Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

BACHELOR THESIS

Learning to Kick from Demonstration with Deep Reinforcement Learning

Department of Informatics
MIN Faculty
Universität Hamburg

Timon Engelke

timon.engelke@studium.uni-hamburg.de
B. Sc. Informatik

Thesis Advisors: Dr. Matthias Kerzel
Marc Bestmann

Abstract

In this thesis, reinforcement learning from demonstration is applied to train a humanoid robot to kick in simulation. As demonstration, an existing kick engine from the RoboCup Humanoid League is used. The proximity to the demonstration is included into the reward function. Thereby, a kick is produced that follows the trajectory of the demonstration while simultaneously improving the kicked distance. To gain a deeper understanding of the neural network, an ablation study on different observation spaces is conducted, i. e. different sensor data is given to the network. The resulting kicks are evaluated with regards to the kicked distance, stability, and sample efficiency. The results show that an observation containing minimal data can already improve an existing solution, adding information about the robot's orientation improves the stability of the motion, and particularly noisy sensor input worsens the result.

Zusammenfassung

In dieser Arbeit wird verstärkendes Lernen mit einer Demonstration genutzt, um einem humanoiden Roboter das Schießen eines Balls in der Simulation beizubringen. Als Demonstration wird eine bestehende Lösung aus der humanoiden Liga im RoboCup genutzt. Die Nähe zu dieser Bewegung wird beim Lernprozess in die Belohnungsfunktion integriert. Es resultiert ein Schuss, der der Trajektorie der Demonstration folgt und gleichzeitig die zurückgelegte Entfernung des Balles im Vergleich zur Demonstration erhöht. Um ein besseres Verständnis des neuronalen Netzes zu erhalten, werden verschiedene Beobachtungsräume, also verschiedene Sensordaten als Eingabe, untersucht. Die resultierenden Schüsse werden in Hinblick auf die Schussdistanz, die Stabilität und die Lerngeschwindigkeit evaluiert. Die Ergebnisse zeigen, dass eine Beobachtung mit wenig Informationen bereits ausreichen kann, um eine bestehende Lösung zu verbessern, Informationen über die Orientierung des Roboters die Stabilität der Bewegung erhöhen und verrauschte Sensordaten das Ergebnis verschlechtern.

Contents

List of Figures

List of Tables

1. Introduction

In the RoboCup Humanoid Soccer Competition [KA98], teams of robots play soccer against each other. In soccer, kicking the ball plays a crucial role, as it is an effective way of moving the ball across the playing field. A good kick leads to faster and more reliable scoring of goals, which in turn leads to winning games and competitions. In addition, kicks can improve the team play of the robots since they make passes possible. Kicking the ball faster than the opponent also leads to a higher chance of winning duels against opponent players. The robot's environment during the game is highly dynamic: During its motions, it can be disturbed by unexpected perturbations due to the uneven surface of the ground or contacts with other players, and the resulting falls usually lead to ball losses. Therefore, a high resistance against external forces is also helpful.

Currently, there are two prevalent approaches to kicking the ball. In keyframe animations, a set of motor positions is prerecorded and played back when the motion should be executed. The disadvantage of this method is that it requires manual fine-tuning and cannot react to its environment. In the other approach, kick engines, a trajectory for the feet is dynamically calculated to kick the ball and react to its environment using techniques from classic control theory. While good results can be achieved with this approach, programming the controllers requires extensive knowledge and adaptation to the exact desired motion.

This thesis proposes a learned approach that can reduce the programming effort and increase the stability compared to previous methods. The approach uses deep reinforcement learning with the Proximal Policy Optimization algorithm [Sch+17] to train a neural network to perform the motion instead of manually programming it. Specifically, learning from demonstration is employed. This technique uses a reference motion, the demonstration, during the learning process to produce a motion that resembles it while also optimizing other objectives, e. g. stability or ball velocity. This way, an existing approach, for example a keyframe animation, can be used as demonstration and thereby be improved with regards to these objectives. Learning from demonstration also avoids common problems of reinforcement learning, mainly because it is less prone to exploits of the reward function, for example by sticking to global minima [3], and less likely to produce the peculiar movements often seen with reinforcement learning, like excessive arm movements [Hee+17].

In this thesis, a static kick generated by the kick engine that is currently used by the Hamburg Bit-Bots RoboCup team is used as demonstration for the learning process.

The resulting kick is compared to the demonstration to observe whether the kick has been improved. To analyze which parts of the neural network's input play a role in the resulting kick, an ablation study on the observation space is conducted. Two different action space representations are implemented and compared to each other to see if the results differ. The stability of the resulting kicks is evaluated by applying random pushes to the robot during the kick. The findings of these experiments are not limited to the kicking motion but can also be applied to other motions that could be learned from demonstration.

The environment and techniques used in this thesis are described and explained in Chapter 2. This includes the used robot platform and fundamentals of reinforcement learning and learning from demonstration. Chapter 3 gives an overview of related work to this thesis. Especially, other approaches to kicking the ball and approaches to learning from demonstration are detailed. Most notably, *DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills* [Pen+18] is described, which is the principal model for the work in this thesis. Chapter 4 explains the kick used for demonstration, the general setup of the learning process, including observation and action spaces, as well as the used reward functions and other details of the approach. This setup is then used to conduct the experiments described in Chapter 5, which are consecutively evaluated. These results and the general setup are then discussed in Chapter 6. To conclude, the findings are summarized in Chapter 7, and future work is proposed. The code used for the training has been made publicly available of GitHub[1].

[1] `https://github.com/timonegk/DeepKick`

2. Fundamentals

In this chapter, the general environment of this thesis and the basic techniques used in the approach are described. The chapter starts with a presentation of the RoboCup competition, which forms the environment for this thesis. The Wolfgang Platform, the robot used in this thesis, is then described in Section 2.2. The concept of inverse kinematics used for the different action spaces is described in Section 2.3, followed by a brief introduction to the Robot Operating System, ROS, in Section 2.4. Section 2.5 gives an introduction to reinforcement learning, and the reinforcement learning algorithm that is used in this thesis, Proximal Policy Optimization, is described in Section 2.6. Section 2.7 explains the fundamentals of learning from demonstration. Finally, the Multiobjective Tree-structured Parzen Estimator, an algorithm for optimizing hyperparameters, is introduced in Section 2.8.

2.1. RoboCup

The RoboCup is an international competition that organizes games of robotic soccer. It was founded in 1997 after a Pre-RoboCup was held at the International Conference on Intelligence Robotics and Systems 1996. The Pre-RoboCup-96 was the first competition using soccer games for the promotion of research and education [1]. In 1997, the first official RoboCup was held, and the RoboCup mission was defined as having a team of robots that can win against the human soccer World Cup champions by 2050 [KA98]. Since 1997, the competition was held annually and split up into different leagues.

The Hamburg Bit-Bots are participating in the Humanoid Kid Size League. Figure 2.1 shows a typical game situation. In this league, a robot's appearance and equipment are restricted to resemble humans. The robot's structure must be humanoid, i. e. it has to have a head, two arms, and two legs. Further restrictions regulate the ratio of the body length to the head and leg lengths, the foot size, the width, and the arm length of the robot. The sensors of the robots are also restricted to human-like sensors. Therefore, cameras, force sensors, microphones, accelerometers, gyroscopes, and similar sensors also present in humans are allowed, but other sensors typically used in robotics (e. g. lidars, infrared sensors, or compasses) are prohibited. In the Humanoid Kid Size League, the robot's height is additionally limited to 100cm. [Com21]

Figure 2.1.: Two robots in the Humanoid Kid Size League. On the left, the Wolfgang-OP used by the Hamburg Bit-Bots, on the right the SigmaBan Platform used by Team Rhoban. Differences between the robots in the Humanoid League are clearly visible. (Image used with permission of the owner.)

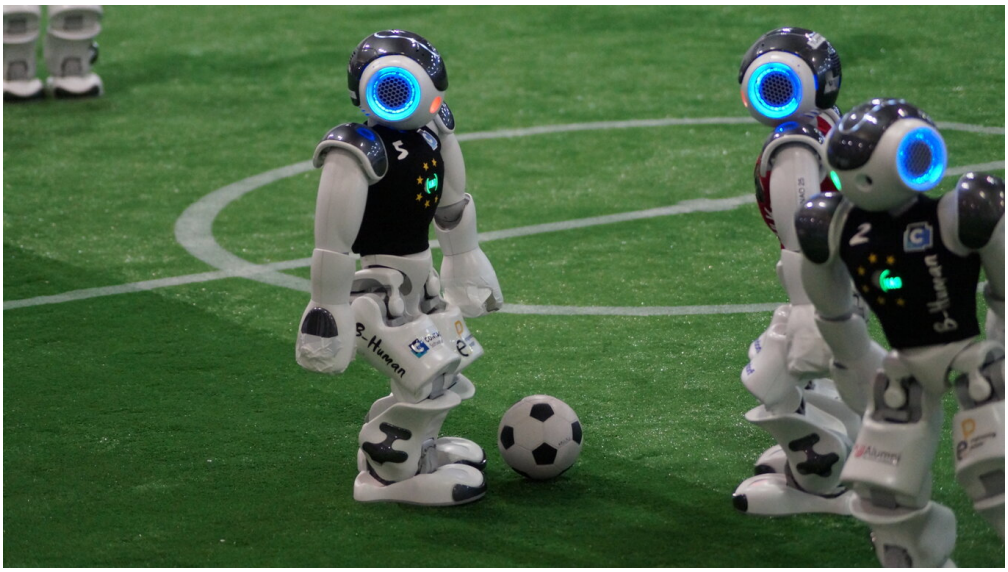


Figure 2.2.: A game in the Standard Platform League. (Image used with permission of the owner.)

In another league, the Standard Platform League, the robots are not constructed by the teams themselves. Instead, all teams use the same robot, the NAO robot by SoftBank Robotics. Its proportions are human-like, and it is equipped with 25 motors, two cameras (pointing forwards and downwards), force sensors in the feet, an inertial measurement unit, and two ultrasonic sensors [2]. An exemplary photo from a game in the Standard Platform League can be seen in Figure 2.2.

There are various challenges in RoboCup Soccer, ranging from hardware and electronics design to high-level behavior planning. To successfully play soccer, a robot has to be able to walk and kick reliably. It has to be able to fall without damaging its hardware and be able to stand up from the ground. A vision system must exist in order to detect the ball, goals, field lines, and other robots on the field. In order to localize itself on the field, the robot must keep track of its own position on the field and use field markers like lines to update its position estimate. Finally, the robots of a team must be able to communicate their strategy in order to play together.

To drive the development forward, the laws of the game are continually updated to make the game more similar to an actual game of human soccer. An incremental, event-triggered roadmap is used to update the rules when specific achievements (e. g. walking speed, kick distance, number of goal kicks, etc.) are reached by a minimum number of teams [Com20].

2.2. Robot Platform

The robot used in this thesis is the Wolfgang Open Platform [Bes+21]. This humanoid robot platform is based on the Nimbro Open Platform [Sch+12] and was designed and developed by the WF Wolves and refined and improved by the Hamburg Bit-Bots. The robot has twenty degrees of freedom, six per leg, three per arm, and two in the head. It weighs approximately 7.5 kg and is 80 cm tall. To provide robustness against falls, series elastic actuators are used in the shoulders and in the neck. They consist of compliant elements that reduce the impact of sudden external forces on the motor and prevent damages. In addition, a torsion spring is added to each knee to reduce the load on the knee joint, especially in situations where the robot stands only on one leg. Its root, or base link, is located at the bottom of its torso, between the legs.

Two inertial measurement units, one in the head and another one in the body, are used to determine angular velocities and linear accelerations of the robot. These measurements can be used to derive the current roll (rotation to the left/right) and pitch (rotation to the front/back) of the robot. Each foot is equipped with four pressure sensors located at the corners of the foot [BGZ19]. The actuators used in the robot are the Dynamixel MX-64, MX-106, and XH540 servos. The applied torque is computed by internal PID controllers that are given the servo's target position. The current positions, velocities, and efforts

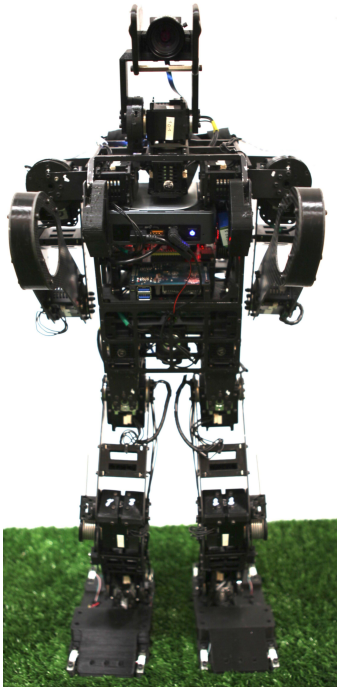


Figure 2.3.: The Wolfgang Open Platform [Bes+21]

can be read from the servos. The control loop frequency, which is the frequency at which the motors can be actuated and the sensor values of the servos, IMUs, and foot pressure sensors can be read, is up to 715 Hz. The camera used in the Wolfgang-OP is a Basler acA2040-35gc camera providing images of 2048x1536 px at 36 FPS via Gigabit Ethernet.

For computation, the platform contains three computers. An Intel NUC is used for most of the high-frequency computation, including the motion stack responsible for walking, kicking, and standing up. The Odroid-XU4 can be used for other, less time-critical tasks but currently remains unused. The computations for neural networks for the vision pipeline happen on a dedicated device, an Intel Neural Compute Stick 2 that provides hardware acceleration for deep neural networks.

2.3. Inverse Kinematics

In robotics, forward kinematics are used to calculate the position of a robot's links or end effectors from the positions of the individual joints. It can easily be solved by starting at the root link and applying the transforms of successive joints until the end effector is reached. The inverse problem is to calculate the joint positions required to place an

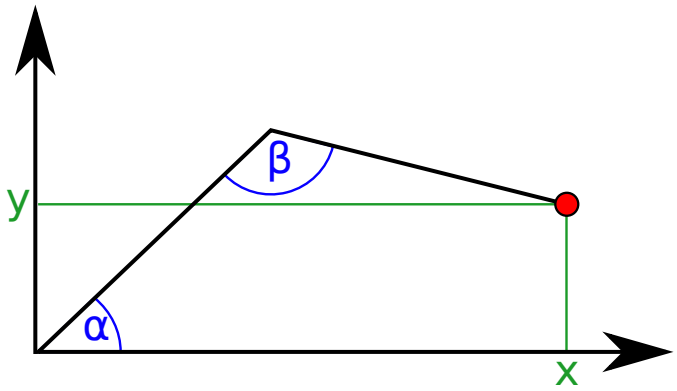


Figure 2.4.: With forward kinematics, the position of the red circle (i. e. x and y) can be calculated from the angles α and β. To get these angles from the position of the circle, inverse kinematics has to be used.

end effector at a given position. The process of solving this problem is called inverse kinematics (IK). A visualization of the problem can be found in Figure 2.4.

In general, there are two different approaches to inverse kinematics: analytic IK and numeric IK. The former calculates the joint positions directly and is therefore significantly faster than numeric approaches. However, many kinematic structures are not suitable for analytic IK because no formula for deriving the target positions has been found. A numeric IK is more general but also slower because the solution has to be derived iteratively. Different numeric algorithms, for example the pseudo-inverse Jacobian method or gradient descent algorithms, exist. However, they are often prone to finding local minima instead of the globally best solution.

In this thesis, the library BioIK [Rup17] is used, which implements a numeric approach. It uses a memetic algorithm that combines evolutionary algorithms capable of finding global minima more reliably with gradient-based optimization methods to quickly optimize the solution found by the evolutionary algorithm.

2.4. ROS

ROS, the Robot Operating System, is a middleware framework commonly used in robotics [Qui+09]. It mainly targets C++ and Python. Software for ROS is divided into several independent nodes that communicate with each other. The ROS master manages the connections between these nodes and provides common parameters. The communication between the nodes happens asynchronously via messages. A message defines a data interface and can be published on a topic. Other nodes can receive the message by subscribing to the corresponding topic.

There are also two bidirectional communication means: services and actions. While services offer a simple request-response pattern and an infrastructure separate from the topics, actions support longer-running queries with intermediate updates and are built on top of the message infrastructure. An action consists of three message definitions: the goal, the result, and the feedback. The goal is sent by the action client to the action server when it requests an action. During the execution of the action, the action server can send back information to the client using the feedback message. When the action is finished, the server sends the result message containing whether the action terminated successfully and further information on the result of the action.

2.5. Reinforcement Learning

Reinforcement Learning is a machine learning technique often used in robotics. Figure 2.5 shows the general cycle of reinforcement learning. An agent receives a representation of its surroundings and its current state. Based on this information, it makes an action that then interacts with the environment. The result of this action is interpreted and, in combination with the new state of the environment after the action was performed, a reward is given to the agent. In other words, the agent follows a trial-and-error approach to finding a successful behavior.

Formally, reinforcement learning is based on Markov Decision Processes. A Markov Decision Process is a tuple (S, A, P, R) where S denotes a set of possible states, A is a set of actions that can be taken, P is a probabilistic function mapping a specific state

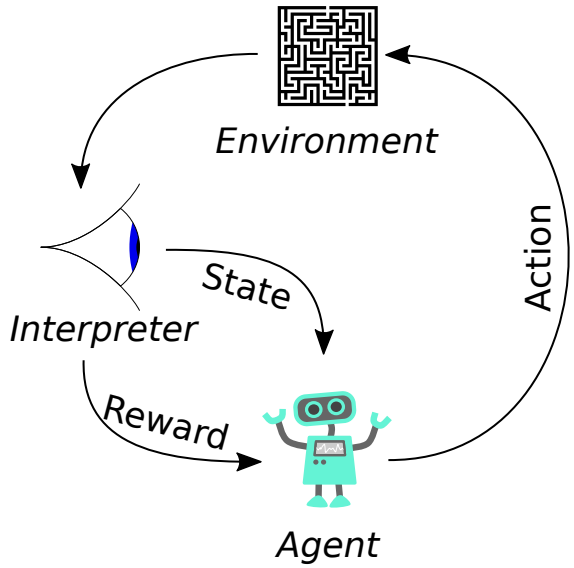


Figure 2.5.: Illustration of reinforcement learning (based on [4]).

and action to a new state, and R is a function mapping a transition between two states to a reward. For continuous states and actions, like they are used in this thesis, S and A are not sets of possible states or actions, as they cannot be described in a discrete set. Instead, states and actions are described by the state (or observation) space and the action space, and all possible states or actions lie in their respective spaces. Additionally, a start distribution and terminal states may be specified. In a Markov Decision Process, the goal is to find a (potentially probabilistic) policy π that maps states to actions and optimizes the received reward. Reinforcement Learning is a way of finding such a policy.

In reinforcement learning, a discounted reward G_t is usually used that also takes into account future rewards. It is defined as

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k},$$

where γ is the discount factor that describes how important future rewards are at a certain point in time. The value function $V_\pi(s)$ describes the value of a state s, that is the discounted reward in state s when the policy π is followed. The action-value function $Q_\pi(s, a)$ describes the value of the action a in state s, i. e. the discounted reward under policy π when action a is taken in state s. [SB15]

The function $A_\pi(s, a)$ is the advantage of action a in state s under policy π [Sch+16]. It describes how much the action a is better than the other possible actions. It is defined as

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s).$$

Usually, the real values of these functions are not known, and estimates of the functions are used. They are denoted with $\hat{V}$, $\hat{Q}$, and $\hat{A}$, respectively.

In deep reinforcement learning, some of these functions are modeled by neural networks. For the approach in this thesis, with continuous state and action spaces, this is the case for both the policy π and the value function V_π. The input of the policy network is the current state, its output is the action, and the reward is used to improve the network incrementally. The value function network has the same input but only a single output. To train the networks, different reinforcement learning algorithms exist. They are divided into on-policy and off-policy algorithms. In off-policy algorithms, the network is trained on data that was prerecorded using a different policy. As opposed to this, on-policy algorithms can update and improve the policy that is used to record the training data during the training.

The learning process can be divided into episodes. Each episode is one complete execution of the robot's task, for example one kick. The observed states, actions, and rewards during the episodes are collected in a rollout buffer. Once the rollout buffer is filled, the collected data is divided into random minibatches that are used to train the policy and the value function networks according to the used reinforcement learning algorithm.

One pass over the full rollout buffer is called an epoch, and usually, multiple epochs are performed on the data of a rollout buffer. For on-policy algorithms, the rollout buffer is then discarded, and the updated policy is used during the collection of new data. The rollout buffer size, the minibatch size, the number of epochs, and the overall number of timesteps (horizon) are hyperparameters to the learning process.

2.6. Proximal Policy Optimization

Proximal Policy Optimization (PPO) [Sch+17] is a model-free on-policy algorithm for deep reinforcement learning. It is based on policy gradient methods which estimate the gradient of a policy and use a stochastic gradient ascent algorithm to optimize the policy. The disadvantage of these methods is that they can produce very large policy updates leading to a significant deviation from the previous policy. This can lead to producing policies that never recover from the harmful update. PPO offers a solution to this problem by introducing a clipped objective function that clips the advantage-based objective function such that large policy updates are prevented. This function is defined as

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

where $\hat{\mathbb{E}}_t$ denotes the expectation of the enclosed function. The function's argument θ is a set of parameters for the policy π. r_t is the probability ratio between the new policy π_θ and the old policy $\pi_{\theta_{\text{old}}}$ which is defined as $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$. In other words, r_t describes

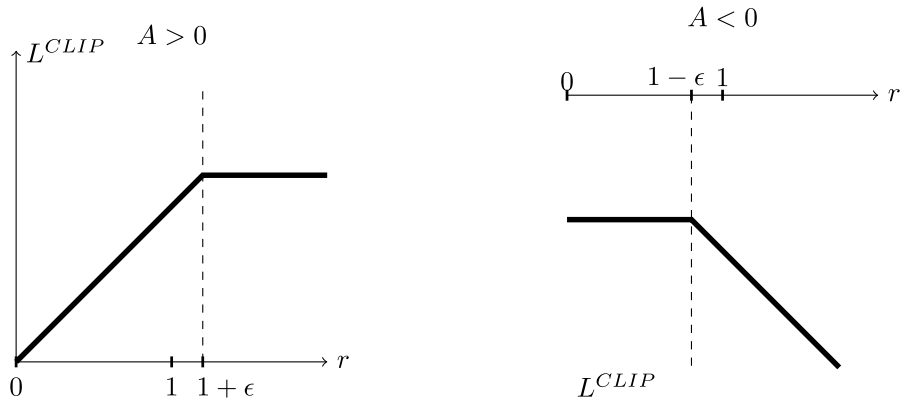


Figure 2.6.: The two cases of clipping the objective function. On the left, the advantage is positive, therefore the action was good. If r is high, a large policy update would be done. This is avoided by clipping the objective function. On the right, the advantage is negative, i. e. the action was bad. Here, the objective function is clipped to avoid updates that would drastically reduce the possibility of the action occurring. (Illustration based on [Sch+17].)

how much the new policy differs from the old policy. When the new policy does not differ from the old one, the value of r_t is 1. When the new policy prefers the action a_t in state s_t compared to the old policy, it is larger than 1; if its probability decreases, the value of r_t is less than one. The min function in the objective results in the policy not deviating too far from the old policy when the policy became better (i. e., the advantage is positive and the action became more likely or the advantage is negative and the action became less likely) to avoid destructively large updates. When the update made the policy worse, the first term of the min function is selected, no clipping happens, and the policy gets appropriately updated because, in this case, large updates are desirable. Figure 2.6 shows a graphical representation of the clipped objective function. For estimation of the advantage function $\hat{A}_t$, the General Advantage Estimator [Sch+16] can be used.

To include the performance of the value function, an additional squared-error loss for the value function, denoted with L^{VF}, is subtracted from the objective. An entropy bonus S can also be added to encourage exploration. The final objective function for PPO, as suggested by the authors, is

$$L(\theta) = \hat{\mathbb{E}}_t \left[L^{CLIP}(\theta) - c_1 L^{VF} + c_2 S_{[\pi_\theta]}(s_t) \right].$$

c_1 and c_2 are weights for the value loss and entropy loss, respectively.

In the paper presenting PPO, the authors achieve better results with this algorithm than with other reinforcement learning algorithms, for example vanilla policy gradient methods or trust-region policy optimization, for a high variety of problems. At the same time, PPO is much easier to implement and less complex than other methods achieving a similar level of performance.

2.7. Learning from Demonstration

In reinforcement learning, a common problem is that the task that should be learned is or cannot be defined very well. Especially if the reward depends on actions over a longer period of time, linking actions to rewards becomes more difficult [Sch97]. To circumvent this problem, learning from demonstration, also called imitation learning, can be used. Similar to how humans can learn faster by imitating the behavior of others, in this approach, a demonstration is added to the training process as a model of the desired behavior. Thus, the learning policy does not have to learn from its own experiences alone but can use the demonstration as an example and learn by imitating it.

One approach to imitation learning is to learn the actions of the demonstration in a supervised way before starting the reinforcement learning process, where the previously learned demonstration is then applied and further optimized with a reward for the task that it should solve. This approach is taken in imitation learning algorithms like Behavioral Cloning [BS95], and numerous improvements have been proposed [RB10; TWS18].

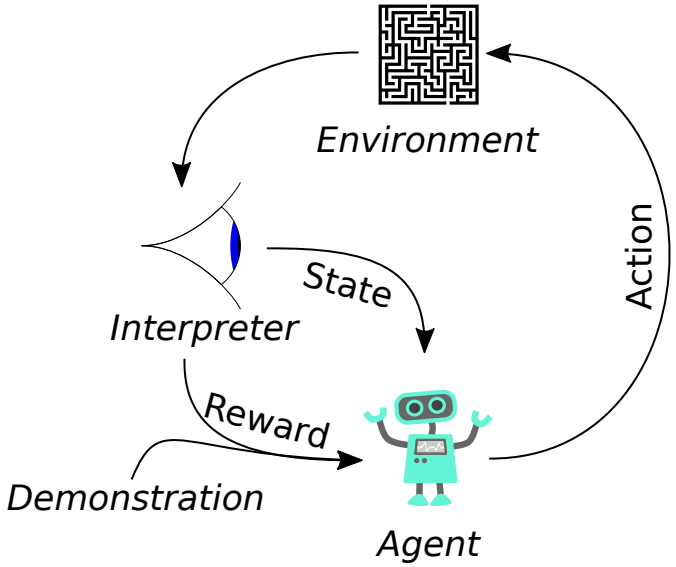


Figure 2.7.: Learning from Demonstration, with demonstration included in the reward function. (Illustration based on [4].)

Another approach is to directly reward a policy for motions that resemble a reference trajectory. For this approach, the reward function consists of two weighted parts, one of which rewards similarity to the reference motion while the other one rewards successful execution of the given task. Figure 2.7 shows an illustration of this progress. The general approach described in Section 2.5 is adapted to include the demonstration in the reward function and no further adaptions to the training have to be made.

While this approach is relatively simple, it has shown excellent results in learning motions for physically simulated characters and complex motions like throwing, running, and kicking in simulation [Pen+18]. Walking motions learned in this way have also been transferred to the real world [Pen+20; Li+21].

2.8. Multiobjective Tree-structured Parzen Estimator

The Multiobjective Tree-structured Parzen Estimator (MOTPE) [Oza+20] is a multiobjective optimization algorithm. It can be used to optimize input parameters for a system under multiple, possibly conflicting, objectives. MOTPE extends the Tree-structured Parzen Estimator (TPE) that can be used for the optimization of single objectives.

In TPE [Ber+11], a set of initial observations (i. e. inputs and their corresponding results) is first split at a given threshold according to their results in a set of good (D_l) and a set of bad (D_g) results. Then, for each set, the respective probability density functions

are modeled with a tree-structured parzen estimator. They are denoted with $l(x)$ and $g(x)$, respectively. The estimator produces the probability density functions by modeling the area around each given observation with a Gaussian distribution and combining the results. Therefore, sampling from $l(x)$ will produce better values and sampling from $g(x)$ will produce bad values. To determine the next candidate to be evaluated, the expected improvement function is calculated. It describes how much a set of input parameters is expected to improve the objective function. Intuitively, it means that the values have a high probability in $l(x)$ and a low probability in $g(x)$. Multiple candidates are sampled from $l(x)$ and the candidate with the best expected improvement is returned. The algorithm is then repeated with the observations including the evaluation of the latest candidate until a terminating condition is met; then, the value with the best objective function result is returned.

In MOTPE, this approach has been extended to make it possible to optimize for multiple objective functions. Because no single threshold for the objective function can be used to split the observations, D_g is instead chosen to include observations that are dominated by other observations (i. e., they perform worse for every objective function). The expected improvement from TPE is replaced by the expected hypervolume improvement that performs a similar estimation of the improvement but accounts for multiple objective functions. This function is again used to determine the candidate with the biggest expected hypervolume improvement, which can then be evaluated and added to the observations for the next step. The algorithm terminates after a given number of iterations and returns all candidates that are not dominated.

In general, both TPE and MOTPE are used for hyperparameter optimization problems, for example for deep learning problems. The algorithms are sample-efficient, which is especially helpful for problems where calculating the value of the objective function is computationally expensive.

3. Related Work

This chapter outlines work related to this thesis. Section 3.1 describes general success on learning motions with reinforcement learning. Then, learning from demonstration approaches are described in Section 3.2. The work in this thesis is heavily influenced by DeepMimic, which is described in Subsection 3.2.1. Its successive publication is then presented in Subsection 3.2.2. Section 3.3 describes different other approaches to kicking motions that are used in the RoboCup Humanoid and Standard Platform Leagues.

3.1. Reinforcement Learning for Motions

Reinforcement Learning of motions in robotics is a topic that became more and more relevant over recent years. It is especially useful to solve complex problems that are hard to approach in a conventional way [KBP13].

For example, in [Hee+17], complex walking in different environments was learned on three different robots. For the training, a distributed variant of PPO was used, and the reward function was chosen to simply reward forward motions. During the training, the environment was varied, and the robots learned impressive motions that overcome hurdles, gaps, and other hindrances on their way.

A study comparing the performance of different reinforcement learning algorithms showed that complex problems like walking with humanoid or animal-like robots could be solved successfully by many of these algorithms [Dua+16]. Especially Truncated Natural Policy Gradient (TNPG) [Kak01] and Trust Region Policy Optimization (TRPO) [Sch+15], the predecessors of PPO, performed very well.

In RoboCup, deep reinforcement learning was used as well. In 2019, Abreu, Reis, and Lau used PPO to learn a running motion on a simulated NAO robot [ARL19]. The same authors also used PPO to learn a dribbling motion on the same robot [Abr+19].

However, these approaches are limited to a simulated environment. The learned motions often show peculiar movements, for example wild arm movements, or behaviors that clearly would not work in the real world, for example when inaccuracies in the simulation are exploited by the policy.

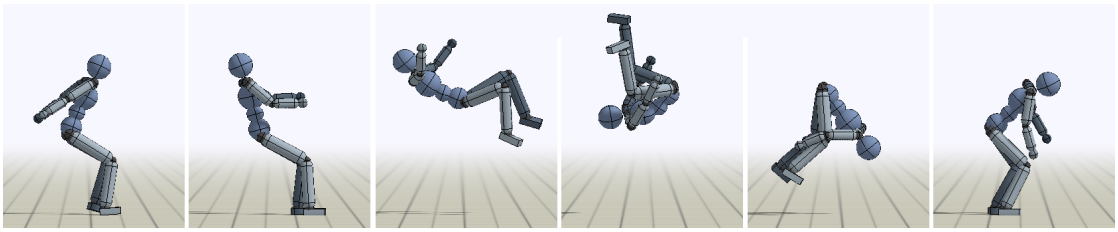


Figure 3.1.: Snapshots of a highly dynamic backflip motion trained in DeepMimic [Pen+18].

3.2. Learning from Demonstration

To overcome the issues of normal reinforcement learning that were mentioned in the previous section, learning from demonstration can be used. Particular successful approaches to learning various motions are described in the following subsections.

3.2.1. DeepMimic

In *DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills* [Pen+18], Peng et al. used imitation learning for complex motions like throwing, running, and kicking on humanoid and non-humanoid robots. They used motion capture data and manually defined keyframe animations as demonstration trajectories. Proximal Policy Optimization was used as the reinforcement learning algorithm.

The robot's state, used as input to the neural network, consisted of the position and orientation of all links relative to the base link that was located at the pelvis, as well as their respective linear and angular velocities and a phase denoting the progress in the reference trajectory. The policy output consisted of positional targets for all motors. Target positions are used instead of torque values, which are often used in other reinforcement learning approaches, because they found that learning target angles improves policy performance and learning speed for several motion control tasks [PP17]. Concerning the network architectures, separate networks for the policy and the value function were used, each consisting of two hidden layers with 1024 and 512 units respectively.

The reward function was defined as a weighted sum of the demonstration reward and the goal reward:

$$r_t = \omega^I r_t^I + \omega^G r_t^G$$

The goal reward r_t^G describes the success at performing the given task, e. g. throwing a ball or moving forward. The demonstration reward r_t^I describes the similarity to the

reference motion. It consists of four weighted parts: The joint position error, the joint velocity error, the end effector offset, and the center of mass offset. Each of these rewards has the following form:

$$r_t^n = \exp\left[-a_n\left(\sum_{k \in V_n} d_k\right)\right]$$

r_t^n describes the partial reward, a_n is a factor for the exponential function, V_n contains the variables of this reward (e. g., the joints), and d_k denotes the difference between the agent and the reference motion for the variable k.

During the training, in order to achieve better results, reference state initialization and early termination were used. With reference state initialization, the initial state for an episode is not the starting state of the motion, as it is often the case in reinforcement learning. Instead, a random state is sampled from the reference motion. This helps mitigate the problem that states that occur later in the motion are difficult to reach with random exploration. In the case of early termination, the episode is terminated and a reward of zero is given for all future states when the agent has reached a particular condition, e. g. when its head link is touching the ground. This has the advantage that the policy does not have to learn states that it will never recover from, and actions leading to such a state are strongly discouraged.

The results of the approach showed that an agent trained with the imitation objective and the goal objective performs significantly better than when only one of the objectives is used. Without the imitation objective, unnatural behavior is developed and without the goal objective, the task is not successfully performed. The combination of both objectives resulted in motions that fulfilled the task while keeping a natural-looking behavior, deviating from the reference motion where necessary to perform better in the task.

3.2.2. Learning Agile Robotic Locomotion Skills by Imitating Animals

In *Learning Agile Robotic Locomotion Skills by Imitating Animals* [Pen+20], the same authors presented another paper on imitation learning, based on the results of DeepMimic. The state representation of the robot was changed to include the three previous poses of the robot, each consisting of the IMU readings of the robot and the joint positions, as well as the three previous actions. Additionally, the target poses from the reference network at four different future timesteps were added to the state. The actions consisted of the target rotations for each joint. A low-pass filter was applied to the actions to smoothen the resulting motions. The reward function is again similar to the one in DeepMimic,

but the center of mass reward was replaced by rewards for the pose and velocity of the root link.

To facilitate sim-to-real-transfer, domain adaptation was used. Similar to domain randomization, the simulation dynamics were changed to achieve a greater robustness against the environment and prevent overfitting on the simulation. To make it possible for the policy to cope with the different dynamics, a representation of the system dynamics was included into the policy input.

In 2021, the same research group published another paper, *Reinforcement Learning for Robust Parameterized Locomotion Control of Bipedal Robots* [Li+21]. Again, good results could be achieved for walking on a two legged robot with a policy trained in a similar way to DeepMimic when domain randomization was used.

3.3. Other Kick Approaches

A common approach for kicking in the humanoid and SPL leagues is to record a static keyframe animations [Riz+19; Bes+18]. These animations consist of a sequence of motor positions and timings that are manually tuned. The movement is generated by interpolating between the keyframe positions, for example using quintic spline interpolation. The advantage of keyframe animations is that little knowledge about the robot is required and programming the keyframe animation framework is generally not very complex. Also, the same animation framework can be used for multiple different tasks, like standup motions and kicks. However, recording the keyframes is a cumbersome task because they have to be tweaked to exactly match the robot and its environment since the animation is static, i. e. the robot cannot react to its surroundings, like uneven ground. To circumvent this problem, stabilizing techniques can be added to the static kick, for example using foot pressure sensors to calculate the center of pressure [All+16].

Some teams integrate their kick directly into the walking engine [Tak+19; PV19]. This approach has the advantage that switching between walking and kicking is very fast because the moving foot of the current step is used to kick the ball during the walking motion. However, this approach tends to violate the *Separation of Concerns* [HL95] code paradigm and leads to more code complexity as the kick becomes more sophisticated. Also, more refined kicks that are able to adapt to different target directions cannot be integrated into the walking as easily.

Another approach is the dynamic kick engine. This engine dynamically calculates a movement, for example using splines, and adapts to changes in the environment, for example by detecting when the robot becomes unstable or the ball moves.

In the Standard Platform League, Müller et al. [MLR11] created a kick engine that uses trajectories based on Bezier curves that can be adapted during the execution of the

motion to account for changing ball positions. To keep the robot stable, the robot's center of mass was balanced over the foot's support polygon and a PID-controller on the angular velocities was implemented.

Also for the NAO robot, Wege [Weg17] proposed Dynamic Movement Primitives to generate the kick trajectories. Stability was achieved by balancing the center of mass over the support polygon and using P-controllers for the ankle motors based on the feedback of the robot's gyroscope.

Kick engines are not very common in the humanoid league. The Hamburg Bit-Bots are using a dynamic kick engine [BES19] that is used as the demonstration in this thesis and will be detailed in Section 4.1. The team ZJUDancer is not using a dynamic kick yet, but plan on developing an approach that dynamically analyzes and optimizes the trajectory of the kicking foot and uses the IMU and servo data as feedback to keep the robot stable [JC20].

While these kinds of controllers and engines can yield stable and well-performing trajectories, designing them requires considerable knowledge of the kinematic structure and the dynamic of the robot as well as the motion that is implemented [Pen+20].

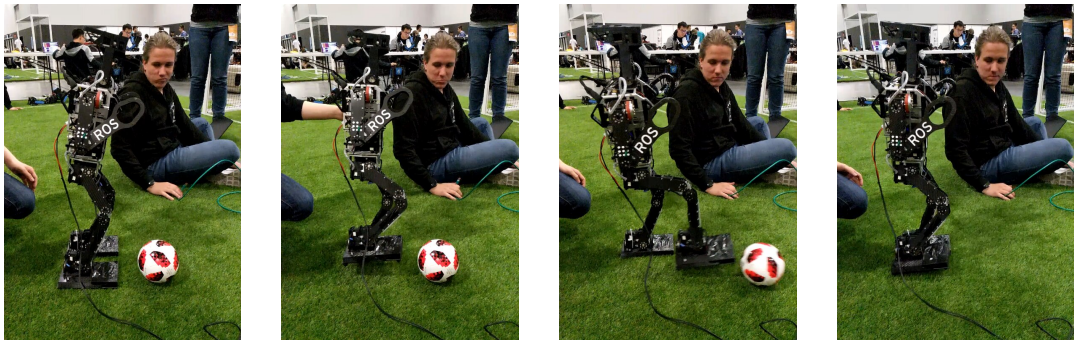
4. Approach

In this chapter, the approach and implementation of this thesis are described. As demonstration and reference motion in the learning process, an existing kick is used. This kick is generated by the kick engine currently used by the Hamburg Bit-Bots which is described in Section 4.1. The training of the agent takes place in the PyBullet simulator. The environment of the simulation and the robot model are described in Section 4.2. Section 4.3 then describes the fundamental parts of the training process, the chosen observation and action spaces as well as the reward function. Finally, the implementation of the framework is detailed in Section 4.4.

4.1. Demonstration

The existing kick engine of the Hamburg Bit-Bots is presented in [BES19]. It consists of four modules: the node, the engine, the stabilizer and the IK. The node provides the ROS interface by subscribing to all relevant messages and providing the action server. The action goal consists of the current position of the ball and the desired kick direction. When an action is requested at the node, it passes the relevant information to the engine. The engine then decides which foot should be used to kick and which side of the foot should be used. Based on that decision, it calculates the splines for the kick. In the engine, the support foot is assumed to be static and all trajectories are calculated relative to it. Therefore, two sets of splines are calculated by the engine, one for the trunk and one for the kicking foot. After the splines are calculated, the main control loop begins. In each step of the loop, the current set of positions is extracted from the splines. These positions are then passed through the stabilizer that performs optional stabilization on the trunk goal, based on the measurements of the IMU or the foot pressure sensors. Once the stabilization has happened, the positions are passed into the IK which performs the inverse kinematics, i. e. calculates the joint positions that should be applied to reach the given cartesian positions. These joint goals are then published by the node and get processed and applied to the motors in a separate node. Figure 4.1 shows the kick in the real world, as used in RoboCup 2019. In Figure 4.2, a side kick generated by the engine is displayed. The simulation environment is the Webots simulator that was used for the virtual RoboCup 2021.

The behavior of the kick can be influenced by various parameters, e. g. the timings of the kick's phases, how far the foot should be raised and retracted, etc. Since the kick will be



(a) Before the kick (b) Raise a foot (c) Kick the ball (d) Retract the foot

Figure 4.1.: A forward kick generated by the kick engine used for the demonstration, on a Wolfgang robot. (Images used with permission of the owner.)

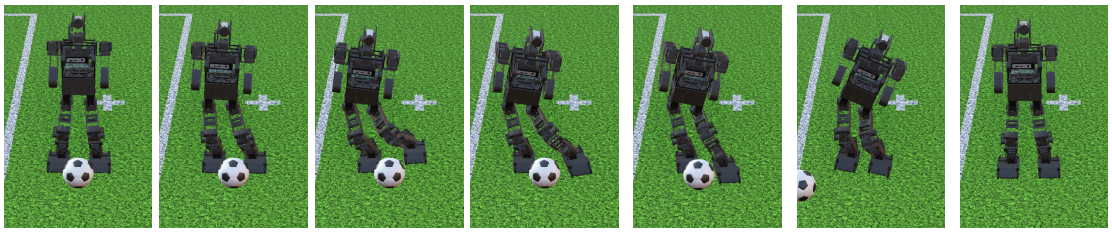


Figure 4.2.: The kick engine performing a side kick in Webots.

used as demonstration for the reinforcement learning, it is important that its parameters are optimized before the learning process. The optimization of the parameters is done using Optuna [Aki+19], a popular framework for hyperparameter optimization. Because Optuna is a Python library and the kick is a C++ project, a Python wrapper module was created to make it possible to call C++ functions from Python.

The best optimization results were obtained using the Multiobjective Tree-structured Parzen Estimator (MOTPE) [Oza+20], a sampler that can optimize different objectives at the same time. The advantage of optimizing multiple objectives is that no manual weighting has to be done and that multiple parameter sets are obtained that perform best for different goals. The used objectives are whether the robot fell after the kick, the kick duration, the maximum velocity of the ball and the directional error of the kick. Using the MOTPE sampler, a parameter set can be obtained that focuses on improving the ball's velocity but might lead to the robot falling and a different parameter set where the kick is stable but the ball is kicked more slowly. The resulting parameters are listed in Appendix A.

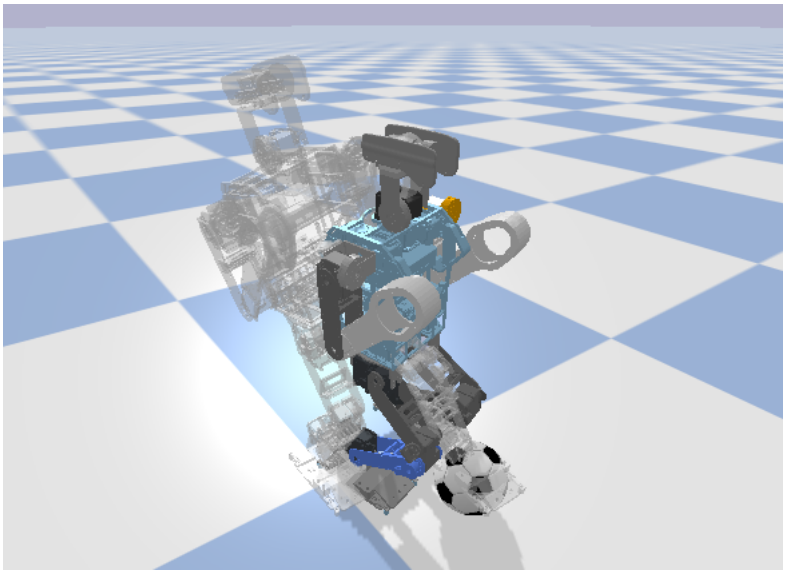


Figure 4.3.: An example image of the robot during training in PyBullet. The opaque robot displays the agent during the training. The transparent robot follows the reference motion.

4.2. Simulation Environment

For the simulation, the PyBullet Physics Engine is used. In the simulation, the robot is placed on an even ground in the *walkready* position, its default position when the kick starts. A ball is placed in front of its left foot. Its size and weight correspond to the ball used in the RoboCup Humanoid Kid Size League. The diameter of the ball is 14cm and its weight is 250g. The friction between the ground and the ball is set to the default values of the *plane* and *sphere* objects in PyBullet. While these values don't perfectly model the actual frictions of the ball and the ground, the exerted forces during the kick are high enough to move the ball at different velocities depending on the strength of the kick.

During the training, the simulator was started in a headless mode without a graphical interface to avoid the overhead of displaying the robot and facilitate running multiple instances of the simulator in parallel on a remote server. However, a graphical interface was available for debugging and displaying results. In this interface, the robot and the reference trajectory were displayed to make it possible to compare the learned motion to the reference motion during the training or evaluation. Figure 4.3 shows the simulation environment with the reinforcement learning agent, the reference motion, and the ball during an early training step.

4.3. Training

In this section, the general procedure of the training is described. Then, the network architectures for the policy and value networks are explained. The observation and action spaces are defined and the used reward function is detailed.

4.3.1. Training Process

The training is divided into episodes, each executing one kick. The duration of the demonstration kick is 1.64s. The episode length is 3s, measured from the beginning of the demonstration. Since reference state initialization is used, the actual length of an episode varies from 3s to 1.36s. The episode length was chosen to be longer than the demonstration so that the robot has to learn to balance itself after the kick. Shorter episode lengths resulted in the robot falling in such a way that the terminating condition would not be reached before the end of the episode.

At the beginning of the episode, the kick engine used as demonstration is initialized with a fixed ball position in front of the left foot and the command to perform a forward kick. Since the objective of this thesis is to produce a single kick and analyze the results for different training setups, the same kick is used for all training episodes; the kick direction and ball position are never changed. This constraint also drastically reduces the complexity of the training. The time of the kick engine is then set to the random time given by the reference state initialization. Then, the position and orientation of the robot and all joint positions are set according to the state given by the reference motion for this time. The velocities of the robot and of each joint are set to zero because no information about the velocity is provided by the kick engine. Since the kick is always performed with the left foot, the ball is positioned in front of it. If the episode is initialized at a point in time where the kick has already happened, the ball is placed at a position further away from the robot, as if it had been kicked correctly.

Then, the simulation is started and the observations are fed into the policy which outputs actions that are applied to the robot. The policy is queried with a frequency of 30Hz, the simulation runs at a frequency of 240Hz. Simultaneously, the demonstration is advanced, also at a frequency of 30Hz.

When the robot's pitch or roll angle, measured at the pelvis, exceed 90 degrees, or when the robot's head falls below the starting height of its pelvis, the robot is assumed to be fallen over and the episode is terminated. A reward of zero is given for this timestep.

The used hyperparameters for the training were obtained with a Tree-Structured Parzen Estimator-based hyperparameter search for the training of a walking policy with the same robot in the same simulator environment using the same reinforcement learning

algorithm. Therefore, the hyperparameters are expected to be adequate for the training even though they are not optimized for this particular problem. The full list of the hyperparameters can be found in Table 4.1.

Parameter	**Value**
Policy Network	512x512, fully connected Activation Function: ReLU Fixed Variance: 0.1
Value Function Network	512x512, fully connected Activation Function: ReLU
Total Number of Timesteps	10 000 000
Number of Timesteps per Batch	2048
Minibatch Size	32
Number of Epochs	5
GAE-λ	0.8
Discount Factor γ	0.98
PPO Clip Range	0.2
Learning Rate	0.000107739192714429
Value Function Loss Coefficient	0.210933418438296
Entropy Coefficient	0.0

Table 4.1.: Full list of hyperparameters used during the training

4.3.2. Network Architecture

In the thesis, two separate neural networks are used for the policy and the value function. Each of the networks consists of two fully connected hidden layers that use the ReLU activation function. The output layer is linear. Only a single output neuron exists for the value function, whereas for the policy function, the number of output neurons depends on the action.

For the policy network, in the output layer, the mean value of a Gaussian distribution is returned. This Gaussian distribution has a fixed variance that is a hyperparameter to the learning process during the training. The fixed variance ensures sufficient random exploration during the training and helps to make the policy more resistant against small random perturbations. During the evaluation, the mean of the Gaussian distribution is always used directly. Figure 4.4 shows a visualization of the network architecture for the policy network.

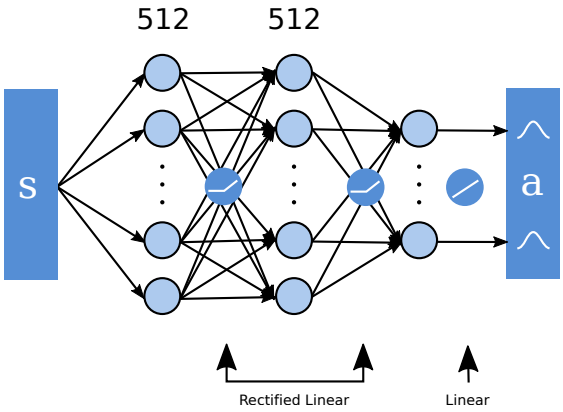


Figure 4.4.: The policy network consists of two fully connected hidden layers with 512 neurons each. The ReLU activation function is used. The output layer is linear and determines the mean of a Gaussian distribution for the actions.

4.3.3. Observation

The observation is the input of the policy network and the basis of all of its decisions. Therefore, the observation can include the current progress in the kick, the state of the robot including different sensor measurements, and any further information about the environment.

Since the sensors available in a humanoid robot are limited and no further exterior information can be accessed, only the sensor input from the inertial measurement unit, the foot pressure sensors, and the motor position sensors is available. In this thesis, the following direct or indirect measurements of the sensors were used:

Phase is a number that monotonously increases with time and is reset to the current phase of the demonstration at the beginning of an episode.

Orientation of the robot, as roll and pitch, derived from the IMU.

Angular Velocities of the robot around its three axes, directly taken from the IMU.

Foot Poses are the position and orientation of both of the robot's feet, calculated with forward kinematics from the joint positions. Each foot pose is represented by x, y, and z for the position and roll, pitch, and yaw for the orientation, both relative to the robot's root.

Foot Velocities are the linear and angular velocities of both feet, calculated by taking the difference between consecutive foot poses and dividing the result by the time difference. The linear velocities are the velocities along the x, y, and z axis of the robot's root, the angular velocities are the velocities around these axes.

	Phase	Orient.	Ang. Vel.	Foot Pos.	Foot Vel.	Pressures
PhaseState	✓	–	–	–	–	–
OrientationState	✓	✓	–	–	–	–
GyroState	✓	✓	✓	–	–	–
FootState	✓	–	–	✓	–	–
FootVelocityState	✓	–	–	✓	✓	–
OrientationFootState	✓	✓	–	✓	–	–
PressureState	✓	–	–	–	–	✓
PressureFootState	✓	–	–	✓	–	✓
ComprehensiveState	✓	✓	✓	✓	✓	✓

Table 4.2.: The different partial observations of which each state consists

Pressures are the measurements of the eight pressure sensors located at the robot's feet. A Butterworth filter [But+30] is applied to them to filter high-frequency noise.

All observations are standardized with a running mean and standard deviation by Stable Baselines 3.

To evaluate the importance of different parts of the observation space, the measurements are combined to a total of nine different states, ranging from the *PhaseState* that only contains the phase information, to the *ComprehensiveState* that contains all of the information mentioned above. Table 4.2 shows the contained information for all states that were used. Notably, the phase information is contained in all states to be able to keep track of the current phase of the kick. On these different states, an ablation study that is described in Section 5.1 is conducted.

4.3.4. Action

The action is the output of the policy. It is applied to the robot to change its state. In this thesis, a joint-space action and a cartesian-space action have been implemented.

The joint-space action consists of a single value for each of the twelve leg joints. To avoid problems due to joint limits, the values get scaled so that -1 and 1 represent the lower and upper bound of the joint, respectively. In addition, the initial bias of the policy's output layer is set to the joint values in its *walkready* position. Thus, at the beginning of the training, the Gaussian distributions in the output layer of the policy network are centered around these initial biases, resulting in exploration around a stable position, and a lower number of falls during the early phases of the training.

The cartesian-space action consists of six values for each foot, describing its position and orientation relative to the base link. To avoid poses that cannot be reached by the

robot, the action space is reduced to a cuboid around the robot's feet for the position and approximately -60 to 60 degrees for the orientation values. This choice of action space results in a similar default position as in the joint-space action when the initial bias is set.

4.3.5. Rewards

In deep reinforcement learning, the reward function is a very important part, as it describes which parts of the agent's behavior are encouraged and which behaviors the agent should not follow. In the approach to learning from demonstration that was taken in this thesis, the reward consists of two parts: the imitation reward and the task reward.

$$r = \omega_I r_I + \omega_T r_T$$

The imitation reward r_I encourages an action that is close to the demonstration, the task reward r_T rewards a good execution of the task. Both parts of the reward are weighted with $\omega_I = 0.7$ and $\omega_T = 0.3$, respectively. These weights and the imitation reward described below were chosen to be the same as in DeepMimic because they achieved excellent results using these rewards in simulation and in the real world.

The imitation reward is composed of four partial rewards:

$$r_I = \omega_R r_R + \omega_E r_E + \omega_P r_P + \omega_V r_V.$$

r_R is the root position reward, r_E is the end effector position reward, r_P is the joint position reward, and r_V is the joint velocity reward. The rewards are weighted with $\omega_R = 0.1$, $\omega_E = 0.15$, $\omega_P = 0.65$, and $\omega_V = 0.1$, respectively. Thus, the joint position reward is the most important of these rewards, by a large margin.

It is calculated in the following way:

$$r_P = \exp\left(-2 \cdot \sum_{j \in J} \|p_j - \hat{p}_j\|_2^2\right)$$

where J is the set of joints and p_j is their respective position. $\hat{p}_j$ denotes the joint's position in the reference motion. The joint velocity reward is calculated similarly, but the factor before the sum is -0.1 instead of -2.

The root position r_R rewards the proximity of the robot's root link to the root link of the reference trajectory. It is calculated in the following way:

$$r_R = \exp\left(-10 \cdot \|R - \hat{R}\|_2^2\right)$$

where R is the agent's root position and $\hat{R}$ is the root position in the reference motion. This reward slightly deviates from the reward in [Pen+18], where the position of the agent's center of mass is used instead. In the Wolfgang robot, the center of mass is normally located very close to the root link; therefore, the position of the root link is a good approximation for the position of the center of mass. Additionally, the reward function in [Pen+20] also uses use the root link instead of the center of mass.

In a similar way to the root link reward, the end effector reward r_E is calculated:

$$r_E = \exp\left(-40 \cdot \sum_{e \in E} \|p_e - \hat{p}_e\|_2^2\right).$$

Here, the squared positional error of the end effectors, i. e. the left and right foot, is used.

Each of the partial rewards of the imitation rewards is scaled exponentially. A perfect resemblance to the demonstration results in a reward of 1. The more the state deviates from the demonstration, the closer the rewards get to 0.

For the task reward, a function that rewards strong kicks has to be chosen. A simple approach to rewarding a kick would be to reward the distance the ball traveled since the beginning of the episode after each timestep. However, this approach would not be well-suited because actions after the kick time would be rewarded according to the kick that happened before, even though they have no way of influencing the kick's strength. If the reward were only given once at the very end of the episode, it would have been extremely difficult to link this reward back to the action responsible for actually moving the ball. Therefore, the reward has to be given directly for the actions that achieve the ball movement. To discourage movements of the ball when they are not desired, for example while lifting the foot, the reward should only be given for a specific time window where the demonstration performs the kick and the ball is moving. To summarize, the task reward has to be a function that rewards a strong movement of the ball at the correct time.

To fulfill these conditions, the following function has been chosen:

$$r_T = \begin{cases} 1 - \exp\left(-2 \cdot v_B\right) & \text{if } t_k \leq t \leq t_k + 0.5 \\ 0 & \text{else} \end{cases}$$

v_B is the ball velocity at the current time t. t_k is the time of the kick in the demonstration. The condition ensures that ball movement before the actual kick, for example while lifting the foot, is not rewarded. The reward is scaled exponentially such that no ball movement results in a reward of 0, and the higher the ball velocity gets, the closer the reward gets to 1.

4.4. Implementation

For reinforcement learning, the Stable Baselines 3 framework [Raf+19] is used. To describe the reinforcement learning problem, an environment implementing the OpenAI Gym environment interface [Bro+16] was created. This interface is used by the framework for interacting with the environment and provides a common wrapper interface for different reinforcement learning problems. The advantage of using this interface is that it is agnostic of the framework and its implementation details. Therefore, the framework or the used learning algorithm could easily be exchanged without changing the implementation of the environment.

The interface mainly consists of two functions: `reset` and `step`. `reset` is called at the beginning of each episode. Its purpose is to reset the environment to an initial state. In the context of this thesis, the robot is initialized at a random position of the demonstration (reference state initialization) and the ball is placed accordingly. The method returns the observation at the beginning of the episode. The `step` method is called for each step of the policy. It receives an action from the policy and applies it to the agent, in this case by moving the feet of the robot accordingly. The simulation is then advanced and a new observation and the reward are calculated. If the fixed time horizon of three seconds is exceeded or the early termination condition is fulfilled, the episode is terminated. The `step` method returns a new observation, a reward, whether the episode is finished, and further custom information. For each step, the current state, actions, and rewards are additionally published as ROS messages to facilitate debugging and visualization of the information. At the end of an episode, the episode reward is also logged.

To make the different states easily interchangeable, they have been implemented as classes extending a common superclass. For the reward, a `WeightedCombinedReward` is used that contains a list of rewards and their corresponding weights. The different partial rewards described above can thus easily be combined and weighted in different ways. Figure 4.5 shows the class hierarchy that is used.

The kick that is used for demonstration could be controlled via ROS messages. However, this would include an asynchronous execution of the kick and the learning process. Therefore, it would not be possible to perform only one step of the demonstration for each timestep of the training. Instead, the Python kick wrapper implemented for the parameter optimization (Section 4.1) is used. It provides methods to reset the kick and to step forward for a given timestep, as well as methods that provide information about the current state of the kick, for example its progress or the current trunk pose.

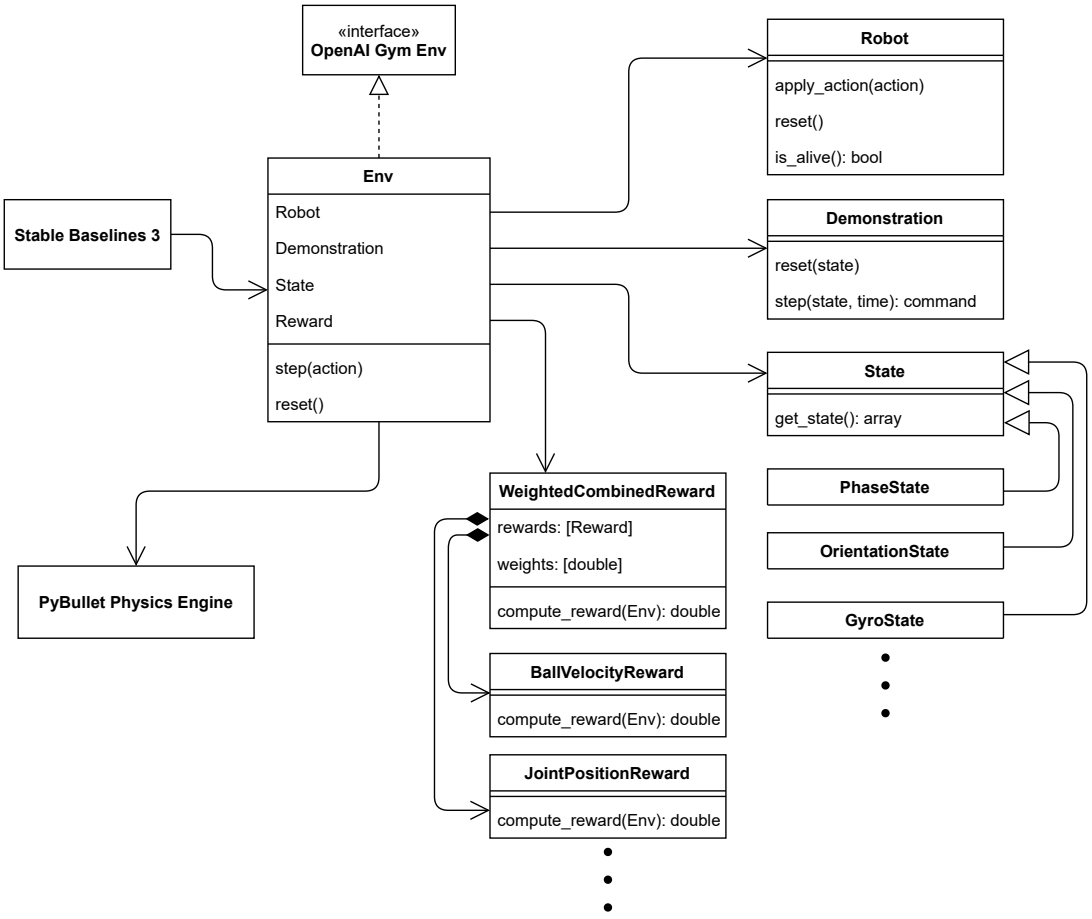


Figure 4.5.: Simplified UML class diagram showing the classes used in the learning setup. The main class is the `Env` class that implements the OpenAI Gym interface and is directly used by Stable Baselines 3. It controls the simulator and uses different other classes to apply an action to the robot, get information from the demonstration, get the current state, or calculate the current reward. The state class is implemented by the different classes used in this thesis, and some examples are shown in the diagram. The reward class is composed of different other rewards that are weighted and combined to form the final reward.

5. Experiments

To evaluate the setup and implementation described above, and to get a better understanding of the underlying mechanics of the learning process, a series of experiments is conducted. First, the training is run on different observation and action spaces. The setup is described and the comparison and evaluation of these results is done in Section 5.1. The results give an insight into which input and output data makes learning the motion easier or harder and can presumably also be applied to other similar problems. To analyze the stability of the obtained motions, another experiment is conducted on the three best-performing kicks from the previous experiments. In this experiment, the robot is randomly pushed during the kick and the number of falls is counted. Further description and evaluation of this experiment can be found in Section 5.2.

5.1. Observation and Action Spaces

5.1.1. Setup

The first set of experiments in this thesis is an ablation study on the observation space for the policy. An ablation study is a process to determine "the contribution of individual components to the performance of complex systems by removing [...] these components" [CH88]. In this case, the individual components are different parts of the observation spaces, e. g. the pressure sensor readings or the angular velocities of the robot. Nine different observation spaces are compared, ranging from the *ComprehensiveState*, which contains all information available to the robot, to the *PhaseState* containing no sensor information. The purpose of the ablation study is to find which additional input is particularly relevant for the policy to achieve a stable and precise kick, which input gives no further advantage, and which input might even have a harmful effect on the policy.

In addition, each of the nine different observation spaces introduced in Subsection 4.3.3 is combined with both of the two action spaces. This line of experiments will be used to determine whether a cartesian-space output improves the sample efficiency of the training. This might be the case because the cartesian-space representation might be more intuitive to understand than the joint-space representation, where more complex connections between the joint positions and the resulting foot movement exist.

For each of these experiments, the policy is trained for ten million timesteps. This corresponds to approximately 150 000 episodes, i. e. 150 000 attempts to kick. The reward function for all experiments is the same, as described in Subsection 4.3.5, the hyperparameters can be found in Table 4.1.

5.1.2. Evaluation

Evaluation Criteria

The learned kicks presented in this thesis are compared using five different evaluation criteria: the time to stability, the kicked distance, how often the robot fell, the number of timesteps required during the learning process, and the visual appearance of the kick.

The time to stability describes the time until the robot reaches a stable position after the kick, i. e. is not moving around a lot. This is the last time where any of the angular velocities of the robot are larger than 0.1 rad/s. This criterion is used because a jittery or unbalanced behavior would result in this time being higher. The ball distance is the distance the ball was kicked, measured from the starting position of the ball to its final position, in meters. The number of falls of the robot simply measures in how many of the robot's attempts to kick, out of 10 total, the robot fell down. The number of timesteps required during the learning process describes the timestep where the policy first reached a reward of 35. From the data available from the experiments, this reward seems to correspond to a solid kick. While the kick can still be improved after this point, the time where this reward is reached gives a good indication of how fast the policy learns to perform a usable kick. Finally, each kick is evaluated visually. This criterion was added to address problems typically faced in reinforcement learning, like motions that appear unhuman or do not resemble a kick movement at all.

Evaluation Results

The results of the experiments on observation and action spaces are shown in Table 5.1. The table shows that overall, stable kicks where the robot does not fall could only be obtained by some of the state-action combinations, as can be seen from fall rates of 0% and kick distances similar to the demonstration. The evaluation graphs used to determine the number of timesteps for each of the experiments are shown in Appendix B.

The first two rows of the table show the *PhaseState*, where no sensor input is used (open loop). For these kicks, no actual reaction to the environment is learned. Instead, one singular motion is learned by heart. Still, the *PhaseState* achieves relatively good results, especially with the *JointAction*, where the ball distance is even higher than for the reference motion, and the robot never falls. The *OrientationState*, where the roll and

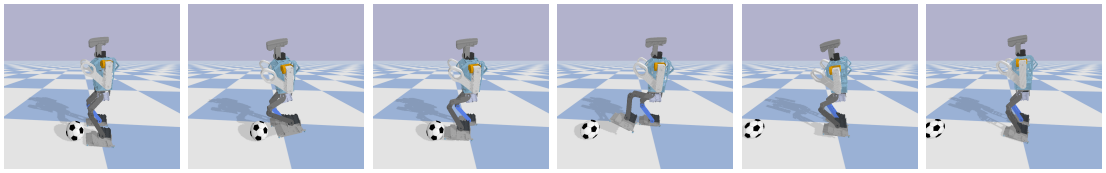


Figure 5.1.: Sequence of the robot during the kick with *PhaseState* and *JointAction*

pitch from the IMU are included in the state, also reaches very good results. The robot never falls and the ball distance is high, especially with the *CartesianAction*. The number of timesteps required until a reward of 35 is reached is also lower than for the *PhaseState*. For this state, the cartesian action works better than the joint action, as can be seen from the ball distance and visual examination. The *GyroState*, where the angular velocities are also included, also leads to kicks where the robot does not fall down. Again, the visual examination and the ball distance show that the *CartesianAction* results in a better kick. The reward threshold of 35 is reached for the *CartesianAction* after two million timesteps which is significantly later than for the *OrientationState*. The *JointAction* for the *GyroState* never reaches the threshold.

The following two states contain information about the foot position or velocity instead of the IMU data. The results for these states are not very good; both states result in a high number of falls and no or barely a visual resemblance to the kick. That means that the robot either does not lift a foot at all or lifts a foot but fails to move it forward in a kicking way.

The *OrientationFootState*, again including roll and pitch derived from the IMU, results in a kick that managed to remain standing. However, the visual analysis showed that the executed motion does not resemble a kick.

The *PressureState*, including the pressure sensor information in addition to the kick's phase, results in motions that are neither stable nor visually resemble a kick. When the foot information is also included, the motions, while becoming more stable, do not improve significantly.

Finally, the *ComprehensiveState* containing all of the above-mentioned information results in a stable kick, and the kick learned with the cartesian action even resembles a kick visually. Still, the performance is much worse than for some of the smaller states described above.

It can also be seen that the time to stability for almost all of the kicks is near or at the time limit of the episode. This means that most kicks never reach a point where they keep standing perfectly still at the end of the episode. Instead, they keep slightly moving their joints around after the kick is finished.

State	Action	Time to Stability	Distance	Fallen	Timesteps ($r = 35$)	Visual
PhaseState	Cartesian	**2.6**	0.246	40%	2 210 000	✓
PhaseState	Joint	**1.4**	**0.358**	**0%**	1 050 000	✓
OrientationState	Cartesian	**2.127**	**0.331**	**0%**	790 000	✓
OrientationState	Joint	3.0	0.199	**0%**	930 000	∘
GyroState	Cartesian	3.0	**0.303**	**0%**	2 000 000	✓
GyroState	Joint	3.0	0.199	**0%**	–	∘
FootState	Cartesian	3.0	0.245	70%	–	∘
FootState	Joint	–	0.213	100%	–	✗
FootVelocityState	Cartesian	–	0.006	100%	2 810 000	✗
FootVelocityState	Joint	3.0	0.03	10%	–	✗
OrientationFootState	Cartesian	2.997	0.161	**0%**	870 000	∘
OrientationFootState	Joint	3.0	0.225	**0%**	–	✗
PressureState	Cartesian	–	0.289	100%	–	✗
PressureState	Joint	–	0.002	100%	–	✗
PressureFootState	Cartesian	3.0	0.181	40%	–	✗
PressureFootState	Joint	3.0	0.067	80%	–	✗
Comprehensive	Cartesian	2.99	0.231	**0%**	1 830 000	✓
Comprehensive	Joint	3.0	0.062	**0%**	–	✗
Demonstration		1.433	0.327	0%		

Table 5.1.: Results of the experiments on observation and action spaces. Time to stability is measured in seconds, distance is measured in meters. The percentages of falls are obtained from ten trials. For the visual examination, ✓ corresponds to a well-executed kick, ∘ represents a motion that resembles a kick in some way, for example because a leg is raised, but does not perform a well-defined kick motion. ✗ corresponds to movements that do not resemble a kick at all. For the first three evaluation criteria, the best three results are marked in bold.

To summarize the findings, the best results are obtained with the *PhaseState*, *OrientationState*, and *GyroState*. Notably, none of the states where the robot falls down contain any information about the orientation of the robot, and all states containing this information result in stable kicks. In particular, the *FootState*, *FootVelocityState*, *PressureState*, and *PressureFootState* do not result in stable kicks and also do not pass the visual examination. The *PhaseState* with *JointAction* results in the overall best kick, with the best time to stability by a large margin and the best ball distance.

The visual appearance of the kick seem to be better for cartesian actions than for joint actions. However, the data is not sufficient to conclude that cartesian actions result in better kicks. With a larger sample size, though, this difference might become significant.

5.2. Stability

5.2.1. Setup

In the next set of experiments, the stability of different setups from Section 5.1 is evaluated. This is necessary because in real-world applications, different external forces are exerted to the robot. These can be due to environmental conditions, for example uneven ground or pushes from other robots, or internal reasons like wear or backlash of motors. These conditions are not well modeled in the simulation. Therefore, a policy that performs well in the setup described in Section 5.1 might perform significantly worse under more imperfect conditions, for example because no sensor input is used.

To simulate external forces that the policy should be robust against, at a random time during the kick, a force is exerted to the robot from a random direction for a third of a second. While this model does not perfectly represent systematic deviations on the robot, e. g. due to motor wear, it models robustness against sudden shoves and should also reward overall robustness in other cases. During the evaluation, the exerted force is gradually increased from 0N to 250N in steps of 10N. For each force strength, the kick is run 50 times.

5.2.2. Evaluation

The results of the stability analysis are shown in Figure 5.2. The resistance test has been run for the three best kicks from the previous evaluation, that is the *PhaseState* with *JointAction*, the *OrientiationState* with *CartesianAction*, and the *GyroState* with *CartesianAction*. On the y-axis, the percentage of stable kicks, i. e. kicks where the robot did not fall down, is displayed. On the x-axis, the force exerted to the robot is shown, as described in Section 5.2. The analysis clearly shows that while all kicks were very stable

for forces up to 50N, the stability starts to differ for larger forces. For forces between 50N and 150N, the states containing IMU information (i. e. *GyroState* and *OrientationState*) are clearly more stable than the state without this information (*PhaseState*). For forces of more than 150N, all approaches show similar rates of stable kicks.

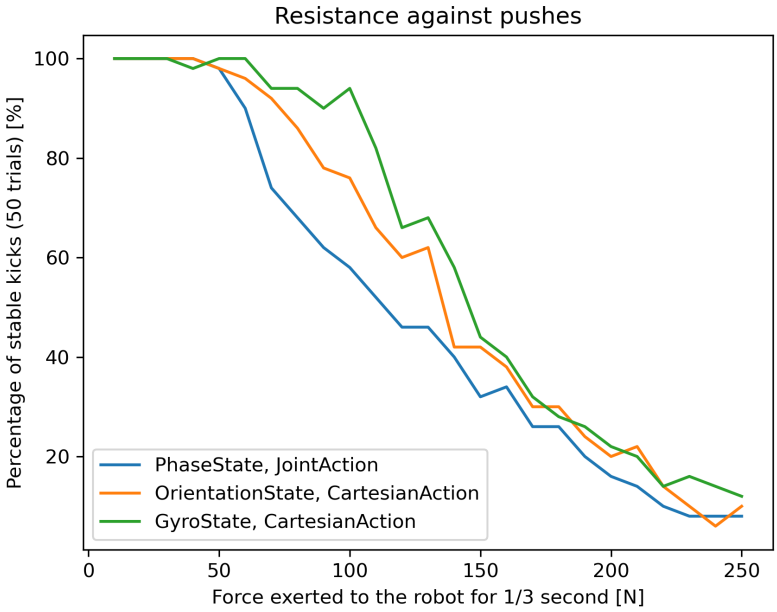


Figure 5.2.: Resistance against pushes. On the x-axis, the exerted force is shown, the y-axis shows how often the robot stayed stable (higher is better). For each kick and force combination, the experiment was run 50 times.

6. Discussion

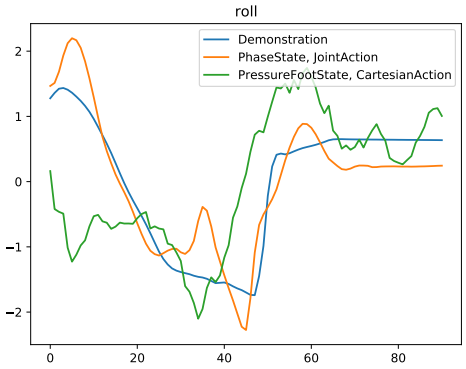
In general, the work in this thesis shows good results for the learned kick, as described in the previous chapter. This chapter focuses on discussing the results of the experiments and the general setup used in the thesis. In Section 6.1, the findings from the experiments are discussed. Section 6.2 outlines possible flaws of the setup and threats to the validity of the findings.

6.1. Experiments

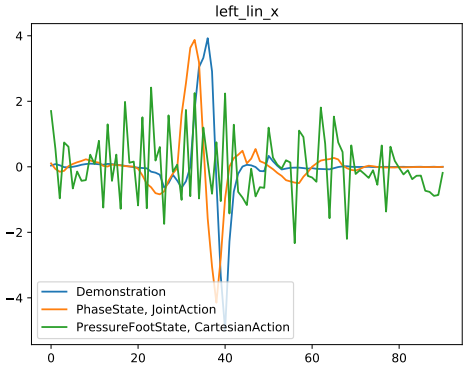
The evaluation of observation spaces revealed that the *PhaseState*, i. e. the state containing no sensor information, performed best. At first glance, that is strange because this policy received the least input. The results can however be explained with the environment of the agent, which is mainly deterministic. Therefore, a policy that simply learns one singular action without reacting to its environment can perform surprisingly well under these circumstances. On the real robot, this simple policy will likely not work as well because the robot's environment is much less deterministic, and external influences like instabilities on the ground or backlash of the motors are added to the system. The stability analysis supports this hypothesis since the *PhaseState* was much less stable than the states containing IMU information.

The high time to stability that was observed for almost all of the kicks, where the experiments showed that they never reached perfect stability after the kick, is most likely due to noise in the network input that the policy could not filter out completely. That would explain the high time to stability for all of the kicks containing sensor data. For the kicks using *CartesianAction*, an additional source of noise is the inverse kinematics, which is not always perfectly deterministic. Both of these problems could probably be solved by using a low-pass filter on the joint positions, so that the high-frequency noise is filtered.

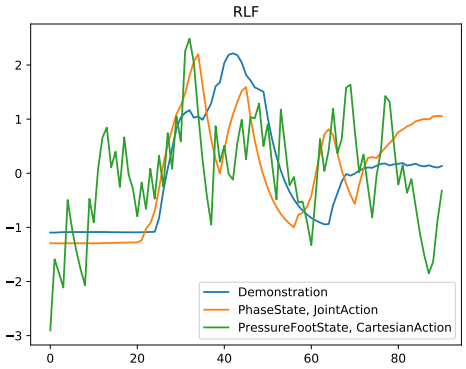
The evaluation also showed that some states showed significantly worse behavior than the simple *PhaseState*, even though the phase was also included in these states. In theory, the neural network should have been able to learn to ignore the useless input data and use only the phase for the motion, as was the case for the *PhaseState*. However, this did not happen. The reason for this is probably that, in this case, the phase is only a very small part of the input. In the *PressureState*, for example, the pressure information



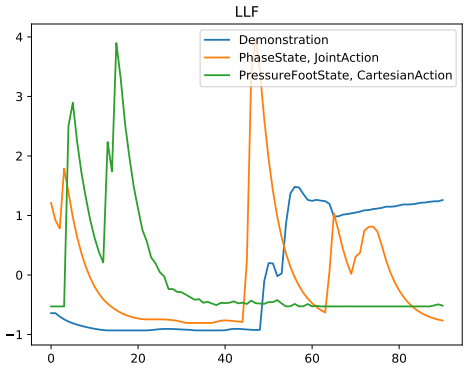
(a) Roll of the robot



(b) Velocity of the left foot along the x-axis



(c) Front left pressure sensor of the right foot



(d) Front left pressure sensor of the left foot

Figure 6.1.: Values of different sensor values during different kicks. The values have been standardized to have a mean of zero and a standard deviation of 1, like it is done by Stable Baselines 3 for the observation input. The x-axis measures the elapsed time since the beginning of the episode in seconds.

uses eight input neurons while the phase is only a single one. When the data from the pressure sensors is not usable, this makes it difficult to ignore this significant part of the input. In addition, the pressure sensor and foot velocity information are very noisy and can show large value jumps. This makes it more difficult to ignore these values as either the weights or the biases for some neurons have to be reduced drastically. If this happens for many neurons, the usable network size effectively becomes smaller, thus making it more difficult to learn the actual motion. The high noise and jumps are clearly visible in Figure 6.1. In the figure, different sensor values are compared for three different kicks: the demonstration, the *PhaseState* with *JointAction*, which performed well, and the *PressureFootState* with *CartesianAction*, which performed poorly. For the roll, which worked well when used as input, the curve is rather smooth for all of the displayed kicks. In contrast, for the velocity of the left foot along the x-axis, it is clearly visible that the

values are very noisy for the kick that performed poorly, while the curve is rather smooth with high jumps for the well-performing kicks. For the values of a pressure sensor on the right foot, a similar effect can be observed. In Figure 6.1d, for a pressure sensor of the left foot, the values are much less noisy, but high jumps in the values can be seen. Since the graphs show the sensor values during the evaluation and not during the training, they do not perfectly represent the actual input of the neural network during the training. However, since a fixed variance is used on the output layer of the network during the training, the values are likely to become even more noisy during the training. This effect especially affects the sensors that are located close to the feet as the foot positions are controlled by the network's output. Inputs derived from the IMU, which is located higher in the robot, are less affected by noisy network output, further explaining why their data improved the kick while the other sensor data did not.

Another observation from the experiments is that the ball kick distance of less than 40cm is very small. The reason for these low values is that the friction values in PyBullet are not configured correctly. Therefore, the ball is more difficult to push over the ground and quickly loses its momentum. Unfortunately, a correct model of the frictions could not be created since no PyBullet configuration that resulted in the desired behavior could be found. Modeling the rolling frictions so that the ball rolls on the ground was not successful, and changing the frictions to make the ball slide further also resulted in less friction between the robot's feet and the ground, changing the dynamics of the kick. Therefore, all frictions were kept as their default values. In reality, the demonstration kick reaches a distance of slightly more than a meter. It can therefore be assumed that the best of the learned kicks reach similar ball distances.

6.2. Critical Discussion

In the thesis, the training for each of the state-action combinations has only been run a single time. Running them multiple times will increase the validity of the acquired data because, due to the randomness included in the reinforcement learning process, some results might be better or worse than for a single run. However, the results of this thesis will most likely not be affected because, for different states, a high correlation between similar input information can be found. For example, most states containing pressure sensor information performed poorly. Thus, the deduction that this information is not learned well is natural, even though no further training runs have been done for the exact same state-action combinations.

Additionally, the hyperparameters were not optimized for the separate state-action combinations. Running hyperparameter optimization is computationally expensive because a lot of different parameter sets have to be tried and evaluated and is therefore not feasible for a high number of different setups. In related work, the same hyperparameters are often used for different but similar problems (c. f. [Pen+20]), and in general,

PPO is relatively robust to hyperparameter changes. Therefore, while a hyperparameter optimization would certainly improve the outcome, especially with regards to sample efficiency, the results of this thesis are not fundamentally flawed, and the usage of the same hyperparameters reflects common practices.

Also, the architecture of the neural network could be adapted for better results. The fact that larger states were not as successful as smaller states might show that the neural network size is too small for the problem. However, a network size of 512x512 layers is frequently used in reinforcement learning.

Another problem of the setup in this thesis might be the task reward function. Currently, it only rewards a small part of the motion, the part from the kick time until half a second later. This produces a sparse reward situation, where the reward is oftentimes difficult or impossible to reach. A possible improvement of the reward function might be to include the distance between the kicking foot and the ball, so that the agent is guided towards moving the foot to the ball. However, this form of reward could result in a motion that does not kick the ball and instead just moves the foot near the ball and slowly pushes it forward. Hence, careful reward shaping would have to be done, or other approaches, like learning the reward function [Chr+17] could be investigated. Additionally, the kick direction of the ball could be included into the reward function to avoid obtaining kicks that kick well but at an undesired angle. Regarding the imitation reward, the currently used partial rewards could not be optimal, as neither these rewards nor their weights have been evaluated.

A general problem of learning from demonstration is also that the type of motion that can be learned is limited by the demonstration. Thus, a fundamentally different motion that solves the task better would never be found in the training. However, this drawback is also an advantage because no motions that exploit the simulation or would not be applicable to the real world can be found, either.

7. Conclusion

In this thesis, an approach for learning motions from demonstration with deep reinforcement learning was implemented. It was used to learn to kick, which is valuable in the RoboCup soccer domain. Different observation and action spaces were compared and evaluated. Additionally, the stability of the best-performing kicks was compared.

Overall, the work in this thesis showed that good results for learning a kick motion with reinforcement learning could be achieved. The excellent performance of the open-loop approach shows that a straightforward imitation learning setup without any sensor input can already improve existing solutions and perform comparably well in simulation. Adding sensor input, especially information about the robot's orientation, drastically improves its stability while still achieving a performance advantage compared to the demonstration. Using noisy sensor data, for example foot pressure sensor readings or foot velocities, resulted in particularly poor performances. The comparison of cartesian and joint action spaces showed promising results for cartesian-space actions. These results should be further investigated to show if they are actually significant.

In future work, the results of this thesis could be transferred to the real world. To achieve a working kick in the real world, techniques such as a low-pass filter for the actions and domain adaptation could be used. A hyperparameter optimization could be performed for the best-performing state-action combinations to improve the performance of the learning process. A performance comparison for cartesian and joint actions could also be made. This analysis could be similar to [PP17], where different action spaces are compared on different robots and for different problems, as well as their resistance to random perturbations and irregular terrain.

In the thesis, only a single kick command (only one ball position and kick direction) has been learned. In future work, this could be extended to different ball positions and kick directions, as well as kicking with different feet. For a single neural network, learning the different kicks with different feet is rather difficult because the motion greatly varies between kicks, depending on the selected foot, the direction of the kick, and the kicking side of the foot. To achieve a good generalization, hierarchical approaches could be investigated instead, with one neural network deciding which foot is used for kicking, and another neural network then performing the kick, or different networks even executing different parts of the kick.

Appendix

A. Kick Engine Parameters

The parameter optimization was run with the following objectives:

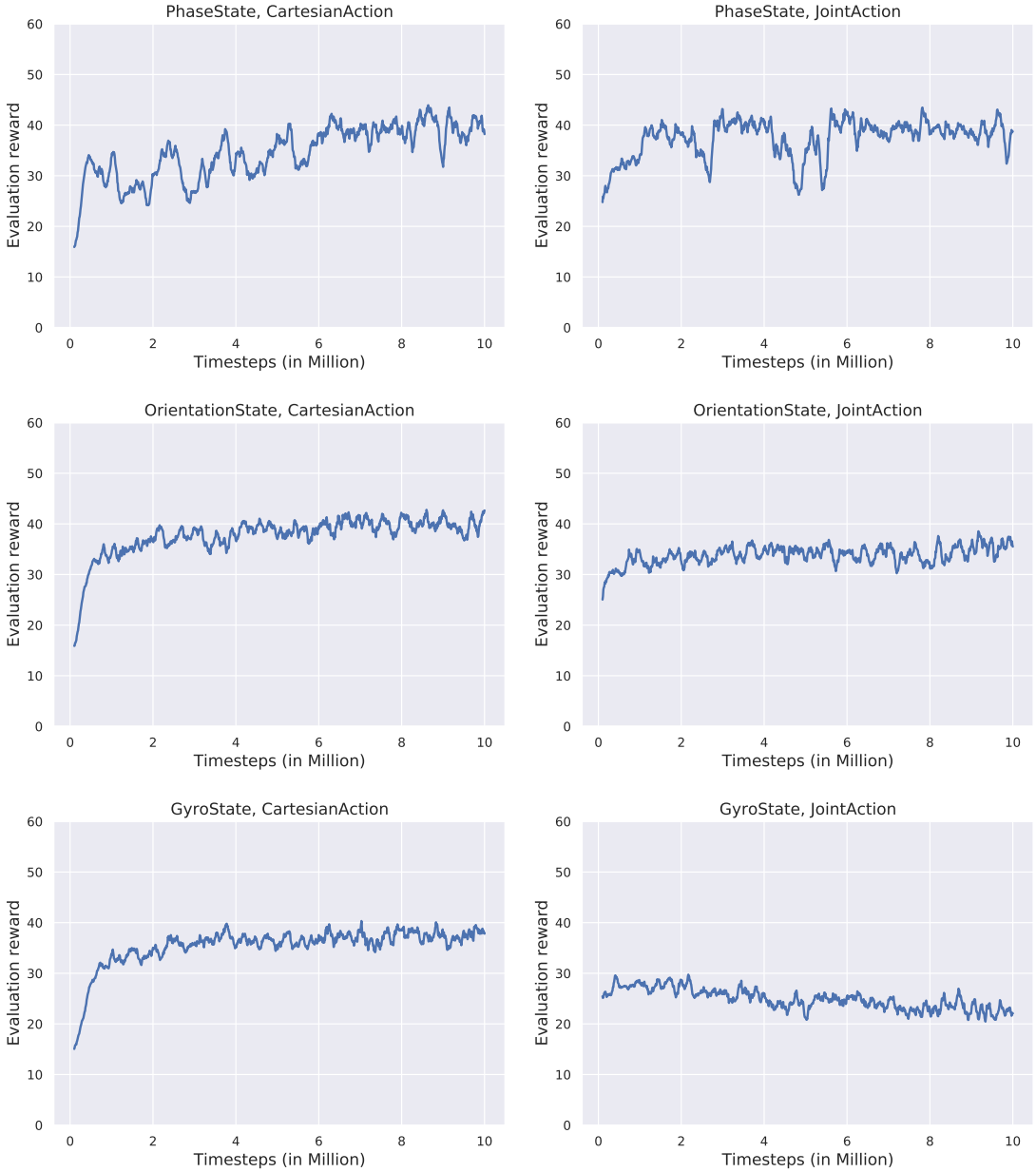
1. whether the robot fell during the kick (0 if it fell, 1 else)
2. the required timesteps for one kick
3. the maximal ball velocity during the kick
4. the angle between the requested kick direction and the actual kick direction

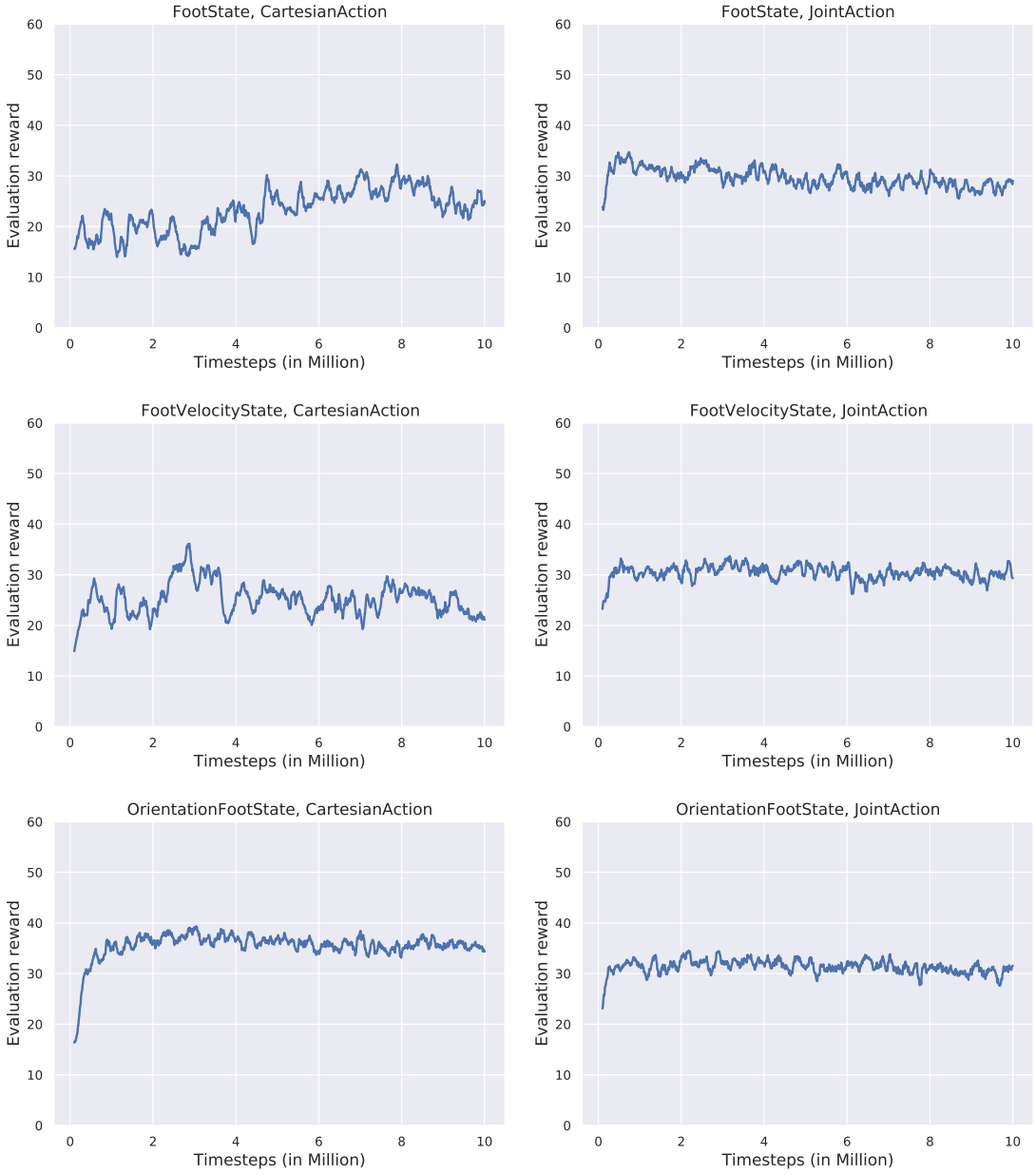
Objectives 1, 2, and 4 were minimized, objective 3 was maximized.

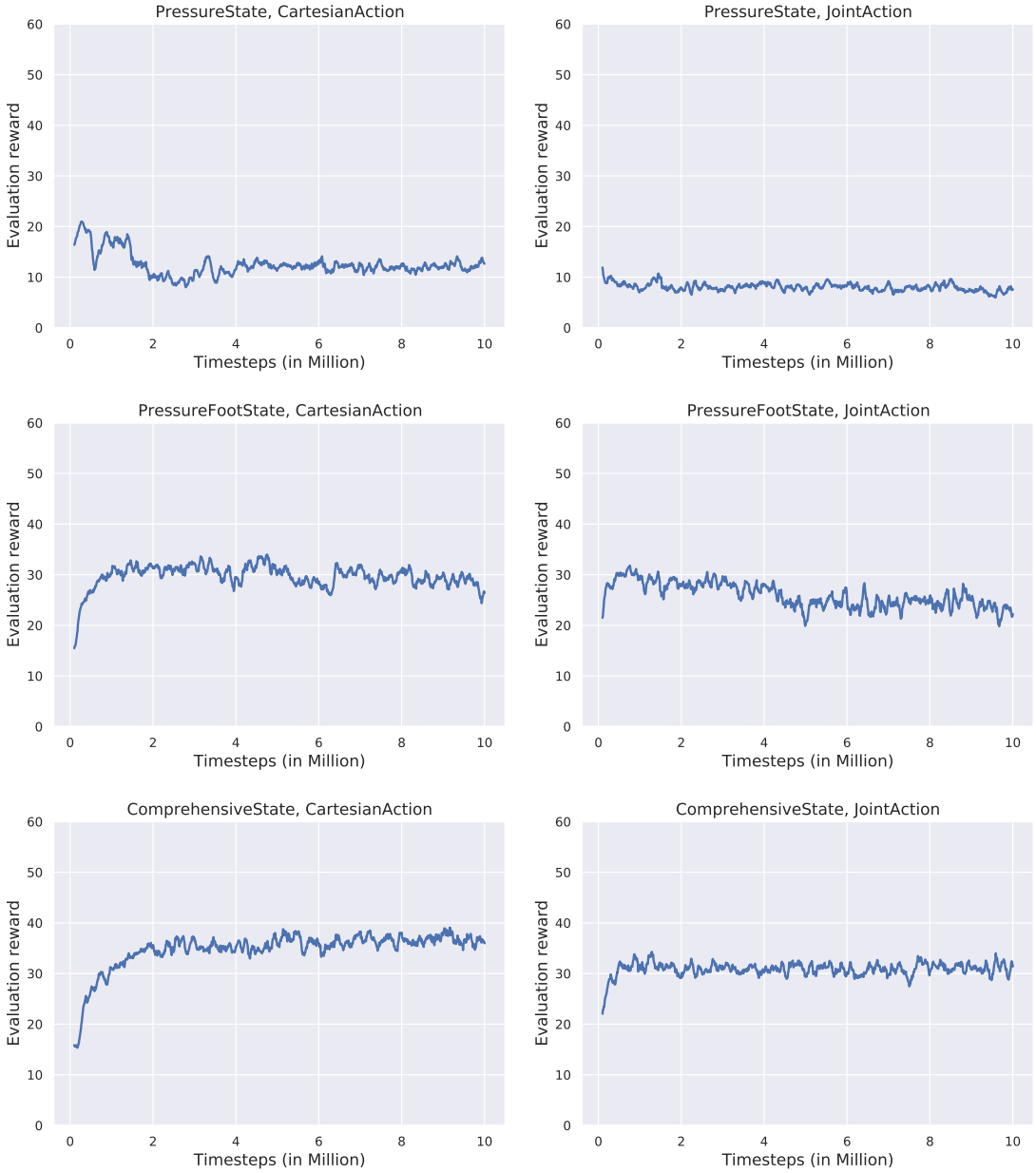
The resulting kick engine parameters were:

Foot Rise	0.08
Foot Distance	0.17
Windup Distance	0.23
Trunk Height	0.36
Trunk Roll	-0.208
Trunk Pitch	0.134
Trunk Yaw	-0.007
Move Trunk Time	0.64
Raise Foot Time	0.24
Move to Ball Time	0.13
Kick Time	0.16
Move Back Time	0.15
Lower Foot Time	0.2
Move Trunk Time	0.12
Stabilizing Point X	-0.02
Stabilizing Point Y	0.01

B. Evaluation Graphs




FootState, CartesianAction
FootState, JointAction
FootVelocityState, CartesianAction
FootVelocityState, JointAction
OrientationFootState, CartesianAction
OrientationFootState, JointAction
Evaluation reward
Timesteps (in Million)
0
10
20
30
40
50
60
0
2
4
6
8
10


PressureState, CartesianAction
PressureState, JointAction
PressureFootState, CartesianAction
PressureFootState, JointAction
ComprehensiveState, CartesianAction
ComprehensiveState, JointAction
Evaluation reward
Timesteps (in Million)
0
10
20
30
40
50
60
0
2
4
6
8
10

Bibliography

Literature

[Abr+19] M. Abreu et al. "Learning low level skills from scratch for humanoid robot soccer using deep reinforcement learning". In: *2019 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*. 2019, pp. 1–8. DOI: 10.1109/ICARSC.2019.8733632.

[Aki+19] Takuya Akiba et al. "Optuna: A next-generation hyperparameter optimization framework". In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 2623–2631.

[All+16] J. Allali et al. *Rhoban Football Club – Team Description Paper*. Tech. rep. 2016.

[ARL19] Miguel Abreu, Luis Paulo Reis, and Nuno Lau. "Learning to run faster in a humanoid robot soccer environment through reinforcement learning". In: *RoboCup 2019: Robot World Cup XXIII*. Springer, 2019, pp. 3–15.

[Ber+11] James Bergstra et al. "Algorithms for hyper-parameter optimization". In: *25th annual conference on neural information processing systems (NIPS 2011)*. Vol. 24. Neural Information Processing Systems Foundation. 2011.

[Bes+18] Marc Bestmann et al. *Hamburg Bit-Bots and WF Wolves Team Description for RoboCup 2019 Humanoid TeenSize*. Tech. rep. 2018.

[Bes+21] Marc Bestmann et al. "Wolfgang-OP: A Robust Humanoid Robot Platform for Research and Competitions". Accepted for IEEE-RAS International Conference on Humanoid Robots. 2021.

[BES19] Frederico Bormann, Timon Engelke, and Finn-Thorben Sell. "Developing a Reactive and Dynamic Kicking Engine for Humanoid Robots". 2019.

[BGZ19] Marc Bestmann, Jasper Güldenstein, and Jianwei Zhang. "High-frequency multi bus servo and sensor communication using the Dynamixel protocol". In: *RoboCup 2019: Robot World Cup XXIII*. Springer, 2019, pp. 16–29.

[Bro+16] Greg Brockman et al. *OpenAI Gym*. 2016. arXiv: 1606.01540 [cs.LG].

[BS95] Michael Bain and Claude Sammut. "A Framework for Behavioural Cloning." In: *Machine Intelligence 15*. 1995, pp. 103–129.

[But+30] Stephen Butterworth et al. "On the theory of filter amplifiers". In: *Wireless Engineer* 7.6 (1930), pp. 536–541.

[CH88] Paul R. Cohen and Adele E. Howe. "How Evaluation Guides AI Research: The Message Still Counts More than the Medium". In: *AI Magazine* 9.4 (Dec. 1988), p. 35. DOI: 10.1609/aimag.v9i4.952.

[Chr+17] Paul Christiano et al. "Deep reinforcement learning from human preferences". In: (June 2017). arXiv: 1706.03741 [stat.ML].

[Com20] RoboCup Humanoid Committee. *RoboCup Soccer: Humanoid League – Roadmap from 2020 to 2050*. Jan. 2020. URL: https://humanoid.robocup.org/wp-content/uploads/roadmap_draft2020v1.pdf.

[Com21] RoboCup Humanoid Technical Committee. *Virtual RoboCup Soccer Humanoid League Laws of the Game 2020/2021*. Apr. 2021. URL: https://cdn.robocup.org/hl/wp/2021/04/V-HL21_Rules_v2.pdf.

[Dua+16] Yan Duan et al. "Benchmarking deep reinforcement learning for continuous control". In: *International conference on machine learning*. PMLR. 2016, pp. 1329–1338.

[Hee+17] Nicolas Heess et al. *Emergence of Locomotion Behaviours in Rich Environments*. July 2017. arXiv: 1707.02286 [cs.AI].

[HL95] Walter L. Hürsch and Cristina Videira Lopes. *Separation of Concerns*. Tech. rep. 1995.

[JC20] Xin Jing and Xinxin Chen. *Extended Abstract of Team ZJUDancer*. Tech. rep. 2020.

[KA98] Hiroaki Kitano and Minoru Asada. "The RoboCup humanoid challenge as the millennium challenge for advanced robotics". In: *Advanced Robotics* 13.8 (1998), pp. 723–736.

[Kak01] Sham M Kakade. "A natural policy gradient". In: *Advances in neural information processing systems* 14 (2001).

[KBP13] Jens Kober, J Andrew Bagnell, and Jan Peters. "Reinforcement learning in robotics: A survey". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274.

[Li+21] Zhongyu Li et al. *Reinforcement Learning for Robust Parameterized Locomotion Control of Bipedal Robots*. 2021. arXiv: 2103.14295 [cs.RO].

[MLR11] Judith Müller, Tim Laue, and Thomas Röfer. "Kicking a Ball – Modeling Complex Dynamic Motions for Humanoid Robots". In: *RoboCup 2010: Robot Soccer World Cup XIV*. Springer, 2011, pp. 109–120.

[Oza+20] Yoshihiko Ozaki et al. "Multiobjective Tree-Structured Parzen Estimator for Computationally Expensive Optimization Problems". In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. Association for Computing Machinery, 2020, pp. 533–541. DOI: 10.1145/3377930.3389817.

[Pen+18] Xue Bin Peng et al. "DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills". In: *ACM Transactions on Graphics (TOG)* 37.4 (Apr. 8, 2018), pp. 1–14. arXiv: 1804.02717v3 [cs.GR].

[Pen+20] Xue Bin Peng et al. *Learning Agile Robotic Locomotion Skills by Imitating Animals*. Apr. 2, 2020. arXiv: 2004.00784v3 [cs.RO].

[PP17] Xue Bin Peng and Michiel van de Panne. "Learning locomotion skills using DeepRL". In: *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. ACM, July 2017. DOI: 10.1145/3099564.3099567.

[PV19] Pedro Peña and Ubbo Visser. "Adaptive Walk-Kick on a Bipedal Robot". In: *RoboCup 2019: Robot World Cup XXIII*. Springer, 2019, pp. 213–226.

[Qui+09] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, p. 5.

[Raf+19] Antonin Raffin et al. *Stable Baselines3*. https://github.com/DLR-RM/stable-baselines3. 2019.

[RB10] Stephane Ross and Drew Bagnell. "Efficient Reductions for Imitation Learning". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Vol. 9. Proceedings of Machine Learning Research. PMLR, May 2010, pp. 661–668.

[Riz+19] Aulia Khilmi Rizgi et al. *EROS - Team Description Paper for Humanoid KidSize League, RoboCup 2019*. Tech. rep. 2019.

[Rup17] Philipp Ruppel. "Performance optimization and implementation of evolutionary inverse kinematics in ROS". Master's Thesis. Universität Hamburg, June 2017.

[SB15] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT press, 2015.

[Sch+12] Max Schwarz et al. "NimbRo-OP humanoid teensize open platform". In: *In Proceedings of 7th Workshop on Humanoid Soccer Robots, IEEE-RAS International Conference on Humanoid Robots, Osaka*. Citeseer. 2012.

[Sch+15] John Schulman et al. "Trust region policy optimization". In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897.

[Sch+16] John Schulman et al. "High-Dimensional Continuous Control Using Generalized Advantage Estimation". In: *4th International Conference on Learning Representations*. 2016. arXiv: 1506.02438v6 [cs.LG].

[Sch+17] John Schulman et al. *Proximal Policy Optimization Algorithms*. July 20, 2017. arXiv: 1707.06347v2 [cs.LG].

[Sch97] Stefan Schaal. "Learning From Demonstration". In: *Advances in Neural Information Processing Systems 9* (1997).

[Tak+19] Kanta Takasu et al. *Extended Abstract from CIT Brains 2020*. Tech. rep. 2019.

[TWS18] Faraz Torabi, Garrett Warnell, and Peter Stone. *Behavioral Cloning from Observation*. May 2018. arXiv: 1805.01954 [cs.AI].

[Weg17] Felix Wege. "Development and Implementation of a Dynamic Kick for the NAO Robotic System". Bachelor's Thesis. Hamburg University of Technology, 2017.

Online

[1] *A Brief History of RoboCup*. URL: https://www.robocup.org/a_brief_history_of_robocup (visited on 05/19/2021).

[2] *Technical overview – NAO Developer Guide – SoftBank Robotics*. URL: https://developer.softbankrobotics.com/nao6/nao-documentation/nao-developer-guide/technical-overview (visited on 05/20/2021).

[3] Jack Clark and Dario Amodei. *Faulty Reward Functions in the Wild*. Dec. 2016. (Visited on 12/22/2020).

[4] Megajuice. *Reinforcement Learning Diagram*. URL: https://commons.wikimedia.org/wiki/File:Reinforcement_learning_diagram.svg (visited on 06/12/2021).

Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Bachelorstudiengang Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Hamburg, den 18. Oktober 2021 Timon Engelke

Veröffentlichung

Ich stimme der Einstellung der Arbeit in die Bibliothek des Fachbereichs Informatik zu.

Hamburg, den 18. Oktober 2021 Timon Engelke