university of groningen / faculty of science and engineering

Level Ground Walking for Healthy and Transfemoral Amputee Models.Deep Reinforcement Learning with Phasic Policy Gradient Optimisation

Bachelor's Project Thesis

Efstratios Mytaros, s3000370, e.mytaros@student.rug.nl
Supervisors: Prof. Dr. Raffaella Carloni, Vishal Raveendranathan, MSc*

Abstract: This paper proposes to use deep reinforcement learning for physics based musculoskeletal model simulations of both healthy and transfemoral prosthesis models, during level-ground walking. The deep reinforcement learning algorithm is based on the phasic policy gradient framework, which expands the proximal policy optimisation method with alternations to its auxiliary components. Initially, the optimisation is carried out on the healthy models, followed up by optimizations on the transfemoral model. The agent was able to develop gait similar to that from a given experimental data set, opening the doors for using the algorithm for efficient simulations of physical models. The results so far suggest a performance equivalent and potentially even better than previous iterations.

1 Introduction

Using computer simulations to represent and explore the capacities of physical and physics-based muscoloskeletal, allows us to discover and test systems that could otherwise take a much longer time and effort to do so. The accuracy and speed of such simulations, grants the opportunity to test multiple aspects, much faster and with control over detailed models.

This paper focuses on DRL by applying the phasic policy gradient (PPG) algorithm, through investigating the performance of the algorithm proposed by Cobbe, Hilton, Klimov, and Schulman (2020), based on research conducted previously by de Vree and Carloni (2020). Prior research focused on adapting findings on similar research conducted by ukasz Kidziski et al (2019) in the NeurIPS competition, on the OpenSim (Scott and et al (2007)) platform, using the healthy models proposed in it and adding to it, a transfemoral amputee model implementation.

The proposed approach includes changes to proximal policy optimisation (PPO) by Schulman, Wolski, Dhariwal, Radford, and Klimov (2017), inspired by the PPG algorithm, by Cobbe et al. (2020), by expanding upon it through the addition of extra features. These changes include the restructuring of the network in use, as well as the addition of an auxiliary learning phase in the training of the network.

The goal is to test whether the use of PPG would be able to allow the agent to learn a forward walking gait. Particularly, implementing the fresh PPG algorithm in a biomechanics topic and compare it to its predecessor, PPO. Due to the recency of this algorithm, it has yet to be implemented in such a method, which will hopefully offer more refined resulting actions from the model, due to its use of an auxiliary phase to refine the value of the original PPO algorithm.

Furthermore, for this additional phase to have an even greater effect, the aim is to alter the auxiliary loss, to a more suitable objective, taking into account what the auxiliary objective of the algorithm in this case could be. Based on Cobbe et al. (2020), the auxiliary objective of the loss, can be tailored to fit the given environment. Considering

the performance of algorithms included in both the NeurIPS'17 ukasz Kidziski, Mohanty, Ong, Hicks, Carroll, Levine, Salath, and Delp (2018) and NeurIPS'18 competitions, certain aspects are observed to assist in successful gait generations. There are examples of off-policy algorithms yielding satisfactory results, usually associated with the use of an experience buffer, as well as the use of separate policies for trajectory generation and learning. By combining the characteristics that seem to allow for good performances with algorithms such as TRPO and PPO, with those of algorithms like DDPG and A3C, the version of PPG suggested in this paper has the potential to perform very well in its environment.

The overall aim is to generate comparable gait patterns, by introducing aspects of the new, adapted, PPG algorithm to biomechanic human walking simulations. With the effect of the auxiliary phase, that enhances the effect of the proposed reward to the system, this algorithm is suitable for robotics tasks and generating stable gait patterns for our forward walking model.

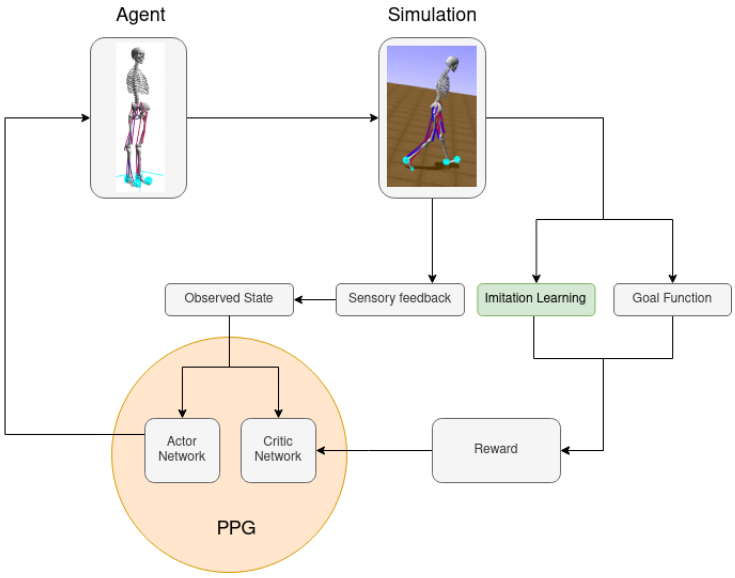


Figure 1.1: An overview of the system used in this study

The designed healthy and transfemoral amputee models, or agent, is being run through a simulation environment (OpenSim). The agent interacts with the environment, by receiving information about the state of the world around it and feedback on its actions through a goal and imitation component. The rewards and observations from the environment serve as core information in updating the algorithm and its networks. Taking these into account, the algorithm produces an action for the agent to take, through which it will interact with its environment once again, and the cycle repeats. To assist with the visualisation of what the overall proposed architecture looks like, Figure 1 can be used as a reference. OpenSim will be the platform in which the simulations of our models/agents will be running. It makes for an excellent simulation tool due to its specialty in detailed muscoloskeletal model design and similar previous work. Those simulations will yield some muscle and joint states, as well as a reward, based on the goal and gait functions, as well as the imitation learning component, which will then be fed into the PPG DRL algorithm. The algorithm will then generate an appropriate action and parameters for the OpenSim controller to simulate the forward dynamics of our agent.

The rest of the paper discusses the theoretical background in section II. This includes some information about the OpenSim environment, explanations on the gait equations and DRL algorithms. Section III outlines the methodology of this study, with information about the DNN used, the optimiser and the reward of the system for the forward motion gait. Section IV will present and discuss the results yielded from the research. The final section, section V, will be the conclusion drawn from this research.

2 Background Literature

The process of information acquisition and relevant theoretical background collected during the research will be outlined in three sub-sections, the models used in previous research, the algorithms investigated relative to the task and finally, research focused on gait patterns.

2.1 DRL for Simulations of Physics Based Models

The field of walking and gait generation has a variety of different tools that can be used. The nature

of the environment being used, has an effect on the algorithm choice and its performance in it.

Some are focused on particular applications, such as the NAO robots, which provide a simulation environment that can be translated to the physical robot as well. Research by Cristyan R. Gil (2019) explored the learning of efficient gait patterns in the NAO robots, by adapting the poses that the agent can take, to a DRL generated gait pattern. Due to the poses available from the NAO robot, they were able to use Q-learning to generate discretely available poses to be put together and compile a complete and efficient gait cycle.

There are examples of research, where the model is applied to a simulated environment, where the agents can have more limbs to control, as well as train on complex environments, such as the study by Azayev and Zimmerman (2020). A blind hexapod agent, in order to train it across different terrains, applying a DRL algorithm to train its walking skills. This study applied the needs of the desired gait to be generated in a MuJoCo environment. Another study in this environment is Nagabandi, Kahn, Fearing, and Levine (2017), which also uses a combination of model based and model free algorithms to investigate the gait of different agents with varying limbs and joints. MuJoCo allows for greater freedom in simulating the desired models and adapting them to the tasks they should be proficient in. et al (2016) create multiple agents in their study of goal driven motion to train them with DRL algorithms in a MuJoCo environment, including humanoid walkers.

2.2 OpenSim Models

Simulations that allow for greater detail and specificity in the realm of muscoloskeletal models, include the OpenSim framework Scott and et al (2007). This allows for realistic representations of muscles and joints present in human anatomy. It has been used by van der Krogt, Delp, and Schwartz (2012) to generate muscle driven simulations, where specific alterations were made to muscle groups to vary their forces and excitations to test the limits of normal gait generation.

Since OpenSim allows for such detail in the models that can be created for human anatomy, it as been used by the ukasz Kidziski et al. (2018) and ukasz Kidziski et al (2019) competitions. Given the application of the research, having detailed representations of the human muscoloskeletal system and access to particular activations and forces that would enable the generation of realistic forward motion seems advantageous to other methods. Similar to the competitions from past years, the algorithms proposed will generate forward walking gait, by activating specific muscles.

Other methods like the NAO and general MuJoCo simulations, although useful and reliable in certain scenarios, can provide a lot of insights about methods and approaches to gait generation. OpenSim however provides a much more fitting platform to conduct experiments in, as it offers unique features, suitable for this study.

2.3 Algorithm Basics

All algorithms found across different DRL frameworks, share some common characteristics. These are the building blocks upon which they are all constructed and developed, while having particular aspects in mind. Since a lot of these aspects are shared along different algorithms, it makes sense for them to be established.

A few definitions will be outlined, in order to establish some terminology that will be used throughout this paper. Initially, there are mentions of the term, policy π, referring to the next action that the model will be generating during its learning phase. Every action, or policy, is being calculated by taking the current environment state into account. Therefore a state s refers to a description of the environment perceived by the agent in order to produce its corresponding policy. Each time the agent performs under a certain policy for a given state, the value $V(s)$, or utility of that state is being calculated and predicted. This is a quantification and estimation of the "goodness" of the state by the agent, known as the value. The agent will thus try to learn the value function through its experiences during its learning phase.

Consequently, the means of determining the policy will dictate the prediction of the value function. In general terms, there are two ways of generating a policy. One case is determining the probability of each potential policy yielding a high value. The result will be a list of policies, one or more of which would be the best choice, given the current state. By selecting the policy with the highest

probability of yielding a high value, our algorithm will act in an on-policy manner. On-policy means that the same policy determined by the deliberation after the learning phase, is the one used to adjust the value function of our agent. The second case will also use this better policy selection based on the highest returning value, but with an added chance of it behaving in some random way from a newly generated policy, unrelated to the ones determined by the learning phase. Such a behaviour is described as an off-policy method, due to the use of a policy other than the one generated by the learning phase prior to the calculation. Both these methods guarantee some sort of exploration of new states and policies for the agent, as well as exploitation of already known policies. These definitions are based on the book by Sewak (2019b)

One application of this sort of algorithm is being used for Deep Q Networks (DQN), algorithm, as described in the book by Sewak (2019b). The method used for obtaining the loss for the network weights is off-policy, since it needs to have a separate behaviour policy for its trajectories. using the following equation, it can have a very good estimate of the value of the system:

$$Q_{(s,a)} = (1-\alpha)Q_{s,a)} + \alpha(r + \gamma \max_{\alpha'} Q_{(s',a')}) \quad (2.1)$$

DQN optimises its policy based on the update of equation 2.1, but learns by updating a separate policy π. The policy optimisation is more sophisticated usually than the trajectory π, which is usually some trivial selection like ϵ-greedy, giving an ϵ chance for the agent to act randomly, in order to encourage exploration.

One application of the policy being the primary focus is called REINFORCE, or Monte-Carlo policy gradient, conceived by Williams (1992). In this case the policy is adjusting the vector parameters of its network, denoted by θ, by using a combination of characteristics mentioned so far. Using the Monte Carlo approach, it uses the estimated return to update θ, based on the learning phase trajectories, the timesteps taken within the episodes during that phase. The policy loss is calculated using:

$$\nabla_\theta J(\theta) = \epsilon[Q^\pi(s,a)\nabla_\theta ln \pi_\theta(a|s)] \quad (2.2)$$

In this equation, the states (s) and actions (a) are measured from real model trajectories and use them to directly update the policy gradient parameters θ. It is important to keep this characteristic of this algorithm in mind for the ones that will follow.

REINFORCE, is a rather simple policy method algorithm that has its shortcomings. Notably, it does not take the value at all into consideration, like DQN would. In order to improve on such algorithms, a hybrid version of the policy and value methods can be used. Value and policy methods coincidentally complement one another because of their respective strengths and weaknesses being accounted for. This hybrid is known as actor-critic methods. These algorithms commonly consist of two networks, one for actor, or policy network and one for the critic, or value network. These networks work the same as the formerly mentioned examples, using a loss function to update the network weights through gradient descent in order to reach some optimal policy or value.

2.4 Actor-Critic Algorithms

These two concepts can be combined into one, in order to maximise the performance of both according to Konda and Tsitsiklis (2000). A critic-only algorithm needs a good behaviour policy, since they rely on value approximations that hopefully results in some optimal policy. A policy-only algorithm needs a means of reducing the large variance of the gradient estimators, and properly utilise the cumulative knowledge acquired from past experiences. Therefore, the actor uses an approximation of the gradient made by the critic.

A later application of this concept is found in the Asynchronous Advantage Actor-Critic (A3C), by Sewak (2019a). The critic network is trained through multiple agents across parallel processes that synchronise every so often. Such an approach works optimally with multiple agents, but is not restricted to it. The actor-network can use any behavioural policy, according to the paper, even varied across processes. It uses a Mean Squared Error (MSE) loss between the predicted value and returned value for its gradient descent and update the value weights (w), as well as policy (θ):

$$J_v(w) = (V_t(s) - V_w(s))^2 \quad (2.3)$$

Similar actor-critic implementations include Soft Actor Critic by Haarnoja, Zhou, Abbeel, and

Levine (2018). This is an early, yet effective adaptation of the novel actor-critic architecture, adding the entropy measure of the policy into the reward of the model. In doing so, it achieves more random behaviour that encourages exploration, yet still maintaining its ability to succeed in its task. Utilising both off-policy characteristics, allowing the reuse of past data and experiences, as well as the entropy factor, it achieves good sample efficiency and exploration of different policies. This algorithm has been applied in a context of robotics locomotion by Haarnoja, Ha, Zhou, Tan, Tucker, and Levine (2019). The primary focus of the study has been to emphasise the usefulness of sample efficiency in real-life robotics applications. Driven by the need to reduce the number of real life experiences that an actual mechanism should use due to its fragile nature and the need to maintain its structural integrity, they proceeded to make a case about the advantages of the sample efficiency brought about by the experience buffers and subsequent randomness from the entropy. This combination made for quick learning with minimal risk for the physical robot due to its trial and error behaviour while learning.

Something to always consider when implementing an algorithm, is its learning stability. More stable systems avoid parameter updates that impose too large of a change. The algorithm known as Trust Region Policy Optimisation (TRPO), by Schulman, Levine, Moritz, Jordan, and Abbeel (2015), tackles this issue, by enforcing a KL divergence constraint on the policy update. Since the true reward of each trajectory cannot be extracted directly, in most of these algorithms, we use the Generalised Advantage Estimate (GAE), for the trajectory cumulative value calculation. Considering the asynchronicity assumed also in A3C, TRPO could be running parallel agents with each performing under a different policy in each world, but optimised with a common one during its update. Let's denote the old policy from the trajectory generation as $\pi_{\theta_{old}}(a|s)$, for an action a in state s and similarly, the optimisation policy as $\pi_\theta(a|s)$. For its loss function, the objective function uses the trust region constraint, limiting the distance between $\pi_{\theta_{old}}(a|s)$ and $\pi_\theta(a|s)$, as measured by the KL divergence, within some bounds δ:

$$KL = D_{KL}(\pi_{\theta_{old}}(\bullet|s)||\pi_\theta(\bullet|s)) \leq \delta \tag{2.4}$$

$$J(\theta) = \mathbb{E}_{s\sim\rho^{\pi_{\theta_{old}}},\alpha\sim\pi_{\theta_{old}}}[KL\hat{A}_{\theta_{old}}(s,a)] \tag{2.5}$$

where equation 2.5 shows the discounted state distribution of policy θ_{old}, as $\rho^{\pi_{\theta_{old}}}$. Combining equation 2.4 with 2.5, shows the loss equation for the policy update of TRPO.

An updated version of TRPO, is Proximal Policy Optimization (PPO), which is the algorithm originally used in this research. To avoid the complicated mathematical proof hidden behind TRPO, PPO takes a simpler, yet similar approach to constraining the policy update. Instead of using the KL divergence between two policies, it simplifies it by using a clipped surrogate objective. Following the definition by Schulman et al. (2017), we initially take the probability ratios between old and new policies and denote it as shown in equation 2.6:

$$r(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} \tag{2.6}$$

The objective loss is almost identical, except for equation 2.4 from 2.5, being substituted by 2.6:

$$J(\theta) = \mathbb{E}_{s\sim\rho^{\pi_{\theta_{old}}},\alpha\sim\pi_{\theta_{old}}}[r(\theta)\hat{A}_{\theta_{old}}(s,a)] \tag{2.7}$$

Once again, same as in TRPO, $\hat{A}_{\theta_{old}}(s,a)$, is an advantage estimate, mostly approximated by GAE. The ratio $r(\theta)$ is clipped around an area of $[1-\epsilon, 1+\epsilon]$, where ϵ is one of the hyperparameters. Adding the clip to 2.7, we get:

$$J(\theta) = \mathbb{E}[\min(r(\theta)\hat{A}_{\theta_{old}}(s,a), clip(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_{\theta_{old}}(s,a))] \tag{2.8}$$

The value is being updated using the MSE loss as shown in equation 2.3, in this case in the form of $(V_w(s) - V_{target})^2$, where w are the value network weights. It accounts for exploration by using an entropy term sampled from the policy distribution.

For further improvements on the performance of PPO, the separation of the networks for policy and value training would reduce interference between the common network parameters for both

value and policy learning. Benefiting from both parameter sharing, as well as reducing the competing objectives between the two, can be achieved through the use of two separate networks. One network can be designated as the critic network, yielding the values, while the other would be the actor network, not only outputting a policy, but also a value. Such a solution is achieved by the Phasic Policy Gradient (PPG) algorithm, by Cobbe et al. (2020). The networks are trained separately according to the clip loss described in equation 2.8 for the actor (parameters denoted by θ_π) and MSE for the critic(parameters denoted by θ_v). An additional auxiliary loss is calculated during an additional auxiliary phase, which ensures that the shared value predicted by θ_π is also used in the loss. This loss is calculated through the auxiliary loss, which on the paper by Cobbe et al. (2020), is stated that it can be any auxiliary objective, but is given the form:

$$L^{AUX} = \frac{\mathbb{E}_t(V_{\theta_\pi}(s_t) - V_{target})^2}{2} \tag{2.9}$$

by default. The formerly mentioned policy update during the auxiliary phase is using the original TRPO KL divergence from equation 2.5. The overall joint loss of the auxiliary phase looks like this:

$$L^{JOINT} = L^{AUX} + \beta_{clone}\mathbb{E}_t[KL[\pi_{\theta_{old}}(\bullet|s)||\pi_\theta(\bullet|s)]] \tag{2.10}$$

In equation 2.10, β_{clone} is a hyperparameter controlling the distance of the two policies. The frequency of the auxiliary phase does not need to be too regular, since that interferes with policy optimisation. It therefore happens infrequently between the clip optimisations. During those intermediate updates, a buffer B is filled with experiences, similar to the formerly mentioned off-policy methods. Doing so allows PPG to train both with policies from the current running trajectories, as well as with varying policies from previously encountered experiences.

3 Method

This paper builds upon the previously used algorithm by de Vree and Carloni (2020), by adding features found to be helpful in various DRL algorithms used in a similar context. Such features include additions to the neural network in use, for specialising in both the policy and value outputs and including an auxiliary phase that will be guided by an auxiliary objective suitable for the problem at hand. The goal is once again to ensure that both healthy and transfemoral models learn to walk forward on a flat plain. Changes in the networks and loss functions will be outlined in the following subsections.

3.1 The Models

This paper uses the same models that were being used in a previous research by de Vree and Carloni (2020). The healthy model in use consists of a total of 18 muscles, 9 per leg, while the transfemoral model has 8 muscles on its right leg and 11 muscles on the left. The right leg in this case is the transfemoral amputee leg, hence the reduced muscles relative to the left. The muscles in the transfemoral model are positioned in such a way that mimics the mechanical structure of a prosthetic, hence the actuators are around the ankle and knee joints, as seen in Figure 3.1.

3.2 The Imitation/Validation Dataset

Any results yielded by this study are being validated against some experimental data collected by Schwartz, Rozumalski, and Trost (2008). The data was collected on 83 typically developing children by measuring the kinematics and kinetics of the hip, knee, and ankle joints, the surface electromyographic signals, and the spatio-temporal data. Additionally, this data is used by the algorithm as an imitation data set and is incorporated in the reward of the system.

3.3 The Networks

Considering the benefits of algorithms such as Deep Deterministic Policy Gradient (DDPG), by Lillicrap et al. (2019), that uses separate networks for the actor and critic, as well as target networks for each for applications outlined on table 2.1. It also uses a replay buffer R for storing its past experiences and has yielded very promising results in its applications in the NeurIPS competitions in the past, outlined in ukasz Kidziski et al (2019). Since

DRL Algorithm list			
Primary algorithm used	Application	Algorithm update	Source article
DDPG (Deep Deterministic Policy Gradient)	Cartpole swing-up, dexterous manipulation, legged locomotion and car driving	Actor-critic, model-free algorithm based on the deterministic policy gradient that can operate over continuous action spaces	Lillicrap, Hunt, Pritzel, Heess, Erez, Tassa, Silver, and Wierstra (2019)
Soft Actor Critic (SAC)	OpenAI gym benchmark suite, rllab implementation of the Humanoid task	Aim to optimise both policy and q-networks with stochastic gradient descent	Haarnoja et al. (2018)
SAC	Quadrupetal minitaur robot	Same as novel SAC, with entropy scaling over time	Haarnoja et al. (2019)
TRPO	SouthHampton Hand Assesment Procedure (SHAP)	Actor network aims to minimise loss through KL-divergence between old and new policy	Mudigonda, Agrawal, Deweese, and Malik (2018)
PPO	OpenAI Gym [Bro+16]	Similar to TRPO, but uses a clipped surrogate objective and combines value network and entropy components	Schulman et al. (2017)
distributed PPO	Variety of bodies tested in MuJoCo	Basic PPO with data collection and gradient calculation are distributed over multiple workers	Heess, TB, Sriram, Lemmon, Merel, Wayne, Tassa, Erez, Wang, Eslami, Riedmiller, and Silver (2017)
BANG BANG PPO	OpenSim muoskeletal model	Basic PPO with simultaneous agents being reduced over time	ukasz Kidziski et al (2019)
PPG	OpenAI Gym	Basic PPO with disjoint policy and value networks, as well as an auxiliary phase that refines the value given to the policy	Cobbe et al. (2020)

Table 2.1: Table outlining DRL algorithms and their applications

PPO has also performed very well in past iterations, considering changes to the network proposed by Cobbe et al. (2020), there were a few adjustments implemented.

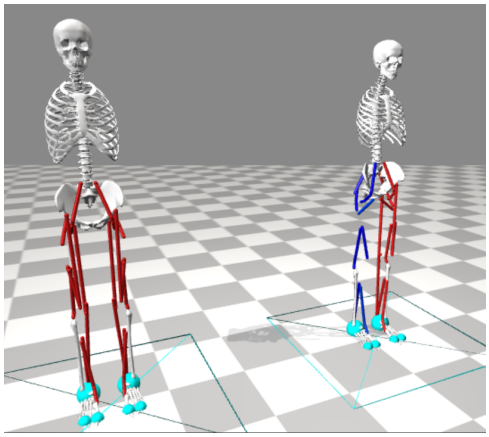


Figure 3.1: The healthy model (left) and transfemoral model (right) used for this paper

The network maintains a similar Multi Layer Perceptron (MLP) structure, as in the study by de Vree and Carloni (2020). Each network is a feed-forward artificial neural network composed of 4 layers. The input layer depends on the observation space of the model in use, 218 nodes for the healthy model and 221 for the transfemoral. It is then followed by 2 hidden layers of 312 nodes each, leading to an output layer of 18 nodes for the healthy model actor-network. The actor-network also has an extra output node, the value V_{θ_π}, sharing the θ_π parameters with the policy output, resulting in a total of 18 + 1 output nodes for the actor. The critic-network with parameters V_{θ_V} has the exact same structure as the actor-network, except the output layer consists of only the value node.

3.4 The Learning Algorithm

For the networks to be properly updated and have their weights optimised to yield the desired actions, a combination of optimisation algorithms is used. The network performs updates based on the clipped update proposed by Schulman et al. (2017), described by equation 2.8. The hyperparameters controlling the clip have been set to an ϵ value of 0.2 and an entropy coefficient of 0.01, determining the scale of the sampled entropy to allow for balanced exploration. These were chosen based on the default recommended values suggested in the original paper by Schulman et al. (2017).

Trajectory generation has been set to a total amount of 1536 timesteps, based on policy π_θ. Optimisation on said gathered trajectories constitutes of 4 training epochs, with the data being split into thirds for batch training per epoch for both value and policy. This part is the policy phase of the algorithm (N_π), which repeats for N_π iterations, until the next phase, the auxiliary phase (E_{aux}) is instigated.

During N_π, the gathered experiences and trajectories are stored in an experience buffer B. For practical reasons of memory consumption on the lab machine, the buffer size was restricted to a quarter of N_π, in order to avoid system crashes. The value of N_π) was set to 64, as Cobbe et al. (2020) state, less frequent updates can lead to less interference, thus the selected value. Following the policy phase, the auxiliary updated was set to fit this environment and set a modified auxiliary goal. This goal was adjusted to:

$$L^{AUX} = \frac{\mathbb{E}_t(1 - e^{-\frac{\beta}{rew_t}})(V_{\theta_\pi}(s_t) - V_{target})^2}{2} \tag{3.1}$$

Equation 3.1 introduces a scaling factor β, which is set to 1.2e-3 and the rew_t reward returned by the environment. The value of rew_t is higher as the agent performs better in the trajectories, and through the scaling it adjusts by how much the auxiliary phase should update the θ_π. The parameter β_{clone} was kept as 1, as suggested in the original paper.

All these parts come together and combined they create an algorithm which can be better visualised through figure 3.3. The auxiliary phase was optimised (E_{aux}) for 6 epochs, in order to avoid overfitting the network. Batch training was also not incorporated to the auxiliary phase, as it would lead to a higher sample reuse over the same data in B. This algorithm structure enables the use of some formerly mentioned aspects of algorithms, such as experience buffers, a refining value update during the auxiliary phase with L^{value}, the use of additional policy restrictions to L^{clip} with L^{joint} to keep the model from having any large policy updates that throw it off.

Algorithm 1 PPG

for phase $= 1, 2, ...$ **do**
  Initialize empty buffer B
  for iteration $= 1, 2, ..., N_\pi$ **do** ▷ Policy Phase
    Perform rollouts under current policy π
    Compute value function target $\hat{V}_t^{\text{targ}}$ for each state s_t
    for epoch $= 1, 2, ..., E_\pi$ **do** ▷ Policy Epochs
      Optimize $L^{clip} + \beta_S S[\pi]$ wrt θ_π
    for epoch $= 1, 2, ..., E_V$ **do** ▷ Value Epochs
      Optimize L^{value} wrt θ_V
    Add all $(s_t, \hat{V}_t^{\text{targ}})$ to B
  Compute and store current policy $\pi_{\theta_{old}}(\cdot|s_t)$ for all states s_t in B
  for epoch $= 1, 2, ..., E_{aux}$ **do** ▷ Auxiliary Phase
    Optimize L^{joint} wrt θ_π, on all data in B
    Optimize L^{value} wrt θ_V, on all data in B

Figure 3.2: Pseudocode describing the PPG algorithm by Cobbe et al. (2020)

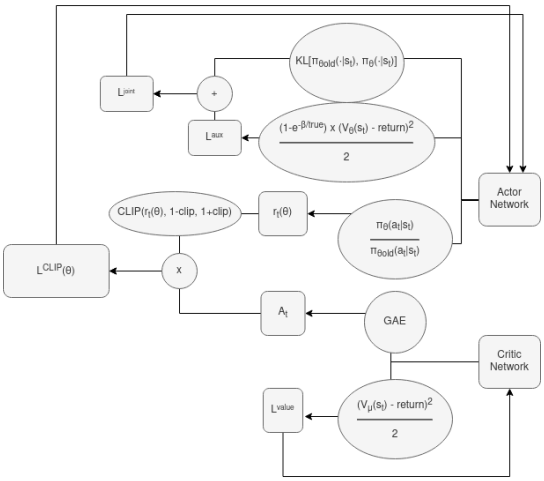


Figure 3.3: A more detailed diagram of the algorithm used in this paper

Lastly, the auxiliary phase includes an additional value update on the critic network. By having an additional value update, the weights of θ_π are updated using the shared value with the updated policy π_{theta}. This updated π_{theta}, occurs after the update based on the experience buffer B, thus the critic parameters of the value network θ_V are also updated on the same data.

4 Results

The results of the proposed algorithm are extracted for both the healthy and transfemoral models. Data validation is performed based on the kinematic results extracted from the two simulated models, against experimental data.

4.1 Algorithm Performance

The following results outline the performance of the algorithm in comparison to the base PPO implementation. Looking at Figure 4.1, the red curve displays the performance of PPG plotted along with the green curve, showing the base PPO performance.

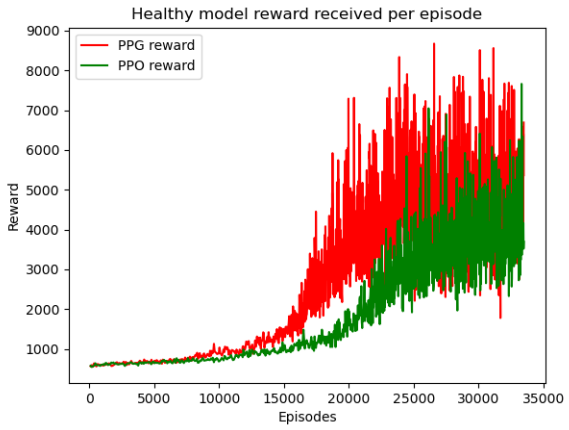


Figure 4.1: Reward received by the PPG algorithm (red), against the standard PPO performance (green) on the healthy model. The reward is plotted on the y-axis and total episodes on the x-axis.

The increase in mean reward received by the model using the PPG algorithm shows a tendency to learn how to develop a rewarding policy more efficiently than the base PPO algorithm. PPG already starts learning a better policy from 15.000 episodes, which then increases at a steady pace and maxes out a little over the PPO curve, shown in figure 4.1. The reward takes around 10.000 episodes less to reach similar heights as well as surpass the

previous implementation. A similar performance can be said for the transfemoral model's results, displayed in figure 4.2, where an optimal policy is found very early, at a comparable number of episodes to that of the healthy model, around 15,000.

	PPO		PPG	
	mean	std	mean	std
Healthy	2724.78	1444.45	3913.29	1781.23
Transfemoral	1797.91	1105.64	3603.93	1733.40

Table 4.1: The mean total reward and standard deviation received by the models following both base PPO and PPG algorithms.

For the healthy model, the mean as increased by almost 1.5 times, when compared to the mean reward of 2724.78. This increase to the mean reward comes with the price of a higher standard deviation, which also increased by a factor of 1.25. There is an increase in the mean of the transfemoral model as well, at a double amount of that of the PPO observation. This increase in mean reward is once again accompanied by a larger standard deviation, at 1.5 times the original. The general observation is that within the same number of episodes for both algorithms, when using the same model, the adapted PPG version clearly finds an optimal policy faster than before.

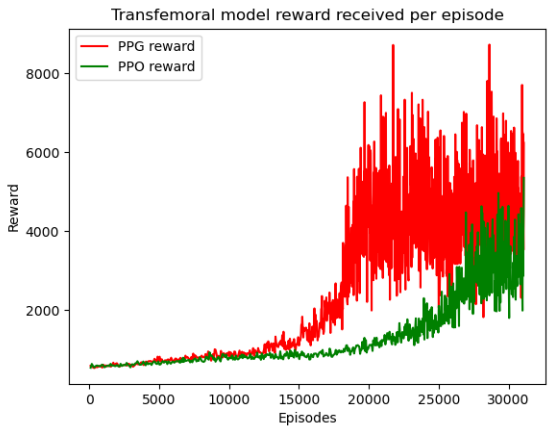


Figure 4.2: Reward received by the PPG algorithm (red), against the standard PPO performance (green) on the transfemoral model. The reward is plotted on the y-axis and total episodes on the x-axis.

4.2 Kinematic Data

Kinematic data was extracted from simulation data for the knee and ankle angles, as well as the muscle activations of the bicep femoris, soleus, vasti and tibialis anterior muscles, along with their fiber forces. Z-scores are calculated to find the closeness of the simulation and experimental data, mean muscle activations are compared between models and the usage of individual legs in each case. Lastly, fiber force graphs are observed for the formerly mentioned muscles, mapping the force behaviour over approximately three gait cycles.

Observing the graphs 4.4 and 4.3, the patterns roughly match the shape and the calculated z-scores show that the simulation data of the healthy model compared to the experimental data are not far apart. Table 4.2 shows the knee and ankle calculated z-scores, where the average from both models does not exceed 1.18 standard deviations distance from the experimental data. The transfemoral data was observed to be very close to the experimental set with low z-scores.

Figure 4.3: Knee angles shown for the healthy (left) and transfemoral (right) models, plotted against the experimental data

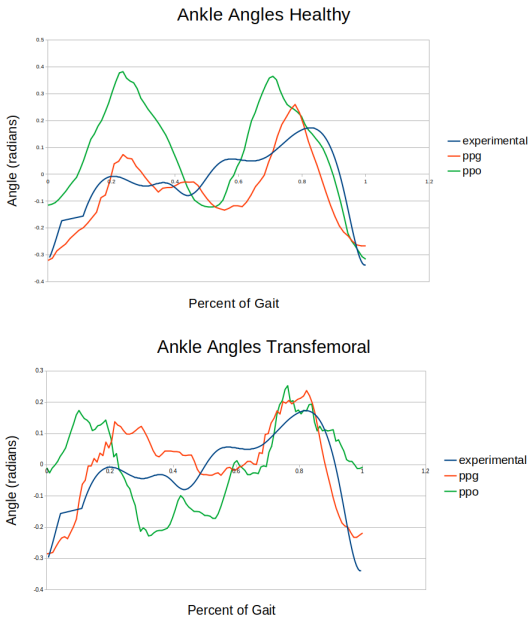


Figure 4.4: Ankle angles shown for the healthy (left) and transfemoral (right) models, plotted against the experimental data

	PPG Angle z-scores	
	Knee	Ankle
Healthy	-1.18	0.25
Transfemoral	-1.81	0.36
	PPO Angle z-scores	
	Knee	Ankle
Healthy	-1.42	0.56
Transfemoral	-2.76	0.66

Table 4.2: Z-scores calculated for the knee and ankle angle between healthy and transfemoral simulation and experimental data for PPO and PPG algorithms

Comparing the shapes from figures 4.4 and 4.3, the two algorithms roughly match the shape of the experimental knee and ankle, with the corresponding z-scores for the healthy knee being slightly below the experimental knee measurements. The ankle measurements show a more faithful shape to the experimental for the PPG algorithm, opposed to the more exaggerated PPO angles. More accurate results are observed for the transfemoral model for the PPG algorithm, both in shape and corresponding z-scores.

The muscle activation data, along with the fiber forces shown in table 4.3, display the average kinematic performance in the two models for both legs. The healthy models generally has smaller total difference values for both measures between the two legs. This is not the case for the transfemoral model, which has a slightly higher difference in the activation means, but a much larger difference calculated for the fiber forces between the left and right legs. Noteworthy differences were found between the Vasti and Soleus muscles, with the right leg exerting a much larger force than the left.

The fiber forces seen in figure 4.5, show rather erratic behaviour for the bicep femoris muscles in both models. Soleus fiber forces shown in figure 4.6, which show relatively more stable patterns for the healthy, when compared to the transfemoral model. The opposite could be said for figure 4.7, where the healthy produced rougher forces. The tibialis anterior muscles also produce similar results with no particular model standing out compared to the other in figure 4.8.

Overall, the observed results show an overall faster learning rate than what PPO showed in previous iterations, leading to quicker results in with less experience episodes required. Moreover, after observing the resulting gait patterns, the resulting PPG gait has more consistent patterns, closer to the experimental data as well. Differences also include the increase use of the knee joint for the right leg of the model. Earlier versions struggled with getting a more consistent knee movement going, but PPG has yielded more satisfactory results, displaying more realistic use of the joint.

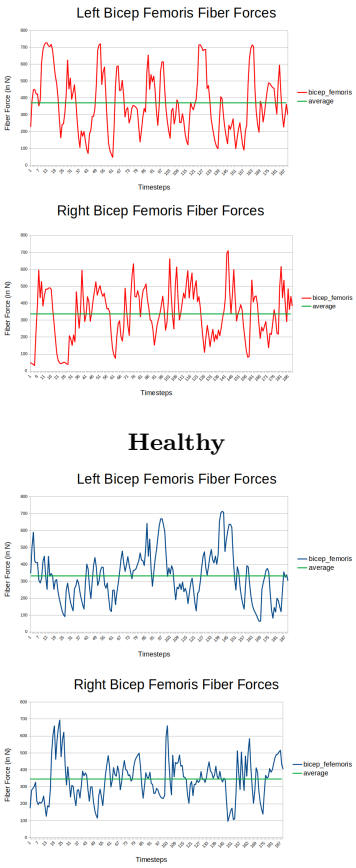


Figure 4.5: Bicep Femoris Fiber Forces for the healthy (red) and transfemoral (blue) models

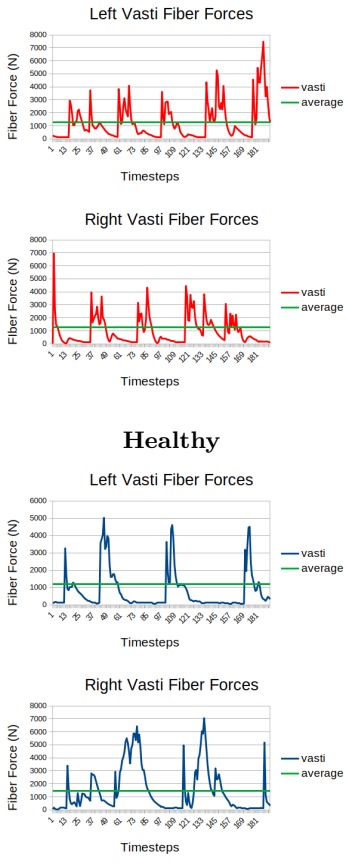
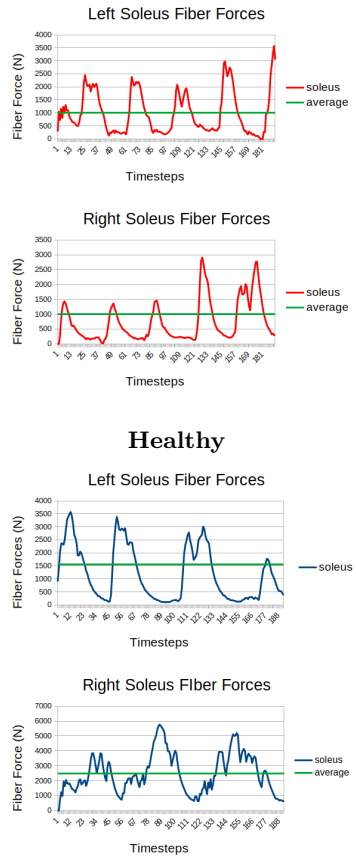
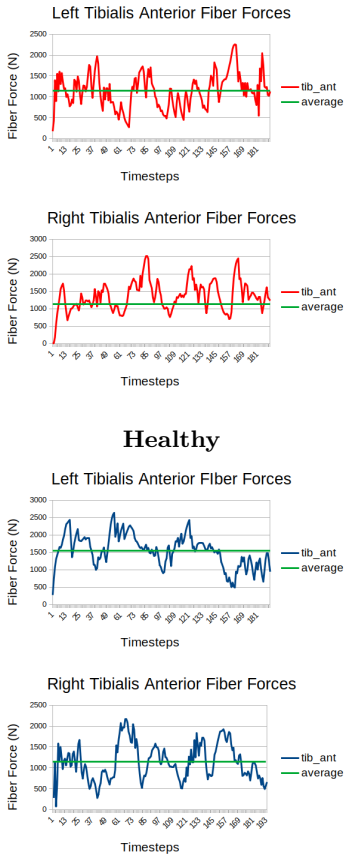


Figure 4.7: Vasti Fiber Forces for the healthy (red) and transfemoral (blue) models





			Healthy	
	Muscle	Left	Right	Delta
Activation mean	Bifemsh	0.68	0.63	0.051
	Vasti	0.21	0.19	0.022
	Soleus	0.23	0.21	0.036
	Tibialis Anterior	0.63	0.73	0.10
Total Difference	in means	0.21		
			Healthy	
	Muscle	Left	Right	Delta
Fiber Force mean	Bifemsh	370.41	330.43	39.97
	Vasti	1326.39	1112.01	214.39
	Soleus	1047.047	822.93	224.10
	Tibialis Anterior	1037.25	1242.90	205.65
Total Difference	in means	684.13		
			Transfemoral	
	Muscle	Left	Right	Delta
Activation mean	Bifemsh	0.64	0.67	0.03
	Vasti	0.07	0.19	0.12
	Soleus	0.33	0.51	0.18
	Tibialis Anterior	0.75	0.69	0.05
Total Difference	in means	0.38		
			Transfemoral	
	Muscle	Left	Right	Delta
Fiber Force mean	Bifemsh	346.20	341.07	5.12
	Vasti	384.47	1107.08	722.61
	Soleus	1194.97	2466.54	1271.57
	Tibialis Anterior	1368.31	1202.18	166.12
Total Difference	in means	2165.44		

Table 4.3: Results comparing the muscle usage between healthy and transfemoral models

4.3 Limitations and Future Work

Considering the use of the particular DRL approach in generating gait patterns in an efficient manner in both healthy and transfemoral models, the algorithm proposed in this paper is successful in its goal.

Despite this success however, the results extracted are lacking in realism. Applying the exact fiber forces into a real-life prosthetic piece would not be wise due to the erratic force outputs. It is also uncertain how the learned policy would translate in any physical system, given that the simulation environment is using the given muscle activations as an input that does not necessarily translate into a mechanical components.

Future work should focus on ensuring that such systems are tested and the performance of the algorithm is equivalent. Additionally, the algorithm utilises the auxiliary phase to better optimise the value of the system. This said reward should be optimised itself in its calculation in order to make the most out of the refining features of the auxiliary updates. One way to encourage less erratic patterns is by incorporating equations that describe muscle behaviour during the gait in each phase it is in and penalise deviations.

5 Conclusions

OpenSim environments combined with the DRL methods imposed by the actor-critic approaches, lead to promising results in learning to develop a steady gait for transfemoral prosthesis. Given the background of transfemoral prosthesis is small and limited, this paper contributes to the overall expansion of literature in the field. Additionally, the new PPG algorithm has not been tested in such environments in the past and was assumed to offer small improvements in performance in non-mujoco tasks.

This research though shows that taking advantage of the network structures, as well as the suggested auxiliary equations to fit the problem at hand, can produce promising results and develop walking patterns in OpenSim environments. The joint angles yielded by the simulations are close to the experimental data set used. The transfemoral model showed that the Vasti and Soleus muscles would need higher activations to compensate for the lack of the healthy muscles to generate a complete gait.

This research can be used as a basis to develop faster and more efficient gait patterns, when using one of the latest developed DRL algorithms. Training a model to reach good results and equivalent walking patterns to PPO can be done faster, with the potential to develop more advanced and sophisticated equations to produce less erratic muscle activations. Such a feat would allow the algorithm to translate the resulting behaviour directly into a real mechanical system

References

Teymur Azayev and Karel Zimmerman. Blind hexapod locomotion in complex terrain with gait adaptation using deep reinforcement learning and classification. *Journal of Intelligent & Robotic Systems*, 99:659–671, 09 2020.

Karl Cobbe, Jacob Hilton, Oleg Klimov, and John Schulman. Phasic policy gradient, 2020.

Humberto Sossa Cristyan R. Gil, Hiram Calvo. Learning an efficient gait cycle of a biped robot based on reinforcement learning and artificial neural networks. *MDPI AG*, 9:24, 02 2019. doi: https://doi.org/10.3390/app9030502.

Leanne de Vree and Raffaella Carloni. *Deep Reinforcement Learning for Physics-based Musculoskeletal Simulations of Healthy Subjects and Transfemoral Prostheses Users during Normal Walking*. PhD thesis, University Groningen, 2020.

Yan Duan et al. Benchmarking deep reinforcement learning for continuous control, 2016.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. Learning to walk via deep reinforcement learning, 2019.

Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin Riedmiller, and David Silver. Emergence of locomotion behaviours in rich environments, 2017.

V. R. Konda and J. N. Tsitsiklis. Actor-critic algorithms. *Advances in Neural Information Processing Systems*, 12:1008–1014, 2000. URL `https://ci.nii.ac.jp/naid/10007790456/en/`.

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, 2019.

Mayur Mudigonda, Pulkit Agrawal, Michael Deweese, and Jitendra Malik. Investigating deep reinforcement learning for grasping objects with an anthropomorphic hand, 2018.

Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning, 2017.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015. URL `http://arxiv.org/abs/1502.05477`.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

Michael H. Schwartz, Adam Rozumalski, and Joyce P. Trost. The effect of walking speed on the gait of typically developing children. *Journal of Biomechanics*, 41(8):16391650, 2008. doi: 10.1016/j.jbiomech.2008.03.015. URL `https://doi.org/10.1016/j.jbiomech.2008.03.015`.

Scott and Anderson et al. Opensim: Open-source software to create and analyze dynamic simulations of movement. *Biomedical Engineering, IEEE Transactions on*, 54:1940 – 1950, 12 2007. doi: 10.1109/TBME.2007.901024.

Mohit Sewak. *Actor-Critic Models and the A3C: The Asynchronous Advantage Actor-Critic Model*, pages 141–152. Springer, 06 2019a.

Mohit Sewak. *Temporal Difference Learning, SARSA, and Q-Learning*, pages 51–63. Springer Singapore, Singapore, 2019b.

Marjolein van der Krogt, Scott Delp, and Michael Schwartz. How robust is human gait to muscle weakness? *Gait & posture*, 36:113–9, 02 2012.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Reinforcement Learning*, page 532, 1992.

ukasz Kidziski, Sharada P. Mohanty, Carmichael Ong, Jennifer L. Hicks, Sean F. Carroll, Sergey Levine, Marcel Salath, and Scott L. Delp. Learning to run challenge: Synthesizing physiologically accurate motion using deep reinforcement learning, 2018.

ukasz Kidziski et al. Artificial intelligence for prosthetics - challenge solutions, 2019.