Generating Human-like Hopping with Deep Reinforcement Learning

Generieren menschähnlicher Hüpf-Bewegungen mittels Deep Reinforcement Learning
Bachelor-Thesis von Rustam Galljamov

1. Gutachten: Prof. Dr. phil. André Seyfarth
2. Gutachten: Prof. Dr. mont. Mario Kupnik
3. Gutachten: M.Sc. Guoping Zhao

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Elektrotechnik und Informationstechnik

Lauflabor Locomotion Lab
Institut für Sportwissenschaften

Generating Human-like Hopping
with Deep Reinforcement Learning
Generieren menschähnlicher Hüpf-Bewegungen mittels Deep Reinforcement Learning

Vorgelegte Bachelor-Thesis von Rustam Galljamov

1. Gutachten: Prof. Dr. phil. André Seyfarth
2. Gutachten: Prof. Dr. mont. Mario Kupnik
3. Gutachten: M.Sc. Guoping Zhao

Tag der Einreichung: 8.11.2018

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

Hiermit versichere ich, Rustam Galljamov, die vorliegende Bachelor-Thesis gemäß § 22 Abs. 7 APB der TU Darmstadt ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

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

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

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

Thesis Statement pursuant to § 22 paragraph 7 and § 23 paragraph 7 of APB TU Darmstadt

I herewith formally declare that I, Rustam Galljamov, have written the submitted thesis independently pursuant to § 22 paragraph 7 of APB TU Darmstadt. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.

I am aware, that in case of an attempt at deception based on plagiarism (§38 Abs. 2 APB), the thesis would be graded with 5,0 and counted as one failed examination attempt. The thesis may only be repeated once.

In the submitted thesis the written copies and the electronic version for archiving are pursuant to § 23 paragraph 7 of APB identical in content.

For a thesis of the Department of Architecture, the submitted electronic version corresponds to the presented model and the submitted architectural plans.

Datum / Date: 08.11.2018

Unterschrift/Signature: R. Galljamov

Abstract

The longstanding search for control approaches with the goal of achieving dynamic and robust locomotion skills in legged robotic systems is still in progress. Deep Reinforcement Learning (DRL) at the same time is advancing very fast and promises the ability to face such complex control tasks.

In this work, we demonstrate the ability of DRL to learn robust model-free controllers for generating human-like hopping in a realistic simulation of the GURO Hopper, a two segmented robotic leg. Therefore we use the Proximal Policy Optimization Algorithm and the MuJoCo physics engine. To make the simulation realistic, we considered joint and motor torque ranges, sensor precisions and identified some critical parameters by experiment.

To train our DRL agent, we use kinematic data of a hopping human as a reference, collected by a motion capture system. Due to the significant differences in morphology and mass distribution between the human and the considered robot, we learn only to control the stance phase, where in the flight phase PD position controllers bring the robotic leg in a desired posture. The agent does not require any information regarding the robot's structure, it's mass distribution, equations of motions and system parameters like joint friction and damping, being critical components in most of the traditional control approaches.

Using the same algorithm, controllers for hopping with one and both motors can be learned. Using two motors, the learned hopping motion can be maintained for 100 hops in a row and is robust against ground level drops of up to 50% of the robots leg length. It is important to mention that no perturbations were included during training. Moreover, the controller learns to mimic the human leg stiffness without being provided with this kind of information.

Finally, we summarize our experience into several recommendations for applying DRL to control robots with the goal of generating human-like motions, believing that DRL offers a promising alternative to traditional control approaches.

Keywords: deep reinforcement learning, legged locomotion, motion control, model-free control, learning-based control, human hopping, stable hopping, perturbation recovery

Table of Contents

1 Motivation and Structure

Science fiction movies are full of humanoid robots walking and running human-like on two legs, performing parcours and often extending the capabilities of human bodies in terms of strength, perception and agility. When it comes to science fact as of July 2018 none of the existing robotic systems are able to perform locomotion tasks on a level really close to humans in terms of dynamics and robustness.

When looking at the finals of the DARPA Robotics Challenge in 2015, the biggest robot competition in the human history with the goal of letting robots perform several tasks in an environment shaped for humans, the results are disillusioning. Almost all robots, which were the best in the world at that time, have trouble standing, walk very slow and unsteady and fall regularly.

Why after more than 30 years of research in bipedal dynamic locomotion robots are still not able to walk in a human-like manner?

The answer lies in the complexity of this seemingly simple motion and the capabilities of the human body in solving very complex control tasks. Every few milliseconds we receive hundreds of raw sensory information and process this data in our central nervous system, formed by the brain and the spinal cord. These units then generate hundreds of commands for up to 600 skeletal muscles, which move our body segments with an efficiency and dexterity no actuators in the world can compete with so far.

It is clear, that the human body is superior to current humanoid robots in several areas, which should be targeted separately. Within the scope of this work and the comparison between man and robots, we focus on the central nervous system, simplified to a model-free black box controller, taking raw sensory data as input and outputting actuator commands. We are interested to find out, whether deep reinforcement learning (DRL) techniques can mimic the capabilities of the central nervous system as a controller and thus provide robots with the competence to move more agile, robust and efficient.

In order to contribute to answering this question, the aim of this work is to show, that DRL can be utilized to develop a model-free learning-based controller, able to generate human-like hopping motions in a realistic simulation of a two segmented robotic leg. Therefore, a realistic simulation environment of the GURO Hopper Robotic leg will be built, considering the motor torque and joint ranges as well as the sensor precision of the real system. A DRL algorithm will then be implemented, able to learn controlling the robot in simulation in order to generate human-like hopping. For training the agent, motion captured human hopping data will be scaled down to the robotic leg and used as a reference. As soon as stable hopping is achieved, we want to formulate suggestions on how to apply deep reinforcement learning techniques in the field of robotics, based on our experiences.

Figure **01** shows six snapshots of a simulated 34 degrees of freedom full body humanoid, performing a cartwheel. Being trained with DRL by Peng et al. (2018a), this humanoid is able to perform a large collection of highly complex dynamic motions from disciplines like dance, acrobatics and martial arts, making DRL the most promising approach to accomplish our goal.

This work begins by presenting related literature in the following paragraphs, describing the development of deep reinforcement learning up to it's current state and illustrating it's use in the field of robotics and animation, where complex bodies have to be controlled to perform dynamic locomotion tasks. In chapter 3, key terminology and concepts regarding the biomechanics of the

hopping motion as well as machine- and reinforcement learning are introduced. The tools used throughout this work are summed up in the next chapter, including a definition of human-like hopping in the context of the used robotic leg. How we implemented our learning-based controller is then outlined in chapter 5. Here, also the simulation we built is described, as well as the process to make it realistic. After this, we present the results of our work (chapter 6) and discuss them in the chapter that follows. Within our discussion, the suggested control approach and it's results are compared with similar learning-based methods and traditionally used control strategies. Finally, our conclusions are drawn in the last chapter.

Figure 1: Snapshots of a simulated 34 DOF humanoid performing a cartwheel. The actuators are controlled using a learning-based approach, using deep reinforcement learning methods. Reprinted from Peng et al. (2018a).

2 Related Work

Legged robots are highly nonlinear complex systems. When, in addition, dynamic movements are to perform in a robust manner, their control reaches an even higher level of complexity. To face this challenging task, besides extending traditional control methods, alternative approaches to linear and optimal control strategies should be found. Reinforcement Learning (RL) has so far been considered to have great potential to meet this challenge, as will be presented in the following.

Benbrahim and Franklin (1997) were among the first to use reinforcement learning to control a bipedal walking robot. However, the robot had a simple limited design and was connected to a cart, similar to a baby walker, which dramatically simplified the task of walking.

Since then, RL was mainly used as a sub component of the controller (Morimoto et al., 2004) or with the goal to improve the performance and efficiency of existing non-learning-based controllers (Fankhauser et al., 2013).

A promising step forward has been made when Mnih et al. (2013) combined RL and deep learning (DL), hence the name deep reinforcement learning (DRL), and were able to play several Atari-Games by only using the game's pixels as the input to the algorithm. One year later Mnih et al. (2015) achieved human-level performance by further advancing this combination.

Atari-Games come with a high input space when raw pixels are used, but are discrete and low dimensional in their outputs. To transfer the results to the field of robotics, it had to be shown that deep reinforcement learning also can handle high dimensional continuous output spaces. That happened a short time later. Lillicrap et al. (2015) applied DRL to 20 tasks requiring continuous actions in a simulation environment, including a cart pole swing-up and legged locomotion with a lower body humanoid. Further locomotion application followed.

Schulman et al. (2016) additionally advanced the algorithms and were able to control a simulated full body humanoid model with 33 degrees of freedom (DOFs) to perform complex dynamic movements like running, jumping over obstacles and getting up from the ground. A new algorithm, Proximal Policy Optimization (PPO) (Schulman et al., 2017), allowed to reduce the implementation effort in achieving such results.

The learned movement however often were unnatural, a gap, Peng et al. (2017) have targeted. By providing the agent with reference trajectories collected from human experiments using a motion capturing system, the agent was able to learn walking and running in a visually human-like style. In addition to that, they combined a low- and a high-level controller, both being based on DRL, to perform more sophisticated tasks like dribbling a ball to a desired position or walking on narrow paths without getting off the road. The high-level controller was trained to plan the footsteps required to perform the desired motions, where the low-level controller had to execute this footsteps.

The simulated biped model used by Peng et al. (2017) had no arms and no head, which shouldn't play a significant role in considered locomotion tasks but makes the model less realistic.

To learn even more complex tasks, Peng et al. (2018a) introduced a new full body humanoid model with 34 degrees of freedom (DOFs), already shown performing a cartwheel in figure 0**1**. Using the PPO algorithm (Schulman et al., 2017) and several new techniques, that will be presented in sections 5.4.4 and 5.4.5, they were able to learn such complex tasks as side- and

backflips, throwing a ball, perform kicks and jumps over obstacles. Furthermore, they trained a simulated model of the humanoid Atlas Robot in it's version from 2015, build by Boston Dynamics, to accomplish most of these tasks. By using reference trajectories from full body motion capture recordings as an input to the algorithm, all learned movements looked human-like.

Due to their focus on computer animation, Peng et al. (2017, 2018a) used massless and unrealistic strong motors, as these parameters are of a subordinate importance in this area of research. The question was still to answer, whether DRL can also be applied on realistic simulations of existing robotic systems, respecting the motor and sensor properties given in the considered hardware. Xie et al. (2018) faced this challenge by building up a close simulation model of the Cassie bipedal robot from Agility Robotics with 20 degrees of freedom and 10 actuators. The model includes joint and motor torque limits presented in the real robot as well as realistic joint damping. In this significantly more realistic environment the robot learned stable walking, robust to pushes and uneven terrain. Similar to Peng et al. (2018a) the DRL algorithm PPO was used. The controller, learned in simulation, haven't been tested on the real robot so far, but Agility Robotics claim on their website that controllers developed against the simulation often can be used with minimal to no additional effort on the real system (www.agilityrobotics.com/sims, Accessed: 2018-10-21). However, Xie et al. (2018) used the learned controller to augment the target angles from a reference motion, which then were followed by PD position controllers. This augmentation was shown to be effective to develop robust controllers for an underactuated system. Their agent indeed, was not able to directly control the robot's motors to generate the desired motion.

To sum up, deep reinforcement learning offers a promising approach for controlling complex nonlinear multibody systems with high dimensional continuous input and output space (Lillicrap et al., 2015; Peng et al., 2017). DRL was utilized to achieve complex human-like locomotion tasks in simulated but unrealistic full body humanoids (Schulman et al., 2016; Peng et al., 2018a) as well as in realistic models of existing bipedal robots (Peng et al., 2018a; Xie et al., 2018). The PPO algorithm (Schulman et al., 2017) reduces the implementation effort of DRL control approaches. Reference motions from humans can be used to learn visually human-like movements in simulated characters (Peng et al., 2018a).

Within the context of these achievements it is to expect, that it is possible to achieve human-like motions in real hardware systems by using human reference motions. Hopping provides the simplest dynamic locomotion task in humans and is therefore well suited to investigate this hypothesis, that will be done in this work. Despite its simplicity, it is still a very important task considered being one of the fundamental subfunctions of locomotion (Sharbafi et al., 2017).

3 Background

3.1 The Hopping Motion

This subsection describes the hopping motion and introduces key terminology from the field of biomechanics.

The hopping motion is a dynamic nonlinear movement with alternating stance and flight phase (Sayyad et al., 2007). The flight phase is characterized by the foot having no contact with the ground. The beginning of the flight phase, the moment when the foot leaves the ground, is called *takeoff*. The end of the phase, when the foot comes into contact with the ground again, is referred to as the *touchdown*. The stance phase can be divided into two sub-phases. In *compression* phase the center of mass (COM) of the human body performs a downward motion and is decelerated to zero. Thereafter, an acceleration in the opposite direction begins, marking the beginning of the *extension* phase, which ends with the takeoff.

The resulting force between the foot and the ground during stance phase is generally referred to as the ground reaction force (GRF). The point on the ground, this force vector is going through, is defined as the center of pressure (COP).

Given the COM and COP positions during hopping, a leg length, as the distance between the COP and COM can be specified (Blum et al., 2009). When defining the touchdown leg length as the rest leg length, a leg compression can be calculated in analogy to a spring. Given the leg compression and the corresponding GRFs at the same point in time, a leg stiffness can be computed as the change in leg length over the change in ground reaction forces in the same time period (Seyfarth et al., 1999).

We define the hopping as stable, when for at least 20 hops the amplitude and frequency aren't changing more than by 15%.

3.2 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) has shown to be effective in providing controllers for achieving sophisticated movements in complex, highly nonlinear multibody systems (Peng et al., 2017; Schulman et al., 2017; Peng et al., 2018a; Xie et al., 2018). Given these examples, DRL is expected to help to achieve the goal of this paper.

This section gives a brief overview of the basic concepts of machine-, reinforcement- and deep reinforcement learning required to understand the decisions and results reported in the Methods. As the goal of this thesis is applying existing algorithms and not developing new ones, (mathematical) details of the algorithms will not be discussed.

3.2.1 Machine- and Deep Learning

Machine Learning (ML) can be understood as an alternative programming paradigm. To let a computer or robot perform an action, traditionally it is necessary to program a set of instructions, to explicit write down the rules and processes that have to happen with the input to get the desired output. It is easy to find examples where defining these instructions is impossible or just unpractically complicated. The task of letting a program decide if a picture contains a cat or not is such an example. For the computer a picture is just a huge multidimensional array of numbers. Finding mathematical or logic operations to apply to each individual pixel in order to

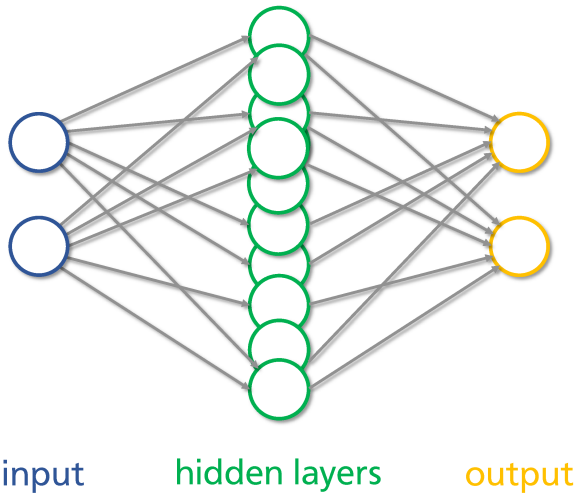


Figure 2: Simplified illustration of a neural net consisting of an input, output and a hidden layer.

output the probability of a cat being portrayed in the given picture is not possible. This is where machine learning comes in. By providing an ML algorithm with several hundred thousand of pictures, each labeled as containing a cat in it or not, it will learn the instructions on it's own, to recognize cats in previously unseen pictures. This kind of learning where each input is labeled with the correct answer is called *supervised learning*. ML powered object recognition often reaches and even surpass the human performance level (Ioffe and Szegedy, 2015; He et al., 2015).

To find the desired output, the inputs are passed through a network of processing units called *artificial neurons*. These neurons receive several inputs, apply a predefined mathematical function to them and output the results. Each of the neuron's inputs is weighted by a scalar parameter. To output the correct answer, the right weights have to be found. Several neurons can be stacked vertically to form a *layer*. Multiple horizontally stacked layers form an *artificial neural network*. The first layer is then called the *input layer*, the last one the *output layer*, where those in between bear the name *hidden layers*. Figure 2 illustrates a neural net with one hidden layer.

Several definitions of the term *deep learning* can be found in the literature (LeCun et al., 2015; Goodfellow et al., 2016). In order to follow the work presented in this paper, it is sufficient to define an artificial neural network with several more then one hidden layer as a deep network and consequently its use in a learning-based control approach as deep learning. Recalling that an artificial neuron can have multiple inputs, for which a correct weight has to be found, and a layer can have more than a thousand of neurons, a deep network can result having hundreds of millions of adjustable weights that have to be optimized (LeCun et al., 2015).

3.2.2 Reinforcement- and Deep Reinforcement Learning

Reinforcement learning (RL) is a subcategory of machine learning (ML) in which an agent is observing and interacting with an environment by choosing actions A_t from an action space in order to maximize a reward signal R_t. The environment's state is described by a vector of parameters S_t from a state space and can be changed by the agent's actions. Each item of the state vector is often referred to as a feature. To maximize the reward, the agent has to find

an optimal mapping of actions to take in every given state (Sutton and Barto, 2018). Figure 3 illustrates the reinforcement learning problem.

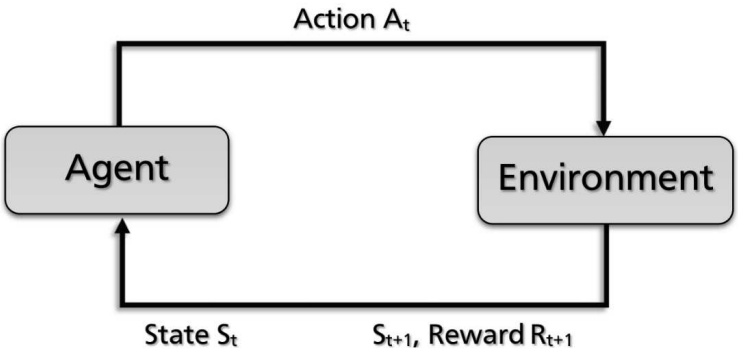


Figure 3: The interaction between an agent and it's environment in a reinforcement learning scenario. At timestep t, the agent chooses actions A_t based on S_t and receives the Reward R_{t+1} and the next state $St + 1$.

An example can help to understand the basic components of an RL problem. Let's consider a robotic arm operating in a 2D space on the surface of a table having the goal of picking up an object with a gripper. The environment is everything mentioned in the last sentence. The robotic arm itself is against expectation not the agent himself but part of the environment. The agent in an RL context is the component making the decisions and trying to improve them. In this example the agent is the algorithm controlling the motors of the arm. The states can be the joint angles and angular velocities, the current in each motor and the position of the object to pick up. The actions will be the current sent to the motors. The reward can be chosen in different ways and could consist of two parts here: reward for navigating the gripper to the object and for picking it up. The first part could target at decreasing the distance of the gripper to the object and thus give a positive reward if an action brings the gripper closer to the object and negative reward, also called punishment, for actions that get it away from the object.

In order to maximize the reward, the agent can follow different strategies. It can estimate how good it is to take an action or to be in a certain state. If the former is chosen, an *action-value function* has to be found, in the latter a *state-value function*. Value in this context is given by the expected long term reward for taking an action or being in a state. RL approaches, combining both of these functions are called *actor-critic* algorithms. Using these functions, the agent aims at finding a *policy*. Given a state vector as input, the policy outputs the action to pick and is thus the actual controller in a deep reinforcement setting.

When the mentioned functions and the policy are approximated by a deep neural network, we speak of a DRL approach. It is common to collect a specified amount of observations, referred to as the *batch size*, then divide this observations in smaller sets, called *mini batches*, and then use these one at a time to update the mentioned neural networks. The *learning rate* or *stepsize* determines, how much the network weights can be changed during an update.

The stepsize is one of several *hyperparameters* of an algorithm. Unlike the weights of the neural network, which are adjusted by the algorithm, these parameters have still to be chosen by the human. The algorithms are often very sensitive to the selection of hyperparameters and suggestions and tuning approaches not always can be generalized across all algorithms.

4 Methods

4.1 GURO Hopper Robotic Leg

As stated in chapter 1, the main goal of this thesis is developing a learning-based controller to reproduce the hopping motion in a realistic simulation of a real hardware system. In the frame of this work, the robotic platform GURO Hopper (GURO) is used. GURO is a two-segmented robotic leg, designed and built by Guoping Zhao in 2018 and extended by Vitus Henning during his Bachelor Thesis (Henning and Zhao, 2018). Figure 4 shows a photo of the robot.

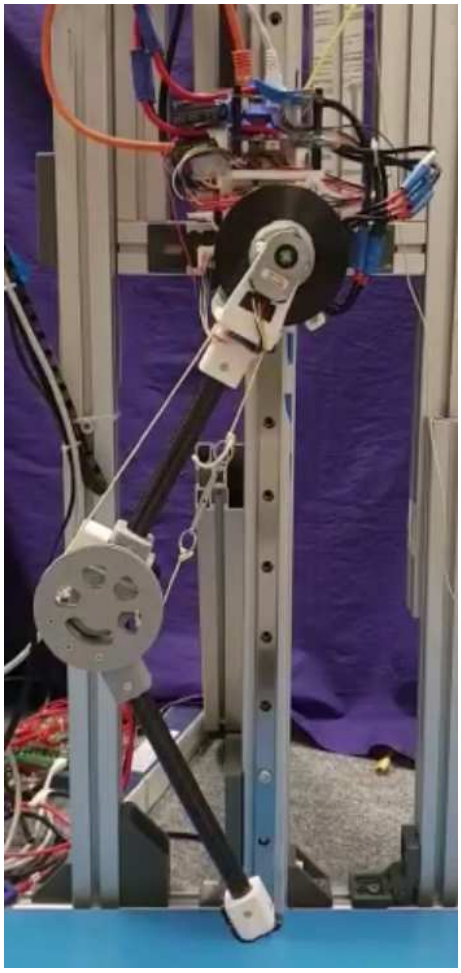


Figure 4: The GURO Hopper robotic leg connected to a vertical linear guide system limiting the hopping motion to 1 degree of freedom.

Both segments, representing the human thigh and shank, are realized with carbon tubes and are connected by a 1 degree of freedom (DOF) hinge joint imitating the human knee. The hip of the robot is also a 1 DOF hinge joint connected to a vertical linear guide system which limits the hopping motion to 1 degree of freedom. All mechanical parts beside of the bearings are 3D printed with the thermoplastics *PLA*, standing for *polylactic acid*. To increase the friction with the ground and thus reduce the slipping, a piece of a bike tire is mounted on the cylindrical foot, which is rigidly connected to the shank tube.

Together with motors and electronics, the robot weighs 2.329 kg.

For actuation of the knee and the hip, two brushless DC motors are used. In order to reduce the inertia, both motors are located at the robot's hip. This allows a direct driven actuation of the hip joint. A rope mechanism with a transmission ratio of 4:1 connects the motor for the knee actuation with the knee axis. The rope mechanism consists of two pulleys, one lying in the same axis with the knee motor and one located at the knee axis. Two ropes connect both pulleys and thereby transmit the torque in the knee motor to the knee axis.

The state of the robot is sensed with several sensors. Mechanical encoders measure joints positions as well as the motors rotor orientations which are required for their low-level control. A pressure sensor mounted on the foot detects ground contact . For a more precise measurement of the GRFs, a Force Plate built by *Kistler* can be utilized. The vertical hip position is measured by an infrared light distance sensor.

The motors are powered by a battery, at the moment located next to the robot to reduce its weight. Electronics, required to read out and amplify sensor data, as well as the motor controllers, are placed on a 3D-printed platform mounted on the hip above the motors.

The communication with the robot is performed at 1kHz. The usage of an embedded low level current controller and the linear relationship between the current and torque in the required range, allow a feed-forward torque control of both motors.

A complete overview over all system components and properties is presented by Henning and Zhao (2018).

4.1.1 Human-like Hopping with GURO

When observing humans hopping in place, it is easy to note that the foot plays an important role. GURO hopper does not have one, so an alternative definition of human-like has to be formulated. Throughout this paper we use *human-like* in context of the GURO robot as having similar leg length kinematics in the vertical axis (scaled down vertical leg length and scaled down leg length derivative), which in a hopping motion are identical with the vertical COM kinematics. In the biomechanics literature several definitions of the leg length in a simplified two segments robotic leg exists (Geyer et al., 2003; Blum et al., 2009). For this work, we choose the definition of (Blum et al., 2009), where the leg length is measured between the COM and the COP.

An important point to mention in this section, is our decision to restrict the DRL algorithm to only control the stance phase. In the flight phase, PD position controllers for the knee and hip angles bring the leg in the desired touchdown posture, specified as having a knee angle of 142° and a hip angle of 17°. The decision has been made due to the significantly different mass distribution in the considered robotic leg and the human, resulting in very distinct flight phase behavior and making the comparison unfair.

4.2 OpenAI Gym and Baselines

With the rising popularity of deep reinforcement learning several open source implementations of the state of the art algorithms are available online. The most cited repository is the *OpenAI Baselines* Library, promising high-quality implementations of the latest algorithms in DRL (Dhariwal et al., 2017).

OpenAI also created the *OpenAI Gym*, a collection of Open Source simulated environments for training and comparing the results of reinforcement learning agents (Brockman, Cheung et

al., 2016). Using these environments, which come with a uniform interface, researchers all over the world get a possibility to focus on implementing the algorithms, a straightforward way for reproducing other researchers results and compare different algorithms against each other.

All environments meet the criteria of a full reinforcement learning problem that receive actions from an algorithm, apply them in simulation and return the following state and a reward signal.

All algorithms in the Baselines repository are designed to work with these environments out of the box. Consequently to reduce the effort of implementing the complex algorithms on our own, it was obvious to create an OpenAI Gym Environment as the interface to our simulation. Details on this will be given in the next sections.

4.3 MuJoCo physics engine

Learning to hop like learning to walk takes time and requires lots of failures until the motion is finally learned. When Adolph et al. (2012) observed 12- to 19-months old children while playing outside, they counted more than 2000 steps and 17 falls per hour. When following a learning-based approach to finding a control to let the robot hop, failed attempts are to expect more frequently, especially when reinforcement learning algorithms are involved due to their exploring nature. To avoid breaking the robot during learning a simulation environment is used.

There are several simulation environments available: Bullet, MuJoCo and ODE to name a few. Some of them are specially designed for simulating sophisticated dynamics of multibody systems, like such found in robotics. For a robot interacting with its environment also contact dynamics - e.g. impact forces when a foot touches the ground - play an important role and therefore have to be integrated into the simulation environment for it to be useful for the field of robotics.

Within the scope of this work the physics engine MuJoCo, mainly developed by Emanuel Todorov (Todorov et al., 2012), is used. MuJoCo stands for **Mu**lti-**Jo**int dynamics with **Co**ntact. The choice of a simulation environment has been strongly guided by the goal of achieving human-like hopping by using deep reinforcement learning. To focus on the research question and less on required implementations, an environment had to be chosen that would minimize the effort of implementing state of the art reinforcement learning algorithms as well as building the interface between the agent and the environment. As was mentioned before, the OpenAI Baselines repository (Dhariwal et al., 2017) provides high quality state of the art deep reinforcement learning algorithms. For complex simulations of multibody systems, OpenAI Gym provides an interface to the MuJoCo Simulator, which allows to build a full reinforcement learning environment around the MuJoCo model. This is one of the reasons why using the MuJoCo Environment for Simulation was the obvious choice.

It is important to note that besides this reason MuJoCo is in a very good position compared to other simulation environments. In their paper about MuJoCo, Todorov et al. (2012) present several advantages over competing products. Xie et al. (2018) used this simulation environment to develop deep reinforcement learning based controls for the bipedal walking 20-DOF robot Cassie. On their website, Agility Robotics, the developer of Cassie, states that there is often “little to no downtime” required to use a controller on the real robotic system developed against the simulation (www.agilityrobotics.com/sims, Accessed: 2018-10-21).

4.4 The Proximal Policy Optimization (PPO) Algorithm

The Proximal Policy Optimization (PPO) Algorithm was introduced by Schulman et al. (2017) and is currently one of the most advanced state-of-the-art algorithms in deep reinforcement learning. It is an actor-critic algorithm, following the approach of policy gradient methods. Instead of trying to estimate the value of being in a state or taking an action, this class of methods directly updates the policy network (Peters and Schaal, 2008).

The main advantages of PPO over other approaches are a more robust and efficient learning progress combined with lower implementation effort, as less hyperparameters have to be tuned. Efficiency comes from the possibility to run multiple agents in parallel, each using a copy of the current policy, to collect training data. The robustness is founded in the usage of the *clipping objective function* as an additional component to limit how much a policy can be changed on one iteration. This way the policy is guaranteed to monotonically improve over the training time without deteriorating due to exploration.

The PPO algorithm has already been shown to be effective in learning to generate complex motions in complex systems with more than ten degrees of freedom and multiple actuators (Schulman et al., 2017; Peng et al., 2018a; Xie et al., 2018).

Being a state-of-the-art algorithm, explaining it in further detail would require the introduction of too many RL concepts, which would go beyond the scope of this work.

5 Implementation

This section begins by shortly naming the tools used throughout the implementation. Following, a definition of human-like hopping motion with the differently shaped GURO Hopper robot will be discussed. The next two subsections present the simulation and the human hopping data, used as a reference for training the learning-based controller. Finally, the section will be closed with a detailed presentation of the actual implementation of our learning based algorithm.

The major part of the code was written in the Python programming language. MathWorks MATLAB and GNU Octave were partially used to process the data from the system identification experiments, presented in section 5.2.

5.1 GURO MuJoCo simulation environment

Our learning-based control approach had to be proven in a simulation environment. Therefore, we developed a simulation model using the MuJoCo physics engine (Todorov et al., 2012). The most important features of the model will be discussed in this section. Figure 5 shows the robot in the simulation environment, specifying the chosen angle definitions. The simulation is running at 2kHz and uses the Euler integration.

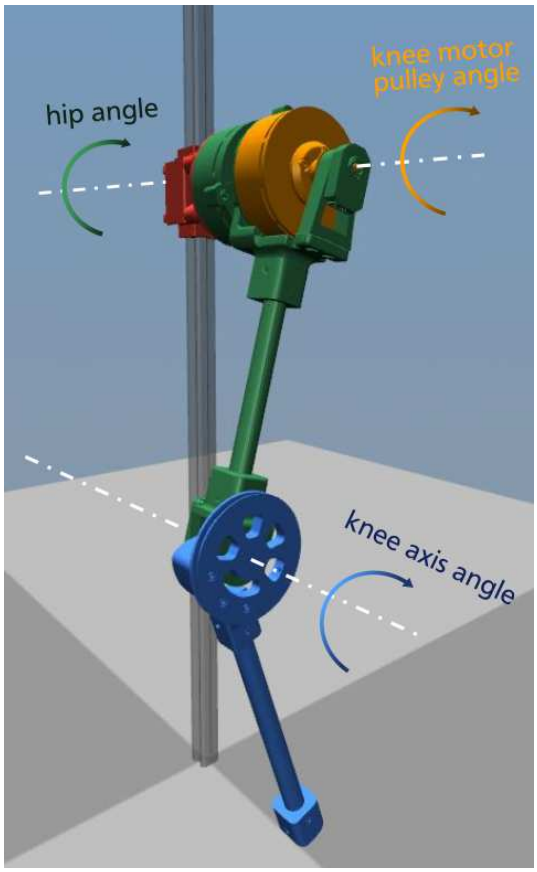


Figure 5: GURO Hopper robotic leg in the MuJoCo simulation environment. The arrows indicate the positive direction of the labeled motor and joint angles.

MuJoCo provides its own modeling format MJCF (MuJoCo Format). By reusing parts of the environment from the open-source MJCF model of the Cassie robot, which has been proven to be realistic (Xie et al., 2018), we had a good foundation on which to build. The meshes

of the GURO Hopper, used for collision detection, were exported from the CAD models and simplified by reducing the number of faces to match the requirements of the physics engine. For that purpose the program *MeshLab* (Cignoni et al., 2008) was used. Due to differences in units([mm] in Inventor and [m] in MuJoCo) all meshes were scaled by 1/1000 in all three space directions. Inertial data like mass and moment of inertial of all parts as well as their positions in 3D space were read out from the CAD models and adjusted by manually weighting the parts, due to the programs wrong estimation for inertial properties of 3D printed objects.

The motors are modeled in a realistic way, matching the inertial data given in the specification sheet. The stator and the rotor of the motors are implemented as individual parts connected by a revolute joint.

The joint angles and sledge position of the simulated robot are directly read out from the simulation. To make this measurements more realistic, the values are rounded to the precision of the sensors in the real system. Sensory delay and noise are not considered so far. Ground reaction forces are attained from the contact forces between the ground plate and the robots foot. Table 1 gives an overview over the motor and joint ranges and table 2 summarize the sensor precisions implemented in the MuJoCo simulation.

Table 1: Summary of the motor torque and joint angle ranges in the MuJoCo simulation.

Motor / Joint	**Range**
Knee and Hip Motors	[-5:5] Nm
Virtual Knee Axis Motor	[-25:25] Nm
Hip Joint, Vertical sledge position	unlimited
Knee Joint	[60:170] [°]

Table 2: Overview over the sensor precisions implemented in the MuJoCo simulation.

Sensor Measurement	**Range**
Joint Angles	0.5 [°]
Joint Angular Velocities	0.5 [°/s]
Vertical sledge position	1 [mm]
Vertical sledge velocity	1 [mm/s]
Ground Reaction forces	0.1 [N]

Our model considers all joint ranges existent in the real robot. The same applies to the torque ranges of both motors. As MuJoCo by default is using soft constraints, modeling them as a non-linear spring-damper, solver parameters (solimplimit, solreflimit) were adjusted for the knee joint in order to match the hard limit in the knee. These parameters represent the damping and spring constant as well as the non-linearity of the constraint and were tuned by hand until the exceedance of the limit joint angles was below 1°, resulting in $(solimplimit, solreflimit) = ((0.98, 0.99, 0.001), (0.02, 10))$. Even harder constraints tended to make the simulation unstable and were not considered.

Due to the direct driven actuation on the hip joint, there is no transmission chain to model, which brings the simulation closer to reality by reducing the number of components which are hard to model in a realistic way. In addition, because a low level current controller is used in the real hardware system and the the current-torque-relationship of the motors is linear in

the required range, the real motor can be feed-forward torque controlled. The hip joint motor simulation as a torque controlled motor with a transmission ratio of 1 and output ranges of ±5Nm is thus very close to reality.

As the capability to model a rope transmission was not yet given in the MuJoCo simulator, a workaround was found. For this purpose an additional virtual (mass-less) motor is added to the MJCF model, acting on the knee axis. Given the transmission ratio of 4:1 the virtual motor was assigned the torque ranges of $25Nm$ in both directions, leaving a 25% buffer for unforeseen additional forces due to dynamic interactions with the environment. In addition, we assume having two mass-less ropes, connecting the knee motor pulley and the knee axis pulley. Both ropes are modeled by a linear spring-damper with identical parameters. When the simulation is initialized, these two ropes have their initial length $l_{rope,0}$. When a torque is applied to the knee motor, the knee motor pulley starts rotating and roll up the virtual rope with a certain speed. Assuming the knee angle being constant at this same moment, the rope gets elongated. This elongation leads to a force in the rope, calculated by

$$F_{rope} = k_{rope}\,\Delta l_{rope} + d_{rope}\,\dot{\Delta l}_{rope} \tag{1}$$

where k_{rope}, d_{rope} and Δl_{rope} denote the stiffness, damping and the elongation of the virtual rope. In order to simplify the names, from now on the knee motor pulley will be referred to as the upper pulley and the knee pulley as the lower one. The rope elongation Δl_{rope} can be calculated using the angle changes of the upper and lower pulley with regard to the initial angles, weighted by the radius of both pulleys (r_{lower} and r_{upper}). As one of the ropes is always slack and does not produces any force, it is sufficient to get the calculate the resulting torque on the knee by only considering one rope. It therefore has to allow negative forces, to account for the antagonistic rope being stretched. The following equation illustrates the calculation of the rope elongation Δl_{rope}.

$$\Delta l_{rope} = \varphi_{upper} r_{upper} - \varphi_{lower} r_{lower} \tag{2}$$

The elongation velocity $\dot{\Delta l}_{rope}$ can be obtained by taking the derivative of equation 2 or directly use the angular velocities of the affected joints.

$$\dot{\Delta l}_{rope} = \dot{\varphi}_{upper} r_{upper} - \dot{\varphi}_{lower} r_{lower} \tag{3}$$

The torque on the knee axis $M_{kneeaxis}$ is then calculated by $M_{kneeaxis} = F_{rope} * r_{lower}$ and is applied to the virtual motor acting directly on the knee axis. The influence of the rope force on the torque in the knee motor is also considered, resulting in only the net torque from the motor and the rope moment being applied to the knee motor in simulation.

The values for the stiffness $k_{rope} = 10^7 N/m$ and the damping $k_{rope} = 100Ns/m$ of the rope were chosen inspired by the values in the Simulink-model of the same robot developed by Guoping Zhao in 2018, already proven to be realistic. Due to a fixed timestep in our MuJoCo simulation and variable simulation step sizes in the Simulink model some adjustment were necessary to find the above mentioned values. In order to reduce high oscillations in the rope force, the angular velocities of both pulleys were slightly filtered by using exponential running smoothing (Gardner Jr, 1985) with a smoothing factor of $\alpha = 0.75$. The equation of the smoothing is given by:

$$s_t = \alpha x_t + (1-\alpha)\, s_{t-1} \tag{4}$$

Here, x_t denotes the new value at time t, s_t is the filtered value at the same time and s_{t-1} the filtered value of the last time the smoothing was applied.

Simple electrical motor dynamics were planned to be implemented by utilizing a low pass filter with the motor's time constant between the set current and the applied torque on the motor. Additionally, the current would be multiplied by the motor's torque constant to get the set torque for the motor. It was however decided against it, due to experiments showing, that the delay until a torque is presented in the motor after applying a current, is insignificantly small and thus can be neglected. Moreover, the current-torque relationship is linear in the required range.

This paragraph explains the choice of the default joint damping values. For the first test of the model, especially to make sure the inertial data and positioning of the segments are modeled correctly, all motors were replaced by position servos, provided by the MuJoCo physics engine. These servos come with a built-in proportional feedback control for the joint position. The rope transmission was not considered at this moment. As these servos only allowed tuning the proportional component of the position control, joint damping had to be introduced, in order to reduce oscillations around the target angles. Both parameters were hand tuned by observing the step response. As a result a damping constant of $0.05 Ns/m$, as a very rough estimate, was specified for all joints. The identified p-part for the hip joint servo was 130 and 30 for the virtual motor on the knee joint.

Since the servos in the simulation were soon replaced by torque controlled motors, much closer matching the real hardware system, we implemented our own proportional-derivative (PD) controllers for the knee and hip angle. These were necessary to control the GURO Hopper in the flight phase and has been shown to be practical for additional purposes. For stance and flight phase a dedicated PD controller was tuned. To figure out the best controller parameters for the flight phase, the robot's hip was held in place while observing the motors' responses to desired angle steps. The best parameters were identified as $(P,D)_{flight,hip} = (0.26, 9.6)$ and $(P,D)_{flight,knee} = (0.18, 2.8)$. Using this controllers parameters as a starting point, the stance phase controller was tuned by following desired knee and hip angle trajectories, resulting in the following parameters: $(P,D)_{stance,knee} = (0.52, 2.2)$ and $(P,D)_{stance,hip} = (0.48, 8.6)$.

In order to use DRL algorithms from the Baseline repository (Dhariwal et al., 2017) in combination with our simulation, an OpenAI Gym Environment (Brockman et al., 2016) was implemented, loading the MuJoCo simulation as an environment of a reinforcement learning scenario. Section 4.2 introduces the mentioned repository and the OpenAI Gym.

The ground properties as well as the frictional forces acting in the vertical sledge could not be derived from the CAD model and therefore were identified by experiment, presented in the next section. More precise damping and friction parameters could also be experimentally found for the joints, but we decided against doing it due to high effort and low expected improvement of the models accuracy.

5.2 System identification and simulation parameter optimization

With the goal in mind to make the simulation as realistic as possible, important system parameters were identified. This section begins by explaining the experiment conducted to identify the parameters and further investigates the techniques used to find the simulation parameters that closely reproduced the experiment's behavior in the simulation. Finally, the results of the identification are presented.

5.2.1 Identification of frictional forces in the vertical linear guide system

It is obvious and has been proven in earlier versions of the GURO Hopper by Zhao, that the forces acting in the sledge, connecting the robot's hip and the vertical linear guide system, highly influences the hopping behavior and stability. To further improve the closeness of the simulation to reality, the ground properties would need to be identified. As a result an experiment was conducted that will be presented in the following.

To target both parameters in one experiment, (Henning and Zhao, 2018) mounted a 3d printed part to the sledge of the linear guide system, having the same rubber being attached at its end as the rubber on the robots foot. The guide was vertically placed, rigidly connected to a table in order to reduce oscillations from the touchdown impact. The sledge was then dropped from four different heights, five times for each height. A Qualisys Motion Capture System recorded the position trajectories of two markers, glued to the sledge.

The experiment data was sampled with 500Hz. For each of the 20 recorded falls, the data was acquired for five seconds. To reduce noise in the recodrings, the mean of the positions of both markers at every timestep was calculated and used as the only reference from here on. Additional cleaning of the data was not necessary. The velocities were calculated with a Matlab-script provided by Zhao, used to get velocity trajectories from motion captured position data for several years. All following steps were implemented in Octave and Python.

This and the next few paragraphs explain the steps that were necessary to find the slide joint parameters. After the sledge was dropped, it entered a free fall, slowed down by the frictional forces in the linear guide. By reproducing the experiment in simulation and finding parameters that closely mimic the free falling trajectory of the sledge in the experiment, we can assume to have found the right parameters. In the first step, all free falling trajectories were extracted from the collected data. To exclude possible disturbances to the sledge position in the moment of dropping, the data was cropped 100ms after the drop and 10ms before the ground contact.

MuJoCo allows to specify a coulomb friction in the joint-attribute *frictionloss*, in the following referred to as μ, as well as a damping constant d, having the same name in the MJFC format.

The conducted experiment was reproduced in the simulation to find the optimal parameters. The interface to the MuJoCo simulation was implemented in Python using the open source library *mujoco-py*, developed by the OpenAI Robotics team. As inertial properties, only the mass was used due to a linear motion being considered. The simulation was run with the same frequency the data was collected with: 500Hz. Bayesian optimization (BO) (Snoek et al., 2012) was chosen to find the best parameters, as it is known in the literature for finding the optimal solution with a small amount of iterations. For its implementation, the open source Python library *BayesOpt* (Martinez-Cantin, 2014) was used.

One optimization iteration consisted of all 20 drops using the same slide joint parameters. For every fall, the initial vertical position and velocity of the sledge in simulation were set to the values collected in the experiment. After touchdown the free falling trajectory from simulation was extracted and compared to the one from the experiment to compute the cost. The cost function consisted of two parts. In the first part, the summed squared error between positions at every timestep in simulation and experiment was computed with the following equation, where N_{steps} denotes the total number of simulated timesteps:

$$Cost(\mu, d) = \sum_{t=1}^{N_{steps}} (x_{sim}(t) - x_{exp}(t))^2 \tag{5}$$

In previous runs of the optimization, where only this part of the cost function has been used, the free fall duration $\hat{t}$ in simulation differed a bit from the experiment durations for several of the 20 drops. For this reason, the cost function was extended by a second part. Here, the difference of the flight duration in simulation and experiment for every fall were taken to the power of 4. To make both parts equally important for the optimization, the first part was divided by 10, resulting in the same order of magnitude for both cost function parts. The cost of all drops were summed to get the cost of one optimization iteration, resulting in the following cost function:

$$Cost(\mu, d) = -\sum_{height=1}^{4} \left[\sum_{fall=1}^{5} \left[(\hat{t}_{sim} - \hat{t}_{exp})^4 + \frac{1}{10} \sum_{t=1}^{N_{steps}} [(x_{sim}(t) - x_{exp}(t))^2] \right] \right] \tag{6}$$

The result of the optimization can be viewed in figure 6, which compares the trajectories from experiment and simulation for one drop of each of the four heights. The corresponding parameters are $damping = 0.318195$ and $frictionloss = 1.499865$.

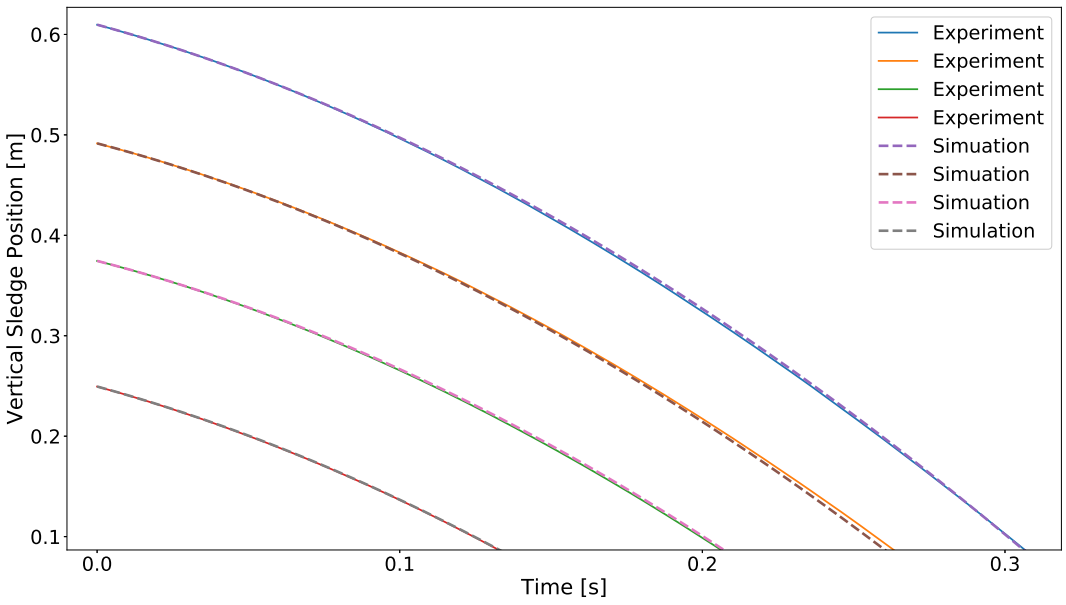


Figure 6: Results of the identification of the frictional forces in the vertical linear guide system. The trajectories collected during the experiment (continuous line) are compared to the trajectories in the simulation reproducing the same experiment.

5.2.2 Identification of ground contact parameters

After the sledge parameters were identified, the same simulation was used to identify the ground properties. The kinematic data has shown, that after touchdown the sledge bounced off the ground three times. The first idea was to determine the flight time after each rebound (hereafter called *bouncing times*) using the GRF data, collected with $1kHz$ with a force plate from *Kistler*, and find the parameters in simulation, that would result in same behavior. MuJoCo provides 5 parameters to determine the contact dynamics between two bodies. The contact is modeled as a non-linear spring-damper. Accordingly, the stiffness, damping and three parameters describing the nonlinear behavior can be optimized.

Due to the small weight of the sledge and small oscillations in the vertical guide after the touchdown, the GRF data was very noisy and could not be used for our purpose. Instead, the kinematic data looked promising for the same task and was utilized. The position and velocity trajectories of all 20 drops were again made use of. For each of them, the data was cropped shortly before touchdown and after the third rebound, following by a calculation of the bouncing times $\bar{t}_1$, $\bar{t}_2$ and $\bar{t}_3$. Subsequently, the simulation was initialized with the sledge having the position and velocity $10ms$ before touchdown. After calculating the bouncing times in simulation, the summed squared error (analogous to equation 5) between these and the bouncing times from the reference drop in the experiment was calculated and used as the cost function. If a set of ground properties resulted in no or less than three rebounds, the bouncing times in simulation were set to 0, leading to the maximum cost this way.

The optimization using this cost function resulted in similar bouncing times, but in a too compliant ground, leading to high ground penetrations. Adding additional cost for ground penetration did not solve the problem neither. For this reason the cost function was changed to a comparison of the whole sledge trajectory after the touchdown. Equation 5 was used for this purpose without any extension. As a result, the rebound trajectories for all drops were very similar, as can be seen in figure 7.

Despite the close matching of the reference trajectories in simulation, the ground parameters identified this way were not usable in the simulation, resulting in a too springy ground. As a consequence, the ground contact parameters were tuned by hand with the goal to reduce ground penetration and having realistic ground reaction forces.

To find the optimal parameters, the first optimization iterations were run with the highest possible ranges for all parameters. When after several iterations the optimal values were always found in a similar region of the allowed interval, the ranges were narrowed. Further enhancement of both optimization results were achieved by normalizing the parameter ranges. Therefore the same interval of $[0 : 1]$ was inputted to the BO algorithm for all parameters. The parameters proposed by the algorithm were then linearly scaled to their actual ranges before applying to the simulation. The last reduction of the cost function was achieved by initializing the optimization with the best parameters, found in previous optimization runs.

Table 3 summarizes all the identified parameters used in simulation.

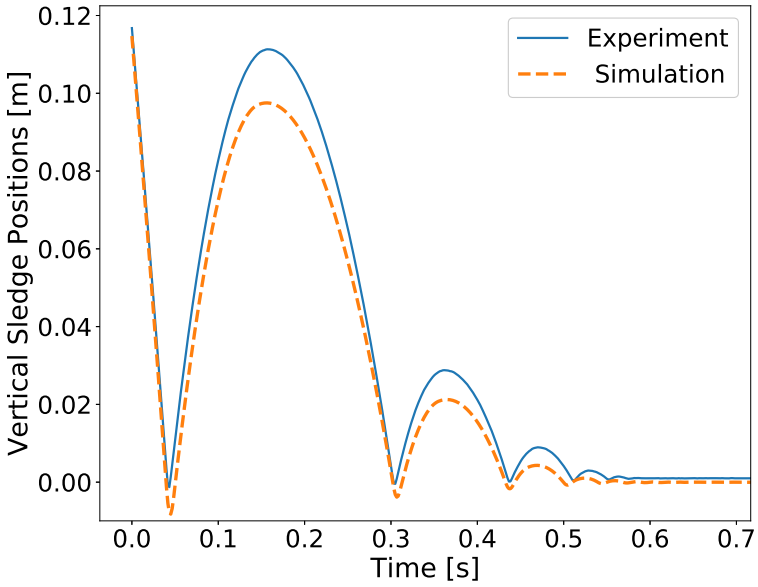


Figure 7: Results of the Bayesian Optimization identifying the ground contact properties. Comparison of the sledge's vertical position trajectory from the experiment (continuous line) and simulation (dashed line) for the maximum dropping height.

Table 3: Summary of all identified simulation parameters

Simulation Parameter	**Value**
Timestep	0.0005 [s]
Sledge, coulomb friction	1.5 [N]
Sledge, damping	0.318 [Ns/m]
Revolution Joints, coulomb friction	0 [N]
Revolution Joints, damping	0.05 [Ns/m]
Ground, friction coefficient	1 []
Ground, solimp (non-linearity parameters)	(0.98, 0.99, 0.001)
Ground, solref (time-constant, damping)	(0.02, 10)

5.3 Hopping Reference Data

Inspired by Peng et al. (2018a), reference data from motion capture recordings were used to train the agent. We now describe the experiment, where the reference data comes from and how we post-processed the data.

The experiment was conducted by Guoping Zhao. One subject hopped in place on a platform, having an integrated function to adjust its vertical position. Six perturbations in form of a negative vertical position change of the plate, also called *ground drops*, were introduced during different phases of the hop. Three of them (25, 50 and 75mm) were introduced during *mid-stance*, after about half of the stance phase duration. Another three ground drops with same amplitudes were performed while the subject was in the flight phase of his third hop. Three hops before the perturbation, the reaction to the ground drop on the fourth hop and three hops hereafter were recorded. Each perturbation was repeated 8 times.

Zhao computed the COM position trajectory, the leg length as defined in section 3.1 and the ground reaction forces. As only the stance phase behavior was to be learned, the data points during flight phase were removed.

Using this data, reference trajectories for the experiment were derived, as described in this and the following paragraphs. First of all, the human leg length trajectories for all hops before the perturbation were extracted. All hops with a touchdown leg length exceeding the mean touchdown length by more than two times the standard deviation were sorted out from the reference data. Next, the leg length of the human was scaled down to the robots leg length. The subject's leg length was approximated by the mean of the touchdown leg lengths of all unperturbed hops. The robots leg length was defined as the distance between the hip axis and the foot contact point with the ground, at 17° hip joint angle and 142° knee joint angle. This posture was identified to characterize the best kinematic touchdown conditions in previous experiments on the real robot.

Given the scaled leg lengths, the reference knee (α) and hip angle (β) trajectories were derived, using the equation 7 below. Here, 0.27 is the actual segment length of the real robot in meters, measured between the hip and the knee axis for the thigh and the knee axis and the COP for the shank. The small shank length changes as a function of the hip and knee angles, due to the foot's form, decided not to be significant and were therefore not considered.

$$\alpha = arccos(\frac{l_{leg}}{2 \cdot 0.27}) \qquad \beta = 2\, arcsin(\frac{l_{leg}}{2 \cdot 0.27}) \tag{7}$$

Having the angle trajectories, the angular velocities were derived by using the gradient function of the open source library *NumPy* (Oliphant, 2006).

As the simulation is run with 2kHz and the data was acquired with 240Hz, the experiment trajectories were linearly interpolated to get 2kHz data for the reference trajectories.

To be able to compare the agents reaction on perturbations to the subjects reaction in the experiment, the perturbed hops were organized in a separate list.

After using the data in training for some time, it has been decided to further clean the data. Considering the touchdown and takeoff velocities as well as the stance phase duration, all reference hops from the experiment deviating from the corresponding mean by more then the standard deviation were removed, resulting in a total of 49 unperturbed reference hops.

5.4 Deep Reinforcement Learning Implementation

From the beginning of the work on this thesis, it was planned to use the Proximal Policy Optimization Algorithm (Schulman et al., 2017), in its implementation from the OpenAI Baselines Library (Dhariwal et al., 2017). This decision promised to heavily reduce the implementation effort, as only the OpenAI Gym environment (Brockman et al., 2016) would need to be implemented, acting as an interface between the PPO algorithm and the MuJoCo model we built. This chapter presents the current state of our Gym environment implementation and points out some of the earlier attempts to achieve the targeted goal.

To meet all criteria of a full reinforcement learning setting, the GURO Gym environment has to allow an agent to input motor commands, in our case in form of desired motor torques, apply these to the environment and return a reward, the next state of the environment as well as the information, whether an episode's end is reached. All mentioned parts will be discussed

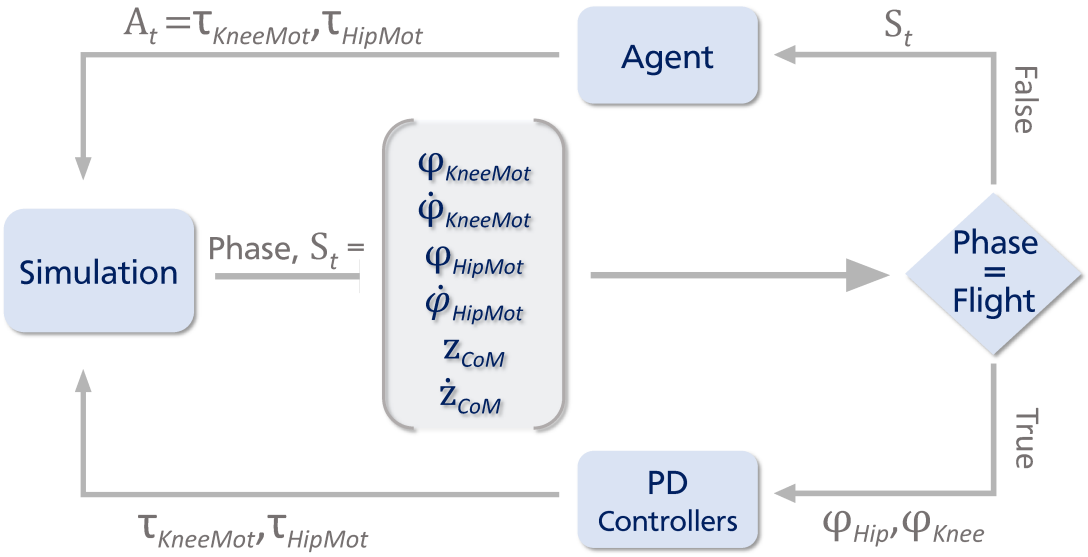


Figure 8: Diagram for our control framework. In flight phase, the robot is controlled by two PD position controllers, where in stance, the agent is taking over, receiving the environment's state S_t and outputs an action A_t.

below in their own subsection. Throughout this chapter, the term *GURO Gym environment* will be abbreviated as *environment*.

Before the individual components of the environment will be presented in detail, an overview over some of the used training approaches will be given. At the beginning the agent was trained to control both, the stance and the flight phase, and no reference data was used. The reward function was based on general assumptions about the hopping motion, like stopping the downward movement in compression phase and reaching a specified takeoff velocity during extension. The episode was always initialized at the same point in the flight phase with the knee and hip angles in their initial state of 40° and 100°. Later reference trajectories from the simulated robot were collected and used for training. It was possible now, to initialize the episodes at each point of the reference trajectory and implement stricter conditions for ending an episode. Finally, it was decided to concentrate training on the stance phase and use trajectories from human experiment, due to reasons described in section 4.1.1.

To monitor the environment during all implementation steps, a test-script was written, able to read out the state of the environment, plotting collected data and sending motor commands. The pseudo code of our algorithm on the next page gives an overview over the used training process.

Algorithm 1 Using DRL to learn human-like motions from reference data

```
1: Randomly initialize the weights of the value (critic) and policy (actor) networks
2: W_value, W_policy = random()
3: Train the agent for N steps
4: for stepTotal = 1, ..., N do
5:     Collect training experience for m = 2048 steps (Batch size)
6:     for step = 1, ..., m do
7:         Initialize new episode
8:         s = getState()
9:         refHop = getRandHop()
10:        hopPosition = getHopPosFromRSI()
11:        Train until episode's termination
12:        while not hasReachedTerminalCondition() do
13:            a = getActionFromPolicy(s)
14:            simulateOneStep(a)
15:            s' = getState()
16:            Simulate Flight phase without recording training experience and increasing step
17:            while in Flight Phase do
18:                hasEnteredFlightPhase = True
19:                a = getTorquesFromPDControllers(s)
20:                simulateOneStep(a)
21:                s' = getState()
22:            end while
23:            Continue training with new refHop if touchdown was detected
24:            if hasEnteredFlightPhase then
25:                refHop = getRefHopBasedOnTouchdownConditions(s')
26:                refHopPosition = 0
27:            end if
28:            r = calculateReward(s', refHop, hopPosition)
29:            Save training experience
30:            trainExp = (s, a, r, s')
31:            save trainExp in TrainExps
32:            hopPosition += 1
33:            step += 1
34:        end while
35:    end for
36:    Update neural networks
37:    W_value, W_policy = updateNNs(TrainExps)
38: end for
```

5.4.1 Environment's State

In the final implementation of the environment, the state consist of the angles and angular velocities of both motors as well as the vertical sledge position and velocity, being identical with the vertical position and velocity of the estimated COM of the robotic leg. The hip motor kinematics are identical to the hip joint kinematics due to the direct drive. The knee motor's angle and angular velocity match the movement of the knee motor pulley, which is linked to the knee joint movement via the rope transmission.

In previous iterations of the environment the states differed from the current implementation. Instead of the knee motor kinematics, the angle and angular velocity of the knee axis were used. Following the implementation of Peng et al. (2018a), a phase variable $\phi \in [0:1]$ was included in the state vector for a while as well. This variable, used only for learning cyclic movements like hopping, is 0 at the beginning of a motion and 1 at its end. In iterations, where also the flight phase was trained, the ground reaction forces were a part of the state vector. First as raw input, later filtered and at the end just as $+0.5$ or -0.5 to indicate whether ground contact is given or not.

As mentioned in section 3.2.1, the state vector has to be normalized before being inputted into a neural network. Therefore min-max normalization (Al Shalabi et al., 2006) was used during the first iterations. Min-max normalization maps a variable x with the range $[x_{min} : x_{max}]$ to the range $[-1:1]$ using simple linear scaling, displayed in the following equation:

$$x_{normalized} = \frac{2\,(x - x_{min})}{x_{max} - x_{min}} - 1 \tag{8}$$

As the next and final approach, the normalization was performed by using the mean $\mathrm{E}[x]$ and variance $\mathrm{Var}[x]$ of the input features, as proposed by LeCun et al. (2012). This way, all input features are transformed to have a zero mean and unified variance, that has to be proven to speed up progress in deep learning applications (LeCun et al., 2012). The equation is presented below.

$$x_{normalized} = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]}} \tag{9}$$

To approximate the mean and variance of each feature, the built in calculations of a running mean and variance of the PPO-implementation from the baseline-repository (Dhariwal et al., 2017) was used at the beginning. This approach however came up against a conflict with another important feature, that will be presented in the section 5.4.5 discussing the episode initialization. Since then, the mean and variance are approximated by using the reference data from human experiment, described in section 5.3.

The reference data does not include information about the speed and angle ranges of the knee motor during the hopping motion, which differs from the knee angles and angular velocities due to the rope transmission. Even we could estimate these ranges given the transmission ratio, another approach was followed to avoid errors coming from the rope elasticity. The reference knee and hip joint angles were used as target angles for PD position controllers to generate hopping in simulation, very close to the reference motion from the human experiment. The speed and angles of the knee motor were collected for 20 hops and used to get the mean and variance for the feature normalization.

Summarized, the agent has to learn interacting with an environment, described by a six dimensional continuous state space.

5.4.2 Actions

We would like to begin this section by reminding the general control strategy used in this work. The learning-based controller, simply put the agent, is only controlling the stance phase. In the flight phase, two PD position controllers bring the robotic leg in a desired posture, specified as having a knee angle of 142° and a hip angle of 17°. The best parameters for the flight phase controller were identified as $(P,D)_{flight,hip} = (0.26, 9.6)$ and $(P,D)_{flight,knee} = (0.18, 2.8)$ by observing the step response while the robot was hold in the air and different knee and hip angles near the desired flight phase posture were to follow.

Our agent can be trained to control only the knee or also the hip motor during stance phase. Therefore it outputs one or two target motor torques. This way, the same agent can directly be used on the real system as described in section 4.1, introducing the hardware system.

The torques outputted by the agent are clipped to match the maximum range of ±5Nm, given in the real motors. The hip motor torque is directly applied in the simulation. The target knee motor torque is first used to calculate the resulting torques in the knee motor and the knee axis under the influence of the rope transmission and is then applied to the corresponding motors in simulation. The implementation of the rope transmission is described in section 5.1.

When the agent is used to only control the knee motor, the target hip motor torque is set to zero during the stance phase.

To sum up, the agent is interacting with the environment by choosing actions from an up to 2 dimensional continuous action space, being the target torques for the hip and the knee motor.

5.4.3 Reward Function

The definition of the reward function has been altered a lot. After a simple reward function has showed first learning progress and this way proven all the features so far to have been implemented correctly, it was decided to follow the way of Peng et al. (2017) and use a reference trajectory for learning.

At this point it was unclear if the agent will be able to learn hopping from human reference data. For this reason a controller was hand tuned to perform a hopping motion, without the goal to make it human-like. It was assumed, that following trajectories, collected from the same simulation, should definitely be possible and definitely be easier. Therefore constant and over time linear torque profiles for each of the three hop phases were specified for both motors and hand-tuned, until the simulated robot showed stable enough hopping. Kinematic data of all joints were collected for three consecutive hops with similar amplitude and frequency and used as a reference during training.

With the usage of reference trajectories, the reward calculation almost took the final form. Since then, the reward function consists of three parts and its output is linearly scaled to the range of $[0 : 10]$, following the example of Dhariwal et al. (2017).

The first part $r_{trajecs}$ calculates a fraction of the reward, based on how close the reference trajectories are matched in simulation. As our goal is to replicate the human COM movement, only the vertical position and velocity of the sledge, as an approximation of the robots COM, are considered for reward calculation. The trajectory reward is then calculated as a weighted sum of

a reward for matching this position r_{pos} and velocity r_{vel}. The best weights have been identified to be $w_{pos} = 0.83$ and $w_{vel} = 0.17$. To calculate the reward of each individual trajectory, a subjective rating for allowed deviations was first chosen (e.g. a reward of 5 for zero deviation, 1 for 40% of maximum allowed deviation, etc.). An exponential function in the form displayed below, was then fitted to this data by finding best parameters (a, b, c):

$$r(t) = a + b\, e^{\,c\,|x_{sim}(t) - x_{ref}(t)|} \tag{10}$$

The identified parameters are $(a, b, c) = (-0.05624185, 5.056242, -0.8997435)$. By using an exponentially falling function, the agent is motivated to follow the trajectories as good as possible, as even small deviations are leading to a high decrease in reward. Figure 9 shows a plot of equation 10.

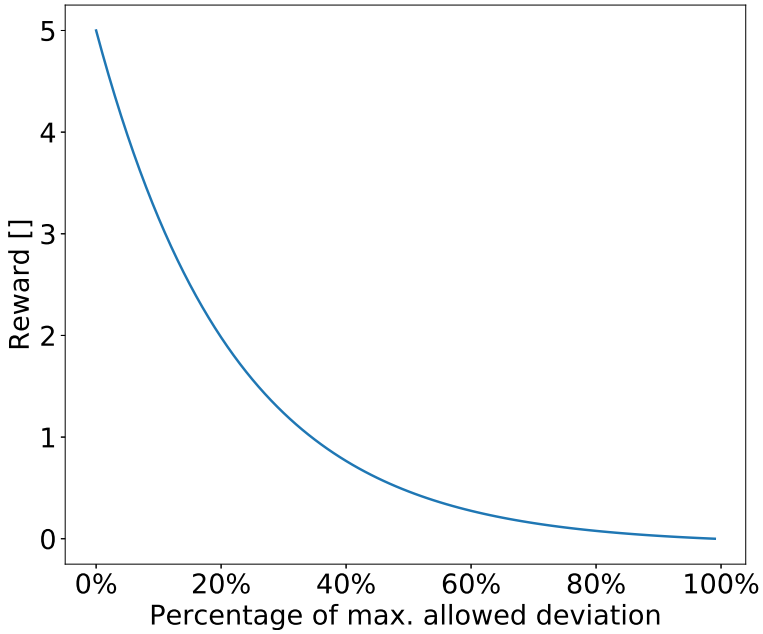


Figure 9: The shape of the reward function used for both components of the trajectory reward $r_{trajecs}$ and the foot position reward r_{foot}.

The second part of the reward function targets the foot positioning r_{foot}. A human hopping in place will land at about the same place after every hop. Furthermore the foot is not slipping on the floor. Analogous to the trajectory reward, the foot position uses equation 10 with zero deviation defined at $y = 0$.

Finally to reinforce hopping stability in a human-like manner, an additional reward is given, referred to as the *episode length reward* r_{eplen}. This part is linearly increasing with the episode length, the time the agent follows the reference trajectory without exceeding the maximum allowed deviations.

The final reward function is again a weighted sum of all reward parts as seen below:

$$r(t) = w_{trajecs}\, r_{trajecs}(t) + w_{foot}\, r_{foot}(t) + w_{eplen}\, r_{eplen}(t) \tag{11}$$

The weights are constant with $(w_{trajecs}, w_{foot}, w_{eplen}) = (0.35, 0.18, 0.47)$ but can be considered to be implicitly time-dependent, targeting for best imitation of the reference trajectories

and desired foot position at the training's beginning and hopping stability thereafter. This is due to the fact that the episode lengths at the beginning of a training are very short, starting at about 20 timesteps. Thus, the reward part r_{eplen}, which is proportional to the episode length, is almost zero. With time, when the agent learns to follow the trajectories for more timesteps, the episode length reward is growing and getting more important. To maximize its reward, the agent recognizes a second possibility, which is to longer follow the desired trajectory.

Before the reward function was based on reference trajectories, another reward calculation consisted of three simple parts, one for each of the three hopping phases. During the flight phase a positive reward for staying in the phase was returned, reduced by a value, proportional to the squared knee and hip angle deviations from a desired flight phase posture, being 40° for the hip and 100° for knee angle at that time. The goal of the compression phase was understood as stopping the vertical downward motion of the sledge. Thus, negative vertical sledge velocities were taken to the power of 2 and returned as a negative reward, with additional punishment for too high compression. Finally in extension phase, with the assumed goal of reaching highest possible takeoff velocity, positive velocities were rewarded. During training the simulation was rendered after each 100k steps to observe the current progress of the training (about every five minutes). At every observation, unwanted behavior was recognized. The reward function was then adjusted in order to avoid this behavior in future training. Indeed new unwanted behavior emerged after each adjustment.

5.4.4 Episode Termination

Early episode termination plays a crucial role in training DRL agents (Peng et al., 2018a). Besides speeding up the training, it controls the agent's observation space. During the first training approaches without the usage of reference trajectories, the episode was terminated based on simple time independent rules, like specifying ranges for joint angles that should not be exceeded. Also remaining in a state twice as long as a dynamic hopping motion requires it, led to the termination of an episode.

With the usage of reference trajectories, time dependent and stricter rules were specified. At a training's beginning, deviations of up to 10% are allowed, linearly increasing to up to 25% with training time to encourage stable hopping over perfect trajectory following.

Different levels of strictness were tested and stricter limits identified to be the best. With this, an additional challenge occurred, that required a change in the episode termination procedure. With the reward function consisting of several components, it was possible that an episode had to be stopped because of a deviation in the joint angle velocity, while other parameters still were matched quite well, thus leading to a high reward. To face this condition, a minimum episode time of 45 steps was introduced, equivalent to 10% of the reference hop.

Finally, as we focus on controlling the stance phase, the episode is also terminated, when the takeoff in the simulation happens more then 12ms earlier or later than the takeoff in the reference trajectory, corresponding to 5% of the mean stance phase duration. If the takeoff in simulation happens within the allowed time window, the training is paused and the flight phase is simulated until touchdown, without increasing the episode's duration. After touchdown the training is again resumed and the previous episode continues.

We assume that in order to be able to hop stably without perturbations, it is sufficient to be able to perform three consecutive hops without entering a terminal condition. Due to that

reason, the episode is also terminated, when the episode length of 1600 steps is reached, being long enough to perform three complete hops in a row within the allowed trajectory deviations. This way the agent cannot maximize it's episode reward by just learning to follow the trajectories good enough for more and more hops. Instead, after reaching the competence to perform three hops, it can only increase it's episode reward by better imitating the desired trajectories.

5.4.5 Episode Initialization

Another crucial component to learn complex behaviors with DRL was pointed out by Peng et al. (2018a), namely Reference State Initialization (RSI). It is only applicable to RL scenarios where reference trajectories are used. Our implementation of RSI and its extension by motion specific features is presented in this section.

At the beginning, when reference trajectories from the simulated robot were used, the RSI technique was utilized without changes. Each episode was initialized at a random point on the reference data. Using this approach and controlling the training progress every 250k timesteps, we made the observations that episodes initialized in the contraction phase were much shorter than such started in the extension phase. Because of this, the agent collected much more data from extension phase and was not able to learn to compress, wherefore a completely different motor torque curve was required then for extension.

To face this issue, the amount of states collected from each phase were counted. After an episode ended, a new state was randomly chosen from the phase with the least amount of observations so far. After this change, the agent was able to learn the whole hopping motion.

Most episode terminations now happened at touchdown and takeoff. Therefore when sampling a new state from contraction phase, a state near the touchdown was sampled with a higher (75%) probability, compared to the rest of the states in this phase. The same was implemented for the extension phase regarding takeoff. The higher probability of the critical state is linearly decreasing with time, until all states from a phase are sampled with the same chance. As training the agent with this settings resulted in good matching of the touchdown and takeoff conditions but poor trajectory following in the middle of the hop, a *midstance* phase was introduced, defined as a 10% interval around the middle of the reference hop.

To monitor the correct function of RSI, at each episode termination the current position on the reference trajectory is collected. This data is then plotted in a histogram together with the number of episode initializations in each phase of the reference motion. Figure 10 shows an examples of such a plot, showing the data after 6 million training steps.

As the last point regarding state initialization, the way of choosing one reference hop from the pool of all unperturbed hops will be discussed. When initializing a new episode after the termination of the previous one, the reference hop is randomly selected. When the agent takes off during training, the flight phase is simulated within the same episode step using PD controllers for the joint angles as presented in section 4.1.1. On touchdown, the position and velocity of the sledge are recorded. The next reference hop is then sampled by comparing these kinematic properties of the sledge with corresponding touchdown conditions of all hops. The similarity $s(hop_{i,ref}, pos_{td,sim}, vel_{td,sim})$ between the touchdown conditions are calculated as follows, assuming the velocity being more important at touchdown than the position:

$$s(hop_{i,ref}, pos_{td,sim}, vel_{td,sim}) = 5|vel_{td,ref}(hop_i) - vel_{td,sim}| + |pos_{td,ref}(hop_i) - pos_{td,sim}| \quad (12)$$

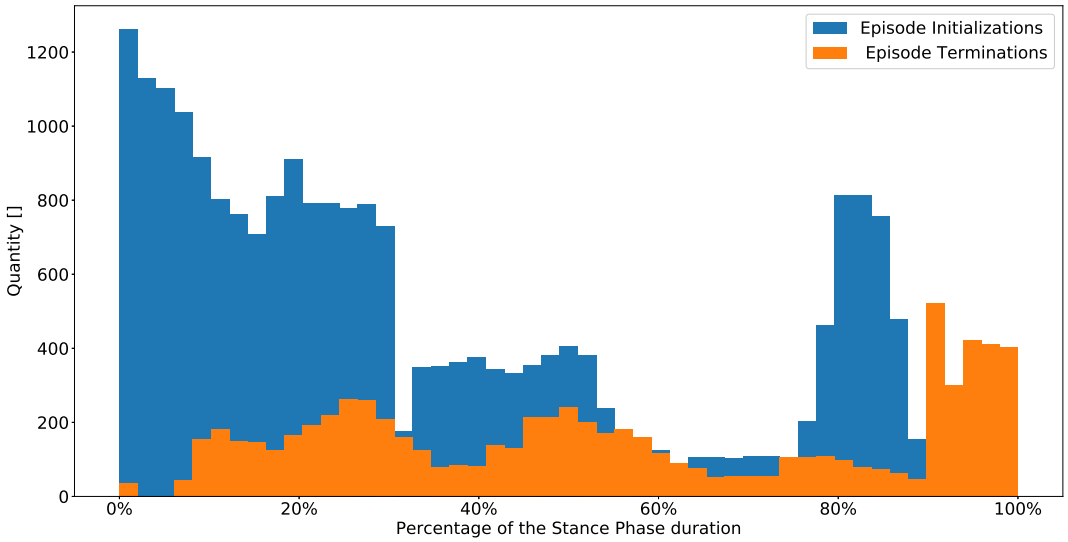


Figure 10: This histogram helps to monitor the success of ET and RSI by plotting the number of state initializations and state termination for each position on the stance phase trajectory. Here we can see, that most episode termination occur around the takeoff, where the most episodes are initialized during the compression phase, being the hard part of the hopping motion to learn.

5.4.6 Hyperparameters

Closing the implementation section, the used hyperparameters in the PPO implementation will be presented.

To begin with, figure 11 illustrates the used network architecture, consisting of a six unit input layer, two hidden layers with 64 units and ReLU activation function and the output layer with one or two units based on how motors are to control. This same architecture was used for both, the actor and the critic network.

The learning rate $lr(t)$ was chosen to be time dependent, described by the following equation, where $g(t)$ is a linear function starting at 1 at the training's start and 0 at it's end:

$$lr(t) = \begin{cases} f(t) & if\ f(t) \leq 10^{-7} \\ 10^{-7} & if\ f(t) > 10^{-7} \end{cases} \tag{13}$$

$$with\ f(t) = 3 \times 10^{-5} \cdot [g(t)]^7$$

This function was a result of several iterations starting from $3 \times 10^{-4} \cdot f(t)$ proposed by Dhariwal et al. (2017). The fast reduction of the learning rate over time was necessary to increase the level of imitation, as the agent learns to hop quickly, but needs much more time to make the hopping stable and human-like. The clipping of the function to a minimal learning rate of 10^{-7} prevents the network updates to be unsignificantly small after about half of the training time of 8 million steps.

All other hyperparameters were chosen as proposed in the PPO implementation of Dhariwal et al. (2017).

6 Results

The aim of this study was to investigate the question, whether learning-based model-free control approaches utilizing deep reinforcement learning (DRL) techniques can generate stable human-like dynamic motions in robotic applications. As an example, hopping in place was investigated in a realistic simulation of the GURO Hopper test bed, which is a two segmented robotic leg, actuated by two motors. In case of a positive answer to this question, best practices and recommendations were to be formulated for using this approach.

Within this work we succeeded in using DRL approaches to implement a robust model-free feed-forward torque controller for generating human-like hopping motions in a realistic simulation of the GURO Hopper robotic leg. The agent controls the stance phase behavior, where in the flight phase the desired touchdown knee and hip angles are achieved utilizing PD position controllers for both motors. The learned stance phase controller produces stable hopping when sampled with 1kHz and 2kHz. The same agent can be trained to generate hopping by controlling both motors and the knee motor only. Figure 12 shows six consecutive snapshots of the hopping motion in simulation.

The simulation is running at 2kHz using the MuJoCo physics engine. The simulation model of the GURO Hopper robotic leg considers the torque ranges of the two brushless DC motors, the precision of the sensors and includes further realistic parameters derived from a system identification experiment.

It is important to note, that our DRL agent haven't needed any information about the robots morphology, it's inertial properties as well as sensor and motor ranges.

The agent was trained using reference trajectories for its COM vertical position and velocity, derived from motion capture data of a human hopping experiment. Reference trajectories for the knee and hip angles coming from the same experiment, were used to normalize the input features and limit the maximum divergence from human-like behavior during training. We further showed the effectiveness of using this approach compared to reference-free training and the necessity to clean the reference trajectories with regard to relevant aspects of the target motion. In case of the hopping motion, these were the touchdown and takeoff kinematic conditions as well as the stance phase duration. Using the motor position and speed was found to result

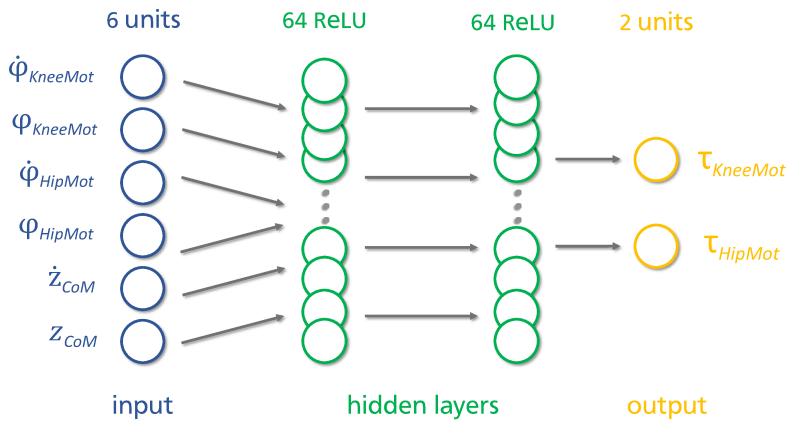


Figure 11: The architecture we used for the actor and critic networks.

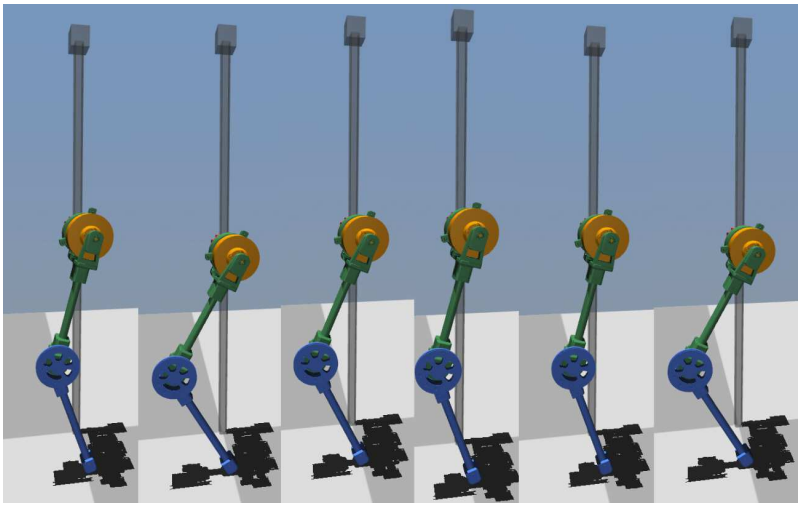


Figure 12: Snapshots of the simulated robotic leg during hopping.

in better training progress compared to inputting the joint angles and angular velocities as the environment's state.

The agent's performance during training was measured by the episode reward. Beside that, the mean episode duration over the last 40 episodes was monitored. The learning curves are presented in Figure 13. To produce stable human-like hopping, up to 8 million samples were required to be collected during training, taking about 6 hours using one 1.8GHz core of an Intel i7-4500U CPU. GPU acceleration for simulation or updating the neural networks was not used.

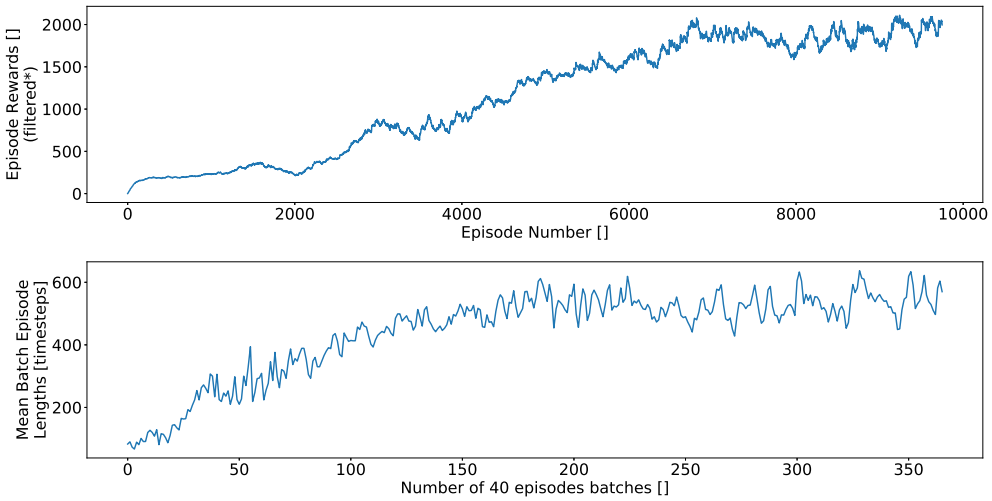
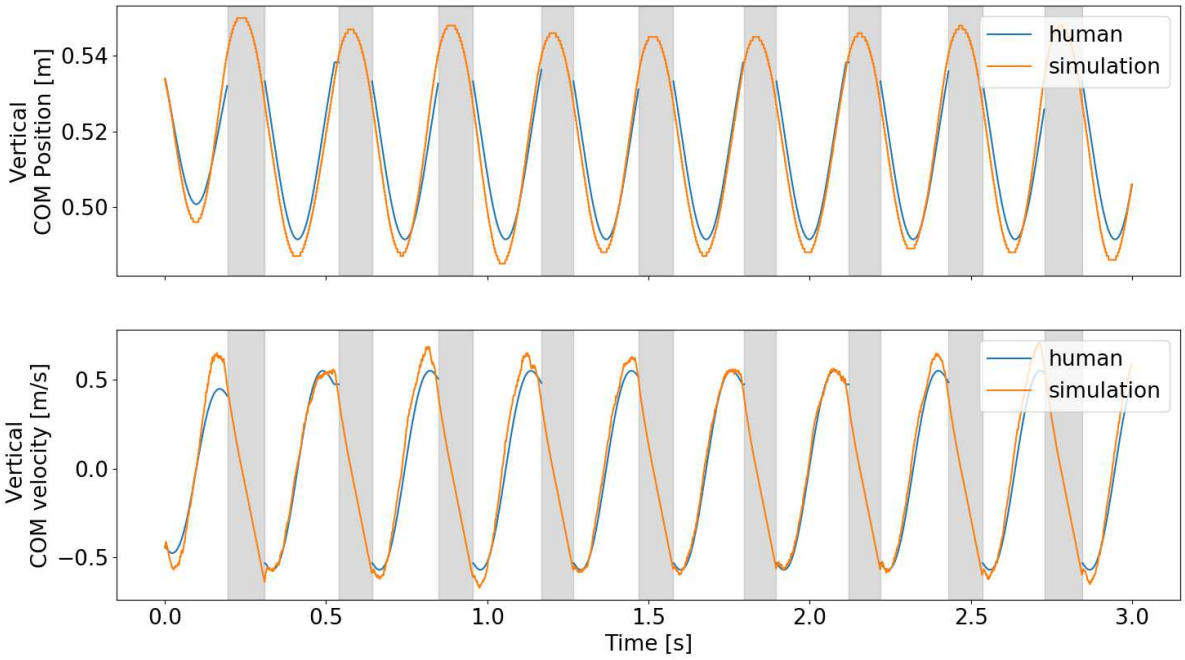


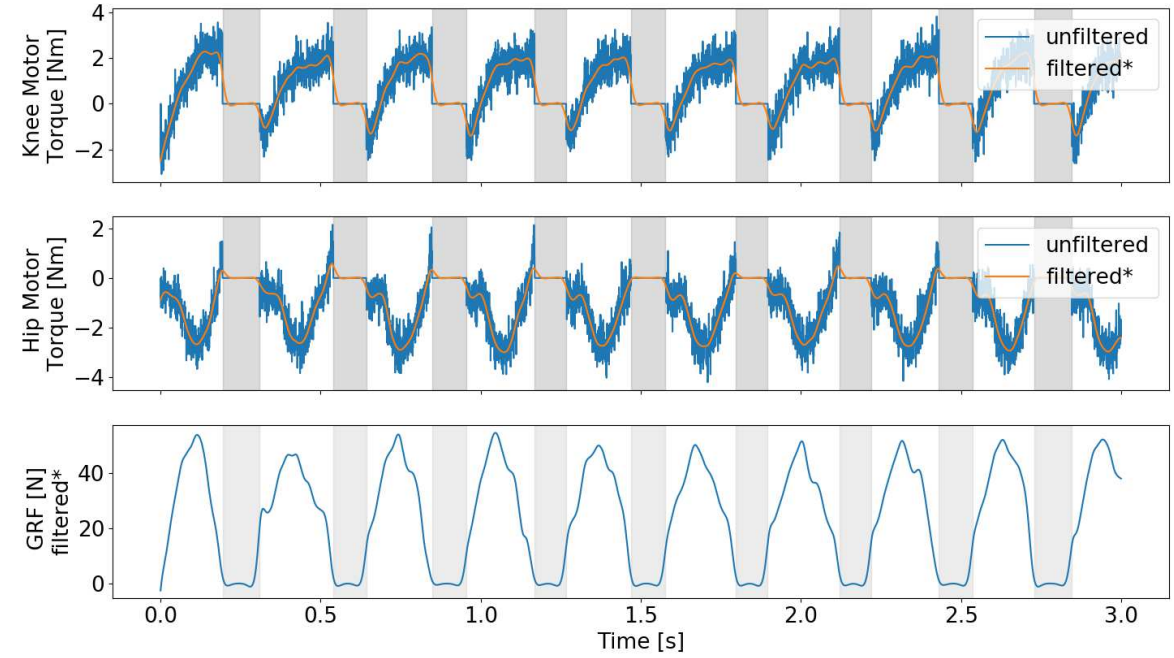
Figure 13: Learning curves of our agent after 6 million steps, shortly before saturation. The upper curve shows the reward during one episode, the lower one the mean episode duration of last 40 episodes. *: Filtered with an exponential running smoothing using a smoothing factor of 0.25.

The reference COM position trajectory is followed with a maximum deviation of 14% percent, the COM velocity trajectory with maximum 10%, when neglecting the deviation of the first hop showing 20% one single time in ten consecutive hops. Figures 14a and 14b illustrate these results.

The learned hopping motion is stable. The robot is able to hop for 100 hops in a row maintaining the same hopping frequency and same COM amplitudes, deviations for single hops of



(a) Comparison of the vertical COM position and velocity between the human (blue) and the simulated robot (orange). The used human reference data lacks recordings during the flight phase (shaded area).



(b) Motor torques outputted by the learned controller and GRFs, simultaneously collected with the COM kinematics from Figure 14a. As the learned controller is only sampled during stance phase, the motor torques during flight phase (shaded area) are zero. *: Filtered with a second order butterworth low pass filter with 20Hz cutoff frequency.

Figure 14: Comparison of COM kinematics between the human and simulation (a) and corresponding kinetics in simulation (b). Shaded areas indicate the flight phase.

up to 15% allowed. Figure 17 compares the means and standard deviations of 100 unperturbed and 8 perturbed hops from simulation and the human experiment, showing that our controller shows less variance in it's hopping behavior over time in contrast to the human. Moreover, the controller is robust against perturbations in form of a ground level change of up to 27cm, which is corresponding to 50% of the robots leg length. Please note, that perturbations were not included during training. Figure 15 compares the robot's and human's (scaled down) trajectories in the setting of a sudden ground level change of 8% during the third flight phase. It can be clearly seen that the robot shows a human-like perturbation reaction, following the human COM position trajectory during the perturbed hop and the hops after.

Moreover, we used the exact same algorithm to learn a controller, able to generate the same hopping motion with a 50% heavier hip or to follow another trajectory, derived from the available one by stretching it in time by 20%.

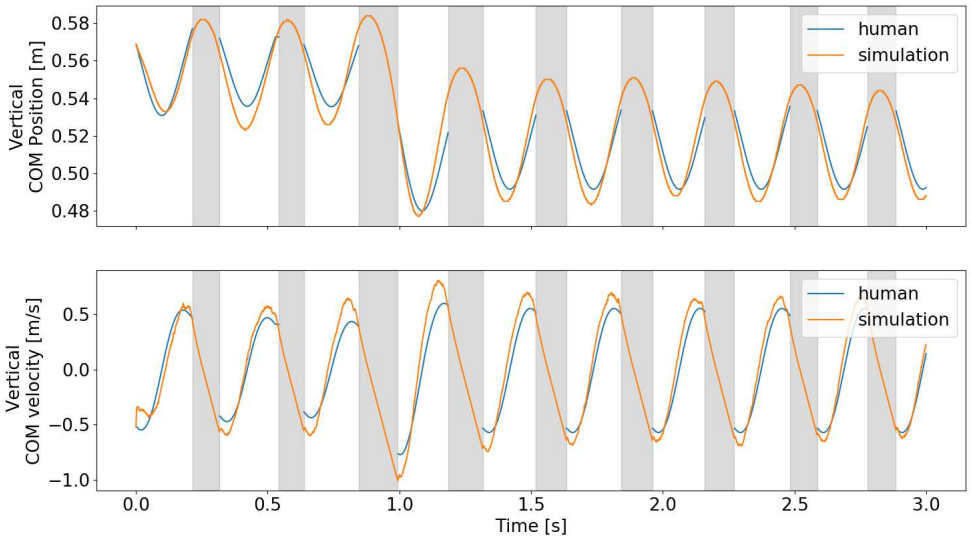
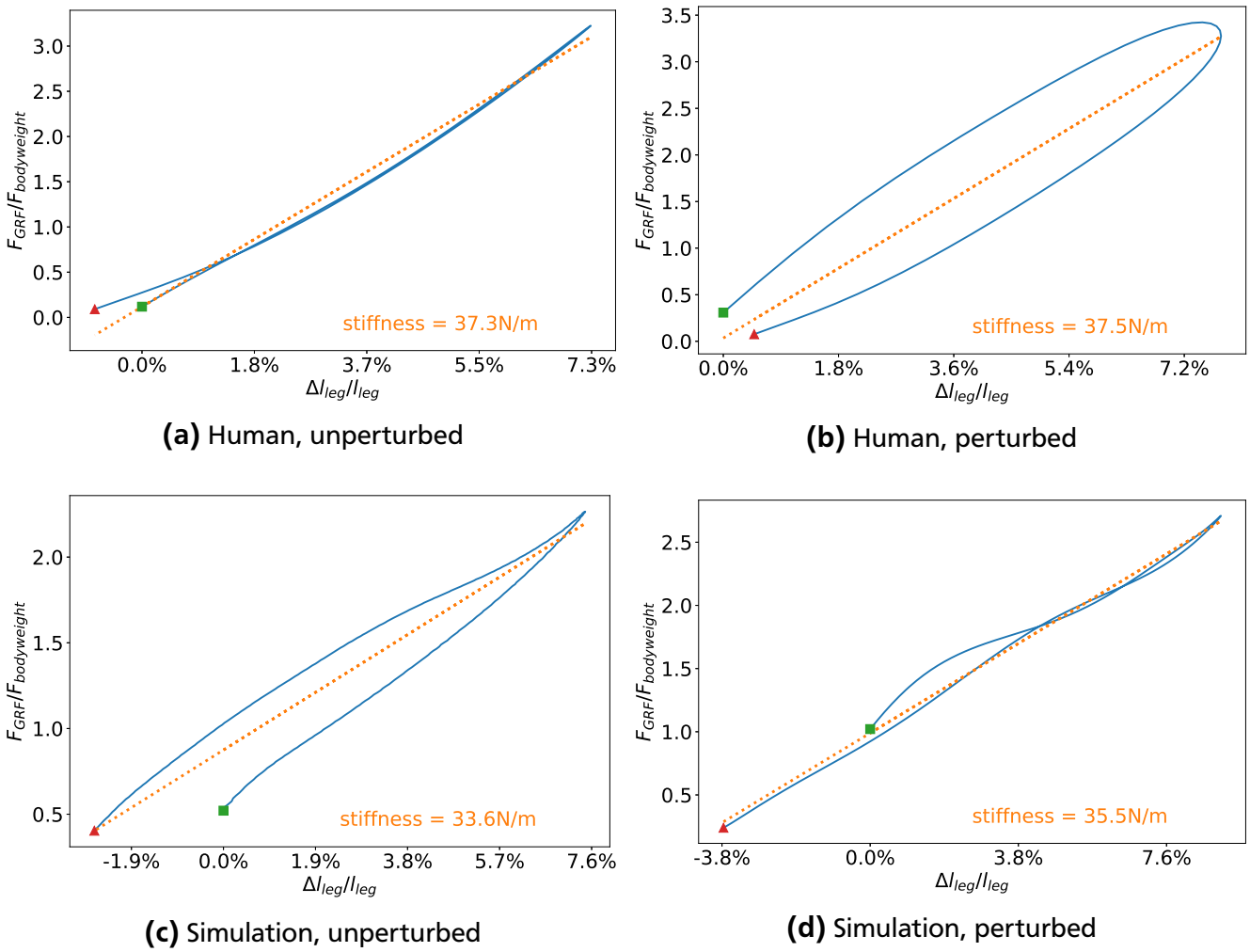


Figure 15: Comparison of vertical COM position and velocity between the human experiment and the simulation, including a ground level change of 8% during the third flight phase. All flight phases are shaded grey. Human reference data during flight is not available.

Without providing the information of the leg stiffness during training and its change as a perturbation reaction, the agent interestingly learned to mimic the human leg stiffness during stance phase of unperturbed and perturbed hops. Figure **16** displays the force-length curve of the robot and human for both cases, with the leg stiffness estimated by fitting a linear function to the graphs. Here, the mean data of 100 hops is displayed and the GRFs were filtered by a second order butterworth filter with 20Hz cutoff frequency. Even the graphs show differently shaped curves, the stiffness differs by 10% for unperturbed and 5% for the perturbed hop. Notably is also the fact, that in both cases the leg stiffness change as a perturbation reaction remains below 5%.

In addition it was shown, that imitating a human motion with DRL is most efficient with the usage of reference trajectories and requires implementation of Early Termination (ET, section 5.4.4) and Reference State Initialization (RSI, section 5.4.5). Softening the termination conditions with training time up to a certain limit, was identified as an important requirement for the learned motion to be stable. By extending RSI with motion specific features, the learning speed



(a) Human, unperturbed

(b) Human, perturbed

(c) Simulation, unperturbed

(d) Simulation, perturbed

Figure 16: Comparison of human and simulation leg stiffnesses for perturbed and unperturbed hops. The green square marks the touchdown, the red triangle the takeoff. The leg compression is normalized by leg length and the GRFs by the body weight.

and final results of a training could be significantly improved. To monitor the success of RSI and ET, a histogram plot was introduced, showing the number of episodes initialized and terminated at each timestep of the reference motion. An example of this plot is displayed in figure 10.

Furthermore, the normalization of input features by using mean and variance, derived from the reference motion, was identified as being more effective compared to normalization by using the minimum and maximum values of each component of the state vector. Reference trajectories for motor kinematics, were collected by using PD position controllers to mimic the reference angle trajectories (Section 5.4.1).

When learning a motion from a reference trajectory, this work revealed that the reward for imitating position trajectories should be weighted significantly higher than that of velocities. The best results were achieved by using a weight of $w_{pos} = 0.83$ for position and $w_{v\,el} = 0.17$ for velocity imitation in the reward function and allowing a maximum deviation of the position by 20% and velocity by 40% during training.

Moreover it has been demonstrated that using a clipped episode length reward in combination with the imitation reward, improves the stability of the learned motion and make it more human-like (Section 5.4.3).

Sampling the learned controller with a frequency below 1kHz, when feed forward torque control is applied, does not lead to stable hopping. Training at smaller sampling frequencies shows a progress, which is however not sufficient to learn the desired motion within 8 million timesteps.

In order to achieve the presented results, we used an adopted version of the PPO algorithm implementation from the *OpenAI Baselines* repository (Section 4.2) and created an *OpenAI Gym Environment* for our MuJoCo simulation (Section 5.4). For both, the actor and the critic network, the learning rate has been found to be the hyperparamter that needed the most significant change with respect to the default values. We decreased the learning rate in an exponential way with regard to the training time (Section 5.4.6).

7 Discussion

During the discussion, the results and implementation of our learning-based controller will be first compared with traditional control approaches. Next, similarities and differences to previously used DRL approaches will be presented and reasoned. Finally, further results will be discussed and possible interpretations will be given.

7.1 Comparison with traditional control approaches

Several traditional model-based control approaches were applied to replicate the hopping motion in legged robots in prior works (Sayyad et al., 2007; Kalveram et al., 2010). The hopping motion is usually represented by a spring-mass model (Blickhan, 1989) or a spring loaded inverted pendulum (SLIP) (Oehlke et al., 2016). In a current paper, Oehlke et al. (2018) compared feed-forward and feedback control approaches in a two segmented robotic leg using the latter model. The robot's motion was also limited to 1 DOF hopping by a vertical linear guide system. Moreover, the control was also limited to the stance phase and applied in a simulation before using it on the real hardware, making it comparable with our robot.

As stated by the authors, the feed-forward control, specifying a sinusoidal motor current trajectory, requires a precise identification of the systems dynamics and losses, which are non-linear and position-dependent. All these steps are not required in our model-free learning-based approach. During training the agent learns to estimate the non-linearities on its own as well as reacting to them in the right way. We furthermore did not have to analyze the equations of motion and provide the agent with information about inertia or shape of the robot. Moreover the feed-forward approach is highly dependent on initial conditions and is very unlikely applicable in a more complex locomotion task then vertical 1 DOF hopping with one motor. On the contrary, the DRL approach was already been shown effective to achieve bipedal locomotion on a 20 DOF robot, actuated by 10 motors (Xie et al., 2018).

In their other approach, Oehlke et al. (2018) modeled the robotic leg by a spring loaded inverted pendulum, having the complete body mass concentrated at the COM. Surprisingly, despite this major simplification the authors achieved stable hopping. Recovery from ground level perturbations are also reported. Indeed, a different comparison with the human is drawn, focusing on high level parameters like hopping frequency instead of trajectory similarity. The relative hip displacement, that can be compared to our COM trajectory, was varying up to 30% from the human, where we have a maximum deviation of 14 % during midstance, which is much smaller elsewhere.

The biggest advantage of our approach is it's quick adaptability to changes in actuation. By only changing few lines of code and taking four hours for training on a single processor core, the same algorithm learned to generate the same hopping motion by using two motors, an additional one for hip actuation, instead of just one knee motor. Traditionally, the extension to a second motor requires significant and time consuming changes in the feedback control and almost starting from scratch, when feed-forward control is to be applied. Because of the presented observations, we expect to be able to quickly learn to control three motors without big effort, when a foot together with an actuated ankle joint is added. Furthermore we have tested the exact same algorithm, to learn mimicking the same reference trajectory with a 50% heavier hip and to imitate another trajectory, derived by scaling the available trajectory by 15% in time. The learning curves showed a rising trend, the robot was able to perform the hopping

motion after 6 million steps, even when the hopping was not stable. These observations still support our expectation, that by replacing the linear guide with a boom and providing the agent with an appropriate reference trajectory, little extra effort will be required to learn a controller for generating 2D hopping. To our knowledge, no traditional control approach can show such kind of capability.

A feature, our controller cannot offer yet, presented in the feedback controller by (Oehlke et al., 2018), is an adjustment of the height and amplitude during hopping, closer discussed in the Limitations, where a solution is also suggested.

To sum up, feed-forward and feedback model-based control approaches permit strong simplification and can achieve stable hopping. They are indeed not easy to adjust to changes in actuation and have to be specifically defined for each desired motion and each robot. Modeling effort strongly rises with the complexity of the system and target motions, unlike in model-free and learning based approaches. Schumacher and Seyfarth (2017) mention in their work that it is still unclear how humans control their motions, which can be seen as an other argument in favor of model-free control approaches.

7.2 Comparison with learning-based approaches

To put our work in the context of similar approaches to develop learning-based model-free controllers in the literature, our results and implementation details are compared to Peng et al. (2017, 2018a) and Xie et al. (2018). First of all, main differences are explained, followed by a comparison of the corresponding implementation. In addition, possible interpretations are given.

To begin with, our study has confirmed previous research showing that DRL techniques can be applied in order to develop robust model-free controllers for generating dynamic motions (Schulman et al., 2017; Peng et al., 2018a; Xie et al., 2018). It is important to mention that these papers worked with much higher state and action spaces. On the other hand, the control approach, presented in this work, was inspired and highly guided by this examples and thus has to be compared with them. Moreover, by targeting at closely imitating human hopping in a realistic simulation environment and aiming at the agent to directly control the motor torques instead of outputting target angles for PD position controllers, we had to overcome challenges not presented in the referenced works, making the comparison more fair.

Our controller generates human-like COM movement with a maximum deviation of 14% in position and 10-20% in velocity, as displayed in figure 14a. Therefore the agent gets sensory input with the precision given in the real hardware system and outputs target motor torques lying in the possible torque ranges of the motors mounted on the robot. Learned motions on a simulated humanoid presented by Schulman et al. (2016, 2017) were a groundbreaking achievement, hence they cannot be considered as human-like, which is easy to recognize by watching the supplemental video material of the papers.

Truly human-like motions on a similar humanoid model were achieved by Peng et al. (2017, 2018a) by using reference trajectories from human motion capturing data. However, these papers are concentrating on character animation and therefore not paying much attention on designing the models in a way that they could be build in reality. All joints except of the knee and elbow, being one DOF revolution joints, are modeled as spherical joints actuated by three virtual motors, having no geometry and no inertial properties. Same also applies to the Atlas

robot, with the exception of the motors being modeled as one rigid part matching their outer shape, having a realistic mass and being placed at their actual position. The motor torque limits selected by Peng et al. (2017) were ranging from 90Nm for the ankle joint to 150Nm for the knee and 200Nm for the hip joint. Achieving this high torques in reality would require strong motors and high transmission ratios, resulting in a significant increase in the robots inertia considering the actuation of more than 30 degrees of freedom. Moreover, the controller in all referenced works does not directly control the motor torques, but instead outputs target joint angles, which are then followed by using PD controllers. Our agent is directly outputting motor torques, thus targeting a significantly harder task, as it has to implicitly learn the rules of linear control theory on it's own.

The Cassie robot, controlled with DRL by Xie et al. (2018), had a very realistic simulation model, also using the MuJoCo engine for it. Even so, the agent was trained with a reference trajectory collected from the robot itself using a previous controller. More importantly to point out is the fact, that the DRL agent from Xie et al. (2018) neither has directly controlled the robot's motor torques, nor outputted target angles for PD position controllers. Instead, their neural network outputs were added to the joint angles from the reference trajectories, which then were forwarded to PD position controllers. We have also started by mimicking trajectories collected from the simulated robot, to be sure to have trajectories that definitely can be followed by the robot. This setting reduced the training time by about two times, indicating that learning trajectories from human reference data is harder to accomplish. This observation can be explained by the different morphology and very different actuation in the human and the robot.

The smaller network size used in our controller (two 64 units hidden layers) compared to other presented results (256 to 1024 units in two hidden layers), can certainly be explained by the fact, that our state and action space is significantly smaller then in other works. The same applies to the complexity of the target motions.

Another notable change made in comparison to referenced works considers the used learning rate, also referred to as the stepsize. Schulman et al. (2017) and Peng et al. (2018a) used constant learning rates for both actor and critic networks in the magnitude of 10^{-4} and 10^{-5}. Xie et al. (2018) linearly decreased the stepsize with each iteration starting from an order of 10^{-3} and ending at 10^{-4}. In our approach we found out that using an exponentially decreasing stepsize over training time, starting from the same magnitude, improves the imitation of the reference trajectory. It can be explained in the way, that a higher learning rate at the beginning allows to change the network parameters in bigger steps to first learn imitating the hopping motion in general, where to further improve the mimicking, fine tuning by smaller learning rates is necessary.

The last significant difference between our and previous implementations to be discussed in this section, was motivated by findings of Peng and van de Panne (2017). It was already shortly mentioned at the beginning of this section, but deserves a more closer look. Comparing different action spaces like torque or muscle activation, Peng and van de Panne (2017) have shown, that using target angles for PD controllers as the network's output improves performance and learning speed. All presented learning-based approaches in this section so far followed this recommendation. Peng et al. (2017, 2018a) in particular, used the stable PD (SPD) controller, specifically developed for the character simulation field by Tan et al. (2011). Given that, Peng et al. (2017, 2018a) sampled their learned controller with 30Hz for the next angle to be reached by the PD controllers running at the speed of simulation with 3kHz and 1.25kHz. Xie et al.

(2018) used a self tuned traditional PD controller executed at 2kHz. On the contrary, our agent directly learned a mapping from states to motor torques, making the task more challenging and explaining the need to sample our policy with at least 1kHz as well as the fact, that better performance is achieved, when running our controller with 2kHz. Training with lower control frequencies has not resulted in stable hopping, which is acceptable given the fact, that the high level controller of the real GURO robot is running with a frequency of 1kHz.

7.3 Possible result interpretations

Hereafter follows a discussion of further findings, including possible interpretations.

Our agent can be trained to control both motors or the knee motor only. When the reference motion is followed with one motor, slightly higher final deviations from the reference trajectories are observed, especially during midstance. In addition, using also the hip motor during stance phase, helps preventing the foot from slipping on the ground. Surprisingly, the hip motor was heavily involved in controlling the hopping motion (Figure 14b), where we expected it to not play a significant role. The agent's outputs for the hip motor were in the range of [-4:4] Nm, which covers 80% of the possible range. The desired knee motor torque however was reduced a bit, indicating that part of the required energy for upwards accelerating the COM was now delivered by the hip motor.

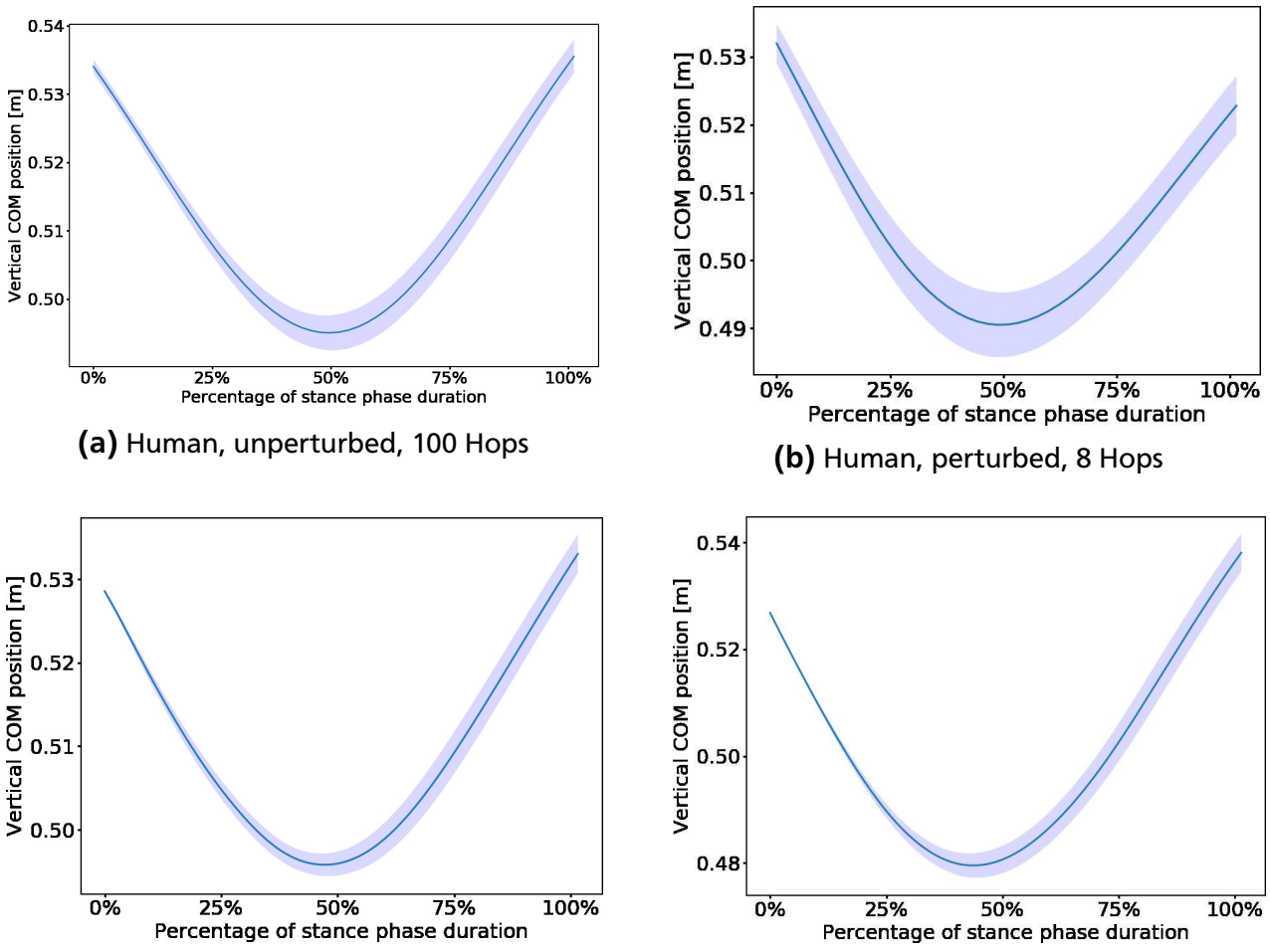
Despite the differences of up to 14% in the robot's and human's COM trajectories, the simulated motion of the robot during stance phase can clearly be considered to be human-like as defined in section 4.1.1. Achieving a higher similarity might be not possible due to morphological differences between the robotic leg and the human body. Interestigly, when investigating figure 17, displaying the mean and variance of several hops, we clearly see that our agent generate less variant hopping compared to human. This could indeed be explained by the fact, that human hops were collected over time and the subject could get tired or bored, where for the simulated robot the time does not influence it's performance. Another reason, that might have led to the smaller variance in human hopping is the fact, that we trained the agent with reference data, where hops deviation too much from the mean regarding the takeoff and touchdown conditions as well as the stance phase duration were filtered out to simplify training.

Another important fact to be discussed in this context is furthermore the robot's hopping frequency of about 2.9Hz, which is definitely higher compared to humans. As our reference data only includes stance phase data, the hopping frequency of the subject has to be estimated by using the takeoff velocity. With a mean takeoff velocity of 1m/s, the flight phase duration is estimated to 204ms and the hopping frequency of the human, given a mean stance phase duration of 223ms, thus equals 2.3Hz. As for our reference trajectory we have scaled down the human COM position trajectory with a scaling factor near to 0.5, the COM velocity trajectory of the robot compared to the human was also scaled by the same factor. With a 50% smaller takeoff velocity, the flight phase of the simulated robot is consequently about half as short as the flight phase of the human, reducing the time for one hop in simulation by about 100ms or 23%, explaining the about 20% higher hopping frequency in simulation.

Interestingly, the training, only including trajectories from unperturbed hops, led to a human-like reaction on perturbations in form of ground drops with up to 8% of the robots leg length. Moreover, the agent was able to recover from perturbation with up to 50% of ground level changes. As we do not have reference data from human experiment for this amplitudes in the ground level change, we cannot evaluate this behavior as human-like. This result could mean,

that the agent has learned the underlying mechanisms of controlling the robotic leg in a human-like manner, but is more probably explained to come from allowed explorations of states around the reference trajectories. The second assertion is supported by the fact that hopping, learned while only small deviations (up to 10% of the whole motion range of the movement) from the reference trajectories were allowed, was less stable compared to learning allowing 25% difference.

Indeed, two other observations support the idea, the agent could learn the underlying control principles of a movement. Firstly, When the agent was trained to follow a trajectory extracted from the robot itself, he learned to output the same torque profiles used to generate the reference motion, disregarding the noise in the agent's output. Secondly, as presented in Figure **16**, the agent learned to mimic the human leg stiffness without having any kinetic information from the human experiment. The different shapes of the force length curves are most probably explained by the differences in the body form and mass distribution. This findings give rise to a little hope, that DRL could be used to learn more about human motor control by building realistic simulation models and training them to follow reference trajectories from human experiments. However, it must be mentioned at this point that with increasing numbers of degrees of freedom and actuators, it becomes more likely, that multiple solutions exist to accomplish a task. The agent



(a) Human, unperturbed, 100 Hops

(b) Human, perturbed, 8 Hops

(c) Simulation, unperturbed, 100 Hops

(d) Simulation, perturbed, 8 Hops

Figure 17: Comparison of human (scaled down) and simulation mean and standard deviation of the leg length over 8 perturbed and 100 unperturbed hops.

might just learn one of the possible solutions, which does not necessarily have to be the one used by humans.

Curiously, COM reference trajectories derived from a motion performed by a three segmented human leg could be imitated with a two-segmented robotic leg. The explanaition of this comparison being fair can be read in section 4.1.1. Extending the model by an additional foot connected via an ankle joint and retraining the model is expected to require little effort, but to offer new research possibilities.

Higher rewards for mimicking the COM position then for imitating the COM velocity leading to better training results, certainly come from the fact that both quantities are highly correlated, the velocity being the rate of the position change.

Having a less stable controller with less positive trajectory matching, when trained and sampled with frequencies of 1kHz and less, met our expectations based on the frequencies of 1.25kHz - 3kHz used for the PD position controllers by Peng et al. (2017, 2018a) and Xie et al. (2018).

The origin of the high oscillations in the agent's torque outputs is probably to be found in the complexity of the neural networks, mapping the states to the actions, and the non-linearities in the system. Applying a running exponential smoothing filter during training, made learning unpractically slow and not successful as a whole. Using the same approach to filter the agent's output after training, led to increasing deviations from the reference trajectory, proportional to the degree of filtering. This was expected, as filtering introduces a delay of the control signal and thus a shift in the mapping between states and actions.

The importance of ET and RSI have already been discussed in the Methods. Here we can add, that too strict definition of terminal conditions, such as a maximum deviation of 5% from the desired position trajectory, makes the learning results less stable and sometimes even impossible. This can be explained as not satisfying the need of exploration on the one hand. On the other hand, the exact same motion could not be possible to be executed on a robot with a body, shaped differently from the human. It has been shown, that a motion specific adjustment of the RSI can further improve learning speed and results. The RSI suggested by Peng et al. (2018a) is based on a random selection of states for each new episode initialization. When the motion indeed can be divided into phases and these phases have areas of different complexity levels, we have shown that initializing more episodes within the complex areas of a phase with a higher probability at the training's beginning, is beneficial for learning speed and robustness of the learned controller. As the episodes, initialized in complex areas of the target motion are most often terminated after a significantly shorter duration, the network is still receiving the same amount of states from all areas of a phase. To be sure, that also the phases are equally represented in the observations, we further suggest to count the number of states observed in each phase and initialize the next episode within the phase with the minimal amount of observations so far. To monitor the quality of RSI and ET, we propose to track the number of episode terminations and initialization in each phase and point of the reference trajectories as seen in Figure 10.

7.4 Limitations

Within this work we aimed at investigating the question, whether DRL can be used to obtain model-free controllers to generate dynamic locomotion tasks in real robotic systems. To stay in the scope of a Bachelor thesis, we limited our research to a simulation of a single robotic leg using one reference motions. Due to different mass distributions between the GURO Hopper robotic leg and the human body, we limited the learning-based control of the robot to the stance phase and tuned PD position controllers for controlling the flight phase posture.

For testing the robustness of the learned controller, only one kind of perturbation, even when with different heights, was considered. Perturbations during midstance, which are also contained in the reference data, could provide better judgment regarding the motion similarity with the human. Also comparing the COM movement produced by two three-segmented human legs, with the corresponding trajectories from the single two-segmented robotic leg, required some limiting assumptions to be fair, as described in section 4.1.1.

During the identification of the ground properties as discussed in section 5.2, optimal simulation parameters to replicate the experiment very closely could be identified, but resulted in a too bouncy ground, that could not be used. Using another cost function to be minimized by the optimal parameters could solve this issue.

A limitation of our controller is the lack of a possibility to change the height or frequency during hopping, which can easily be accomplished by the human. To overcome this boundary, the agent would need to be trained with trajectories collected from human hopping with different frequencies and amplitudes and having the desired motion features included in the state vector.

As explained in section 5.3, the reference data was cleaned by removing outliers, defined as having a deviation of more then the standard deviation from the corresponding mean. This procedure showed an improvement in the learning process. Indeed, when aiming at learning a human-like movement and especially when targeting at learning more about human control by using DRL, this outliers might be especially important and therefore should be considered during the training. In addition, reference trajectories for the knee motor could not be derived from the human reference data to calculate the mean and variance required for the feature normalization. For this reason, we collected the knee motor position and speed trajectories by letting PD position controllers mimic the knee and hip joint angles from the reference motion for several hops. The agent indeed, most probably will not control the knee motor the same way a PD controller did it and the normalization by mean and variance will not be correct in this case. This procedure has not shown a significant influence in our case, but should be solved when our approach is used with more complex systems.

Our agent learns to perform the hopping motion by only getting kinematic data as input. It might be useful to also include kinetic parameters like motor torques, ground reaction forces or the set current, which has to be investigated in further research.

Furthermore, all presented hyperparameters of the PPO algorithm as well as the weights in the reward function and maximum allowed deviations from reference trajectories were altered by observing the learned behavior, the agent's learning curves and comparing a few educated guessed parameters from all the possible parameter space with each other. Here, a systematic grid search approach or, more professional but also costlier, the usage of Bayesian Optimization as proposed by Snoek et al. (2012) could help to identify the truly best parameters.

7.5 Further work

The obvious next steps would be using the controller trained in simulation to generate hopping motions on the real robot. Before doing so, a technique called *dynamics randomization* proposed by Peng et al. (2018b) would need to be added to the current algorithm, simplifying the transition from the imperfect simulation to the real world. Besides that, training the controller to be sampled with 1kHz, that would be required on the real system, might need further improvements.

If the simulation turns out as not realistic enough and dynamics randomization will not be sufficient to use the controller from simulation on the real robot, the environment can be further improved. The most recent version of the MuJoCo physics engine allows implementing the rope transmission directly in the model. Moreover further identification experiments can be conducted to find realistic estimations of joint friction and damping. The ground properties can be found by using a different cost function from the already collected and processed data.

If the simulation as expected turns out to be realistic enough to transfer controllers learned in simulation to the real hardware, the next step would be to learn hopping in 2D. This way, a big non-linearity of the system, presented in the vertical linear guide will be gone by replacing it with a boom. To learn hopping in the second dimension, new reference trajectories would have to be collected. These can come from online data bases like (Müller et al., 2007) or better be collected by ourselves to completely be aware of how the experiment was conducted and how the data was processed. The current setup with two motors would be sufficient to accomplish this task and only little extra effort is expected to extend the current algorithm to the new motion, given the new reference data. Same applies to extending the simulation model with a foot or a second leg, where additional research questions could be targeted.

An other future step could be using the human muscle model, implemented by Zhao based on (Song and Geyer, 2015), and let the agent learn to generate hopping by outputting muscle activations. Different numbers of muscles, starting from a single knee extensor muscle with gravity acting as the knee flexor (Schumacher and Seyfarth, 2017), to using several uni- and bi-articular muscles would be an interesting topic to investigate. By extending the muscle implementations with a neuromuscular reflex model as proposed by Schumacher and Seyfarth (2017) and letting the agent learn the blending and gain parameters to generate human-like hopping, a deeper insight in the human neuromuscular control could be achieved.

8 Conclusion

We met the objective of this work and developed a learning-based model-free controller to generate human-like vertical hopping in a realistic simulation of the two segmented GURO Hopper robotic leg by using deep reinforcement learning (DRL). The DRL agent feed-forward controls the torques in two motors during the stance phase, where in the flight phase PD position controllers brings the robot in the desired touchdown posture. Generated hopping is stable and robust against perturbations in form of ground level changes of up to 50% of the robot's leg length, showing a human-like perturbation recovery. Moreover, the controller shows human-like leg stiffness during the stance phase (figure **16**) without explicitly being trained to mimic this feature and is overall more human-like then hopping achieved by traditional model-based approaches.

To meet our aim, a simulation model of the GURO Hopper robot was built from scratch using the MuJoCo physics engine, considering joint and motor torque ranges as well as the sensor precisions, given in the real robot (Section 5.1). Some important parameters were identified by experiment (Section 5.2). The simulation is running with 2kHz.

The PPO algorithm (Section 4.4) was utilized in order to learn following reference COM trajectories derived from a human hopping experiment. The same algorithm has shown the capability to imitate a version of the same trajectory, scaled in time by 20%. In addition, it handled to follow the trajectories when the weight of the robots hip was increased by 50%.

In addition, best practices for developing learning-based controllers for generating human-like hopping were identified. Linearly softened termination conditions and a motion specific reference state initialization (RSI) have been shown to improve learning speed and robustness of the learned controller.

Finally, our research supported the assertion, that deep reinforcement learning techniques offer a promising approach to develop controllers, able to generate sophisticated dynamic locomotion tasks in complex robotic systems in a human-like manner.

9 Acknowledgements and References

Being "just" a Bachelor-Thesis, this work required several months of intensive and focused work to get familiar with several before unknown topics like Simulation and Deep Reinforcement Learning and implement them to achieve the stated goals.

I want to thank everyone, supporting me on this journey: Guoping Zhao for his helpfulness, even on the weekends, willingness to compromise and the ideas, pushing our work forward; My family and especially my girlfriend Oksana, heavily supporting me through this intensive time and showing much understanding for my situation; Prof. Seyfarth and my colleagues from the Lauflabor for the help- and insightful discussions as well as the Feedback, tipps and tricks and Prof. Kupnik for co-supervising and his interest in this work.

Hereafter, the References are following.

Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (Proc. SIGGRAPH 2018)*, 37(4), 2018a.

Hamid Benbrahim and Judy A. Franklin. Biped dynamic walking using reinforcement learning. *Robotics and Autonomous Systems*, 22:283–302, 1997.

Jun Morimoto, Gordon Cheng, Christopher G Atkeson, and Garth Zeglin. A simple reinforcement learning algorithm for biped walking. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 3, pages 3030–3035. IEEE, 2004.

Peter Fankhauser, Marco Hutter, Christian Gehring, Michael Bloesch, Mark A Hoepflinger, and Roland Siegwart. Reinforcement learning of single legged locomotion. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 188–193. IEEE, 2013.

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

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

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

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel van de Panne. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (Proc. SIGGRAPH 2017)*, 36(4), 2017.

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

www.agilityrobotics.com/sims. Agility robotics official website - simulation. http://www.agilityrobotics.com/sims, Accessed: 2018-10-21. Accessed: 2018-10-21.

Maziar A. Sharbafi, David Lee, Tim Kiemel, and André Seyfarth. Chapter 2 - fundamental subfunctions of locomotion. In Maziar A. Sharbafi and André Seyfarth, editors, *Bioinspired Legged Locomotion*, pages 11 – 53. Butterworth-Heinemann, 2017. ISBN 978-0-12-803766-9. doi: https://doi.org/10.1016/B978-0-12-803766-9.00003-8. URL http://www.sciencedirect.com/science/article/pii/B9780128037669000038.

Ajij Sayyad, B. Seth, and P. Seshu. Single-legged hopping robotics research—a review. *Robotica*, 25(5):587–613, 2007. doi: 10.1017/S0263574707003487.

Yvonne Blum, Susanne W Lipfert, and Andre Seyfarth. Effective leg stiffness in running. *Journal of biomechanics*, 42(14):2400–2405, 2009.

A Seyfarth, A Friedrichs, V Wank, and Reinhard Blickhan. Dynamics of the long jump. *Journal of biomechanics*, 32(12):1259–1267, 1999.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

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

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction (bibinfoedition2 ed.), 2018.

Vitus David Valentin Henning and Guoping Zhao. Sensor-map based control of a hopping robot. TU Darmstadt, 2018.

Hartmut Geyer, Andre Seyfarth, and Reinhard Blickhan. Positive force feedback in bouncing gaits? *Proceedings of the Royal Society of London B: Biological Sciences*, 270(1529):2173–2183, 2003.

Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. https://github.com/openai/baselines, 2017. Last updated on: 2018-08-18.

Karen E Adolph, Whitney G Cole, Meghana Komati, Jessie S Garciaguirre, Daryaneh Badaly, Jesse M Lingeman, Gladys LY Chan, and Rachel B Sotsky. How do you learn to walk? thousands of steps and dozens of falls per day. *Psychological science*, 23(11):1387–1394, 2012.

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

Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008.

Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. MeshLab: an Open-Source Mesh Processing Tool. In Vittorio Scarano, Rosario De Chiara, and Ugo Erra, editors, *Eurographics Italian Chapter Conference*. The Eurographics Association, 2008. ISBN 978-3-905673-68-5. doi: 10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136.

Everette S Gardner Jr. Exponential smoothing: The state of the art. *Journal of forecasting*, 4(1): 1–28, 1985.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

Ruben Martinez-Cantin. Bayesopt: A bayesian optimization library for nonlinear optimization, experimental design and bandits. *The Journal of Machine Learning Research*, 15(1):3735–3739, 2014.

Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

Luai Al Shalabi, Zyad Shaaban, and Basel Kasasbeh. Data mining: A preprocessing engine. *Journal of Computer Science*, 2(9):735–739, 2006.

Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

Karl Theodor Kalveram, DFB Haeufle, Sten Grimmer, and André Seyfarth. Energy management that generates hopping. comparison of virtual, robotic and human bouncing. In *Proc. Int. Conf. on Simulation, Modeling and Programming for Autonomous Robots 2010*, 2010.

Reinhard Blickhan. The spring-mass model for running and hopping. *Journal of biomechanics*, 22(11-12):1217–1227, 1989.

Jonathan Oehlke, Maziar Ahmad Sharbafi, Philipp Beckerle, and Andre Seyfarth. Template-based hopping control of a bio-inspired segmented robotic leg. In *Biomedical Robotics and Biomechatronics (BioRob), 2016 6th IEEE International Conference on*, pages 35–40. IEEE, 2016.

Jonathan Oehlke, Philipp Beckerle, Andre Seyfarth, and Maziar Ahmad Sharbafi. Human-like hopping in machines: Feedback- versus feed-forward-controlled motions. *Biological Cybernetics*, 10 2018. doi: 10.1007/s00422-018-0788-4.

Christian Schumacher and André Seyfarth. Sensor-motor maps for describing linear reflex composition in hopping. *Frontiers in Computational Neuroscience*, 11:108, 2017. ISSN 1662-5188. doi: 10.3389/fncom.2017.00108. URL https://www.frontiersin.org/article/10.3389/fncom.2017.00108.

Xue Bin Peng and Michiel van de Panne. Learning locomotion skills using deeprl: Does the choice of action space matter? In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '17, pages 12:1–12:13, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5091-4. doi: 10.1145/3099564.3099567. URL http://doi.acm.org/10.1145/3099564.3099567.

Jie Tan, Karen Liu, and Greg Turk. Stable proportional-derivative controllers. *IEEE Computer Graphics and Applications*, 31(4):34–44, 2011.

Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018b.

Meinard Müller, Tido Röder, Michael Clausen, Bernhard Eberhardt, Björn Krüger, and Andreas Weber. Documentation mocap database hdm05, 2007.

Seungmoon Song and Hartmut Geyer. A neural circuitry that emphasizes spinal feedback generates diverse behaviours of human locomotion. *The Journal of physiology*, 593(16):3493–3511, 2015.