Understanding motion kinematics using prosthetic devices after lower limb amputation through AI-engineered model

Catalina Botia
Universidad de Los Andes
201729189

Isabella Ramos
Universidad de Los Andes
201730255

Daniela Tamayo
Universidad de Los Andes
201730499

Abstract

Limb loss is very common worldwide. Lower limb amputations are distressful experiences that require adjustment over long periods of time. However, prosthetic devices enable the patients to walk and run simulating regular limb function. Still, the knowledge that we have about how the human gait will change after using a prosthesis is not yet fully understood. In order to understand motion kinematics using a prosthetic device, the NeurIPS 2018 AI for Prosthetics challenge was reproduced using the simulation environment OpenSim. The problem was addressed with Reinforcement Learning (RL) using a Deep Deterministic Policy Gradient (DDPG) algorithm. Further experiments were performed with reward shaping functions and target velocity which invigorates the skeleton to improve its behavior in the environment. Our best model obtained a reward of 43.39.

1. Introduction

Humans have a strictly bipedal locomotion, averaging 6500 daily steps at 1.3 m/s in urban environments [24]. However, lower limb amputations that affect and completely change locomotion are very common every year. For instance, in 2007 there were approximately 1.7 million people with limb loss in the United States [17]. Amputations are caused by many possible reasons; most of them are due to vascular and circulatory disease (70%), trauma (23%), tumor (4%) and congenital conditions (3%) [17]. Additionally, 90% of amputations concern a lower limb. [24]. In Colombia, there are no recent exact statistics, but the Colombian Association of Physical Medicine and Rehabilitation estimated that in 2006 about 200-300 persons for every 100,000 had had amputations [5]. These numbers suggest that this is a common disability and therefore it is crucial to both understand the biomechanical problem and provide solutions for the people who struggle with it daily.

Limb amputations are significantly distressful experiences that require both physical and psychological adjustment over long periods of time. In fact, physical difficulties such as pain or problems with balance and ambulation are very common in patients who have lost a limb, regardless of the cause. However, recent advances in prosthetic devices offer options that enable the patient to walk, run or simulate regular limb function so they can have a normal locomotion [3]. Still, the costs of amputation and rehabilitation in the community are considerable. A significant percentage of these costs includes the provision of the prosthesis and the process of retraining the patients in order to achieve functional mobility in the community [4]. The significant medical challenge that is not yet fully understood and must be taken into consideration is explaining how walking will change after using prosthetics. This problem is specifically challenging for lower-limb amputees, which will be the target group of our investigation.

Artificial intelligence (AI) has been a formidable tool for biomedical research. In fact, the whole healthcare system has been benefited by technological advances built through machine learning. For instance, automated medical diagnosis, image classification for pathogen identification, recognition of ailments and personalised treatment strategies, health data control, management of patient information, among others. Reinforcement Learning (RL), a sub-field of machine learning, has also proven to be particularly useful in biomedical research. The general idea of RL is to endow an agent with skills and capabilities in behavioural decision making by using evaluative feedback and interaction experience with a specific environment. In simpler words, RL aims to find the best possible behavior or path to maximize a reward in a specific situation or environment. Contrary to supervised learning, RL solves sequential decision making problems with sampled, evaluative and delayed feedback simultaneously. Moreover, RL does not seek a supervised reward signal but attempts to maximize the reward as much as possible. Such unique features make RL a distinctive strategy to tackle healthcare and biomedical problems. The lower-limb prosthetics problem is not an exception. [26] [21]

Due to the great need of new strategies to improve and assist rehabilitation with prosthetic devices, we are going to implement the NeurIPS 2018 *AI for Prosthetics* challenge. This challenge consists of developing a controller to enable a human model with a prosthetic leg to walk and run effectively. To achieve this, we will use the simulation environment *OpenSim* where we can use a human musculoskeletal model to obtain normal motion kinematics data and synthesize physically and physiologically accurate motion with a prosthesis [1]. In other words, we will be modeling how walking and running changes after getting a prosthesis. To approach this problem we will use RL in order to find the balance that maximizes the human model performance while walking. Additionally, we will explore the importance of some parameters of the human model and their effects in motion kinematics, for example the effects of the human pose.

2. Related Work

The state-of-the-art on reinforcement learning with musculoskeletal models can be found in the papers of the previous NeurIPS 2017 challenge *Learning to run*. The paper *Learning to Run Challenge Solutions: Adapting Reinforcement Learning Methods for Neuromusculoskeletal Environments* (Kidziński et al.) [8] explores eight possible solutions, all of which use deep reinforcement learning approaches. The method introduced by the authors for the challenge is Actor-Critic Ensemble (ACE), which aims to improve the performance of Deep Deterministic Policy Gradient (DDPG) algorithm. This method won second place in NeurIPS 2017 challenge *Learning to run* [8]. The DDPG algorithm was proposed in 2015 by Lillicrap et al. The authors presented a method based on the general ideas of Deep Q-Learning by applying them to the continuous actor domain. The algorithm is an actor-critic, model-free method based on the deterministic policy gradient that can operate over continuous action spaces, it is to say, the algorithm is off-policy. [13]

For the *Learning to run* challenge, Kidziński et al. outperformed the DDPG algorithm by adding the ACE. Their idea was to use a critic ensemble at interference time in order to select the best action from proposals of multiple actors running in parallel. This solves one of the problems of DDPG which is dooming actions. In terms of the problem, a "dooming action" occurs when the skeleton enters an unstable state with limbs swinging and falling down. This is a state from which the skeleton cannot possibly recover from. By adding the ACE, the network can inspect the actions at interference time so the critic can anticipate low scores and recognize doom actions. [8]

The biomechanical analysis of this problem and its representation on the *OpenSim* environment for the *Learning to Run* challenge (which could be modified for the *AI for Prosthetics* challenge) is present in a more recent publication by Kidzińsky et al [7]. They explain how a muscoloskeletal simulation environment must be used to answer to two fundamental questions: how to model human systems through the generation of their corresponding equations of motion and how to synthesize motions by integrating these equations over time. Their model was similar to that of Ong et al. in 2017 [15] and included 7 bodies as subdivisions of the real human body: a set of upper leg, lower leg and foot for each of the two legs and a single body representing the pelvis, torso and head. The model also included 18 musculotendon actuators (9 on each leg) to represent the muscle groups in charge of walking. The force of these actuators was dependent on length, velocity and activation level of the muscles, which is what the investigators set as a starting point for developing the equations of motion. [7] They used TRPO (Trust Region Policy Optimization, first presented by Schulman et al. [20]) and DDPG [13] as a baseline due to their previous results and experience on the 2015 publication. [7][8]

Additionally, there has also been a recently advancing research in the use of recurrent neural networks (RNNs) in reinforcement learning (RL). One of the challenges for RL is where part of the state of the environment is hidden, so the agent not only needs to learn the mapping from the environmental states to actions, but also needs to determine in which environmental state it is. These tasks are known as non-Markovian or Partially Observable Markov Decision Processes. Long Short-Term Memory Networks (LSTMs) are a type of RNN that solve the problems in recurrent learning algorithms when learning timeseries with long-term dependencies. They solve this problem by enforcing constant error flow in a number of specialized units, called Constant Error Carrousels (CECs) which have linear activation functions that do not decay over time. They also use input gates (specialized multiplicative units) to access the CECs and prevent them from filling up with useless information. RNNs can be used in RL in many ways; one way is to let the RNN learn a model of the environment which learns to predict observations, rewards and to infer the environmental state at each point. This could be done with an LSTM because they allow the predictions to depend on information from long ago. It is also very useful in the non-Markovian tasks because the agent can learn the mapping from the predicted environmental states in a Markovian way with Q-learning. Another application is to use the RNN to directly approximate the value function of a RL algorithm. This model-free approach approximates the state of the environment by the current observation (input of network) and also with the recurrent activations that represent the agent's history. [2] [19]

3. Approach

For starters, the algorithm generates the Open-Sim Prosthetic environment which is set to 3D, prosthetic limb and minimum difficulty. The action space of this particular setting is a vector of length 19, which refers to the number of muscles that could be excited at some point during the simulation. As stated before, one of the promising approaches for developing RL algorithms is Deep Deterministic Policy Gradient (DDPG) [12], which even has great results for bipedal walking problems [11]. Because of this, our baseline method consists of a DDPG agent which incorporates penalties and rewards that are specific for the context of this problem.

The agent was initially built up by a fully connected state action input Q-function. Along with it, the agent used a fully connected deterministic policy, which was adapted from the open-source code by Mohammedalamen [14], which uses the Chainer library [22]. Once the agent was set up, it started interacting with the environment by running for a determined number of episodes. In our case, this number was set to 2000 in an effort to provide the agent with enough room for experimentation and possible improvements. During each episode the agent chose an action (muscle excitation) depending on the observation of the current state (the state of muscles, joints and velocity) and the information it has learned previously. By performing this action, the reward was calculated according to a reward equation based on the context of the problem. The process was repeated iteratively and the best model was saved every time a new higher reward was obtained.

Furthermore, this baseline included an explorer and a reward shaping function. The first one encourages the skeleton to excite different muscles and learn about them, even if it has knowledge that other muscle excitations might lead to nearly guaranteed positive results. Initially two explorers were tried, an ϵ- Greedy and an Additive Ornstein-Uhlenbeck process, but the ϵ- Greedy method was chosen for this baseline. Secondly, the reward shaping function is essential in the baseline since it improves the learning process by penalizing wrong actions [13]. One of them is falling, which reduces the reward in 10 points.

As in most RL problems, the proposed solutions needed a lot of run time and memory, which was not ideal for the initial baseline because it was not compatible with GPUs (because of the Chainer [22] implementation). Since this issue restricted the process of obtaining results, the Chainer-DDPG model was replaced by an adaptation from Yoon's DDPG implementation for pytorch [25], which was crucial in order to use the server, run multiple experiments at the same time and implement further penalties and reward shaping functions. Therefore, additional penalties were calculated with body velocity, pelvis position and body position. [10]

4. Experiments

4.1. Dataset

Since we are implementing NeurIPS 2018 challenge and using the *OpenSim ProstheticsEnv* simulation environment, we decided to use the same evaluation specified in the contest. A sample of the environment and its parameters can be seen on Figure 1. The general idea is to build a function that takes the dictionary describing the current state observation and returns a 19-dimensional vector of the muscle excitation actions. The objective is to follow a velocity vector and try to maximize the reward, defined by R in the following equation:

$$R = 9 * s - p * p$$

This equation can be interpreted as the request to run at a constant speed of 3 meters per second. The absolute difference between horizontal velocity and 3 $\frac{m}{s}$ is represented by p while s stands for the number of steps reached before one of the stop criteria. There are two stop criteria that end the trial. First, if the pelvis of the model falls below 0.6 meters, and second, if 1000 iterations are reached (which is to say, 10 seconds in the virtual environment). In the *AI for Prosthetics* challenge all competitors were ranked according to their maximum reward.

# of muscles	19
# degrees of freedom	14
reward	negative distance from requested velocity

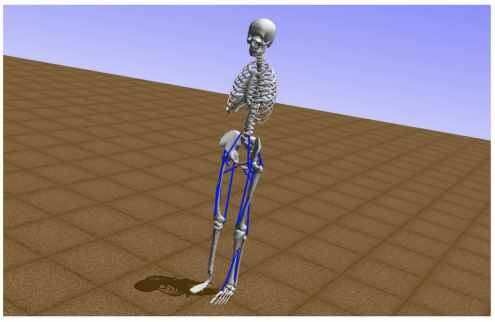

Figure 1. OpenSim's Prosthetics Environment showing a sample of a person with a lower limb prosthetic device along with the 19 main muscles that will be studied. [8]

4.2. Baseline Results

For the baseline with the chainer library presented previously the agent was trained for a total of 2000 episodes and the reward was evaluated after each episode. For the baseline with pytorch, the agent was trained for a total of 10000 episodes. Table 1 shows the results and the parameters used for both baselines. Figures 2 A and D show this behavior and the changes in reward over time.

ID	Episodes	Batch size	Step size	Target vel.	Max memory	Max Test R	Max R	Average R	Time (h)
1	2000	-	500	None	-	-314.62	77.27	-62.54	16
2	10000	1500	1000	None	50000	-462.15	59.82	-488.82	62
3	2000	-	500	None	-	-426.71	101.53	-401.2	16
4	8000	-	500	None	-	-453.23	693.65	52.51	65.5
5	3000	-	500	None	-	-513.23	362.24	-389.37	24.5
6	9500	600	600	None	50000	-26.17	54.75	-524.35	59
7	8000	600	600	None	50000	18.82	63.89	-287.95	61
8	10000	400	500	3	50000	-267.09	30.78	-557.02	57
9	8500	300	500	6	50000	-87.53	36.50	-383.25	52
10	7500	400	500	3	50000	-403.45	-187.39	-742.22	35
11	8500	300	1000	None	50000	-44.65	23.48	-58.04	46
12	6841	50	1000	None	50000	**36.76**	60.60	-115.41	33.2
13	12715	200	1000	None	50000	-202.63	77.39	-467.16	61.6
14	6498	200	8000	None	50000	-412.78	73.02	-469.20	35.3
15	10252	200	1000	None	500000	-426.65	78.49	-459.15	70.6
16	5200	200	8000	None	500000	-470.56	75.23	-470.41	31.7
17	16207	500	1000	None	50000	4.20	68.36	-534.30	93.58
18	16207	50	1000	None	50000	-502.27	55.92	-522.74	71
19	4341	50	1000	None	50000	-202.63	-635.86	-1740.88	26.25
20	11919	300	1000	None	50000	-412.78	-415.83	-1686.20	60.77
21	8072	300	1000	None	50000	**43.39**	-190.18	-813.58	57.4

Table 1. **Experimental Results:** The table is divided in blocks, each referring to a set of experiments individually referenced with an ID number. **1-2.** Baseline algorithms using chainer (1) and pytorch implementation (2). **3-5.** Experiments using chainer algorithm with ϵ-greedy exploration (3), reward penalties (4) and both approaches (5). **6-7** Experiments with reward shaping A (6) and B (7). **8-10** Experiments implementing a velocity loss according to a target (8-9) and a velocity penalty (10). **11** Environment wrapper and reward shaping B. **12** Experiment with the pytorch baseline changing batch size. **13-16** Experiments with the environment step and penalties from [23]. **17-18** Experiments with reward B changing batch size. **19-21** Penalties for torso falling backwards (19), for standing still (20) and both (21), all using reward B. **Note:** The two IDs in bold show the best result in terms of higher test reward and better performance after visualization. R stands for reward.

4.3. Experiments

All experimental results are categorized in Table 1. These correspond to multiple modifications which will be explained subsequently. Some of these experiments are also presented graphically in Figure 2, which shows standard and average training rewards for each episode.

4.3.1 Chainer Experiments

Multiple experiments were performed using the *chainerrl* library. First, for the chainer library two experiments were done: with and without explorer and reward shaping function. Note that different strategies result in peaks on the graphs, representing episodes where the individual reward was outstanding. We can also see that the Figure 2 C has more peaks in the reward value which means that the agent is probably learning better with the reward penalties that were implemented.

The figures also show that the maximum reward achieved by penalizing is way better compared to the initial method. This is a very important aspect that some competitors mentioned because they showed that using different penalties to shape the reward actually allowed the agent to learn the actions that it was supposed to take in order to get better results [9]. The previous point makes sense because if the agent falls, the pelvis is very low, or the body position is not adequate then the reward will be affected significantly and therefore the agent will learn not to repeat those actions that have negative results.

As for ϵ-greedy exploration, the contrast is clear in Figures 2 B and C. What can be seen is that with extra exploration capacities, the model diverges more than with the Ornstein-Uhlenbeck Process (OU). The results with ϵ-greedy exploration show a bigger fluctuation in rewards after epoch 1000, taking more time for the agent to reach a good reward. On the other hand, the OU exploration with the reward penalties converges faster. This is why this explorer was used for the next experiments.

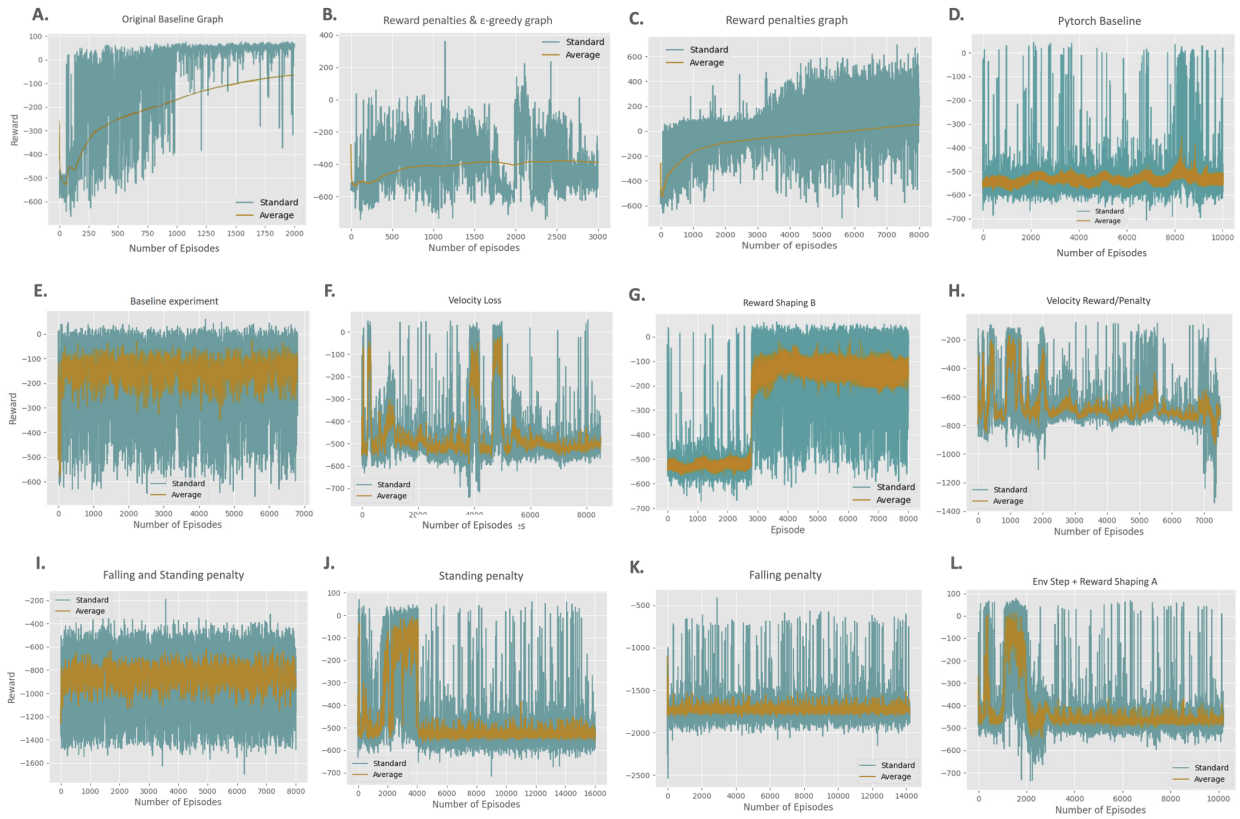

Figure 2. **A.** Baseline chainer algorithm with no modifications for ϵ-greedy exploration or reward penalties. **B.** Baseline chainer DDPG algorithm with ϵ-greedy exploration and reward penalty helper functions. **C.** Baseline chainer DDPG algorithm with Ornstein-Uhlenbeck exploration and reward penalty helper functions for 8000 episodes. **D.** Pytorch baseline. **E.** Training curve for the baseline experiment with 50 batch size. **F.** Training curve for velocity loss experiment. **G.** Training curve for applying the Reward Shaping function B to the baseline. **H.** Training curve for velocity reward/penalty **I.** Implemenation of backwards falling and standing still penalties on pytorch baseline. **J.** Penalty for standing still **K.** Penalty for torso falling backwards. **L.** Training curve for the environment step and reward shaping A. With 200 batch size, 1000 steps and 500000 max memory.

4.3.2 Pytorch experiments

The original baseline using the chainerrl library was very time consuming because it was impossible to use the GPU, specifically, 2000 episodes took 16 hours. Also, we could not experiment with the actor and critic networks due to the fact that the library already had them implemented. Because of these reasons, we decided to write another code implementing the DDPG algorithm using pytorch. We expected this new algorithm to be faster and to let us experiment and understand the networks. In this new baseline we used the experience replay buffer, the actor (policy) & critic (value) networks, the target networks and the Ornstein-Uhlenbeck Process exploration. In continuous action spaces the exploration is done by adding noise to the action itself. All of the hyperparameters used were from the original DDPG paper "Continuous Control With Deep Reinforcement Learning" [13] and experiments with some of them were performed.

DDPG experiments focused mostly on following a target velocity, shaping the reward and changing hyperparameters such as number of steps, batch sizes, etc. The experiments tried were:

- **Reward shaping A:** The reward obtained in each episode was modified by adding extra penalties for falling and not following the expected velocity [23]. This experiment corresponds to 6 in table 1.
- **Reward shaping B:** The reward obtained in each episode was modified by adding penalties for crossing legs, deviation from walking forward, and taking steps to the sides. Furthermore, a bonus for bending knees was added. Coefficients for this penalties and bonuses were set to 0.1. [10]. These experiments correspond to 7, 17 and 18 in table 1.

- **Velocity Loss** In addition to the critic loss and actor loss, a velocity loss was calculated comparing the XYZ velocity relevant to the pelvis center point against a target velocity which was set to 3. The loss was calculated using Mean Squared Error (MSE). These experiments correspond to 8-9 in Table 1.

- **Velocity penalty/reward** The target velocity was set to 3 and the reward was shaped according this target velocity. If the calculated velocity exceeded the target velocity the reward increased, otherwise it was penalized.

- **Environment Wrapper + Reward Shaping B** The opensim environment wrapper was adapted from [10] and **Reward Shaping B** was used in this experiment. This experiment corresponds to 10 in Table 1.

- **Environment step + Reward shaping A** Some changes were made to the OpenSim environment regarding the observation and some reward penalties were implemented adapted from [23]. A longer time limit was used to reduce likelihood of diving strategy, the reward was penalized proportional to the velocity of the pelvis in z and x, and also if the body position in the pelvis fell. The step function was changed to include the reward with the penalties. One of this experiments can be seen in figure 2 L. These experiments correspond to 13-16 in Table 1.

- **Penalty for torso falling backwards** Considering prior experimentation it was clear that a recurring issue was falling backwards. Hence, an extra penalty was implemented based on the velocity of the torso, which was penalized when negative (eg. falling) with a coefficient of 1. This experiment corresponds to 19 in Table 1.

- **Penalty for standing still** Another recurrent problem was not taking a forward leap, which in terms of the skeleton meant a minimum center of mass velocity. Therefore, the reward was penalized with different coefficients during experimentation. This experiments correspond to 20 in Table 1.

4.4. Evaluation and visualization

The results were also evaluated qualitatively since it is important to understand how the skeleton is moving. The animations of baseline and different experiments are shown below.

4.4.1 Comparison with other methods

Though multiple efforts were made in order for the skeleton to walk, the results were not comparable to that of some

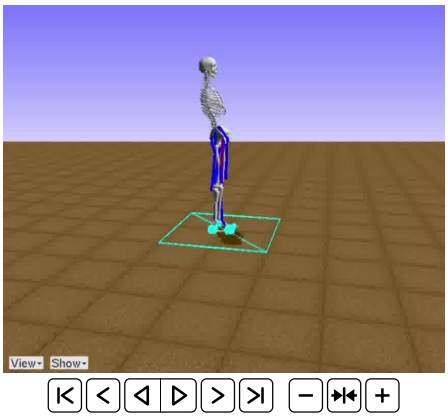

Figure 3. Reward Shaping B animation run with 8000 episodes.

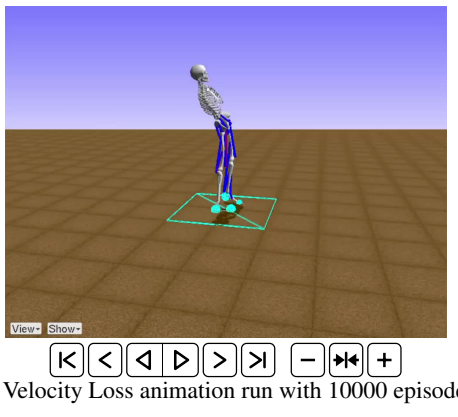

Figure 4. Velocity Loss animation run with 10000 episodes.

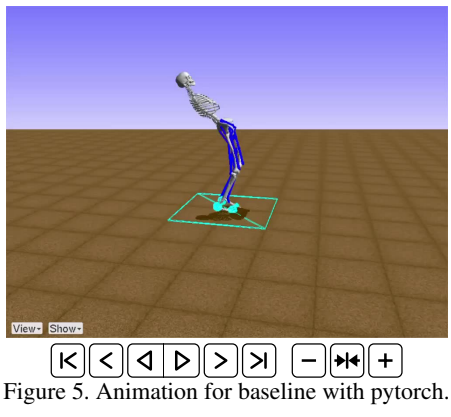

Figure 5. Animation for baseline with pytorch.

of the competitors in the NeurIPS challenge. This difference can be attributed mainly to time and computational limitations, which will be discussed afterwards. Investigators working with this environment and challenge managed to get a cumulative reward of up to 9980.46 [1], which is far from our obtained reward. However, the winners used 500 CPUs and highlighted the importance of training for at least 100 hours in order to get comparable results [16]. In short, the lack of time was the greatest difference in our implementation and the cumulative reward during testing was low because the agent failed to walk consistently and in a

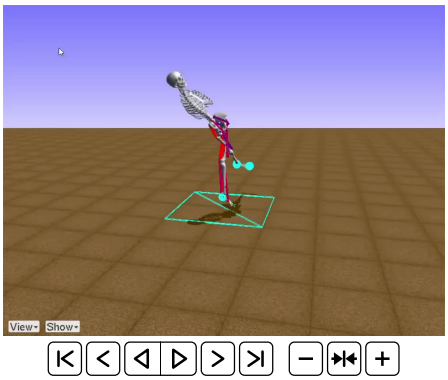

Figure 6. Animation of Environment Wrapper + Reward Shaping A run with 5200 episodes.

human-like manner. It is worth highlighting that even the third place solution for this challenge managed to get remarkable results in terms of reward but the movement that can be seen in their demo is far from resembling that of a human.

5. Discussion

5.1. Results Discussion

In the first baseline experiment reducing the mini batch size (Figure 2 E.), the performance was significantly better. The mini batch size is the number of samples that the agent collects from the replay buffer to continuously update the model parameters. We found that in this application a smaller batch size had a better average reward.

There are several strategies to increase performance in RL, one of the most common and effective is creating a reward shaping function. The reward signal defines the goal of the problem and the idea is to maximize it. As stated before, the reward is calculated with the number of steps reached before one of the stop criteria and the absolute difference between horizontal velocity and 3 $\frac{m}{s}$. Since the reward signal determines the desirability of each state, shaping the reward by adding penalties or bonuses could help the agent understand which behavior is good in a deeper matter. Therefore, two types of reward shaping function were proposed as explained in the experiments section. The reward shaping A function did not performe as expected since the maximum reward obtained in test was -26.17. Probably, the reward function focused too much in penalizing for falling and not following an expected velocity without a proper exploration that will allow the agent to discover new muscle and joint activation combinations and getting better rewards. The reward shaping B function is more specific which invigorates the agent to behave better in its environment. In fact, the function takes into account crossing legs, deviation from walking forward, taking steps to the sides and bending knees. A more robust reward shaping function contributes to better results as seen in Table 1; the test maximum reward was 18.82 and visualization can be seen in the animation Figure 3. For this experiment is very important to notice that reward was considerably low until 3000 episodes, moment in which it started to increase significantly as seen in Figure 2.G. This is an example of time and computational requirement for this problem.

Reward shaping is an important strategy but still not enough for an adequate performance. A second approach to tackle this problem is following a target velocity. Particularly, in the *OpenSim ProstheticsEnv* simulation environment it is important that the agent keeps up with a specific velocity. Moreover, learning is easier while trying to walk faster [16]. Initially, the target velocity was incorporated to the baseline with two different methods explained in the experiments section: Velocity Penalty/Reward and Velocity Loss. On one hand, the first method got poor results as seen in Table 1; mostly this is due to the fact that the penalty for not reaching the desired velocity was too severe and would not allow the agent to learn properly. Actually, the agent did not get a bonus for reaching the target velocity as expected. On the other hand, calculating the velocity loss and adding it to the network seemed like a good approach but performance did not improve much. As seen in Table 1, the best test reward obtained with this approach was -87.53. As explained before, using the minor batch size got better results. This was the overall best result in terms of target velocity approach. According to this, is useful to calculate the velocity loss with MSE and adding it to the network so it will decrease each episode rather than penalizing the agent for not reaching the target velocity. This explains why velocity loss got better results than velocity reward/penalty.

Using an environment wrapper is also an interesting strategy to endow the observations with more information thus, on each step the state observation is more robust. For instance, the experiment Environment Wrapper + Reward Shaping B added to the observations a right and left knee flexion state which is helpful for calculating the bending knee bonus. These experiments got good results but using Reward Shaping B without the environment wrapper outperformed this experiment. Nevertheless, the experiments performed with the Environment step + Reward shaping A from [23] were not successful. All of them had a very low reward. Is possible that the step environment wrapper did not contribute to the observation state and gave enough information to reward shaping function A. On the contrary, reward shaping B articulated correctly with the environment wrapper as exemplified with the bending knees bonus.

Additionally, the effect of other penalizations was evident (as can be seen in Table 1), and it generally changed the maximum rewards and performance significantly. This was especially clear in experiments that penalized falling backwards and standing still (see Figure 2I-K). These results

were also shifted with the penalizing coefficients, which ranged from 5 to 20. As can be seen, the best result (Table 1.21, Fig 2I) was obtained by implementing both of these penalties. Though the maximum rewards stay negative, this was probably due to heavy penalization resulting from high coefficients, and this does not directly translate to bad results within the visualization. The joint effort of both of these penalizations resulted in the best visualization and had higher rewards over time compared to Figure 2J,K.

5.2. Biomechanical analysis

While evaluating and visualizing the agent in the environment, we found out that it tended to perform one action most of the time: all of the muscles were activated at the same time and the legs were crossed ending up with the prosthesis passing through the other leg and falling (Figure 6). This action happened most of the time with the experiments that had a very low reward. Additionally, the agent performed some other actions that achieved a better reward: for example, trying to take a step forward and eventually falling because the upper body was not straight or excessively bending the knees and taking a step but sliding in the ground. In order to understand the biomechanics of the actions , it is necessary to understand the human gait.

Human gait consist of using the strategy of double pendulum. While moving forward, the leg leaves the ground and moves forward from the hip, which corresponds to the first pendulum and then comes the second inverted pendulum. A gait cycle begins with the initial contact between the heel and the ground of one leg and goes up to the subsequent contact between the heel the ground of the same leg. The step is the basic unit that composes the human gait and it is divided in two phases: the support phase and the swing phase. In the first one, the support phase, the foot touches the ground and takes approximately 60% of the overall cycle duration. The support limb transfers the pelvis and upper body from the back to the front so that the limb intercepts a body fall and protects the centre of gravity. It is followed by the activation of the quadriceps and dorsal flexor muscles. Finally the push-off begins with the plantar flexion of the ankle, letting the heel roll away with the help of the retromalleolar muscles. Next, the swing phase begins after the push-off, taking 40% of the gait cycle. First, knee flexors are activated and adduct the lower leg onto the ground which makes the swing disturb and causes the swing of the entire lower limb forward. The movement of the thigh and quadriceps help the lower leg transfer beyond the vertical axis to extension. [18]

In individuals with both lower limbs, the weight is symmetrically distributed in a 50:50 ratio, which facilitates the balance. Therefore, the energy consumption is not increased and there are no unnecessary movements to compensate the gait. In the other hand, for individuals with a lower limb amputation the load is moved to the healthy limb side because the centre of gravity also moves laterally. Stabilization of the prosthesis depends on the residual muscles that remain on the residual limb. This is why the purpose of a prosthesis is to ensure stability and comfort to minimize the energy cost and gait asymmetry. [18]

During the gait with a prosthesis, the transfer of load and the first contact of the foot and the ground are very important [18]. This is why it was the most difficult part for the agent to learn in our simulations; it was difficult and time consuming for the agent to learn the transfer of load with a prosthesis. Also, if the prosthesis is not properly aligned and used while walking some deviations develop which causes overloading of certain groups of muscles [18]. This is probably why the most common action mentioned before (activating all of the muscles) was not successful; basically because the agent did not learn well enough to walk and align the prosthesis, therefore it increased the energy cost during motion and fell. In conclusion, the agent had problems for correctly learning the support phase, therefore the swing phase was impossible. Still, in the best models it learned to activate less muscles and try to transfer the load, so it was learning almost 60% of the overall cycle duration.

5.3. Limitations

The major limitation encountered in handling this RL problem, which was already introduced in the discussion section, is the computational and time requirement. In particular, in terms of CPU, the computational requirement was an issue due to the replay memory that continuously grows until a max memory. As discussed before, better results were obtained with a smaller batch size but that requires more time which turned out to be the biggest limitation. According to [16] and [23], at least a 100 hours are necessary for descent results, but because of CPU limitations experiments were run for an average of 50 hours and a maximum of 93.58 hours could be reached. Moreover, the time requirement made it difficult for the team to analyse and test results in order to come up with new strategies. Also, due to computational requirements, a full hyperparameter search was not feasible.

In terms of the baseline model, the DDPG method implemented did not had multiple actor and critic pairs which allows the training to be more effective [25]. Furthermore, in several experiments the environment wrapper did not matched perfectly the reward shaping function which limited the training. Another challenge in this area of AI is that reproducing results for state-of-the-art RL methods is seldom straightforward. [6]

6. Credits

The project was done equitably between all members. The group worked together to build the DDPG baseline and

general strategies to tackle the problem.

References

[1] Neurips 2018: Ai for prosthetics challenge. https://www.crowdai.org/challenges/neurips-2018-ai-for-prosthetics-challenge, 2018. (Accessed on 08/24/2020).

[2] B. Bakker. Reinforcement learning with long short-term memory. *Advances in neural information processing systems*, 14:1475–1482, 2001.

[3] C. Crerand and L. Magee. Amputations and prosthetic devices. In *Encyclopedia of body image and human appearance*, pages 1–7. Elsevier, 2012.

[4] B. Davies and D. Datta. Mobility outcome following unilateral lower limb amputation. *Prosthetics and orthotics international*, 27(3):186–190, 2003.

[5] El Tiempo. Sí hay salida para los amputados - eltiempo.com. https://www.eltiempo.com/archivo/documento/MAM-1992341, April 2006. (Accessed on 11/26/2020).

[6] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*, 2017.

[7] Ł. Kidziński, S. P. Mohanty, C. F. Ong, J. L. Hicks, S. F. Carroll, S. Levine, M. Salathé, and S. L. Delp. Learning to run challenge: Synthesizing physiologically accurate motion using deep reinforcement learning. In *The NIPS'17 Competition: Building Intelligent Systems*, pages 101–120. Springer, 2018.

[8] Ł. Kidziński, S. P. Mohanty, C. F. Ong, Z. Huang, S. Zhou, A. Pechenko, A. Stelmaszczyk, P. Jarosik, M. Pavlov, S. Kolesnikov, et al. Learning to run challenge solutions: Adapting reinforcement learning methods for neuromusculoskeletal environments. In *The NIPS'17 Competition: Building Intelligent Systems*, pages 121–153. Springer, 2018.

[9] Ł. Kidzinski, C. Ong, S. P. Mohanty, J. Hicks, S. Carroll, B. Zhou, H. Zeng, F. Wang, R. Lian, H. Tian, et al. Artificial intelligence for prosthetics: Challenge solutions. *The NeurIPS'18 Competition: From Machine Learning to Intelligent Conversations*, page 69, 2019.

[10] S. Kolesnikov and O. Hrinchuk. Catalyst.rl: A distributed framework for reproducible rl research. *arXiv preprint arXiv:1903.00027*, 2019.

[11] A. Kumar, N. Paul, and S. Omkar. Bipedal walking robot using deep deterministic policy gradient. *arXiv preprint arXiv:1807.05924*, 2018.

[12] S. Li, Y. Wu, X. Cui, H. Dong, F. Fang, and S. Russell. Robust multi-agent reinforcement learning via minimax deep deterministic policy gradient. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4213–4220, 2019.

[13] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[14] M. Mohammedalamen. Reinforcement learning for prosthetics. https://github.com/montaserFath/Reinforcement-Learning-for-Prosthetics, 2019.

[15] C. F. Ong, T. Geijtenbeek, J. L. Hicks, and S. L. Delp. Predictive simulations of human walking produce realistic cost of transport at a range of speeds. In *Proceedings of the 16th International Symposium on Computer Simulation in Biomechanics*, pages 19–20, 2017.

[16] PaddlePaddle. The winning solution for the neurips 2018: Ai for prosthetics challenge. https://github.com/PaddlePaddle/PARL/tree/develop/examples/NeurIPS2018-AI-for-Prosthetics-Challenge, 2018.

[17] M. R. Pitkin. *Biomechanics of lower limb prosthetics*. Springer, 2009.

[18] V. Rajťúková, M. Michalíková, L. Bednarčíková, A. Balogová, and J. Živčák. Biomechanics of lower limb prostheses. *Procedia engineering*, 96:382–391, 2014.

[19] A. M. Schäfer. Reinforcement learning with recurrent neural networks. 2008.

[20] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[21] G. L. Team. Use of reinforcement learning in healthcare. *Great Learning Blog, Power ahead*, 2020.

[22] S. Tokui, R. Okuta, T. Akiba, Y. Niitani, T. Ogawa, S. Saito, S. Suzuki, K. Uenishi, B. Vogel, and H. Yamazaki Vincent. Chainer: A deep learning framework for accelerating the research cycle. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2002–2011, 2019.

[23] R. Wightman. Pytorch reinforcement learning for opensim environments. https://github.com/rwightman/pytorch-opensim-rl, 2018.

[24] M. Windrich, M. Grimmer, O. Christ, S. Rinderknecht, and P. Beckerle. Active lower limb prosthetics: a systematic review of design issues and solutions. *Biomedical engineering online*, 15(3):140, 2016.

[25] C. Yoon. Rlcycle. https://github.com/cyoon1729/RLcycle, 2020.

[26] C. Yu, J. Liu, and S. Nemati. Reinforcement learning in healthcare: A survey. *arXiv preprint arXiv:1908.08796*, 2019.