{E}}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta (a_t \mid s_t)\, A^{\pi}(a_t|s_t)\right]$$

The free choice of the policy parametrization in policy gradient methods allows to induce prior knowledge into the learning process (Sutton and Barto, 2018). Furthermore, it makes it easier to explore the state and action spaces by utilizing a stochastic policy, most commonly outputting diagonal Gaussian distributions in the continuous case and or softmax distribution when discrete actions are required (Nachum et al., 2017).

As policy gradient methods optimize the actual objective they are guaranteed to converge to - most often a local - optimum given a small enough learning rate (Mousavi et al., 2017) and are therefore a popular choice to solve RL problems.

3.2.4. Trust Region Policy Optimization

In policy gradient methods the policy is updated by taking small steps in the parameter space facing in the direction of rising performance. The new policy after the update is thus very close to the old one in parameter space. Small changes in parameter space can however significantly change the resulting distribution and a big change in the distribution might significantly worsen the behavior of the policy (Schulman et al., 2015).

To overcome this issue and guarantee convergence, the changes in parameter space might further be reduced. That however leads to slower learning and increases the sample complexity. Trust region methods tackle this circumstances by limiting the change between the distributions before and after the gradient update (Schulman et al., 2015).

A common way used in the TRPO algorithm (Schulman et al., 2015) is to constraint the KL divergence between the two policies. Other algorithms clip the objective function to reduce the chance of a big change in the distributions after the update (Schulman et al., 2017).

4. Related Work

4.1. Deep Reinforcement Learning for Bipedal Locomotion

Mnih et al. (2015) were the first to use the ability of deep neural networks to automatically extract useful features from high-dimensional state observations within a reinforcement learning scenario. Their Deep Q-Network (DQN) was the first deep reinforcement learning (deep RL) agent and achieved a human-like or better performance on a set of 49 Atari video games. Hereby, raw game pixels were the agent's input resulting in a state space of multiple thousand dimensions.

While the state space dimensionality was very high, the action space contained only a few discrete choices. To solve tasks with continuous and high-dimensional action spaces, (Lillicrap et al., 2015) extended DQN to an off-policy actor-critic algorithm, called the Deep Deterministic Policy Gradient (DDPG) algorithm. To the best of our knowledge, these authors were the first to apply a deep RL agent to solve bipedal locomotion by solving the *walker2d* environment, one of RL benchmark problems within the OpenAI Gym (Brockman et al., 2016).

The first deep RL agent able to generate bipedal locomotion in the *walker3d* environment consisting of a full-body humanoid able to move in all three directions was presented by Schulman et al. (2018). Meanwhile, Peng and van de Panne (2017) compared different action spaces in the context of learning locomotion in physics-based character animations. The authors recommend using PD position controllers in each joint and train the policy to output joint target angles. Outputting joint torques led to the weakest final performance.

Heess et al. (2017) stressed the necessity of a high quality reward signal in the context of learning locomotion. Alternatively, they reported stable behavior to emerge also from simple reward formulations when shaping environments appropriately. Peng et al. (2018a) utilized motion capturing data to get a high-quality reward signal and proposed DeepMimic, a framework for learning locomotion for simulated characters able to learn walking, running, gymnastics, and martial arts. The high generality of this approach reported by the authors has also been proven in other works.

Anand et al. (2019) successfully applied DeepMimic to a lower body musculoskeletal model to closely imitate the walking behavior of a human up to the activation signals in individual muscles. Lee et al. (2019) too underlined the promise of this approach by using DeepMimic to train a full-body musculoskeletal model reproducing ground reaction force patterns and muscle activation signals. Finally DeepMimic has been shown to with additional effort transfer the policies learned in simulation to real robots (Xie et al., 2018; Peng et al., 2020).

4.2. Sample Efficient Learning of Bipedal Walking

Aiming for quicker learning in deep RL, parallelization is a general approach to follow given the required computational power is provided (Nair et al., 2015; Clemente et al., 2017). When expert demonstrations are available, behavior cloning could be used to pretrain the policy in a supervised learning fashion and provide a warm-start for the RL agent (Kober and Peters, 2010; Zhu et al., 2018). To cope with the known drawbacks of behavior cloning multiple techniques like dataset aggregation (Ross et al., 2011) and generative approaches (Ho and Ermon, 2016; Merel et al., 2017) were proposed. Furthermore, inverse reinforcement learning (Abbeel and Ng, 2004) could be used to derive a near-optimal reward function that has the potential to improve the learning speed by providing better guidance during training.

Off-policy methods have been shown to significantly reduce the required samples to convergence at the cost of longer wall-clock time (Lillicrap et al., 2015; Zheng et al., 2018). Also model-based methods are known for excellent sample efficiency when a precise model of the environment is given or can be easily learned (Polydoros and Nalpantidis, 2017; Kaiser et al., 2019).

Reda et al. (2020) investigate the influence of the environment design on learning locomotion and report a correctly specified control frequency to strongly improve the learning speed. Huang et al. (2017) and Metelli et al. (2020) explicitly mention a lower control frequency to increase the sample efficiency.

Peng and van de Panne (2017) report the choice of the action space having a high impact on the sample efficiency of learning bipedal locomotion in a 2D space. In their investigations, policies outputting target angles for PD position controllers

Abdolhosseini et al. (2019) incorporate the symmetry of locomotion into the training procedure. Following different approaches, they report more symmetric walking patterns but can only observe an insignificant improvement in the learning speed.

Finally, curriculum learning methods have been shown to speed up the learning process. Yu et al. (2018) provide assistive forces during training helping the character to move forwards and keep its balance. As the agent gets better with time, assistance is reduced until the character walks completely on its own. Peng et al. (2018a) also report to first train an agent on even ground before putting it in a rough terrain to reduce the number of samples until the more complex environment can be traversed.

4.3. The DeepMimic Approach in Detail

The DeepMimic approach (Peng et al., 2018a) is to our knowledge the most successful state of the art method to learning controllers for human-like locomotion in a physics-based simulation environment. Originally developed to control simulated characters for computer animations it has been successfully applied in other domains. (Lee et al., 2019) used the approach to train an agent to control over 200 muscles of a full-body musculoskeletal model to achieve walking, running and multiple sport exercises. (Peng et al., 2020) trained a four-legged robot to perform different motions in simulation and transferred the learned controllers to the real robot.

The main idea of DeepMimic is a combination of imitation and reinforcement learning. The agent is trained in a reinforcement learning setting to replicate the behavior of an expert. The expert behavior is provided in form of motion capturing data, *mocaps* for short. It contains either the COM position, orientation and velocity of individual limbs over time or the joint angle and angular velocity trajectories in combination with the body's COM kinematics.

The mocaps are used to shape the reward during training. This way a rich and dense learning signal is provided after each individual action taken in the environment. In addition, the learned motion is guaranteed to be similar to the reference data, e.g. human-like if the mocap data was collected from a human performing a task.

The approach has been shown to generalize well to different environments and tasks with very little to no tuning of the hyperparameters. The authors also implemented the possibility to specify additional goals while following the reference motion as good as possible. Examples are walking in different directions despite only having a straight walking reference or throwing a ball at targets different from those in the recorded data.

The authors use their own implementation of the Proximal Policy Optimization algorithm (Schulman et al., 2017) to train the policy. The policy is represented as a fully connected neural network with two hidden layers of size 1024 and 512 respectively. It maps the states to a diagonal Gaussian distribution over actions with a fixed covariance matrix. The network parameters are optimized utilizing Stochastic Gradient Descent with Momentum.

4.3.1. States and Actions

The state of the environment consists of the relative position and rotation of individual links of the character as well as their linear and angular velocities. The root of the coordinate frame is placed at the COM of the pelvis. The x-axis shows in the direction the pelvis is facing. In addition, a phase variable $\phi \in [0, 1]$ indicates the current timestep on the reference trajectories with $\phi = 0$ being the start and $\phi = 1$ the end of the motion.

In case of additional goals, goal-specific information is added to the state vector. The approach has also been shown successful to deal with locomotion over uneven terrain. In this scenario, a heightmap of the environment is reduced to a flat representation using convolutional layers and is added to the state vector.

The actions specify target angles for individual joints of the character. Proportional-derivative (PD) position controllers then generate joint torques to reach the desired angles. The policy thus operates as a high-level controller at 30Hz while the PD controller works at the speed of the simulation as a low-level controller at $1.2k$Hz.

4.3.2. Reward Function

The reward function is a weighted sum of multiple components that we explain after presenting the equation with the corresponding weights:

$$r = w_p r_p + w_v r_v + w_e r_e + w_c r_c$$
$$w_p = 0.65,\ w_v = 0.1,\ w_e = 0.15,\ w_c = 0.1$$

r_p is the reward for matching the joint positions of the reference motion at each simulation timestep. r_v encourages the agent to imitate the angular velocities of individual joints. r_e stands for the end-effector reward and is high when the character's hands and feet match the positions in the mocap data. Finally, r_c is calculated by comparing the body's COM position.

All four components have the same mathematical form and only differ in the choice of the scaling factor α_i:

$$r_i = \exp[-\alpha_i(\sum_{j||e} \|\hat{x} - x\|^2)]$$

$$\alpha_p = 2,\ \alpha_v = 0.1,\ \alpha_e = 40,\ \alpha_c = 10$$

x represents joint positions in r_p, joint angular velocities in r_v, describes the end-effector positions for r_e and the body's COM position vector when used to calculate r_c. $\hat{x}$ stands for the corresponding kinematics from the reference motion. The squared normed differences are either summed over the joints j or the end-effectors e.

4.3.3. Training Specifics

Next to the dense reward signals, the success of the DeepMimic approach is based on two important adjustments to the training procedure: *Early Termination* (ET) and *Reference State Initialization* (RSI).

ET is a well known idea in reinforcement learning where a training episode is terminated when the agent enters a state it cannot recover from. In the framework of imitation learning, Peng and his colleagues stop an episode when the animated character falls, detected by his head or torso having contact with the ground. ET limits the observation space to samples close to the distribution of the reference trajectories and avoids collecting samples from an area of the state space that is irrelevant for the task at hand.

Once an episode is terminated, the next one has to be initialized. While it is common in RL to have a single or few initial states, the authors propose to initialize each episode in a randomly selected point from the reference trajectories. RSI allows a better exploration of the desired state space by enabling the agent to collect experience from the whole state distribution from the beginning of the training. By putting the agent in a high-value state at the beginning of the episode, the value function is trained on states with widely distributed values instead of seeing undesired states most of the time. This encourages a quicker convergence.

4.4. Deep RL Algorithm: Proximal Policy Optimization (PPO)

PPO (Schulman et al., 2017) is a popular model-free on-policy gradient method which is simpler to implement compared to similar performing algorithms, still being sample-efficient, showing robust convergence, generalizing to different tasks and environments as well as being parallelizable and fast with regards to the wall-clock time. This deep RL algorithm has been often applied to solve legged locomotion in simulation and real robots (Heess et al., 2017; Anand et al., 2019; Haarnoja et al., 2019; Yang et al., 2020).

PPO approximates the policy gradient from sampled experiences and uses it to change the parameters of the policy network to increase the probability of actions leading to high returns and decrease the probability of ineffective actions. In order to quickly converge to a local optimum, it is necessary to take big or multiple smaller steps in the direction of this gradient. However the steps are taken in parameter space and even small changes in parameter space can cause huge changes in the resulting action distribution and badly influence the policy performance.

To avoid destructively large policy updates, PPO builds on the idea of Trust Region Policy Optimization (TRPO) and limits the maximum possible change in action distribution during the policy update. Trust region methods often limit the change in distribution by constraining the KL divergence between the current and the new policy. Like TRPO, PPO uses the probability ratio $r_t(\theta)$ calculated as the division of the next and current policies:

$$r_t(\theta) = \frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta_{\text{old}}}\left(a_t \mid s_t\right)}$$

$r_t(\theta)$ is bigger than 1 when the action is more likely under the new policy and smaller than 1 in the opposite case. In contrast to TRPO, PPO limits the maximum deviation between consecutive policies by allowing the probability ratio to only slightly deviate from 1. This is achieved by clipping the probability ratio and optimizing the following objective function, called the Clipped Surrogate Objective Function:

$$L(\theta) = \hat{\mathbb{E}}_t\left[\min\left(r_t(\theta)\hat{A}_t,\ \text{clip}\left[r_t(\theta), 1-\epsilon, 1+\epsilon\right]\hat{A}_t\right)\right] \tag{4.1}$$

The authors denote the maximum deviation with ϵ and recommend a default value of $\epsilon = 0.2$. By limiting the maximum possible change in action distribution during an update step, PPO allows using the same batch of experiences to perform multiple policy gradient steps increasing its sample efficiency.

Being an actor-critic method, a second neural network is maintained predicting the state and the action values to approximate the advantage $\hat{A}_t$ used in the objective function. The value network is trained by minimizing the squared loss between the predicted and the target state values, which are computed using TD(λ). Finally, the authors propose to add an additional entropy bonus encouraging exploration and preventing early convergence to sub-optimal deterministic policies (Mnih et al., 2016).

5. Methods

Our goal is to achieve sample-efficient learning of stable human-like bipedal walking using deep reinforcement learning. We begin by implementing a sample-efficient state of the art approach for learning locomotion named *DeepMimic* (Section 4.3). The training is guided by expert trajectories provided in the form of motion capturing data collected from a human performing the desired task (Section 5.1). The agent gets rewarded for actions replicating the reference motion and punished when trajectories in simulation deviate from the expert trajectories (Section 5.3.2).

Our investigations are conducted using a simulation model of a bipedal walker (Section 5.2). The policy is trained using the PPO algorithm (Section 4.4). After achieving stable walking following the DeepMimic approach, we implement multiple ideas on what can be changed in the algorithm or the environment to increase the learning speed (Section 5.5). To compare our approaches, we develop a new metric to measure the sample efficiency of an algorithm considering the quality of the learned controller (Section 5.4.2).

5.1. Motion Capturing Data for Imitation Learning

Following an imitation learning approach of learning human-like walking, expert demonstrations in form of joint trajectories are necessary. For our experiments, we use the processed motion capturing (mocap) data presented in (Anand et al., 2019). Here, 20 markers were attached to the lower body of a single male subject walking on a treadmill. The treadmill was operated at different constant speeds as well as followed a desired velocity profile accelerating from 0.7m/s to 1.2m/s and back to 0.7m/s. In the context of this work we use the phrases *motion capturing data*, *reference trajectories* and *reference motion* interchangeably.

The mocap dataset contains the trajectories of the knee, ankle and hip joint as well as COM and trunk kinematics in Cartesian coordinates. Besides, it also contains ground reaction forces, electrical muscle signals (EMG) and metabolic cost measurements. This additional information can be utilized to optimize the human-likeness of walking on different individual or combined levels (e.g. similar joint kinematics with similar energy consumption or muscle activation for musculoskeletal models).

The trajectories are split into individual steps. A step starts with the touchdown of a foot and ends with the touchdown of the opposite foot. For our experiments, we use 35 steps of constant speed walking at a constant speed of 1.5m/s and 250 steps recorded while the treadmill followed the described velocity profile.

The processed data is provided with a sample frequency of 400Hz. To obtain reference trajectories at a lower sample frequency, the data is down-sampled by skipping an appropriate whole number of steps. By skipping each second data point for example we obtain trajectories at 200Hz.

Due to a significant asymmetry in the recorded gait, we also create an artificial symmetric dataset. Therefore, the joint trajectories of both legs are swapped. The hip joint angles in the frontal plane are in addition negated.

The negation is also applied to the COM position in the frontal plane as well as for the trunk rotations around the x and z axes. Corresponding velocities are transformed analogically.

To assure the correct usage of the trajectories in our implementation, randomly selected steps from the dataset are played back in the simulator to make sure the expected walking motion is observed. The used motion capturing data is provided in the project code repository. Figure A.1 in the appendix illustrates the organization of the data. Here, we also present the distribution of the trajectories across a whole gait cycle (Figure A.2).

5.2. Bipedal Walker Simulation Model

Building on the DeepMimic approach which successfully generalizes to environments and tasks of different complexity, we decided to start our investigations with a simple 2D walker model and put more focus on sample efficiency improvement. We decided to use a popular benchmark environment and adapted it for use within the DeepMimic framework. Our experiments however revealed the need for a more complex environment to better reflect the effects of our approaches on sample efficiency. This Section introduces the original environment and describes its extensions to arrive at the final 3D walker model.

5.2.1. MuJoCo Physics Engine and OpenAI Gym

The walking environment used for our experiments is a combination of the well known *walker2D* MuJoCo environment from OpenAI Gym (Brockman et al., 2016) and the *human7segment* MuJoCo environment presented in (Anand et al., 2019).

OpenAI Gym is a benchmark suite of simulation environments for testing reinforcement learning algorithms. For environments with high-dimensional continuous state and action spaces like our chosen walker the authors use MuJoCo. MuJoCo, short for **Mu**ltiple **Jo**ints with **Co**ntact, is a state of the art physics engine (Todorov et al., 2012) especially suited for simulating robotic system (Erez et al., 2015). Being the fastest physics engine for robotics related simulations (Erez et al., 2015), MuJoCo is especially suitable to quickly generate a high amount of data required to train deep RL agents. A focus on precise contact modeling qualifies the physics engine for tasks like walking where contact plays a crucial role. An example of successfully using the MuJoCo engine to apply deep RL to a complex bipedal robot including sim-to-real transfer is presented in (Xie et al., 2018, 2020).

MuJoCo offers a big range of actuators from direct torque controlled motors to position and velocity controllers allowing to compare different action spaces. Beside these, it includes muscle models and therefore can be used for investigation of walking in musculoskeletal models in following works.

5.2.2. The 2D Walker Model

Our walking environment consists of the lower part of a simplified humanoid and a flat surface simulating the ground illustrated in Figure 5.1b. The three-segmented legs are attached to a trunk over frictionless hinge joints with soft range constraints. Virtual massless motors generate joint torques. Ground contact is modeled by a spring-damper system with tuneable parameters (Todorov et al., 2012).

Table 5.1.: **Controller Gains of the PD Position Servos.** The gains were tuned using the original 2D model we started our experiments with. When the model was extended to 3D, the additional joint (hip frontal) got the same PD gains assigned as its counterpart in the sagittal plane. The gains of individual joints are the same for both legs.

Joint	Hip Sagittal	Hip Frontal	Knee	Ankle
P gain (k_p)	3200	3200	1600	2800
D gain (k_d)	28	28	12	20

Following the DeepMimic approach, which has been shown to be environment agnostic (Peng et al., 2018a), we first decided to do our experiments with a 2D model with all motions of the walker being constrained to the sagittal plane. This decision allowed us to use the well known *walker2D* environment from OpenAI Gym displayed in Figure 5.1a.

In order to make sure the walker is physically able to replicate the joint reference trajectories from the expert, we adjusted the position, dimensions and inertial properties of all segments to match the once of the person the reference data was collected from. In addition, joint ranges were adapted according to (Anand et al., 2019). All other parameters of the environment remained unchanged. In case of using the mocap data to train bipedal walking controllers for a fixed robotic setup, the adjustment would need to happen on the site of the mocap trajectories as described in (Peng et al., 2020). In our case it was simpler to adapt the model properties than doing otherwise.

To consider torque limits present in real motors, the maximum motor torque in each joint is limited to 300Nm. This value is intentionally set above the required range to make sure the motors are strong enough to generate walking and we can focus on improving sample efficiency.

The simulation runs at 1kHz, uses radians to specify joint angles and the Runge-Kutta method of 4th order for numerical integration.

5.2.3. Extension of the Model with PD Position Servos

In order to train the agent to output the recommended target angles for proportional derivative (PD) position controllers (Peng and van de Panne, 2017; Peng et al., 2018a), we duplicated our walker model and replaced the motors with position servos. PD position controllers compare the current joint position q with a target angle q_{tar} and consider the current joint angular velocities $\dot{q}$ to calculate the joint torque in the following manner:

$$\tau = k_p(q_{tar} - q) - k_d\dot{q} \tag{5.1}$$

The PD gains k_p and k_d for individual joints were hand-tuned by fixing the trunk of the walker in the air and using the PD controllers to follow the reference trajectories of each individual joint. Tuning the gains for a single leg was enough as the controller parameters of a joint are the same for both sides. The cumulative undiscounted reward was used to measure the similarity between the simulation and the expert trajectories.

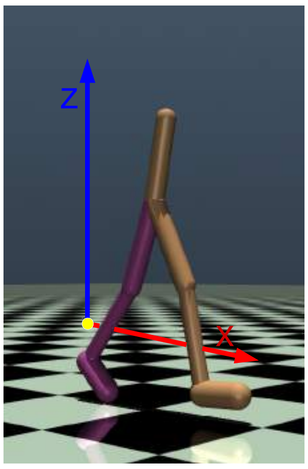
The control parameters of the joints influence each other to a high degree. The controller in the knee has to compensate for the inertial forces caused by the hip joint torque. The required torque in the ankle is dependent on both, the knee and the hip. Given these dependencies, we started tuning the PD gains of the hip in isolation by forcing the knee and the ankle joints to stay in a fixed position. Thereafter, the knee joint parameters were tuned while the hip was already following the desired trajectories. After the knee trajectories were followed to a satisfying degree, the hip PD values were adjusted. As the second last step, we repeated the same procedure for the ankle joint having the hip and the knee joints both follow their trajectories. Finally, the gains of all joints were fine tuned together considering the joint's interactions with each other. During this step, the gains were increased to account for the higher interaction forces during ground contact.

It is important to mention that MuJoCo's built-in position servos only allow to specify a gain proportional to the position error, hence a P gain only. The D part of the PD controller is set implicitly by choosing appropriate joint damping. Table 5.1 summarizes the PD gains of each joint. The capability to track the reference trajectories with the specified gains when holding the torso in the air is shown in the appendix in Figure A.3.

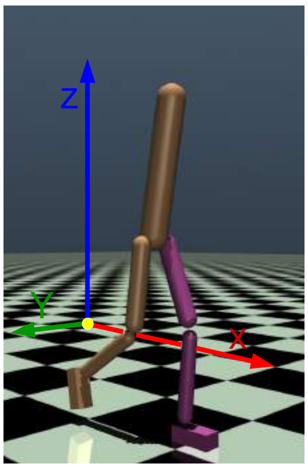
5.2.4. Extension of the Model to 3D Walking

After conducting the first experiments with the simpler 2D model, we got the impression to have reached the lower bound of sample efficiency in the given environment and thus decided to increase the complexity of the task by solving walking in 3D. The following changes were performed:

- 3 additional degrees of freedom (DOFs) in the trunk allowing linear and rotational motions in all three anatomical planes.
- 1 additional actuated DOF in each hip to generate torques in the frontal plane and allow hip adduction and abduction.
- The shape of the feet was changed from being a capsule in the walker2D environment to a box. This decision increased the contact points to 4 instead of having only two, improving the stabilization of the trunk in the frontal plane and making the model more realistic.



(a) Original Walker2d Model



(b) Our Final 3D Walker

Figure 5.1.: **Walker Model Before and After Our Adaptations.** We extend the popular *walker2d* model from OpenAI Gym (a) to the third dimension and adapt its morphology and inertial properties to match that of the subject our reference trajectories were collected from. The resulting walker model (b) is used in all our experiments. The three arrows illustrate the world coordinate frame. Physics are simulated using MuJoCo.

The resulting model significantly increased the kinematics dimensionality from 17 to 27 and added 2 additional motors that have to be controlled. Moreover, the task got much more complicated by requiring to balance in an additional plane, adding the possibility to deviate from the straight walking direction, and introducing multiple additional failure scenarios. The PD gains for the additional two motors were copied from the sagittal hip joint and not retuned.

All experiments presented in this work are conducted using the 3D model with flat feet. Figure 5.1 compares the original 2D environment and our final walker model and shows the world coordinate frames.

5.3. Our DeepMimic Implementation

The DeepMimic framework is to the best of our knowledge the only way to learn controllers able to generate visually human-like walking gaits in physics based simulations. It is therefore the foundation of all our approaches. The basis of DeepMimic is an extension of the RL environment with expert demonstrations provided in form of joint kinematic trajectories. These reference trajectories build the core of the reward function and allow the agent to better explore the state space. Figure 5.2 provides an overview by relating the named components. In the following subsections, we describe our implementation of the 3D Walker environment in the context of the imitation learning framework, changes to the original DeepMimic approach, and present design decisions that we considered necessary to guarantee the learning of stable walking. Our code can be accesses over the following link: `https://github.com/rgalljamov/learn2walk`.

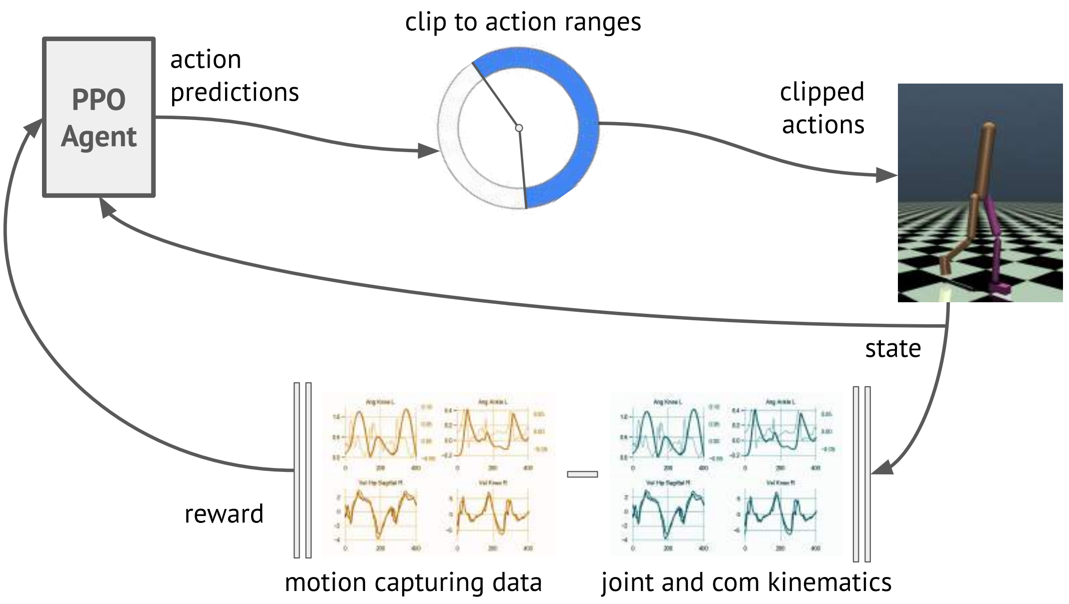


Figure 5.2.: **Overview of our DeepMimic Implementation.** Given the environment's state containing the joint and COM kinematics of the character, the PPO algorithm predicts the actions for the next timesteps. The action predictions are clipped to the corresponding ranges and remapped to the original range in case they were normalized. The reward is calculated based on the difference between the characters posture and the corresponding joint kinematics from the motion capturing data.

5.3.1. State and Action Space

State Space. The state of our environment consists of joint angles and angular velocities in Cartesian joint space as well as COM position and velocities. The COM position in the walking direction (x axis) is not included in the state observations, to make the controller independent of the walked distance. We choose to specify joint kinematics which are easier to measure in a robotic system instead of the relative position and orientation of the character links proposed by Peng et al. (2018a), which are more common in computer graphics.

Besides the kinematics of the current timestep we include the desired walking velocity and a phase variable ϕ indicating the start and the end of a step cycle. Desired velocity is calculated from mocaps as the average COM forward velocity during a step. The phase variable is a monotonically increasing scalar value in the range $[0, 1]$ indicating the start of a step by $\phi = 0$ and the end by $\phi = 1$.

End effector positions were left out. One main reason for including this information was to avoid idiosyncratic motions. These are most often produced by the hands which are not included in our model. Moreover, this information is redundant as it can be obtained by forward kinematics from observed joint angles if necessary.

The states are normalized dimension-wise with the corresponding running mean μ_i and variance σ_i^2 calculated in the environment by the following formula with $\epsilon = 10^{-8}$ for numerical stability:

$$s_{i,norm} = \frac{s_i - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$

Action Space. We consider two different action types during our experiments: joint torques and target angles for the PD position controller in each joint proposed in the original DeepMimic implementation (Peng et al., 2018a). The policy is queried at 200Hz, thus the same control input is applied for 5 simulation frames which is often referred to as *frameskip*. Actions predicted by the policy are clipped to the allowed torque or angle ranges of each joint. In case of normalized action spaces, the actions are clipped to $[-1, 1]$ and mapped to the actual ranges in the RL environment.

5.3.2. Reward Function

The reward function is like in the original DeepMimic approach a weighted sum of individual reward parts encouraging the imitation of the expert reference trajectories. We reward the agent for matching the joint positions with r_p, joint angular velocities with r_v and the COM kinematics by r_c. We decided against using the end-effector reward considering its high redundancy with the position reward and the absence of arms in our simulation model. The weighting of this reward component is added to the position reward. The final imitation reward function is presented in the following equation:

$$r = w_p r_p + w_v r_v + w_c r_c$$
$$w_p = 0.8,\ w_v = 0.1,\ w_c = 0.1$$

The individual reward components at each timestep are calculated by comparing the difference between the appropriate values x in simulation with the corresponding values on the reference trajectories $\hat{x}$ as follows:

$$r_i = \exp[-\alpha_i(\sum_j \|\hat{x}_j - x_j\|^2)]$$
$$\alpha_p = 3,\ \alpha_v = 0.05,\ \alpha_c = 16$$

x takes the role of joint angles for the position reward r_p, stands for joint angular velocities in r_v and corresponds to COM position and linear velocities within r_c. The scaling factors α_i in each reward component are tuned by hand until the reward for different deviations corresponded with our subjective rating.

By using the exponential function with negative exponents, having a minimum deviation of 0, and choosing the weights of the individual reward components to sum up to 1, the reward at each timestep lies in the range $[0, 1]$. To encourage longer episodes and thereby falling avoidance, an alive bonus is added to the imitation reward at each step. The bonus is chosen as 20% of the maximum possible step reward.

5.3.3. Episode Initialization and Termination

The way an episode is initialized and terminated has been shown to drastically influence the learning speed and performance (Peng et al., 2018a). We consider both recommendations in the framework of DeepMimic and implement Early Termination (ET) and Reference State Initialization (RSI). While RSI is adopted without changes, we change the trigger criteria for ET as well as the calculation of the reward in terminal states.

Initialization. Each episode is initialized in a state uniformly sampled from the available mocap data. Therefore, we randomly select one of the steps from the expert demonstrations. Then, we randomly select one point on the step trajectories and use the corresponding reference kinematics to set the initial COM and joint position and velocities in the simulation. This procedure allows the agent to explore all parts of the state space from the training's beginning on leading to quicker convergence.

Episode Termination due to Maximum Duration. Even walking is a cyclic motion and should be framed as an infinite horizon MDP, it is common to define a maximum episode length to diversify the collected experiences (Reda et al., 2020). We terminate the episode after a maximum amount of 3k steps. Given our control frequency of 200Hz and a minimum walking distance in the expert demonstrations of 1.4m/s, the agent is provided with enough time this way to walk at least 21 meters.

Peng et al. (2017) do not report a special treatment of the reward calculation in the terminal state due to maximum episode duration. It is likely they don't distinguish between terminal and non-terminal states. Treating the terminal step the same as all previous steps however, results in the same state-action pair getting different returns and thus ratings based on where in the episode they occur. Accordingly, good actions taken at the episode's end get rated badly and result in contradicting experiences the policy has to learn from (Pardo et al., 2018).

Pardo et al. (2018) investigated different time limits in reinforcement learning and confirmed the importance of correctly treating terminal states. They propose to estimate the return of the terminal action by querying the Q function which approximates the cumulative discounted future return of state-action pairs.

To reduce implementation effort and put more focus on optimizing sample efficiency, we decided to approximate the return of the terminal action from current training statistics. Therefore, we maintain a running mean of the reward during training $\bar{r}_t$ and use it to calculate the average cumulative future return of an average action using the maximum episode duration and the discounting factor γ. This value, which is calculated as follows, is then used as a simple to calculate estimation of the expected return in the terminal state $\hat{R}_t$.

$$\hat{R}_T = \left[\sum_{t=0}^{T} \gamma^t \bar{r}_t \right]$$

Table 5.2.: **Early Termination Conditions.** The table summarizes the maximum allowed deviations in trunk angles (in radians) and COM positions (in meters) before we stop the episode early. The trunk angles in the sagittal and frontal plane as well as the COM Z position indicate early falling detection. COM Y and the trunk angle in the traverse plane a deviation from straight walking.

	Trunk Saggital	Trunk Frontal	Trunk Transverse	COM Y	COM Z
Max Allowed	0.3	0.2	0.5	0.15	2.0
Min Allowed	-0.05	-0.2	-0.5	-0.15	0.95

The proposed way to calculate the reward on the end of an episode allows to freely change the reward function, the scale of the reward, and the episode duration, without the need to adapt other hyperparameters.

Early Termination. The authors of the DeepMimic approach stop the episode early when the trunk or the head of the animated character touch the ground. The reward for the terminal state is set to 0 (Peng et al., 2018a). Using this terminal condition and reward, we observed the agent to not always converge to stable walking. Avoiding falling however is more crucial than human-like appearance when complex and expensive robotic hardware is concerned, which is why we changed the terminal conditions and decided to punish falls much harder.

Our termination criteria is based on early falling detection indicated by a low COM position or significant deviations from the desired trunk angles in all three directions. In addition, we stop the episode when walking direction changes above a certain threshold to guarantee straight walking. Table 5.2 summarizes the allowed trunk rotations and COM heights before falling or direction changes are detected and the episode is terminated.

In addition to terminating the episode earlier, far before the character has touched the ground with its head or torso, we observed it as necessary to provide clear negative feedback in form of a negative reward on terminal conditions. Choosing an appropriate reward turned out to be challenging. Punishing falling lightly with a reward of -1 or -10 had no significant impact on the learned walking stability. Punishing falling with -1000 which is close to the maximum possible episode return resulted in stable walking however took much longer to converge.

We explained the delay in convergence when using high negative rewards to punish falling as being too hard for the short episodes at training's beginning which result in a maximum return of 50. This way, all actions in the episode had a similar negative return and even good actions were rated badly resulting in a weak training signal slowing down policy improvement. The cause for the absence of convergence in case of the low punishments was understood to be an unclear signal to avoid falling. Therefore, we implemented an adaptive ET reward by maintaining a running mean of the episode return and using the negative of for terminal conditions. Following this approach, the agent converged to stable walking almost without exceptions maintaining high sample efficiency.

5.3.4. PPO Hyperparameter Choices

We use the *stable-baselines* implementation of PPO (Hill et al., 2018) in version 2.10.0. It builds upon TensorFlow 1.14 (Abadi et al., 2015) and supports parallelization and the use of a GPU (PPO2). All used packages and their versions are listed in the appendix APX. We use default hyperparameters where possible and specify them otherwise in this Section. An AMD Ryzen Threadripper 2990WX processor with 32 cores is used to collect experiences and an Nvidia GeForce GTX 1080 Ti GPU for updating the network weights.

We train our agents on two variations of the 3D walker environment, one having torque controlled joints and the other using PD position servos to track desired joint angle trajectories. All considered action spaces are normalized to the interval $[-1, 1]$. The hyperparameters were tuned using the torque controlled model. Table X summarizes the most important hyperparameters.

The torque model is trained for 8M timesteps, the PD controlled model for 16M timesteps. We collect a batch of 32k experiences with a fixed policy and perform 4 optimization epochs using Adam (Kingma and Ba, 2014) splitting the data in minibatches of 2k samples. The experiences are collected on 8 parallel environments, each using a different random seed. The VecNormalize environment from stable-baselines is used to collect the states and returns of all parallel environments and calculate running statistics for normalizing both of these values.

We use the same network architectures with two hidden layers of 512 units for the value function and the policy with the exception of the number of outputs, which is 8 for the latter and 1 for the former. Even Peng et al. (2018a) propose to use 1024 and 512 units in hidden layers, networks with the same number of units in hidden layers always performed better during hyperparameter tuning. No hidden layers are shared between the actor and the critic. ReLU (Glorot et al., 2011) is used as the activation function in all layers except for the output layers which have linear activation. We use orthogonal initialization of the network weights and scale these down by a factor of 0.01 after initialization. Especially the scaling of the policy's output layer was reported to be important, making the initial action distributions symmetric around zero and independent of the states (Andrychowicz et al., 2020). The orthogonal matrix is obtained by QR decomposition of a matrix with entries randomly sampled from a normal Gaussian distribution.

All considered action spaces are normalized to the range $[-1, 1]$ for better comparison. The initial standard deviation of the Gaussian policy is set to 0.5 allowing sufficient exploration of the normalized action space. With this choice, only 4.55% of actions lands outside the allowed normalized range. In contrast, when using the default standard deviation of 1 in combination with zero means and normalized action spaces, 27.18% of sampled actions will be outside of the allowed range. Actions outside the range will be clipped to the boundaries and therefore result in an unevenly distribution of actions across the whole space.

To avoid early collapsing to a deterministic policy and continue exploration until the end of the training, the standard deviation is bounded to the interval $[0.1, 1]$. On the other side, too high exploration after convergence to stable walking has been observed to regularly harm the stability. Therefore, we add an entropy punishment with a coefficient of 0.0075 resulting in a smooth decay of the exploration during training that has proven to positively influence convergence behavior and walking stability.

The learning rate is linearly decayed over the course of the training starting with 5×10^{-4} and ending with 1×10^{-6}. Finally, the performance of the learned controller is crucially influenced by the discounting factor γ. To choose an appropriate value, we used the formula $\gamma = \exp(-1/\tau)$ with τ specifying the considered time horizon in number of agent-environment-interactions after which the influence of actions on the return is exponentially decreased (Wright, 2019). The best results were achieved considering a time horizon of 1 second - 200 steps at a control frequency of 200Hz - with the corresponding discounting factor $\gamma = 0.995$.

5.4. Sample Efficiency Evaluation

In this section, we present the performance metrics used for the evaluation of the sample efficiency in the context of learning human-like bipedal walking. Next to conventional metrics, the *Summary Score* is introduced, a new metric developed in this work to evaluate the sample efficiency of learning bipedal walking considering the human-likeness of the learned walking gait as well as the walking stability. At the beginning, the process of evaluating an agent's performance is described.

5.4.1. Evaluation Procedure

To encourage exploration of the state and action space, reinforcement learning agents are trained using a stochastic policy. After training, the stochastic components are removed and a deterministic version of the same policy is used from now on to accomplish the desired task. It is therefore important to evaluate the deterministic policy. Typical learning curves however show the average performance of the stochastic policy during training. Moreover, when training in multiple parallel environments, the performance metrics are averaged over all environments. A low return due to falling in a single of the parallel environments would therefore only insignificantly reduce the metric, averaging out the important information of a fall and hiding the fact of the policy not being fully stable.

Considering the listed scenarios, we propose the following evaluation procedure. We pause the training in regular intervals, load the current policy and evaluate it in a deterministic manner in a single environment using the outputted means of the Gaussian action distributions as predictions. The agent's behavior is monitored for 20 episodes, initializing each in a different step from the reference data.

The initial phase of the step cycle is chosen to be 75%. This way, we avoid initialization in states rich in contact and impacts which are hard to model precisely. At 75% of the step cycle we are no longer in the double support phase halving the contacts and are maximally far away from touchdown and takeoff characterized by high impacts.

The evaluation intervals are chosen as follows. For the first 1M timesteps of training, evaluation is performed every 400k steps, allowing 10 policy updates. After the agent reaches an average walked distance of 5 meters during evaluation, we reduce the interval to every 200k steps and to 100k after 10 meters are reached. After surpassing 20 meters, the interval is again increased to 400k to avoid longer training durations due to frequently repeating evaluations.

To make the curves of individual agents easier to compare, we smooth them using exponentially weighted smoothing with a smoothing factor of 0.25. In detail, the smoothing is achieved by weighting the returns of previous episodes with exponentially decreasing weights and using their sum instead of the return for the current episode.

5.4.2. Performance Metrics for Sample Efficiency Evaluation

In reinforcement learning, sample efficiency is generally measured as the number of required agent-environment interactions until a specified performance threshold is reached. This section answers the question of how to measure the performance of an agent learning to walk stably in a visually human-like fashion as well as how to choose an appropriate threshold. We start by explaining why the return as an obvious performance measure is not suitable for the specific case we consider.

Episode Return. The obvious performance measure in a reinforcement learning setting is the return, calculated as a discounted cumulative sum of future rewards. In our case, the reward measures the similarity between joint and COM trajectories in simulation with the motion capturing data. By punishing falling with a high negative reward, the return also implicitly covers the walking stability which is crucial when training robots to walk. A detailed reward formulation is presented in Section 5.3.2.

However, the return based sample efficiency will be highly dependent on the defined threshold which is hard to specify fairly as following examples portraits: Consider an agent that learns to walk stably reaching 70% of the return after 4M steps and reaching 80% after 10M steps. Another reaches 70% of the return after 6M and 80% after 8M experiences. Setting the threshold at 70%, the first agent is 2M samples more efficient, setting the threshold at 80% the second leads with the same advantage. In addition, the return curve is often very noisy, making it more difficult to specify a suitable threshold and compare the approaches.

Combine two metrics to measure sample efficiency evaluation. To consider the walking stability, we observe the agent over 20 consecutive episodes during evaluation and count the number of times the balance was successfully kept until the end of the episode. We call this metric the *Number of Stable Evaluation Walks*. The episode duration is set to 15 seconds giving the character enough time to reach 22.5 meters when following the desired walking speed of 1.5m/s. To consider the human-likeness of the learned walking gait, we record the average imitation reward over the 20 evaluation episodes.

To consider both metrics at the same time we need to specify a threshold for each of these graphs. The threshold for stable walks is set to the maximum 20 episodes. Empirically we've observed a reward of 50% to be enough to achieve visually human-like walking. If we want to guarantee close imitation of the reference trajectories, we can specify the threshold at 75%. Please note that reaching 100% is impossible in practice due to differences in morphology, contact dynamics and alike. The maximum achieved reward during our experiments was 82%. As a first sample efficiency metric we then obtain the number of sampled experiences until the agent accomplishes to walk stably while replicating the expert trajectories to an satisfying degree. We define stable walking as reaching the episode's end without falling on all 20 evaluation episodes.

5.4.3. Convergence Stability

During our experiments we observed that policies generating stable walking occasionally diverge from the stable behavior if they're trained further. After a following policy update, they are no longer able to reach the episode end on all 20 evaluation runs without falling. The third approach in Figure 5.3 (a) illustrates this behavior. We explain this fact of having learned a policy that has converged to an unstable local optimum. Such a policy is expected to generalize poorly to unseen or noisy states and is therefore undesirable. Therefore, this behavior is important to be considered and is worth giving it a name.

In the context of this work, the term *convergence behavior* is always referring to the number of stable evaluation walks after reaching stable walking for the first time. The convergence is stable, when the agent continues to walk stably after future changes to the policy, hence the number of stable walks remains at 20. It is unstable, when the stable walks curve drops as the agent starts falling on some of the evaluation episodes after future policy updates.

5.4.4. Summary Score Metric

Given the argumentation in previous sections, an optimal metric measuring the sample efficiency of algorithms aiming at learning stable walking should consider the following points:

1. How quickly does the agent achieves stable walking for the first time?
2. How human-like does the character move?
3. How do the first two metrics change when policy is trained further (convergence behavior), e.g. to further improve human-likeness or robustness of the learned walking gait?

The area under the learning curve has been previously proposed as an estimator for the learning speed (Peng and van de Panne, 2017). To fully include the human-likeness and walking stability in our evaluation, we need to consider two different learning curves: the number of stable evaluation episodes and the average step reward during evaluation. As human-like walking without stability is of no interest and stable walking that is not human-like is undesired, we multiply both curves pointwise and compute the area under the resulting curve.

To significantly punish divergence from stable walking reflected by drops in the number of stable evaluation episodes, we take it to the power of 4. To also clearly punish deviations from expert demonstrations but still consider the higher importance of stability over human-likeness, the reward is taken to the power of 2 only. The final metric is calculated with the following formula:

$$\mathrm{SumScore}(\mathrm{n_t}, \bar{\mathrm{r}}_\mathrm{t}, \mathrm{T}) := \frac{100}{T} \int_0^T (\frac{n_t}{20})^4 \, \bar{r}_t^2 \, dt \tag{5.2}$$

T denotes the whole training duration in samples, n_t the number of stable walks and $\bar{r}_t$ the average step reward during an evaluation. $100/T$ normalizes the score to the range $[0, 100]$ given the maximum summary score per timestep is 1. When the summary score is computed during training, it results in a monotonically increasing curve clearly rating different approaches at the end of the training.

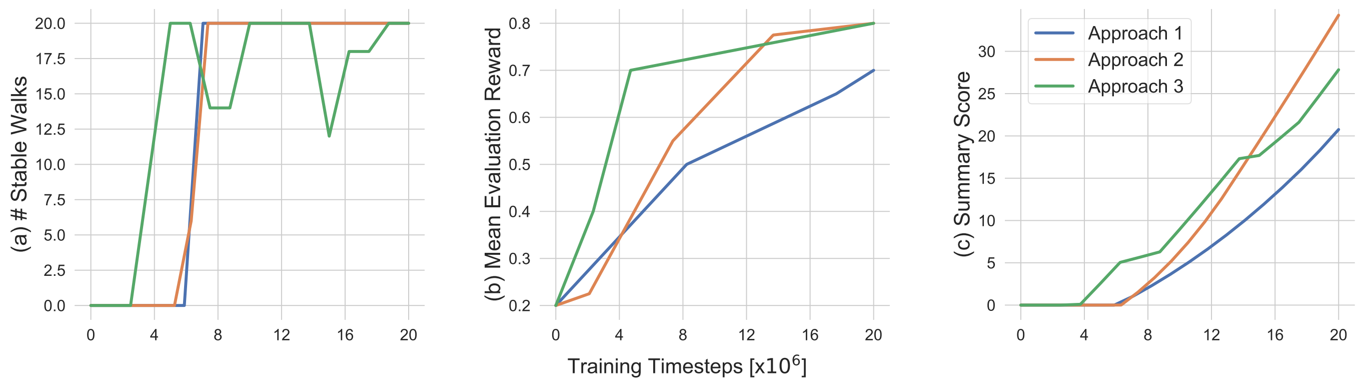


Figure 5.3.: **Advantages of Using the Summary Score** 3 Approaches are compared: Quick convergence at a low reward (1), same time to convergence at a higher reward (2), and quickest convergence at highest reward, but diverging from stable walking (3). Where it is hard to specify thresholds for the first two graphs to correctly rate the three approaches, the final summary score reflects our rating correctly and distinctly.

Figure 5.3 shows the summary score together with the learning curves used for its calculation. The curves were specifically generated to highlight the advantages of our metric. Looking at the number of stable walks alone (a), the first two approaches perform equally well. The summary score (c) considers the higher rewards of the second approach and rates it significantly better compared to the first.

Approach 3 illustrates the punishment of a bad convergence behavior. The agent is the earliest to achieve stable walking. However, the performance diverges during the following policy updates and the agent is no longer able to maintain the balance on all 20 evaluation walks (a). Despite also achieving a higher reward during the whole training duration (b), this approach is rated second after the orange approach which shows a significantly better convergence behavior.

5.5. Approaches to Improve Sample Efficiency of DeepMimic

Our first attempt to sample efficient learning of bipedal walking was to adapt the DeepMimic approach for our specific learning scenario. After this was accomplished, our focus changed to further improving the sample efficiency over this baseline. Figure 5.4 shows an overview of the considered approaches shortly summarized in the following.

Using PD position servos in the joints as proposed by the authors of DeepMimic (Peng et al., 2018a), it was possible to generate state-action pairs out of the reference trajectories and pretrain the agent in a supervised manner, an approach known as Behavior Cloning. Next, we investigated the influence of different action spaces on learning speed and performance.

Considering the results of previous experiments, our second group of approaches were implemented using the torque controlled walker model. The goal here, was to induce prior knowledge about the task in the learning process. We considered the symmetry of the desired motion and included additional information into the state observations. Finally, we tested techniques proven effective in the context of deep or deep reinforcement learning including experience replay, dimensionality reduction and hyperparameter scheduling.

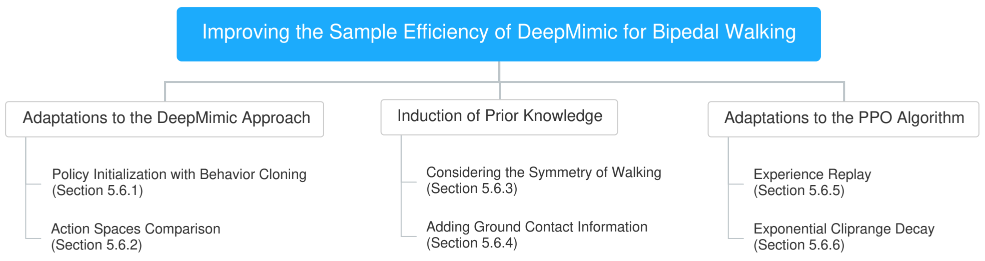

Improving the Sample Efficiency of DeepMimic for Bipedal Walking
Adaptations to the DeepMimic Approach
Induction of Prior Knowledge
Adaptations to the PPO Algorithm
Policy Initialization with Behavior Cloning (Section 5.6.1)
Action Spaces Comparison (Section 5.6.2)
Considering the Symmetry of Walking (Section 5.6.3)
Adding Ground Contact Information (Section 5.6.4)
Experience Replay (Section 5.6.5)
Exponential Cliprange Decay (Section 5.6.6)

Figure 5.4.: **Overview over Considered Approaches to Improve Sample Efficiency**. We consider multiple methods targeting at improving the sample efficiency of our DeepMimic implementation. The approaches can be categorized in changes to the DeepMimic approach, incorporation of prior knowledge into the training process as well as adaptations to the PPO algorithm.

5.5.1. Better Initial Policy with Behavior Cloning (BC)

Analyzing the outcomes of our first experiments, we observed all learning curves to show a very slow improvement for quite a significant duration after the training's start. We explained these observation by the random weight initialization leading to a poor initial policy that is far away from being even sub-optimal. It took a high number of updates to bring the policy in a shape to quicker improve the performance.

To improve the initial policy, we decided to pretrain it on the reference trajectories in a supervised manner. This approach is known as Behavior Cloning (BC) (Schaal, 1997) and has been often applied in the field of robotics (Atkeson and Schaal, 1997; Argall et al., 2009; Kober and Peters, 2010). The basis of this approach are expert demonstrations in form of state-action-pairs. When we use the walker model with position servos in each joint, our policy observes the joint kinematics of the current timestep and has to predict the joint positions at the next point in time. In this case, it is easy to generate state-action-pairs from the reference data and use it to pretrain the policy.

Even knowing the approach is likely to fail as proposed by the literature (Ross et al., 2011), BC is a must-try approach when aiming at higher sample efficiency. When succeeding, it allows to pretrain the policy in a fully supervised manner without a single agent-environment interaction. Moreover, the drawbacks of pure behavior cloning were planed to be compensated by combining it with reinforcement learning (Kober and Peters, 2010). We expected the pretraining on mocap data to either produce useful features that will speed up learning process in a transfer learning fashion (Pan and Yang, 2010) or less likely already learn a near-optimal policy. This policy was expected to overfit to the state distribution in the expert demonstrations (Ross et al., 2011), but should be improved by continue training it in a reinforcement learning setting.

The value function network could have also been pretrained from expert demonstrations. However, since the value function loss quickly converged to zero during training and was unlikely the bottleneck to better sample efficiency, we haven't considered this scenario in our experiments.

Generation of the Supervised Learning Data Sets from Reference Data. To pretrain the policy, we need to transform the expert reference trajectories into state-action-pairs. The policy for the PD position controlled model gets the current environment state as input and outputs desired joint angles for the next timestep as actions. The state consists of joint and COM kinematics as well as the desired walking speed and a phase variable. The extraction of this information from the reference data is straightforward. The actions specify the target joint angles at the next timestep and are estimated from the next point on the mocap trajectories.

The generated dataset was shuffled to decrease the violation of the i.i.d. assumption due to states being influenced by former actions (Ross et al., 2011). Finally, we split the data in a train, validation and test set using 80, 10 and 10 percent of the data respectively. After the hyperparameters were tuned, we merged the validation and training set.

Pretraining of the Policy Network. To implement BC, the Keras module of the TensorFlow framework (Abadi et al., 2015) is used. We build a fully-connected neural network having the same size and activation functions as the policy network used in PPO. The weights of the network are also initialized as an orthogonal matrix with entries scaled down by 0.01.

After comparing different loss functions the Mean Absolute Error (MAE) was chosen to be minimized during pretraining. We trained the network for 200 epochs and used Adam (Kingma and Ba, 2014) with linear learning rate decay from 0.01 to 0.0005 for the weight optimization. Even the learning curves didn't signal significant overfitting, we still utilized L2 regularization (Ng, 2004) with a coefficient of 0.0005 hoping for better generalization to unseen states. The absence of overfitting can be explained by the high variance in the data with all steps having slightly different trajectories as can be seen in the appendix (Figure A.2).

Usage of the trained model. After training the model to output desired joint angles for a given state on the reference trajectories, we save the model parameters. During the network initialization step of PPO, we then load the pretrained network instead of initializing a new random network and begin the training. During this step, we tried loading only the hidden layers of the pretrained network as well as the whole network using the outputs as the mean actions of a Gaussian distribution.

5.5.2. Sample Efficient Action Spaces

Peng et al. (2018a) propose to use target angles for PD controllers in each joint as the agent's action. Next to this action space, we also investigate outputting joint torques as well as differences to the current joint angles which are added to the current posture to determine the target angles for the PD servos. We abbreviate the latter approach with outputting *angle deltas*.

Drawbacks of the Target Angle Space. Outputting target angles is expected to often result in unrealistic target angles. The policy can output a desired angle of $45°$ even the current angle is currently at $-30°$. The policy however is queried at 200Hz allowing only 5 milliseconds to reach the desired angle. In this time, the motors realistically are only able to change the current joint angle by a small amount. Moreover, the target angles of consecutive time steps can vary a lot resulting in very noisy sequences like $[0°, 120°, -30°, 40°, ...]$. Thus, when target angles are outputted by the policy, there is a high chance to make unrealistic predictions.

Besides being unreachable, unrealistic target angles come with multiple drawbacks. PD controllers choose the motor torque by scaling the difference between the current and desired joint position (Equation 5.1). Is this difference high, the joint torques are high too. After surpassing a certain difference threshold, the motors will go into saturation and output the maximum peak torques. In a real robotic system, this results in high energy consumption and significant wear of drive train components. High sudden torques are in addition likely to have a perturbing effect on the system and prevent stable walking. Finally, joint angle trajectories will be unsmooth and jittering motions get highly probable.

Outputting angle deltas for realistic target angles. To avoid unrealistic target angles, we propose to output angle deltas: small angle differences which are added to the current joint positions to obtain the target angles for the next timestep. By choosing an appropriate maximum angle difference, the risk of target angles that cannot be reached can be drastically reduced or fully eliminated. We expect this approach to significantly improve the smoothness of the joint trajectories, reduce motor torques and increase the quality of the learned motion. More importantly, this approach promises to increase the learning speed by strongly limiting the action space at each point in time to only angles close to the current joint configuration.

To output target angle deltas, the maximum possible angle differences for each joint have to be specified. These hyperparameters can be easily estimated from the reference trajectories by looking at the maximum angular velocities of individual joints. To consider out-of-distribution states requiring higher torques and to be able to recover from perturbations, we scale up the maximum velocities observed in the mocap data. After testing different values, a scaling factor of $a = 4$ was declared to be the best choice.

The maximum possible angle differences further depend on the time interval between two control steps. Therefore, the maximum possible velocities, specifying the angle change per second, have to be scaled down in proportion to the control frequency f_{CTRL}. The higher the control frequency, the smaller the time between two control commands and the smaller the possible change in the angle. The following simple formula sums up the calculation of the individual maximum delta angles Δq_{max} from the corresponding maximum joint velocity, denoted by v_{max}:

$$\Delta q_{max} = a \frac{v_{max}}{f_{CTRL}} \tag{5.3}$$

To guarantee a fair comparison, we normalize all action spaces to the range of $[-1, 1]$ and remap the predicted actions within the environment to the specified joint, torque or delta ranges. The standard deviation for all joints is uniformly initialized at $\sigma^2 = 0.5$. With the initialization of the action means around zero (Section 5.3.4), this choice allows to sufficiently explore the whole action space, exceeding its bounds with a probability of only 4.5%.

5.5.3. Symmetry Prior Induction

Bipedal locomotion in humanoids is a symmetric motion w.r.t. the sagittal plane. In this Section we present two possibilities proposed by Abdolhosseini et al. (2019) to induce this prior knowledge in the training process. Before that, we explain the required transformations to mirror experiences given their representation as a one dimensional state vector. The vector contains the joint position and velocities as well as COM kinematics, the desired forward velocity and the phase variable (see Section 5.3.1 for details).

To mirror the state vector, we have to:

- swap the left leg joint kinematics with their correspondences from the right leg and
- negate the hip angles in the frontal plane.
- The phase variable and desired velocity remain unchanged. COM kinematics as well as trunk angle and angular velocities also stay at the same position in the state vector.
- The COM displacement in y-direction, the trunk angle around the x and the z-axes in addition change their signs.
- The same applies for the corresponding COM and trunk velocities.

Mirroring the action vector is much easier, requiring only to swap the left leg action positions with the right leg actions and negate the actions of the hip in the frontal plane. In the context of this work, we denote (s'_t, a'_t) as the mirrored state-action pair of (s_t, a_t) at a given timestep t.

Due to asymmetries in every human walking style, the collected trajectories cannot be fully symmetric. To test the effect of asymmetry in the expert demonstrations, we conduct our experiments using the original reference trajectories as well as the artificially symmetrized version.

1. Duplicating and Mirroring Made Experiences (DUP). Joint angle and angular velocity trajectories collected during taking a step with the left leg can be mirrored to get the corresponding trajectories of a step with the right leg. Therefore, it is possible to generate additional artificial experiences from true agent-environment interactions. This approach is similar to data augmentation techniques used in supervised learning (Shorten and Khoshgoftaar, 2019).

To duplicate the amount of training data following this approach - hence the abbreviation DUP - we need to estimate additional information about the mirrored experiences. Besides states and actions, our deep RL algorithm PPO requires the resulted return of the state-action pair $R(s_t, a_t)$, the value of the observed state under the current policy $V^{\pi}(s_t)$ as well as the negative logarithm of the action probability $-log[\pi(a_t|s_t)]$.

We implement two ways to approximate the required values for the mirrored experiences. Within the first attempt, we let the agent collect a batch of experiences (s_t, a_t, r_t, s_{t+1}) and calculate the discounted returns $R(s_t, a_t)$, the state values $V^\pi(s_t)$ and the negative log probabilities of the actions in corresponding states $-log[\pi(a_t|s_t)])$. We then mirror the states and actions and copy these values for the mirrored states and actions. This decision implicitly makes the very strong assumption of mirrored states having the same value under the current policy and mirrored actions having the same probabilities as the originally predicted actions: $\pi(a'_t|s'_t) = \pi(a_t|s_t)$ and $V^\pi(s'_t) = V^\pi(s_t)$.

In the second approach, we query the value function and the policy networks to calculate the true state values and action probabilities under the current policy. To get the state values, we load the current value function network weights, including the bias terms, stack the mirrored states into a matrix and perform the required multiplication operations, considering the activation functions of the network layers. This way, we get a more precise value estimate for the mirrored states under the current policy.

The procedure of getting the true action probabilities is similar. First, we load the current policy network weights and biases as well as the standard deviation of the action distributions. The stacked mirrored states are then propagated through the policy network giving us the means of the Gaussian action distribution for each state. Given the distribution, we calculate the negative log probability of the mirrored actions and use these in the PPO update.

Without further changes, this approach rarely crashes the training due to NaNs in the policy update equations. The reason for that is the near zero probability of some of the mirrored actions. We work around this issue by clipping the action probabilities to an acceptable range or delete experiences with mirrored actions that are impossible to occur under the current policy. The threshold for the minimum considered action probability is set to one fifth of the minimum probability of the actions in the observed trajectory τ:

$$p_{min}(a'_t) = \frac{1}{5} \min_{(a_t, s_t) \in \tau} \pi(a_t|s_t)$$

Phase-based Mirroring (PHASE). The joint and COM kinematics of a step with the left leg are identical with the mirrored kinematics of a step with the right leg. Another approach to consider the symmetry of walking is therefore to train the policy only outputting actions for stepping with the left leg. To get the actions for the step with the other leg, the observations are mirrored, corresponding actions are calculated and then again mirrored to get the actions for the counterpart leg. The big advantage of this approach is the very simple implementation and the high generalization of the approach to other algorithms as the changes are exclusively implemented in the environment.

5.5.4. Adding Ground Contact Information to State Observations

The walking motion can be split into multiple phases. The contact of both feet with the ground plays a crucial role in determining the individual phases (Li and Dai, 2006). Therefore, we want to investigate how adding the ground contact information to the state observation will influence the sample efficiency of the learning process as well as the final walking performance.

We extend the state space by a binary flag for each foot indicating if it has ground contact or not. Next to binary values, we also investigate a value of -1 to indicate the absence of contact. Further, we try introducing a separate flag for the double stance phase where both feet are in contact with the ground. In this scenario, we compare the usage of a three dimensional binary vector indicating the double stance phase by $(0, 0, 1)^T$ as well as a dense vector completely filled with ones when both feet touch the ground.

5.5.5. Experience Replay Despite Being On-Policy

One of the reasons preventing a higher sample-inefficiency of the PPO algorithm is its on-policy nature, allowing it to use only data collected by the current policy to update the network parameters. Experiences collected with previous versions of the policy are no longer suitable to estimate the gradient of the current policy, increasing the need for more agent-environment interactions. Off-Policy algorithms which are not limited by that constraint, often store experiences collected with previous versions of the policy in a replay buffer and use them multiple times improving the sample efficiency.

Being a trust region method, we can use PPO with a small clip range and guarantee consecutive policies to be similar to each other. This way it might be possible to reuse prior experiences to estimate the gradient of the current policy. In this work, we want to investigate if former experiences can be used to improve sample efficiency and if so, how many prior experiences can be maintained until the policy gradient can no longer be correctly estimated.

To answer these questions, we save collected experiences in a replay buffer. An experience is a tuple consisting of a state-action pair (s_t, a_t), the corresponding return $R(s_t, a_t)$, the predicted state value under the current policy $V^{\pi}(s_t)$ and the negative log probability of the taken action $-log[\pi(a_t|s_t)]$. The buffer is implemented as a first-in last-out queue that is shortened to the specified buffer size after each update. Newly collected experiences are merged with those from previous iterations into a new batch. Minibatches are then uniformly sampled from this mixed collection of fresh and previous experiences and used to update the policy.

To maintain the same batch and minibatch sizes for a fair comparison, we decrease the number of collected experiences per batch by the number of considered batches per update, being one more than the buffer size. A simple implementation of the replay buffer reuses prior experiences with the state values and action probabilities calculated with the old policy and value function: $V^{\pi_{old}}(s_t)$ and $-log[\pi_{old}(a_t|s_t)]$. These values are expected to deviate from true values the older the policy the experiences were gathered with. To correct for these deviations, we can query the current policy and value function networks to update the values and probabilities of past experiences as described in Section 5.5.3.

5.5.6. Exponential Cliprange Decay

PPO builds upon trust region policy methods and limits the change in action distributions of consecutive policies. Instead of constraining the KL-divergence, this algorithm defines a clipped surrogate objective function defined in Equation 4.1. The maximum allowed change is controlled by the cliprange ϵ which we schedule to exponentially decrease over the course of the training from 0.55 to 0.1 with the following formula where T denotes the training duration:

$$\epsilon(t) = 0.1 + 0.45\, e^{-5t/T}$$

Our motivation for the exponential decay of the cliprange is the following. As the policy is initialized with random weights, it performs poorly at the training's beginning. Tens to hundreds of updates are necessary to result in a near-optimal behavior. By restricting the maximum possible behavior change during a policy update, the number of required updates further increases. Choosing a high cliprange can help quickly improve upon the random policy. As far as the policy however reaches near-optimal performance, big changes to the action distributions can have a destructive effect on the policy's behavior and thus should be limited. An exponentially decaying schedule allows big changes to the initial random policy, moderate changes during the following early stage of the training and finally strongly limits the difference of consecutive policies as the agents performance increases.

6. Results

We evaluate each approach on a minimum of 5 different random seeds. The number of required samples to convergence portraits the whole distribution over the experiments as a violin plot. All other curves show the mean and one std over the seeds.

For each approach we in addition report two groups of learning curves. The first focuses on metrics used to evaluate the sample efficiency. With the exception of the episode return, the three other subfigures are collected during deterministic evaluation of the current policy (Section 5.4.1). From left to right, the graphs show:

1. **Number of stable walks** out of 20 evaluation runs as a measure for the walking stability and convergence behavior.
2. **Mean step reward** during the 20 evaluation runs approximating the average similarity of produced joint and COM trajectories with the expert demonstrations. We highlight the achieved reward at convergence as well as the required time to achieve 75% of the reward indicating visually fully human-like walking.
3. **Summary Score** as an overall measure for the suitability of an approach for quickly learning bipedal walking, considering the sample efficiency in combination with walking stability and human-likeness of the learned walking gait.
4. **Normalized episode return** as the standard learning curve in deep RL.

We furthermore highlighted important points for the sample efficiency evaluation on these graphs with markers and sum them up in a table: The time to convergence with corresponding reward and return values at that point as well as the time required to reach an average imitation reward of 75% empirically guaranteeing visually smooth and human-like walking. The required timesteps are computed individually for each of the random seeds and are then averaged to get the time for an approach. The graphs however are averaged pointwise and show the mean over the 5 runs. Therefore, the highlighted convergence timesteps do not correspond to the point on the stable walks graphs where the mean reaches the maximum value of 20 steps for the first time. The same procedure causes the 75% reward markers being positioned not exactly at a value of 0.75.

The second group of curves breaks down the imitation reward obtained during training in its three components describing the imitation accuracy of different parts of the expert demonstrations: joint positions, joint angular velocities, and COM kinematics.

All metrics except of the summary score and the number of stable walks are smoothed for better comparison using the exponential window function with a smoothing factor of 0.25. Algorithmic design choices and hyperparameters listed in Section 5.3.4 are held constant across all implementations where possible. Exceptions are explicitly mentioned during the results presentation.

6.1. Comparison of Different Action Spaces

The baseline for the following comparison is our implementation of the DeepMimic approach adjusted to maximize its effectiveness in learning stable bipedal walking (Section 5.3). We use this framework with three different action spaces: the default target angles for PD position servos in each joint, target angle deltas and joint torques (Section 5.5.2). All action spaces are normalized to the same range $[-1, 1]$ which is remapped to the actual ranges before being applied to the environment.

Figure 6.1 shows the distribution of required samples to achieve stable walking over 5 random seeds. Training the policy to output target angles requires an average of 9.6 million interactions with the environment until the agent achieves stable walking for the first time. By outputting angle deltas, stable walking is achieved using 41% less samples. In case of joint torques, the sample efficiency is more then doubled compared to the baseline, requiring 5.1 million (53%) less samples. Therefore, the torque model training is stopped after 8 million steps compared to 16 for the other approaches. The violin plots further show a reduction in variance across random seeds of more then 2 times when using our action spaces.

Figure 6.2 compares the learning curves of the considered action spaces highlighting the points important for the sample efficiency evaluation. Our proposed action space achieves significantly better scores across all considered metrics with the torque model outperforming the angle deltas. The average evaluation reward and the training return are at all points in time higher compared to the baseline.

The number of stable walks (a) underlines the earlier convergence of our approaches. The baseline model reaches the first stable walks almost at the same time as the other two approaches. Our action spaces thereafter however, require less than 2 milliion timesteps to converge to fully stable walking while the baseline shows a more then twice slower improvement and higher variance across the seeds.

The evaluation reward (b) shows a significantly closer replication of the expert demonstrations throughout the whole training duration. At convergence, the torque approach achieves an average step reward of 66% compared to 53% of the baseline while angle deltas achieve 58%. Interestingly, the rewards of all methods continue to rise after converging to stable walking. However, only our methods reach the 75% reward, empirically guaranteeing a visually human-like walking gait. Even after 16 million steps, the baseline only reaches an average episode reward of 69%.

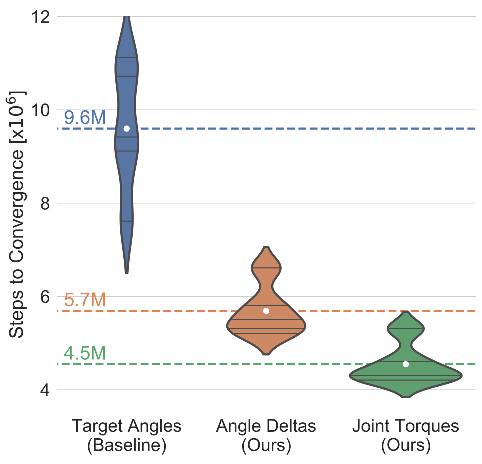


Figure 6.1: **Sample Efficiency of different action spaces.** The violin plots show the distribution of required agent-environment interactions until the agent achieves stable walking. Stable walking is defined as walking 20 meters on 20 consecutive evaluation episodes without falling. Lower numbers correspond to better sample efficiency.

Table 6.1.: **Sample efficiency comparison.** Both our approaches achieve better values across all metrics with the torque model performing the best. Convergence is defined as learning to walk without falling. 75% of the maximum reward indicate visually human-like walking.

Approach	Summary Score	Steps to Convergence	Reward at Convergence	Steps to 75% Reward	Final Reward
Target Angles	7.2 ± 1.0	9.6 M ± 1.2 M	0.53 ± 0.05	not reached	0.69 ± 0.03
Angle Deltas (Ours)	15.1 ± 0.7	5.7 M ± 0.5 M	0.58 ± 0.05	7.2 M ± 0.5 M	0.81 ± 0.01
Joint Torques (Ours)	**23.4 ± 3.2**	**4.5 M ± 0.4 M**	**0.66 ± 0.06**	**5.1 M ± 0.4 M**	**0.82 ± 0.01**

The summary score (c) is more than twice as high for the delta model as the baseline's value and is more than three times as high when outputting joint torques. The return (d) as the typical learning curve in deep RL supports the presented results and shows different values for each approach at convergence. Table 6.1 summarizes the most important points on these curves and highlights the best results, all achieved by the policy outputting torque commands.

A comparison of the individual reward components in Figure 6.3 shows a different rating of the action spaces. Both of our approaches show a higher position rewards (a) compared to the baseline after 2 million timesteps and higher velocity rewards (b) from beginning on. After 8M samples, the torque model mimics the joint positions the best, the delta model the second best. After further training however, the angle delta manages to reach and surpass this value. Interestingly, the same model better mimics the reference joint velocities compared to our second approach through the whole training duration. The baseline in contrast constantly increases its capability to match the angular velocities but never surpasses 2% of the maximum possible reward.

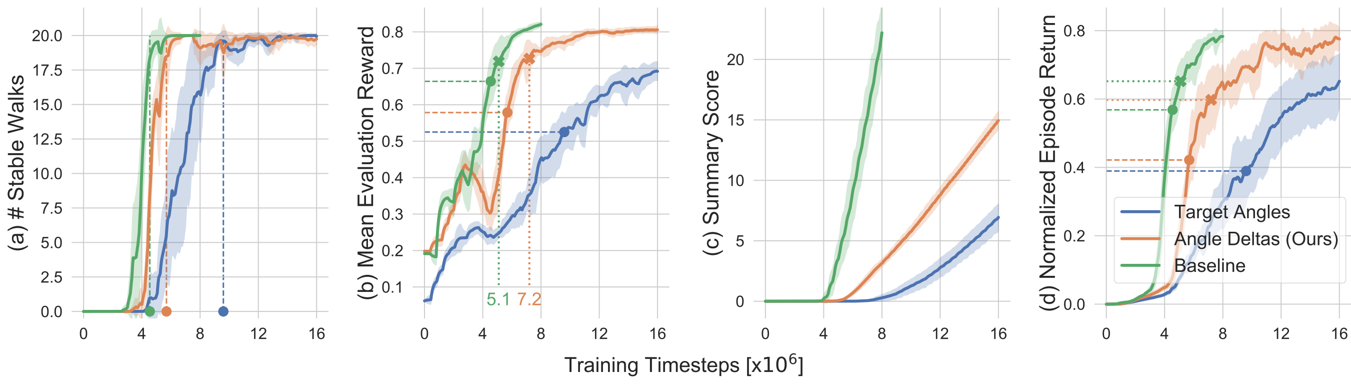


Figure 6.2.: **Benefits of changing action space.** We compare 3 action spaces: target angles (DeepMimic), angle deltas (ours), joint torques (ours). On all metrics, angle deltas improve upon target angles and joint torques perform the best. Circular markers and dashed lines highlight the convergence point. X-markers and dotted lines emphasize the duration until 75% of imitation reward are reached, necessary to ensure visually human-like walking.

In general, we see higher scores for matching the joint angles from the expert demonstrations compared to the angular velocities. Moreover, all approaches start with a high COM reward which monotonically decreases over time. The better a model performed in the learning curve comparison, the stronger the decrease.

Finally, we collect the joint position and velocity trajectories of an agent from each action space on two consecutive gait cycles. Due to high similarity between both legs, only results of the right leg are shown. We clearly see in Figure 6.4 (a) that the agent outputting target angles doesn't manage to generate smooth joint velocity trajectories. On the contrary, they are extremely noisy. Also the joint position over time includes jittering being especially high for the ankle joint. Links to videos showing the corresponding motions are linked in the Readme file of our GitHub repository under `https://github.com/rgalljamov/learn2walk`.

Our action spaces in contrast, displayed in Figures 6.4 (b) and (c), generate smooth joint angle trajectories very similar to the once from the mocap data. Even the angular velocity trajectories overlap significantly less, the obtained trajectories still show smooth transitions. A significant difference in trajectory smoothness between our two proposed methods is not observed. Figure 6.5 shows multiple snapshots of the character controlled by the torque model.

In summary, both of our approaches significantly improve across all metrics upon the baseline. The torque model achieves the best sample efficiency requiring 53% less data to converge to stable walking. With 5.1M steps it also requires the least amount of samples to reach 75% of the imitation reward. When the considered robotic hardware is equipped with PD position servos in each joint, a higher sample efficiency is reached by outputting angle differences instead of target angles. This action space reduces the required samples to convergence by 38%, generates smooth joint trajectories and is able to replicate the expert demonstrations closely after 7.2M steps. The rating of the three compared approaches is clearly reflected in the summary score which evaluates the overall suitability of a method for quickly learning to walk on two legs using deep RL.

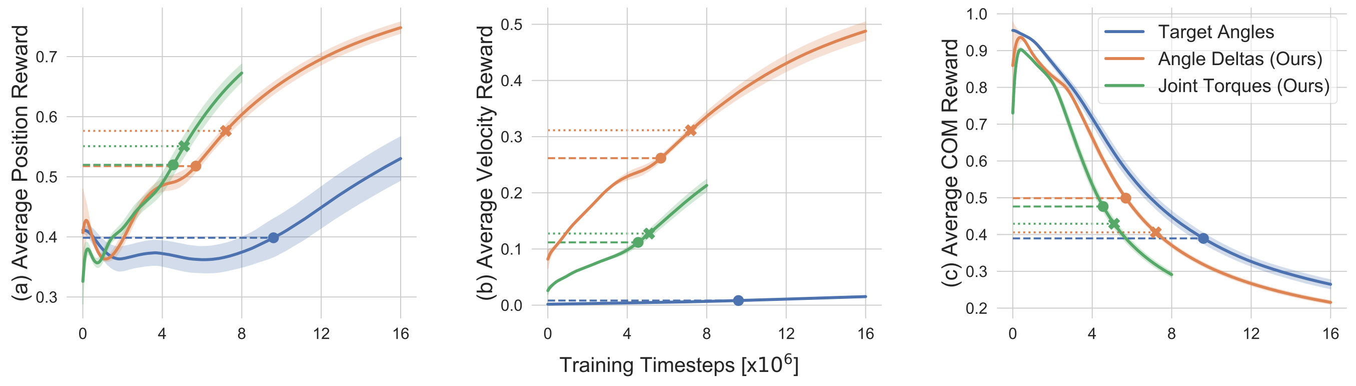
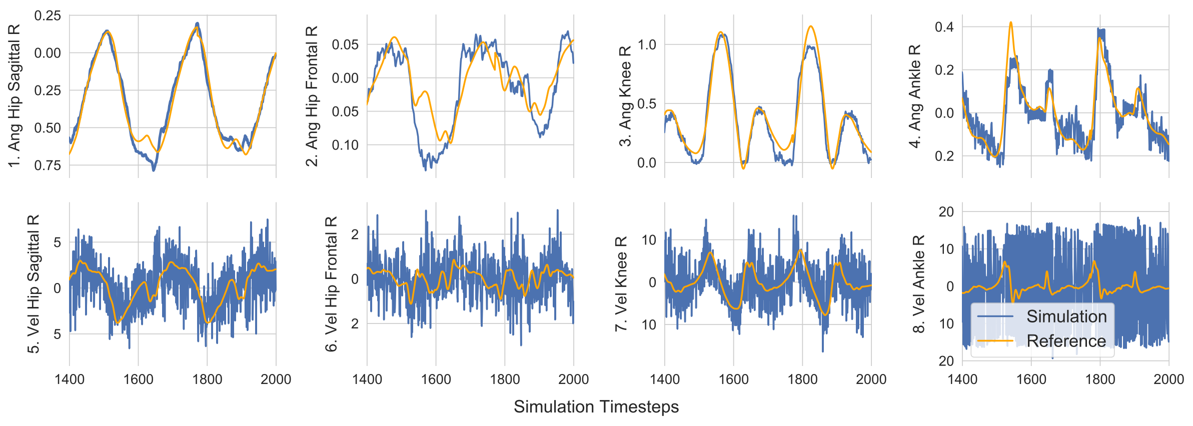
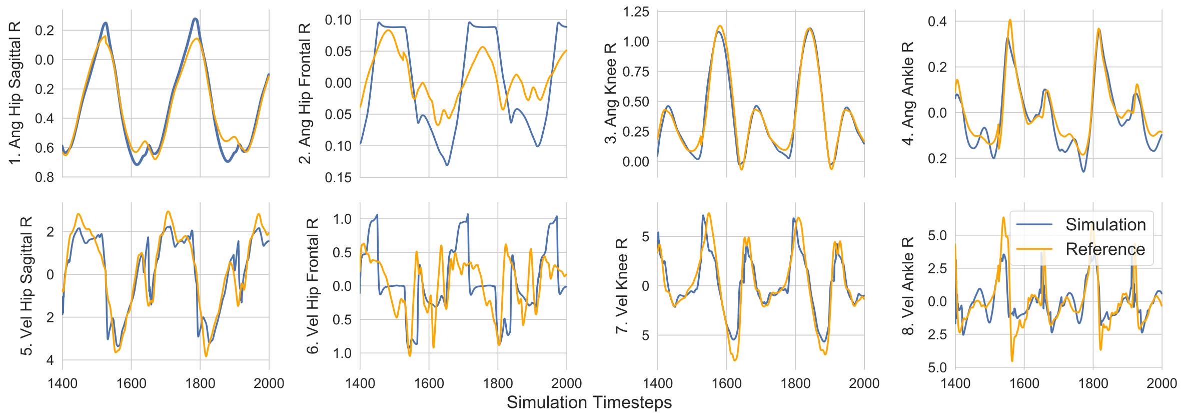


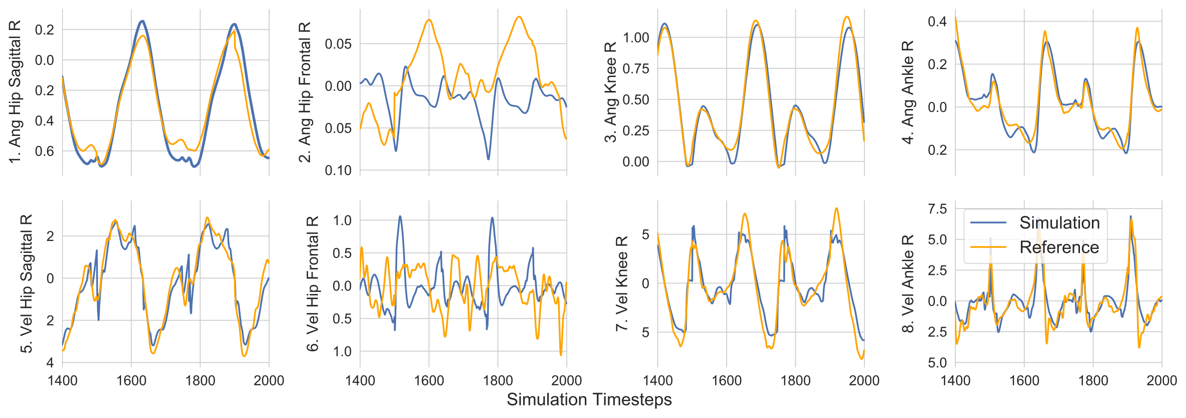
Figure 6.3.: **Individual imitation reward components during training.** Our approaches outperform the baseline in replication of joint kinematics (a,b). Angle deltas show higher velocity values, the torque model the highest position reward during its halved training time. The baseline model stands out having an almost constant velocity reward of 0 (b). No approach manages to match the COM trajectories (c). Circles and crosses mark the values when reaching stable walking and 75% of imitation reward respectively.



(a) Target Angles (Baseline)



(b) Angle Deltas (Ours)



(c) Joint Torques (Ours)

Figure 6.4.: **Smoother joint kinematic trajectories with alternative action spaces.** Comparison of the right leg joint kinematic trajectories between the motion capturing reference data (blue) and the trajectories collected from the models after training (orange) for two gait cycles. The upper rows display joint angles. The corresponding angular velocities are positioned below. Our approaches result in significantly smoother kinematics, especially concerning joint velocities.

6.1.1. Result Interpretations

Why are angle deltas better than target angles? Outputting normalized angle deltas instead of target angles significantly improves sample efficiency, the similarity with the expert demonstrations and results in much smoother joint trajectories. We believe the latter is the most important observation and the reason for the other two improvements.

Our choice of the action space can be seen as a strong inductive bias guaranteeing smooth joint trajectories. It explicitly limits the difference in joint angles between consecutive control steps. On the contrary, outputting target angles allows arbitrarily high angle difference between the current and desired angle. A high difference in joint angles between two control steps corresponds to a high desired joint velocity. By having a sequence of highly different target angles at consecutive timesteps like $[0°, 120°, -30°, 40°, ...]$ we get a corresponding noisy target velocity sequence. This is very likely the reason for the high oscillations in joint velocities illustrated in Figure 6.4 (a). These noisy trajectories are also reflected in the reward component evaluating the imitation of joint angular velocities. These values stay close to zero for the whole training (Figure 6.3 (b)).

High deltas in addition can bring the PD controller into saturation and result in high joint torques. Sudden changes in angle differences cause sudden high jumps in the torque. This might help to quickly recover from undesired states but is at the same time likely to have a perturbing effect on the walker's stability, as Figure 6.2 (a) suggests. We see the model outputting target angles reaching its first stable walk almost at the same time as our approaches. However, while our methods from here on converge to fully stable walking in under 2 million steps, the baseline progresses more then twice as slowly until it manages to maintain the balance on all 20 evaluation episodes. The recovery effect of the high joint torques can explain the early achievement of first stable walks.

On the other side, sudden joint peaks are likely to perturb the bipeds balance and bring him to fall during a few evaluation episodes. We believe this to be the reason for the slower improvement of the baseline from first to all 20 stable walks in Figure 6.2 (a). In addition, high joint torques cause high joint accelerations quickly increasing the joint velocities and making them spiky as seen in Figure 6.4 (a).

Why are torques better then target angles? The superior performance of the torque model may result from the PD controller gains being sub-optimally tuned. These parameters were tuned by holding the trunk in the air without any foot contact with the ground. Next to tuning the control parameters by hand, we tried tuning

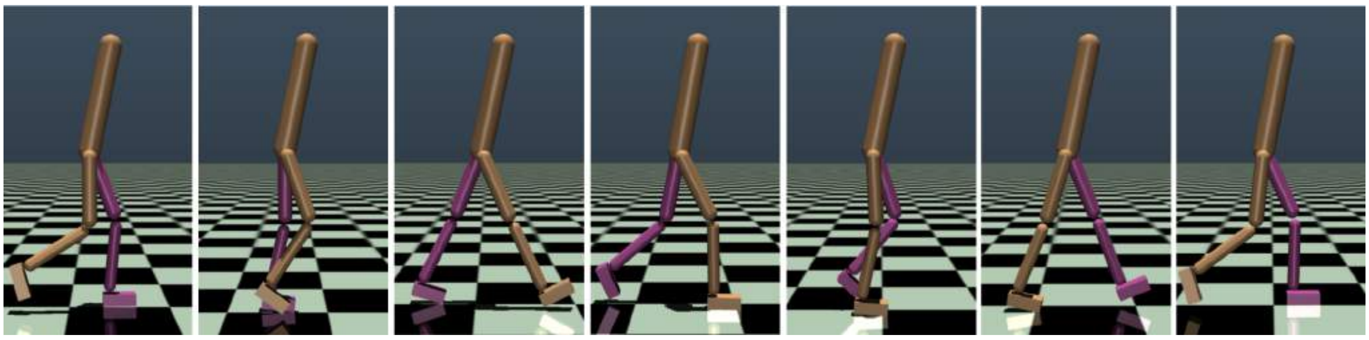


Figure 6.5.: **Human-like walking with torque control.** Seven snapshots of a gait cycle recorded using the final policy outputting joints torques. Please note the symmetry of the gait by comparing corresponding postures during the left and the right step. Corresponding postures are three snapshots apart.

them using Bayesian Optimization (Frazier, 2018). This optimization resulted in a better tracking of the expert trajectories during tuning when the trunk was in the air. However, the training performance with these optimized gains was always worse compared to the hand tuned controllers. We interpreted the optimization to overfit to the flight phase and used the hand tuned PD gains for our experiments. As we expected also the hand-tuned gains to overfit, we significantly increased the control parameters after tuning them in the flight phase and tested if they're able to follow the reference trajectories also during contact with the ground. However, it is very likely the gains are still not optimal and make learning more difficult.

Another reason for the advantage of torque control might be the high control frequency of 200Hz. Peng and van de Panne (2017) as well as Reda et al. (2020) also compare different action spaces for locomotion tasks and report torque control to improve with higher control rates. On the other side, higher control frequencies result in shorter time intervals for a PD controller to reach a desired target angle. In combination with the target angle space which allows outputting angles far away from the current joint positions, the target angles are likely to be unreached which is expected to negatively influence the learning performance.

A final advantage in favor of the torque model is the symmetry of the action space especially in combination with our specific policy initialization. The joint torque range is symmetric w.r.t. zero. When in contrast outputting target angles, the action space is limited by the joint ranges. These are asymmetric around zero. In case of the knee it even only allow positive values from 0 to 2.6. Our policy is initialized with normal Gaussian action distributions for each joint as proposed by Reda et al. (2020). All action spaces are normalized to the interval $[-1, 1]$. Actions outside the allowed range are clipped at the boundaries. With this setup, the torque model starts outputting zero torque with the highest probability and explores positive and negative torques equally. In the case of the knee target angle however, a action of zero corresponds to $75°$ which is almost never occurring in the reference trajectories. Moreover, one half of the sampled actions for the knee result in even higher angles. The situation of the other joints is less critical but similar. Summarizing, the initial policy for the torque model performs significantly better with the corresponding action space where the same initialization results in a poor initial performance in the case of target angles. Figure 6.2 (b) supports that claim showing a lower initial imitation reward during the first evaluations.

The **poor imitation of the reference trajectories in the frontal plane of the hip** in Figure 6.4 (second column) are expected as we haven't tuned the PD values for the position controller in these joints but reused the PD gains from the sagittal plane.

Joint torque vs. angle deltas. Specifying joint torques leads to better results across all but one evaluation metric when compared with the action space outputting differences to the current angle: the average velocity reward during training (Figure 6.3 (b)). Angle deltas are equivalent to a desired joint velocity scaled by the control frequency and are calculated based on the maximum velocities observed in the reference data (Equation 5.3). By limiting the maximum allowed angle differences, we therefore also limit the maximum target joint velocity. This is likely the explanation for the drastic reduction of the high oscillations reported in Figure 6.4 (a) compared to the target model without velocity restrictions. In addition, this limitation is expected to allow better tracking of the reference joint velocities.

Velocity can be calculated by integrating the acceleration trajectory. Thus, to obtain smooth joint velocity trajectories, the joint accelerations must not change their sign too often and abruptly. Joint accelerations are directly proportional to the applied joint torque. When we start with an initial policy centered around zero, the torque model has a high probability of consecutive targets being of different signs. This can explain the lower velocity reward at the training's beginning as illustrated in Figure 6.1 (b) that the torque model is not able to make up for during training.

Finally, the limitations of angle differences leads to lower joint torques. While it can be an advantage when aiming for an energy efficient controller, high joint torques are necessary to recover from perturbations. Beside that, higher torques allow to quicker reduce deviations between the simulation and expert trajectories. The torque model has a higher probability to output higher joint torques when necessary. We believe this might be a reason for the higher imitation reward and the quicker convergence of the torque model to stable walking.

6.2. Inducing Information about the Symmetry of Walking

For the remainder of this chapter, we use the torque model as our new baseline. In this section, we compare two methods of incorporating the symmetry of the walking motion into the training procedure. The methods are abbreviated with DUP and PHASE (Section 5.5.3). In DUP, we duplicate the collected samples, mirror the duplicates and treat them as additional experiences. In PHASE, we train the policy only for stepping with one foot. For the other side, the state observations are mirrored, inputted into the policy network and predicted actions are mirrored again before being applied in the environment.

Remaining all hyperparameters fixed was not possible in these experiments. DUP copies collected experiences increasing the doubling the batch size. To retain the batch and the minibatch size constant as well as the number of minibatches, we halve the number of collected experiences. This however, results in two times more parameter updates during training. We compare the results therefore with the original baseline having the same batch size as DUP and a version of it collecting half as much experiences and thus having the same number of parameter updates as DUP. PHASE is also executed with a batch size of 16k.

Figure 6.6 shows the baseline with halved batch size requiring 33% less samples to converge to stable walking. Considering the symmetry of walking, the reduction of samples until walking stably further increases to 53% and 49% for PHASE and DUP respectively. Both of our methods show very similar results with a difference of only 0.1M steps. Remarkable is furthermore the 4 and 2 times reduction in the variance of the results across the random seeds.

The similar performance of symmetry inducing methods and their smaller variance in the results is also reflected in the learning curves in Figure 6.7. The number of stable walks (a) of DUP rises and converges slightly before the PHASE counterpart. Both reach the first stable walks with a small difference but show the same quick increase from the first to all 20 stable walks. This slope is twice less steep for the baseline. The

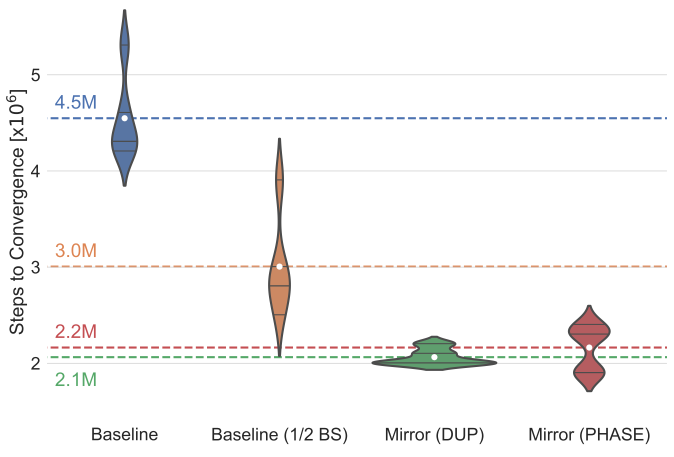


Figure 6.6: **Considering symmetry of walking improves sample efficiency.** The violin plots show the distribution of required agent-environment interactions until the agent achieves stable walking. Interestingly, symmetry inducing methods also drastically reduce the variance in sample efficiency across random seeds. White dots, the dashed lines and the colored labels indicate the mean of the distribution. Lower numbers correspond to better sample efficiency. BS = batch size.

baseline with halved batch size increases the slope significantly and reaches an average of 18 walks quickly but shows a surprisingly high variance from there on, especially indicated by the timesteps difference between the convergence illustrated by the vertical dashed line and the corresponding mean curve reaching 20 walks.

The mean evaluation reward curves (b) show always completely overlapping curves corresponding to the symmetry inducing approaches. Despite that high similarity, PHASE in average converges to stable walking at a 13% higher reward. The highest reward at convergence however is achieved by the baseline with 66%. Interestingly, halving the batch size leave the shape of the reward curve almost unchanged shifting it only to the left by 1M samples.

The summary score (c) most clearly shows the significant effect of the mirroring approaches on the variance reduction across random seeds. The earlier convergence to stable walking of the DUP approach is reflected in a higher score. Finally, the episode returns (d) show an earlier increase for the DUP model but consistently higher values after 4M timesteps for the PHASE approach. Surprisingly, despite having the best results across all other metrics, DUP ends the training at the lowest return. Furthermore, it shows three of the approaches converging at about the same return value around 25%. Table 6.2 summarizes the presented results.

The individual imitation reward components in Figure 6.8 lead to the same rating of all four approaches as presented before. Interestingly, all models converge in average showing a similar position (a) and velocity rewards (b). The position reward thereby is with about 50% 5 times as big as the value of the velocity component. The COM reward in all approaches show a clear downwards trend after a short phase of increase at the beginning.

A comparison of joint kinematic trajectories between simulation and reference is not presented due to insignificant differences between the individual approaches. Distinguishing the approaches based on the rendered walking gaits is also not possible.

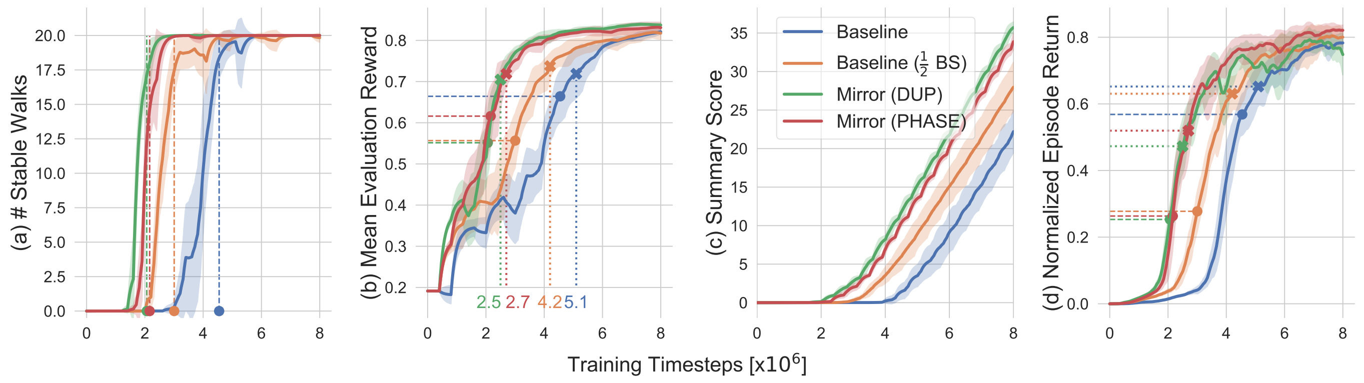


Figure 6.7.: **Better sample efficiency by considering symmetry of walking.** We compare the baseline performance using full and halved batch size (BS) with two methods of inducing the symmetry prior of the learned motion: DUP and PHASE (Sec. 5.5.3). Both methods show similar results and outperform the baselines on all metrics. Halving the batch size positively influence the baseline performance. Circular markers and dashed lines highlight the convergence point. X-markers and dotted lines emphasize the duration until 75% of imitation reward are reached, necessary to ensure visually human-like walking.

Table 6.2.: **Effect of inducing symmetry information on sample efficiency.** Both methods significantly improve the baseline across all but one metric: the reward at convergence. Convergence is defined as learning to walk without falling. 75% of the maximum reward indicate visually human-like walking. BS = batch size.

Approach	Summary Score	Steps to Convergence	Reward at Convergence	Steps to 75% Reward	Final Reward
Baseline	23.4 ± 3.2	4.5 M $\pm$ 0.4 M	$\mathbf{0.66 \pm 0.06}$	5.1 M $\pm$ 0.4 M	0.82 ± 0.07
Baseline (1/2 BS)	28.8 ± 3.4	3.0 M $\pm$ 0.5 M	0.56 ± 0.07	4.2 M $\pm$ 0.6 M	0.82 ± 0.02
Mirror (DUP)	$\mathbf{36.2 \pm 0.5}$	**2.1 M** $\pm$ **0.1 M**	0.55 ± 0.08	**2.5 M** $\pm$ **0.2 M**	$\mathbf{0.84 \pm 0.01}$
Mirror (PHASE)	35.4 ± 1.5	2.2 M $\pm$ 0.2 M	0.62 ± 0.07	2.7 M $\pm$ 0.4 M	0.83 ± 0.01

6.2.1. Result Interpretations

Role of the batch size. Both mirroring approaches collect half as much experiences per policy update compared to the baseline to maintain all the remaining hyperparameters the same. This results in twice as much policy updates during the same training duration. Does this fact explain the superiority of the mirroring approaches? Only partially.

Increasing the number of parameter updates in the baseline model by halving the batch size improves the sample efficiency by 33%. The number of updates therefore definitely has an influence on the learning speed. However, our approaches significantly outperformed the baseline even when it is used with halved batch size. Therefore we are sure, our approaches induce additional useful information into the training process which we discuss in the following.

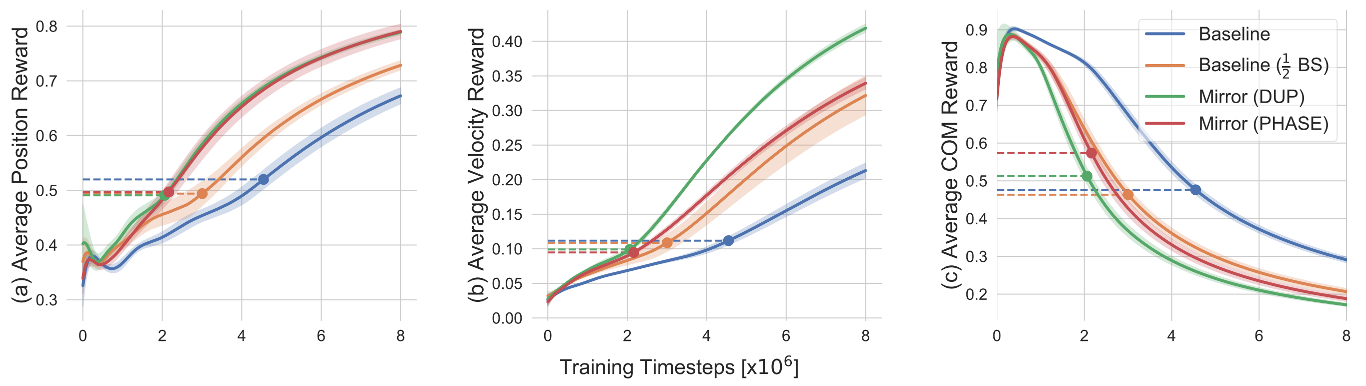


Figure 6.8.: **Individual imitation reward components during training.** Our approaches outperform both baselines in replication of joint kinematics (a,b), showing overlapping position components. DUP stands out reaching the highest velocity values. No approach manages to match the COM trajectories (c). Circles and crosses mark the values when reaching stable walking and 75% of imitation reward respectively.

Before that, we want to closely look at the batch size. Our experiments show the halved batch size being big enough to correctly estimate the policy gradient. This is very likely due to a high distribution of experiences across the whole action space. This scenario likely results from the parallel training on 8 environments. The environments are not synchronized, so each of them follows its own episodes collecting experiences from different regions of the state and action space. Also Reference State Initialization (RSI) is very likely to play a role in evenly exploring the whole state space as reported by Peng et al. (2018a). Finally, using 0.5 as the standard deviation of the initial action distributions, only 4.6% of actions lands outside the allowed normalized range and are clipped to the action space boundaries. This prevents too many actions at the borders being applied on the environment and distributes the actions better across the whole space.

An argument against the halved batch size being big enough is based on the curve of stable walks in Figure 6.7 (b). The baseline with halved batch size increases the slope of this curve significantly indicating a good policy gradient estimation. However, the curve quickly reaches an average of 18 walks but shows a surprisingly high variance from there on and slowly increases to full convergence. This is especially good to see in the timesteps difference between the convergence illustrated by the vertical dashed line and the point at which the corresponding mean curve reaches 20 walks. In addition, the baseline with halved batch size is the only one to diverge from stable walking once it has been reached indicated by the small drops in the curve after 6M timesteps and again on the training's end. These observations might be explained by poor policy gradient estimates due to an insufficient number of considered experiences. The inaccurate estimates then might have led to updates of the policy in the wrong direction explaining the drops in performance.

Reward at Convergence. Despite both symmetry inducing approaches achieving higher imitation rewards in all evaluation runs during the training, the baseline converges to stable walking at a higher imitation reward. One simple explanation for that lies in the monotonically increasing nature of the reward curves (Figure 6.7 (b)). As the baseline converges much later in the training process, it has more time until that point to improve its capability to imitate the expert demonstrations. Analogically, this may also explain the significantly higher convergence reward of the PHASE approach. The reward curves of DUP and PHASE show a very high slope in the interval around 2M timesteps. Even a slightly longer time to convergence of 0.1M timesteps is enough to result in a significantly higher reward.

Another explanation for the highest reward at convergence being achieved by the baseline model is also related to the time to convergence. Our results suggest that mirroring experiences has a bigger positive influence on walking stability than on trajectory imitation. Our approaches seem to favor longer episodes due to more stable walking to achieve higher returns over matching the trajectories more closer. This in turn, can be explained by the way we shape the reward. By punishing falling strongly, the value of a state is dominated by the fact if falling is likely to appear from that state on or not. How likely it is to match the reference trajectories in this and consecutive steps plays only a minor role. Mirrored experiences in both approaches get the same values assigned as their original counterparts. With this, we make the critic aware of the symmetry in the provided task as well as of the fact that falling can appear in opposite directions. This in our opinion explains the quick convergence of the approach as well as the lower imitation rewards at convergence.

Looking at the individual reward components in Figure 6.8, it is interesting to note that the position component at convergence is in average 5 times as high as the velocity part. This supports the expectation of matching joint positions being more relevant for achieving stable walking compared to having similar joint velocity trajectories. On the other side, this is more likely the result of weighting the similarity of joint positions with the expert demonstrations in the reward function significantly higher. Due to different scaling factors α in the individual reward functions and different weightings w_i of the components (Section 5.3.2) combined with different ranges of joint positions and velocities, it is hard to specify the real weighting more precisely.

Reasons for high similarity between DUP and PHASE. Both symmetry inducing approaches perform very similarly across all considered metrics despite their different methodology. DUP mirrors collected samples and treat them as truly made experiences. This can be framed in the context of data augmentation (Shorten and Khoshgoftaar, 2019). PHASE doesn't generate new samples but instead trains the policy to only master a step with one of the legs and mirrors the observations and predicted actions to control the joints when stepping with the other foot. DUP let the state space unchanged but uses twice as more examples for exploration. PHASE let the number of experiences the same but mirrors one site of the action space thus resulting in a twice as small space to explore. Both therefore increase their exploration capabilities to the same degree. The small superiority of the DUP approach is very likely due to the small number of considered random seeds.

6.3. Approaches with Minor to No Sample Efficiency Improvements

Some of our approaches led to only an insignificant improvement in sample efficiency or even worsened the overall performance of the agent. In this section, we evaluate two extensions to the PPO algorithm that achieve minor improvements on sample efficiency: cliprange decay and experience replay. Due to their minor improvements, only a part of the learning curves is presented directly followed by the interpretation of the results. In the last subsection, we discuss the results of approaches which didn't improve the sample efficiency.

6.3.1. Exponential Cliprange Decay Improves Sample Efficiency at the Cost of Human-likeness

In this approach, we schedule the cliprange to exponentially decrease over the course of the training from 0.55 to 0.1 (Section 5.5.6). Cliprange is a hyperparameter PPO uses to limit the change between consecutive policies (Section 4.4). The torque model with the original PPO implementation using a constant cliprange of 0.15 during the whole training duration serves as the baseline in this comparison.

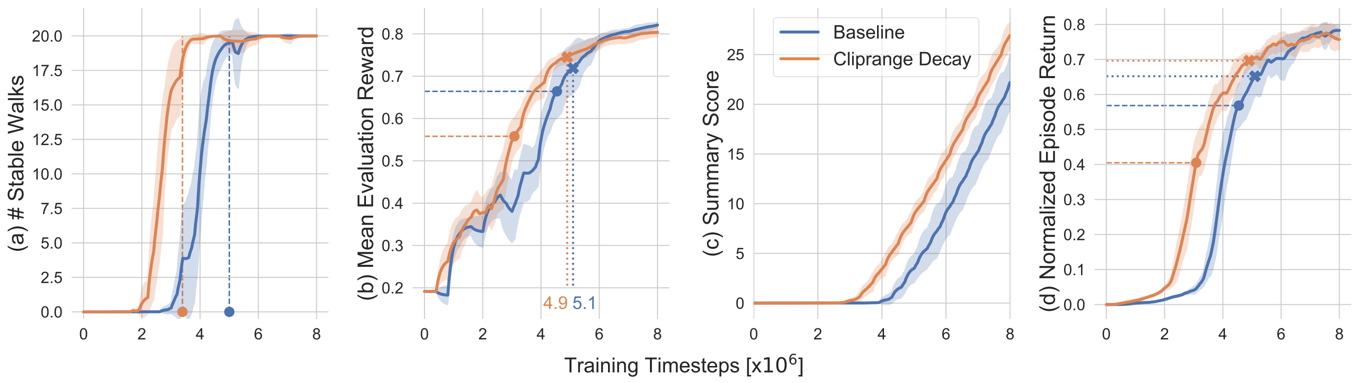


Figure 6.9.: **Exponential Cliprange Decay Favors Stability over Human-likeness.** We investigate the effect of starting with a high cliprange and exponentially decaying it over the course of the training. This approach reduces the time to convergence (a) reaching however a smaller imitation reward at that time (b). Circular markers and dashed lines highlight the convergence point. X-markers and dotted lines emphasize the duration until 75% of imitation reward are reached.

Figure 6.9 shows minor improvements of the scheduling compared to the baseline across all metrics with a few exceptions. Our approach converges 27% earlier to stable walking compared to the baseline (a). In (b) however, the baseline reaches a 15% higher imitation reward at convergence and needs about the same amount of samples to reach the 75% mark as when using the decay. At the training's end, the baseline even surpasses the scheduling showing a reward of 0.82 compared to 0.8. The summary score of our approach is 18% bigger for our approach (c) and the episode return curves reflect its earlier convergence and the advantage of the baseline at the training's end (d).

Result Interpretation. The improvement in sample efficiency confirms our expectations. The policy at training's beginning is initialized with random weights and is therefore likely to be far away from a policy able to achieve stable walking. In the baseline, the small cliprange of 0.15 limits the change between consecutive policies resulting in more required policy updates to arrive at a desired behavior. The idea of limiting the difference in action distributions was introduced to avoid big changes between consecutive policies that might harm its performance (Schulman et al., 2015). At the training's beginning however even big updates to the random policy are unlikely to decrease its already poor performance. Allowing bigger changes to the random policy is therefore likely to speedup the learning process.

Another observation in Figure 6.9 (b) suggests that cliprange decay favors the stability of walking over its human-likeness. It converges to stable walking 27% earlier but does it at a 17% lower imitation reward. The higher cliprange at the beginning of training allows bigger steps from the random initial policy in the direction of the policy gradient. The policy gradient indicates the direction of the highest achievable performance improvement. These results therefore confirm the higher importance of walking stability compared to the human-likeness of the learned movement due to strongly punishing falling in our reward formulation.

A final remark concerns the Figures 6.9 (b) and (d). In both, our scheduling approach shows higher values for the first 75% of the training. Thereafter however, the cliprange decay shows a flatter slope and is surpassed by the baseline curve. The reason for that is most probably the lower cliprange at the end of the training. While the baseline uses a constant cliprange of 0.15, our schedule converges to the smaller cliprange of 0.1. This limits the change between consecutive policies and slows down the progress in performance.

6.3.2. Experience Replay Achieves Same Sample Efficiency but Improves Human-Likeness

Our final approach recalculates the state values of experiences collected with the previous policy and uses them together with newly collected experiences to update the current policy (Section 5.5.5). The torque model with the original PPO implementation and halved batch size serves as the baseline in this comparison.

Replaying experiences does not have any effect on the number of required samples until stable walking is reached for the first time (Figure 6.10(a)). At the time of convergence however, our approach replicates the expert trajectories 18% closer compared to the baseline (b). The same subfigure further shows experience replay to converge 19% earlier to 75% of the imitation reward. Interestingly, these improvements are reached despite the lower rewards for the first 1.5 million timesteps. The summary score (c) and the episode return (d) also rate the approach replaying experiences at any time higher compared to the baseline.

Results Interpretation. Reusing experiences from previous policy rollouts positively influence the sample efficiency in off-policy algorithms (Lillicrap et al., 2015; Fujimoto et al., 2018). PPO is however an on-policy algorithm and requires the experiences to be all gathered by following the current policy to correctly estimate its gradient. Due to the use of a small cliprange of 0.15 however, PPO limits the distance between consecutive policies and with it the change in state and action distributions collected by the current and the previous policy. This way, experiences collected earlier still contain useful information about the direction of better performance in parameter space and can positively influence the learning process. By recalculating the values of previous experiences under the current policy, we further increase this information.

Our ablation studies support these claims. The more previous rollouts were stored in the replay buffer, the more the performance across all the metrics decreased. The best results were achieved by considering only experiences obtained by following the previous policy. Moreover, training the same agent without value recalculation showed a minor but consistent decrease in the summary score across random seeds.

The off-policy nature of experiences from previous policy rollouts is likely the reason for the absence of improvement regarding the required samples to reach stable walking for the first time (Figure 6.10 (a)). When looking at the times the agents achieve 75% of the reward though, experience replay shows a 19% improvement and generally higher rewards and returns. We believe, the additional experiences have a regularizing effect improving the generalization to unseen states and explaining the improvements in the imitation of expert trajectories.

A last remark concerns the lower reward values for the first 1.5M timesteps when using experience replay (Figure 6.10). The reason for that is probably the random policy at the beginning of the training. First, the random behavior of the initial policy is likely to cause bigger changes during policy updates and therefore weaken the assumption that previously collected experiences are still close to being on-policy. Second, the poor performance of the policy results in collecting mostly states of lower values. The high imbalance between states of high and low values is expected to increase the chance of incorrect policy gradient estimation. By reusing experience from previous rollouts, this balance is further increased. After the first policy updates after about 1M timesteps, the policy is able to better replicate the motion capture trajectories reducing the imbalance between good and bad states.

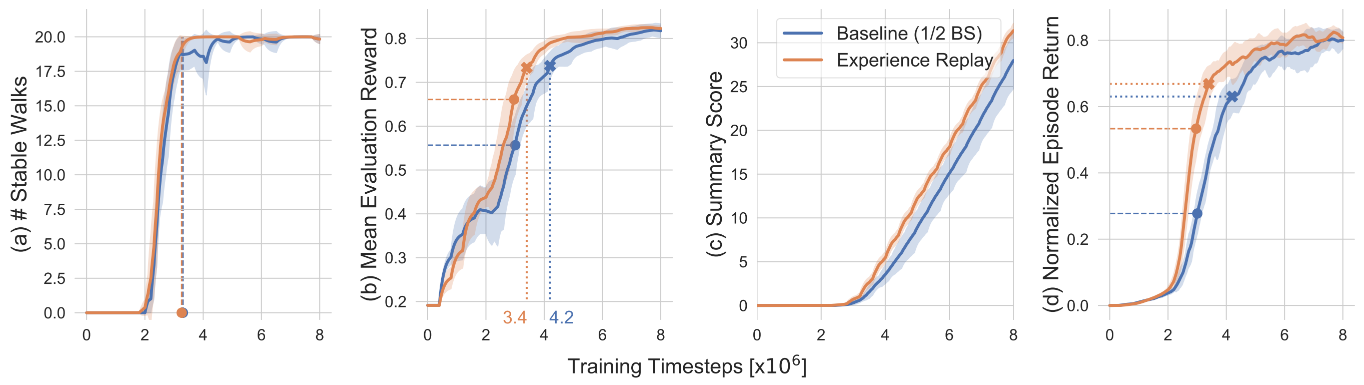


Figure 6.10.: **Experience Replay Quicker Increases the Human-Likeness.** Replaying experiences from a previous batch together with freshly gathered trajectories converges to stable walking after the same amount of samples (a, lines overlap) but reaches a higher reward at that time (b). In addition it reduces the time to reach visually human-like walking indicated by 75% of the maximum imitation reward (b). Circular markers and dashed lines highlight the convergence point. X-markers and dotted lines emphasize the duration until 75% of imitation reward are reached. BS = batch size.

6.3.3. Why Did Other Approaches Not Improve the Sample Efficiency?

Adding ground contact information to the agents observations (Section 5.5.4) showed no significant changes in the agents behavior or the learning curves. Using the motion capturing data to pretrain the policy network (Section 5.5.1) even worsened the results. Following, we discuss the reasons for these outcomes.

Ground Contact Information. Adding the ground contact information was expected to improve the training performance and sample efficiency due to useful additional information about the agent's state. The information was expected to be useful as ground contact helps to distinguish the different phases of the walking gait (Section 3.1.3). However, the binary nature of this signal is likely to be only of an insignificant help for the agent. In addition, the agent can learn to obtain this information from the other signals in the state observations if necessary and even extract additional information like the duration of the stance phase.

Behavior Cloning to Warm-Start the Policy. Starting our experiments with the walker model outputting target angles for PD position controllers, the motion capturing data containing joint angle trajectories of a human walking yielded the possibility to extract state-action pairs coming from an expert policy. We did not expect the approach to result in a policy that will be able to generate stable walking directly after supervised training. The violation of the i.i.d. assumption in our dataset as well as the limited coverage of the state space within the expert demonstrations were expected to cause difficulties (Ross et al., 2011). However, using the data to pretrain the policy in a supervised manner and use it to initialize the policy of our PPO agent, had the chance to warm-start the RL training and improve the sample efficiency above using random weight initialization (Kober and Peters, 2010; Zhu et al., 2018).

Nevertheless, our experiments showed a significant decrease in performance and sample efficiency when using the pretrained model for initialization. We see two strong reasons for that. First, generating state-action pairs from the motion capturing data makes multiple implicit assumptions that are likely to be unfulfilled. For the data to result in a useful policy, it should be possible to closely replicate the reference trajectories by the walker model and replicating the trajectories should results in stable walking. With a maximum imitation reward of 83% (Figure 6.7 (b)) this assumption is only partially fulfilled. Figure 6.4 shows in addition that stable walking was achieved by generating trajectories different from the reference trajectories by all considered agents. Finally, the PD controllers should be able to closely follow any of the learned trajectories which cannot be guaranteed given the control parameters were tuned for the swing phase (Section 5.2.3).

The second and more crucial reason for the ineffectiveness of behavior cloning is the strong distribution mismatch between the states generated from the expert trajectories and the states observed in simulation. Especially the distribution of joint velocities were very prominent. Despite being initialized in a state taken from the reference trajectories, after just a few training steps the observed states were no longer within the state distribution the network was trained on. The high difference in the joint velocity distribution most probably comes from the very different actuator systems used by expert (muscles, multiple for each joint, some biarticular) and the robot (motors, one per joint, uniarticular) as well as different joint properties (damping and friction).

Finally, this result underlines the high effectiveness of the policy initialization we use in the baseline PPO implementation (Section 5.3.4), which has been reported to significantly improve training speed and performance in on-policy methods (Reda et al., 2020; Andrychowicz et al., 2020).

6.4. Results Summary and Practical Recommendations

Aiming at making the field of deep reinforcement learning easier to enter for roboticists focusing on bipedal locomotion, we sum up the results of our experiments and provide practical recommendations.

Our experiments confirm the suitability of the DeepMimic approach (Peng et al., 2018a) for learning human-like and stable walking in simulation. When possible, joint torques should be used as actions achieving 53% higher sample efficiency compared to outputting target angles proposed by the original approach. If the robot at hand is equipped with PD position controllers, the policy should be trained to output angle deltas, small angle differences that are added to the current joint positions to determine the target angles. This approach generates significantly smoother trajectories and achieves a 17% closer replication of the human walking gait. Results of the action space comparison are presented in Section 6.1. Details on implementation can be found in Section 5.5.2.

The proximal policy optimization (PPO) algorithm (Schulman et al., 2017) has proven its effectiveness in the context of the DeepMimic framework and is therefore recommended. It generalizes well to all considered action spaces and shows better wall-clock times compared to more recent algorithms. Multiple high quality open-source implementations of the algorithm are available online, e.g. from stable-baselines (Hill et al., 2018). To improve the data efficiency of PPO, we recommend starting with a high cliprange and exponentially decay it through the course of the training. Please, see Section 5.5.6 for details on implementations and Section 6.3.1 for the results.

Finally, the sample efficiency can be further increased by mirroring the experiences (Section 6.2) or reuse the samples collected during the previous policy rollout (Section 6.3.2). Combining the mirroring approach with the torque action space, the best overall improvement in sample efficiency of 78% upon the original DeepMimic implementation is achieved. For inducing the symmetry information in the learning process, we recommend using the DUP approach above PHASE (Section 5.5.3). DUP achieves a higher sample efficiency gain, don't require making changes to the RL environment and is expected to achieve higher robustness to perturbations compared to PHASE.

7. Discussion

7.1. Importance of Deterministic Evaluation

To ensure exploration during training, it is common to induce stochasticity in the policy's action predictions. Common learning curves like the episode return or episode duration evaluate the performance of the stochastic policy. In case of training the agent on parallel environments, it is in addition common to average the performance over the environments. In summary, conventional learning curves show an average performance of the stochastic policy. This might be acceptable, when different algorithms are compared. In case of training a policy with the goal to apply it on a real robot later, we're however interested in the performance of the deterministic counterpart of this policy. Using a Gaussian policy during training, the policy predicts the mean of a distribution that is then used to sample an action from. In the deterministic counterpart, the mean predicted by the policy is directly applied to the environment.

Our experiments have shown it is not reliably possible to predict the performance of the deterministic policy based on the learning curves. An example is illustrated in Figure 6.7. The first three graphs (a,b,c) evaluate the performance of the deterministic model and uniformly rate DUP as the best approach. Looking at the return (d) calculated based on the stochastic policy, DUP shows remarkably lower values compared to PHASE starting from 4M timesteps and even the worst return at the end of the training.

7.2. Comparison of Sample Efficiency Evaluation Metrics

A common way to estimate the sample efficiency of an algorithm is to measure the amount of data required to reach a specified performance threshold. Aiming at learning visually human-like locomotion, the performance metric has to consider the walking stability as well as the similarity of the learned motion to reference trajectories. While the return implicitly considers both of these metrics, it fails at correctly predicting the most sample efficient algorithm as discussed in Section 7.1. Another reason disqualifying the return can be seen in Figures 6.2 (d) and 6.7 (d). The markers highlight the return at two different points during training: convergence to stable walking and reaching an average imitation reward of 75%. We can clearly see that stable as well as human-like walking correspond to different returns. This makes it impossible to specify a fair threshold for sample efficiency evaluation.

What other learning curves can we use? To consider the timestep an agent reaches stable walking for the first time, we can obviously look at the number of stable walks (Figures 6.2 (a) and 6.7 (a)). This graph however doesn't include any information about the human-likeness of the learned walking motion. This information is available in the graph showing the average reward during the 20 episode evaluations (Figures 6.2 (b) and 6.7 (b)). It can also be used to get the number of training timesteps to reach 75% or the imitation reward. This graph although doesn't contain any information about the walking stability. The before mentioned figures show in addition that there is no unique threshold to detect stable walking. Finally the reward curves of two approaches strongly overlap in Figure 6.7 (b) even one of the methods is more sample efficient.

It is clear that we have to consider both graphs to get a sound evaluation of the sample efficiency. But how to handle contradictions, when one graph favors one approach and the second graph another. Achieving stable walking with a reward of 30% is likely to result in an idiosyncratic walking gait. On the contrary, reaching 75% reward but falling once out of 20 evaluation episodes is also an unsatisfying result. Finally, how to handle divergence from stable walking? Our summary score solves these issues by multiplying both graphs and calculating the area under the resulting curve. By exponentiating both curves with different exponents, we can specify the importance of both graphs individually. Area under the learning curve has been previously suggested as a measure for the sample efficiency (Peng and van de Panne, 2017). In all our results, the summary score rate the presented approaches in agreement with all other metrics despite overlapping curves. Finally, our summary score considers the whole course of the curves during training instead of just reaching individual points. It therefore punishes drops in performance and is much more robust to noise compared to a simple threshold. We recommend the use of the summary score for sample efficiency estimation in future works.

In case a simpler metric is preferred, still considering the walking stability and human-likeness, our experiments propose another metric: the number of samples until stable walking is achieved while surpassing a reward threshold. We use the threshold of 50% and 75%. Both metrics always agree concerning the rating of approaches with the summary score as can be seen in Tables 6.1 and 6.2.

7.3. Why Not Using TD3, SAC & Co.?

PPO (Schulman et al., 2017) is a popular algorithm used in most of scenarios applying deep RL to solving bipedal locomotion in simulation (Peng et al., 2018a; Lee et al., 2019; Yang et al., 2020). It is however an on-policy algorithm and in addition model-free. Off-policy and model-based algorithms are known to be more sample efficient. Why haven't they been used in our work then?

Why not Model-based approaches? Model-based approaches are far more sample-efficient when having access to a precise model (Achiam, 2018). Alternatively, a model can be learned by the RL algorithm. A model allows the agent to plan its actions ahead, predict the outcomes of possible actions and choose the best without interacting with the environment (Camacho and Alba, 2013), explaining its high potential to improve sample efficiency. Errors in the model however accumulate the further in the future the model predicts and make it hard to reliably learn from the model predictions.

A precise model in addition is very hard to come up with, especially in the context of bipedal walking robots. Moreover, to be able to use a precise model, high quality sensors are necessary to correctly determine the current state of the environment (Zhao et al., 2020). Polydoros and Nalpantidis (2017) review the application of model-based RL in the field of robotics. None of the presented robots show nearly the complexity of a humanoid robot. Building precise models is in addition time consuming and requires high domain expertise (Xie et al., 2018). Contact dynamics of the feet with the ground are especially hard to model precisely. Finally, most common RL approaches in literature are model-free and therefore high quality implementation of algorithms are seldom available for model-based approaches.

Why not using off-policy algorithms? Off-policy algorithms like TD3 (Fujimoto et al., 2018) and SAC (Haarnoja et al., 2018b) show better performance and sample efficiency on benchmark environments compared to PPO (Achiam, 2018). However, they reach better sample efficiency by building up a replay buffer of experiences and training Q-functions with Bellman-Error-Updates (Achiam, 2018). While they drastically reduce the number of required samples to converge to a local optimum, the wall-clock time of these algorithms is significantly higher due to replaying a high portion of past experiences after collecting a batch of new experiences.

Our goal is to lower the entry barrier for roboticists working on bipedal walking to the promising field of deep reinforcement learning. One approach to this goal is to make deep RL much more sample-efficient so it can be applied to real robots directly (Haarnoja et al., 2018a; Lee et al., 2020). We follow the approach of Peng et al. (2020) and propose to pretrain an agent in simulation with a fast on-policy algorithm and transfer the controller to the real robot using sample-efficient domain adaptation techniques (Zhao et al., 2020). We believe this approach to be more promising and easier to enter, as it allows to try deep RL without risk of breaking hardware, obtain first results more quickly and do more iterations in the same time.

7.4. Comparison of Our Approaches with Related Work

As the results of our approaches were already interpreted directly after result presentation, here we discuss our results in the context of related literature.

7.4.1. Comparison with the Original DeepMimic Approach

Before starting our comparison it is important to note an important difference. DeepMimic was developed focusing on maximizing its generalization across different environments and tasks and achieved that goal (Peng et al., 2018a). Within this work, we use DeepMimic for a single specific task in a single specific environment. Our hyperparameters and algorithmic design choices are optimized for achieving stable human-like walking and will probably not generalize to strongly different tasks morphologies.

Different early termination (ET) conditions Peng et al. (2018a) terminate the episode after detecting contact of the characters head or torso with the ground. We in contrast stop the episode far earlier after the trunk exceeds an angle thresholds in one of the 3 possible directions or the body's COM falls below a certain threshold in the z direction still being far away from touching the ground with the torso or the head. This way, we limit the observable state space to the region that is most relevant for the task. This might harm the robustness of walking against perturbation as the agent doesn't encounter states far outside the distribution produced by stable walking and has no chance to learn to recover. On the other side, the limited state space increases the chance of quicker exploration and faster convergence. In case of unsatisfying robustness, one could follow a curriculum learning approach, starting by limiting the state space drastically at the training's beginning and softening the constraints as the agent improves. A similar procedure has been shown to improve sample efficiency by about two times (Yu et al., 2018).

Different discounting factor. The discounting factor γ plays a crucial role to achieve stable walking and avoid divergence from this point on. Peng et al. (2018a) propose a value of $\gamma = 0.99$. Given their control frequency of 30Hz this corresponds to considering the rewards of the next 3.3 seconds for the return calculation (Wright, 2019). During tuning the hyperparamters we achieved the best results using a discount factor of $\gamma = 0.995$. At our control frequency of 200Hz, this factor considers 200 timesteps equivalent to 1 second of simulated time. Interestingly, this approximately corresponds to the duration of a whole gait cycle consisting of two steps. Yang et al. (2020) train a bipedal controller for the Valkyrian humanoid robot and choose a discounting factor of 0.95 considering 0.5 seconds of future rewards. This result is closer to ours then to the original DeepMimic implementation.

High reliance on the phase variable. We want to close the comparison of both implementations with a common drawback and present a possible explanation. As in the original framework, our training still relies on including the phase variable as a part of the state observations (Section 5.3.1). The phase variable is a linearly increasing variable indicating the temporal position in the current gait or step cycle. Its effect can be interpreted as inducing a time awareness to the learning process. Being aware of the time, the agent is more likely to learn the simpler feed-forward instead of a feedback controller (Pardo et al., 2018). We believe this is at least partially what happens in our case and provide a possible explanation for why the phase variable is so crucial.

7.4.2. Torque vs. Target Angles

In contrast to related work favoring target angles above joint torques (Peng and van de Panne, 2017; Peng et al., 2018a; Reda et al., 2020), we achieve better results outputting joint torques compared to target joint angles. In the following, we discuss a few possible explanations for that.

The obvious possible reason might be grounded in sub-optimally tuned PD gains as already discussed in detail in Section 6.1.1. But even if our parameters were optimal, the other authors (Peng et al., 2017, 2018a) applied the DeepMimic approach in context of physic-based character animation where instead of the traditional PD controllers used in our model, a stable PD (SPD) controller (Jie Tan et al., 2011) was utilized. This controller allows for arbitrarily high gains in imitation tasks by using the reference joint kinematics of the character at the next timestep. Reference trajectories however will not be available when the controller is applied to a real robot which is why we used a traditional PD controller. Therefore, the superiority of the target angles as action space in the others works might be caused by the use of an SPD.

Next, it is important to consider that Peng and van de Panne (2017) compared the action spaces on characters limited in their motion to the sagittal plane. An additional third dimension besides increasing the state and action spaces makes the balancing much harder allowing falling in each of the 360° instead of just forward, backwards and down as in the 2D case. With these significant changes in the task complexity also the rating of individual action spaces might have changed.

Another reason for the torque model performance might be the high joint damping of our model. As MuJoCo only allows to specify P-value for a position servo in the joint, the D-value was specified as recommended indirectly by changing the damping of the corresponding joints. As we switched to the torque model, the position servos were exchanged by a simple motor but the damping in the joints remained at their values tuned for the PD position control. Damping has been reported to have a positive effect on stability (Kawamura et al., 1988) and is generally known for its smoothing effect. We therefore believe, these higher damping values to be a possible reason for the smooth joint trajectories and the sample efficiency of this approach.

Finally, Peng and van de Panne (2017) do not report to have normalized the action spaces in their comparison. Their Figure 11 in the appendix displaying "policy actions", show an a 100 times bigger range for the torque model compared to the policy outputting target angles. We assume they accounted for the higher ranges by changing the covariance matrix of the Gaussian distribution Σ which they chose as a diagonal matrix with constant manually specified parameters $\Sigma = \mathrm{diag}(\sigma_i^2)$. The specified parameters σ_i to our knowledge are not reported. It is therefore possible that the torque model had a disadvantage in the comparison and therefore performed worse.

In our experiments, we normalize the torque actions to the range of [-1,1] and do the same for the model outputting target angles. However, even this procedure allow us to remain all the hyperparameters fixed indicating a fair comparison, the different action ranges lead to differently performing initial policies as described in Section 6.1.1. With that, the model specifying target angles starts with a disadvantage, too.

7.4.3. Angle Deltas vs. Target Velocities

Outputting angle deltas can be interpreted as outputting scaled joint velocities. Peng and van de Panne (2017) compared different action spaces for learning locomotion tasks with deep RL and next to policies outputting desired joint angles tested outputting target joint velocities. In their experiments the former was shown to be superior. Even though our results seem to stand in contradiction, there is an important difference between our and the author's approach to angle velocities.

It is correct, that angle differences can be interpreted as desired joint velocities. However, instead of treating them as velocities, we add the deltas to the current joint positions and use the sum as target angles for the PD position controllers. Peng and van de Panne (2017) however, output joint velocities and use these as targets for a proportional (P) velocity controller in each joint. So, there are two fundamental differences:

1. PD-controlled joint positions vs. P-controlled joint velocities
2. strongly limited maximum joint velocities specified by maximum deltas in our case vs. unlimited target joint velocities

Using a P controller for following the target velocity trajectories, the damping effect of the D-part compared to the PD-controlled positions is missing. Damping is however known for its stabilizing properties. By choosing an appropriately high D-gain, a PD controller can guarantee a critically damped behavior avoiding oscillations when transitioning from one position to another and improving robustness against perturbations (Kawamura et al., 1988). The damping component in the control might therefore play a crucial role in learning to replicate the smooth joint angle trajectories from the mocap data.

Strongly limiting the maximum joint velocities reduces the maximum possible oscillation range. In addition, it limits the action space to the region of interest, making it easier to find optimal actions. This in turn increases the chance of smooth velocity trajectories and explains the better velocity reward of our approach illustrated in Figure 6.3 (b).

7.4.4. Effect of Symmetry on Sample Efficiency

Other works comparing different methods to induce symmetry into learning bipedal locomotion do not report a positive effect on sample efficiency in contrast to our experiments. Yu et al. (2018) augment the loss function by a specific term punishing asymmetric action predictions. While the authors report an approximately two times higher sample efficiency using the loss function, it is only achieved in combination with a specific curriculum. Inducing symmetry information within the loss function does not affect the sample efficiency as can be seen in their Figure 8. Abdolhosseini et al. (2019) compare this loss function with DUP, PHASE, and a symmetric policy network design and too report no consistent impact on learning speed against their expectations.

Both authors also used the PPO algorithm for their investigations. The presented results however were observed training bipedal walking in significantly different scenarios which are expected to cause differences. First of all, the authors use different models of the walker and do not consider reference trajectories to shape the reward. Instead, a simplistic reward function is used encouraging forward locomotion. The reward function however is known to be crucial. By using the reference trajectories, our reward function implicitly encourages symmetry given the symmetric expert demonstrations.

When mirroring experiences as made in DUP, additional information is necessary to use the generated samples for the policy update in PPO. We need to specify the return, the state values and the probability of actions. Yu et al. (2018) do not report the details of their implementation. In our experiments, we simply take the required information over from the original experiences.

Given an implicitly symmetric reward function, the reward of mirrored actions a'_t in a mirrored states s'_t should be the same $r(s, a) = r(s', a')$. With this, the true value of mirrored states should be the same too: $V^*(s_t) = V^*(s'_t)$ assuming the optimal policy $\pi^*(a_t|s_t)$ is outputting mirrored actions in a mirrored state s'_t: $\pi^*(a_t|s_t) = \pi^*(a'_t|s'_t)$. Given all that, the expected return of mirrored state-action pairs should be the same too. Even though we cannot guarantee all these assumptions to hold exactly, they support our choice of approximating the missing values for mirrored experiences.

In addition, this argumentation should positively influence one drawback of using DUP with PPO: the generated experiences are strictly not on-policy (Abdolhosseini et al., 2019). Moreover, by initializing the output layer of the policy with values close to zero and using the same initial standard deviation for all joints, the initial distribution of mirrored joint actions should be very similar. With the symmetric reward function and assuming the observed states being equally distributed over both legs due to random state initialization, the action distributions of same joints on different legs should change in a similar direction resulting in a symmetric policy.

7.5. Limitations and Outlook

We achieved significant improvements in sample efficiency using a simplified walker environment. This was only the first small step to make deep RL more attractive and practical for roboticists working on bipedal walking and there is still a far way to go.

Generalization to more complex realistic models and sim2real methods. The transfer of policies trained in simulated environments to real robots requires extending the training procedure. Environment randomization (Peng et al., 2018b) or learning a latent dynamics embedding are examples from the literature. (Peng et al., 2020). Our approaches need to show generalization to these scenarios requiring few to no adjustments to be useful in practice.

Training a controller for a real robot will require further optimization like energy efficiency or avoidance of high impacts through interactions with the environment. If the robot uses cameras to sense its environment it will be important to walk without shaking the camera and guarantee sharp high quality images. We believe our methods will generalize to these scenarios expecting them to require only changing the reward function.

Realistic simulation models will probably reveal new challenges. To focus on SE, joint peak torques were chosen much higher as what realistically can be achieved on a real hardware system without using high gear ratios which will increase the inertia and decrease the maximum joint angular velocities. Furthermore, torque control might require the policy to operate at higher frequencies. Short ablation studies has shown generalization to realistically chosen joint torque ranges but revealed a drastic reduction in the learning speed and performance when querying the policy at higher rates. Finally, perfect sensors were assumed correctly measuring the current state at 200Hz without any noise.

To guarantee a fair comparison of individual approaches, we tried to maintain all hyperparameters fixed across our methods. Mirroring experiences and replaying samples from previous policy rollouts however made this impossible. Even we adapted the affected hyperparameters in our baseline to make the models better comparable, optimizing the hyperparameters of individual approaches might further improve the fairness of our experiments at the cost of high computational effort.

High importance of the phase variable. Adding a phase variable as the part of the state observations (Section 5.3.1) has been shown to be crucially important for quick convergence to stable walking. While this information is available in simulation, it will be not on a real robot. When focusing on straight unperturbed walking on even ground, the motion is periodic and the phase variable can be easily estimated. When however targeting at more complex motions and realistic scenarios, the phase variable might no longer be determined and the agent would behave unexpectedly. While most of the related work also rely on a phase variable (Peng et al., 2018a; Lee et al., 2019; Yang et al., 2020), Anand et al. (2019) achieved stable walking using a torque and muscle model without the use of the phase variable shading light on possible directions for future works.

Outlook. The shape of the learning curves displaying the episode return or the number of stable walks (Figures 6.7 (a) and (d)) show a step like curvature. They start with a small value and after some training quickly increase close to convergence. We believe this is due to the random initialization of the network weights. By comparing the networks at training's beginning and the end a future work might reveal a better initialization of network parameters that will quicker converge to stable walking.

Hessel et al. (2019) reports strong inductive biases to benefit learning speed and performance. In the context of bipedal walking, insights from the field of biomechanics on how to measure walking stability, detect falling early or enrich the state observations with useful information promise further improvements (Zhang et al., 2019; Yang et al., 2020; Song et al., 2020). In addition, the inclusion of muscles models in the simulation may result in more human-like walking and restrict the solution space for learning based methods (Anand et al., 2019; Lee et al., 2019).

Transfer Learning (Taylor and Stone, 2009; Zhu et al., 2020) and Meta-Learning (Bengio et al., 1990; Wang et al., 2016) might significantly improve the performance at training's beginning and are in general a promising future direction for sample efficient learning with deep RL. The usage of graph neural networks (Sanchez-Gonzalez et al., 2018; Wang et al., 2018) or action branching architectures (Tavakoli et al., 2017) are expected to further increase the success of these and other approaches. Using ideas from curriculum learning (Yu et al., 2018) is another promising area to investigate.

8. Conclusion

Aiming at making deep reinforcement learning more attractive and convenient to apply to bipedal locomotion in humanoid robots, this work presents a sample-efficient learning-based controller for stable and human-like bipedal walking. We adapt the state-of-the-art approach for learning visually human-like motions *DeepMimic* to maximize its effectiveness for the targeted motion. High quality motion capturing data are used to provide a dense reward signal and guide the agent's behavior towards human-like walking. Following an environment-agnostic approach, we extend the popular *walker2d* model from OpenAI Gym to the third dimension and adapt its morphology and inertial properties to match that of the subject the reference trajectories were collected from. MuJoCo is used for physics simulation.

Our first experiments reveal the need of a reliable metric to measure the sample efficiency of an approach while considering the stability and human-likeness of the learned walking gait. We meet this request by developing the *Summary Score* metric, which multiplicatively combines the scaled areas under two important learning curves: the number of evaluation episodes showing stable walking behavior as well as the mean imitation reward across the evaluation runs.

Having a reliable measurement of sample efficiency, we investigate multiple approaches to further reduce the number of required samples to reach stable human-like walking. Our comparison of different action spaces reveals joint torques being 53% more sample efficient in contrast to specifying target angles for PD position controllers. For robotic systems relying on PD servos, we propose outputting normalized angle differences that besides improving the sample-efficiency by 41% further increase the replication quality of the motion captured gait by 17% and generate significantly smoother joint position and velocity trajectories.

Another major step in sample efficiency improvement is achieved by leveraging the symmetry of the walking gait. Combined with joint torques as policy outputs the number of required samples to achieve stable walking is reduced by 78% compared to our original implementation of the DeepMimic approach.

At the end of our work, we're able to learn a controller generating human-like stable walking in a lower-body humanoid simulation with a state space of 29 dimensions and an 8-dimensional continuous action space in approximately two hours using 8 parallel CPU cores of the AMD Ryzen Threadripper 2990WX processor to collect experiences and an Nvidia GeForce GTX 1080 Ti GPU for updating the network weights.

Future works should focus on generalization of our sample efficiency improvements to more complex and realistic simulations of humanoids as well as domain adaptation techniques required to transfer the policy learned in simulation on the real hardware.

9. Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Twenty-first international conference on Machine learning - ICML '04*, page 1, Banff, Alberta, Canada, 2004. ACM Press. doi: 10.1145/1015330.1015430. URL `http://portal.acm.org/citation.cfm?doid=1015330.1015430`.

Farzad Abdolhosseini, Hung Yu Ling, Zhaoming Xie, Xue Bin Peng, and Michiel van de Panne. On Learning Symmetric Locomotion. In *Motion, Interaction and Games*, pages 1–10, Newcastle upon Tyne United Kingdom, October 2019. ACM. ISBN 978-1-4503-6994-7. doi: 10.1145/3359566.3360070. URL `https://dl.acm.org/doi/10.1145/3359566.3360070`.

Joshua Achiam. Spinning Up in Deep Reinforcement Learning. 2018.

Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

Aliakbar Alamdari and Venkat N Krovi. A review of computational musculoskeletal analysis of human lower extremities. In *Human Modelling for Bio-Inspired Robotics*, pages 37–73. Elsevier, 2017.

Akhil S Anand, Guoping Zhao, Hubert Roth, and Andre Seyfarth. A deep reinforcement learning based approach towards generating human walking behavior with a neuromuscular model. In *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*, pages 537–543, October 2019. doi: 10.1109/Humanoids43949.2019.9035034. ISSN: 2164-0580.

Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphael Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study. *arXiv:2006.05990 [cs, stat]*, June 2020. URL `http://arxiv.org/abs/2006.05990`. arXiv: 2006.05990.

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

Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A Brief Survey of Deep Reinforcement Learning. *IEEE Signal Processing Magazine*, 34(6):26–38, November 2017. ISSN 1053-5888. doi: 10.1109/MSP.2017.2743240. URL `http://arxiv.org/abs/1708.05866`. arXiv: 1708.05866.

Kavosh Asadi, Dipendra Misra, Seungchan Kim, and Michael L. Littman. Combating the compounding-error problem with a multi-step model. *CoRR*, abs/1905.13320, 2019. URL `http://arxiv.org/abs/1905.13320`.

Karl J Åström and Björn Wittenmark. *Adaptive control*. Courier Corporation, 2013.

Christopher G. Atkeson and Stefan Schaal. *Robot Learning From Demonstration*. 1997.

Millard F Beatty. *Principles of Engineering Mechanics: Volume 2 Dynamics–The Analysis of Motion*, volume 33. Springer Science & Business Media, 2005.

Yoshua Bengio, Samy Bengio, and Jocelyn Cloutier. *Learning a synaptic learning rule*. Citeseer, 1990.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv:1606.01540 [cs]*, June 2016. URL `http://arxiv.org/abs/1606.01540`. arXiv: 1606.01540.

Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer Science & Business Media, 2013.

Alfredo V Clemente, Humberto N Castejón, and Arjun Chandra. Efficient parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1705.04862*, 2017.

Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking Deep Reinforcement Learning for Continuous Control. *arXiv:1604.06778 [cs]*, May 2016. URL `http://arxiv.org/abs/1604.06778`. arXiv: 1604.06778.

Tom Erez, Yuval Tassa, and Emanuel Todorov. Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 4397–4404. IEEE, 2015.

Peter I Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

Scott Fujimoto, Herke Van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. Learning to walk via deep reinforcement learning. *arXiv preprint arXiv:1812.11103*, 2018a.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *arXiv:1801.01290 [cs, stat]*, August 2018b. URL `http://arxiv.org/abs/1801.01290`. arXiv: 1801.01290.

Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. Learning to Walk via Deep Reinforcement Learning. *arXiv:1812.11103 [cs, stat]*, June 2019. URL `http://arxiv.org/abs/1812.11103`. arXiv: 1812.11103.

Hado V Hasselt. Double q-learning. In *Advances in neural information processing systems*, pages 2613–2621, 2010.

Herbert Hatze. The meaning of the term" biomechanics". *Journal of biomechanics*, 7(2):189, 1974.

Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin Riedmiller, and David Silver. Emergence of Locomotion Behaviours in Rich Environments. *arXiv:1707.02286 [cs]*, July 2017. URL `http://arxiv.org/abs/1707.02286`. arXiv: 1707.02286.

Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining Improvements in Deep Reinforcement Learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, April 2018. URL `https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17204`.

Matteo Hessel, Hado van Hasselt, Joseph Modayil, and David Silver. On Inductive Biases in Deep Reinforcement Learning. *arXiv:1907.02908 [cs, stat]*, July 2019. URL `http://arxiv.org/abs/1907.02908`. arXiv: 1907.02908.

Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. `https://github.com/hill-a/stable-baselines`, 2018.

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

Zhewei Huang, Shuchang Zhou, BoEr Zhuang, and Xinyu Zhou. Learning to run with actor-critic ensemble. *arXiv preprint arXiv:1712.08987*, 2017.

Jie Tan, Karen Liu, and G. Turk. Stable Proportional-Derivative Controllers. *IEEE Computer Graphics and Applications*, 31(4):34–44, July 2011. ISSN 0272-1716. doi: 10.1109/MCG.2011.30. URL `http://ieeexplore.ieee.org/document/5719567/`.

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

Sadao Kawamura, Fumio Miyazaki, and Suguru Arimoto. Is a local linear pd feedback control law effective for trajectory tracking of robot motion? In *Proceedings. 1988 IEEE International Conference on Robotics and Automation*, pages 1335–1340. IEEE, 1988.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Jens Kober and Jan Peters. Imitation and Reinforcement Learning. *IEEE Robotics Automation Magazine*, 17(2): 55–62, June 2010. ISSN 1558-223X. doi: 10.1109/MRA.2010.936952. Conference Name: IEEE Robotics Automation Magazine.

Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, September 2013. ISSN 0278-3649, 1741-3176. doi: 10.1177/0278364913495721. URL `http://journals.sagepub.com/doi/10.1177/0278364913495721`.

Arthur D Kuo. The six determinants of gait and the inverted pendulum analogy: A dynamic walking perspective. *Human movement science*, 26(4):617–656, 2007.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

Cynthia R Lee and Claire T Farley. Determinants of the center of mass trajectory in human walking and running. *Journal of experimental biology*, 201(21):2935–2944, 1998.

Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning Quadrupedal Locomotion over Challenging Terrain. *Science Robotics*, 5(47):eabc5986, October 2020. ISSN 2470-9476. doi: 10.1126/scirobotics.abc5986. URL `http://arxiv.org/abs/2010.11251`. arXiv: 2010.11251.

Seunghwan Lee, Moonseok Park, Kyoungmin Lee, and Jehee Lee. Scalable muscle-actuated human simulation and control - Full body Musculosceletal Model. *ACM Transactions on Graphics*, 38(4):73:1–73:13, July 2019. ISSN 0730-0301. doi: 10.1145/3306346.3322972. URL `https://doi.org/10.1145/3306346.3322972`.

Yan Li and D XQ Dai. *Biomechanical engineering of textiles and clothing*. Woodhead Publishing, 2006.

Yuxi Li. Deep Reinforcement Learning. *arXiv:1810.06339 [cs, stat]*, October 2018. URL `http://arxiv.org/abs/1810.06339`. arXiv: 1810.06339.

Aaron D. Likens and Nick Stergiou. Chapter 2 - basic biomechanics. In Nick Stergiou, editor, *Biomechanics and Gait Analysis*, pages 17 – 63. Academic Press, 2020. ISBN 978-0-12-813372-9. doi: https://doi.org/10.1016/B978-0-12-813372-9.00002-6. URL `http://www.sciencedirect.com/science/article/pii/B9780128133729000026`.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv:1509.02971 [cs, stat]*, July 2019. URL `http://arxiv.org/abs/1509.02971`. arXiv: 1509.02971.

Galo Maldonado, Philippe Souères, and Bruno Watier. From biomechanics to robotics. In *Biomechanics of Anthropomorphic Systems*, pages 35–63. Springer, 2019.

Josh Merel, Yuval Tassa, Dhruva TB, Sriram Srinivasan, Jay Lemmon, Ziyu Wang, Greg Wayne, and Nicolas Heess. Learning human behaviors from motion capture by adversarial imitation. *arXiv:1707.02201 [cs]*, July 2017. URL `http://arxiv.org/abs/1707.02201`. arXiv: 1707.02201.

Alberto Maria Metelli, Flavio Mazzolini, Lorenzo Bisi, Luca Sabbioni, and Marcello Restelli. Control frequency adaptation via action persistence in batch reinforcement learning. *arXiv preprint arXiv:2002.06836*, 2020.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN 0028-0836, 1476-4687. doi: 10.1038/nature14236. URL `http://www.nature.com/articles/nature14236`.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

Seyed Sajad Mousavi, Michael Schukat, and Enda Howley. Traffic Light Control Using Deep Policy-Gradient and Value-Function Based Reinforcement Learning. *arXiv:1704.08883 [cs]*, May 2017. URL `http://arxiv.org/abs/1704.08883`. arXiv: 1704.08883.

Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. Bridging the Gap Between Value and Policy Based Reinforcement Learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2775–2785. Curran Associates, Inc., 2017. URL `http://papers.nips.cc/paper/6870-bridging-the-gap-between-value-and-policy-based-reinforcement-learning.pdf`.

Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, et al. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*, 2015.

Andrew Y Ng. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78, 2004.

Jonathan Oehlke, Philipp Beckerle, André Seyfarth, and Maziar A Sharbafi. Human-like hopping in machines. *Biological cybernetics*, 113(3):227–238, 2019.

Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22:1345–1359, 2010.

Fabio Pardo, Arash Tavakoli, Vitaly Levdik, and Petar Kormushev. Time Limits in Reinforcement Learning. *arXiv:1712.00378 [cs]*, July 2018. URL `http://arxiv.org/abs/1712.00378`. arXiv: 1712.00378.

Xue Bin Peng and Michiel van de Panne. Learning locomotion skills using DeepRL: does the choice of action space matter? In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 1–13, Los Angeles California, July 2017. ACM. ISBN 978-1-4503-5091-4. doi: 10.1145/3099564.3099567. URL `https://dl.acm.org/doi/10.1145/3099564.3099567`.

Xue Bin Peng, Glen Berseth, Kangkang Yin, and Michiel Van De Panne. DeepLoco: dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics*, 36(4):41:1–41:13, July 2017. ISSN 0730-0301. doi: 10.1145/3072959.3073602. URL `https://doi.org/10.1145/3072959.3073602`.

Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. DeepMimic: example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics*, 37(4): 1–14, August 2018a. ISSN 0730-0301, 1557-7368. doi: 10.1145/3197517.3201311. URL `https://dl.acm.org/doi/10.1145/3197517.3201311`.

Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3803–3810, May 2018b. doi: 10.1109/ICRA.2018.8460528. URL `http://arxiv.org/abs/1710.06537`. arXiv: 1710.06537.

Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Lee, Jie Tan, and Sergey Levine. Learning Agile Robotic Locomotion Skills by Imitating Animals. *arXiv:2004.00784 [cs]*, July 2020. URL `http://arxiv.org/abs/2004.00784`. arXiv: 2004.00784.

Athanasios S Polydoros and Lazaros Nalpantidis. Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86(2):153–173, 2017.

Marko B Popović. *Biomechanics and robotics*. CRC Press, 2013.

Daniele Reda, Tianxin Tao, and Michiel van de Panne. Learning to Locomote: Understanding How Environment Design Matters for Deep Reinforcement Learning. *arXiv:2010.04304 [cs]*, October 2020. doi: 10.1145/3424636.3426907. URL `http://arxiv.org/abs/2010.04304`. arXiv: 2010.04304.

Douglas H. Richie Jr. *Human Walking: The Gait Cycle*, pages 45–61. Springer International Publishing, Cham, 2020. ISBN 978-3-030-54201-6. doi: 10.1007/978-3-030-54201-6_2. URL `https://doi.org/10.1007/978-3-030-54201-6_2`.

Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. *arXiv:1011.0686 [cs, stat]*, March 2011. URL `http://arxiv.org/abs/1011.0686`. arXiv: 1011.0686.

Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. *arXiv:1806.01242 [cs, stat]*, June 2018. URL `http://arxiv.org/abs/1806.01242`. arXiv: 1806.01242.

Stefan Schaal. Learning from Demonstration. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 1040–1046. MIT Press, 1997. URL `http://papers.nips.cc/paper/1224-learning-from-demonstration.pdf`.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv:1707.06347 [cs]*, August 2017. URL `http://arxiv.org/abs/1707.06347`. arXiv: 1707.06347.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation. *arXiv:1506.02438 [cs]*, October 2018. URL `http://arxiv.org/abs/1506.02438`. arXiv: 1506.02438.

Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, 2019.

Jonah Siekmann, Srikar Valluri, Jeremy Dao, Lorenzo Bermillo, Helei Duan, Alan Fern, and Jonathan Hurst. Learning Memory-Based Control for Human-Scale Bipedal Locomotion. *arXiv:2006.02402 [cs]*, June 2020. URL `http://arxiv.org/abs/2006.02402`. arXiv: 2006.02402.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Ozgur Simsek, Simon Algorta, and Amit Kothiyal. Why Most Decisions Are Easy in Tetris—And Perhaps in Other Sequential Decision Problems, As Well. In *International Conference on Machine Learning*, pages 1757–1765. PMLR, June 2016. URL `http://proceedings.mlr.press/v48/simsek16.html`. ISSN: 1938-7228.

Seungmoon Song, Łukasz Kidziński, Xue Bin Peng, Carmichael Ong, Jennifer L. Hicks, Serge Levine, Christopher Atkeson, and Scot Delp. Deep reinforcement learning for modeling human locomotion control in neuromechanical simulation. *bioRxiv*, page 2020.08.11.246801, August 2020. doi: 10.1101/2020.08.11.246801. URL `https://www.biorxiv.org/content/10.1101/2020.08.11.246801v1`. Publisher: Cold Spring Harbor Laboratory Section: New Results.

Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction*. Adaptive computation and machine learning series. The MIT Press, Cambridge, Massachusetts, second edition edition, 2018. ISBN 978-0-262-03924-6.

Richard S Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In S. A. Solla, T. K. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 1057–1063. MIT Press, 2000. URL `http://papers.nips.cc/paper/1713-policy-gradient-methods-for-reinforcement-learning-with-function-approximatio`
`pdf`.

Csaba Szepesvári. Algorithms for reinforcement learning. *Morgan and Claypool*, 2009.

Arash Tavakoli, Fabio Pardo, and Petar Kormushev. Action branching architectures for deep reinforcement learning. *arXiv preprint arXiv:1711.08946*, 2017.

Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7), 2009.

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

John N Tsitsiklis and Benjamin Van Roy. Analysis of temporal-diffference learning with function approximation. In *Advances in neural information processing systems*, pages 1075–1081, 1997.

Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. NerveNet: Learning Structured Policy with Graph Neural Networks. February 2018. URL `https://openreview.net/forum?id=S1sqHMZCb`.

C. Wright. Understanding the role of the discount factor in reinforcement learning. Cross Validated, 2019. URL `https://stats.stackexchange.com/q/428157`. URL:https://stats.stackexchange.com/q/428157 (version: 2019-09-21).

Zhaoming Xie, Glen Berseth, Patrick Clary, Jonathan Hurst, and Michiel van de Panne. Feedback control for cassie with deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1241–1246. IEEE, 2018.

Zhaoming Xie, Patrick Clary, Jeremy Dao, Pedro Morais, Jonanthan Hurst, and Michiel Panne. Learning locomotion skills for cassie: Iterative design and sim-to-real. In *Conference on Robot Learning*, pages 317–329, 2020.

Chuanyu Yang, Kai Yuan, Shuai Heng, Taku Komura, and Zhibin Li. Learning Natural Locomotion Behaviors for Humanoid Robots Using Human Bias. *IEEE Robotics and Automation Letters*, 5(2):2610–2617, April 2020. ISSN 2377-3766. doi: 10.1109/LRA.2020.2972879. Conference Name: IEEE Robotics and Automation Letters.

Wenhao Yu, Greg Turk, and C. Karen Liu. Learning Symmetric and Low-energy Locomotion. *ACM Transactions on Graphics*, 37(4):1–12, August 2018. ISSN 0730-0301, 1557-7368. doi: 10.1145/3197517.3201397. URL `http://arxiv.org/abs/1801.08093`. arXiv: 1801.08093.

Kuangen Zhang, Zhimin Hou, Clarence W de Silva, Haoyong Yu, and Chenglong Fu. Teach biped robots to walk via gait principles and reinforcement learning with adversarial critics. *arXiv preprint arXiv:1910.10194*, 2019.

Wenshuai Zhao, Jorge Pena Queralta, and Tomi Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. *arXiv preprint arXiv:2009.13303*, 2020.

Zhuobin Zheng, Chun Yuan, Zhihui Lin, Yangyang Cheng, and Hanghao Wu. Self-Adaptive Double Bootstrapped DDPG. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 3198–3204, Stockholm, Sweden, July 2018. International Joint Conferences on Artificial Intelligence Organization. ISBN 978-0-9992411-2-7. doi: 10.24963/ijcai.2018/444. URL `https://www.ijcai.org/proceedings/2018/444`.

Yuke Zhu, Ziyu Wang, Josh Merel, Andrei Rusu, Tom Erez, Serkan Cabi, Saran Tunyasuvunakool, János Kramár, Raia Hadsell, Nando de Freitas, and Nicolas Heess. Reinforcement and Imitation Learning for Diverse Visuomotor Skills. *arXiv:1802.09564 [cs]*, May 2018. URL `http://arxiv.org/abs/1802.09564`. arXiv: 1802.09564.

Zhuangdi Zhu, Kaixiang Lin, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey. *arXiv preprint arXiv:2009.07888*, 2020.

A. Appendix

A.1. PPO Hyperparameters

Table A.1.: PPO Hyperparameters used in our experiments.

Hyperparameter	Value
Policy Network Layer Sizes	512 - 512 - 8
Value Function Layer Sizes	512 - 512 - 1
Activation Functions	ReLU
Weight Initialization	Random Orthogonal Matrix. Weights in the policy's output layer are scaled by 0.001.
Initial standard deviation	0.5
Learning rate	Linear decay from 5×10^{-4} to 1×10^{-6} over the cource of training
Discounting factor	0.995
Number of parallel environments	8
Batch size	32768
Minibatch size	2048
Number of optimization epochs per batch	4
Cliprange (same for policy and value function)	0.15
Lambda	0.95
Entropy Coefficient	-0.0075

A.2. Additional Figures

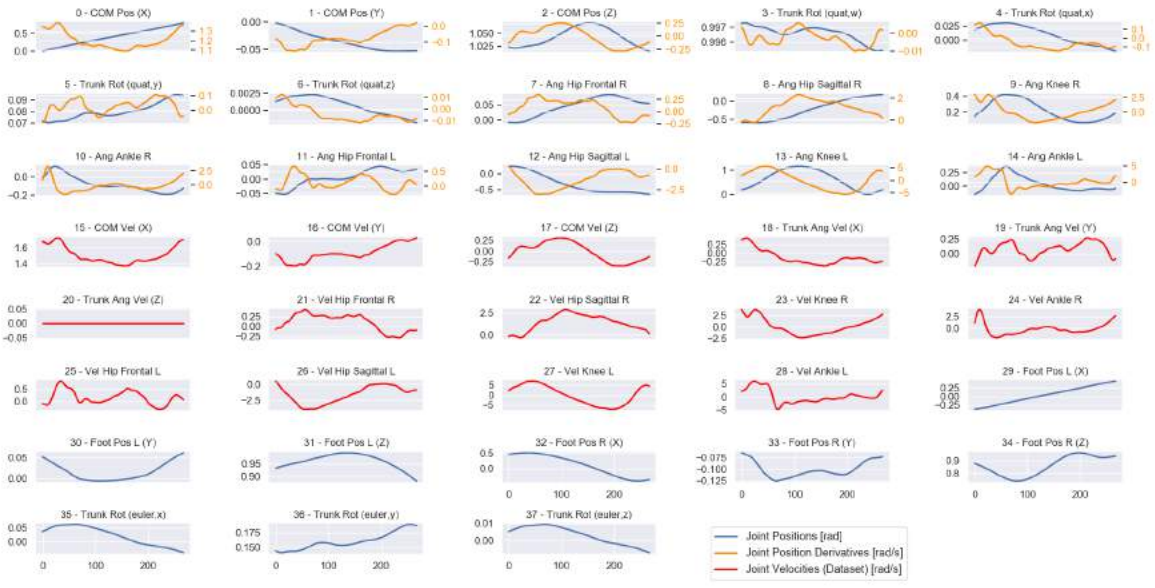


Figure A.1.: **Organization of the Motion Capturing Data.** The plots show the available kinematics data for a single step of the expert trajectories in the same order they are saved in. The orange lines illustrate the first derivative of joint positions to assure the corresponding joint velocities are correctly assigned to each joint.

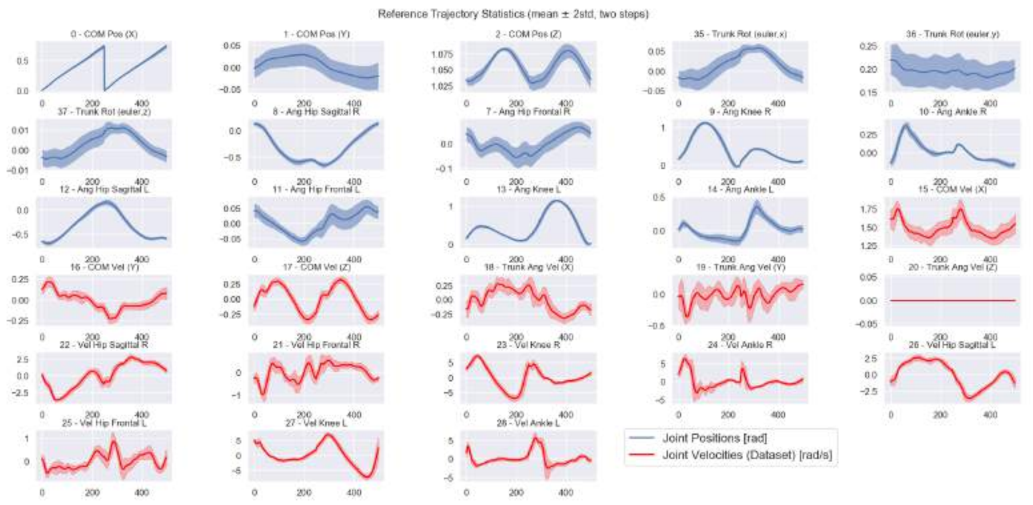


Figure A.2.: **Distribution of Kinematic Trajectories.** This plot helps to understand the variance in the used mocap data. The subfigures display the mean and two standard deviations over the 30 available steps for a whole gait cycle. The kinematics are restricted to those used in our experiments. We can clearly see the high difference in variance across different joints. The variance of joint velocities is higher compared to the joint positions with exception of the hip in the frontal plane. The COM position in the X direction is set to zero at the beginning of each step. The numbers in the subfigure titles refer to the position of the corresponding joint trajectory in the dataset. Please refer to Section 5.1 for a detailed description of the data.

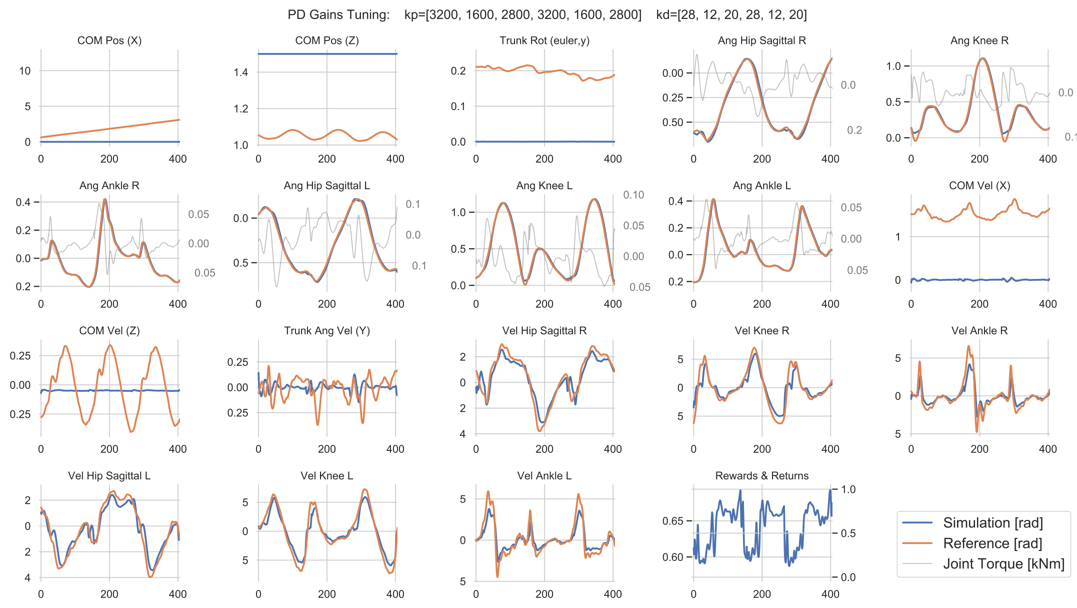


Figure A.3.: **Quality of the Tuned PD Controllers.** In order to investigate the influence of different action spaces on sample efficiency, we extend our walker model with PD position controllers in each joint. Section 5.2.3 offers a detailed description on the tuning process. Table 5.1 presents the specified PD gains for each joint. This figure shows the capability of the chosen gains to closely follow the desired trajectories while the character's torso is held in the air (hence the constant COM positions and Trunk Angle).

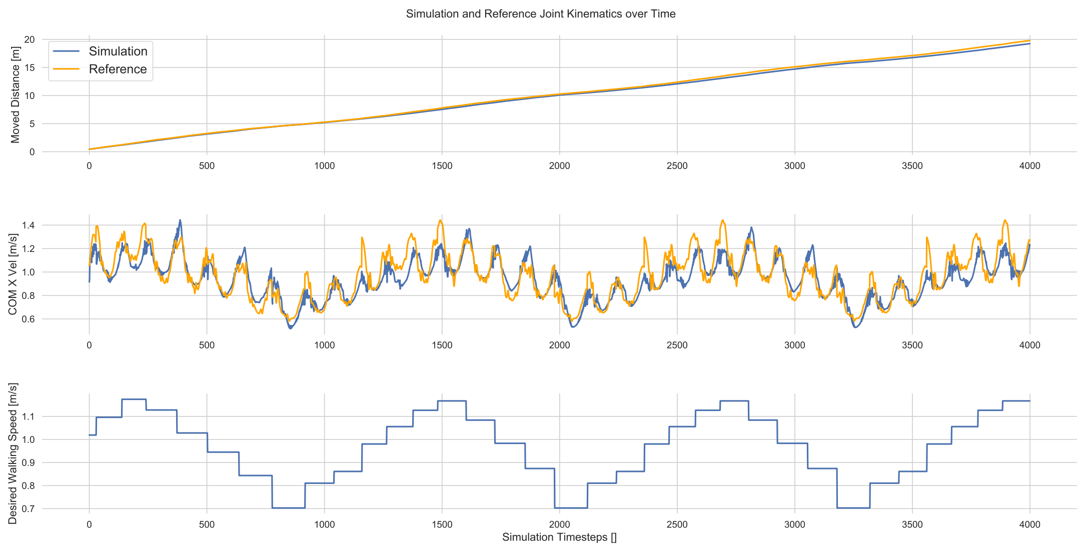


Figure A.4.: **Our Approach Generalizes to Learning Acceleration and Deceleration During Straight Walking.** Next to using the motion capturing data containing the 30 steps recorded during walking at a constant speed, we use the data containing 250 steps recorded while the treadmill velocity followed a ramp profile (Section 5.1). With this data, our approach learns to accelerate and decelerate during walking at a maximum rate of $0.1m/s^2$.