university of groningen / faculty of science and engineering

Deep reinforcement learning for physics-based musculoskeletal simulations of healthy subjects and transfemoral prosthesis users during normal walking

Bachelor Thesis

Leanne de Vree, s3002195
Supervisor: Prof. dr. Raffaella Carloni

Abstract: This paper focuses on the implementation of a Deep Reinforcement Learning algorithm for the simulation of physics-based musculoskeletal models of both healthy subjects and transfemoral prostheses' users during normal level ground walking. The algorithm is based on the Proximal Policy Optimization approach in combination with reward shaping and imitation learning to reduce the computation time of the training while guaranteeing a natural walking gait. Firstly, the optimization algorithm is designed for the OpenSim model of a healthy subject and validated with experimental data. Afterwards, the optimization algorithm is applied to the OpenSim model of a transfemoral prosthesis' user which has been obtained by substituting the healthy muscles with muscle-like linear motors. The model of the transfemoral prosthesis' user shows a stable gait, with a forward dynamic comparable to the healthy subject model, yet using higher muscle forces.

1 Introduction

To support the understanding of biomechanics of gait patterns, research makes use of computer simulations (Geijtenbeek et al., 2013; Ranz et al., 2017; Ong et al., 2019; Harandi et al., 2020). These simulations are used to study both healthy and unpaired gaits and gaits supported by assistive devices and prostheses.

This paper builds upon this branch of research by employing computer simulations to study transfemoral prostheses and walking patterns. In this study, we use computer simulations with two models: a model of a healthy subject and a model of a transfemoral amputee. We apply Deep Reinforcement Learning (DRL) algorithms for both simulations to optimize movements such that the models can learn how to walk forward on a flat surface. Specifically, two algorithms are compared: Proximal Policy Optimization (PPO) (Schulman et al., 2017) and PPO with Imitation Learning. After training, the resulting gait patterns are compared in order to study the effects of having a prosthesis on gait patterns and muscle forces.

This study consists of three main steps. Firstly, to implement an optimizer using DRL algorithms on a healthy subject model in OpenSim via computer simulations. OpenSim is an open-source simulation software system that allows users to develop models of musculoskeletal structures and create dynamic simulations of movement (Delp et al., 2007). For this study, we use the healthy subject model as presented by Kidzinski et al. (2018). The model has two biological legs including 18 muscles to control 10 degrees of freedom (DOF): 3 DOF between the pelvis and the ground (2 translational and 1 rotational), 1 DOF at the hip joints, 1 DOF at the knee joints, 1 DOF at the ankle joints and 1 DOF that allows the pelvis to move freely in space. The DRL algorithm is applied to this healthy subject model

and validated by experimental data from Schwartz et al. (2008).

Secondly, to build a model for a transfemoral amputee in OpenSim. Previous research has already demonstrated that it is possible to build realistic biomechanical human models with several deficits. However, there has not yet been a model specifically designed for studying transfemoral amputees. Therefore, this paper will present this model of which the performance will be compared to the the healthy subject model. The transfemoral amputee model built for this study has 19 muscles to control 14 degrees of freedom. 15 out of 19 muscles are modelled as healthy muscles, while those at the knee and ankle joints of the right leg represent the prosthesis. The 14 DOF include the 10 DOF from the healthy subject model, plus an additional 2 DOF for each leg allowing for the hip to adduct and abduct which increases stability.

Thirdly, to use the optimizer of step one for the transfemoral amputee model. After having validated the performance of the optimizer on the healthy subject model with experimental data, we apply it to the transfemoral amputee model of step two. This allows for analyzing the effect of a transfemoral prosthetic leg on walking patterns and muscle forces.

This paper will address the following research question: *Does DRL provide a useful optimization method for studying locomotion in transfemoral prostheses' users?* The presented study has the following contributions, it:

- uses State-of-the-Art Deep Reinforcement Learning algorithms to analyze walking patterns.
- introduces a modification of Proximal Policy Optimization (PPO) using Imitation Learning that allows for the optimization of the algorithm.
- finds the differences in knee- and ankle- forces for amputees and non-amputees.

The remainder of the paper is organized as follows. Section 2 reviews the literature on walking patterns for transfemoral amputees. Based on previous literature, this section develops this paper's hypothesis on the usefulness for simulations on studying the characteristics of a transfemoral prosthetic leg. Section 3 introduces the methodological framework for this study: the deep neural network, algorithm and optimizer for training our model how to walk while minimizing the cost function. In section 4 the implementation is described with an explanation of OpenSim and the biomechanical components of the model for both a healthy subject and a transfemoral amputee. The empirical results are presented and discussed in section 5. Finally, concluding remarks are drawn in section 6.

2 Theoretical background

This study focuses on state-of-the-art deep reinforcement learning algorithms for simulations, hence studying emerging walking patterns of a physics-based model, hereafter called *agent*, that has no information about the environment. Before turning to the methodology of this paper, we discuss the current academic contributions surrounding the central concepts of this study: optimization strategies and modelling transfemoral prostheses in OpenSim. The first part gives an overview of studies that use OpenSim as a simulation tool. The second part introduces the most widely used optimization strategies in prosthetics research. The third part applies the theoretical findings by stating the novelty and contributions of this study: combining deep reinforcement learning as a state-of-the-art optimization strategy with transfemoral prosthetics.

2.1 OpenSim

OpenSim (Delp et al., 2007) is an open-source software tool for modeling, simulating, controlling and analyzing the human neuromusculoskeletal system. By performing inverse dynamics analysis and forward dynamics simulation, OpenSim allows for simulations of human locomotion and is used in a wide range of studies.

We have several justifications for using OpenSim in this study. Firstly, OpenSim allows for simulating close to physically accurate motions with non-evasive motion capture techniques, meaning that measuring the motion does not affect the motion itself (Galloy et al., 2018). Running gait pattern experiments on both healthy subjects and transfemoral amputees in real life takes large amounts of resources and there is an additional need to find

participants. The OpenSim simulation software can be used without these costs and still provide means to study gait patterns which are close to physically accurate.

Secondly, OpenSim provides a flexible platform that can be easily used in combination with optimization routines based on reinforcement learning. For this study, we use an implementation platform that has been used in the Neural Information Processing Systems (NIPS) challenge where participants were asked to build controllers for models designed in OpenSim (Kidzinski et al., 2018). The open-source OpenSim software allows for both designing models and connecting to this platform that allows users to control the built models.

Furthermore, OpenSim is widely used in current research on locomotion. Van der Krogt et al. (2012) use OpenSim to generate muscle-driven simulations of normal walking and then progressively weakened muscle groups to examine how much weakness could be tolerated in order to maintain a normal gait pattern. Steele et al. (2010) compare the contributions of mass center accelerations and joint angular accelerations during single-limb stance in crouch gait to those in unimpaired gait. LaPre et al. (2018) analyze the interaction between residual limb and prosthesis socket in transtibial amputees using OpenSim as a simulation tool. OpenSim is also used in transfemoral prosthetics research, for instance to examine the influence of limb alignment and surgical technique on a muscle's capacity to generate force (Ranz et al., 2017) and to study gait compensatory mechanisms (Harandi et al., 2020).

Most studies that use OpenSim do so by importing experimental kinematics. In this paper, we are interested in generating simulations de novo, by making use of an optimization strategy. This research will be discussed in the next section.

2.2 Optimization strategies

Optimization strategies are algorithms that aim to optimize a policy in order to perform a certain task. Algorithms have been designed to optimize find a global minimum of an objective function using computer power efficiently. Figure 2.1 provides an overview of a general state-of-the-art framework for dynamic optimization. In this study, we focus on the optimizer part for which a variety of optimization strategies can be used.

Papers that generate predictive simulations to study gait patterns of prosthetic users generally use evolutionary algorithms (EAs) for optimization (Geijtenbeek, 2012). The choice for using EA methods is often based upon its resemblance with natural selection, thus assumed to produce more natural behaviors (Alexander, 2001). As in evolution, the chances of reproduction for individuals in a population are determined by the fitness of its genes. In these applications, the population would be represented by candidate solutions and its genes are the gait parameters, such as the activation for each muscle. After each timestep, individuals reproduce based on their fitness. Candidate solutions with a higher fitness have a higher chance of reproduction, with the goal of ultimately finding the optimal solution for the task.

2.2.1 Covariance Matrix Adaptation

Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is an evolutionary algorithm often applied as an optimizer for robotic applications (Hansen, 2006). By continuously updating a matrix, the algorithm searches for optimal solutions following an objective function.

Geijtenbeek et al. (2013) develop an optimization strategy based on CMA-ES for the simulated motion of 3D bipedal characters. They created physics-based characters after which the geometry is optimized to allow for natural walking. Parameters that are optimized include muscle routing, physiological properties, and attachment points. These were optimized following an objective function to minimize errors based on speed, head orientation, head velocity and effort.

Yin et al. (2018) use CMA-ES to solve the problem of torque control for a powered ankle-foot prosthesis. Optimization of the control parameters demonstrates that the overshoot of the CMA-ES PID is 15 times lower than that in the original control parameters. Similary, Ong et al. (2019) use CMA-ES as an optimizer to create gait patterns. They trained two simulation models in OpenSim: one healthy subject and a subject with specific ankle weaknesses, finding divergence in walking patterns.

Although CMA-ES works well on a variety of tasks, Ong et al. (2019) note that reflex-based controllers are unable to predict changes in the

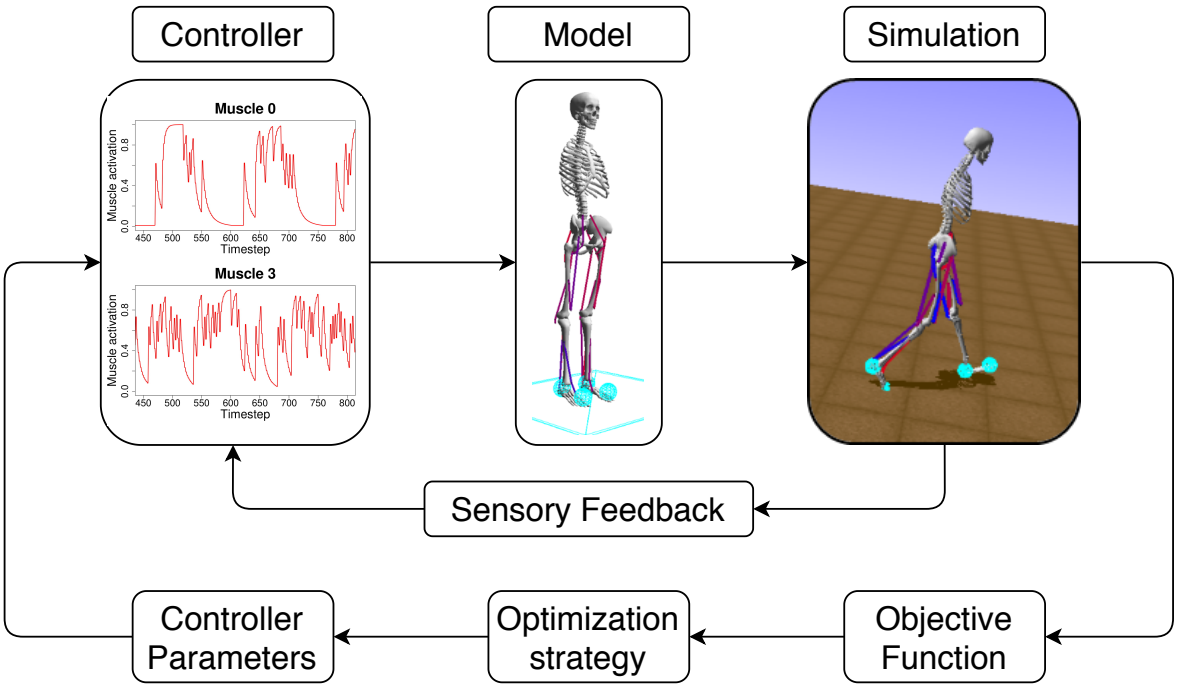


Figure 2.1: Description of the general state-of-the-art framework for dynamic optimization, based on Fig 1 in Ong et al. (2019). The musculoskeletal model is trained via musculotendon actuators by optimizing the parameters of a gait controller based on an objective function, e.g. to minimize metabolic cost, avoid falling and maintain head stability. The gait controller computes muscle excitations for the musculoskeletal model to generate a forward simulation. Sensory feedback, based on the model's muscle and joint states, is used in a feedback loop with the gait controller. The objective function returns a measure of performance for each simulation, forming a feedback for the optimizer that updates the values of the variables in the optimization problem.

underlying control structure, such as changes in muscle length or maximum amount of excitation. They propose using recent advances in reinforcement learning to generate more robust simulations. Therefore, this study explores the use of state-of-the-art deep reinforcement learning algorithms for simulations, and applies them on a new model, a transfemoral amputee model.

2.2.2 Biogeography-Based Optimization

Biogeography-Based Optimization (BBO) is an evolutionary algorithm similar to CMA-ES (Simon, 2008). Instead of updating a matrix, BBO uses the migration behavior of candidate solutions which are divided into a population where the sharing process of decision variables is analogous. Across generations, each candidate solution immigrates decision variables from and emigrates them to others.

Davis et al. (2014) use BBO to train a prosthetic leg testing robot to walk with a prosthesis. Their results show that BBO could achieve a 62% decrease in the ground reaction force error with regard to gait data. Thomas et al. (2013) train artificial neural networks with BBO with regard to prosthetic knee control. It is demonstrated that BBO improved the average performance of the powered knee prosthesis by up to 8%. Abdelhady et al. (2017) use BBO to optimize the parameters of an active prosthetic knee and demonstrate that it works well to predict these values.

Ammu et al. (2013) list several disadvantages of using BBO as an optimizer. First, although BBO is designed to converge to a solution, it often ends up in local minima. This could be solved by increasing the rate of mutation next to migration to increase randomness. However, high rates may complicate

the algorithm after which it cannot find a solution. Second, there is no provision that requires to always select the best members. Whether an individual reproduces in BBO is dependent upon the rates of migration. To prevent local minima, these probabilities are rarely set to 100%. Therefore, useful information in the best members can be lost while using BBO. Third, taking over features from the previous generation is independent of the individual's resultant fitness, hence infeasible solutions are generated. This unnecessarily increases the computational resources needed making the algorithm less efficient.

2.2.3 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is an evolutionary algorithm that focuses more on swarm behavior rather than on information sharing, and is one of the most commonly used optimization techniques (Elbeltagi et al., 2005). PSO is defined by self-organization: it is not controlled by any factor inside or outside the system. It performs searching on a swarm of particles that updates each timestep. To find the global minimum, each particle moves towards its previously best solution and the global best solution in the swarm.

Ferreira et al. (2019) propose a simulated transtibial amputee model with both a passive and an active prosthesis optimized by means of PSO. The fitness was calculated in terms of distance traveled, minus a penalty for falling and not adhering to a desired velocity. They found that with an assymetric gait pattern amputees consume around 20%-30% more metabolic energy compared to healthy individuals. Azimi et al. (2015) introduce a robust model as an adaptive controller for an active transfemoral prosthetic leg. In their framework, PSO is used to optimize the design parameters of the controller where the cost function consists of control signal magnitudes and tracking errors. They demonstrated that PSO obtains an 8% improvement in the objective function.

Kameyama (2009) reports several issues with regard to the stability and convergence of PSO. The paper notes that the trajectory of each particle becomes an oscillation, hence leading to instability of the algorithm. In addition, the algorithm was found to rarely lead to final convergence after a local minimum.

2.2.4 Deep Reinforcement Learning

Although there exists a rich literature on studying transfemoral prosthetics using various optimization strategies, research that has examined the effects of transfemoral prostheses on walking patterns using deep reinforcement learning as an optimization strategy is relatively sparse.

Deep Reinforcement Learning (DRL) consists of two parts: Reinforcement Learning (RL) and Deep Learning (DL). RL is a branch of Machine Learning in which the learner is a decision-making agent that takes actions in an environment receiving feedback for its actions when judged in terms of solving a certain problem (van der Ree and Wiering, 2013; Sutton and Barto, 2018). Hence, the agent can take actions in an environment and based on the reward that action provides, the agent decides whether this action should be taken again when being in the same state.

Figure 2.2 shows the DRL structure how it is also used in this study. The agent starts in a specific state and based on this state, the neural network outputs an action. This action is performed on the state (e.g. joint angles and joint muscles), which returns a reward with which the agent updates the weights in its neural network. Moreover, the taken action results in a new state, and the process continues.

Most of the current contributions to prosthetics research that use DRL focus on arm prosthetics. Katyal et al. (2016) use a simple non-deep neural network in combination with RL to learn a policy for in-hand manipulation directly from raw image pixels. Vasan and Pilarski (2017) use RL to teach a powered prosthetic arm to perform tasks as a an intact arm. Mudigonda et al. (2018) demonstrate that it is possible to learn robust grasp policies for anthropomorphic hands by means of DRL.

Table A.3 in Appendix A provides a complete overview of the articles discussed in this section applying the 4 different algorithms.

2.2.5 Limitations of earlier methods

The studies above have several limitations which does not allow to apply their optimizers to our problem of studying transfemoral prosthetics. Firstly, Katyal et al. (2016) study a method requiring a simplified action space which would not

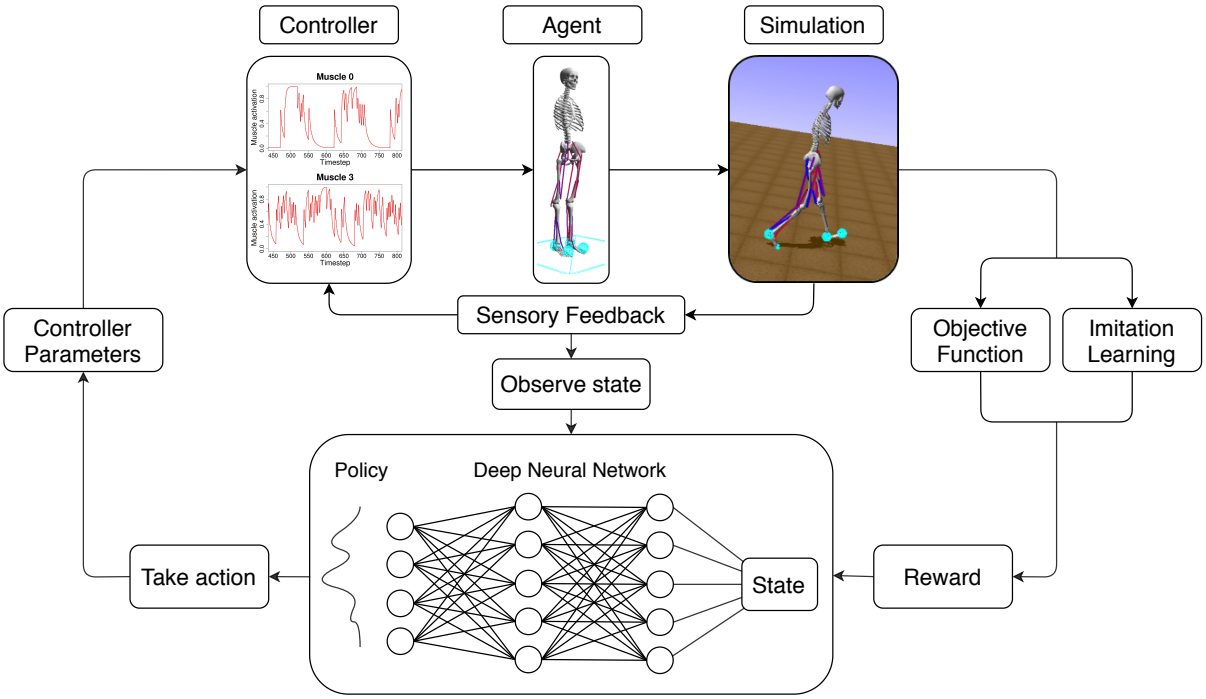


Figure 2.2: The Deep Reinforcement Learning structure applied to gait pattern research. The musculoskeletal model serves as an agent which aims to select actions such that it maximizes the reward. The reward and the observed state of the environment (e.g. joint angles, muscle angels) are the input of the neural network, which outputs an action. The action is applied to the gait controller resulting in forward simulation.

work for walking. Their study uses a robotic hand with eight possible actions, hence using a limited discrete action space. The problem presented in this paper requires a method that can solve for a continuous action space, i.e. non-binary values for activating muscles. Therefore, the method in this paper extends their approach by allowing the DRL algorithm to learn in a continuous action space.

Secondly, Vasan and Pilarski (2017) combine a RL algorithm with Learning from Demonstration (LfD) techniques, thus requiring experimental data for muscle activations. Since this study does not have access to muscle activations data for a transfemoral prosthetic leg, we use a technique of Imitation Learning instead. This method uses data on model's state, such as the joint angles, rather than muscle activations. This allows the algorithm to find the optimal muscle activations to generate a walking pattern without information of what those activations should be.

Lastly, Mudigonda et al. (2018) use trust region policy optimization (TRPO), an algorithm similar to the one used in this paper yet having several limitations. PPO optimizes TRPO by modifying the surrogate objective function such that it prevents large policy updates. This adaptation improves the performance of the algorithm as well as decreasing the complexity of both implementation and computation (Schulman et al., 2017).

2.3 Transfemoral Prosthetics and Deep Reinforcement Learning

Previous research has demonstrated that DRL may provide an effective tool in studying arm prosthetics. However, so far it has not yet been applied to transfemoral prosthetics. Therefore, this study will test various state-of-the-art DRL algorithms for a transfemoral prosthetic user.

Throughout this study, the term 'transfemoral prostheses' will be used to account for a wide range

of prosthetic devices. They share the characteristic of serving to replace limbs following a transfemoral amputation, i.e. losing both the knee and ankle joints.

This paper argues that the advantages of DRL as explained in the previous section can be effectively applied to the problem of studying transfemoral prosthetics. We have the following justifications for using DRL in this study.

First of all, DRL has proven to work well for studying transtibial (below-knee) amputees so provides a promising method to apply to transfemoral amputees. In the NeurIPS 2018 Artificial Intelligence for Prosthetics challenge, participants were asked to build a controller for a humanoid model of a transtibial amputee with the goal of making it move forward (Kidzinski et al., 2018). They used a musculoskeletal model of 19 muscles with one leg having the lower muscles replaced by a prosthesis. Participants were given an environment that encourages the use of DRL. Results have shown that these techniques can find solutions in which the agent learns a policy to efficiently move forward.

Secondly, a musculoskeletal model of a transfemoral amputee performs its motions in a continuous action space and DRL is specialized to deal with continuous action spaces. Muscles cannot only activate or deactivate, they can also partly activate by imposing less or more power on the muscle. Combined with the degrees of freedom, this creates a large action space which makes usual RL unsuitable. Therefore, DRL can provide solutions allowing an agent to learn how to walk with various versions of a transfemoral prosthetic leg.

Thirdly, using DRL limits the need for experimental data. In transfemoral prosthetics research, collecting experimental data can be complicated due to the costs of finding and testing suitable participants. The advantage of using computer simulations in combination with DRL techniques is that it does not require the agent to have knowledge about the environment. The agent purely acts on the rewards and penalties it receives from trying actions, which are designed based on an objective, such as moving forward. Therefore, little experimental data is needed to find an efficient solution.

Fourthly, DRL allows for making and studying adaptations of the prosthetic device quickly. Due to material, time and participant constraints, experimental methods for transfemoral prosthetics complicate the adaptation of a prosthetic device and test it directly. With computer simulations, changes to the device can be made quickly and specifically. This allows for testing several variations of e.g. joint motor sizes in a short amount of time, allowing for finding the most optimal combination of the device's characteristics that lead to the highest performance. DRL algorithms are flexible to change, meaning that the same algorithm can be applied to an agent with a different prosthetic leg and still converge to a solution.

Therefore, this paper tests the following hypothesis: *Deep reinforcement learning provides a useful optimization method for studying locomotion in transfemoral prosthetics.* This hypothesis will be discussed further and presented in a workable form in the methodological section.

3 Methods

The aim of this study is to implement a deep reinforcement learning algorithm that allows the models introduced above how to walk so the emerging walking patterns can be compared. This section introduces the algorithms used in this paper. It starts with an explanation of how Deep Reinforcement Learning (DRL) can be applied to this problem. The subsequent sections introduce the specific algorithms used in this study: PPO and PPO with Imitation Learning.

3.1 Deep neural network

The foundation of any DRL algorithm is a deep neural network (DNN). A DNN is an artificial neural network (ANN) with multiple layers between the input and output layers. Figure 2.2 provides a visualization of the structure of a DRL network.

ANNs consist of layers of neurons, which receive input and produce output. The output is based on the combination between the input, the neuron's internal state (activation) and a threshold using an activation function. Each layer has a number of neurons, where each of the neurons is connected to each of the neurons in the next layer (fully connected). Furthermore, each connection is assigned a weight that defines its relative importance in the network. The input moves through the layers calculating the probability of a certain output.

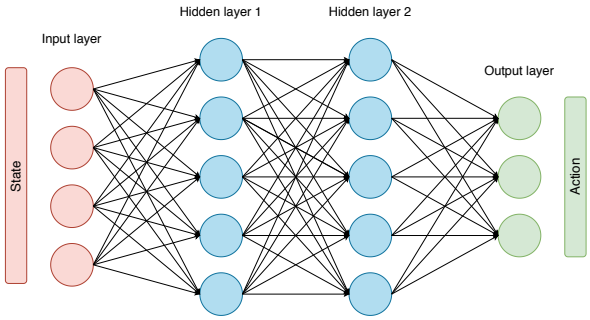


Figure 3.1: The structure of an example of a Multi-Layer Perceptron. In this study, the input layer consists of 214/218 neurons, the hidden layers each of 312, and the output layer consists of 18/19 neurons. The black arrows indicate the connections in the fully connected feed-forward artificial neural network.

In this study, we train a Multi-Layer Perceptron (MLP). A MLP is a type of feed-forward ANN composed of multiply layers of perceptrons, a binary classifier neuron that includes a threshold activation. The network consists of 4 layers which are visualized in figure 3.1:

- Input layer: state size, 214 or 218
- Hidden layer: size 312
- Hidden layer: size 312
- Output layer: action size, 18 or 19

For each of the neurons v_i, the output y is calculated using a general output function, which is described by:

$$(3.1)\quad y(v_i) = tanh\left(b + \sum_{i=1}^{n} x_i w_i\right)$$

Where n is the number of inputs from the previous layer, x the input to the neuron, w the weight between the current and previous neuron, and b denotes the bias term. The layers in the MLP use a tanh nonlinear activation function. The advantage of using this function is that the negative inputs will be mapped strongly negatively and the zero inputs will be mapped near to zero, allowing for more reliable outcomes. This results for each neuron in an output or *activation* which serves as the input for the next layer.

For our problem, the input of the neural network in the DRL structure (see figure 3.1) is the state of the environment. The state consists of the following parts: joint angles, ground reaction forces, muscle activities, muscle fiber lengths, muscle velocities, tendon forces, positions, velocities and accelerations of joint angles and body segments, up to a total of 214 variables for the healthy subject model and 218 for the amputee model (see appendix A.1).

The output of the network is an action, depending on the model an 18 or 19-dimensional vector, where each variable represents the activation of one of the muscles in the model. Both the state- and action vector are continuous. This entails that the values they contain are neither binary nor binned to a certain distribution. Our goal is to optimize the weights in the neural network such that for each given state, it outputs the optimal action to allow the model to move forward.

3.2 The learning algorithms

In order to achieve this goal, we can train the MLP such that it performs input-output behavior that satisfies our task of moving forward. During training, the weights of the connections between neurons are optimized such that the network outputs desirable actions based on the state input. The following sections discuss the learning algorithms used to train the MLP for the task of moving forward.

3.2.1 Proximal Policy Optimization

The main algorithm used in this paper is Proximal Policy Optimization (PPO), a DRL algorithm designed by OpenAI (Schulman et al., 2017). It is introduced as a novel alternative to policy gradient methods. By combining the benefits of Trust Region Policy Optimization (TRPO) it becomes more effective yet simple to implement.

In PPO, the agent alternates between sampling trajectories with the newest policy and performs optimization on the objective using the earlier sampled trajectories. It aims to not allow for large updates of the policy, the mapping from states to actions. This is done by keeping the Kullback-Leiber (KL) divergence, a measure how one probability distribution is different from another, between the new and the old policy within the trust region. To achieve this, PPO clips the probability ratio and

adds the KL-divergence term to the loss. The ratio $r_t(\theta)$ denotes the probability ratio between the new and the old policy:

$$r_t(\theta) = \frac{\pi_\theta(\alpha_t|s_t)}{\pi_{\theta_{old}}(\alpha_t|s_t)} \tag{3.2}$$

where α_t is the action at timestep t and s_t is the state at timestep t.
PPO is defined by the following objective function:

$$L^{CLIP}(\theta) = \hat{E}_t\left[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right] \tag{3.3}$$

where $\hat{A}_t$ is the advantage estimation, the difference between the expected and the real reward from an action, and ϵ is the clip value which we set to 0.2. If the probability ratio falls outside the range (1 - ϵ) and (1 + ϵ). the advantage function will be clipped to prevent too large policy updates. Appendix A.2 lists the algorithm's hyperparameters and specifications.

To provide a more intuitive explanation, we will consider two scenarios:

High advantage. A high advantage means that the difference between the expected and the estimated reward is positively turned to the estimated reward. Hence, the new action in this state gives more reward than expected by our current policy. Since a higher reward is better, we want to ensure that the probability of taking the new action in this state increases to increase future reward. Therefore, we update the weights in the neural network such that there is a higher probability of taking this action in this state. The clip surrogate makes sure that the update is not too large, as disruptions in the current policy may have a negative influence on previous updates made.

Low advantage. On the contrary, for a low advantage the new action in this state gives less reward than expected by our current policy. Therefore, we want to decrease the probability of taking this action in the current state. Again, we update the weights in the neural network such that there is a lower probability of taking this action in this state, while the clip surrogate ensures that the update is not too large.

The advantage is calculated using the expected and the real reward from an action. The expected reward is based on the old policy, while the real reward is the output of the reward function which will be discussed in the next section.

3.2.2 Reward function

The reward function provides the agents, which are in our case either the healthy subject model or the transfemoral amputee model, with information about the value of its actions. In RL, the agent does not have any information about the environment. The only information it gets is that after each action, it receives a reward for that action. This reward can be both positive and negative, and the agent's goal is to optimize the rewards. The agent's behaviors are based upon maximizing the rewards, so the reward function has a large influence on the agent's ultimate behaviors. Therefore, we have to design it carefully for each task. In this task, we ultimately want the agent to move forward in a life-like manner which is defined in the reward function.

In order to manipulate the model such that it takes the desired actions, the reward function in this study consists of four parts: the timesteps the agent manages not to fall, the amount of distance travelled, the adherence to a desired velocity, and muscle fatigue. Each of these elements will be discussed in the subsequent sections.
It results in the following reward function where t stands for the timestep:

$$J(\pi) = reward_{move} + reward_{alive} = \sum_t(reward_{distance} - penalty_{velocity} - penalty_{costs}) + \sum_t(reward_{alive}) \tag{3.4}$$

The first part of the reward function concerns the distance the agent has travelled. This is expressed by adding a goal reward that calculates the difference in x-position of the agent's pelvis between the current and the previous timestep. The higher this difference, the more distance the agent has travelled during a timestep and the higher its reward.

The second part rewards the agent for adhering to a target velocity. During each of the simulations, the agent gets as input a certain target velocity. This velocity changes two or three times for each simulation. The goal of the agent is to develop a walking pattern in the specific target velocity, such as 1.25 m/s, corresponding to 4.5 km/h which is a reasonable speed for human walking. During each

timestep, the difference between the velocity of the agent's pelvis and the target velocity is calculated. The sum of squared error of this difference denotes a penalty for deviating from the target velocity.

The third element of the reward function gives a penalty for muscle fatigue. Transport costs the amount of energy that the muscles use for moving from point A to B. Certain gait patterns, such as one-legged walking, may also allow the agent for reaching its goal. However, this gait pattern puts more pressure upon muscles compared to a more natural, symmetric two-leg walking pattern. Since we are looking for the most efficient gait for reaching the goal, we add a penalty for muscle fatigue. For each timestep, the squared sum of the muscle activations serves as a penalty for the agent.

Lastly, the fourth part of the function regards the number of timesteps the agent manages not to fall. The simulation stops every time the pelvis of the agents drops below 0.7 meters. Since that is not a desired outcome, the agent is explicitly rewarded for every additional timestep it stays without falling down.

3.2.3 Imitation Learning

The reward function above is designed to lead to realistic motions, however, it does not ensure that the motions are human-like. Therefore, this paper introduces PPO with Imitation Learning. This method has proven to decrease training times as well as to increase performance, for instance as used by the Lance team in the NIPS 2018 competition (Kidzinski et al., 2019). To implement this method, we introduce an imitation term next to the original reward function.

The added imitation term uses experimental data to ensure that the algorithm converges to a solution and that the agent develops a natural walking pattern. For each timestep, both the position and the velocity loss of the pelvis, knee and hip joints are calculated. This is done by taking the sum of squared error of the difference between the current angle of the agent's joint and the joint in the kinematic data at a specific timestep. The same is done for the velocity, where its loss is calculated by the difference between the current speed of a joint and the speed of the joint in the kinematic data at a specific timestep. These losses serve as a penalty to the reward. The higher the losses, the lower the reward and the other way around. It encourages the agent to keep its states as close to the ones in the kinematic data as possible, hence encouraging a more natural walking pattern.

Goal - Imitation	Reward	Observations
25 - 75	6900	Mostly standing still
30 - 70	6380	Moving forward, falls often
35 - 65	**6120**	**Moving forward, little falling**
40 - 60	5400	Moving forward, falls often

Table 3.1: Trials for finding the optimal weights of the imitation learning and goal components of the reward function. The reward column shows the average reward after 25,000 episodes of training. Best results are printed in bold.

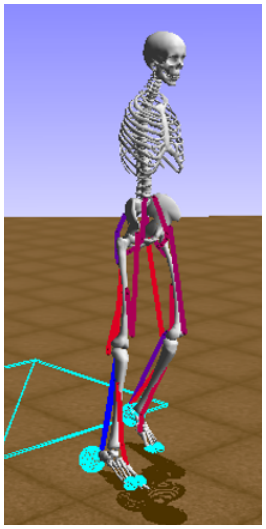
The total reward consists of the sum of the goal reward and the imitation reward. After tuning the weights for each of the elements by trial and error, it was found that 35% for the goal reward and 65% for the imitation reward returns the highest results. In combination with equation 3.4, the complete reward function that was used in the final model looks as follows:

$$Reward_t = reward_{imitation,t} * 0.65 + reward_{goal,t} * 0.35 \tag{3.5}$$

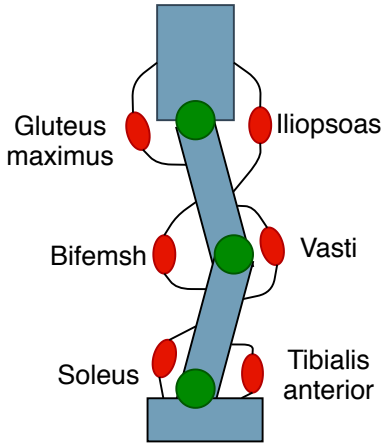
Table 3.1 reports the trials for finding the optimal weights of the imitation learning and goal components of the reward function.

4 Implementation

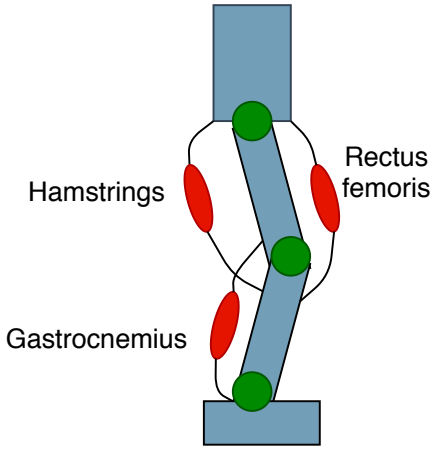
To test the hypothesis that Deep Reinforcement Learning can be used to study transfemoral prostheses, the paper follows two steps: defining an algorithm to be used as optimization strategy and implementing a transfemoral amputee model. For the second step, we focus on the tool OpenSim (Delp et al., 2007) to provide a physics-based model. In addition, the study makes use of an environment created by Stanford university which provides a link between the OpenSim software and Python programming that allows for the implementation of DRL algorithms (Kidzinski et al., 2018).



(a) Healthy subject model



(b) Uniarticular muscles



(c) Biarticular muscles

Figure 4.1: Overview of the muscles used in the models. Figure (a) shows the healthy subject model with 18 Hill-type muscles, shown in red and blue. Figure (b) displays in red the six uniarticular muscles in each leg that produce flexion or extension torques at single joints. Figure (c) shows the three biarticular muscles in red that generate torques at pairs of joints. The healthy subject model contains both the uniarticular muscles from (b) and the biarticular muscles from (c). The transfemoral amputee model contains both in the left leg and only the uniarticular muscles from (b) in the right leg.

4.1 Two models

This study tests two models: a healthy model with two biological legs and a model in which the right leg is replaced by a transfemoral prosthesis. The healthy model is used to validate the walking pattern that the optimizer finds against experimental data, hence serves as a method to test the performance of the optimizer. The transfemoral amputee model uses the same validated optimizer to analyze the effects of a transfemoral prosthetic leg on locomotion. Figure 4.1 visualizes and table A.1 (Appendix A) lists the muscles included in the two models.

4.1.1 OpenSim

OpenSim (Delp et al., 2007) is an open source software tool that provides dynamic simulations of the human neuromusculoskeletal system. In OpenSim, a model consists of rigid body segments which are connected by joints. Hill-type muscles connect these joints and produce forces and motion. Therefore, once a musculoskeletal model is created, OpenSim allows users to investigate a wide range of topics, such as the effects of geometry, joint kinematics and the muscle-tendon properties on the muscle's forces and joint moments. The left part of figure 4.1 shows an example of the animation in OpenSim. The red lines represent activated muscles, while the blue lines are deactivated muscles.

4.1.2 Healthy subject model

The simulated agent is a musculoskeletal model of a human. For this study, we use the original model provided in the NIPS competition with two biological legs (Kidzinski et al., 2018). The final model of the healthy agent includes 18 muscles to control 14 degrees of freedom which is visualized in figure 4.1. Table A.1 lists all muscles in the model.

4.1.3 Transfemoral amputee model

For the prosthetic part, the three biarticular (See the right part of figure 4.1) muscles were removed

from the right leg. This was done by replacing the OpenSim model input file from Kidzinski et al. (2018) with an adjusted model such that the right leg does no longer include the muscles: gastrocnemius, hamstrings and rectus femoris. The short head of the biceps femoris, vasti, soleus, and tibialis anterior muscles were kept so they can simulate sources to power the prosthetic leg. Hip adductors and abductors were added to both legs to make the model more realistic and allow it for maintaining stability. These hip muscles were taken from the transtibial model from Kidzinski et al. (2019) and added to the OpenSim model file. The final model of the agent with a prosthetic leg has 19 muscles to control 16 degrees of freedom.

4.1.4 Muscle actuation

The two models as presented above contain simulated biological muscles based on a first-order dynamic Hill-type muscle model between excitation and activation (Thelen, 2003). Figure 4.2 shows the Hill-type muscle model including a contractile element (CE), parallel element (PE) and series element (SE). The muscle force generated is a function of three factors: the length (l), velocity (v) and activation (a) level, which can range between 0% and 100% activated.

Based on the observation vector of internal states, e.g. the joint angles and body positions, the designed controller outputs a vector of muscle excitations. OpenSim calculates muscle activations from excitations using first-order dynamics equations of a Hill-type muscle model from Thelen (2003). Muscle activations generate movement as a function of muscle properties such as maximum isometric force, muscle fiber length (L^M), tendon slack length (L^T), maximum contraction velocity and the pennation angle (α^M).

To summarize, during each timestep in the simulation (0.01 seconds), the environment:

- computes activations of muscles based on the provided excitation vector.
- actuates muscles.
- computes torques based on the activations.
- computes ground reaction forces.
- computes positions and velocities of joints and bodies.

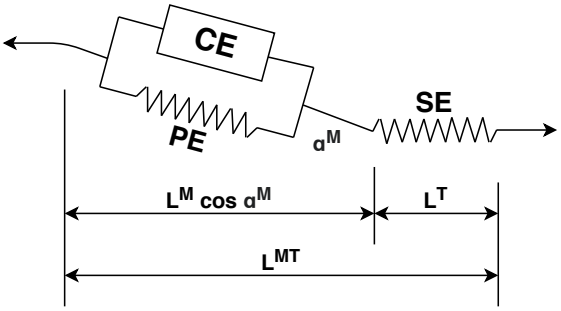


Figure 4.2: A Hill-type muscle model to describe the musculo-tendon contraction mechanics in the models. It includes a contractile element (CE), parallel element (PE) and series element (SE). The elements generate a force on tendon, following the equations as described in Thelen (2003).

- generates a new state based on the forces, velocities and positions of joints.

4.2 Experimental data

For implementing the imitation learning term and validating our optimizer, we use experimental data taken from healthy subjects (Schwartz et al., 2008). The data was collected by measuring the three-dimensional lower extremity joint kinematics, joint kinetics, surface electromyographic (EMG), and spatio-temporal data on 83 typically developing children. It contains the means of all subjects over one gait cycle at speeds ranging from very slow to very fast (>3 SD below/above mean free speed).

Joint kinetics and velocity data were used to implement the imitation learning term. Joint kinetics and ground reaction forces were used to validate the performance of our optimizer.

5 Results

This chapter contains the results of the study. The first section shows the results for the healthy model and demonstrates that it can be validated against experimental data. It first presents the performance of the different algorithms that were tested after which it compares the kinematic results of the simulation to experimental data. The second section presents the results of the amputee model.

5.1 Healthy subject model

5.1.1 Algorithms' performance

Figure 5.1 shows the performance of the algorithms on the healthy model. The red curve shows the average reward for PPO over a total of 25,000 episodes (one simulation), and the blue curve the same results for PPO with the added imitation learning term.

The results show that using PPO with imitation learning allows for better learning compared to normal PPO, with a reward mean of 3533.85 compared to 992.77 (see table 5.1). These numbers indicate that the healthy subject model learned to optimize the objective function around four times better with than without imitation learning. After around 15,000 episodes there is a steep rise in rewards which means that the agent has learned a policy, i.e. a division of weights in the neural network, that allows it to maximize returns based on the given reward function.

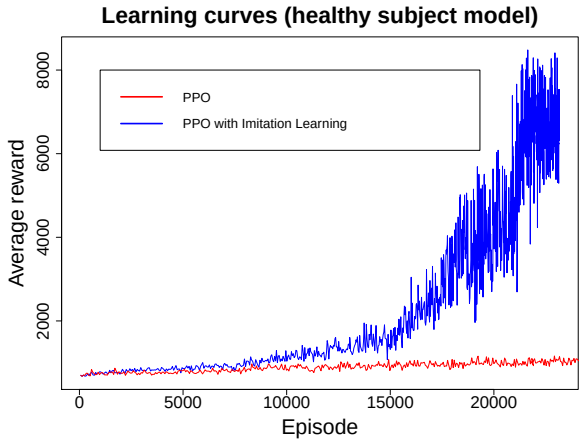


Figure 5.1: The learning curves of the two algorithms applied to the healthy subject model. The y-axis shows the reward and the x-axis the amount of episodes. It can be seen that PPO with imitation learning performs better as over time it learns to maximize rewards.

	PPO		PPO with IL	
	mean	**SD**	**mean**	**SD**
Healthy	992.77	126.73	3533.85	2329.87
Amputee	940.07	123.89	2848.93	1871.70

Table 5.1: The mean total reward received and the standard deviation for the combinations of algorithm and model.

5.1.2 Kinematics

Figure 5.2 shows the kinematics of the healthy model's emergent gait pattern learned using PPO with imitation learning compared to experimental data, including joint angles for hip, knee and ankle, and the horizontal and vertical ground reaction forces. It demonstrates that the simulated kinematic and kinetic trajectories of the emerging gait pattern are similar both in value and shape to experimental data.

Table 5.2 describes the kinematic results in terms of the root-mean-squared error (RMSE) and the Z-score. The RMSE compares simulation mean trajectories to those of experimental data. It is computed by taking the square root of the difference in errors squared and reported in units of standard deviation (SD). The Z-score denotes how close the simulation data is to the mean of the experimental data. The closer it is to 0, the closer the simulation is to experimental data. It is computed for the simulation data at every 2% of the gait cycle.

	Angles			GRFs	
	Hip	**Knee**	**Ankle**	**Hor**	**Vert**
RMSE	0.92	2.39	1.45	0.4	1.33
Z-score	-0.12	-1.40	-0.45	0.09	-0.18

Table 5.2: Similarity metrics between experimental and simulated data of the emerging gait pattern. The root-mean-squared error (RMSE) is reported in units of standard deviation (SD). The Z-score denotes the mean taken over all scores for each 2% of the gait cycle.

For the majority of the gait cycle, kinematic and kinetic trajectories of the emerging gait pattern are within 1 SD of experimental data describing a natural walking pattern. Furthermore, for each of the measures the RMSE between simulated and experimental data is no more than 2.39 SD and the Z-score is at most 1.40 away from the mean. The

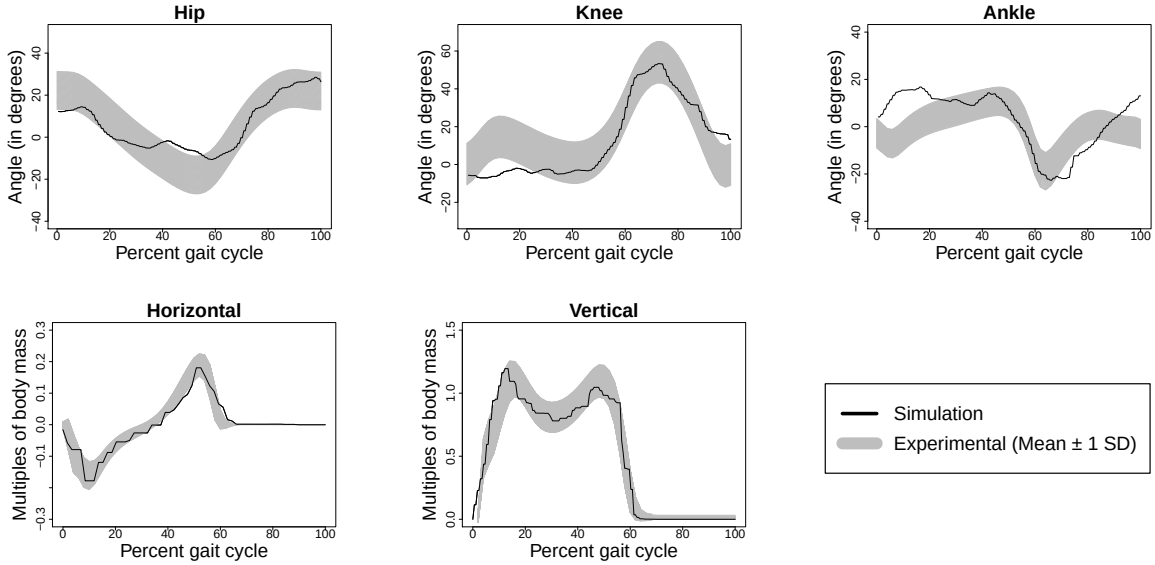


Figure 5.2: The angles and ground reaction forces of emerging gait pattern learned using PPO with imitation learning compared to experimental data (Schwartz et al., 2008). The black lines indicate the simulated data and the grey area is the healthy subject experimental data varying 1 standard deviation from the mean.

ground reaction forces were found to match well with experimental data with Z-scores of 0.09 and -0.18.

5.2 Transfemoral amputee model

5.2.1 Algorithms' performance

Figure 5.3 shows the performance of the algorithms on the model with a transfemoral prosthesis. The blue curve shows that the model is able to learn to maximize its rewards, with a steep increase in average reward around the 20,000th episode.

The results show that PPO with imitation learning also allows for the transfemoral amputee model to learn to walk forward with a reward mean of 2848.93 compared to 940.07 for normal PPO (see table 5.1). These numbers indicate that the transfemoral amputee agent as well has learned a policy that allows it to maximize returns based on the given reward function.

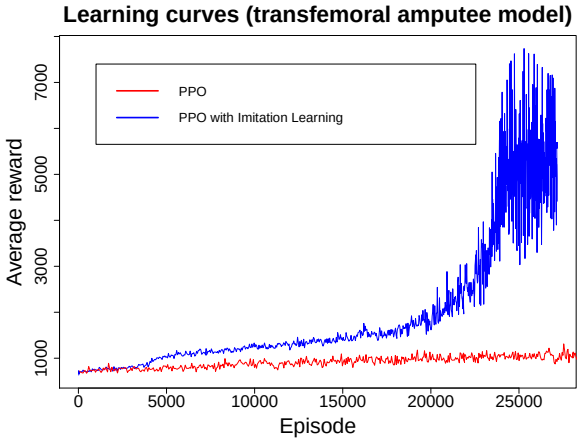


Figure 5.3: The learning curves of the two algorithms applied to the transfemoral amputee model. The y-axis shows the reward and the x-axis the amount of episodes.

5.2.2 Kinematics

Figure 5.4 shows the kinematic data for the emerging walking pattern of the transfemoral amputee

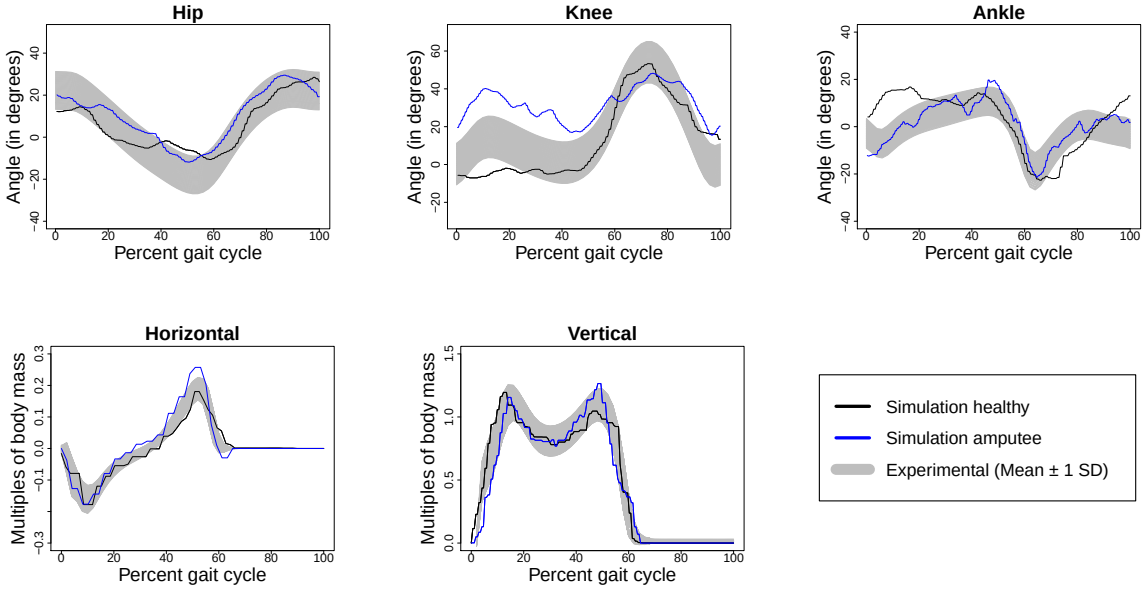


Figure 5.4: The angles and ground reaction forces of the emerging gait pattern using PPO with imitation learning of the transfemoral amputee model compared to the healthy and experimental data (Schwartz et al., 2008). The blue line indicates the simulated data from the transfemoral amputee model, the black line that from the healthy subject model, and the grey area is the healthy subject experimental data varying 1 standard deviation from the mean.

model. It compares the angles of the hip, knee and ankle and the horizontal and vertical ground reaction forces of the transfemoral amputee model with the healthy subject model and the healthy subject experimental data along the gait cycle.

The emerging gait pattern was found to have a similar shape and value for the hip and ankle, while it differs for the knee. The hip angles were found to be not much diverging from before, hence there is not much influence of the prosthesis on the movements of the hips. For the ankle, the resulting angles were found to have a similar shape and value compared to experimental data, yet being less smooth compared to the healthy model. This could be explained by that having less muscles in both the lower and upper leg allows for less control over the ankle hence making it more prone to uncontrolled and larger movements. The angles for the knee were found to be most diverging from both the healthy and the experimental data. The shape along the gait cycle is maintained, yet the angles are higher and the peak is less steep compared to the other data.

The ground reaction forces were also found to have a similar shape for the transfemoral amputee model. However, for both the forces the general value is higher compared to the healthy subject model. Hence, the ground exerted more force to the left leg of the amputee model compared to the left leg of the healthy model. This finding implies that the amputee model's left leg consumes more strength than the healthy model's left leg, as higher strength can be related to increased ground reaction forces (Laroche et al., 2011).

5.2.3 Comparing fiber forces

One of the goals of this study is to find the difference in used power for the muscles around the knee and ankle between the healthy and the prosthetic leg. Table A.4 (Appendix A) shows the results of our simulations with regard the used muscle power. It shows the mean in activation and fiber force values taken over 500 episodes of the trained model ($\pm$ 3 gait cycles). It demonstrates that the difference in activation for the amputee model (0.631) is 57.3%

higher compared to the healthy model (0.401). The difference in fiber force is 29.2% higher for the amputee model (3586.849 vs. 2775.849).

The same conclusion is reached from figures 5.5 and 5.6 which show for each of the four muscles the fiber force over time. It can be seen that the difference in mean fiber force between the left- and right leg is larger for the amputee model compared to the healthy model (see also figure A.1 in Appendix A). Furthermore, the amputee model's right (prosthetic) leg shows a higher fiber force compared to the healthy model's right leg.

Overall, the results demonstrate that PPO with imitation learning can provide a close to natural walking pattern for a healthy biomuscular model, and that it can be applied to do the same for a transfemoral model. The angles for the hip, knee, and ankle as well as the ground reaction forces were similar compared to experimental data. Furthermore, our results suggest that the muscles around the knee and ankle need around 57% more activation and 29% more fiber force in a transfemoral prosthesis to generate a natural gait.

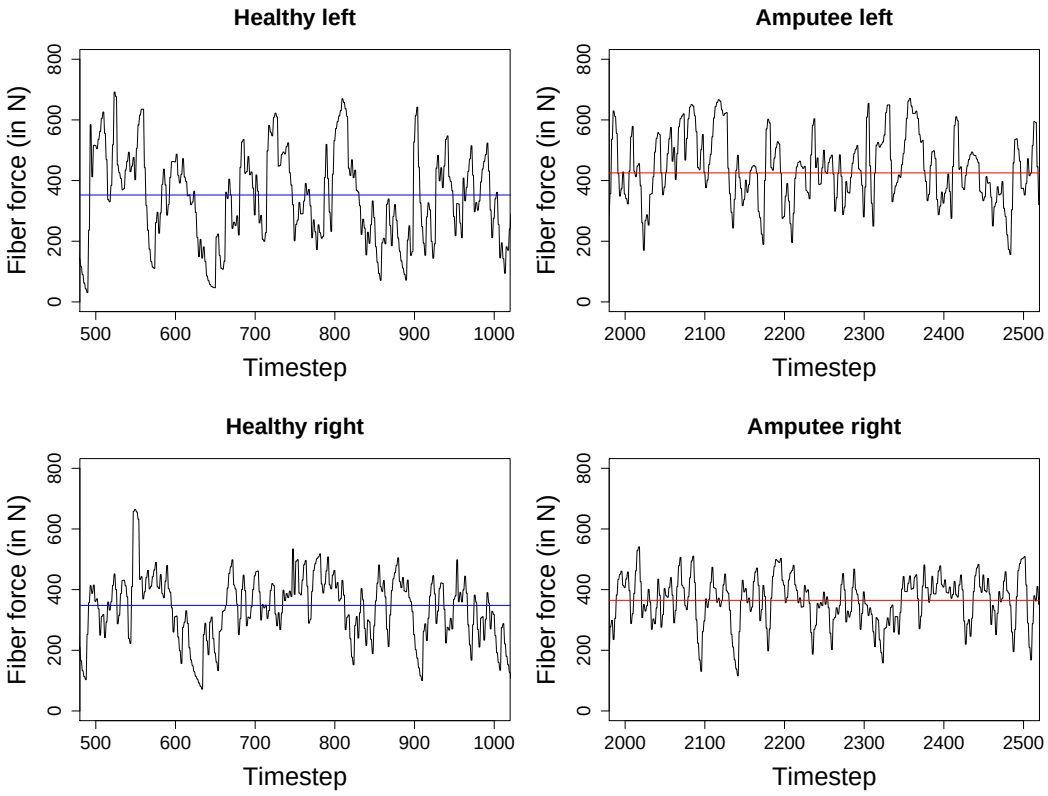
6 Conclusion

By examining the usage of computer simulations to study transfemoral prostheses and walking patterns, this paper contributes to an expanding research field on studying gait patterns. Having large amounts of literature on healthy walking patterns as well as transtibial prostheses, this paper hypothesized that computer simulations could be used for studying transfemoral prostheses as well.

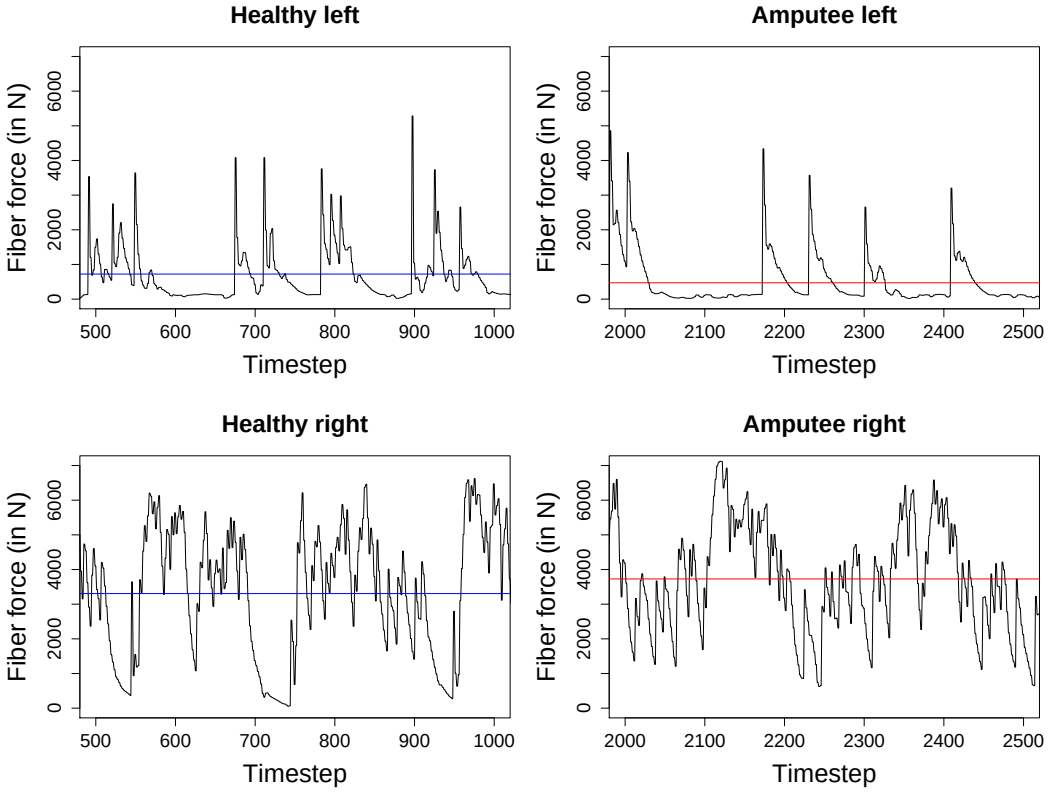
Testing these predictions, we presented two main contributions of this study to the existing literature. We found that the use of state-of-the-art Reinforcement Learning algorithms are useful for studying walking patterns for transfemoral prostheses. The presented modification of the Proximal Policy Optimization (PPO) algorithm that includes Imitation Learning allowed for an optimization of the algorithm and a 2.5 times increased mean reward compared to regular PPO. The emerging gait pattern was validated against experimental data, with close to natural ground reaction forces as well as angles for hip, knee and ankle.

In addition, our results suggest that a transfemoral prosthesis would need around 57% more activation around the knee and ankle compared to the healthy leg. Furthermore, there should be an increase of 29% in force that the replacing muscles would have to provide in order to generate a natural walking pattern.

In terms of broader academic literature, this study demonstrates that the use of computer simulations is useful to support the understanding of biomechanics of gait patterns. The simulations used in this study help the research of both healthy gaits and gaits with weaknesses.

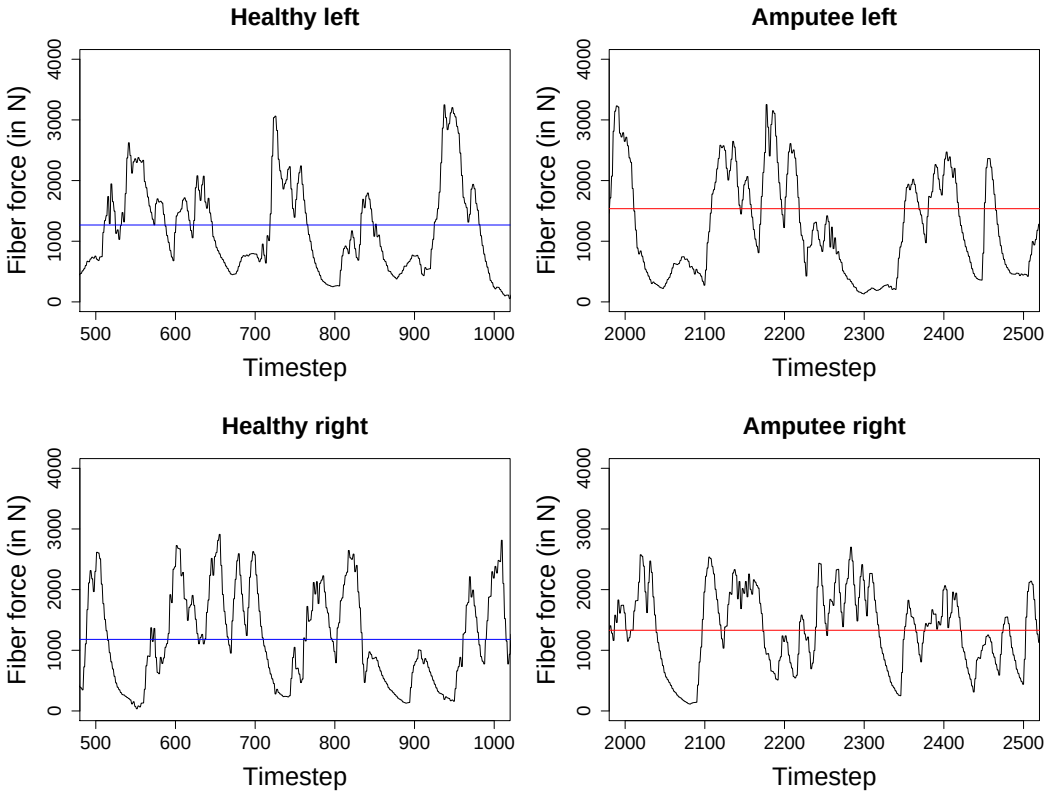


(a) Short head of the biceps femoris

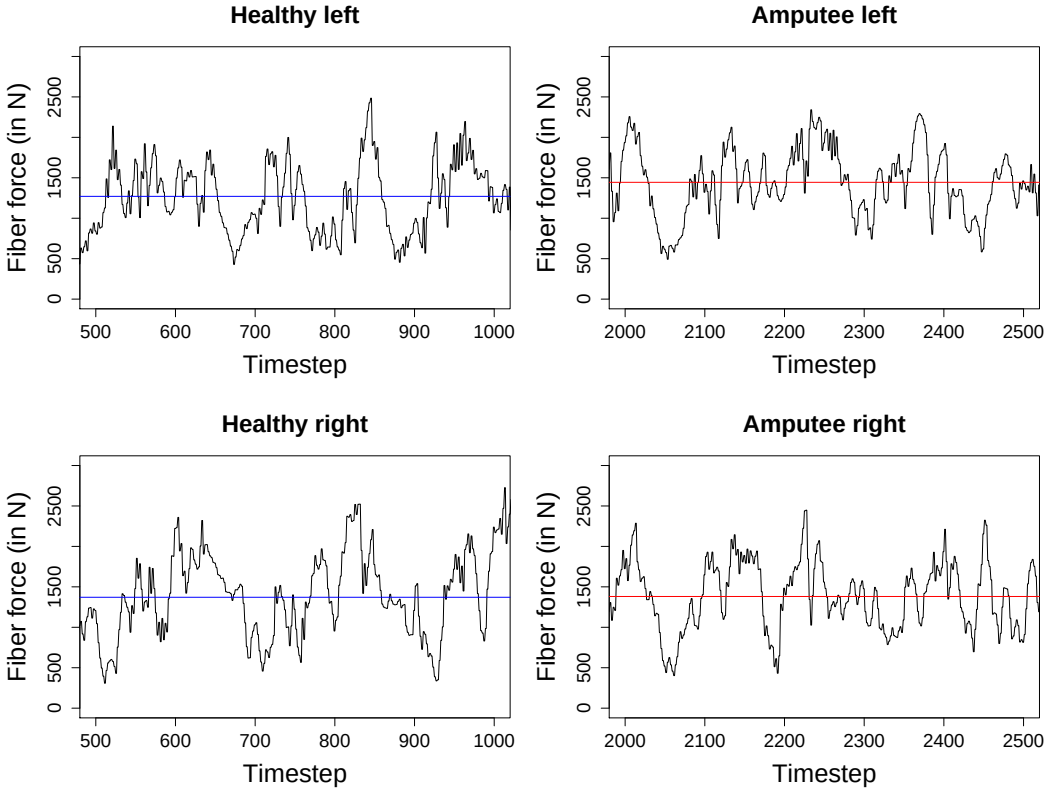


(b) Vasti

Figure 5.5: Fiber forces for the Short head of the biceps (a) and Vasti (b) over 500 timesteps (3 gait cycles). The blue and red lines note the mean fiber force over the time period. It can be seen that in general, that the muscles in the amputee model's right leg have a higher fiber force compared to the healthy model's right leg.



(a) Soleus



(b) Tibialis anterior

Figure 5.6: Fiber forces for the Soleus (a) and Tibialis anterior (b) over 500 timesteps (3 gait cycles). The blue and red lines note the mean fiber force over the time period. It can be seen that in general, that the muscles in the amputee model's right leg have a higher fiber force compared to the healthy model's right leg.

References

M. Abdelhady, A. Rashvand, H. Richter, and D. Simon. System identification and control optimization for an active prosthetic knee in swing phase. volume 1. In *Proceedings of the American Control Conference*, Seattle, USA, pp. 857-862, 2017.

R. Alexander. Design by numbers. *Nature*, 412: 591, 2001.

P.K. Ammu, K.C. Sivakumar, and R. Rejimoan. Biogeography-based optimization - a survey. *International Journal of Electronics and Computer Science Engineering*, 2.1:154–160, 2013.

V. Azimi, D. Simon, and H. Richter. Stable robust adaptive impedance control of a prosthetic leg. In *Proceedings of the ASME Dynamic Systems and Control Conference (DSCCC)*, Columbus, Ohio, USA, 2015.

R. Davis, H. Richter, D. Simon, and A. Van den Bogert. Evolutionary optimization of ground reaction force for a prosthetic leg testing robot. In *Proceedings of the American Control Conference*, Portland, Oregon, pp. 4081-4086, 2014.

S.L. Delp, F.C. Anderson, A.S. Arnold, P. Loan, A. Habib, C.T. John, E. Guendelman, and D.G. Thelan. Opensim: Open-source software to create and analyze dynamic simulations of movement. *IEEE Transactions on Biomedical Engineering*, 55:1940–50, 2007.

E. Elbeltagi, T. Hegazy, and D. Grierson. Comparison among five evolutionary-based optimization algorithms. *Advanced Engineering Informatics*, 19:43–53, 2005.

C. Ferreira, F. Dzeladini, A. Ijspeert, L. P. Reis, and C. P. Santos. Development of a simulated transtibial amputee model. In *Proceedings of the IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, Porto, Portugal, pp. 1-6, 2019.

A. Galloy, K. Frisch, and D. Schmitz. Planning an opensim simulation. https://simtk-confluence.stanford.edu:8443/display/OpenSim/Planning+an+OpenSim+Simulation, 2018.

T. Geijtenbeek and N. Pronost. Interactive character animation using simulated physics: A state-of-the-art review. *Computer Graphics forum*, 31 (8):1–25, 2012.

T. Geijtenbeek, M. van de Panne, and A.F. van der Stappen. Flexible muscle-based locomotion for bipedal creatures. *ACM Trans. Graph.*, 32:1–11, 2013.

H. Hansen. The cma evolution strategy: A comparing review. *StudFuzz*, 192:75–102, 2006.

V.J. Harandi, D.C. Ackland, R. Haddara, L.E.C. Lizama, M. Graf, M.P. Galea, and P.V.S. Lee. Gait compensatory mechanisms in unilateral transfemoral amputees. *Medical Engineering and Physics*, 77:95–106, 2020.

K. Kameyama. Particle swarm optimization: A survey. *IEICE Transactions on Information and Systems*, 92(7):1354–1361, 2009.

K.D. Katyal, E.W. Staley, M.S. Johannes, Wang. I., A. Reiter, and P. Burline. In-hand robotic manipulation via deep reinforcement learning. *30th Conference on Neural Information Processing Systems*, 1:1–5, 2016.

L. Kidzinski, S.P. Mohanty, C. Ong, J. Hicks, S. Francis, S. Levine, M. Salathe, and S. Delp. Learning to run challenge: Synthesizing physiologically accurate motion using deep reinforcement learning. In *NIPS 2017 Competition Book*. Springer, 2018.

L. Kidzinski, C. Ong, S.P. Mohanty, J. Hicks, S. Carroll, B. Zhou, ..., and W. Jaskowski. Artificial intelligence for prosthetics: Challenge solutions. *The NeurIPS'18 Competition: From Machine Learning to Intelligent Conversations*, 69: 1–49, 2019.

A.K. LaPre, M.A. Price, R.D. Wedge, B.R. Umberger, and F.C. Sup. Approach for gait analysis in persons with limb loss including residuum and prosthesis socket dynamics. *International Journal for Numerical Methods in Biomedical Engineering*, 34(4):1–11, 2018.

D.P. Laroche, E.D. Millett, and R.J. Krailian. Low strength is related to diminished ground reaction forces and walking performance in older women. *Gait Posture*, 33(4):668–672, 2011.

M. Mudigonda, P. Agrawal, M. Deweese, and J. Malik. Investigating deep reinforcement learning for grasping objects with an anthropomorphic hand. In *Proceedings of the International Conference on Learning Representations, pp. 1-5*, 2018.

C.F. Ong, T. Geijtenbeek, J.L. Hicks, and S.L. Delp. Predicting gait adaptations due to ankle plantarflexor muscle weakness and contracture using physics-based musculoskeletal simulations. *PLOS Computational Biology*, 15:10, 2019.

E.C. Ranz, J.M. Wilken, D.A. Gajewski, and R.R. Neptune. The influence of limb alignment and transfemoral amputation technique on muscle capacity during gait. *Computer methods in BiomeChaniCs and BiomediCal engineering*, 20:1167–1174, 2017.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *OpenAI*, 2017.

M.H. Schwartz, Z. Rozumalski, and J.P. Trost. The effect of walking speed on the gait of typically developing children. *Journal of biomechanics*, 41: 1639–1650, 2008.

D. Simon. Biography-based optimization. *IEEE Transactions on Evolutionary Computation*, 12: 702–713, 2008.

K.M. Steele, A. Seth, J.L. Hicks, M.S. Schwarts, and S.L. Delp. Muscle contributions to support and progression during single-limb stance in crouch gait. *Journal of Biomechanics*, 43:2099–2105, 2010.

D.G. Thelen. Adjustment of muscle mechanics model parameters to simulate dynamic contractions in older adults. *Journal of Biomedical Engineering*, 125(1):70–77, 2003.

G. Thomas, S. Szatmary, T. Wilmot, and D. Simon. Evolutionary optimization of artificial neural networks for prosthetic knee control. *Efficiency and Scalability Methods for Computational Intellect*, 1:142–161, 2013.

M.M. Van der Krogt, S.L. Delp, and M.H. Schwarts. How robust is human gait to muscle weakness? *Gait Posture*, 36:113–119, 2012.

G. Vasan and P.M. Pilarski. Learning from demonstration: Teaching a myoelectric prosthesis with an intact limb via reinforcement learning. In *Proceedings of the International Conference on Rehabilitation Robotics*, London, United Kingdom, pp. 1457-1464, 2017.

K. Yin, M. Pang, K. Xiang, and C. Jing. Optimization parameters of pid controller for powered ankle-foot prosthesis based on cma evolution strategy. In *Proceedings of the IEEE Data Driven Control and Learning Systems Conference*, Enshi, Hubei Province, China, pp. 175-179, 2018.

A Appendix: Additional tables

			Healthy		Prosthetic	
	Name	**Primary function**	**Left**	**Right**	**Left**	**Right**
1	Bifemsh (Biceps femoris)	Knee flexion	Yes	Yes	Yes	Motor
2	Gastrocnemius	Knee flexion and ankle extension	Yes	Yes	Yes	No
3	Gluteus maximus	Hip extension	Yes	Yes	Yes	Yes
4	Hamstrings	Hip extension and knee flexion	Yes	Yes	Yes	No
5	Iliopsoas	Hip flexion	Yes	Yes	Yes	Yes
6	Rectus femoris	Hip flexion and knee extension	Yes	Yes	Yes	No
7	Soleus	Ankle extension (plantarflexion)	Yes	Yes	Yes	Motor
8	Tibialis anterior	Ankle flexion (dorsiflexion)	Yes	Yes	Yes	Motor
9	Vasti	Knee extension	Yes	Yes	Yes	Motor
10	Hip abductors	Hip abduction (away from body)	No	No	Yes	Yes
11	Hip adductors	Hip adduction (toward body)	No	No	Yes	Yes

Table A.1: A list of muscles that describe the muscles in both the healthy subject model from Kidzinski et al. (2018) and the adjusted transfemoral amputee model. Note that the biarticular muscles are not present in the amputee model.

	Healthy	**Amputee**
Position body parts	13	13
Rotation body parts	13	13
Velocities body parts	13	13
Rotational velocities of body parts	13	13
Accelerations of body parts	13	13
Rotational accelerations of body parts	13	13
Positions of joints	17	17
Velocities of joints	17	17
Accelerations of joints	17	17
Muscle forces	72	76
Miscellaneous forces	13	13
Total size state vector	214	218

Table A.2: The values included in the state observation vector

Article	Inputs	Parameters	Outputs	Objective function	Kind of validation	Application
Covariance Matrix Adaptation (CMA-ES, Hansen (2006))						
Geijtenbeek et al. (2013)	Activation state, total muscle length	Muscle routing, physiological properties, attachment points	Muscle excitation	Minimize error: speed, head orientation / velocity, slide, effort	Finding a stable controller within 300 generations	Model 3D bipedal characters
Yin et al. (2018)	Step signal, torque approximation	Peak torque, peak torque time, rise times, fall times	Muscle excitation	Proportional / integral / differential gain	Lower overshoot compared to original PID	Powered ankle-foot prosthesis
Ong et al. (2019)	Initial state, simulation performance	Gains, offsets, and transition	Muscle excitation	Minimize cost, avoid falling, head stability	RMSE and normalized cross-correlation experimental data	Healthy human and ankle weakness model
Biogeography-Based Optimization (BBO, Simon (2008))						
Davis et al. (2014)	Clinical human gait data	Ground reaction forces	Motor torques	Minimize error: difference GRF's robot vs. reference	RMSE GRF's robot data vs. reference data	Prosthetic leg testing robot
Abdelhady et al. (2017)	Reference velocity	Motor controller: center of mass, shank length	Angular velocity	Minimize error: system identification vs. control	Integral square error system identification vs. control	Active prosthetic knee
Particle Swarm Optimization (PSO)						
Ferreira et al. (2019)	Muscle, joint over-extension prevention, ground sensory data, stability feedback	Stance and swing time	Sensor and joint outputs	Distance travelled, penalty for falling / velocity	Ankle angles and torques model vs. experimental data	Simulated transtibial amputee model
Azimi et al. (2015)	Generalized joint displacements	Joint displacements, velocities, accelerations	Optimal design parameters	Control signal magnitudes and tracking errors	RMSE control signal magnitudes and tracking errors	Active transfemoral prosthetic leg
Deep Reinforcement Learning (DRL)						
Katyal et al. (2016)	Raw image pixels from simulator	Linear and angular velocities of ball	Eight possible actions	Positive rotation of ball along the z-axis	Increase in average reward over time	Robotic hand
Vasan and Pilarski (2017)	EMG signals from the user	The system's weight vectors	Continuous actions	Minimize error in joint angles	Ability to achieve desired joint angles	Powered prosthetic arm
Mudigonda et al. (2018)	Position and velocity of the joints and object position	The neural network's weight vectors	Continuous actions	Minimize distance between the hand and the object	Increase in average return over time	Anthropomorphic hand
Kidzinski et al. (2018)	Joint angles and positions	The neural network's weight vectors	Continuous actions	Amount of meters traveled, penalty for overusing ligaments	Increase in average reward over time	Transtibial amputee simulations

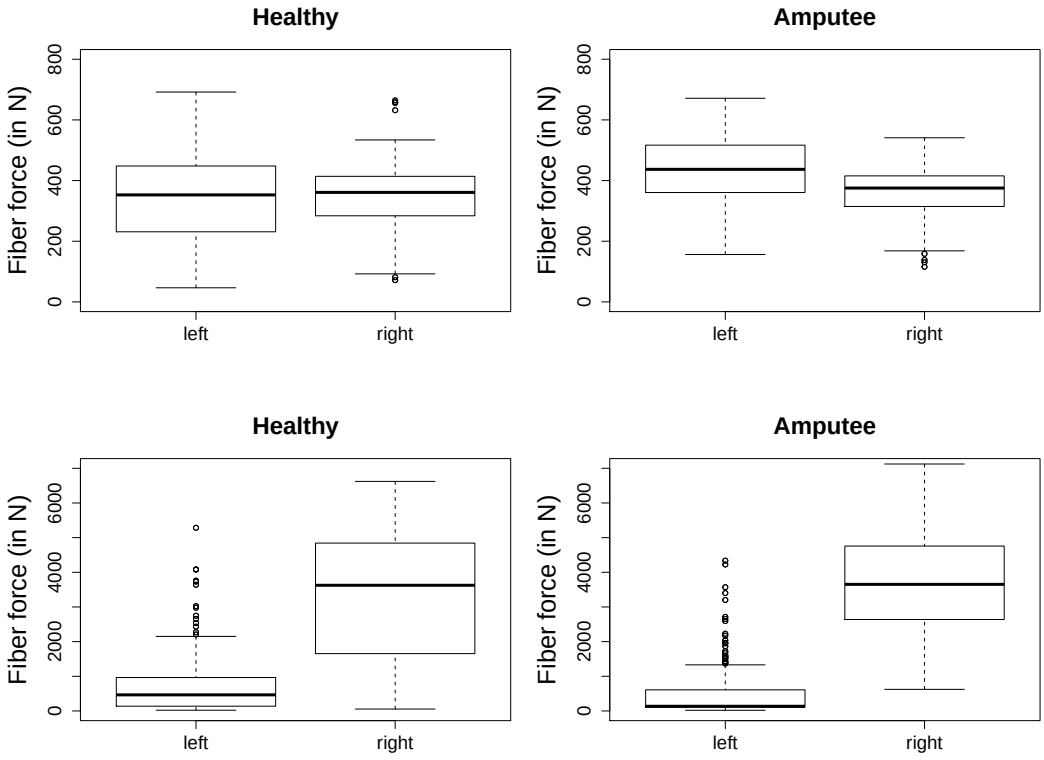
Table A.3: Overview of articles applying the 4 different algorithms from the theoretical background.

		Healthy			Transfemoral		
	Muscle	**Left**	**Right**	**Δ**	**Left**	**Right**	**Δ**
Activation mean	Bifemsh	0.644	0.657	**0.013**	0.772	0.688	**0.084**
	Vasti	0.123	0.444	**0.321**	0.069	0.486	**0.417**
	Soleus	0.323	0.282	**0.041**	0.327	0.294	**0.033**
	Tibialis anterior	0.699	0.725	**0.026**	0.786	0.689	**0.097**
Total difference in means		**0.401**			**0.631**		
Fiber force mean	Bifemsh	352.448	348.026	**4.422**	425.253	364.651	**60.602**
	Vasti	722.891	3305.804	**2582.913**	471.529	3728.453	**3256.924**
	Soleus	1267.526	1179.705	**87.821**	1536.586	1330.320	**206.266**
	Tibialis anterior	1270.542	1371.137	**100.595**	1443.990	1380.933	**63.057**
Total difference in means		**2775.751**			**3586.849**		

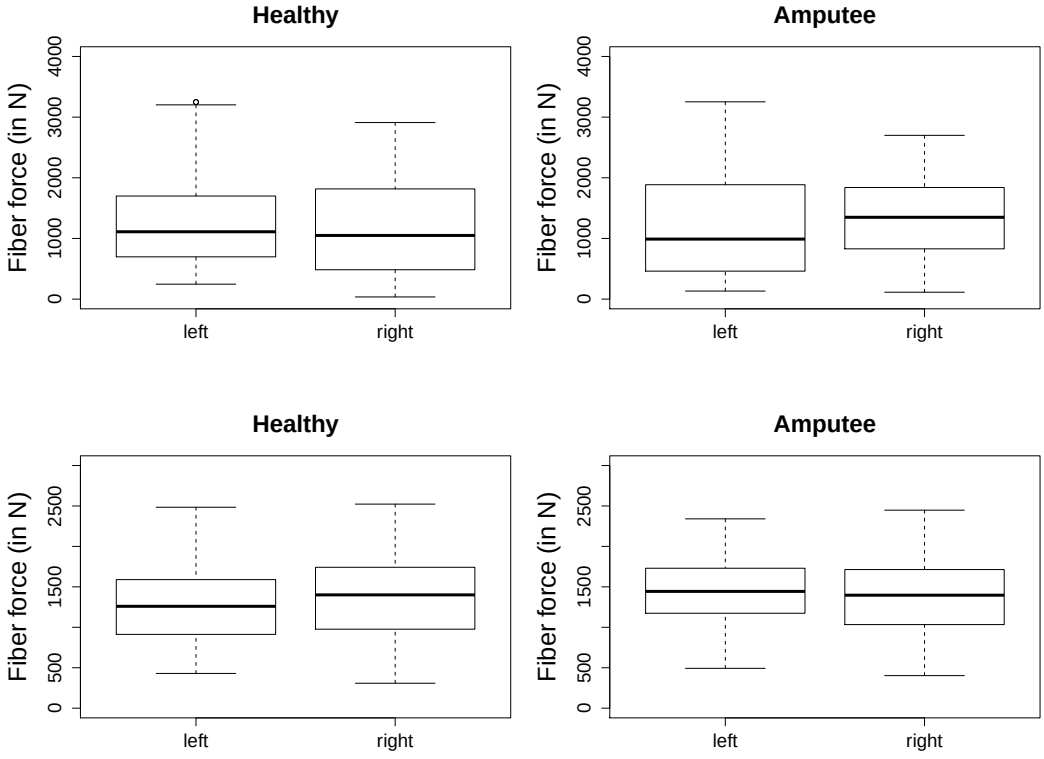
Table A.4: Results for the comparison of the difference in muscle usage between the healthy and simulated transfemoral prosthetic leg. We compared the mean in muscle activation as well as the mean in fiber force for the four muscles around the knee and ankle. For each of the models, the difference in left and right leg were computed and added to return a total difference in means. The results confirm our hypothesis that for the transfemoral model, both activation and fiber force are higher for right leg and the difference between legs is higher compared to the healthy model.

Parameter	**Value**
Parallel runs	4
Iterations	895
Episodes	20,000
Steps	5,000,000
Training time [hours]	12
Workers	4
Steps per worker	1,536
Steps per action	4
Entropy coefficient	0.01
Parameter noise	yes
Policy networks	1
PPO clip parameter (ϵ)	0.2
PPO batch size	512
PPO optimizations per epoch	4
PPO entropy coefficient	0.01
PPO δ	0.9
PPO γ	0.999
Neural network amount	2
Neural network size	312
Activation function	tanh

Table A.5: The hyperparameters and specifications used for the PPO learning algorithm during training.



(a) Short head of the biceps femoris (above) and Vasti (below)



(b) Soleus (above) and Tibialis anterior (below)

Figure A.1: Comparison of distribution and mean fiber forces for the Short head of the biceps femoris ((a) above), Vasti ((a) below), Soleus ((b) above) and Tibialis anterior ((b) below) over 500 timesteps (3 gait cycles). The blue and red lines note the mean fiber force over the time period. It can be seen that in general, that the difference in mean fiber force between the left- and right leg is higher in the amputee model compared to the healthy model.

B Appendix: Implementation details

The code from this project is based upon Stanford University's NeurIPS challenges which combines reinforcement learning with musculoskeletal OpenSim models. More information about this environment can be found on http://osim-rl.stanford.edu/. The algorithm in this project is largely inspired by the 7th-placed solution to the 2018 NeurIPS AI For Prosthetics competition: http://osim-rl.stanford.edu/docs/nips2018/. OpenAI's Baselines implementation of PPO serves as the basis for the learning algorithm. To speed up training, state trajectories at different walking speeds are included in the osim-rl/osim/data folder.

The project is written in Python. The main files that have been altered for this project include *main.py*, *osim.py*, *pposgd_simple.py*, and the *models* folder.

Firstly, in *main.py* the training algorithm is called by *pposgd_simple.learn()*. The function passes all the learning parameters which can be found in table A.5 (Appendix A). For the algorithm, we use OpenAI's Baselines implementation of PPO from the *ppo1* folder. The PPO algorithm is implemented following these steps:

1. Two neural networks (class *MlpPolicy*) are initialized: one for the new and one for the old policy. These neural networks have four layers: one input layer (size *ob_space*), two hidden layers (size 312), one output layer (size *ac_space*).
2. A target advantage function and empirical return are initialized.
3. The losses are computed following function (3.2 and 3.3).
4. The two neural networks are updated following the losses.

After each training iteration in the simulation, a summary is printed containing:

- **EpLenMean**: Mean length of episodes.
- **EpRewMean**: Mean reward of episodes, following the loss function.
- **EpThisIter**: Episode number during this iteration.
- **EpTrueRewMean**: Mean true reward of episodes, following the overall objective function.
- **EpisodesSoFar**: Number of episodes so far.
- **TimeElapsed**: Elapsed time in seconds.
- **TimestepsSoFar**: Number of timesteps so far.
- **ev_tdlam_before**: Explained variance between predicted value function and temporal difference (λ) estimator.
- **loss_ent**: Entropy loss.
- **loss_kl**: Kullback-Leibler loss.
- **loss_pol_entpen**: Policy entropy penalty loss.
- **loss_pol_surr**: Policy surrogate loss.
- **loss_vf_loss**: Value function loss.

In *osim.py*, several alterations are made to implement the goals of this project:

- The observation function. Observations have been added for all the body parts of the healthy subject model.
- The reward function. Rewards have been added for all the hip, knee, ankle muscle positions and velocities so it works well with the healthy model.
- The reward function now returns 65% on imitation and 35% on goal reward.
- Two models have been added to the models directory: *healthy_subject_model_customized*, *transfemoral_amputee_model_customized* and *transfemoral_amputee_model_customized_2*. The second is the final model that I used. The implementation section in this paper explains how the two models were created.
- Function that gets the observation space size. This is dependent upon which model you are using. It should be 214 for the healthy subject model and 218 for the transfemoral amputee model.

Furthermore, because there is a memory leak in the original OpenSim environment, a separate runfile was created. The memory leak disrupted training as it slowed down considerably when RAM memory of the computer got full. Therefore, the *run_command_file.py* ensures that the training is restarted every 3 hours. The current training parameters are written to separate csv files which are loaded once the training restarts. This ensures that no training progress is lost while restarting the process.

The code can be tested using these commands:
- *source activate opensim-rl*: activates the opensim reinforcement learning environment.
- *python run_command_file*: runs the file containing more specific running commands.