A REINFORCEMENT LEARNING BASED CONTROLLER TO MINIMIZE FORCES ON THE CRUTCHES OF A LOWER-LIMB EXOSKELETON

by

Aydın Emre Utku

B.S., Mechanical Engineering, Middle East Technical University, 2019

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Systems & Control Engineering

Boğaziçi University

2022

ACKNOWLEDGEMENTS

First of all, I would like to thank my supervisors Assoc. Prof. Evren Samur and Assist. Prof. Sinan Öncü for their guidance that helped me substantially to overcome the difficulties I have faced. Moreover, I want to thank Assoc. Prof. Emre Uğur for supporting me with his ideas that also assisted me quite a lot. I would also like to express my gratitude to Prof. Dr. Duygun Erol Barkana for accepting to be a jury member of my thesis.

I am grateful to Assist. Prof. Barkan Uğurlu, my dear friends Muhammet Hatipoğlu, Suzan Ece Ada and Mustafa Derman for supporting me with their enlightening ideas.

This thesis has been in part supported by TÜBİTAK (The Scientific and Technological Research Council of Türkiye) under the project no 118E922 and the BİDEB 2210-A program.

Finally, I would like to express my gratitude to my parents Oğuz and Nihal who have been always with me throughout my entire life.

ABSTRACT

A REINFORCEMENT LEARNING BASED CONTROLLER TO MINIMIZE FORCES ON THE CRUTCHES OF A LOWER-LIMB EXOSKELETON

The majority of the metabolic energy consumption of a lower-limb exoskeleton user comes from the upper body effort, since the lower body can be considered to be passive. However, the upper body effort of lower limb exoskeleton users is ignored during motion controller development process in the literature. In this thesis study, deep reinforcement learning is used to develop a locomotion controller that minimizes the ground reaction forces (GRF) on crutches. The rationale for minimizing the ground reaction forces is to minimize the upper body effort of the user. A model of the human-exoskeleton system with crutches is created in URDF and XML formats. Reward functions that encourage the forward displacement of the center of mass of the exoskeleton-human system without falling and extreme joint torques are shaped. The state-of-the-art methods, Twin Delayed Deep Deterministic Policy Gradient (TD3) and Proximal Policy Optimization (PPO), are employed with the RaiSim and MuJoCo physics simulators and with different algorithm specific parameters in multiple training trials. The employed networks generate the joint torques based on the joint angle and velocities along with the ground reaction forces on feet and crutch tips. These generated joint torques are directly sent to the exoskeleton model and a new state is observed after implementing the action that the deep RL framework provides. Policies trained by the TD3 and PPO methods on RaiSim are observed to fail to generate proper control commands for a stable and natural looking gait. In general, it is observed that the PPO method generated higher rewards than the TD3 method on RaiSim. After failing to develop a desired policy with RaiSim, MuJoCo is employed as the simulator. Eventually, a policy that can generate a reasonable gait with a desired crutch usage and with 35% minimization in GRFs with respect to the baseline policy is developed.

ÖZET

ALT EKSTREMİTE DIŞ İSKELETLERİNİN BASTONLARINDAKİ KUVVETLERİ MİNİMİZE EDEN PEKİŞTİRMELİ ÖĞRENME TABANLI BİR KONTROLCÜ

Bir alt ekstremite dış iskelet kullanıcısının metabolik enerji tüketiminin çoğu, alt vücut pasif olarak kabul edilebileceğinden, üst vücudun harcadığı efordan gelir. Ancak literatürde alt uzuv dış iskelet kullanıcılarının üst vücut eforu hesaba katılmamaktadır. Bu tez çalışmasında, bastonların yer tepki kuvvetlerini (YTK) optimize eden bir hareket kontrolcüsü geliştirmek için derin pekiştirmeli öğrenme kullanılmıştır. Yer reaksiyon kuvvetlerini minimize etmenin amacı kullanıcının üst vücut eforunu en aza indirmektir. Bastonlar ve insan-dış iskelet sisteminin modeli URDF ve XML formatlarında oluşturulmuştur. Düşmeden ve aşırı eklem torkları olmadan kütle merkezinin ileriye doğru yer değiştirmesini teşvik eden ödül fonksiyonları şekillendirilmiştir. Son teknoloji yöntemler olan Twin Delayed Deep Deterministic Policy Gradient (TD3) ve Proximal Policy Optimization (PPO), RaiSim ve MuJoCo simülatörleri ve çeşitli parametre setleriyle uygulanmıştır. Kullanılan sinir ağları, eklem açısı ve hızları ile ayaklar ve baston uçlarındaki zemin reaksiyon kuvvetlerine dayalı olarak eklem torklarını üretir. Oluşturulan bu eklem torkları doğrudan dış iskelet modeline gönderilir ve derin PÖ çerçevesinin sağladığı aksiyon uygulandıktan sonra yeni bir durum gözlemlenir. TD3 ve PPO yöntemleriyle RaiSim'de eğitilen politikaların, dengeli ve doğal görünümlü bir yürüyüş için uygun kontrol komutları üretemediği gözlemlenmiştir. Genel olarak RaiSim'de PPO yönteminin TD3 yöntemine göre daha yüksek ödül değerlerine ulaştığı görülmektedir. RaiSim ile istenen tarzda bir robot politikası elde edemedikten sonra MuJoCo simülatör olarak kullanılmıştır. Sonuç olarak, uygun baston hareketlerini de içeren ve istenen tarzda lokomosyonu yer tepki kuvvetlerini %35 oranında azaltarak sağlayan bir robot politikası geliştirilmiştir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

$\mathbb{P}$	Probability
s_t	State at time t
a_t	Action at time t
r_t	Reward at time t
R_t	Discounted reward at time t
Γ	Discount factor
V_t^*	Actual reward
V_t	Reward prediction
α	Learning rate
π_i	Policy with parameters i
$A_{\pi\theta}$	Advantage of policy $\pi\theta$
Θ	Policy parameters
τ	Trajectory
H	Entropy function
$\widehat{A}_t$	Estimation of advantage at time t
$\widehat{E}_t$	Expectation at time t
$\boldsymbol{T_{horizon}}$	Horizon of PPO algorithm
h	Beta distribution
Γ	Gamma function
Q^{π}	Critic/Value function
y	Target value in TD3

$\boldsymbol{\varphi_i}$	Critic network parameters
$\mathcal{N}$	Normal distribution
$\mathbb{B}$	Replay buffer
∇_i	Gradient with respect to i
β	TD3-exploration noise coefficient
β_{decay}	TD3-exploration noise decay coefficient
N	TD3 batch size
c	Maximum action
τ	TD3 – Target update coefficient
σ	Standard deviation of Normal distribution
ε	PPO-Clip rate
K_{epochs}	PPO-Epoch number
L_a	PPO-Actor slice length
L_c	Critic slice length
λ	Lambda used in advantage estimation for PPO algorithm

LIST OF ACRONYMS/ABBREVIATIONS

3-D	3 Dimensional
CoM	Center of Mass
CPG	Central Pattern Generator
EMG	Electromyography
GRF	Ground Reaction Force
IMU	Inertial Measurement Unit
LLRE	Lower Limb Rehabilitation Exoskeleton
LR	Learning Rate
MDP	Markov Decision Process
PD	Proportional-Derivative
PPO	Proximal Policy Optimization
ReLU	Rectified Linear Activation
RL	Reinforcement Learning
TD3	Twin Delayed Deep Deterministic
TD	Temporal Difference

1. INTRODUCTION

1.1. Motivation

Locomotion of legged robots is one of the fundamental domains in the field of robotics. Thus, the locomotion behavior is at utmost importance for developing a robotic device. Lower-limb exoskeletons, which are subclass of legged robots, even have an increased level of locomotion complexity as human movements are also involved. Lack of some degrees of freedoms may oblige the exoskeleton user to use crutches for stability purposes. As a consequence, the user uses the upper body to control the crutches, which results in increased metabolic energy consumption. This increased metabolic energy consumption should be minimized in order to improve the comfort of the exoskeleton user. However, there has not been any work in the literature that addresses this issue. The motivation of this thesis is to fill this gap by developing a controller that takes the crutch movements and the crutch contact forces into account in order to minimize the metabolic energy consumption of an exoskeleton user.

1.2. Objectives

In this study, an implementation of state-of-the-art RL algorithms with a reward function including crutch reaction force regulating term for a lower-limb exoskeleton is presented. The developed RL agents are aimed to exhibit a walking behavior with desired gait characteristics and crutch usage.

The first aim is to develop a motion controller for an exoskeleton that minimizes the reaction forces exerted on the crutch tips by the ground. By introducing a cost term into the reward function, excess reaction forces can be penalized.

The second aim is to keep the other gait characteristics such as linear displacement, lateral displacement, tilt angle of the human body and angle of the feet soles with respect to the ground within an acceptable range while achieving the first aim. By achieving these goals, the long term goal to improve the comfort of lower-limb exoskeleton users can be realized.

2. PRELIMINARIES

2.1. Definition of Exoskeleton

The role of wearable robotic devices in daily life is increasing more and more [1]. Recent developments in powered exoskeleton technology offers many new possibilities to paraplegics and will offer increased motion capabilities for physically and neurologically intact people from many different application areas such as military or industry. To put it all in simple terms, a powered exoskeleton is a wearable mobile machine that is composed of links actuated by the attached motors in order to improve the motion capabilities of its user.

Even though exoskeletons are still a hot topic of research, the history of exoskeletons dates back to 1890. The earliest known device that resembles an exoskeleton was developed in that date by a Russian engineer named as Nicholas Yagn [2]. The source of energy of this system was compressed gas bags. In the 1960's, first active exoskeletons were developed in Mihajlo Pupin Institute. The goal of the legged locomotion systems developed in Mihajlo Pupin Institute was assisting in the rehabilitation of paraplegics.

2.2. Classification of Exoskeletons

In general, exoskeletons are classified according to their body part focus, structure (rigid or soft), action (active or passive), actuator type (electric, hydraulic, pneumatic etc.), purpose (recovery or performance) and application area [3]. Apart from full body, lower body and upper body exoskeletons, there are also exoskeletons for specific limbs and joints such as hand, ankle or foot. The action category is about whether there is any actuator to help the user in an active manner without requiring any energy from the user. Passive exoskeletons do not have any actuators on them and they only facilitate the movement of the user. Recovery class exoskeletons are designed for rehabilitation for patients who have lost some of their motor functions, and performance class exoskeletons are designed for augmenting the physical performance of able-bodied users [2].

Recovery exoskeletons can be divided into two parts as full assistance and partial assistance devices. Aim of full assistance exoskeletons is to provide full mobilization to the

people who have lost their lower body motor functions completely by providing movement for their legs via actuators [4]. Since the legs of the user are assumed to be completely passive, the control structure for these exoskeletons excludes the consideration of the torque supplied by the legs which simplifies the control algorithm significantly. On the other hand, partial assistance devices are developed for those who did not lose their lower body motor functions completely. These devices are used to increase the motion capabilities of users who suffers from less serious injuries or aging. The control algorithm for partial assistance devices are more complex than that of full assistance devices because the decreased motion capabilities of the user also come into play. Increased complexity of the interaction between the user and the device leads to increased complexity of the control structure. In this study, an active lower body exoskeleton powered by series elastic actuators with crutches to be designed for full assistance rehabilitation of paraplegics will be considered.

2.3. Control of Exoskeletons

While many hardships such as mechanical design, strength analysis of critical parts of such devices are still present, control of exoskeletons in accordance with the users remaining lower body functionality is also a very challenging problem. Physical and cognitive interaction between user and the exoskeleton plays a huge role in a successful exoskeleton control algorithm. This aspect requires the concepts of trajectory and trajectory optimization to be mentioned. Trajectory is the time history of position and velocity of the joint angles in an exoskeleton and it is the main element that shapes the gait pattern of an exoskeleton or any other walking device. A good trajectory should be feasible in terms of the joint angle restrictions of a human and be comfortable considering the anatomy of humans. Hence, an optimal trajectory generated for an exoskeleton will be crucial for comfort, health and performance of the user.

Exoskeleton control strategies are generally divided into three main layers which are high level, mid-level and low level [4]. The operation mode which describes activities such as walking or stair climbing is decided within the scope of high level control. High level control is not critically important for rehabilitation purposes, but it is still quite important for daily usage of robotic exoskeletons. Mid-level control algorithms make sure that the device acts in a continuous manner for all types of movements and it maps the high level commands

into low level commands [4]. Stability and equilibrium analysis, estimation of environmental factors and calculating the required joint torques are taken care in middle level. Finally, low level control layer executes torque control commands such that higher level intentions are realized as in Figure 2.1.

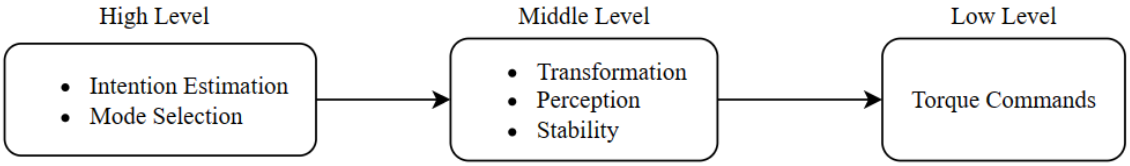


Figure 2.1. Flow chart of exoskeleton control layers.

Even though there are many different possible numerical or analytic motion control methods that can solve the problem of generating correct low level torque control commands, they generally require complex mathematical models that describe the dynamic behavior of the whole system. For example, classic PID control is often used in locomotion tasks, but we still need to mathematically model the system and especially the contact dynamics for being able to minimize the contact forces. Considering the highly nonlinear nature of this kind of problem, it is a good option to explore the capabilities of the reinforcement learning techniques which does not need these expressions of complex system dynamics.

2.4. Reinforcement Learning Background

In this section, some prior information that reinforcement learning methods use will be introduce in order to build a base. Reinforcement learning is a machine learning method based on trial and error. A simulation environment that captures the dynamics of the problem is required in order to implement this method. Reinforcement learning agent interacts with the environment adjust its actions based on the feedback it gets from the environment as shown in Figure 2.2. This feedback is calculated as a reward function which is a design consideration. Desired behaviors are rewarded whereas undesired ones are penalized within the scope of the reward function. However, reward engineering process is a non-trivial process and trials for very different combinations of reward functions and coefficients in the

reward function may be required. The RL agent aims to maximize the cumulative reward over time after doing thousands of iterations.

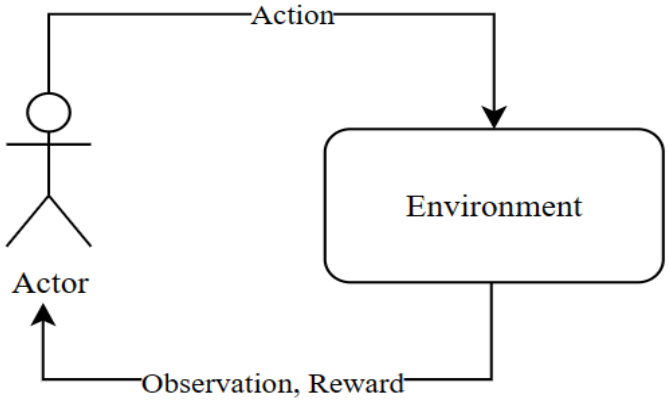


Figure 2.2. Reinforcement learning main structure.

Reinforcement learning methods can be used to solve some various different types of problems such as games, statistics, swarm intelligence, robotics and control theory. Even though reinforcement learning is one of the hot topics in the field of AI, its applications in the real world conditions are still not very widespread. Since reinforcement learning heavily depends on the interaction with the environment, using this approach may not be always feasible especially for situations that require interacting with complex physical environments. The time required to solve this kind of a problem makes it unreasonable to use reinforcement learning. However, if a realistic model of the complex environment can be simulated and by using multiple hardware to train the algorithm, reinforcement learning can still be an option.

There are two main classes of reinforcement learning algorithms, model based and model free reinforcement learning. In the model based approach, the agent tries to build a model of the environment based on how it interacts with the environment [5]. Therefore, the agent is able to predict the corresponding reward before applying the action on the environment. Hence, preferences become more important than results of actions the agent takes and the agent will try to take an action with the maximum reward without any regard of the possible results of that action. On the other hand, model free approaches aim the agent to establish the connection of their actions and consequences based on past experience. Thus,

the agent cannot predict the corresponding reward before applying the action. In such an algorithm, the agent carries multiple actions and adjust the policy according to the reward.

Another way to examine different types of reinforcement learning methods is to distinguish whether the algorithm is off-policy or on-policy. On-policy algorithms need the way that the data is collected to be improved or evaluated [6]. On the other hand, off-policy algorithms learn a policy from a data set which is a result of an arbitrary policy, so they evaluate and improve a policy which is a different policy than that of generates the actions. The past experience of the agent is stored in a structure called replay buffer. The stored batch of data are used to train the network and update the policy accordingly, so the training data generated by the previous policies are also utilized in the off-policy learning.

Even though reinforcement learning is a powerful method for complex tasks with unknown system dynamics, not all types of reinforcement learning algorithms are suitable for continuous control tasks. Continuity of the action and observation spaces, high dimensional action and observation spaces make some reinforcement learning methods unusable. In this situations, model-free deep reinforcement learning methods are better choices as they can learn high dimensional actions and explore high dimensional state spaces, without any need of a mathematical equation system for the dynamics of the system[7].

Deep reinforcement learning is essentially a combination of deep learning and reinforcement learning. It is one of the most rapidly growing areas of machine learning and AI as it can solve many different types of difficult problems which are not possible to be solved by other reinforcement learning methods. These methods owe their performance to deep learning, which incorporates artificial neural networks in order to mimic the learning behavior of living things by modeling the neurons, connections between these neurons and how they interact with each other. Very large scaled raw inputs can be supplied to a neural network without any structuring layer for the observation space. Another advantage of deep RL algorithms is improved generalization [8]. Since deep RL algorithms can work with raw data (such as image pixels), the environment does not need be necessarily defined beforehand. Thus, the algorithm can work together with some new set of inputs that was not a part of training session. For example, the algorithm can predict that there is a cat on the

image although the particular cat or the same species of the particular cat was not introduced during the training.

In deep reinforcement learning, the agent is defined as a neural network or some combinations of multiple neural networks. The number of layers and the number of neurons in each layer, their activation functions and connectivity structure are some of the design considerations. Required actions that should be taken for a specific observation and assessments such as how good it is to be in a state are calculated by these neural networks. The weight of the connections between the layers are initiated and then adjusted on every update sequence depending on the error calculated every step and the result of backpropagation algorithm applied. This general combination of reinforcement learning and deep learning can be seen as a diagram in Figure 2.3.

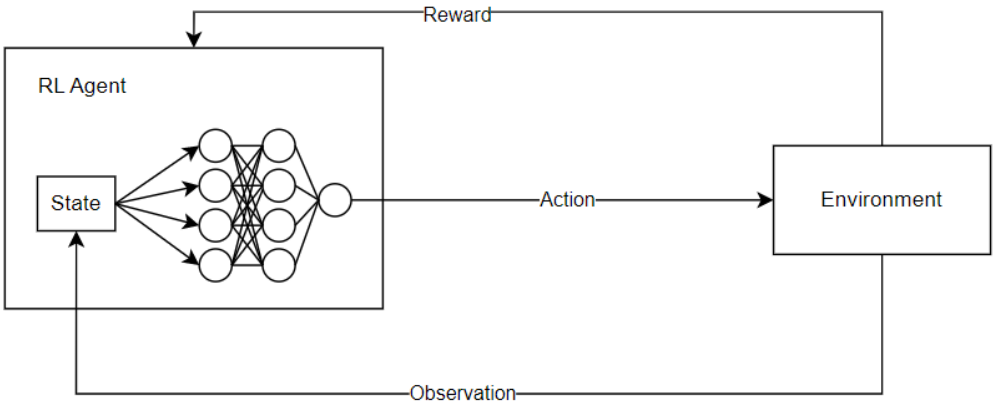


Figure 2.3. Deep reinforcement learning diagram.

2.4.1. Activation Functions

Selection of the activation function is one of the critical parts of the neural network design. It determines how the neurons from different layers interact with each other, which means that it has a direct effect on the learning process. Selection of the output layer activation function affects the generated output by the neural network. Activation functions are mathematical expressions that take the weighted sum of the activations generated by the previous layer as input and apply a transformation based on its internal mathematics. Neural networks are built layer by layer. Input layer takes the raw input, calculates the activations and passes them to the next layer. Output layer is the layer that generates the predictions that

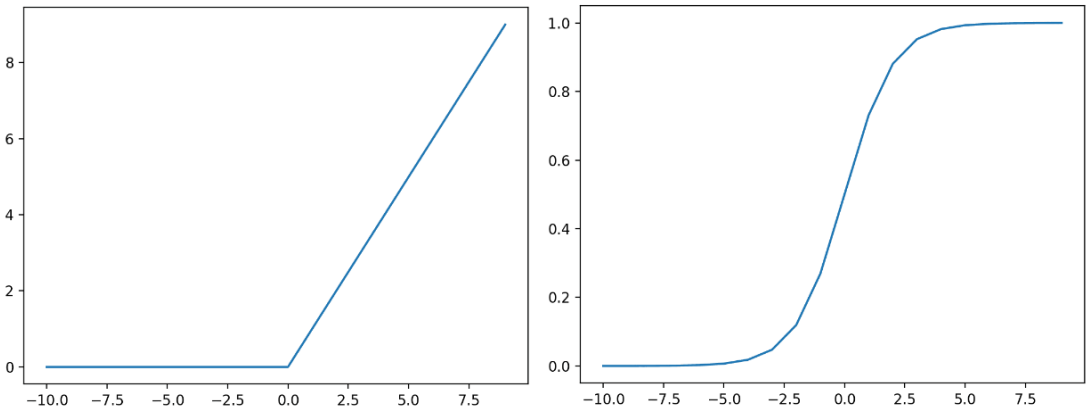
neural network is expected to make. The layers between input and output layers are called hidden layer. Number of the hidden layers is also another design choice. Generally, same type of activation function is used for all hidden layers and a different activation function for output layer may be selected depending on the desired output of the network. Activation functions are required to be differentiable since the error in the output layer should be differentiated in order to optimize the network [9].

The most frequently used activation functions for hidden layers are Rectified Linear Activation (ReLU), Logistic (Sigmoid), and Hyperbolic Tangent (Tanh). Nonlinear dynamics of these functions allow the network to learn more complex functions when compared to linear activation functions. ReLU activation function is the most popular one for the hidden layers thanks to ease of implementation and its lower vulnerability to vanishing gradients problem [9], which describes that the gradients cannot be properly back propagated and the desired behavior is not learned eventually. ReLU function returns the input value unless the input is negative. If the input is negative it returns 0. The behavior of ReLU activation can be seen in Figure 2.4(a). It is generally advised that the input values shall be normalized in the range [0, 1] if the activation function is ReLU. Sigmoid function takes real values as input and squeezes them between the range [-1, 1]. It is a good practice to normalize the input values in the range [0, 1] when using Sigmoid as activation function. Shape of sigmoid activation function can be seen in Figure 2.4(b). Mathematical expression of sigmoid function is

$$\sigma = \frac{1}{1 + e^{-x}}. \tag{2.1}$$

Tanh function takes real input values as input and outputs a value in the range [1, -1] just as sigmoid function does. It is suggested that the input values shall normalized in the range [1, -1]. Plot of tanh activation function is also available in Figure 2.5. The equation of tanh function is

$$\tanh = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}. \tag{2.2}$$



a) ReLU b) Sigmoid

Figure 2.4. Input/output plot of ReLU and sigmoid.

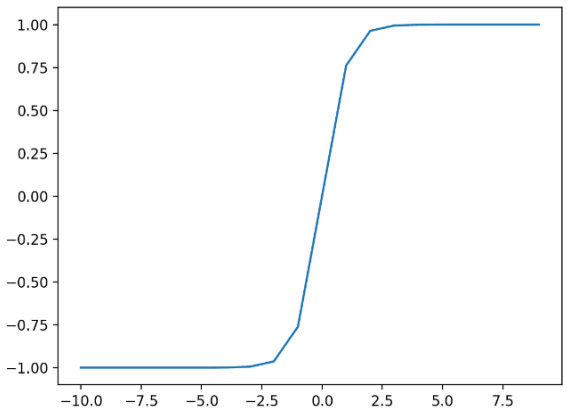


Figure 2.5. Input/output plot of tanh.

The selection of activation function for hidden layers is done according to some considerations. A neural network uses the same activation function for all hidden layers in most of the situations. Tanh activation function was the default option for hidden layers until the end of the first decade of 2000s. However, tanh and sigmoid functions are vulnerable to vanishing gradient problem and ReLU activation function has become a more viable option in the modern neural network implementations [9]. The type of the network plays a huge role for selecting a proper activation function. For Convolutional Neural Networks and Multilayer Perceptron, ReLU is the best option most of the time, whereas sigmoid and tanh function is more reliable for Recurrent Neural Networks.

There are four types of output layer activation functions that worth mentioning, which are linear, logistic (sigmoid), softmax and softplus. Linear activation function is also known as “identity” or “no activation” because it outputs the input value directly. Sigmoid function

has already been in introduced in the previous paragraphs. The softmax activation function outputs a vector of values and the sum of these values are one, so it can be interpreted that these values are probabilities. Mathematical expression of softmax is

$$\text{softmax} = \frac{e^x}{\sum e^x}. \tag{2.3}$$

Softplus activation function is the smoothed verison of ReLU and it can be used in the hidden layers as well. ReLU has a discontinuity around the origin. Even though this does not cause much trouble in practice, researchers proposed the usage of softplus at the beginning of 2000s because of its continuity [10]. Dombrowski et al. [11] claimed that the gradients with respect to the input becomes unstable in case any slight perturbation exists and softplus solves this problem. The plot of softplus function is available in Figure 2.6 and it expressed by the following equation

$$softplus = \frac{\log\left(1 + e^{(\beta * x)}\right)}{\beta}, \tag{2.4}$$

where β is a parameter with default value one. Softplus is differentiable everywhere and its derivative is basically sigmoid function.

Selection of the activation function for output layer is what determines the output essentially, so it is a significant aspect of neural network usage. The most important aspect to consider when choosing the activation function for output layer is the type of the variable that network predicts. If the problem is a regression problem, linear activation function is generally the best option. However, sigmoid or softmax generally lead to better results for classification type of problems [12].

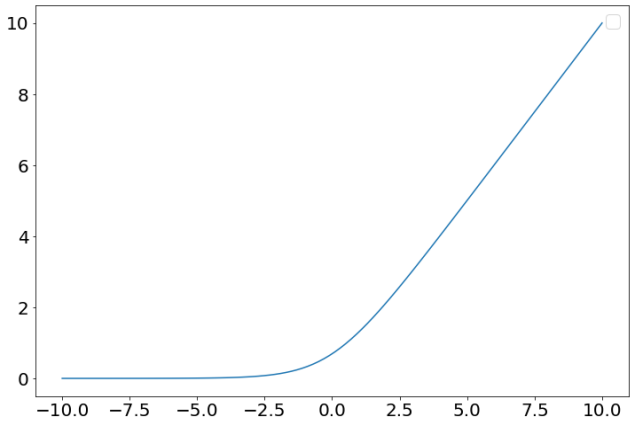


Figure 2.6. Input/output plot of softplus.

2.4.2. Markov Decision Process

Markov Decision Process (MDP) is a mathematical environment that describes a formulation where the current state characterizes the future process and this framework is useful to model the decision making processes [13]. Markov Decision Process is a process where all state-action pairs have Markov property. Markov property is defined as

$$\mathbb{P}\{s_{t+1} = s' | s_t, a_t, s_{t-1}, a_{t-1}, \dots, s_0, a_0\} = \mathbb{P}\{s_{t+1} = s' | s_t, a_t\}, \quad (2.5)$$

where s is state, a is action.

The meaning of the Equation 1.5 is that if the state and action pair is known at time t, then the state at time $\mathrm{t}+1$ only depends on state and action pair at time t. The state-action values before time t is irrelevant in this condition. If the environment has this Markov property, then the next state, next reward can be predicted, given the current state of the environment and current action. This property is crucial for reinforcement learning point of view and most problems can be described as MDP. Thus reinforcement learning methods are based on MDP's.

2.4.3. Temporal Difference Learning

Temporal Difference (TD) Learning is an approach used to estimate the total reward over a time horizon by reinforcement learning methods in an unsupervised way [14]. The most important concept of TD learning is the discounted reward which is expressed as

$$\begin{aligned} \mathrm{R_t} &= \mathrm{r_{t+1}} + \gamma \mathrm{r_{t+2}} + \gamma^2 \mathrm{r_{t+3}} + \cdots \gamma^{\infty} \mathrm{r_{\infty}} \\ &= \sum_{\mathrm{k=0}}^{\infty} \gamma^{\mathrm{k}} \mathrm{r_{t+k+1}}\,, \end{aligned} \quad (2.6)$$

where γ is discount factor and t is time. Discount factor depicts how much the future rewards are valued. It is a real value between 0 and 1.

TD error is based on the difference between the actual reward $\mathrm{V_t^*}$ and the reward prediction $\mathrm{V_t}$:

$$\mathrm{E_t} = \mathrm{V_t^*} - \mathrm{V_t}$$

$$= r_{t+1} + \sum_{k=1}^{\infty} \gamma^k r_{t+k+1} - V_t$$
$$= r_{t+1} + \gamma(\sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k+1}) - V_t$$
$$= r_{t+1} + \gamma(\sum_{k=0}^{\infty} \gamma^k r_{(t+1)+k+1}) - V_t$$
$$= r_{t+1} + \gamma V_{t+1} - V_t, \quad (2.7)$$

After calculating the error, the update is done to the current value as

$$V_t \leftarrow V_t + \alpha E_t$$
$$= V_t + \alpha(r_{t+1} + \gamma V_{t+1} - V_t), \quad (2.8)$$

where α is the learning rate. Learning rate determines how fast the learning process will be. It is a real value between 0 and 1. Learning rate should be tuned as optimal as possible because high learning rates may cause diverging from optimal results and aggressive fluctuations. A low learning rate is somewhat safer in terms of convergence, but it leads to slower training.

2.4.4. Actor-Critic Methods

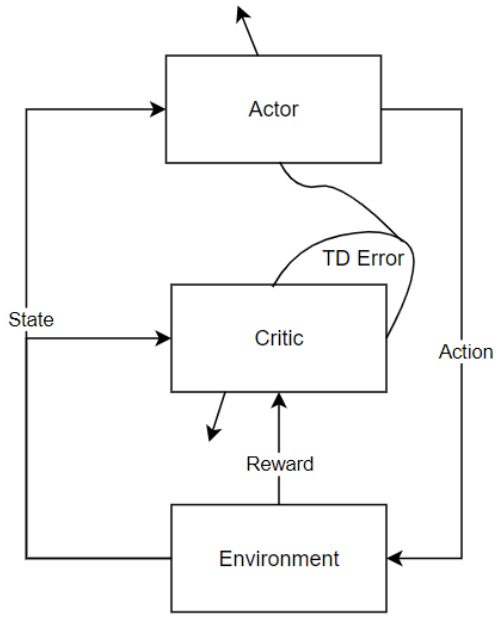


Figure 2.7. Actor-critic architecture.

Actor-Critic methods are TD learning methods that uses two different neural networks which are policy network and value network. Actor network maps the observation vector to the probability distribution over the possible actions that the agent can take or directly the actions. Value network predicts the expected return that the agent will obtain if it starts from a particular state and acts according to the current policy from this point on. In other words, critic network criticizes the actions generated by actor based on the current policy. A general Actor-Critic architecture is shown in Figure 2.7.

Actor-Critic methods are based on the gradient that is calculated by the equation

$$\nabla J(\theta) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t, s_t)\, A_{\pi\theta}(s_t, a_t), \tag{2.9}$$

where $A_{\pi\theta}$ is the advantage function and π_θ is the policy network with parameters set θ. Advantage function is basically the TD error which was introduced earlier as the Equation 2.7. Equation 2.9 is also known as policy-gradient expression. In each iteration, these gradients are used to update the policy network parameters according to the equation

$$\theta \leftarrow \theta + \alpha \nabla J(\theta), \tag{2.10}$$

where θ is the policy network parameters.

2.4.5. Policy Gradients

Policy gradients are frequently used in RL algorithms due to their dependence of simple principles [15]. The policy is defined as

$$\pi(a|s), \tag{2.11}$$

where π depicts the probability of taking the action a, given a state s. Expected rewards can be formulized as

$$\begin{aligned} J(\theta) &= E\left[\sum_{t=0}^{T} R(s_t, a_t); \pi_\theta\right] \\ &= \sum_{\tau} P(\tau; \theta) R(\tau), \end{aligned} \tag{2.12}$$

where τ depicts a trajectory. The ultimate goal is to develop a policy π_θ that creates a trajectory τ which includes state-action pairs

$$(s_1, a_1, s_2, a_2, \dots, s_T, a_T),$$

which maximizes the expected return. Thus the objective function becomes

$$\max_\theta J(\theta) = \max_\theta \sum_\tau P(\tau;\theta)R(\tau), \tag{2.13}$$

which means a maximization problem of the sum of the expected rewards.

Policy gradient approach makes uses of a derivation trick defined as

$$\begin{aligned} f(x)\nabla_\theta \log f(x) &= f(x)\frac{\nabla_\theta f(x)}{f(x)} \\ &= \nabla_\theta f(x). \end{aligned} \tag{2.14}$$

Replacing $f(x)$ with policy $\pi_\theta(\tau)$ gives

$$\pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau) = \nabla_\theta \pi_\theta(\tau). \tag{2.15}$$

The expected return or continuous space can be expressed as

$$E_{z\sim p(x)}[f(x)] = \int p(x)f(x)dx, \tag{2.16}$$

and the goal becomes the following equation defined as

$$\begin{aligned} \theta^* &= \arg\max_\theta J(\theta) \\ &= \arg\max_\theta E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(s_t, a_t)\right]. \end{aligned} \tag{2.17}$$

Modifying the Equation 2.16 according to $\pi_\theta(\tau)$ gives

$$\begin{aligned} J(\theta) &= E_{\tau\sim\pi_\theta(\tau)}[r(\tau)] \\ &= \int \pi_\theta(\tau)r(\tau)d\tau, \end{aligned} \tag{2.18}$$

and taking the gradient of this objective function gives the policy gradient defined as

$$\nabla_\theta J(\theta) = \int \nabla_\theta \pi_\theta(\tau)r(\tau)d\tau$$

$$= \int \pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau) r(\tau) d\tau$$
$$= E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)]. \quad (2.19)$$

Taking the gradient of $\log \pi_\theta(\tau)$ is required in order to compute the policy gradient. $\pi_\theta(\tau)$ is defined as

$$\pi_\theta(\tau) = \pi_\theta(s_1, a_1, \dots, s_T, a_T)$$
$$= p(s_1) \prod_{t=1}^{T} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t), \quad (2.19)$$

and taking the log of both sides gives

$$\log \pi_\theta(\tau) = \log p(s_1) + \sum_{t=1}^{T} \log \pi_\theta(a_t|s_t) + \log p(s_{t+1}|s_t, a_t), \quad (2.20)$$

where $p(s_{t+1}|s_t, a_t)$ is the probability of transitioning into s_{t+1} from s_t if the action a_t is applied. The first and the last term in Equation 2.20 do not depend on network parameters θ, so they can be ignored while substituting it into Equation 2.19. Then Equation 2.19 becomes

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \right) \left(\sum_{t=1}^{T} r(s_{i,t}, a_{i,t} \right), \quad (2.21)$$

and it is used to update the weights such that

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta). \quad (2.22)$$

The term $\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})$ in Equation 2.21 is called maximum log likelihood and it is a measure of how likely the trajectory is under the current policy. The aim of multiplying it with the rewards is to increase the likelihood of a policy if the trajectory gives a positive reward. Simultaneously, if the policy results in negative reward, the likelihood of this policy is decreased.

2.4.6. Entropy in Reinforcement Learning

In many reinforcement learning algorithms such as policy gradient and actor-critic algorithms, the actions are defined over a probability distribution like Gaussian distribution. Entropy is a measure of randomness for the agent's actions. The term "entropy" is originally

a physics term and it refers to the lack of order in a system whereas its meaning in terms of reinforcement learning corresponds to the unpredictability of the actions generated by the agent. In some RL algorithms, entropy is introduced into the objective function as

$$J(\theta) = E[R(s_t, a_t) + \beta H(\pi_\theta(s_t))], \tag{2.23}$$

where $H(\pi_\theta(s_t))$ is the entropy of the policy π_θ in the state s_t. This added term is sometimes called entropy bonus and it is defined as

$$H\big(\pi_\theta(s_t)\big) = -\sum_a \pi_\theta(s_t, a) \log \pi_\theta(s_t, a). \tag{2.24}$$

Entropy bonus improves the exploration behavior of the agent and prevents the policy to get stuck at a local optimum. However, the entropy coefficient should be tuned carefully because if it is too high, entropy dominates the other components of objective function which introduces a high degree of chaos. Similarly, too low entropy prevents the agent to explore the environment and converge to better optima.

2.4.7. Q Learning

Q Learning is a model free reinforcement learning algorithm which is primarily built on estimating the value of an action in a particular state [16]. It utilizes a table named as Q-table which is a lookup table in order to estimate the expected returns for each possible action in each state. "Q" stands for quality and intuitively Q value can be described as the quality of the action. Q learning uses Bellman equation which is shown as

$$V(s_t) = \max_{a_t} R(s_t, a_t) + \gamma V(s_{t+1}). \tag{2.25}$$

The stochastic version of Bellman equation is

$$V(s_t) = \max_{a_t} \left(R(s_t, a_t) + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t)\, V(s_{t+1}) \right). \tag{2.26}$$

Q function takes inputs s_t and a_t in order to estimate the Q value as

$$Q^\pi(s_t, a_t) = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | s_t, a_t], \tag{2.27}$$

which is basically expected discounted reward. This calculation is used to update Q values according to the equation

$$Q_{new}(s_t, a_t) = Q(s_t, a_t) + \alpha[R(s_t, a_t) + \gamma * \max Q^{\pi}(s_t, a_t) - Q(s_t, a_t)]. \quad (2.28)$$

The next action is selected based on the new version of Q-table such that the action has the maximum Q value. This update is applied iteratively until convergence.

3. LITERATURE

In this section, state-of-the-art studies in the field of exoskeletons, bipedal robots, trajectory optimization and reinforcement learning which can provide some insight about the thesis topic are summarized.

Optimal control strategies are widely used in the literature in the context of exoskeletons and bipedal robots. One of them is direct collocation. In this method, system dynamics, cost function and all constraints are discretized by integration approximations such as Simpson's rule or trapezoidal rule. Then the problem is reduced to nonlinear programming problem that can be solved by numerous software packages such as MATLAB. Chao et al. [17] used this method to generate optimal gait trajectories with the cost function known as "mechanical cost of transport" which corresponds to the amount of mechanical energy the device spends over the period of gait and generated feasible and optimal gait patterns. Ackermann et al. [18] also showed functionality of direct collocation methods on optimizing trajectories of human gait. Kelly [19] created an open source MATLAB trajectory optimization library which uses direct collocation methods and demonstrated efficacy of these methods on different problems including a five-link bipedal robot. Hereid et al [20] used direct collocation method for trajectory optimization and showed that optimization results yield dynamic and stable gaits for a bipedal robot. Consequently, usability and efficiency of direct collocation methods on walking robotic systems are shown many times in the related literature.

Luo et al. [21] proposed a novel motion controller that uses reinforcement learning for squat assistance on a lower body exoskeleton. The aim of the exoskeleton which has actuation on ankle joint for both sagittal and front planes is to help the user to complete squat movement without any loss of efficiency, stability and robustness. A PD controller is used the track the desired joint angles which are the outputs of the reinforcement learning algorithm. Multi-layer perception that was used in the study is composed of 3 fully connected layers with ReLu activation function. Proximal Policy Optimization which is one of the most effective policy gradient algorithms is used to train the network. The center of pressure is estimated by the help of force sensors on the foot and utilized in the observation space and

in the reward function. Additionally, dynamics randomization and perturbations forces are also used during the training session in order to improve the robustness. The effectiveness of the proposed strategy is shown in the experiments with different environmental conditions.

Taylor et al. [22] introduced a method to teach a real world bipedal robot to walk directly from motion capture data by using reinforcement learning. Required actuator commands to realize a desired motion are derived as a result of the proposed method. The acceleration and angular velocity values of the torso that are measured by an IMU, joint angles, surface reaction forces and time measured by a built in sensor are used as the observations for the reinforcement learning algorithm. The state vector is composed of the velocities of the joint angles. Resultant joint velocities are converted into joint position targets by an integrator and PD control is used to track these target joint positions. This incorporation of integration has some advantages such as avoiding motor jitter and smoothing the joint angle curve. PPO algorithm is utilized in the learning process. Policy and value networks are created such that they have 3 layers which are composed of 128 neurons with softsign as the activation function. Reward function is shaped such that it includes a term that checks the difference between the target link orientations and the actual ones, a term that penalizes self-collisions, a term about foot contacts and orientation of the foot during contact and a term that penalizes out of range joint angles. The algorithm is trained for different initial conditions during 2500 episodes. Domain randomization is also incorporated such that random motor backlash, friction coefficient and Young's modulus are included during training. Consequently, the trained network can successfully imitate the human gait even though there is still room for improvement in terms of overall robustness.

Lillicrap et al. [23] demonstrated the generalized efficacy of deep reinforcement learning on various tasks. A novel algorithm called Deep Deterministic Policy Gradient (DDPG) is introduced to solve continuous control problems. It is a combination of Deep Q Network algorithm [24] and actor-critic approach where actor generates action and critic criticizes it. It is tried on very different continuous control problems that include multi-joint dynamics and unstable contact dynamics such as cart pole, pendulum swing-up, puck shooting, robotic manipulation, legged locomotion and autonomous driving. One of the most

interesting aspects of the work is that the same hyper parameter settings and network topology which decreases the amount of engineering required significantly.

Peng et al. [25] presented a imitation learning based method that can teach quadrupedal robots very complex animal movements that include extreme agility skills. The input of the proposed framework may be motion capture data obtained from the movements of real animals. There are 3 main steps implemented in order to achieve the challenging goal of imitating real animal movements. Firstly, the reference motion is processed for doing the mapping between reference morphology and robot morphology. Secondly, the mapping done in the first step is used to train the network that imitates the reference movements. Finally, the developed robot policy is transferred to a real robot. Policy network which has 2 layers with 512 and 256 neurons outputs a Gaussian action distribution. The framework can successfully learn to imitate a wide range of dynamically challenging movements such as backward trot, running, turning, hop-turning. Moreover, the developed policy is able to perform these movements after a sim-to-real transfer process.

Margolis et al. [26] developed a framework for a controller that enables a quadrupedal robot to sprint and turn with high speeds on very different and challenging natural terrains such as ice and grass. Reward function is engineered such that linear, angular velocity are tracked and some additional conditions for stability, smoothness and safety are regulated. Since some tilting movement through forward direction is observed during training sessions, some penalty terms for body height and orientation are also added into reward function. PPO algorithm is trained with randomized parameters for body mass, center of mass, motor parameters, friction and restitution coefficients of ground in order to realize a better sim-to-real transfer. Like many of the reinforcement learning studies, neural network outputs some target joint positions which are to be tracked by a PD controller. Additionally, curriculum learning is incorporated in order to enable high speed locomotion. It is found out that the policy cannot learn desired behaviors without any curriculum. The result is that the random exploration at the beginning rarely leads to movements with fast body motion, which eventually keeps the reward small. Trained policy is tested on various real world conditions by using a quadrupedal robot. The robot has outperformed the previous sprint speed record which was achieved by model predictive control for the same robot configuration. Then the robot is also tested on outdoor conditions and even if there is a decrease of sprint speed, the

robot can still sprint at a quite high speed. Finally, the robot is tested for yaw control which resulted in with a yaw rate of 5.7 rad/s and for terrain change, hardware failure conditions, which demonstrated improved agility.

Deep reinforcement learning has shown its efficacy in not only bipedal/quadrupedal locomotion tasks, but also in other modes of locomotion such as snake locomotion. Shi et al. [27] used reinforcement learning in order to derive gaits for multi-link snake shaped robots in both aquatic and terrestrial environments. Links are attached at the end of each other and this simple design is used for terrestrial and aquatic versions of snake robot. Terrestrial version includes some wheels as well. The observation space consists of relative angles of two joints of the robot and orientation of the robot in inertial frame whereas the action space consists of the joint velocities. Two different reward functions are utilized depending on two different tasks, moving forward or backward and changing the direction of the robot to right or left. Forward displacement, orientation and robots orientation displacement in the desired direction are used to shape the reward function. Since the dimensionality of the problem is low, a network with 2 hidden layers with 50 neurons is enough. The created network is trained by Deep Q Network method and it converges to an optimal value after 5000 iterations for both configurations. A prototype of the wheeled snake robot is created and transferring the learned policy into real world conditions is presented as a future work.

Another type of robot geometry that can be used along with deep reinforcement techniques is tensegrity robots. Zhang et al. [28] has developed a novel method that uses reinforcement learning to achieve efficient gait for tensegrity robots. Tensegrity robots are composed of some solid links and elastic elements that connect these links to each other. Despite their intricate motion dynamics which makes it cumbersome to control them, they are used for exploring planets. A tensegrity robot called SUPERball which has 6 solid rods, and 24 elastic cables is used in the study. 12 of the elastic cables can be actuated. The complex gait dynamics of this robot includes contracting the elastic cables such that area of the base shrinks. The decrease in the area of the support polygon makes the robot tip over. A 12-dimensional observation space which incorporates the acceleration measurements along the bar axis. Action space is composed of desired positions of 12 different motors. A neural network with 3 hidden layers each with 64 neurons activated by ReLU activation

function. The network is trained by an improved version of MDGPS algorithm [29] that encompasses periodic behaviors. During training, some parameters such as terrain slope, gravitational acceleration, noise level for the inputs are varied in a systematic way. The robot can successfully learn to perform an efficient gait on the simulation environment. Moreover, the trained policy is directly transferred into real world without any post-tuning and the robot can travel distances such as 12m, 9m and 8m by performing rolling movements. Therefore, it is stated that the proposed method achieved even better results when compared to some hand engineered control algorithms.

Ouyang et al. [30] has demonstrated that reinforcement learning based methods can be used to develop bioinspired adaptive locomotion controllers for hexapod robots as well. Central Pattern Generators (CPGs) are a part of the central nervous system and they generate some commands in order to achieve complex and rhythmic locomotion patterns in animals. A DDPG reinforcement learning agent that takes the position and velocity of the robot body, joint torques, angles and first derivatives of the angles is implemented. This agent optimizes 2 parameters of CPGs, which are amplitude and phase difference between the hip joint and knee joint. CPG based controller generates the desired joint positions based on these parameters and these target angles are tracked by PD controller. Velocity direction and the energy consumption are considered in terms of reward function. In conclusion, it has been showed that the proposed controller structure can very good maneuverability, adaptability t different environments, robustness and stability.

Deep reinforcement learning has been used to generate the desired joint torques for bipedal robots or lower body exoskeletons most of the time. However, setting the output layer such that it outputs the joint torques directly is also a valid option. Rose et al. [7] used deep neural networks in order to develop a controller that generates the correct torque values directly. A neural network that generates joint torques based on observations for joint angle, velocity, acceleration, actuator torque, speed and joint angle goals is established and trained by DDPG algorithm. A common reward scheme is used to generate a general reward function that encompasses the successful tracking of goal joint angles and a penalty for exceeding the joint limits. This common reward scheme is implemented for all the joints and they are summed up in order to generate the overall reward. OpenSim-RL, which is an open source reinforcement learning environment for multibody physics simulations, is used as the

simulation environment. Desired gait pattern is obtained via applying inverse kinematics after using the joint torque profiles which are available in OpenSim. The network is tested after the training and it is observed that the desired joint trajectories are tracked even in the presence of small perturbations. Being able to track the unseen trajectories in the training is given as a future work.

Another deep reinforcement learning method that may be used for developing exoskeleton controllers is TD3 (Twin Delayed Deep Deterministic Policy Gradient). TD3 is an improved version of DDPG algorithm. Some shortages of DDPG algorithm such as unstable convergence, overestimation of the critic network outputs, and over-reliance of hyper parameters are solved in TD3. Oghogho et al. [31] proposed a TD3 based method to regulate the assistance levels of upper limb exoskeletons. The aim of the user is detected based on the EMG activity of the muscles and assistive actions are generated, so EMG signals are included in the observation space along with the joint angles. Based on these observations, the network outputs the assistive gains for the actuators. Large overshoots in the joint positions, EMG activities of the corresponding muscles and out of range assistive gain values are penalized in the reward function. The proposed system is able to learn how to output the desired assistive gains without any knowledge of muscular capability of the user and the complex dynamics between the exoskeleton and human. The resultant TD3 agent is used in the real world experiments and helped the user to lift a 4 kg object by decreasing the muscular activity by 15 %.

Luo et al. [32] presented novel controller based on a deep reinforcement learning framework which includes three independent neural networks for lower limb exoskeletons. High degree of uncertainty in the dynamics between the human and exoskeleton poses challenge for developing efficient controller and conventional methods require much effort in order to tune many different parameters within the control system. In this study, deep RL is utilized as a way to come up with a controller that considers the interaction forces between exoskeleton and human. Muscle-actuated human control loop is used to generate realistic interactions between human and exoskeleton. The aim of the LLRE (Lower Limb Rehabilitation Exoskeleton) control policy loop is to generate the required joint torques that track a reference motion. Muscle coordination network in muscle-actuated human control loop is a network trained in a supervised way and it predicts the muscle activations of the

user while walking with the exoskeleton based on the joint accelerations and muscular state by minimizing error for desired joint acceleration and actual joint acceleration. Interaction network outputs the human joint angle targets based on joint angles and joint angle velocities such that the interaction forces between the human and exoskeleton are small. PD controllers are used to generate the joint torques for LLRE control loop and joint accelerations for muscle-actuated human control loop such that the target joint angles are tracked. All the neural networks in the loop are trained simultaneously. Reward function for motion imitation network consists of following multiple sub rewards. Imitation reward encourage the exoskeleton to track the reference trajectories. Root reward aims to track the root's position and rotation trajectories. Center of Pressure reward encourages the controller to produce actions that leads to situations where the center of pressure is within the support polygon, which is the safe region in terms of stability for the center of pressure. Action smoothness reward penalizes high second order derivatives of actions. Foot clearance reward tries to ensure that roll and pitch angle of the swing foot in order to keep the foot parallel to the ground as much as possible for avoiding tripping. Lastly, torque reward encourages the agent to generate actions with superabundant joint torques. PPO algorithm is used to train the network. During training, maximum isometric forces of muscles are randomized within a range in order to generalize better such that different people with different muscle properties can use the system without patient specific parameter tuning. After the training, the controller is tested with multiple users with varying physical health status such as healthy, muscle weakness or hemiplegic in an open source simulator DART. The proposed multilayered control algorithm can successfully help the users to have a stable and efficient gait.

4. METHODOLOGY

This thesis study is part of a TÜBİTAK project (project no 118E922) about lower-limb exoskeletons. Minimization of metabolic energy is determined as one of the topics that we will contribute as Boğaziçi University. After examining usability of conventional control methods, it has been decided that RL shall be used to minimize the necessary reaction forces because of less required mathematical model complexity when compared to the classic methods. Since the lower portion of the body is assumed to be completely passive in this project, lower body metabolic energy consumption is ignored. The GRF's on the crutch tips, which can be considered as an indicator for the upper body metabolic effort, are identified as the minimization goal. The exoskeleton-human system is modeled in accordance with the real exoskeleton developed in Özyeğin University. Physics simulators RaiSim and MuJoCo are used to simulate the dynamics of the problem. After failing in RaiSim trials, MuJoCo is adopted as the physics simulator and different RL agents that reaches the goal of GRF minimization are developed.

4.1. Simulation Environment and Modeling

Machine learning methods generally operate on a readily available training datasets and optimize their mathematical models by using the dataset. However, reinforcement learning methods do not require any training dataset which is available prior training. Instead, they require a simulation environment to be able to realize the problem virtually. This simulation environment supplies the training data as a result of the interactions between the RL agent and the environment. Moreover, the data is not given as a whole by simulation environments. The data becomes available to the algorithm piece by piece. As all other machine learning methods, reinforcement learning also depends heavily on the quality of training data coming from the simulation environment. Therefore, the simulation environment is expected to capture all the important dynamics of the problem as realistic as possible and supply the observations such as joint states, reaction forces and dynamic effects to the agent. Especially in legged locomotion, the stability of the overall structure is extremely dependent of the accuracy of contact force modeling. These forces establish the basis for interaction between the robot and the ground. Even though there are many different

physics engines that simulate these contacts, the methods they use for these calculations differ a lot and it is necessary to pick the most suitable one based on the advantages and disadvantages.

4.1.1. OpenAI Gym

One of the most well-known, popular and easy to use reinforcement learning environments is OpenAI Gym and it contains many different problems varying from robotics and control tasks to various type of games. It includes a wide variety of environments such as bipedal walker, inverted pendulum and cart pole. OpenAI Gym is generally used for benchmarking the efficacy of new algorithms and as a learning environment for reinforcement learning new starters thanks to ease of implementation.

There are some crucial and useful features of OpenAI Gym in terms of interacting the environment and they generally form the basis of reinforcement learning algorithms. These functions are:

- reset : Resets the environment to the initial conditions and returns the initial state values
- step: Implements the action to the environment and returns some information about the environment after the action. The retuned information are:
    - state: State of the environment after the action
    - reward: Reward obtained after applying the action
    - done: The signal that terminates the episode based on whether the goal is reached or any termination criteria are satisfied
    - info: Any other information depending on the environment which can help the debugging.

By following the same structure of readily available OpenAI Gym functions, new classes that include a physics simulator and customized definitions of required functions such as reset and step are shaped in this study. For example, problem specific termination criteria should be described within done function such as terminating the episode if the robot falls. Additionally, the state vector and reward function that was engineered such that the desired functionalities are mathematically described also have been added into this new class

that draws the boundaries of the task to be solved. This process requires a meticulous selection of the physics simulator and the integration of it into the reinforcement learning environment. In the next parts, some important physics simulators will be introduced.

4.1.2. MuJoCo

MuJoCo (Multi-Joint Dynamics with Contact) is an open source multi body physics simulator that can calculate the rigid body contact forces accurately. It has very different types of robot models available such as one legged, bipedal, quadrupedal robots. MuJoCo has been one of the most frequently used multibody physics simulators for reinforcement learning methods, especially for locomotion tasks. Fujimoto et al. [33] has showed the efficacy of MuJoCo for different types of locomotion tasks by producing efficient gaits for all the available robot configurations in MuJoCo. Due to its proven efficacy in locomotion tasks, MuJoCo is selected as one of the physics simulators that have been tried in this study.

4.1.3. RaiSim

RaiSim is another multibody physics engine designed to simulate robotic systems and it is one of the most recent ones. Moreover, any multibody object that is built by URDF convention can be simulated in RaiSim and the overall system can be visualized by RaiSim Unity including the direction and magnitude of ground contact forces and contact points. Hwangbo et al. [34] has tested this physics simulator that they created and it is claimed that speed and accuracy is generally better than some other popular physics simulators. The general information about these frequently used simulators is available in Table 4.1 and the comparison between these simulators on different task is available in Table 4.2. These tables are inspected to select the simulators used in this study. RaiSim has a distinctive method for simulating the contact forces and this helps it to simulate multibody or single body tasks quite successfully. Thus, it is expected that RaiSim will become one of the most frequently used multibody physics simulators due to its prowess. As an addition to MuJoCo, RaiSim is also selected to model the exoskeleton and human system that includes the crutches also in order to use the overall model as a simulation environment for the reinforcement learning process in this study.

Table 4.1. General information about frequently used physics simulators [34]

	RaiSim	Bullet	ODE	MuJoCo	DART
Initial release	Unreleased	2006	2001	2015	2012
Author	J. Hwangbo D. Kang	E. Coumans	R. Smith	E. Todorov	J. Lee et al
License	Proprietary	Zlib (open-source)	GPL / BSD (open-source)	Proprietary	BSD
Main purpose	Robotics	Game, Graphics	Game, Graphics	Robotics	Robotics
Language	C++	C / C++	C++	C	C++
API	C++	C++ / Python	C	C	C++
Contacts	Hard	Hard/Soft	Hard/Soft	Soft	Hard
Solver	Bisection	MLCP	LCP	Newton / PGS / CG	LCP
Integrator	Semi-implicit Euler	Semi-implicit Euler	Semi-implicit Euler	Semi-implicit Euler / RK4	Semi-implicit Euler
Coordinates	Minimal	Minimal	Maximal	Minimal	Minimal

Table 4.2. Comparison of physics simulators [34]

	RaiSim	Bullet	ODE	MuJoCo	DART
Rolling	++	+++	-	+	-
Bouncing	++++	++	+++	-	+
666	+++	+	++	+	+
Elastic 666	++++	++	+++	-	+
ANYmal PD	+++++	+++	+	++++	++
ANYmal Momentum	+++	++	+++++	++++ (RK4) ++ (Euler)	+
ANYmal Energy	++++	+++	++	+++++ (RK4) +++ (Euler)	+

4.1.4. Exoskeleton-Human System Modeling

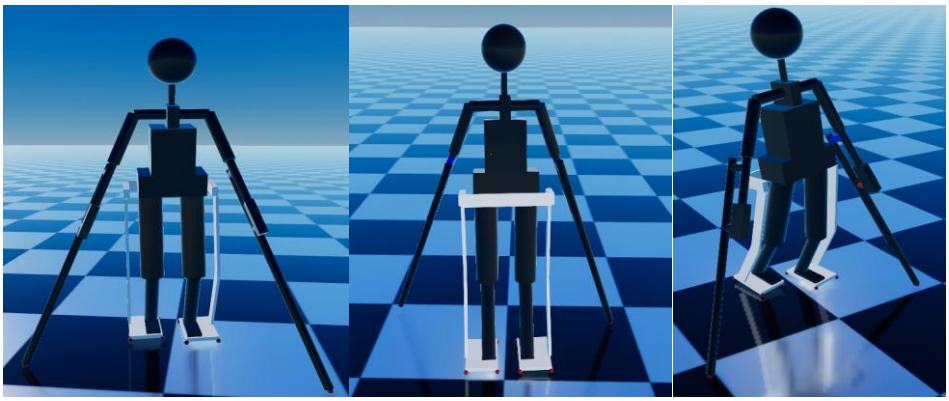
In order to create the exoskeleton-human compound model, a human model which has already been implemented with URDF convention is surveyed. Even though, there are not many implementations of a human model with joint actuations, a URDF model which was originally created for simulations on Gazebo, which is another dynamics simulator used for robotics, is available. [35] The models in this repository are generated by Human Model Generator [36] and using motion capture data. The links in the human body are modeled as cylinders and boxes with different dimensions. There are 8 different models with different measurements as shown in Table 4.3. Subject 2 is chosen as the model to be used because it has the closest measurements the real exoskeleton in Ozyegin University in terms of hip height and knee height. This model contains nearly every present degree of freedoms in a healthy human body. In order to decrease the dimensionality of the problem and simplify the problem, some unnecessary degrees of freedom are removed by setting the joints as fixed joints. Ankle, knee, hip, arm and shoulder joints in the sagittal plane are preserved.

Table 4.3. Measurements of human subjects [36]

Subject	Mass [kg]	Height [cm]	Foot size [cm]	Arm span [cm]	Ankle height [cm]	Hip height [cm]	Hip width [cm]	Knee height [cm]	Shoulder width [cm]	Shoulder height [cm]	Sole height [cm]
1	62.2	168	24	163	8	91	25	48.5	35.4	140	-
2	79.4	176	26	169	8	94	33	48	40	140	-
3	75.4	180	27	190	8	102	28	58	43	148	-
4	72.7	182	26	197	8	102	29	56	42	150	-
5	55	168	24	168	8	98	25	52	38	139	-
6	71.2	179	29	180	8	100	31	49	43	147	-
7	78.9	178	28	192	8	102	30	52	44	148	-
8	55.2	166	25	170	8	90	28	45	37	139	-

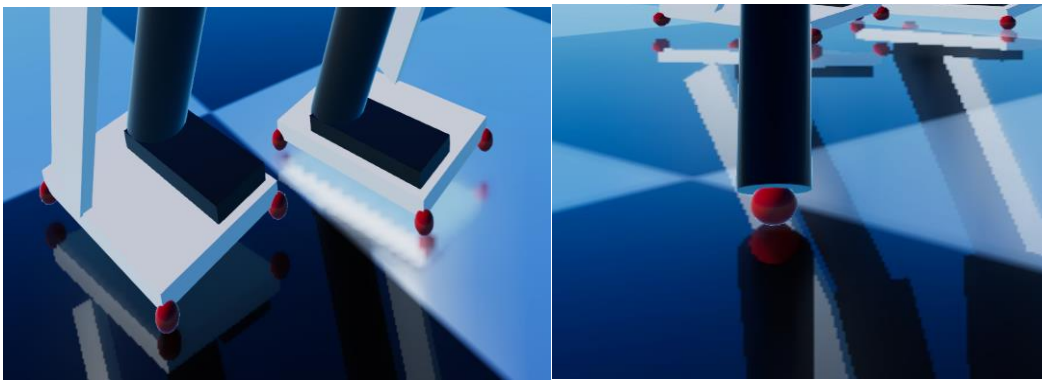
After having the human model which includes the necessary joints, a lower body exoskeleton has been modeled and attached to the human model along with the crutches that were fixed to left and right arm as shown in Figure 4.1. In order to sense whether the feet sole contact the ground or not, 4 spheres which are connected to the feet sole are added for each foot. Similarly, spheres are connected to the tips of the crutches in order to sense the reaction forces applied to the crutches. These artificial sensors are defined as prismatic joints with predefined stiffness and they behave as springs that enable force measurement. This

contact information is very useful in terms of the observation vector in reinforcement learning. The aforementioned contact spheres beneath the feet are shown in Figure 4.2(a), whereas the contact sphere at the tip of the crutch can be seen in Figure 4.2(b).



(a) (b) (c)

Figure 4.1. Exoskeleton-human system with crutches from different views. (a) Front view (b), Rear view (c), Transverse view.



(a) (b)

Figure 4.2. Contact spheres. (a) Feet contact, (b) Crutch contact.

4.1.5. Action and Observation Space in RaiSim

In a similar way described in Section 4.1, observation and action spaces have been created. Observation space consists of following variables; x, y, z positions of the root, rotation angles with respect to x, y, z axes in quaternion representation, rotation angles of ankle, knee, hip, shoulder and arm joints in sagittal plane, displacements in the prismatic joints that are used to attach the contact spheres and velocities of all these variables. Action space is defined as the joint torques to be applied on ankle, knee, hip, shoulder and arm joints. Note that the arm and shoulder joints are actuated by the human in a real world

environment, but it is necessary to simulate these actuations in order to obtain a complete gait with crutch usage and the arm actuations are assumed to be a part of the robotic exoskeleton. RaiSim successfully computes the variables required and communicates with Python which enables the usage of all these variables in order to build the observation and action spaces to be used within the scope of reinforcement learning.

4.1.6. Action and Observation Space in MuJoCo

In order to shape the observation space, a script originally written for simulating a humanoid model in MuJoCo[37] is used [38]. In this script, current positions of the center of mass of the exoskeleton-human system in x, y, z coordinates are excluded from the observation space. Thus, the observation consists of rotation angles with respect to x, y, z axes in quaternion representation, rotation angles of ankle, knee, hip, shoulder and arm joints in sagittal plane, displacements in the prismatic joints that are used to attach the contact spheres, velocities of all these variables, center of mass inertia and velocity and actuator forces.The center of mass inertia is defined as the body inertia based on CoM in MuJoCo. The same action space as in the RaiSim trials which consists of the joint torques is used for the MuJoCo trials as well.

4.2. Implemented RL Methods

As explained by many different studies in the Literature Review section, there are numerous state of the art reinforcement learning methods that may be used to develop a locomotion controller for a lower limb robotic exoskeleton. In this section some of these recent algorithms will be explained in detail. In order to develop a reinforcement learning based controller which utilizes neural networks and does the mapping between the robot states and required joint torques, Proximal Policy Optimization (PPO) [39] and Twin Delayed Deep Deterministic Policy Gradient (TD3) [33] methods will be utilized along with the help of RaiSim and MuJoCo simulation environments.

4.2.1. Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a policy gradient method that can be used to solve the problems with discrete or continuous action spaces. The neural networks are

utilized in an Actor-Critic architecture and a stochastic policy is trained in on-policy manner. PPO builds on the Vanilla Policy Gradient with advantage function which is expressed as

$$\nabla J(\theta) = \hat{E}_t\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\hat{A}_t\right], \tag{4.1}$$

where π_θ is the stochastic policy that outputs a probability distribution of actions, $\hat{A}_t$ is the estimation of advantage at time t and $\hat{E}_t$ is the expectation which is an average calculated over a finite number of batches.

Equation 2.9 is the gradient of the policy loss function which is

$$L_{PG}(\theta) = \hat{E}_t\left[\log \pi_\theta(a_t|s_t)\hat{A}_t\right]. \tag{4.2}$$

Even though it seems reasonable to update the network parameters constantly in a batch of collected experience , the resulting update rate of the parameters generally come out very aggressive which is called “destructively large policy updates”. One way to avoid this problem is to use Trust Region Policy Optimization [40]. Schulman et al. [40] proposed this method such that the update rate of the policy is constrained within a trustable region, so the destructively large update rates are avoided. First, a ratio is defined as

$$r_{(t)}(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, \tag{4.3}$$

in order to determine the similarity between the current policy and the previous one. For example, the ratio $r_{(t)}(\theta)$ will be greater than 1 if the particular action is more probable for the current policy when compared to the previous policy. Similarly, the ration will be between 0 and 1 if that action is less probable for the current policy when compared to the previous one. Trust Region Policy Optimization (TRPO) uses this ratio to shape an optimization problem defined as

$$\max_\theta \hat{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t\right] \tag{4.4}$$

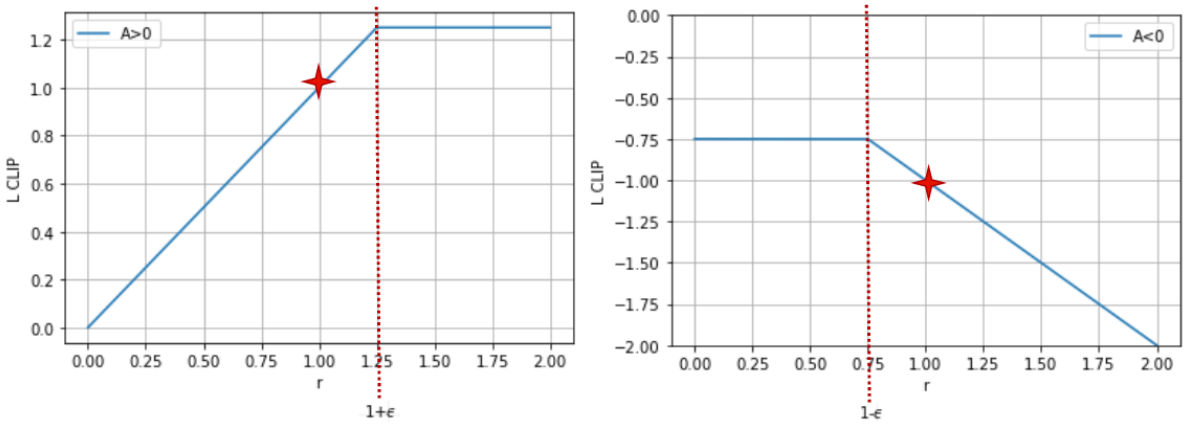
$$\text{subject to } \hat{E}_t\left[KL\left[\pi_{\theta_{old}}(.|s_t), \pi_\theta(.|s_t)\right]\right] \leq \delta, \tag{4.5}$$

where the objective function is called surrogate objective.

PPO combines the approaches the approaches described above and tries to strike a balance between the efficacy, use of implementation and robustness to hyper parameter tuning. The main contribution of PPO is Clipped Surrogate Objective defined as

$$L^{CLIP}(\theta) = \hat{E}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t\right)\right], \tag{4.6}$$

where the function "clip" clips the first argument $r_t(\theta)$ within a range $[1-\varepsilon, 1+\varepsilon]$. Here, the objective function takes the minimum value between the original policy gradient and the expectation is computed over this selected value.



(a) (b)

Figure 4.3. Behavior of L^{CLIP} with $\varepsilon = 0.25$ for (a) $A > 0$, (b) $A < 0$.

The behavior of L^{CLIP} is shown on Figure 4.3. When the advantage is positive the selected action has affected the outcome better than expected, but loss function is flattened if the likelihood of the action gets too high when compared to the previous policy. Similarly, the loss function is also clipped if the likelihood of the action is much less than the previous policy. The pseudocode of PPO-Clip algorithm is available in Table 4.4.

Table 4.4. Pseudocode of PPO-Clip algorithm

Algorithm PPO-Clip

1: Input: initial policy parameters $\boldsymbol{\theta_0}$, initial value function parameters $\boldsymbol{\varphi_0}$
2: **for** $\mathbf{k = 0, 1, 2, \ldots}$ **do**
3: Collect set of trajectories $\mathbf{D_k} = \{\boldsymbol{\tau_i}\}$ by running the policy $\boldsymbol{\pi(\theta_k)}$ in the environment
4: Compute rewards-to-go $\widehat{\mathbf{R}}_\mathbf{t}$
5: Compute advantages $\widehat{\mathbf{A}}_\mathbf{t}$ based on the current value function $\mathbf{V}_{\boldsymbol{\varphi_k}}$
6: Update the policy by maximizing the PPO-Clip objective by using slice length $\boldsymbol{L_a}$ after every $\boldsymbol{T_{horizon}}$ timestep:

Table 4.4. Pseudocode of PPO-Clip algorithm (cont.)

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} \min\left(r_{(t)}(\theta) A^{\pi\theta_k}(s_t, a_t), \mathrm{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon) A^{\pi\theta_k}(s_t, a_t)\right)$$

typically with gradient ascent with Adam optimizer

7: Fit value function by mean squared error and using slice length L_c::

$$\varphi_{k+1} = \arg\min_{\varphi} \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} \left(V_{\varphi}(s_t) - \hat{R}_t\right)^2$$

typically by gradient descent algorithm.

8: end for

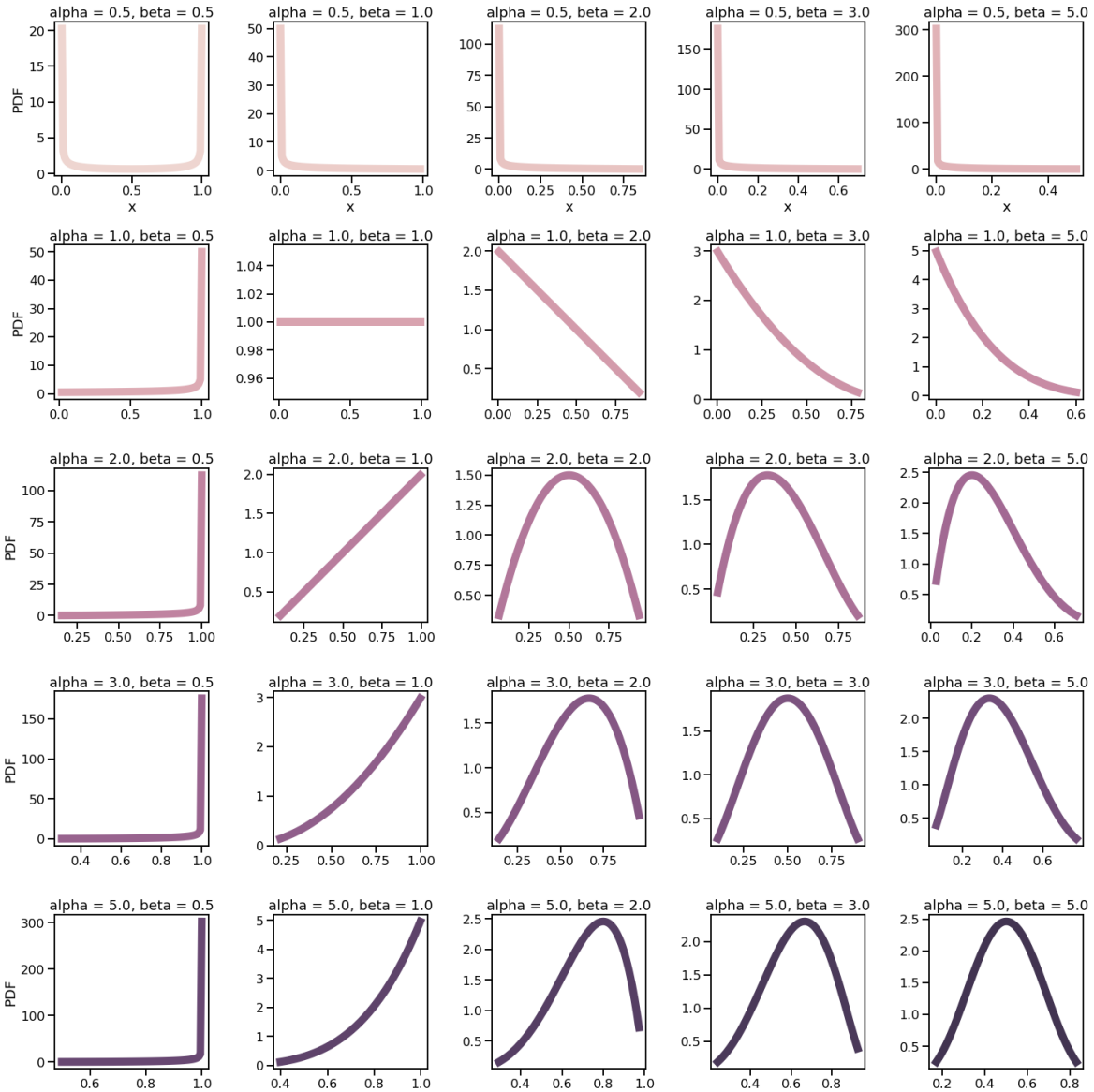


Figure 4.4. Beta distribution for different α and β values.

If the policy and value function shares some parameters, squared loss of critic should also be included in the overall loss. If the policy a value functions are different networks, then they should be updated independently. Then the final form of the loss function becomes

$$L_t^{CLIP+VF+S}(\theta) = \hat{E}_t\left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)\right], \tag{4.7}$$

where c_1 and c_2 are hyper parameters to tune, S is entropy bonus and $L_t^{VF}(\theta)$ is defined as

$$L_t^{VF}(\theta) = \left(V_\theta(s_t) - V_t^{targ}\right)^2. \tag{4.8}$$

The stochastic policy π_θ may be model by probabilistic distributions such as Gaussian distribution or beta distribution. Chou et al. [41] has proposed to use the beta distribution for continuous control tasks with deep reinforcement learning. Unlike Gaussian distribution which can give out of range action values, beta distribution supplies a bounded action space between 0 and 1. If beta distribution is used in the policy, the neural network outputs α and β values which characterizes the shape of beta distribution which can be seen in Figure 4.4.

Beta distribution is expressed as

$$h(x: \alpha, \beta) = \frac{\Gamma(\alpha\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}, \tag{4.9}$$

where the function Γ(Gamma) is defined as

$$\Gamma(n) = (n-1)!. \tag{4.10}$$

When acting in a stocasthic manner in training phase, the outputs of the policy network, which are α and β, used to generate the distribution and the action value is sampled from the distribution within the range [0,1]. If a deterministic action is required, the action is taken according to

$$\pi_\theta = \frac{\alpha}{\alpha + \beta}, \tag{4.11}$$

where α and β is the output of the policy network π_θ.

In order to get $\alpha, \beta > 1$, 1 is added to the output layer activations for both α, β. Petrazzini et al. [42] used beta distribution with PPO along with softplus activation at the output (with +1 offset) and stated that beta distribution outperforms Gaussian distribution

with PPO in terms of fast and stable convergence to an optimal solution. Another useful result that is reached in Petrazzini et al.'s study is that continuous control tasks where the action space is bounded and observation space is high dimensional are facilitated if the policy is described by beta distribution. Therefore, PPO with beta distribution is employed in the first part of this thesis where RaiSim is used as the physics simulator in order to provide a continuous control strategy for a lower limb robotic exoskeleton. In the later stages of this study, used PPO algorithm is also changed along with the transition from RaiSim to MuJoCo. During the MuJoCo trials, PPO algorithm implemented by OpenAI Baselines is used without any modification [43]. Therefore, Gaussian distribution is used in the later stage.

4.2.2. Twin Delayed Deep Deterministic Policy Gradient

Even though the value overestimation is studied in the literature, it is still a problem for most of the actor-critic family algorithms that is used for continuous control tasks. Overestimation bias is present in Q-learning method because of the noise in the value estimation. In a TD learning setup, this noise causes further problems as TD learning uses the noisy estimates to update which eventually leads to accumulation of noise. This accumulation of noise results in poor updates and being unable to converge to a desired policy. This overestimation caused by noisy estimates also poses a problem for deterministic policy gradient methods with continuous action and state spaces. Fujimoto et al. [33] proposed a solution to this problem by introducing a pair of independent critic networks. The idea of using multiple critic networks belongs to Van Hasselt et al. [44] Fujimoto et al. employed a few tricks such as target networks which is a frequently used concept in deep Q-learning, delayed policy updates and a novel regularization strategy. These updates are applied to Deep Deterministic Policy Gradient (DDPG) [24] which is one of the modern deep reinforcement learning techniques.

In Actor-Critic reinforcement learning methods, the policy network is updated based on the equation

$$\nabla_\theta J(\theta) = E_{s\sim p_\pi}\left[\nabla_a Q^\pi(s,a)|_{a=\pi(s)} \nabla_\theta \pi_\theta(s)\right], \tag{4.12}$$

where $Q^\pi(s,a)$ is the critic or value function. It is defined as

$$Q^{\pi}(s,a) = E_{s\sim p_{\pi},a\sim\pi}[R_t|s,a], \tag{4.13}$$

which is the expected return after π taking the action a in the state s. As explained earlier in 2.4.7., the value function $Q^{\pi}(s,a)$ can be learned by Bellman equation which establishes the bridge between values of state-action pairs belonging to subsequent time samples.

For large state spaces, neural networks can be used as function approximators in order to estimate the value function. In this case, the value function becomes $Q_{\varphi}(s,a)$ where φ is the parameter set of the neural network. Another network called frozen target network $Q_{\varphi'}(s,a)$ is used along with TD-learning. Hence a fixed objective value is obtained as

$$y = r + \gamma Q_{\varphi_{t+1}}(s_{t+1}, a_{t+1}), \tag{4.14}$$

where $a_{t+1}\sim\pi_{\theta_{t+1}(s_{t+1})}$. Since there are two critic networks in Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm, sometimes $Q_{\varphi_2}(s,\pi_{\theta}(s))$ will be greater than $Q_{\varphi_1}(s,\pi_{\theta}(s))$. This situation may introduce problems, because Q_{φ_1}will overestimate values most of the time. The larger Q_{φ_2}value estimate will increase the degree of overestimation. Fujimoto et al. suggested to use the minimum value between Q_{φ_1} and Q_{φ_2} for calculating the target update such that

$$y_1 = r + \gamma \min_{i=1,2} Q_{t+1,\varphi_i}(s_{t+1}, \pi_{\theta}(s_{t+1})), \tag{4.15}$$

where min function picks the smaller value between Q_{t+1,φ_1} and Q_{t+1,φ_2}. Even though the equation 4.15 may introduce underestimation, it is not more dangerous than overestimation as it will not propagate throughout the update sequence. If there is only one actor network the same target $y_2 = y_1$ will be used for updating $Q_{\varphi 2}$ and this option is more preferable if computation cost is important.

Apart from the variance introduced by the overestimation, variance itself is addressed in TD3 algorithm as well, since high variance estimates results in noisy gradients. Using these noisy gradients in policy update reduces learning speed according to Sutton et al. [6] and may cause poor performance. Variance causes the estimation error to grow as training progresses. By using a stable target network, convergence behavior can be improved. Fujimoto et al.[33] states that the high variance in policy updates causes divergence if there is not any target network is used. Therefore, target networks can be used to decrease the

estimation error whereas policy updates on the states with high error may cause the policy to diverge. One of the main ideas of TD3 algorithm is to update value network more frequently than the policy network. This approach ensures that the error is decreased before optimizing the policy network.

Table 4.5. Pseudocode of TD3

Algorithm TD3

1: Input: initial policy parameters $\boldsymbol{\theta_0}$, initial critic network parameters $\boldsymbol{\varphi_1}$and $\boldsymbol{\varphi_2}$, initial target networks $\boldsymbol{\varphi'_1 \leftarrow \varphi_1}$, $\boldsymbol{\varphi'_2 \leftarrow \varphi_2}$ and $\boldsymbol{\theta' \leftarrow \theta}$, initial empty replay buffer $\mathbb{B}$
2: **for** $\mathbf{t = 1, 2, \ldots}$ **T do**
3: Select action with exploration noise $\boldsymbol{a_t = \pi_\theta(s_t) + \epsilon}$ where $\boldsymbol{\epsilon = \beta \mathcal{N}(0, \sigma)}$ and $\boldsymbol{\beta}$ is exploration coefficient
4: Update exploration coefficient: $\boldsymbol{\beta \leftarrow \beta * \beta_{decay}}$
4: Implement the action $\boldsymbol{a_t}$ on the environment
5: Observe $\boldsymbol{s_{t+1}}$, $\boldsymbol{r_t}$ and store transition information $\boldsymbol{(s_t, a_t, r_t, s_{t+1})}$ in $\mathbb{B}$
6: Sample N number of random transitions from $\mathbb{B}$
7: $\boldsymbol{\tilde{a} \leftarrow \pi_{\theta'}(s_{t+1}) + \epsilon}$, where $\boldsymbol{\epsilon = clip(\mathcal{N}(0, \tilde{\sigma}), -c, c)}$
$\boldsymbol{y \leftarrow r + \gamma \min_{i=1,2} Q_{\varphi'_i}(s_{t+1}, \tilde{a})}$
8: Update critics $\boldsymbol{\varphi_i \leftarrow argmin_{\varphi_i} N^{-1} \sum \left(y - Q_{\varphi_i}(s_t, a_t)\right)^2}$
9: **if** t mod policy_delay **then**
Compute policy gradients for $\boldsymbol{\theta}$ and update policy:
$\boldsymbol{\nabla_\theta J(\theta) = N^{-1} \sum \nabla_a Q_{\varphi_1}(s_t, a_t)|_{a=\pi_\theta(s_t)} \nabla_\theta \pi_\theta(s_t)}$
Update the target networks:
$\boldsymbol{\varphi'_i \leftarrow \tau \varphi_i + (1 - \tau)\varphi'_i}$
$\boldsymbol{\theta' \leftarrow \tau \theta + (1 - \tau)\theta'}$
10: **end if**
11**: end for**

Another contribution that is introduced by Fujimoto et al. [33] is target policy smoothing regularization. Deterministic policies often induce an inaccuracy caused by function approximation errors. The resulting high variance should be regularized. The approach introduced in TD3 ensures that similar actions have similar value. Fitting the value within an area around target action

$$y = r + E_\epsilon\left[Q_{t+1,\varphi}\left(s_{t+1}, \pi_{t+1,\theta}(s_{t+1}) + \epsilon\right)\right], \tag{4.16}$$

smooths the value estimate, where ϵ is the noise parameter clipped between an upper and lower limit pair. Using Equation 4.16 over mini batches and averaging reduces the variance.

TD3 is an off-policy algorithm that trains a deterministic policy. In order to facilitate the exploration, some amount of noise is added to the actions generated by the policy. The magnitude of the noise may be reduced gradually as the training progresses to have better quality of training data. Overall algorithm for TD is available in Table 4.5.

4.3. Reward Shaping

4.3.1. Reward Shaping for RaiSim

During the trials made on RaiSim, the reward function is designed in order to make the human-exoskeleton system walk in the straight direction without diverging through left or right, without tilting the body too much and without falling. The terms that shape the reward function are r_{walk}, $r_{walk\_straight}$, r_{dont_fall}, $r_{orientation}$ and r_{action}. During the first trials, simple expressions are used to shape the reward, but after failing to converge to a useful policy, exponential functions are adopted. Due to the shape of the exponential functions, undesired actions are penalized more when compared to linear functions. The position of the exoskeleton is based on the root attached to the middle section of the exoskeleton model as seen in Figure 4.5.

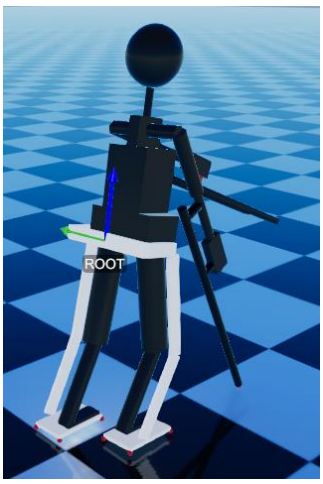


Figure 4.5. Position of the root attached to the hip section of the exoskeleton.

Walking reward encourages the human-exoskeleton system to translate its root position in the forward direction. It is defined as

$$r_{walk} = e^{-50\left(100\left(\dot{p}_x - \dot{p}_{x,des}\right)\right)^2}, \tag{4.17}$$

where $\dot{p}_x$ is the derivative of the root position in x coordinate based on the x coordinate of the previous state, whereas $\dot{p}_{x,des}$ is the desired forward velocity and it is set as 0.0005 m/s.

Walk straight reward discourages the system to walk to left or right side. It is defined as

$$r_{walk\_straight} = e^{-50\dot{p_y}^2}, \tag{4.18}$$

where $\dot{p_y}$ is the difference between the position in y coordinate of current state and position in y coordinate of the previous state.

Do not fall reward encourages the human-exoskeleton system to keep the height of the root close to the desired height. It is defined as

$$r_{dont\_fall} = e^{-50\left((p_z - p_{z,des})\right)^2}, \tag{4.19}$$

where p_z is the position in z coordinate of current state, whereas $p_{z,des}$ is the desired root height and it is set as 0.75 m.

Orientation reward discourages the human-exoskeleton system to tilt the body forward too much. It is defined as

$$r_{orientation} = e^{-20{a_x}^2}, \tag{4.20}$$

where a_z is the angle of the body to the forward direction.

Action reward discourages the exoskeleton to generate exaggerated actions. It is defined as

$$r_{action} = e^{-\frac{\|action\|}{50}}, \tag{4.21}$$

where $action$ is the joint torques generated by the neural network.

The sub-rewards described above are combined and an overall reward function is shaped as

$$r = w_1 r_{walk} + w_2 r_{walk\_straight} + w_3 r_{dont\_fall} + w_4 r_{orientation} + w_5 r_{action}, \tag{4.22}$$

where $w_1, w_2, w_3, w_4$and w_5 are 0.8, 0.05, 0.05, 0.05, 0.05 respectively. This reward function is used along with the PPO and TD3 algorithms on RaiSim simulation environment in order to obtain a natural looking crutched gait.

4.3.2. Reward Shaping for MuJoCo

During the trials made on MuJoCo, the reward function is shaped in a similar way as done during the trials on RaiSim. On the contrary to the trials made with RaiSim, the termination criteria for episode length has not been used. Expressions which are mathematically simpler worked, so exponential functions are not used as done during the trials on RaiSim. After observing that the agent has learnt to walk as desired, additional terms to optimize the gait are added. The terms that shape the reward function are $r_{walk}, r_{walk\_straight}, r_{dont\_fall}, r_{action}, r_{orientation}, r_{flat\_contact}, r_{crutch\_reaction\_force}, r_{hip\_angle}$ and $r_{ensure\_crutch\_contact}$.

As in the trials made on RaiSim, walking reward encourages the human-exoskeleton system to translate its root position in the forward direction. It is defined as

$$r_{walk} = 1.25(2 - 25(\dot{p}_x - \dot{p}_{x,des}))^2, \tag{4.23}$$

where $\dot{p}_x$ is the derivative of the CoM position in x coordinate of current state based on the CoM position in x coordinate of the previous state, whereas $\dot{p}_{x,des}$ is the desired forward velocity and it is set as 0.25 m/s.

Walk linear reward discourages the system to walk to left or right side. It is defined as

$$r_{walk\_straight} = -|p_y|, \tag{4.24}$$

where p_y is the position in y coordinate.

Do not fall reward encourages the human-exoskeleton system to keep the height of the root within an acceptable height range. It is defined as

$$r_{dont\_fall} = \begin{cases} 5\ , & if\ min_z < p_z < max_z \\ 0\ , & else \end{cases}, \tag{4.25}$$

where p_z is the position in z coordinate of current state, min_z is 0.65, max_z is 3, whereas $p_{z,des}$ is the desired root height and it is set as 0.75 m.

Action reward discourages the exoskeleton to generate exaggerated actions. It is defined as

$$r_{action} = -3\|action\|, \tag{4.26}$$

where $action$ is the joint torques generated by the neural network.

Orientation reward discourages the human-exoskeleton system to tilt the body forward too much. It is defined as

$$r_{orientation} = -8(a_z - 0.35)^2, \tag{4.27}$$

where a_z is the angle of the body to the forward direction. 0.35 rad which corresponds to 20 degrees is selected based on the observations done on exoskeleton users.

Flat contact reward encourages the exoskeleton to keep the feet parallel to the ground in order to avoid tripping. It is defined as

$$r_{flat\_contact} = -10(r_{foot\_flat} + l_{foot\_flat})^2, \tag{4.28}$$

where

$$r_{foot\_flat} = (a_z + a_{right\_hip} + a_{right\_knee} + a_{right\_ankle})^2, \tag{4.29}$$

$$l_{foot\_flat} = (a_z + a_{left\_hip} + a_{left\_knee} + a_{left\_ankle})^2, \tag{4.30}$$

and a_{x_y} is the angle of related foot-joint combination.

Crutch reaction force reward discourages the human-exoskeleton system to have too much ground reaction force on the crutches. It is defined as

$$r_{crutch\_reaction\_force} = -w_{crutch\_reaction\_force}\left({d_{crutch_r}}^2 + {d_{crutch_l}}^2 \right), \tag{4.31}$$

where d_{crutch_r} and d_{crutch_l} are the displacements on the tip of the right crutch and left crutch, respectively. These displacements represent the ground reaction forces on the crutch tips.

Hip angle reward discourages the exoskeleton to extend the hip angle backwards with respect to the upper body as the hip angle always stays positive during a natural looking lower body exoskeleton gait. This reward term is defined as

$$r_{hip\_angle} = \begin{cases} -2\,, & if\ a_{hip\_r} < 0\ and\ a_{hip\_l} < 0 \\ 0\,, & else \end{cases}, \tag{4.32}$$

where a_{hip_r} and a_{hip_l} are the angle of right hip and left hip, respectively.

Ensure crutch contact reward discourages the human-exoskeleton system to stop the contact of the two crutches on the ground at the same time. It is defined as

$$r_{ensure\_crutch\_contact} = \begin{cases} -2, & if\ d_{crutch\_r} < 0.003\ and\ d_{crutch\_r} < 0.003 \\ 0, & else \end{cases}, \tag{4.33}$$

where d_{crutch_r} and d_{crutch_l} are the displacements on the tip of the right crutch and left crutch, respectively. The value 0.003 is selected as the threshold to infer the contact information with the ground.

The sub-rewards described above are combined and an overall reward function is shaped as

$$r = r_{walk} + r_{walk\_straight} + r_{dont\_fall} + r_{action} + r_{orientation} + \tag{4.34}$$

$$r_{flat\_contact} + r_{crutch\_reaction\_force} + r_{hip\_angle} + r_{ensure\_crutch\_contact}.$$

4.4. Overall Implementation

4.4.1. Implementation for RaiSim

By using the reward scheme shown in Equation 4.22, TD3 algorithm has been implemented by using the RaiSim as the simulator. 6 different training runs have been made with different parameters sets and a maximum episode length of 7000 timesteps. The hyperparameters in Table 4.6 are kept constant throughout the TD3 training runs. The other hyperparameters are varied across the different training runs. These varied parameters are available in Table 4.7. Network properties for actor and critic networks are shown in Table 4.8.

Table 4.6. Varied hyperparameters for TD3 method

Parameter Set	γ	β_{decay}	Actor LR	Critic LR
1	0.9900	0.999	1e-4	1e-4
2	0.9996	0.9996	1e-4	1e-4
3	0.9900	0.9996	1e-4	1e-4
4	0.9900	0.9996	5e-4	5e-4
5	0.9900	0.9996	5e-5	5e-5
6	0.9999	0.9996	5e-5	5e-5

Table 4.7. Constant hyperparameters for TD3 method

	N	c	τ	σ	β	Network Width	policy_delay
Parameters	256	160	0.005	0.25	0.25	200	2

Table 4.8. Network properties for TD3 method

Network	Hidden Layer Number	Hidden Layer Width	Input Layer Activation	Hidden Layer Activation	Output Layer Activation
Actor	1	200	Tanh	Tanh	Tanh
Critic 1	1	200	ReLu	ReLu	Linear
Critic 2	1	200	ReLu	ReLu	Linear

After being unable to reach a satisfactory result with TD3, the algorithm is changed. The same reward function is used along with the PPO algorithm and RaiSim. In a similar manner to TD3 implementation, different parameters sets are tried with a maximum episode length of 7000 timesteps. The actor and critic network architecture is built by using the network parameters in Table 4.11. The hyperparameters in Table 4.10 are kept constant whereas the hyperparameters in Table 4.9 are varied as shown in the table.

Table 4.9. Varied hyperparameters for PPO method

Parameter Set	γ	$T_{horizon}$	Actor LR	Critic LR
1	0.9900	2048	2e-4	2e-4
2	0.9900	2048	1e-4	1e-4
3	0.9996	2048	1e-4	1e-4
4	0.9996	2048	5e-5	5e-5
5	0.9999	2048	5e-5	5e-5
6	0.9999	7000	5e-5	5e-5

Table 4.10. Constant hyperparameters for PPO method

	ε	K_{epochs}	L_a	L_c	**Entropy Coefficient**	**Network Width**	**Entropy Coefficient Decay**	λ
Parameters	0.2	10	64	64	1e-3	200	0.99	0.95

Table 4.11. Network properties for PPO method

Network	**Hidden Layer Number**	**Hidden Layer Width**	**Input Layer Activation**	**Hidden Layer Activation**	**Output Layer Activation**
Actor	1	200	Tanh	Tanh	Softplus
Critic	1	200	Tanh	Tanh	Linear

4.4.2. Implementation for MuJoCo

Since the training runs have not yielded successful results with RaiSim, it has been decided to change the physics simulator. URDF file that models the system is transformed into XML format and MuJoCo is integrated to the training loop. As described in Section 4.2.1, PPO implementation of OpenAI Baselines is used together with MuJoCo. Equation 4.34 is used as the reward function. The properties of the used networks are tabulated in Table 4.14. PPO hyperparameters are kept constant as shown in Table 4.13 throughout the training runs in MuJoCo, because the trained RL agents exhibit desired behaviors for most of the cases.

Table 4.12. Varied weights for PPO method on MuJoCo

RL Agent No	$w_{crutch\_reaction\_force}$
1	40000
2	30000
3	20000
4	10000

The term $w_{crutch\_reaction\_force}$ is varied in order to see the effect of different weights on the GRF's on the crutch tips. These varied weights are tabulated in Table 4.12. Each different $w_{crutch\_reaction\_force}$ value is tried with 5 different seeds and 4 different RL agents are trained for each $w_{crutch\_reaction\_force}$ value. However, the training runs are designed

without any termination criteria about average reward. In order to be able to plot the variances for different runs with different $w_{crutch\_reaction\_force}$ parameters, all average reward trajectories should be of equal length. Therefore, all training runs are conducted for 8000 iterations.

Table 4.13. Constant hyperparameters for PPO method on MuJoCo

ε	K_{epochs}	L_a	L_c	**Entropy Coefficient**	**Net Width**	λ	γ	$T_{horizon}$	**Actor LR**	**Critic LR**
0.02	10	64	64	0	64	0.95	0.99	2048	1e-4	1e-4

Table 4.14. Network properties for PPO method on MuJoCo

Network	**Hidden Layer Number**	**Hidden Layer Width**	**Input Layer Activation**	**Hidden Layer Activation**	**Output Layer Activation**
Actor	2	64	Tanh	Tanh	Linear
Critic	2	64	Tanh	Tanh	Linear

After developing RL agents based on a reward function with GRF minimization term, baseline RL agents have been also developed in order to assess whether the GRF's are minimized or not. The same PPO implementation and exoskeleton-human system model is used. The reward function used for baseline is also the same as the other training runs on MuJoCo, but without any term for GRF minimization. The network properties and the PPO hyperparameters are also the same and these training settings for baseline have also been tried with 5 different seeds.

5. RESULTS

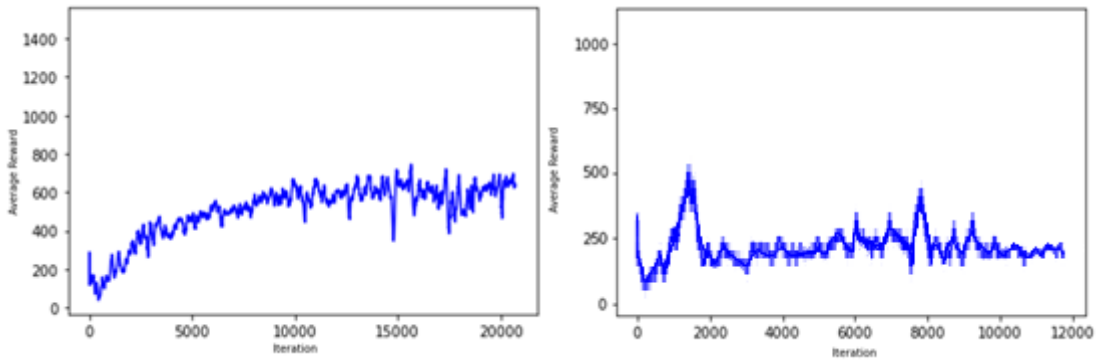
The algorithms and reward function described in Section 4 are used together in order to train the human-exoskeleton system with crutches. The average values of the last 100 episodic rewards are calculated (by moving average filter) and average reward versus episode plots are generated in the next subheadings. In section 4.4.1, six different trials with different parameters for each algorithm were done by using RaiSim. After each training run, the resulting policy is tried 10 times with an episode length of 7000 timestep and the average rewards for these 10 trials are also tabulated.

In Section 5.3, the results of PPO implementation of OpenAI Baselines with different $w_{crutch\_reaction\_force}$ parameter and MuJoCo are presented. Based on the observations during the rendering of the training process and resulting RL agents, an intuition about the reward amounts has been established. If the exoskeleton-human system walks in forward direction by using crutches in a reasonable, sequential and symmetric way without falling the reward amount is observed to be around 2000. Thus this value can be used as the success criteria for the training.

In Section 5.4., a comparison between the resulting RL agents and the baseline agent is made based on the desired gait characteristics such as GRF's on crutch tips and velocity tracking error.

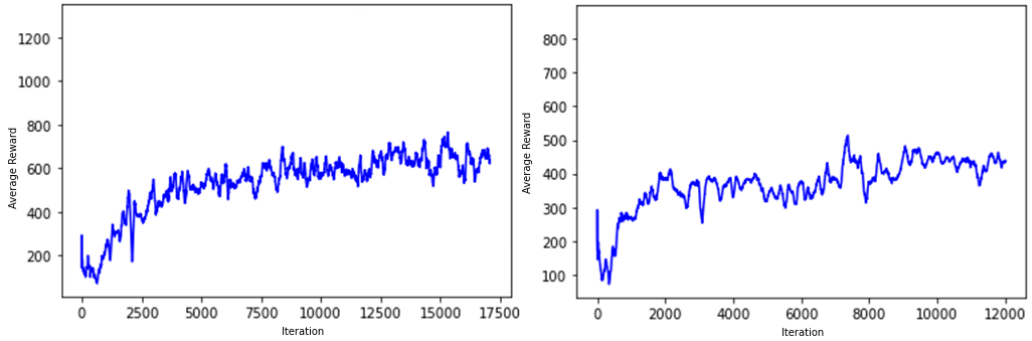
5.1. Results for TD3 with RaiSim

The average reward amounts of six different trials made with TD3 algorithm and RaiSim simulator have not reached a sufficient reward amount as the maximum reward has been designed as 7000 in the training setup, as seen in Figure 5.1, 5.2, and 5.3. Even though, there are some upward slopes in the average reward, the resulting RL agents have been observed to be unsuccessful in terms of moving forward, staying upright and other gait characteristics as can be seen from the low average test rewards in Table 5.1.



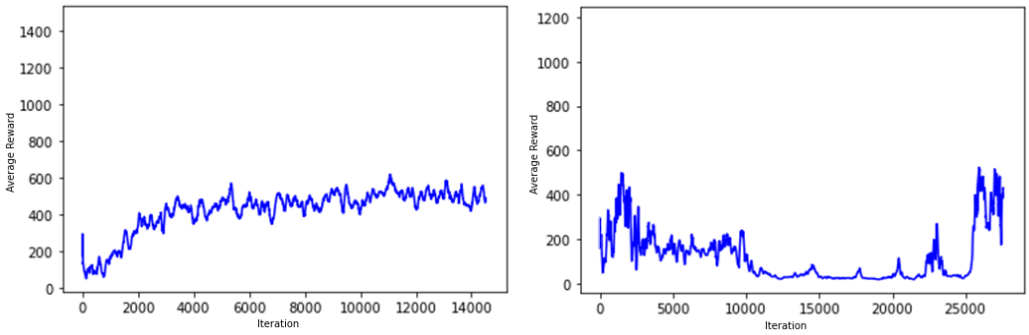
a) TD3 parameter set 1 b) TD3 parameter set 2

Figure 5.1. (a) Result of TD3 parameter set 1, (b) The result of TD3 parameter set 2.



a) Parameter set 3 b) Parameter set 4

Figure 5.2. (a) Result of TD3 parameter set 3, (b) The result of TD3 parameter set 4.



a) Parameter set 5 b) Parameter set 6

Figure 5.3. (a) Result of TD3 parameter set 5, (b) The result of TD3 parameter set 6.

Table 5.1. Test results for TD3 method.

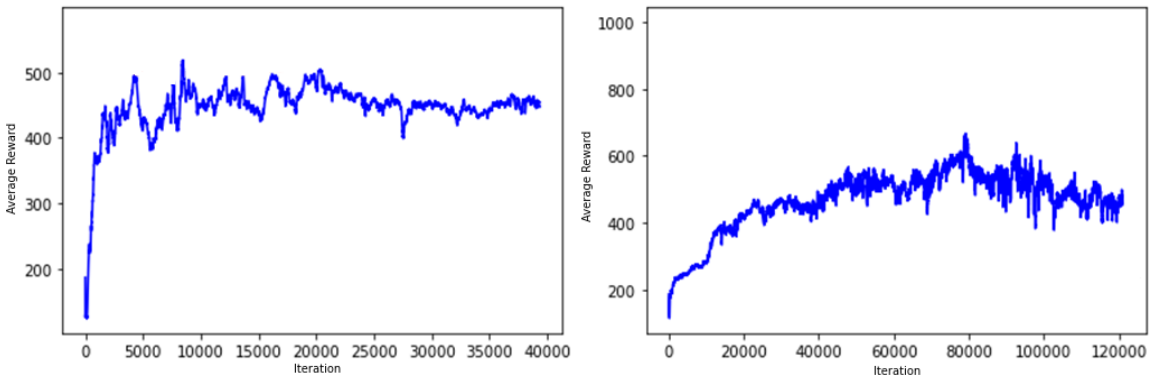
Trial No	Average Test Reward
1	392.66
2	145.20
3	654.05

Table 5.2. Test results for TD3 method (cont.)

4	441.39
5	752.63
6	62.72

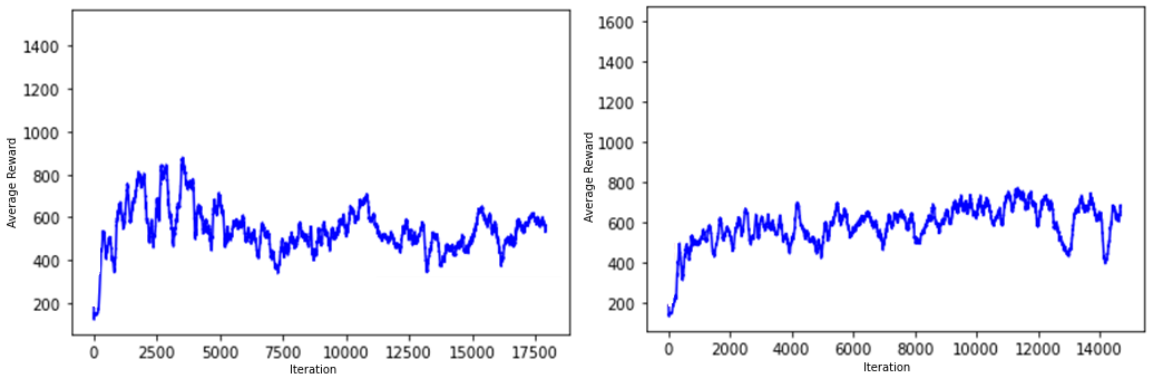
5.2. Results for PPO with RaiSim

The average reward amounts of six different training runs made with PPO algorithm and RaiSim simulator also have not reached a sufficient reward amount, as seen in Figure 5.4, 5.5, and 5.6. Even though, there are some upward slopes in the average reward just as in Section 5.1, the resulting RL agents also have been observed to be unsuccessful in terms of moving forward and other gait characteristics. However, the RL agent trained with parameter set 6 surpassed the average reward value 3000, which is considerably higher than the other trials. This RL agent is also tested and the average test reward has been calculated as 3210.54, but it has been observed that the desired forward displacement is not achieved and the generated motions are redundant. The average test rewards for other trials are also tabulated in Table 5.2.



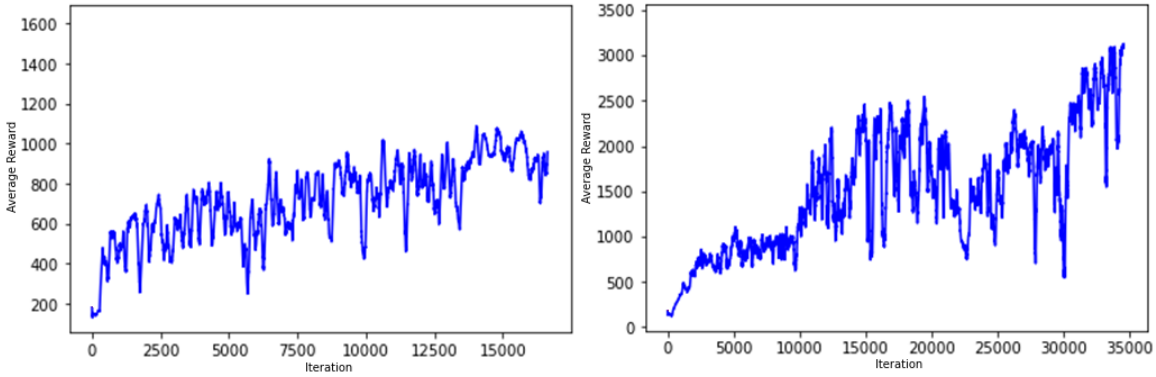
a) Parameter set 1 b) Parameter set 2

Figure 5.4. (a) Result of PPO parameter set 1, (b) The result of PPO parameter set 2.



a) Parameter set 3 b) Parameter set 4

Figure 5.5. (a) Result of PPO parameter set 3, (b) The result of PPO parameter set 4.



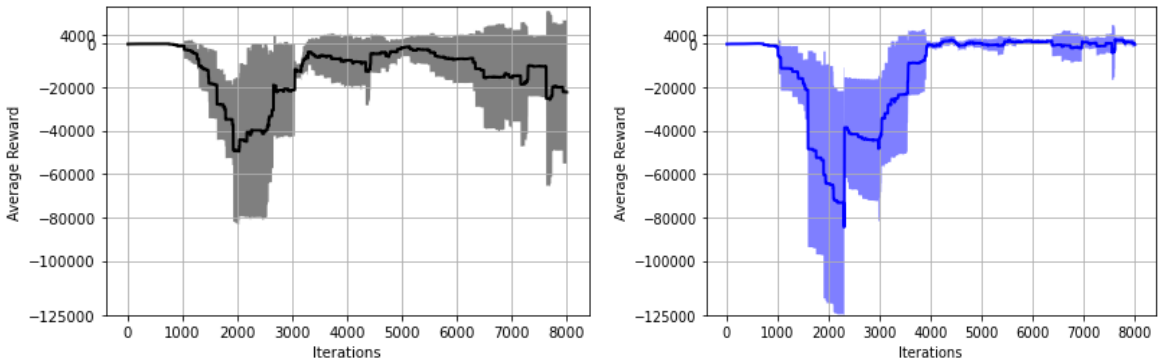
a) Parameter set 5 b) Parameter set 6

Figure 5.6. (a) Result of PPO parameter set 5, (b) The result of PPO parameter set 6.

Table 5.3. Test results for PPO method.

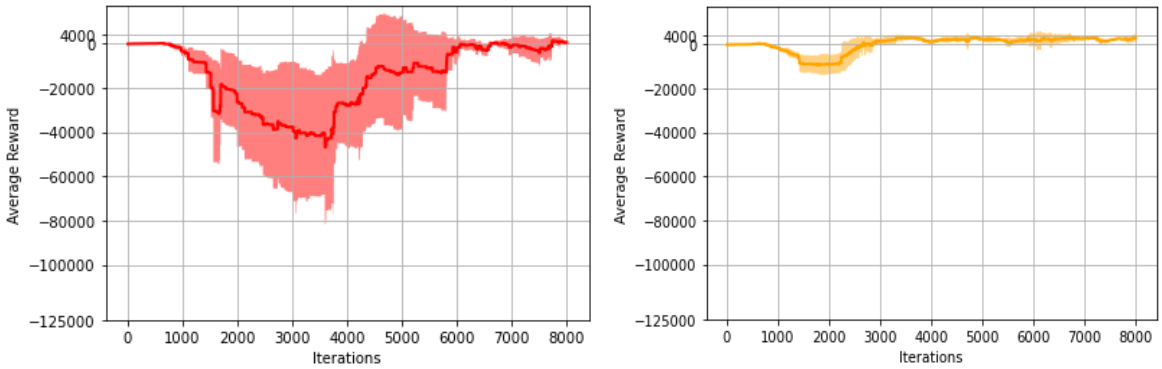
Trial No	Average Test Reward
1	447.67
2	494.60
3	502.42
4	703.02
5	1049.54
6	3210.54

5.3. Results for PPO with MuJoCo



a) RL agent 1 b) RL agent 2

Figure 5.7. (a) Average reward plot for RL agent 1 with $w_{crutch\_reaction\_force} = 40000$, (b) Average reward plot for RL agent 2 with $w_{crutch\_reaction\_force} = 30000$.



a) RL agent 3 b) RL agent 4

Figure 5.8. (a) Average reward plot for RL agent 3 with $w_{crutch\_reaction\_force} = 20000$, (b) Average reward plot for RL agent 4 with $w_{crutch\_reaction\_force} = 10000$.

In this section, the results of PPO implementation with MuJoCo is presented. As mentioned in the beginning of Results section, 2000 is a reward value that represents a successful training run. Even if a parameter set has not been successful, the same hyperparameters are tried multiple times, considering the stochastic nature of the PPO algorithm. This approach actually paid off and 4 successful agents have been trained after failing the first time. Even though the average reward drops drastically after starting from a value around 100, it generally reaches the value 4000 in all trials. As can be seen from the Figure 5.7 and 5.8, there is a huge variance across all the training runs and average reward does not converge to some specific value. Even if the average reward surpasses 4000, it drops significantly and recovers again. This problem can be overcome by setting a specific

training termination criteria for the average reward, but the average reward trajectories will not be of equal length, which makes it impossible to draw the variance plots. For this reason, all trainings are run for 8000 iterations. Additionally, the plot for the average and variance of the reward values obtained from baseline training runs with 5 different seeds is available in Figure 5.9.

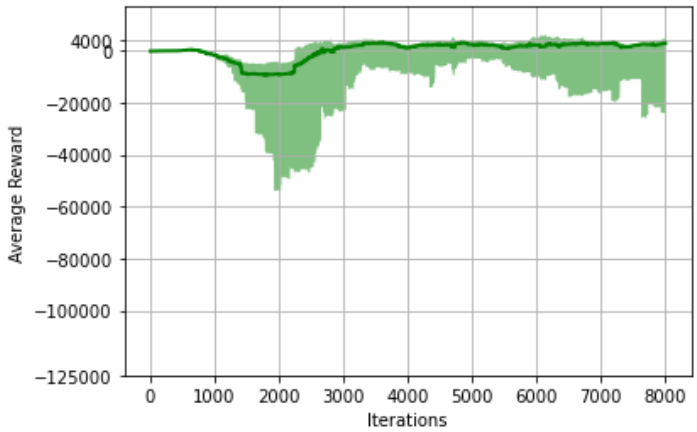
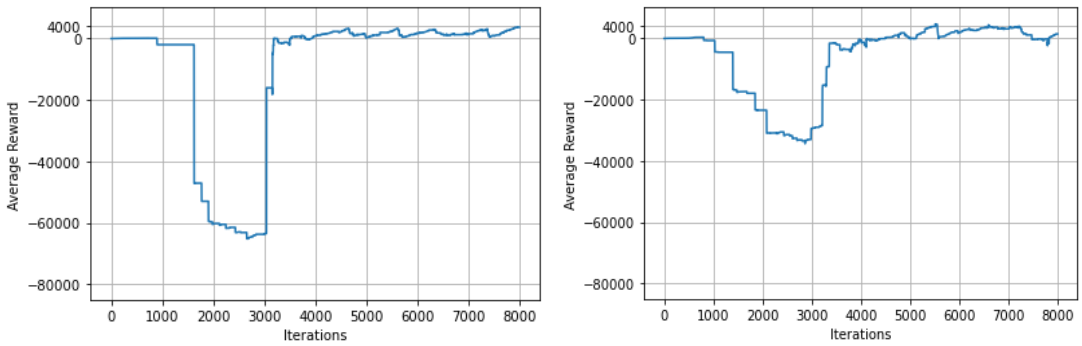


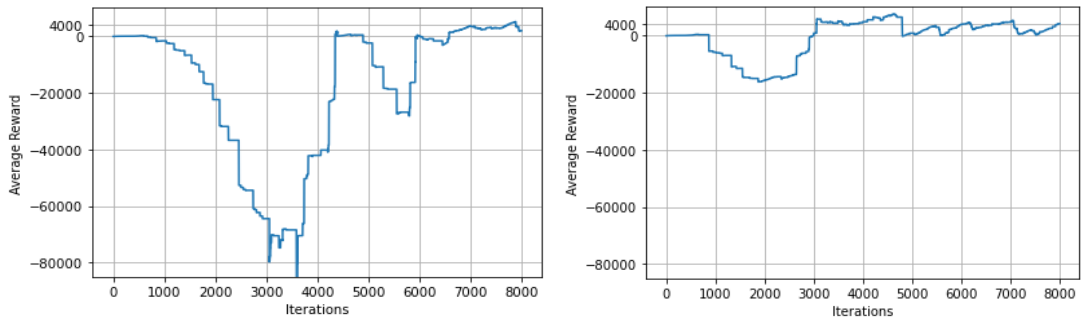
Figure 5.9. Average reward plot for baseline agent training sessions with 5 different seeds.

Among the 5 different trials made for different $w_{crutch\_reaction\_force}$ values, the best training results are given separately in Figure 5.10 and 5.11 for each different $w_{crutch\_reaction\_force}$ value .



a) RL agent 1 b) RL agent 2

Figure 5.10. (a) Average reward plot of the best trial with $w_{crutch\_reaction\_force} = 40000$, (b) Average reward plot of the best trial with $w_{crutch\_reaction\_force} = 30000$.



a) RL agent 3 b) RL agent 4

Figure 5.11. (a) Average reward plot for the best trial with $w_{crutch\_reaction\_force} = 20000$, (b) Average reward plot for the best trial with $w_{crutch\_reaction\_force} = 10000$.

5.4. Comparison for PPO with MuJoCo

The best RL agents selected for each different $w_{crutch\_reaction\_force}$ parameters are tested by simulating the agents for 2000 timesteps. Performance metrics such as GRF's exerted on the crutch tips, velocity tracking error, orientation tracking error are integrated over the 2000 timesteps and average values for this performance variables are plotted and tabulated in Table 5.3.

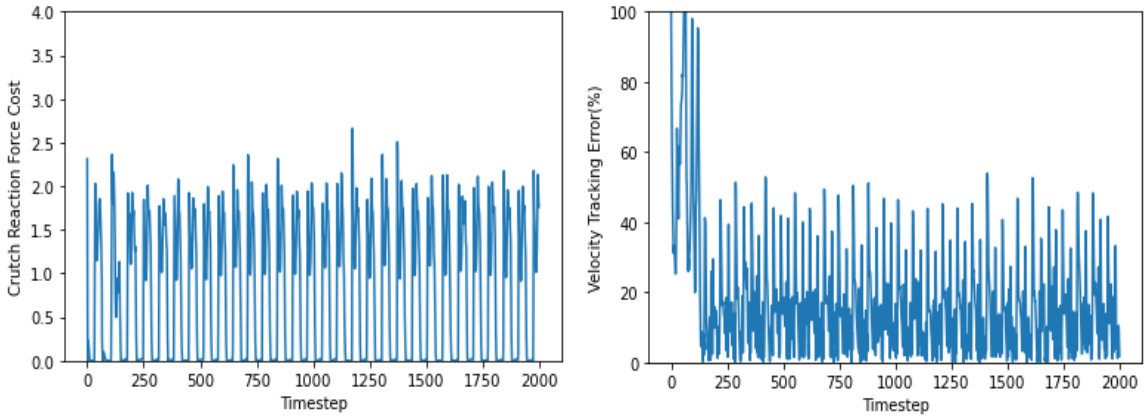
Table 5.4. Comparison of resulting RL agents with a baseline RL agent.

RL Agent	**Average Crutch Reaction Cost**	**Average Absolute Velocity Error (%)**	**Average Absolute Orientation Error (%)**	**Average Absolute Lateral Displacement (m)**
Baseline-1	0.911	20.51 %	17.40 %	0.03
Baseline-2	0.769	17.10 %	16.71 %	0.03
Baseline-3	0.797	17.81 %	18.61 %	0.08
1	0.501	11.11%	34.96 %	0.07
2	0.588	17.26 %	12.19 %	0.17
3	0.619	16.78 %	22.51 %	0.07
4	0.665	15.58 %	23.82 %	0.05

The average crutch reaction cost has been calculated based on the Equation 4.31 with $w_{crutch\_reaction\_force}$ = 40000 for all RL agents in each timestep. The average of these values from each timestep have been calculated and given as average crutch reaction cost in Table 5.3. Note that $w_{crutch\_reaction\_force}$ selected as 40000 only during the testing in order to compare the GRFs of different RL agents, because we are interested in comparing the

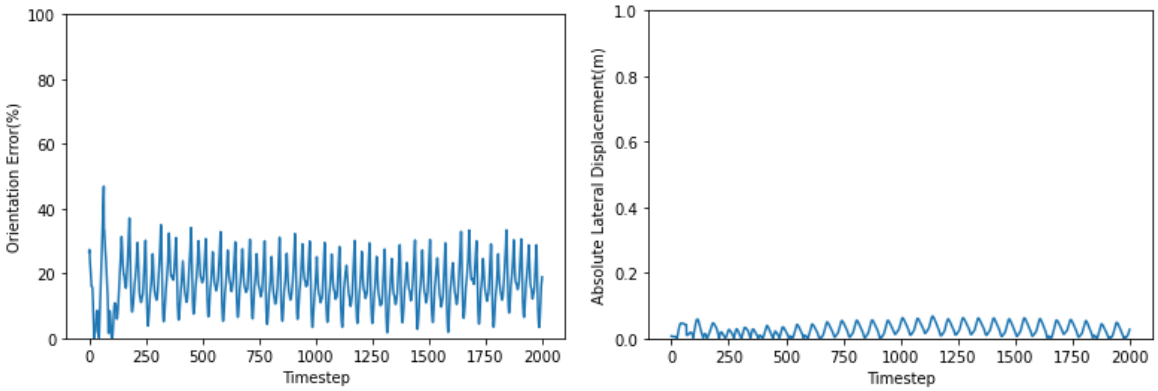
term ($d_{crutch\_r}{}^2 + d_{crutch\_l}{}^2$) in Equation 4.31 for different RL agents. In order to calculate the average absolute velocity error in Table 5.3, center of mass velocities have been subtracted from the desired velocity, which was defined as 0.25 m/s for each timestep, and absolute values of these subtractions have been summed up over 2000 timesteps. Then these integrated values have been divided by 2000 to calculate the average absolute velocity error. The average absolute orientation error has also been calculated by the same method as used to calculate the average absolute velocity error. In order to calculate the average absolute lateral displacement, the absolute values of lateral displacements in each timestep are summed up and the sum is divided by 2000.

In the following parts, the performance plots for the best ones of the trained agents are presented. Baseline-2 is selected as the best baseline as it generates less crutch reaction cost, average absolute velocity error, average absolute orientation error and average absolute lateral displacement error compared to the other baseline RL agents. Baseline-2 is simulated for 2000 timesteps and GRF cost is calculated based on Equation 3.31 and with $w_{crutch\_reaction\_force} = 40000$ as seen in Figure 5.12 (a). The percent velocity/orientation tracking error and lateral displacement are also plotted in Figure 5.12 (b), Figure 5.13 (a) and (b), respectively.



a) Baseline - GRF b) Baseline - Velocity error

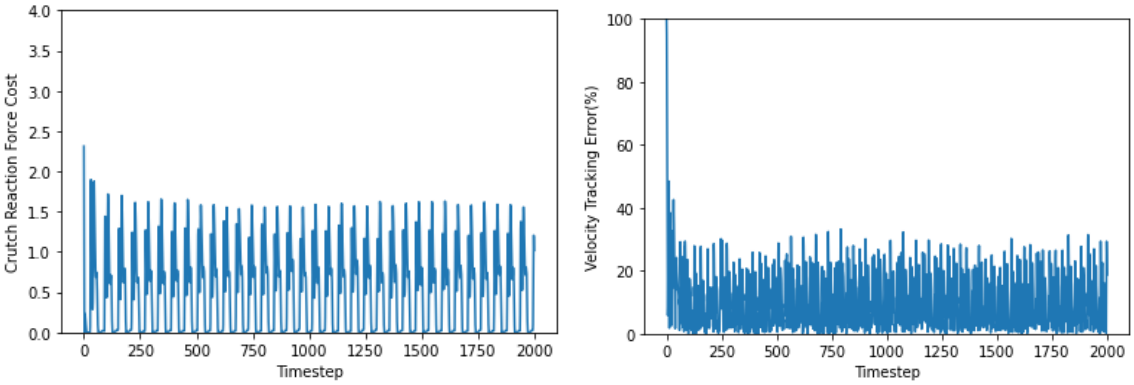
Figure 5.12. (a) GRF on crutch cost for PPO Baseline, (b) Percent velocity tracking error for PPO Baseline.



a) Baseline - Orientation error b) Baseline - Lateral displacement

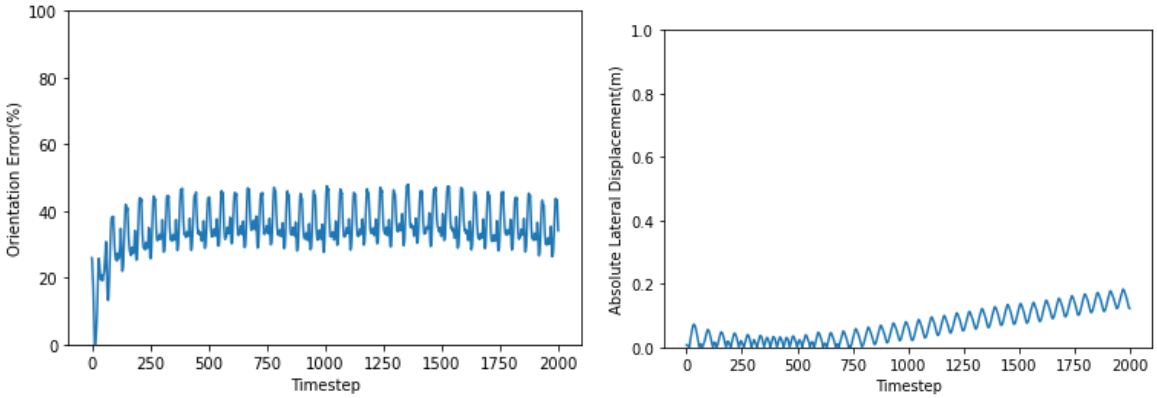
Figure 5.13. (a) Percent orientation error for PPO Baseline, (b) Lateral displacement error for PPO Baseline.

The RL agent 1 is simulated for 2000 timesteps and GRF cost is calculated based on Equation 3.31 and with $w_{crutch\_reaction\_force} = 40000$ as seen in Figure 5.14 (a). The percent velocity/orientation tracking error and lateral displacement are also plotted in Figure 5.14 (b), Figure 5.15 (a) and (b), respectively.



a) RL agent 1 - GRF b) RL agent 1 - Velocity error

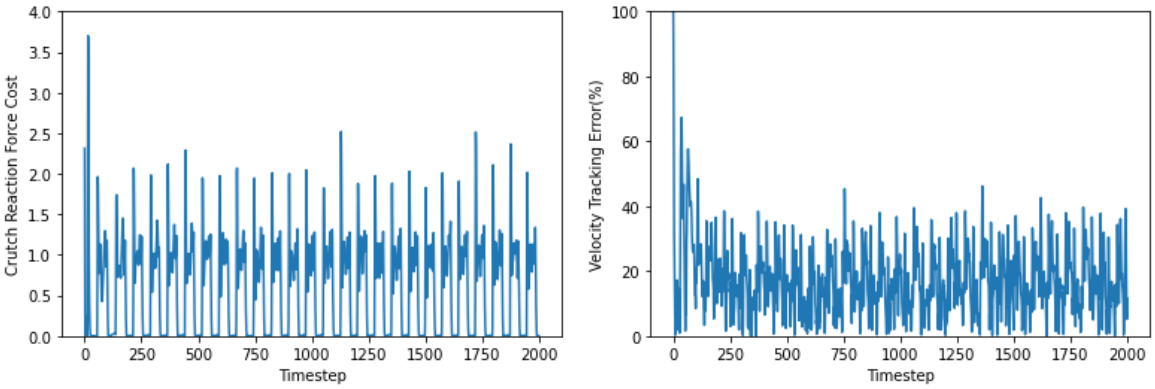
Figure 5.14. (a) GRF on crutch cost for RL agent 1, (b) Percent velocity tracking error for RL agent 1.



a) RL agent 1 - Orientation error b) RL agent 1 - Lateral displacement

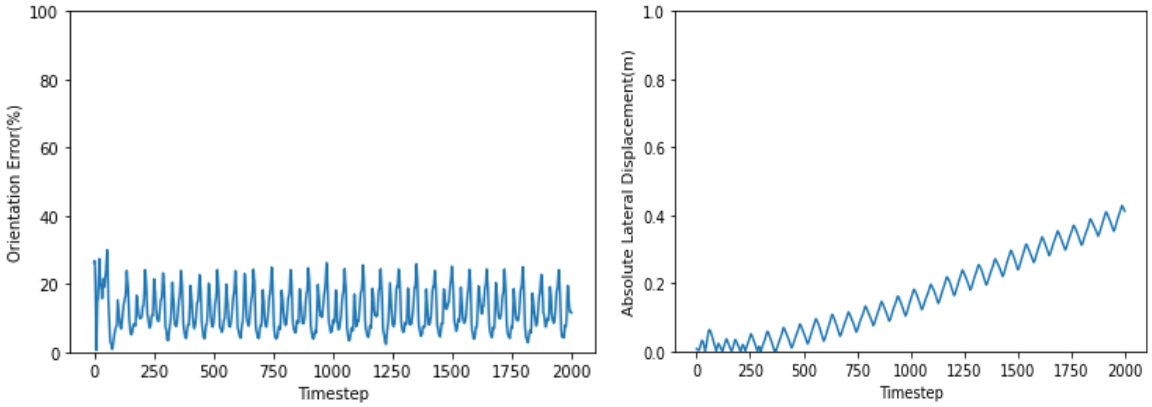
Figure 5.15. (a) Percent orientation error for RL agent 1, (b) Lateral displacement error for RL agent 1.

The RL agent 2 is simulated for 2000 timesteps in the same way as the others. The performance plots for the RL agent 2 are presented in Figure 5.16 and Figure 5.17.



a) RL agent 2 - GRF b) RL agent 2 - Velocity error

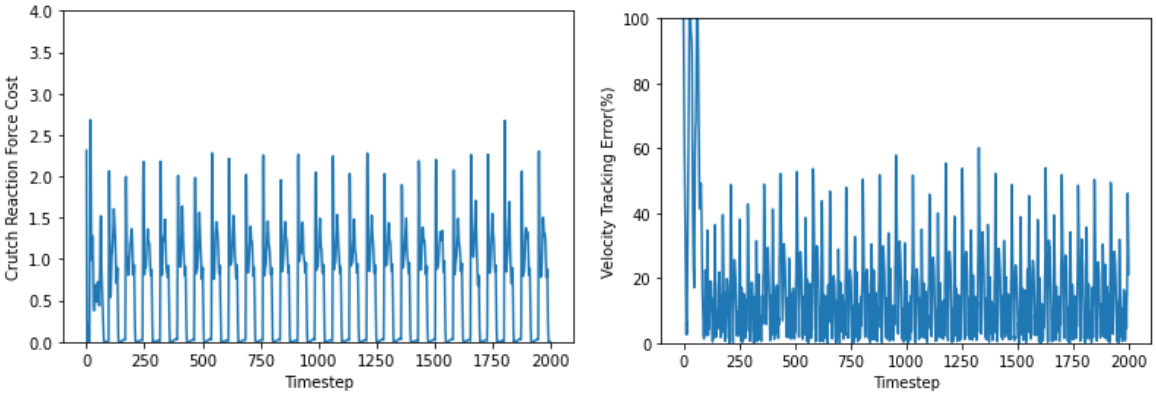
Figure 5.16. (a) GRF on crutch cost for RL agent 2, (b) Percent velocity tracking error for RL agent 2.



a) RL agent 2 - Orientation error b) RL agent 2 - Lateral displacement

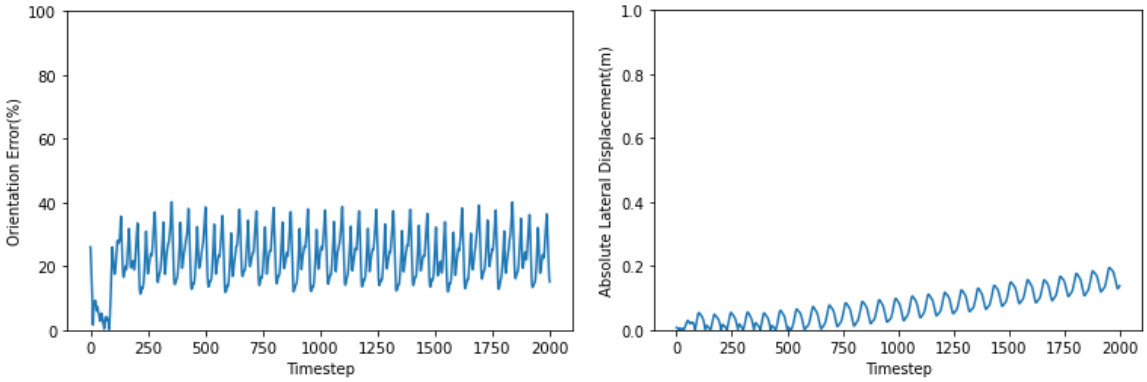
Figure 5.17. (a) Percent orientation error for RL agent 2, (b) Lateral displacement error for RL agent 2.

The RL agent 3 is simulated for 2000 timesteps in the same way as the others. The performance plots for the RL agent 3 are presented in Figure 5.18 and Figure 5.19.



a) RL agent 3 - GRF b) RL agent 3 - Velocity error

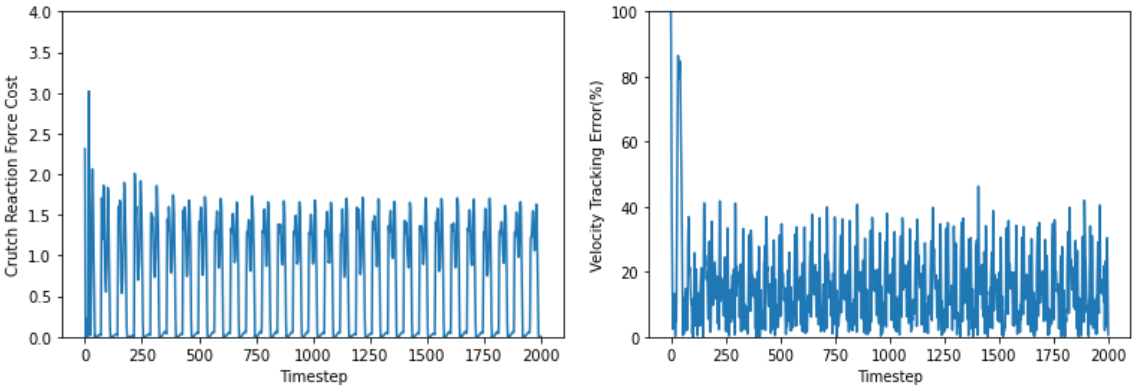
Figure 5.18. (a) GRF on crutch cost for RL agent 3, (b) Percent velocity tracking error for RL agent 3.



a) RL agent 3 - Orientation error b) RL agent 3 - Lateral displacement

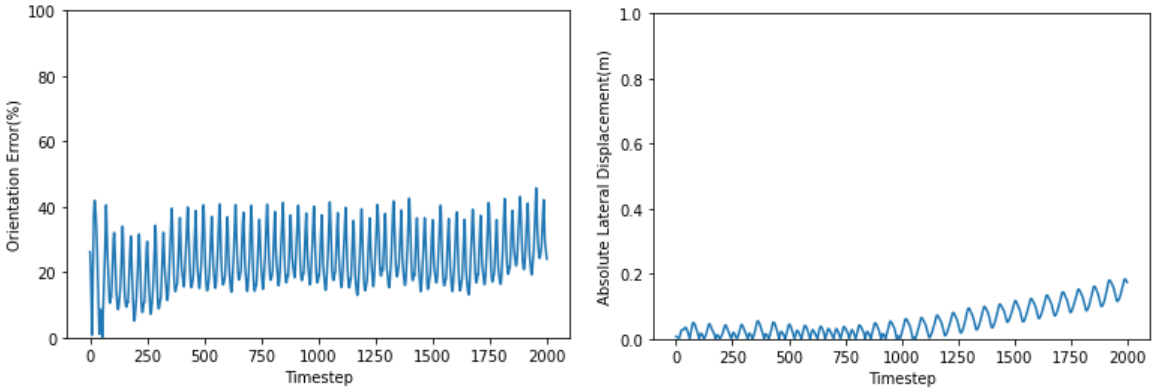
Figure 5.19. (a) Percent orientation error for RL agent 3, (b) Lateral displacement error for RL agent 3.

The RL agent 4 is simulated for 2000 timesteps in the same way as the others. The performance plots for the RL agent 4 are presented in Figure 5.20 and Figure 5.21. Note that $w_{crutch\_reaction\_force}$ selected as 40000 only during the testing in order to compare the GRFs of different trials, because we are interested in comparing the term (${d_{crutch\_r}}^2 + {d_{crutch\_l}}^2$) in Equation 4.31 for different trials.



a) RL agent 4 - GRF b) RL agent 4 - Velocity error

Figure 5.20. (a) GRF on crutch cost for RL agent 4, (b) Percent velocity tracking error for RL agent 4.



a) RL agent 4 - Orientation error b) RL agent 4 - Lateral displacement

Figure 5.21. (a) Percent orientation error for RL agent 4, (b) Lateral displacement error for RL agent 4.

The screenshots from the gait generated by RL agent 1 is available as a sequence in Figure 5.22.

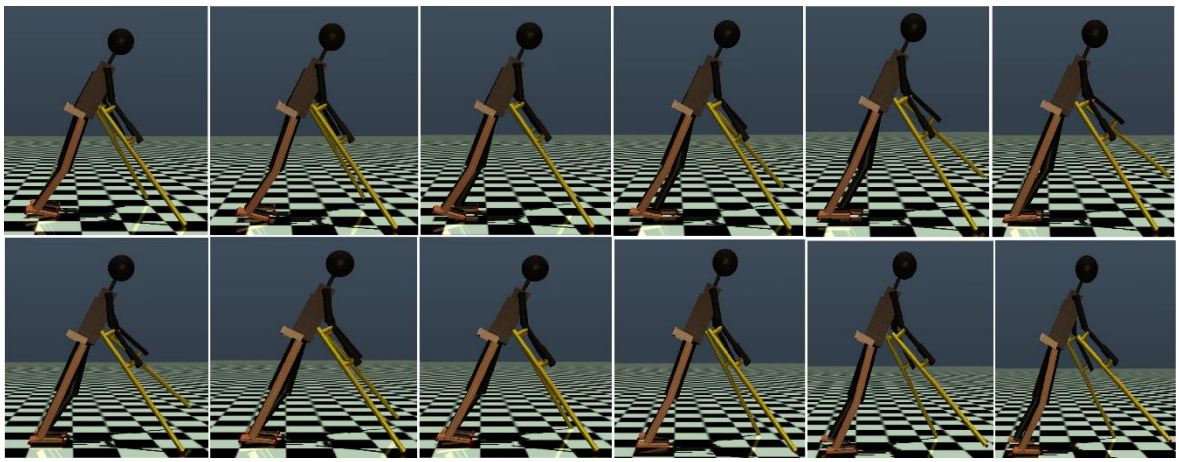


Figure 5.22. Gait sequence of RL agent 1

6. DISCUSSION

It is important to have an intuition about the amount of reward in reinforcement learning techniques such that the results can be assessed properly. The reward function used throughout all the trials made in RaiSim have a mathematical upper limit of 1. In other words, the agent should get a reward of 1 in each timestep if it can generate perfect actions that can fulfill the reward function 100%. Since maximum length of each episode is 7000, it can be said that a perfect reinforcement learning agent should reach a reward of 7000 for each trial episode. However, this is not physically possible. Instead, expecting a reward around 4000-5000 would be a more realistic approach for RaiSim trials. For MuJoCo trials, the reward amounts are observed along with the physical rendering of the system and it is concluded that a reward amount around 2000 corresponds to a reasonable gait with crutch usage. If the agent can reach these reward levels, it can be said that the agent generates actions that leads to a forward gait with crutch usage in a straight direction without falling and generating exaggerated joint torques.

For results of TD3 trials, it is observed that most of the trials have converged to very poor local optima. The worst result for TD3 is obtained with parameter set 6 with an average test reward of 62.72 which is very far from the desired reward range of 4000-5000. Considering that the training may start from rewards around 100, it can be concluded that even a random RL agent can achieve this result. Thus, it is observed that training runs with parameter sets 1, 3, 4 and 5 have actually learned some non-random behavior. The best result for TD3 has been obtained with the parameter set 5 with an average test reward of 752.63. Even though this reward is larger than the other TD3 trials, it is still not enough for an efficient gait as it is quite far away from the desired reward range. In conclusion, TD3 method failed to converge a meaningful and useful result in terms of crutched exoskeleton gait during RaiSim trials.

For the results of PPO trials made by using RaiSim, it can be observed that the results are generally better than the results of TD3 if a comparison between Table 5.1 and Table 5.2 is made. However, it is also visible that the most of the trials converged to poor local optima. The worst result has been obtained with parameter set 1 with an average test reward of

447.67. This reward score is far from a reward that a random RL agent is expected to achieve, but it is still very far away from the desired reward range. The best result for PPO trials belongs to parameter set 6 with an average test reward of 3210.54. This reward score is significantly higher than the other trials and it is also very close to the desired reward range. When the network trained with parameter set 6 is rendered and tested, it is observed that the agent actually translates to the forward direction without falling and by making use of the crutches in order to maintain the stability. However, the translation amount is quite small when compared to the expected forward displacement. The desired forward displacement was 0.0005 meters in between consecutive time samples. Since each episode is 7000 timesteps long, the expected forward displacement is 3.5 meters in a perfect scenario. Even though this expected forward displacement is unachievable in a realistic scenario, it is plausible to expect a forward displacement around 2 meters. Nevertheless, the agent's forward displacement is observed as around 0.5 meters at most. Moreover, joint torques generated by the agent may have some extreme values sometimes and this is observable from the rendering results. These unnecessarily high joint torque values manifest themselves in the form of unnecessary movements of the hip section. The solution for these kind of problems is to penalize the joint torques further in the reward function. Thus, a more natural looking gait may be obtained.

Another solution for making the solutions better may be redesigning the other parts of the reward function, because finding the optimal reward function depends on trial and error most of the time and this is a non-trivial process. In order to encourage the forward displacement more, the weight of forward walking reward may be increased. Another reason for not being able to reach the desired reward range and desired behavior may be the physics simulator. Even though RaiSim is rated better than the other available simulators for most of the aspects, it is still a new simulator which has not been tried extensively on reinforcement learning based locomotion studies. In the supporting direction of this doubt, there is a study that considers the usage of RaiSim and Mujoco for the purpose of simulating the contacts by legged robots during cat-like jumping and landing. Rudin et al. [45] states that even though RaiSim is quite good at simulating the hard contacts between feet and ground and is faster than Mujoco, it can have some problems when it comes to handle the problems with complex nonlinear dynamics. The reason of this situation is simply put as RaiSim's integration method Euler method can diverge quickly in the study by Rudin et al.

This exact problem may be also present in the crutched exoskeleton gait as well, as it also depends on complex nonlinear dynamics. Therefore, the reason for failing during RaiSim trials may be the instability of the Euler integration method.

In the latter part of this study, the simulation environment is changed to MuJoCo as its efficacy at locomotion tasks is shown in many different studies and examples. MuJoCo physics simulator is combined with high quality PPO implementation from OpenAI Baselines [30]. After seeing that locomotion patterns are achieved by some simple reward terms such as walking forward and not falling, the reward function is shaped further to capture an improved gait that includes reasonable crutch usage. Since the resulting gait is satisfactory, the last reward term for optimizing the ground reaction forces on the crutch tips is added. After numerous trials with different reward weights and seeds, 4 best results with optimized GRF's on crutch tips are selected. 5 separate trainings are run for the baseline without GRF minimization term and 3 of them were successful in terms of exoskeleton gait with crutches. Baseline-2 can be regarded as the best baseline RL agent, because the GRF's and other performance metrics in Table 5.3 are better than those of other two baseline versions. Based on the Table 5.3., it is observed that average crutch reaction cost for RL agent 1 is decreased around 34.85 % compared to the Baseline-2, which is the best baseline. Even though the average absolute orientation error of RL agent 1 is greater than the other RL agents, the orientation of the body does not seem very different compared to an exoskeleton user with crutches visually. Moreover, all of the other RL agents trained with different GRF minimization coefficients also minimize the GRF's on the crutch tips when compared to the baselines. The GRF's on the crutch tips tend to decrease as the parameter $w_{crutch\_reaction\_force}$ increases, just as expected. The reason is that the GRF's are penalized more as the parameter $w_{crutch\_reaction\_force}$ increases. The GRF's are minimized by 23.54 %, 19.51% and 13.52% by RL agent 2, 3 and 4 respectively. As a result, it can be concluded that the RL agent 1 trained with $w_{crutch\_reaction\_force}$ = 40000 yields best result which corresponds to 34.85 % improvement in GRF's on the crutch tips, considering the observations made during the rendering and Table 5.3.

In Table 5.3, it can be seen that GRF's are minimized successfully. However, the tracking errors for orientation, velocity and lateral displacement errors are slightly higher than expected. By using some conventional control methods such as PID control, Linear

Quadratic Regulator or any kind other state-of-the-art optimal control method, these tracking errors and lateral displacement errors may be decreased with respect to the outcomes of this work. Then mathematically modeling a complex system such as exoskeleton gait with crutch usage would be the case. By using RL instead of these conventional methods, the burden of complex mathematical modeling is avoided, but greater tracking and lateral displacement errors should be tolerated in this case. Thus, the selection of the method can be regarded as an engineering consideration which requires an analysis of these trade-offs.

7. CONCLUSION

In this study, we have developed a motion controller for a lower limb powered exoskeleton using deep reinforcement learning. The exoskeleton considered in this study has actuation only on its sagittal plane, which obligates the user to use crutches to keep the equilibrium. This situation results in increased metabolic energy consumption for the user. This issue has not been tackled in the literature yet as it is quite difficult to measure or formulize the metabolic energy consumption. However, there is a relation between the upper body metabolic energy consumption and GRFs on the crutch tips. By introducing a reward term about the GRFs exerted on the crutch tips, a minimization about these GRFs is aimed to decrease the upper body effort of the exoskeleton user. After trying different physics simulators and parameter sets, four different RL agents that can generate a gait with crutch usage and minimized GRFs are developed by using PPO algorithm and MuJoCo. These RL agents are compared with a baseline agent trained without any reward term about GRFs on crutch tips. It is shown that one of the developed RL agents can achieve 34.85 % minimization in GRFs on crutch tips, compared to the baseline RL agent. Thus, a contribution with respect to the state-of-the-art by means of minimizing the metabolic energy consumption is made in this study. Nevertheless, the findings in this thesis work are reached in simulation only.

A shortcoming of this work is the coupling between the robot actuators and the human actuators. The goal of the part of actor network that generates the actions for the arm movements is providing a prediction about the human upper body movements during crutched exoskeleton gait. However, there is a common network for generating the human actuations and the robot actuations. This results in that the human actuations are also optimized during the training along with the robot actuators. This coupling may be an obstacle while transferring the simulation work to the real world. On the other hand, decoupling the human and robot actuators will result in a need of establishing a realistic bridge between robot actuations on the lower body and the human actuations in the upper body.

7.1. Outlook and Future Work

In this study, the findings are reached by simulation only. Thus, conducting real world experiments and comparing the developed RL agents with the baseline versions and some other lower-limb exoskeleton controllers would be beneficial to validate the findings of this thesis.

Apart from the consideration of upper body metabolic effort, another gap in the area of lower-limb exoskeleton locomotion is the bridge between lower body movements generated by the exoskeleton and the arm movements of the user. If there have been more work that establish this connection between the upper and lower body movements of an exoskeleton user with crutches, this information could have been used to train a neural network. This pre-trained network would act as a human upper body motion predictor and could be used in the reinforcement learning loop. This approach would result in a better result in real world trials, but the essential problem is the need of a huge training data that includes lower body trajectories and upper body trajectories. Filling this gap would be a fine contribution to the state-of-the-art exoskeleton control methods.

REFERENCES

1. Secciani, N., C. Brogi, M. Pagliai, F. Buonamici, F. Gerli, F. Vannetti, M. Bianchini, Y. Volpe and A. Ridolfi, "Wearable Robots: An Original Mechatronic Design of a Hand Exoskeleton for Assistive and Rehabilitative Purposes", *Frontiers in Neurorobotics*, Vol. 15, No. 750385, 2021.

2. de la Tejera, J., R. Bustamante-Bello, R. A. Ramirez-Mendoza and J. Izquierdo-Reyes, "Systematic Review of Exoskeletons Towards a General Categorization Model Proposal", *Applied Sciences*, Vol. 15, No. 76, pp. 1-25, 2021.

3. Tiboni, M., A. Borboni, F. Vérité, C. Bregoli and C. Amici, "Sensors and Actuation Technologies in Exoskeletons: A Review", *Sensors*, Vol. 22, No.3, p. 884, 2022.

4. Baud, R., A.R. Manzoori and A. Ijspeert and M. Bouri, "Review of Control Strategies for Lower-limb Exoskeletons to Assist Gait", *Journal of NeuroEngineering and Rehabilitation*, Vol.18, No.1, pp. 1-34, 2021.

5. Abbasi, M., A. Shahraki, M.J. Piran and A. Taherkordi, "Deep Reinforcement Learning for QoS Provisioning at the MAC Layer: A Survey", *Engineering Applications of Artificial Intelligence*, Vol. 102, No.104234, pp.20, 2021.

6. Sutton, R. S. and A. G. Barto, *Reinforcement Learning: An Introduction, Volume 1*. MIT Press Cambridge, 1998.

7. L. Rose, M. C. F. Bazzocchi and G. Nejat, "End-to-End Deep Reinforcement Learning for Exoskeleton Control", *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Toronto, Canada, pp. 4294-4301, 2020.

8. Ouyang, W., Y. Wang, S. Han, Z. Jin and P. Weng, "Improving Generalization of Deep Reinforcement Learning-based TSP Solvers", *IEEE Symposium Series on Computational Intelligence (SSCI)*, Orlando, USA, pp. 01-08, 2021.

9. Brownlee J., “How to Choose an Activation Function for Deep Learning”, 2021, https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/, accessed on December 11, 2022.

10. Serengil S.I., “Softplus as a Neural Networks Activation Function”, 2017, https://sefiks.com/2017/08/11/softplus-as-a-neural-networks-activation-function/, accessed on December 11, 2022

11. Dombrowski, A., M. Alber, C.J. Anders, M. Ackermann, K. Müller and P. Kessel, “Explanations Can Be Manipulated and Geometry is to Blame”, *Conference on Neural Information Processing Systems (NeurIPS)*, Vancouver, Canada, pp. 13589-13600, 2019.

12. Pramoditha R., “How to Choose the Right Activation Function for Neural Networks”, 2022, https://towardsdatascience.com/how-to-choose-the-right-activation-function-for-neural-networks-3941ff0e6f9c, accessed on December 11, 2022.

13. Van Heeswijk W., “The Five Building Blocks of Markov Decision Processes”, 2022, https://towardsdatascience.com/the-five-building-blocks-of-markov-decision-processes-997dc1ab48a7, accessed on December 11, 2022.

14. Uther, W., *Temporal Difference Learning*. Encyclopedia of Machine Learning, Springer, Boston, MA., 2011.

15. Hui J., “RL-Policy Gradient Explained”, 2018, https://jonathan-hui.medium.com/rl-policy-gradients-explained-9b13b688b146, accessed on December 11, 2022.

16. Shyalika C., “A Beginners Guide to Q-Learning”, 2019, https://towardsdatascience.com/a-beginners-guide-to-q-learning-c3e2a30a653c, accessed on December 11, 2022.

17. Chao, K., and P. Hur, “A Step Towards Generating Human-like Walking Gait via Trajectory Optimization Through Contact for a Bipedal Robot with One-Sided

Springs on Toes", *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),* Vancouver, Canada, pp. 4848-4853, 2017.

18. Ackermann, M. and A. J. van den Bogert, "Optimality Principles for Model-based Prediction of Human Gait", *Journal of Biomechanics*, Vol. 43, No. 6, pp. 1055 – 1060, 2010.

19. Kelly, M., "An Introduction to Trajectory Optimization: How to Do Your Own Direct Collocation", *SIAM Review*, Vol. 59, No. 4, pp. 849-904, 2017.

20. Hereid, A., E. A. Cousineau, C. M. Hubicki and A. D. Ames, "3D Dynamic Walking with Underactuated Humanoid Robots: A Direct Collocation Framework for Optimizing Hybrid Zero Dynamics", *IEEE International Conference on Robotics and Automation (ICRA),* Stockholm, Sweden, pp. 1447-1454, 2016.

21. Luo, S., G. Androwis, S. Adamovich, H. Su, E. Nunez and X. Zhou, "Reinforcement Learning and Control of a Lower Extremity Exoskeleton for Squat Assistance", *Frontiers in Robotics and AI,* Vol. 8, No. 702845, 2021.

22. Taylor, M., S.D. Bashkirov, J. Rico, I. Toriyama, N. Miyada, H. Yanagisawa and K. Ishizuka, "Learning Bipedal Robot Locomotion from Human Movement", *IEEE International Conference on Robotics and Automation (ICRA)*, Xi'an, China, pp. 2797-2803, 2021.

23. Lillicrap, T.P., J.J. Hunt, A. Pritzel, N.M. Heess, T. Erez, Y. Tassa, D. Silver and D. Wierstra, 2015, "Continuous Control with Deep Reinforcement Learning", arXiv:1509.02971.

24. Mnih, V., K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski and S. Petersen, "Human-level Control Through Deep Reinforcement Learning", *Nature*, Vol. 518, No. 7540, pp. 529-533, 2015.

25. Peng, X.B., E. Coumans, T. Zhang, T.W. Lee, J. Tan, and S. Levine, 2020, “Learning Agile Robotic Locomotion Skills by Imitating Animals”, arXiv:2004.00784.

26. Margolis, G.B., G. Yang, K. Paigwar, T. Chen and P. Agrawal, 2022, “Rapid Locomotion via Reinforcement Learning”, arXiv:2205.02824.

27. Shi, J., T. Dear, and S.D. Kelly, “Deep Reinforcement Learning for Snake Robot Locomotion”, *IFAC-PapersOnLine*, Vol. 53, No. 2, pp. 9688-9695, 2020.

28. Zhang, M., X. Geng, J. Bruce, K. Caluwaerts, M. Vespignani, V. SunSpiral, P. Abbeel and S. Levine, “Deep Reinforcement Learning for Tensegrity Robot Locomotion”, *IEEE International Conference on Robotics and Automation (ICRA)*, Singapore, Singapore, pp. 634-641, IEEE, 2017.

29. Montgomery W. and S. Levine, “Guided Policy Search as Approximate Mirror Descent”, 2016, arXiv:1607.04614.

30. Ouyang, W., H. Chi, J. Pang, W. Liang and Q. Ren, “Adaptive Locomotion Control of a Hexapod Robot via Bio-inspired Learning”, *Frontiers in Neurorobotics*, Vol. 15, No. 627157, 2021.

31. Oghogho M., M. Sharifi, M. Vukadin, C. Chin, V. K. Mushahwar and M. Tavakoli, “Deep Reinforcement Learning for EMG-based Control of Assistance Level in Upper-limb Exoskeletons”, *International Symposium on Medical Robotics (ISMR)*, Atlanta, USA, pp. 1-7, 2022.

32. Luo, S., G. Androwis, S. Adamovich, E. Nunez, H. Su and X. Zhou, “Robust Walking Control of a Lower Limb Rehabilitation Exoskeleton Coupled with a Musculoskeletal Model via Deep Reinforcement Learning*”, Research Square*, 2021.

33. Fujimoto, S., H. Hoof and D. Meger, “Addressing Function Approximation Error in Actor-Critic Methods”, *International Conference on Machine Learning*, Stockholm, Sweden, Vol.80, pp. 1582-1591, 2018.

34. Kang D. and J. Hwangbo, “SimBenchmark”, 2018, https://leggedrobotics.github.io/SimBenchmark/, accessed on December 11, 2022.

35. Tirupachuri Y., L. Rapetti, C. Latella, R. Grieco, D. Ferigo, K. Darvish, “Human-Gazebo”, *GitHub Repository*, 2019, https://github.com/robotology/human-gazebo, accessed on December 11, 2022.

36. Latella C. and L. Rapetti, “Human-model-generator”, *GitHub Repository*, 2020, https://github.com/ami-iit/human-model-generator, accessed on December 11, 2022.

37. Todorov E., T. Erez and Y. Tassa, “MuJoCo: A Physics Engine for Model-based Control”, *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Algarve, Portugal, pp. 5026-5033, 2012.

38. https://zoo.cs.yale.edu/classes/cs470/materials/hws/aima/gym/gym/envs/mujoco/humanoid_v3.py, accessed on December 11, 2022.

39. Schulman, J., F. Wolski, P. Dhariwal, A. Radford and O. Klimov, 2017, “Proximal Policy Optimization Algorithms”, arXiv:1707.06347.

40. Schulman, J., S. Levine, P. Abbeel, M. Jordan and P. Moritz, “Trust Region Policy Optimization”, *International Conference on Machine Learning*, Lille, France, pp. 1889-1897, 2015.

41. Chou P.W., D. Maturana, and S. Scherer, “Improving Stochastic Policy Gradients in Continuous Control with Deep Reinforcement Learning Using the Beta Distribution”, *International Conference on Machine Learning*, Sydney, Australia, pp. 834–843, 2017.

42. Petrazzini, I.G. and E.A. Antonelo, “Proximal Policy Optimization with Continuous Bounded Action Space via the Beta Distribution”, *IEEE Symposium Series on Computational Intelligence (SSCI)*, Orlando, USA, pp. 1-8, 2021.

43. Dhariwal, P., C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu and P. Zhokhov, "OpenAI Baselines", *GitHub Repository,* 2017, https://github.com/openai/baselines, accessed on December 11, 2022.

44. Hasselt, H., "Double Q-learning", *Advances in Neural Information Processing Systems*, Vol.23, pp. 2613-2321, 2010.

45. Rudin, N., H. Kolvenbach, V. Tsounis and M. Hutter, "Cat-like Jumping and Landing of Legged Robots in Low Gravity Using Deep Reinforcement Learning", *IEEE Transactions on Robotics*, Vol. 38, No.1, pp.317-328, 2021.