UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DEPARTMENT OF INFORMATION ENGINEERING

COMPUTER ENGINEERING - AI & ROBOTICS

Gait Generation for Lower Limb Exoskeletons through Reinforcement Learning

`Candidate:`
Francesco Crisci

`Supervisor:`
Prof. Stefano Tortora

`Co-Supervisor:`
Dr. Edoardo Trombin

December 2nd, 2024
Academic year 2023-2024

Abstract

Using deep reinforcement learning (DRL) in the control of lower limb exoskeletons (LLE) remains largely unexplored, with most prior research focusing on bipedal or general walking robots. This study investigates the application of DRL to generate trajectory patterns for LLE movement, specifically targeting the foot's trajectory. By leveraging DRL, an agent can learn optimal behaviors by interacting with the environment. By defining an appropriate reward function, designing the network architecture, and conducting rigorous testing, we successfully generated the foot's trajectory for the LLE. The results demonstrate the potential of DRL to produce functional movement trajectories for lower limb exoskeletons.

Contents

List of Figures

List of Tables

List of Acronyms

LLE Lower Limb Exoskeleton
RL Reinforcement Learning
DRL Deep Reinforcement Learning
NN Neural Network
TD Temporal Difference
HLC Hip Lowering Constraint
ROS Robot Operating System
CFFTG Collision-Free Foot Trajectory Generator

Chapter 1

Introduction

1.1 Exoskeletons

Exoskeletons are biomechatronic devices coupled to the person's body. In general, exoskeletons are composed of a structural mechanism with joints and links, which is worn by a human user [1]. This thesis will focus on lower limb exoskeletons (LLEs).
A LLE is an exoskeleton designed for the lower limbs of the human body. As shown in Figure 1.1, each wearable leg is composed of:

- Hip
- Shin
- Thigh
- Foot

Each of the links is connected through joints, where some of them are powered by an electrical actuator.

Figure 1.1: Image of an LLE

Powered and passive exoskeletons differ primarily in their use of external energy sources. Powered exoskeletons rely on batteries or cables to activate actuators, enabling them to assist or enhance the user's movements. In contrast, passive exoskeletons lack these powered components, relying solely on mechanical structures like springs or dampers. This fundamental limitation prevents passive exoskeletons from enhancing the user's capabilities to the same extent as powered ones, as their passive joints cannot actively generate or amplify movement [2].

1.2 Reinforcement Learning for Bipedal Walking

Bipedal walking has always been a challenge in robotics. Most of the research done on bipedal walking is focused on bipedal robots and exoskeletons. In the literature, such as in [3, 4, 5] Reinforcement Learning (RL) used for Gait Planning is still an emerging field, focusing mainly on walking robots than exoskeletons. Using RL to generate trajectories for bipedal robots, and more specifically for LLE, is an open challenge, since the research is still new to this day.
Reinforcement Learning is a machine learning technique used to teach an agent how to interact within an environment. The agent learns about the environment through feedback from its actions. A RL model is composed of:

- An agent, learning how to solve a specific problem;
- States, i.e., the possible configurations the agent can find itself in;
- Actions, i.e., the possible decisions the agent can take in a given state;
- An environment where the agent operates, giving feedback as rewards.

Further exploring the research on the employment of RL on bipedal walking, we discovered some researches, as in [6, 7, 8], that explored the use of Deep Reinforcement Learning (DRL), a variation of RL that uses Deep Neural Networks to learn the environment. As stated in [8], DRL approaches can provide accurate and robust control in robotics applications

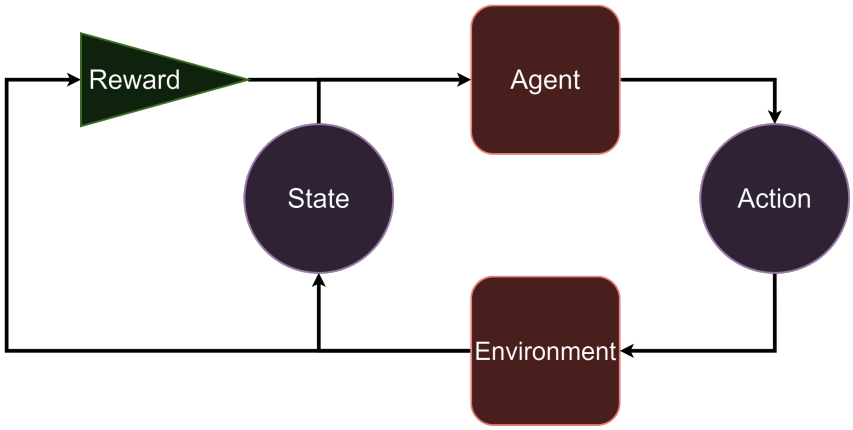

Figure 1.2: Basic Scheme of RL

and have shown potential for LLEs. Considering this statement, I decided to employ DRL to achieve the goal of this thesis, i.e., to generate trajectories for the LLE's foot.

1.2.1 Deep Reinforcement Learning: Temporal Difference

TD(λ) - a general reinforcement learning approach that covers a broad spectrum of methods ranging from Monte Carlo to SARSA to Q-Learning [9].

By changing the parameters of the aforementioned RL algorithm, it is possible to obtain the behaviors of different RL models. For the purpose of this thesis, I decided to set the parameters in order to achieve the same behavior as the Q-Learning model.

How the DRL and the TD model behave will be discussed further in the thesis. In our case, having a RL algorithm that can be tuned offers a variety of advantages that can be exploited for different needs. First of all, we are learning a model, so that the agent does not have a prefixed strategy when generating the trajectory. On the other hand, we can switch from a basic Q-Learning algorithm to a more complex one, such as SARSA (State-Action-Reward-State-Action), which is considered a state-of-the-art method.

1.3 Thesis aim and objectives

In this research we used a lower limb exoskeleton (LLE), with the aim of assisting people avoiding low obstacles while walking. The nature of the thesis is also related to the fact that this kind of robots have started to emerge as rehabilitation tools [10], so I decided to further explore their application with the goal of assisting people during daily life.
As stated, the aim of the thesis is Gait Generation for Lower Limb Exoskeletons through Reinforcement Learning, that is, employing DRL to generate the trajectory of the foot of the LLE. This research is a continuation of [11] carried previously. This solution, and the majority of obstacle avoidance methods, are polynomial. In this thesis, I decided to employ a different approach in order to introduce more generality in the generations of the trajectories. As a further development, taking into account the previous section, we decided to investigate the use of RL, and later of DRL, to achieve the same goal. That is, teach the agent to generate trajectories, autonomously, according to the obstacle it has to overcome.

Chapter 2

Methods

In this research the LLE used is a powered exoskeleton, where the joint of the ankle is passive, i.e. is not powered, while the others are, allowing the LLE to move.

2.1 Problem Description

The problem we address in the thesis is the following: we want to generate the foot trajectory of the LLE through the use of DRL in order to avoid low obstacles. The RL elements in our experiment are the following:

- The agent is the foot, which can move in a 2 dimensional space, the y-z plane (see Figure 2.1), with 10° increment between each direction.
- The environment is the 2 dimensional plane representing the sagittal plane where the foot, and in general the whole exoskeleton, can move and execute a given trajectory.
- The set of actions are the possible movements the foot can make in the plane, in our case are all directions with a 10° difference. That is, the agent can take 36 possible actions of unit distance from its current position, covering an angle of 360° of possible directions.
- The states are all the possible points in the positive y-z Cartesian plane where the agent can find itself.
- The reward function is a weighted sum of two components: the trajectory rewards and the exoskeleton constraint rewards. The form of

the reward function is the following:

$$r = \alpha \cdot T + \beta \cdot E, \text{ with } r \in [-\infty, 0] \tag{2.1}$$

with T and E the reward vectors and α, β the weights of each reward vector, respectively, and $\cdot$ is the scalar product. We also want the following to be true:

$$\sum_{i \in \alpha} i = 0.5, \qquad \sum_{i \in \beta} i = 0.5 \tag{2.2}$$

2.2 Agent Environment

In the environment it is not considered the LLE's movements in the x-axis, and it has the following characteristics:

1. The agent's initial position, in our case the foot initial position, which is the origin of the Sagittal plane. We put the agent in this position to not have transformations between the LLE's reference frame and the generated trajectory points.

2. An obstacle, which is set according to the received data. That is, once the position of the obstacle is known, we set it in the environment and this position is known to the agent during training and testing time.

3. The goal position which is where we want the agent to be after it overcame the obstacle. This position is set according to the desired step length, which for the purpose of this thesis has been set equal to the average step length, which is 70 cm.

4. The mid-goal position, which is a way-point set on a safe distance on top of the obstacle where we want the agent to go through when calculating the trajectory. The midpoint is centered w.r.t. the obstacle alongside the y axis.

The environment takes as input the initial position of the agent, the position of the obstacle and the step length. Then, considering the length of the two joints, the shin and the thigh as 50 cm each, we also calculated the

initial position of the hip from fixing the pivot foot position. An example can be seen in the Figure 2.1, where we set the position of the pivot foot in correspondence to the center of the obstacle. As it can be seen from

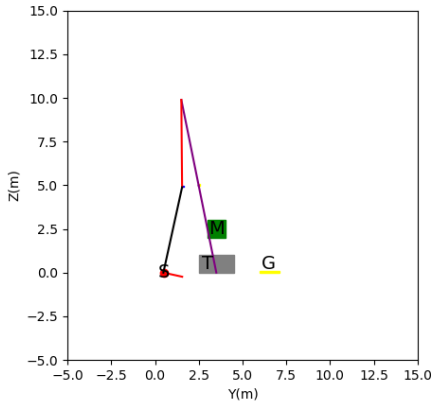

Figure 2.1: Example of initial configuration of the LLE in the environment

Figure 2.1, the starting position is the heel's starting position and it has been assigned the label 'S'. An example of obstacle is reported, represented as a gray rectangle and it has been assigned the label T. The mid-goal is the green square on top of the obstacle and has label 'M'. The goal is thin yellow line with label 'G'.
The hip starting position has been calculated as follows:

$$y = (t + s)\cos(\alpha), \qquad z = (t + s)\sin(\alpha) \tag{2.3}$$

where t is the length of the thigh, s is the length of the shin and α is the angle between the line connecting the two heels and the hip.
The measurements reported in Figure 2.1 are on the scale of dm, and the sagittal plane as been enlarged for clarity.
The trajectory generated in this example is going to be the trajectory of the left foot while keeping fixed the right one, i.e. the pivot foot.
Once the hip position and the foot position are known, we can calculate the position of the knee by using inverse kinematics. We can then write the following equations:

$$r = \sqrt{(h_y - f_y)^2 + (h_z - f_z)^2} \tag{2.4}$$

where r is the euclidean distance between the hip (h) and the considered foot (f).

$$\gamma = \frac{t^2 + r^2 - s^2}{2 \cdot t \cdot r} \tag{2.5}$$

where γ will be used to calculate the angle formed by the hip and r is the value calculated in (2.4). As a safety measure, γ is clamped in the interval $[-1, 1]$ according the following strategy, since the arccos can only get values from $[-1, 1]$:

$$\gamma = \begin{cases} 1 & \text{if } \gamma > 1 \\ -1 & \text{if } \gamma < -1 \end{cases} \tag{2.6}$$

We then calculate the hip angle as:

$$\theta = \arccos(\gamma) \tag{2.7}$$

We compute the angle between the horizontal axis and the pointing vector from the hip to the foot as:

$$\alpha = atan2\Big((f_z - h_z), (f_y - h_y)\Big) \tag{2.8}$$

Now we have two solutions for the knee, i.e., the solution that considers the knee to bend backwards and the solution that considers the knee to bend forward. To solve this problem, we calculated both cases and took the one with an higher y value. That is, we always consider the knee that goes forward since a knee can only bend, naturally, in that direction.

$$k'_y = h_y + t \cdot \cos(\alpha + \theta), \qquad k''_y = h_y + t \cdot \cos(\alpha - \theta) \tag{2.9}$$

$$k'_z = h_z + t \cdot \sin(\alpha + \theta), \qquad k''_z = h_z + t \cdot \sin(\alpha - \theta) \tag{2.10}$$

From equations (2.9) and (2.10), we can find the final value of the knee position as follows:

$$k_y = \begin{cases} k'_y & \text{if } k'_y > k''_y \\ k''_y & \text{otherwise} \end{cases}, \qquad k_z = \begin{cases} k'_z & \text{if } k'_y > k''_y \\ k''_z & \text{otherwise} \end{cases} \tag{2.11}$$

The same reasoning is then applied for the pivot foot, obtaining the y and z coordinates of the knee.
We now have all the elements of the environment and all the elements to know the position of the exoskeleton and the states of the links and joints.

2.3 Temporal Difference

TD learning is a combination of Monte Carlo ideas and dynamic programming (DP) ideas. [12]. The main advantage of TD learning is that the agent does not have to wait for the final outcome in order to update the estimates, but it can update them starting from other estimates. That is, the estimate of the current state S_t can be updated at time $t+1$ instead of waiting for the whole episode to end. To achieve so, they use the reward at time $t+1$, i.e., R_{t+1} and the estimate $V(S_{t+1})$. In the case of the base implementation of TD, known as TD(0), we have the following estimate update rule [12].

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1} - V(S_t) \right] \tag{2.12}$$

We can now write the pseudo-code for TD(0) estimating v_π, i.e., the estimate of the states under policy π: Now, to deal with the trade-off between

Algorithm 1 TD(0) prediction

Input: The policy π to be evaluated
Initialize V(s) arbitrarily
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        $A \leftarrow$ action given by π for S
        Take action A; observe reward, R, and next state, S'
        $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
        $S \leftarrow S'$
    until S is terminal

exploration and exploitation, we need to introduce two kind of TD controls: On-policy and off-policy TD controls. In RL exploitation is the process of choosing an action with the aim to get the best reward possible. Exploration is the process of choosing an action in order to explore new

states. Finding a balance between these two elements is important to have an algorithm that performs and converges efficiently.

2.3.1 Off-Policy TD Control

In the case of one-step Q-learning, we approximate the optimal action-value function with the learned one, by following the update rule:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right] \quad (2.13)$$

This method updates the action-value function independently from the policy being followed. On the other hand, we still take into account the pairs of state and action to visit and how they are updated. A fundamental requirement that leads to convergence is the constant update of all action-state pairs. Below, we show the Q-learning algorithm in procedural form [12]:

Algorithm 2 Off-policy TD control

Initialize Q(S,A), $\forall s \in S, a \in \mathcal{A}(s)$, arbitrarily, and Q(terminal-state,.)=0
Repeat (for each step of episode):
    Initialize S
    Repeat (for each episode):
        Choose A from S using policy derived from Q
        Take action A; observe reward, R, S'
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\big]$
        $S \leftarrow S'$
    until S is terminal

2.3.2 On-Policy TD control

The main difference w.r.t. the off-policy method is that in this case we focus on learning action-value function rather than state-value ones [12]. That is, for all states s and actions a following the current policy π, we need to estimate $Q_\pi(s, a)$, which can achieved as described in the TD(0) method. We can write the update as follows:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \right] \quad (2.14)$$

After every transition from a non-terminal state S_t we execute the update. If S_{t+1} is a terminal state, we define $Q(S_{t+1}, A_{t+1})$ as zero [12]. In Algorithm 3, the implementation of On-Policy control is reported:

Algorithm 3 Off-policy TD control

Initialize Q(S,A), $\forall s \in S, a \in \mathcal{A}(s)$, arbitrarily, and Q(terminal-state,.)=0
Repeat (for each episode):
    Initialize S
    Choose A from S using policy derived from Q
    Repeat (for each step of episode):
        Take action A; observe reward, R, S'
        Choose A' from S' using policy derived from Q
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$
        $S \leftarrow S'$, $A \leftarrow A'$
    until S is terminal

2.3.3 N-Step TD Prediction

Differently from TD(0) methods, n-step TD models are not just based on the next reward, but on the next N rewards. As done in the TD methods, the estimates of the rewards are changed according to the future rewards [12]. To describe more formally how a n-step approach works, we are gonna introduce the notion of cumulative discounted reward at time t. For the 1 step case we can write it as:

$$G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1}) \tag{2.15}$$

Following the same reasoning, we can write the 2 step case as follows:

$$G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2}) \tag{2.16}$$

Following (2.15) and (2.16), we can now write the cumulative discounted reward at time t for n-steps:

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n}) \tag{2.17}$$

where R_t is the reward at time t, $V(S_{t+n})$ is the estimated value of the next state and γ is the discount factor. The value calculated in equation (2.17) is the addition between the estimated value of the n-th state and

the truncated return after n steps. Moving toward a tabular state-value implementation we can write the increment to $V_T(S_t)$ as [12]:

$$\Delta V_t(S_t) = \alpha \left[G_t^{(n)} - V_t(S_t) \right] \tag{2.18}$$

where α is a positive step-size parameter.
In this approach, we need to deal with the size of n, i.e., the number of steps to observe before updating the state-value pairs. That means, in real word scenarios we cannot use extremely high values of n since it can become problematic.

2.3.4 TD(λ)

TD(λ) can be achieved by two different meanings: the forward view or the mechanistic (backward) view.
In the forward view we can extend further the concept of n-step returns. In fact, we can calculate the increment not only depending on the n-step returns, but also with any average of the n-steps. We can think of TD(λ) as a specific way to average the n-steps [12]. We can introduce now a different kind of return by using the so called λ-return:

$$G_t^{\lambda} = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)} \tag{2.19}$$

As it can be seen from equation (2.19), the λ-return contains all the n-steps. The more steps we take into account, the smaller the weight of the next return. We want to give a bigger importance to the closer states than the ones further away. According to the choice of λ, we can tune the behavior of the return:

- if $\lambda = 1$, we calculate the return as done in a Monte-Carlo approach.

- if $\lambda = 0$, we reduce ourselves to the TD(0) approach.

We can now write the update rule for the TD(λ) model in terms of increment:

$$\Delta V_t(S_t) = \alpha \left[G_t^{\lambda} - V_t(S_t) \right] \tag{2.20}$$

Also in this case, the update can be on-policy or off-policy. [12].
By working in a continuous space, it is not ideal to use the forward view. That is because we need to wait for an episode to end before the current state S_t can be updated; in a continuous environment it could possibly take an infinite amount of time. By using eligibility traces, i.e., by assigning how much credit to a previous state, we can update the current state S_t [9]. We can express the state-value update as follows:

$$V(s) \leftarrow V(s) + \alpha[V_t - \hat{V}(S_t)]E_t(s) \tag{2.21}$$

where we can define the expected value of state s at time t as follows:

$$E_t(s) = \begin{cases} \gamma E_{t-1}(s) & \forall s \in \text{ visited} \\ \gamma E_{t-1}(s) + 1 & \text{if } s = S_t \end{cases} \tag{2.22}$$

2.3.5 Function Approximation

The framework described in section 2.3.4 is the baseline of the approach used in the thesis. Considering the fact that it would be impossible to know all the state-value values for all possible infinite states in a continuous environment, we need to approximate the values of states that are similar to each other. We can refine the action-value function as a weighted sum of features [9]:

$$Q(s, a; \theta) = f(\theta^T \phi(s, a)) \tag{2.23}$$

where θ are the parameters of our network and $\phi(s, a)$ are the features of the state-action pair. For the purpose of the thesis, we adopted the same approach as done in [9], which uses a neural network to approximate the action-value function. The neural network used, as you can see from Figure 2.2, is a 5-layer network, with the following characteristics:

- The input layer is composed of two nodes, i.e. the y-z coordinate of

the agent.

- The second layer is composed of 64 neurons.
- The third, fourth and fifth layer are composed of 32 neurons each.
- The output layer is composed of 36 neurons, each one indicating a possible action the agent can take.

All neurons in the hidden layer uses the ReLU activation function [13], and all the layers are fully-connected. Now, to update the parameters of

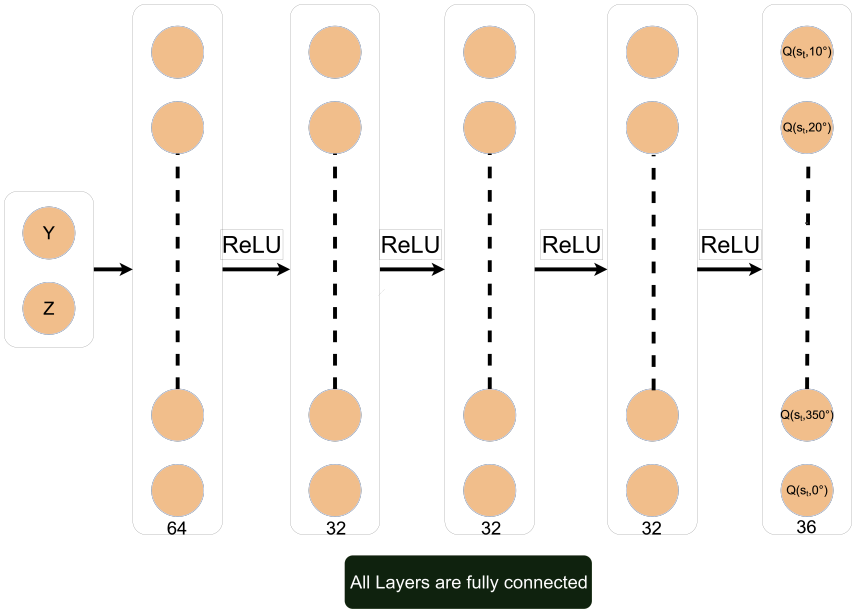

Figure 2.2: Neural Network used to learn the environment

the neural network we use the gradient descent rule. Starting from the following equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[Q_t - \hat{Q}(S_t, A_t)]E_t(s,a) \tag{2.24}$$

and by using the mean squared error as loss function:

$$J(\theta) = \frac{1}{2}||[Q_t - Q(S_t, A_t; \theta)]E_t(s,a)||^2 \tag{2.25}$$

obtaining equation (2.26)[9]:

$$\theta \leftarrow \theta + \alpha \cdot \delta_t \cdot E_t(s,a)\nabla_\theta Q(S_t, A_t; \theta), \qquad \delta_t = Q_t - q(S_t, A_t; \theta) \tag{2.26}$$

2.4 The Agent

The agent of our DRL problem is the LLE's foot, which moves in a continuous 2D environment in order to reach the desired final position. The agent is represented by the NN defined in section 2.3.5, and an action selection strategy. The NN and the selection strategy are then used in the TD(λ) algorithm.

2.4.1 The TD(λ) class

Once we set the architecture of the NN, we created the TD(λ) class, which is composed of two identical networks with an ADAM optimizer [9]. The reason why we need two NNs is simple: while the weights of the network change during training, our estimates will improve, but our target output will change, making the agent not being able to understand the target we are focusing on. As done by [9], we have the following two NNs:

1. Q_{main} related to the estimates $Q(S_t, A_t; \theta)$;

2. Q_{target} related to the target value Q_t, which is updated periodically.

While Q_{main} will be updated systematically at each step, improving our estimates
$Q(S_t, A_t; \theta)$, as stated previously Q_{target} is updated periodically from the newly learned parameters from Q_{main}, allowing us to decouple the relation between Q_t and $Q(S_t, A_t; \theta)$.

2.4.2 Selection Strategy

As a baseline strategy for the agent to choose the best action, as done by [9], we used a ϵ-greedy policy, meaning that the agent will explore new actions with probability ϵ, while it will choose the best action seen so far with probability $1 - \epsilon$, with ϵ small. That means that the agent will prefer to take the best action discovered so far the majority of time,i.e., the agent will prefer exploitation over exploration.

2.4.3 Agent training

As baseline for the project, as done in [9], we decided to employ a limit to the number of steps the agent can take during each episode. That is, the agent cannot go on forever in order to reach the desired goal, so we are interested in finite episodes. To update the NN parameters, we store the outputs of all the steps taken into a buffer, from which a batch is randomly sampled. With the use of a dictionary we keep track of the trace of visited state-actions pair, and they are then reset at the beginning of each episode to keep them independent one from another [9]. In RL a trace is the ability of the agent to keep track of the states and the actions encountered while interacting with the environment. This allows the agent to learn from past experiences, making the learning process more efficient [14].

2.4.4 Network update

Once the batch is sampled from the memory buffer, the weights of the NN are updated taking into account if the model is on or off-policy.

- **On-policy**: starting from the Q_{target} we obtain:

$$Q_t = [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})] \tag{2.27}$$

- **Off-Policy**: we try to find the action that returns the best Q-value

$$Q_t = [R_{t+1} + \gamma \max_a Q(S_{t+1}, a)] \tag{2.28}$$

From here, we use the value of Q_t, $Q(S_t, A_t; \theta)$ and the eligibility trace $E_t(s, a)$ to calculate the loss and then update the parameters of the network [9].

2.5 The Reward Function

To evaluate the behavior of the agent during each episode we needed to implement a specific reward function in order to teach him how to behave. As can be seen from equation (2.2), the reward function is composed of two

elements: the reward related to the foot trajectory itself, so how the agent should move to recreate a physiological step, and a second part related to the exoskeleton kinematic constraints, so how the trajectory should be in order to be feasible for the LLE. Each one of the two components of the reward function are calculated with a different number of constraints, according to the behavior of the agent during training.

2.5.1 The Trajectory Reward Function

The trajectory reward function is composed of 8 constraints that controls and corrects the behavior of the agent so that the generated trajectory is as close as possible to a feasible trajectory.
The first constraint takes care of the number of steps the agent takes in the environment to reach the goal. The aim is that the agent does not take to many steps in order to achieve the final position, but also we don't want it to take to few so that the trajectory is too flat. So, we set a threshold to decide whether the number of steps are to many or to few. The reward of this component is then normalized by the threshold, so that the maximum reward is achieved when the agent takes exactly `threshold` steps to reach the goal. More formally, the first constraint can be expressed as:

$$r_0 = \begin{cases} \frac{s}{n} & \text{if } s \leq n \\ -\frac{s}{n} & \text{otherwise} \end{cases} \tag{2.29}$$

where s is the number of steps the agent took so far and n is the threshold. The second constraint checks whether the agent hits the obstacle while trying to reach the goal. That is, since we are in a simulated environment, if the agent finds itself "inside" the region filled by the obstacle, the agent will receive a penalty. Otherwise it is rewarded. More formally, we can express the second constraint as:

$$r_1 = \begin{cases} -c & \text{if } agent \cap obstacle \\ 0 & \text{otherwise} \end{cases} \tag{2.30}$$

where c is a defined positive constant. This constraint ensures that the agent never hits, or more precisely goes into, the obstacle. That is, we want the agent to learn that the obstacle is a region to avoid while generating the trajectory.

The third constraint is related to the behavior of the trajectory alongside the y axis. Since we want the agent to overcome an obstacle and to reach the goal, we want to make sure that the agent always has a monotonically increasing tendency alongside the y axis. That is, the position of the agent always as to be further, or in the same position, alongside the y direction. More formally, we can express the third constraint as:

$$r_2 = \begin{cases} -y & \text{if } y_{t-1} > y_t \\ 0 & \text{otherwise} \end{cases} \tag{2.31}$$

where y is a defined positive constant, y_t is the position of the agent alongside the y direction at time t, and y_{t-1} is the position of the agent alongside the y direction at time $t-1$.

The fourth constraint checks if the agent reached the goal position. We check if the position of the heel or the tip of the foot is inside the region where the goal position has been placed. If that is the case, we highly reward the agent and we reset the episode. If that is not the case, the agent is neither penalized nor rewarded. More formally we have:

$$r_3 = \begin{cases} g & \text{if } \mathit{heel} \in \mathit{goal} \\ g & \text{if } \mathit{tip} \in \mathit{goal} \\ 0 & \text{otherwise} \end{cases} \tag{2.32}$$

where g is a defined positive constant.

The fifth constraint takes care of the agent going through the midpoint position above the obstacle. This constraint has been introduced to suggest the agent to avoid the obstacle by going above it. That is, we want to ensure that the generated trajectory has a parabolic shape, passing through the midpoint. Unlike the constraint for the goal position, expressed in (2.32), we penalize the agent if the generated trajectory doesn't go through the

midpoint. We can express this constraint as:

$$r_4 = \begin{cases} 0 & \text{if } foot \in goal \\ -m & \text{otherwise} \end{cases} \tag{2.33}$$

where m is a defined positive constant.

The sixth constraint controls the agent displacement alongside the z-axis. We don't want the trajectory to be too high in order to avoid the obstacle. That is, when overcoming low obstacles, it would be inefficient and unnatural to raise the leg too much. For this purpose, we set a maximum acceptable height, defined as max_z. We can now express the sixth constraint as:

$$r_5 = \begin{cases} -z & \text{if } z_t > max\_z \\ 0 & \text{otherwise} \end{cases} \tag{2.34}$$

where z_t is the z-axis position of the agent at time t and z is a defined positive constant.

The seventh constraint makes the agent going around the obstacle with a given distance. While avoiding the obstacle, we want the agent to be far enough in order to ensure that the collision with the latter is unlikely. The safety distance is defined as a positive constant, expressed in dm, while the distance between the foot and the obstacle is calculated with a specific function explained in the next sub-section. If the foot-obstacle distance is below the defined constant, we heavily penalize the agent, otherwise we do not reward it. We can define the constraint as:

$$r_6 = \begin{cases} -f & \text{if } d_t < min\_dt \\ 0 & \text{otherwise} \end{cases} \tag{2.35}$$

where f is a positive defined constant, d_t is the distance from the obstacle calculated through $get\_min\_distance\_from\_trap$, (see subsection 2.5.2) and max_dt is the minimum required distance from the obstacle we want the agent to achieve.

The eight constraint is related to the behavior of the trajectory alongside the z-axis. We want the trajectory to be monotonically increasing before

the reach of the midpoint and monotonically decreasing after. That is, we want to induce a raising behavior before reaching the obstacle position, while we want to induce a descending behavior after. To achieve so, we introduced the following constraint, divided in two smaller constraints:

$$r_7^{bm} = \begin{cases} 0 & \text{if } z_{t+1} \geq z_t \\ -k & \text{otherwise} \end{cases}, \qquad r_7^{am} = \begin{cases} 0 & \text{if } z_{t+1} \leq z_t \\ -k & \text{otherwise} \end{cases} \tag{2.36}$$

From equation (2.36), we can write the constraint as follows:

$$r_7 = \begin{cases} r_7^{bm} & \text{if } y_t <= m_t \\ r_7^{am} & \text{otherwise} \end{cases} \tag{2.37}$$

Where r_7^{bm} is the reward before the midpoint, r_7^{am} is the reward after, k is a defined positive constant, z_{t+1} is the position of the agent alongside the z-axis at time $t+1$, and z_t is the position of the agent alongside the z-axis at time t.

All the 8 constraints are then multiplied with a corresponding weight vector, that is then normalized as follows:

$$\alpha = \frac{\alpha}{\sum_{i \in \alpha} i} \tag{2.38}$$

Putting everything together, we can now rewrite the first part of the reward function as:

$$r_{traj} = \alpha^T \cdot T = \sum_{i=0}^{7} \alpha_i \cdot r_i \tag{2.39}$$

with:

$$\alpha = \begin{bmatrix} \alpha_0 \\ . \\ . \\ . \\ \alpha_7 \end{bmatrix}, \qquad T = \begin{bmatrix} r_0 \\ . \\ . \\ . \\ r_7 \end{bmatrix} \tag{2.40}$$

2.5.2 Calculating minimum distance from the obstacle

In order to calculate the distance between a segment and a point, following [11], we adapted the `point_to_segment` function. In the thesis' case, the segment would be the connection between the heel and the tip of the foot. The points to which we want to calculate the distance are sampled from the obstacle region, and then the minimum is kept. That is, we consider the smallest distance between the heel-tip segment and the closest obstacle point to it. If the newly found distance is below the given threshold, the agent is penalized. We now report the code for calculating the distance between the obstacle and the heel-tip segment, visible in **Algorithm 4**.

Algorithm 4 point_to_segment

Input: heel and hip positions
Initialize dist = 0, ab (heel-hip segment), bp (distance between heel and trap closest point) and ap (distance between tip and furthest point)
ab_bp = $ab \cdot bp$
ab_ap = $ab \cdot ap$
if ab_bp > 0 : return $\sqrt{bp_x^2 + bp_y^2}$
else if ab_bp < 0 : return $\sqrt{ap_x^2 + ap_y^2}$
else :
    $mod = \sqrt{ab_x^2 + ab_y^2}$
    return *seg_dist* (*eq.* (2.41))

$$seg\_dist = \frac{ab_x \cdot ap_y - ab_y \cdot ap_x}{mod} \tag{2.41}$$

2.5.3 The LLE Kinematics Reward Function

The LLE kinematics reward function is composed by 4 different constraint. They ensure that the generated trajectory is feasible for the LLE, and that the LLE does not violate the physical joint limits or finds itself in an unfeasible position.

The first constraint checks if the hip position is below a given value during the execution, or generation, of a trajectory. To better understand this constraint, we need to introduce how the hip position is calculated from the position of the foot at time t, i.e., when the agent is at a given state at time t. The constraint, the Hip Lowering Constraint (HLC), is related only to the z position of the hip. In order to check if it is violated, we need

to compute the z position of the hip as follows:

$$dist\_y = \frac{p_y - p_z}{2}, \qquad h_z = \sqrt{(t+s)^2 - dist\_y^2} \tag{2.42}$$

where p_y, p_z are the positions of the pivot foot in the y-z plane, t is the thigh length, s the shin length and h_z is the hip position alongside the z axis. Now, we can express the HLC and the agent reward as follows:

$$r_0 = \begin{cases} -\frac{o \cdot (t+s)}{h_z} & \text{if } \frac{h_z}{t+s} < hlc \\ 0 & \text{otherwise} \end{cases} \tag{2.43}$$

where s is the shin length, t is the thigh length, h_z is the hip position alongside the z axis, HLC is the defined threshold and o is a defined positive constant. As you can see, the agent is penalized the more the constraint is violated, so that it can learn not to lower the hip to much when overcoming the obstacle.

The second constraint is the kinematic constraint, related to the leg length during the generation of the trajectory. That is, we check if the euclidean distance between the hip and the heel is physically feasible during the execution of the trajectory. To do so, we calculate the euclidean distance as follows:

$$M = \sqrt{(f_y - h_y)^2 + (f_z - h_z)^2} \tag{2.44}$$

where f is the heel position and h is the hip position. Now we can defined the constraint and the reward as follow:

$$r_1 = \begin{cases} -l & \text{if } M - s - t > n \\ 0 & \text{otherwise} \end{cases} \tag{2.45}$$

where l is a defined positive constant, s is the shin length, t is the thigh length and n is the threshold value.

The third constraint is related to the limits of the left and right knees and hip angles. From equation (2.43) we know the distance between the heel and the hip. Using that distance, we can calculate the hip angle, following

the reasoning applied in equations (2.5) and (2.6):

$$\alpha = \arccos\left(\frac{t^2 + M^2 - s^2}{2 \cdot t \cdot M}\right) \tag{2.46}$$

We now have to calculate the tilt angle to get the knee position from the hip's one:

$$tilt = \arcsin\left(\frac{h_y - f_y}{M}\right) \tag{2.47}$$

Now, as done in equations (2.9) and (2.10), we get the hip positioning as follows:

$$k_y = h_y + t + \sin(\alpha + tilt), \qquad k_z = h_y - t + \cos(\alpha + tilt) \tag{2.48}$$

Now we can get the hip and knees angles in order to check the joint limits:

$$h_\theta = \arcsin\left(\frac{k_y - h_y}{t}\right), \qquad k_\theta = \left(\arcsin\left(\frac{f_y - k_y}{s}\right) - h_\theta\right) \tag{2.49}$$

We can now express the constraint and the reward formally as follows:

$$r_2 = \begin{cases} -j & \text{if } h_\theta > H_\theta^U \\ -j & \text{if } h_\theta < H_\theta^L \\ -j & \text{if } k_\theta > K_\theta^U \\ -j & \text{if } k_\theta < K_\theta^L \\ 0 & \text{otherwise} \end{cases} \tag{2.50}$$

where j is a defined positive constant, H_θ^U is the hip upper bound limit, H_θ^L is the hip lower bound limit, K_θ^U is the knee upper bound limit and K_θ^L is the knee lower bound limit.

The fourth constraint controls that the hip z position is not to high while executing the trajectory. That is, we do not want the agent to move to much forward making the trajectory unfeasible and physically impossible to execute in a real world environment. We can express the constraint and

the reward as follows:

$$r_3 = \begin{cases} -n & \text{if } h_z > max\_hz \\ 0 & \text{otherwise} \end{cases} \tag{2.51}$$

where n is a defined positive constant and max_hz is the maximum z position allowed.
We can now express, as done in equation (2.39) the LLE kinematics reward function as:

$$r_{exo} = \beta^T \cdot E = \sum_{i=0}^{3} \beta_i \cdot r_i, \text{ with } \beta = \frac{\beta}{\sum_{i \in \beta} \beta_i} \tag{2.52}$$

As done for the weights of the trajectory reward function, we normalized the weights of the LLE reward as well.

2.6 Testing

After the agent has been trained on a given number of episodes, we stop the training phase and we move onto the testing. During testing, the agent has to reach the goal autonomously using an ϵ-greedy policy in order to reach the goal position.
For each test episode we save the total reward the agent got while trying to reach the goal and we store the trajectory it executed. Once we have the trajectory, we plot it in order to see whether it is qualitatively acceptable and if the agent actually reached the goal autonomously. As can be seen in Fig 2.3, we plot the whole structure of the LLE, with a bigger focus to the moving leg. The arrows describe the trajectory taken by the foot in order to reach the goal position. As it can be seen in Fig 2.3, the trajectory looks smooth, because the generated trajectory by the agent is smoothed using a B-spline interpolation, which will be explained in section 2.7.

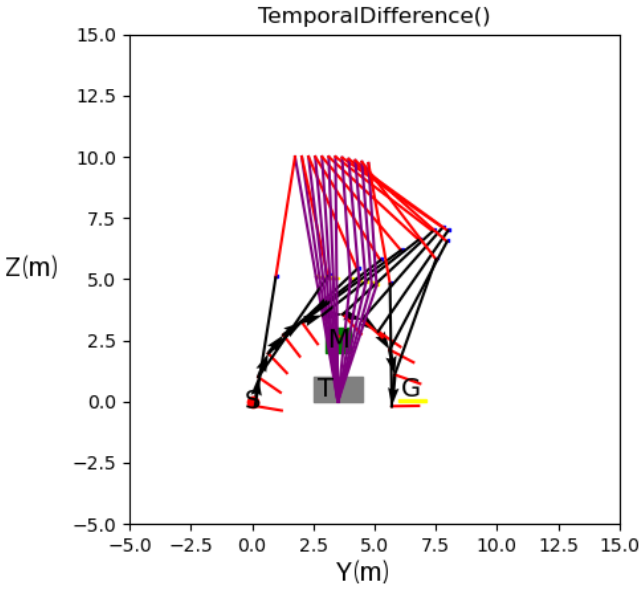

Figure 2.3: Example of generated trajectory during test

2.7 Trajectory Interpolation

Trajectory interpolation is used for two purposes: the first one is to smooth the agent trajectory in order to avoid sudden changes while executing it; the second one is to calculate the whole hip trajectory knowing the hip starting and finishing positions.

As anticipated in the previous section, we used a B-spline interpolation to achieve these two goals. As it can be seen in Algorithm 5, the interpolation is calculated as follows: where points is a $2 \times n$ matrix, and it contains

Algorithm 5 B-Spline Interpolation

Input: points, n=100, k=3, s=0.5
$x\_point \leftarrow points[:, 0]$
$y\_point \leftarrow points[:, 1]$
$tck \leftarrow splprep([x\_points, y\_points], k, s)$
$u \leftarrow linespace(0, 1, n)$
$x\_int, y\_int \leftarrow splev(u, tck)$
return list(x_int, y_int)

the list of points we want to interpolate, n is the number of points we want in our interpolated trajectory, k is the degree of the spline, i.e., the grade of the function we want to use (for example k=3 represents a cubic function). s is the smoothing factor (for example s=0 means that the spline will interpolate all the points).

Chapter 3

Experiments and Results

In order to understand the results of the model we crafted, we run the experiments in two different environments: a plotting environment to understand if the generated trajectory qualitatively makes sense; a simulated environment where we run simulations of the exoskeleton in order to see the execution of the proposed trajectory in a more complex environment and to see the behavior of the knee and the hip as well.
The Robot Operating System (ROS) is a framework used in the robotics community due to its incredible advantages. It is possible to use other developers code easily without the need to adapt or rewrite the code from zero. That is due to the fact that ROS uses software modules in the form of packages, allowing the communication between them using a a structure based on nodes that can publish or subscribe to other nodes. Each node or package can be coded in a different programming languages, giving priority to C++ and python as baseline coding languages.

3.1 Experiment Setup

The experiments we carried were divided in two categories: the ones were the agent was trained over 10000 epochs and the ones were the agent was trained over 25000 epochs. Then, for each of the two categories we decided to test the behavior of the agent when the position of the pivot foot changed. We decided over 3 different pivot foot positions:

- The pivot foot and the agent are aligned alongside the x axis, so they

both have same starting y coordinate;

- The pivot foot is aligned alongside the x axis with the center of the obstacle;

- The pivot foot is between the agent starting position and the center of the obstacle.

During training the agent is trained over three different kind of obstacles with different heights and widths, as reported in Table 3.1. During testing, we introduce also two new different kinds of obstacle, with completely different shapes and different dimensions compared to the ones used in training. The measurements reported in Table 3.1 are the ones from the

Name	Width (cm)	height (cm)	Training/Testing
Base 1	20	10	Training
Base 2	10	15	Training
Base 3	5	20	Training
Triangle	20	20	Testing
Half-circle	20	25	Testing

Table 3.1: List of obstacles used during the experiments

simulated environment since the majority of the testing has been carried out in a simulation.
During the testing phase, the results of the trained DRL model are compared with the results of the Collision-Free Foot Trajectory Generator

(CFFTG) from [11] in terms of execution time, time needed to generate the trajectory and correctness of the trajectory.

3.2 Experiments

As stated in the previous section, we divided the experiments into two macro categories and for each of them we run 3 different scenarios. For the 10000 epochs we named the experiments from **(a)** to **(c)**, while for the 25000 epochs we named the experiment from **(1)** to **(3)**. For all experiments, we considered the average human step length to position the final goal position, i.e., the step length is 70 cm.

3.2.1 Experiments

For experiment **(a)** the initial configuration of the LLE and the agent is visible in Figure 3.1. The positions of the pivot foot and the agent are

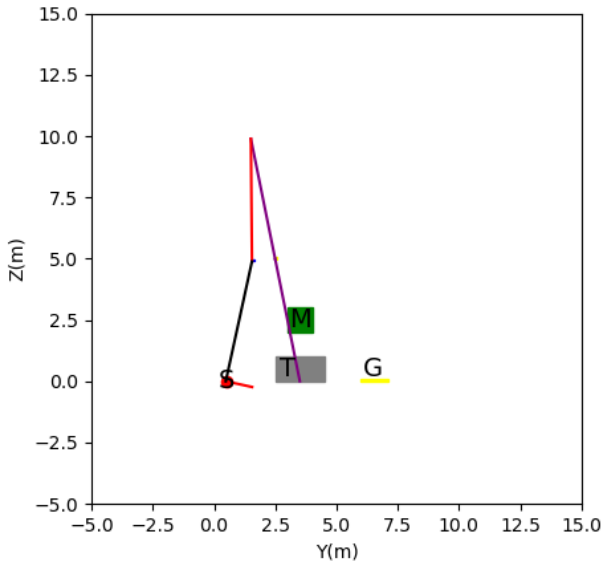

Figure 3.1: 10000 epochs LLE initial configuration of experiment (a)

shifted. That is, the pivot foot is aligned with the center of the obstacle, while the agent is positioned in the origin of the sagittal plane.
For experiment **(b)** the initial configuration of the LLE and the agent is visible in Figure 3.2. The positions of the pivot foot and the agent are the same alongside the y and z directions. That is, the pivot foot is aligned

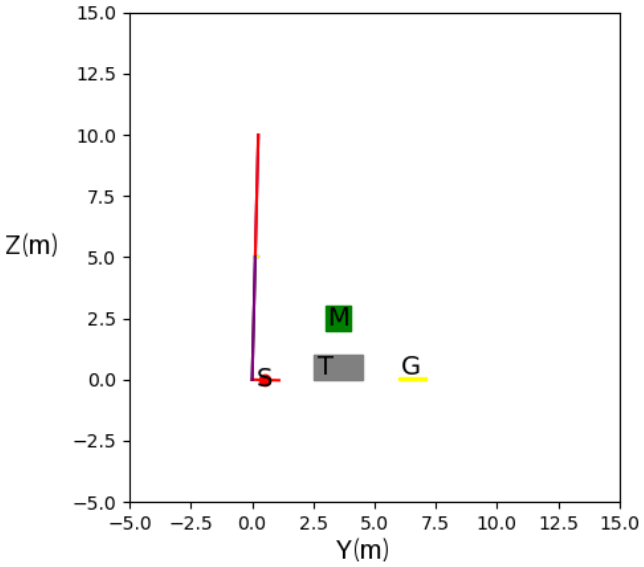

Figure 3.2: 10000 epochs LLE initial configuration of experiment (b))

with the agent in the origin of the Sagittal plane and they lie on the same line alongside the x direction.

For experiment **(c)**, the initial configuration of the LLE and the agent is visible in Figure 3.3. The positions of the pivot foot and the agent are

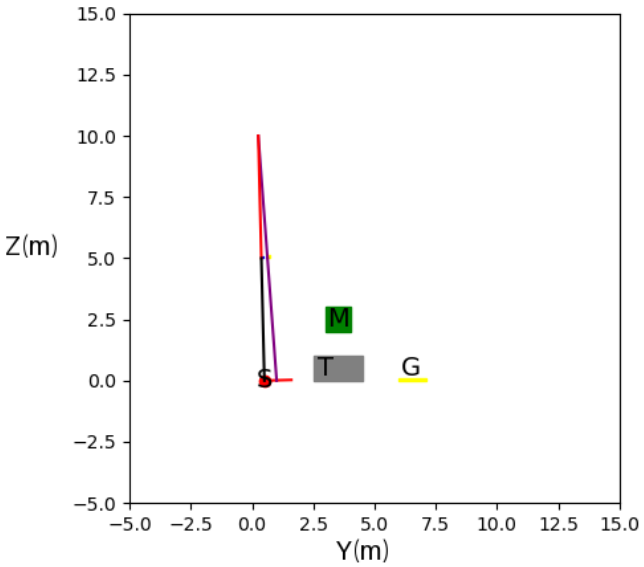

Figure 3.3: 10000 epochs LLE initial configuration of experiment (c)

shifted. The pivot foot is placed between the agent and the obstacle, while the agent is positioned in the origin of the sagittal plane.

As done for experiment **(a)**, in experiment **(1)** the initial position of the

agent and the pivot foot are shifted, with the latter being aligned with the center of the obstacle alongside the x direction.
As for experiment **(b)**, in experiment **(2)** the agent and the pivot foot are aligned alongside the x direction.
In experiment **(3)**, the agent and the pivot are not aligned, with the latter being positioned between the agent and the center of the obstacle.

3.3 Evaluated Metrics

To evaluate the results of the experiments we used as metrics the following:

- time elapsed to generate a trajectory, in ms;
- angle variations of the LLE joints, such as hip and knee angles;
- difference between generated trajectory of CFFTG [11] and the one generated by the model.
- Accuracy, defined as:

$$Accuracy = \frac{g - h}{g} \tag{3.1}$$

where g is the number of generated trajectories and h is the number of trajectories that go through the obstacle region.

3.4 Results

In this section, we show the results achieved from the 6 different experiments and the training trend of the reward of each of them. We then show also the results of each of the trained model if put in new conditions. In other words, we will show the scenario where, for example, the model trained in experiment **(a)** is used in the conditions of the model trained in experiment **(c)**. The experiments have been run on the following setup:

- CPU: Ryzen 7 7000 Series;
- GPU: Nvidia GeForce 4060 Laptop 8GB;

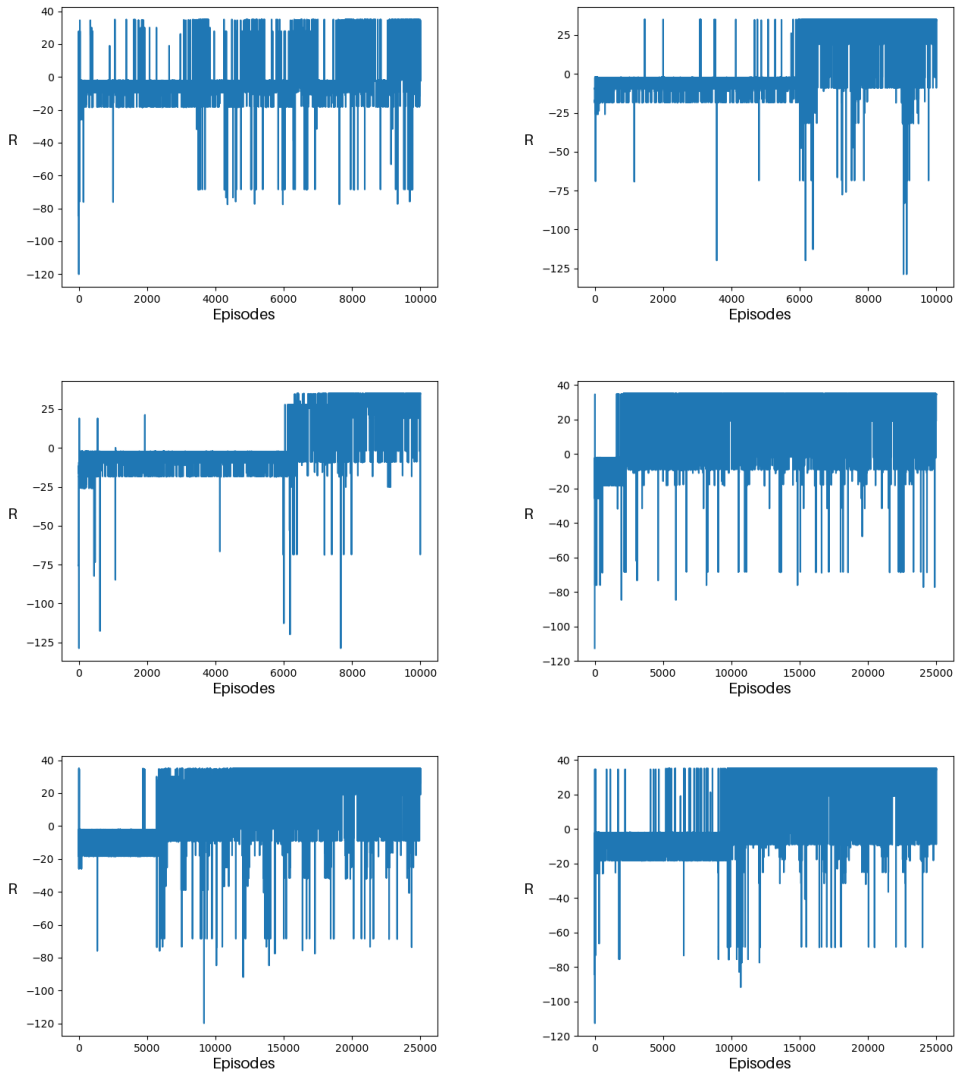

Figure 3.4: Training trends of experiments: (a), (b), (c), (1), (2), (3)

- RAM: 16GB DDR5;

The training trends shown in Figure 3.4 have an increasing trend, converging to the maximum reward achievable. As it can be seen from 3.4, the plot present positive and negative spikes. The negative spikes are related to the episodes where the agent hits the obstacle. When this happens, the agent gets a high penalization, leading to a low reward for that episode. The positive spikes tends to emerge more frequently through the episodes, and they represent the episode in which the agent took the optimal number of steps and reached the goal. As it can be seen from Table 3.2, the Average Generation Time for each model, considering all 5 kinds of obstacle discussed in Table 3.1, is of the order of 2 ms, exception made for the model of experiment **(2)**.

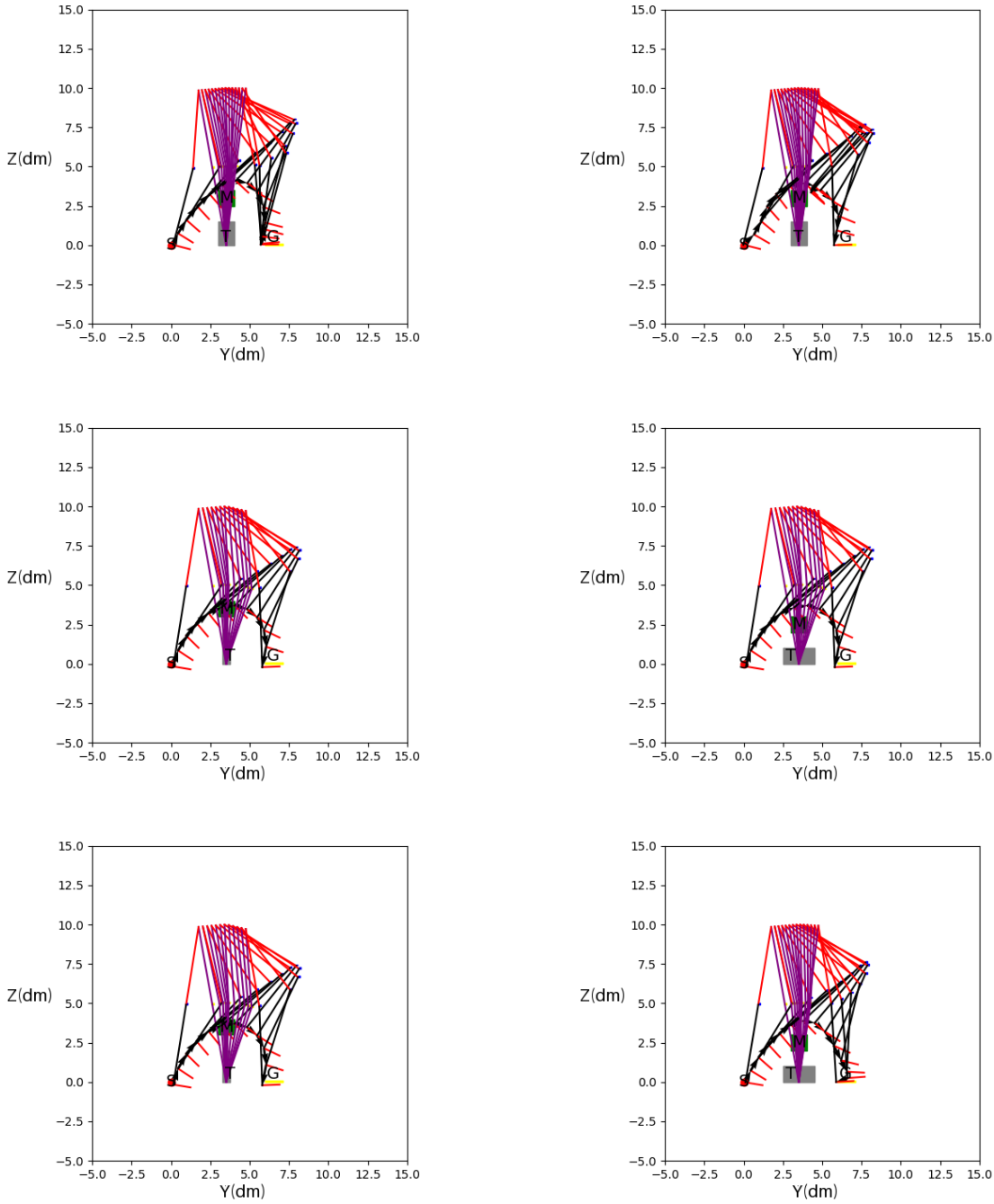

Figure 3.5: Trajectories generated by experiment (a) on the training samples

Table 3.2 also reports the Average Joint Angles formed by the knees and the hip during the generation of the trajectories. This metric is important to evaluate the trajectories because small angles variations means that the trajectory is less costly during execution and is aligned to a natural step behavior. To further explain the Average Joint Angles metric, we sampled randomly some generated trajectories to analyze how the joints angles varies during each test episode. What we expect is that the the an-

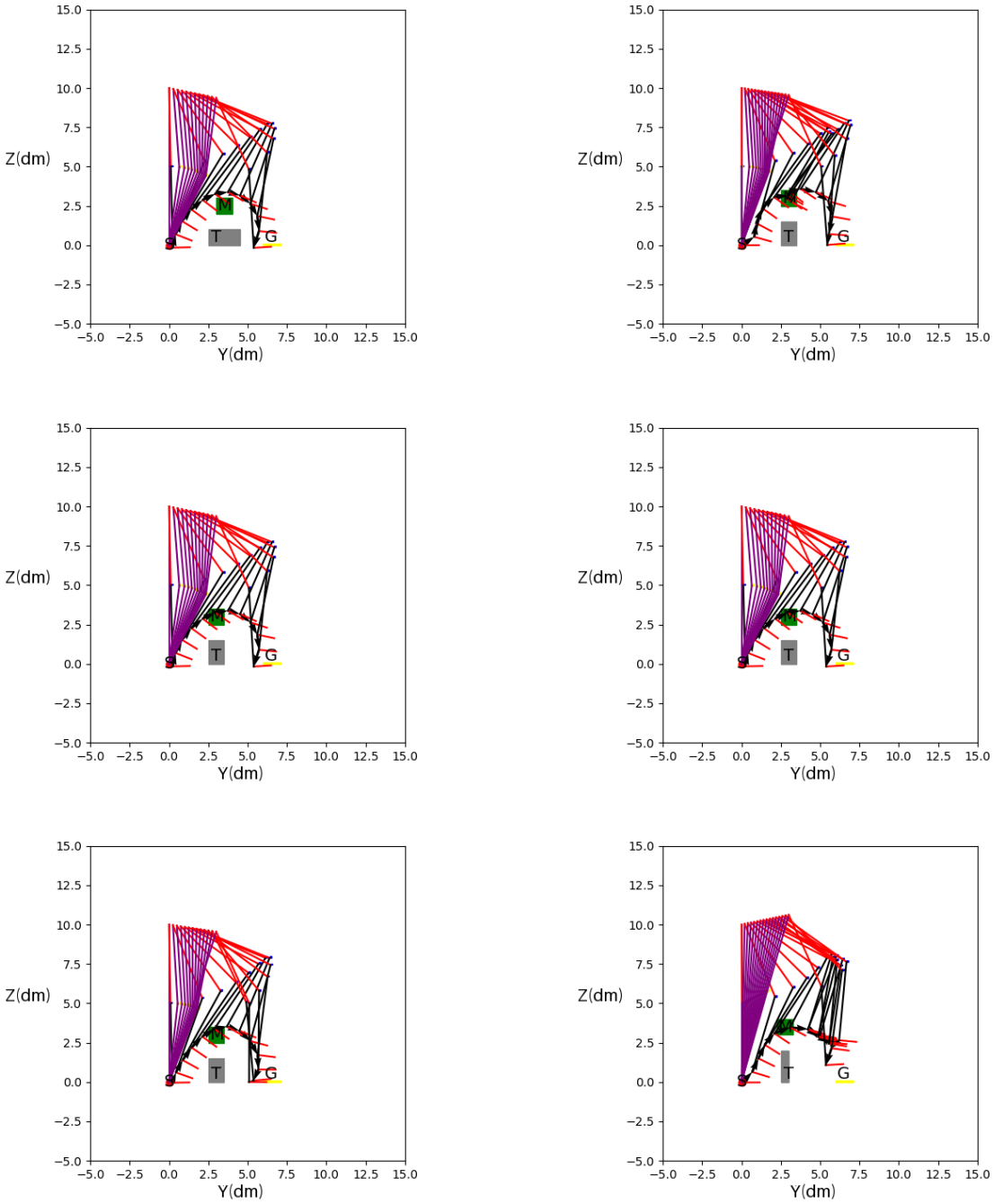

Figure 3.6: Trajectories generated by experiment (b) on the training samples

gles do not oscillate too much in a wide range of values, but that they stay inside a small interval. As shown in Figure 3.11, during experiment **(a)**, the angle of the knee in the agent leg varies uniformly between states. As it can be seen from the first plot in Figure 3.11, the angle variation between states is small. That means, that the knee angle increases, or decreases, with slight variations between the states of the episode, which is exactly what we want. On the other hand, as it can be seen from the second plot

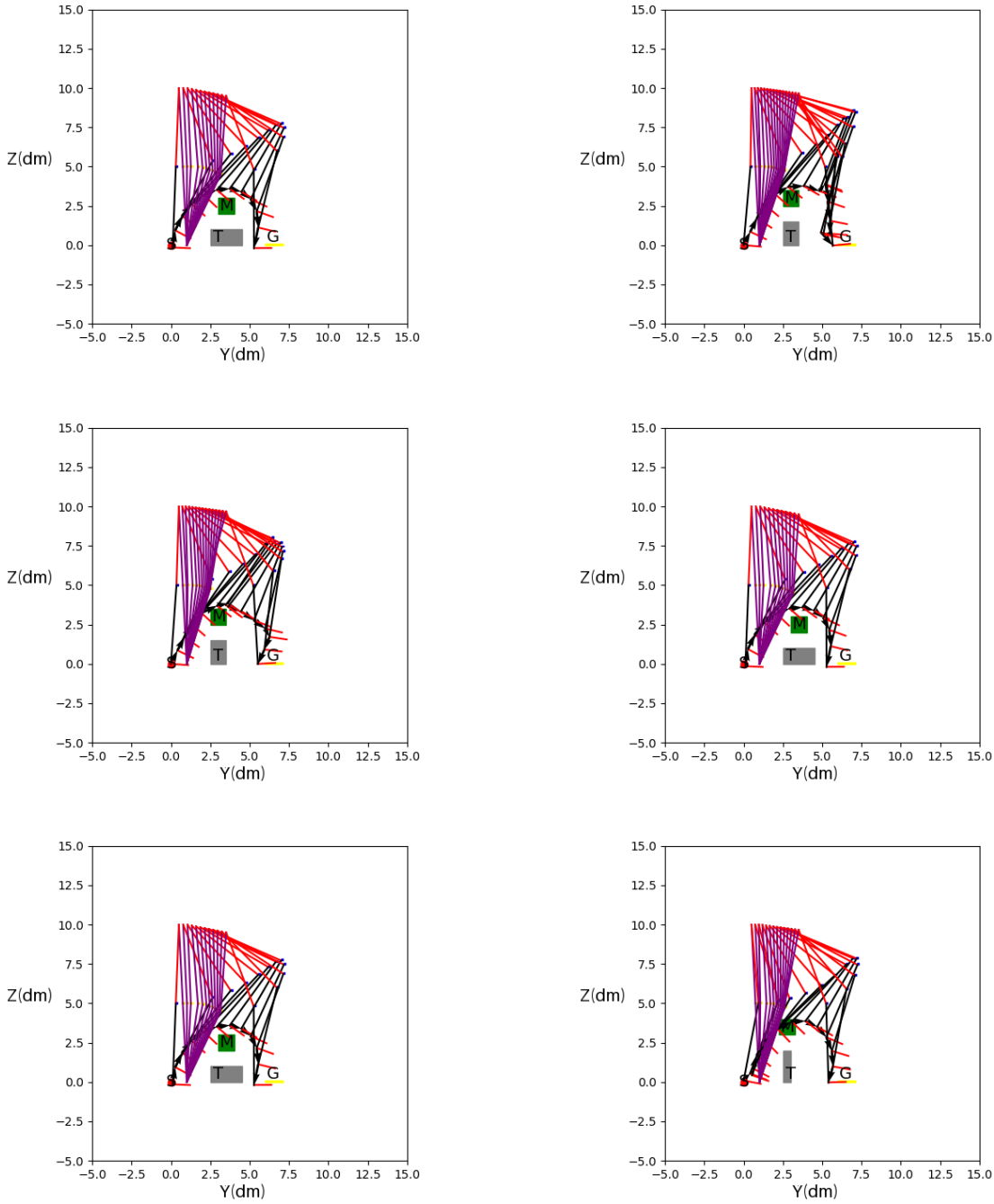

Figure 3.7: Trajectories generated by experiment (c) on the training samples

of Figure 3.11, there are some cases where the angle varies really quick between states. This is also expected since, being a learned model, there can be cases where the agent does not take the most natural action. That is visible, for example, in the second plot of Figure 3.11, in the iterations between 16 and 18.

The concave shape of the plot reflects the behavior of the knee angle while taking over the obstacle. That is because while raising the leg, to pass the

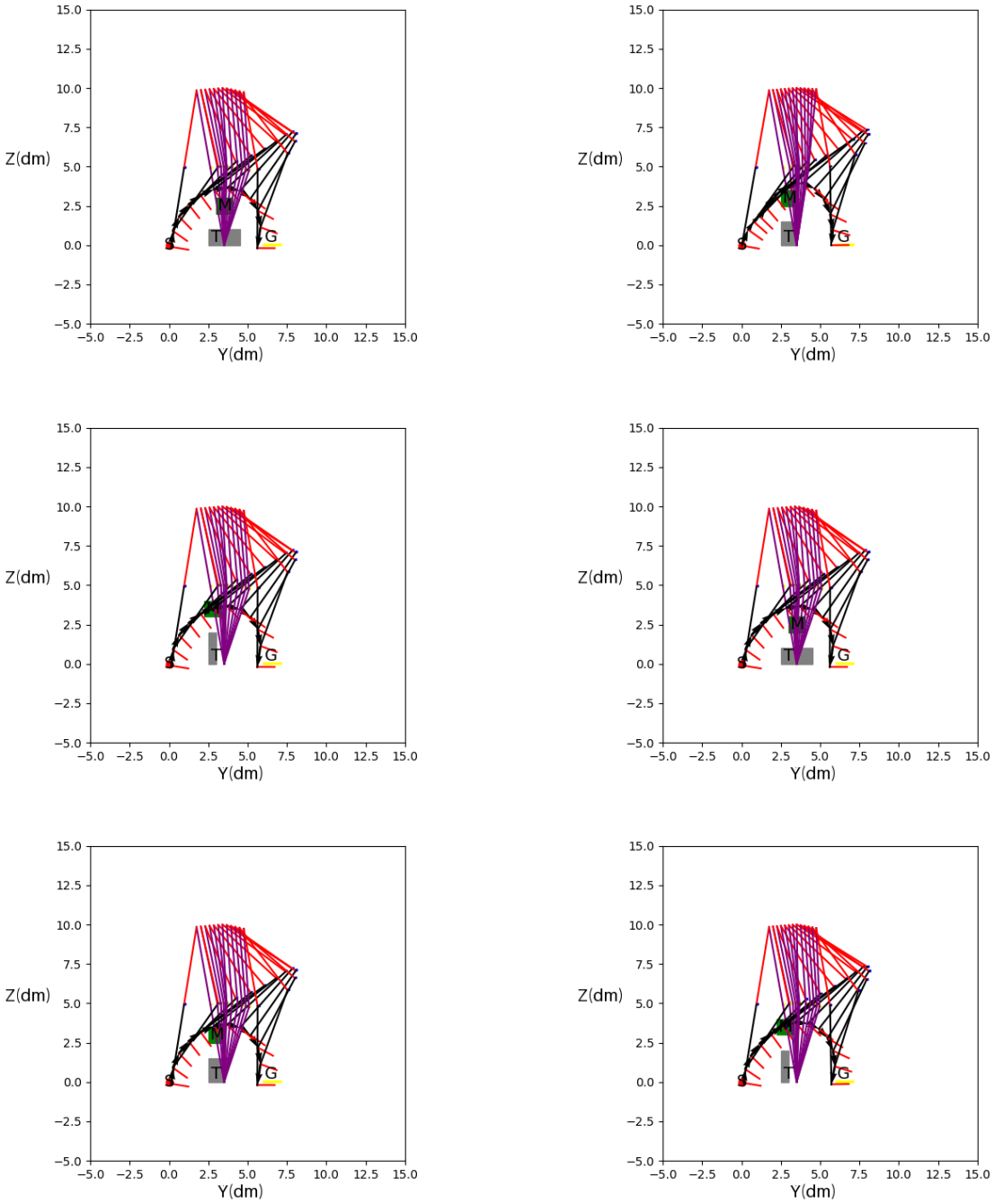

Figure 3.8: Trajectories generated by experiment (1) on the training samples

obstacle, the knee is flexed, meaning a decrease in the angle. When the agent passes over the obstacle, the knee is extended, meaning that the angle increases. As it can be seen from the experiments **(a),(b),(c),(1)** and **(3)**, the hip angle varies without sudden changes and they follow a natural behavior: the hip extends when over taking the obstacle, so the angle increases, while it is flexed when the obstacle is avoided, making the angle decrease. The only exception is from experiment **(2)**, where it can be

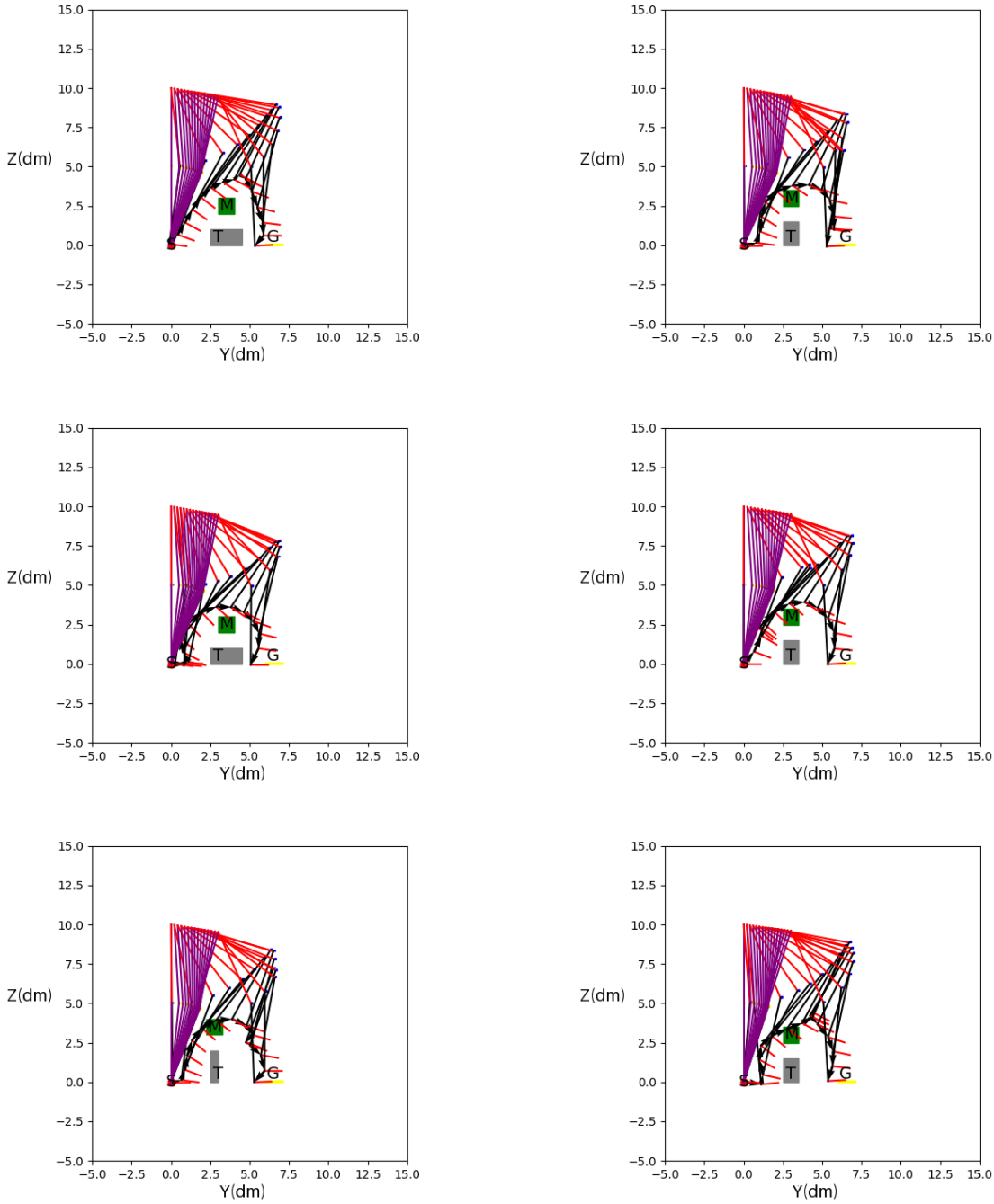

Figure 3.9: Trajectories generated by experiment (2) on the training samples

seen that the angle only increases. That is because the trajectories tends to be flat, as earlier stated. Meaning that the LLE only moves the agent leg such that the hip angle only increases.

Now, we discuss the results of all 6 models in the other conditions, to see which one of the proposed ones adapts the best to different configurations without earlier training. I have decided to opt for this option in order to recreate the same methodology used in DL to evaluate the results

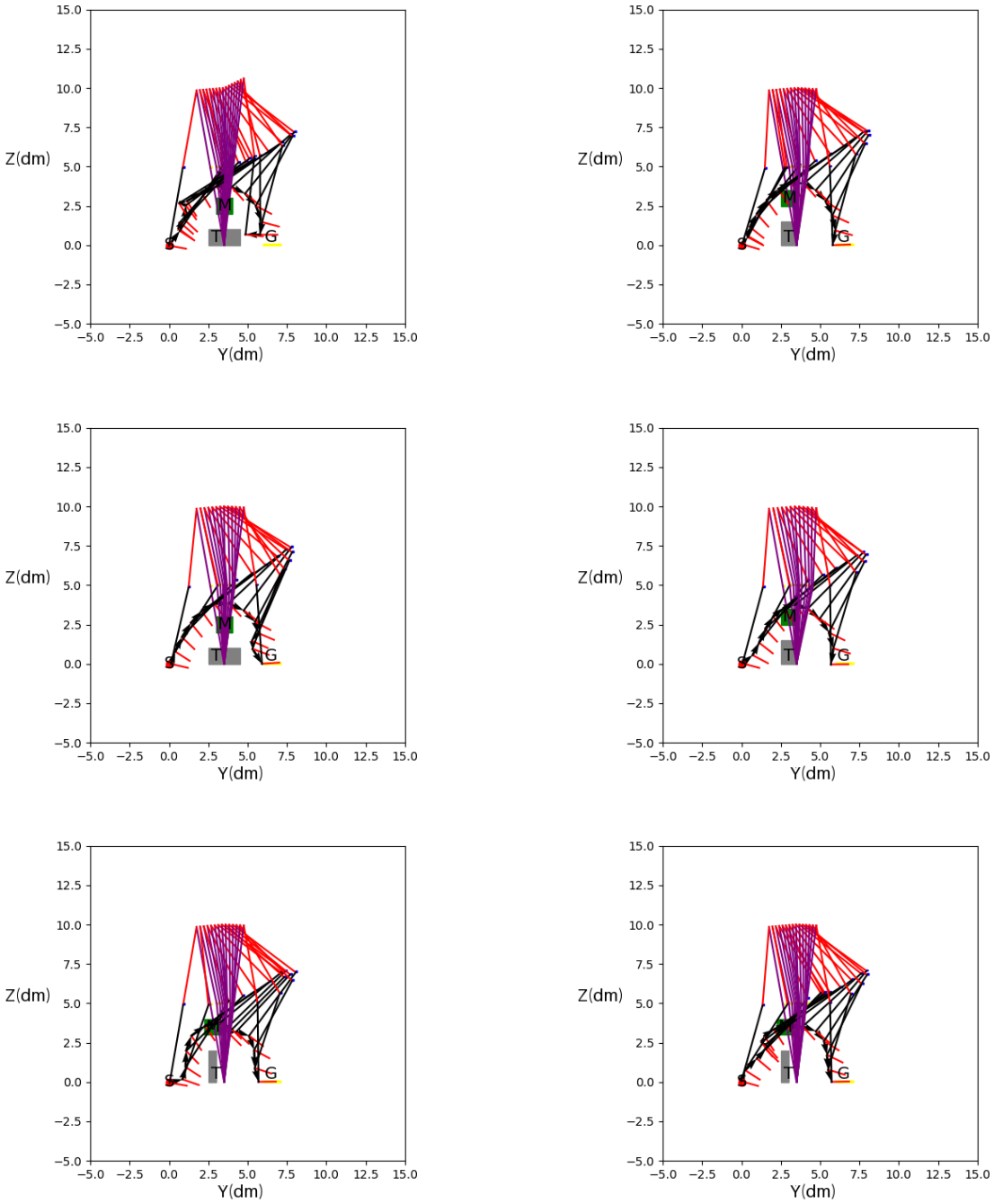

Figure 3.10: Trajectories generated by experiment (3) on the training samples

of the models. The methodology I am referring to is the validation of the model. To better understand the strengths and weaknesses of each singular configuration, I have firstly decided to do a separate analysis for each of the configurations. As it can be seen from Table 3.7, experiment **(2)**, the Average Generation Time is really low, that is, because as stated previously, the agent, in this specific model, prefers to go through the obstacle region. That can also be seen from the average knee and hips angle, as

Name	Average Generation Time (ms)	Average Joint Angles [knees, hip]	Number of Test Episodes
(a)	2.46	$[-51.031, 35.363]$	1000
(b)	2.03	$[-52.995, 33.744]$	1000
(c)	2.12	$[-49.407, 33.0341]$	1000
(1)	2.50	$[-50.483, 33.729]$	1000
(2)	1.21	$[-10.544, 14.458]$	1000
(3)	2.64	$[-49.462, 33.703]$	1000

Table 3.2: Tabular results of the six experiments

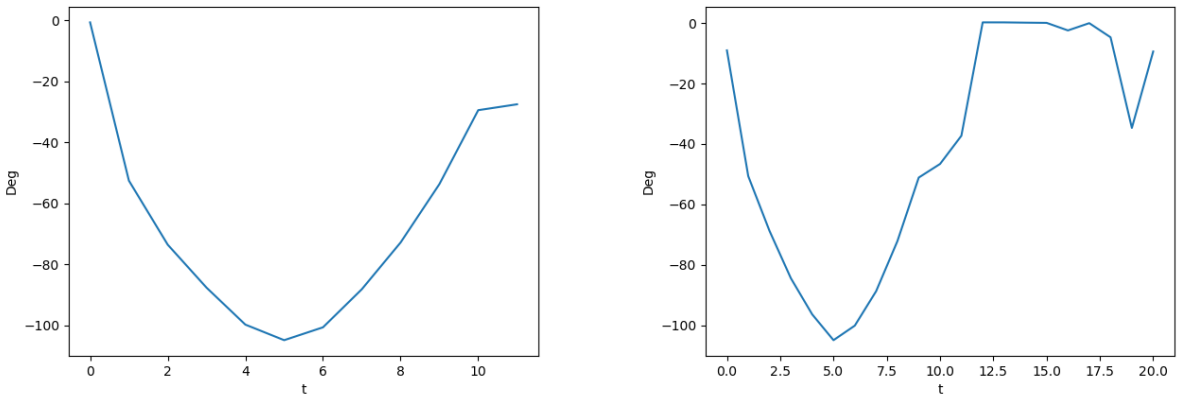

Figure 3.11: Average Joint variation of the knee angles of experiment (a)

shown in Figure 3.27, where the plots tend to increase or decrease only, meaning that the agent is only moving the hip, either flexing it (first plot), or extending it (second plot). In other words, the agent is not overtaking the obstacles but just moving the limbs without bending the knees.
The other 5 experiments tend to deal well with the new conditions pro-

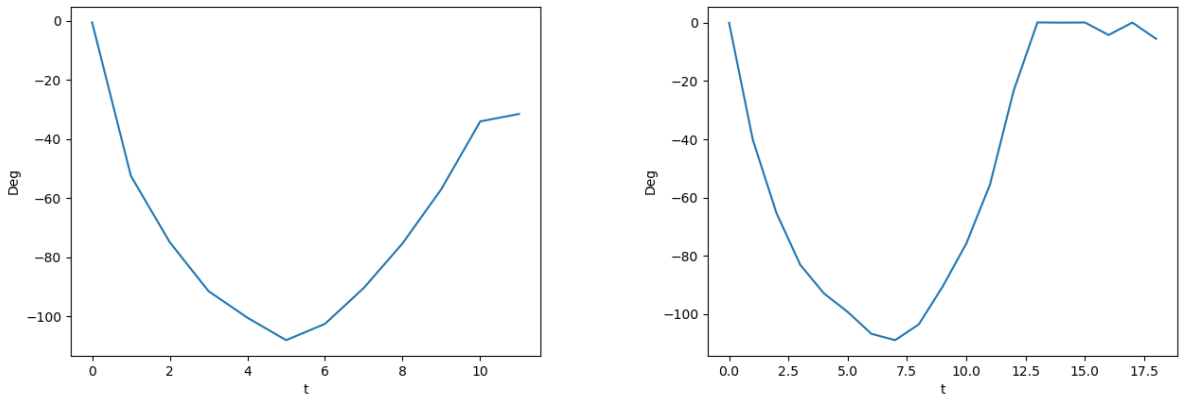

Figure 3.12: Average Joint variation of the knee angles of experiment (b)

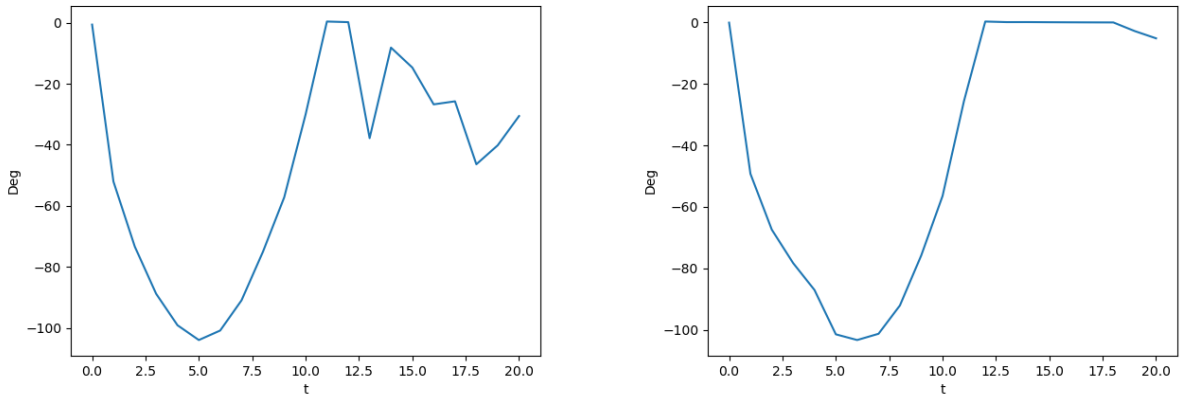

Figure 3.13: Average Joint variation of the knee angles of experiment (c)

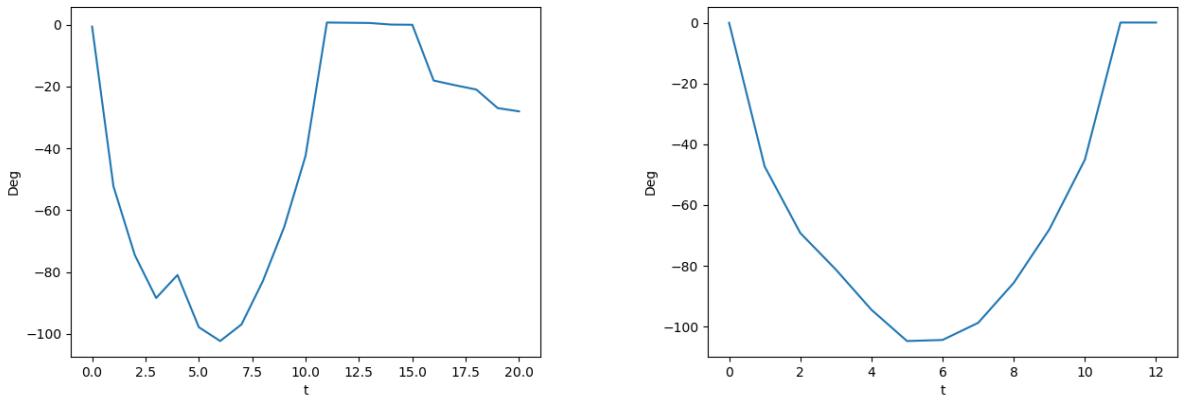

Figure 3.14: Average Joint variation of the knee angles of experiment (1)

posed, without having one of them standing out particularly compared to the others. In general, the models trained over 10000 epochs tend to deal better than the ones trained over 25000 epochs. This could be related to the fact that the 10000 epochs model have generalized better the environ-

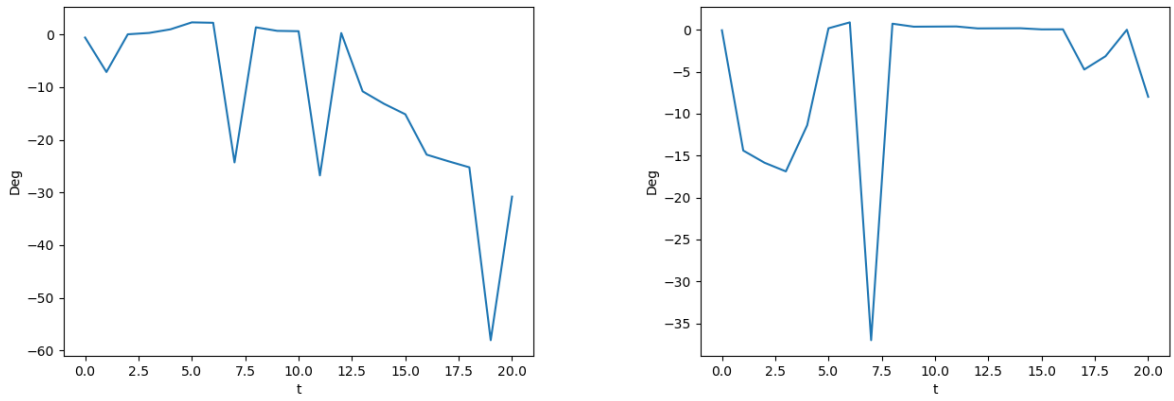

Figure 3.15: Average Joint variation of the knee angles of experiment (2)

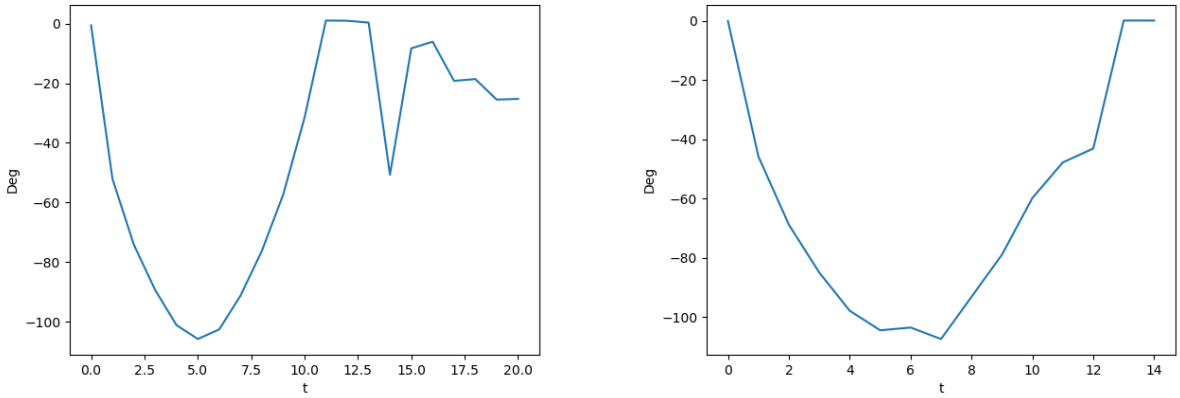

Figure 3.16: Average Joint variation of the knee angles of experiment (3)

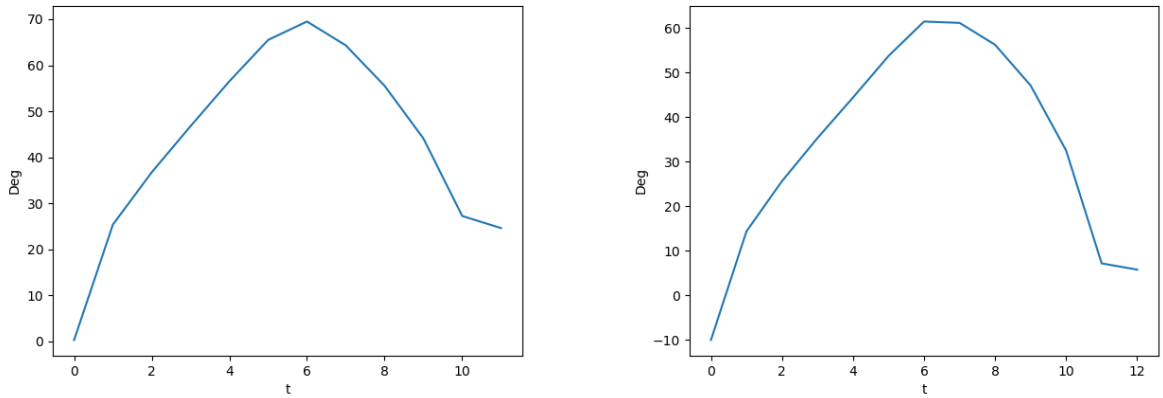

Figure 3.17: Average Joint variation of the hip angle of experiment (a)

ment, while the 25000 might have learned more specifically theirs, having less generalization power. The final model I trained and opted for has the following characteristics:

- seeing the training trend from the previous models according to the

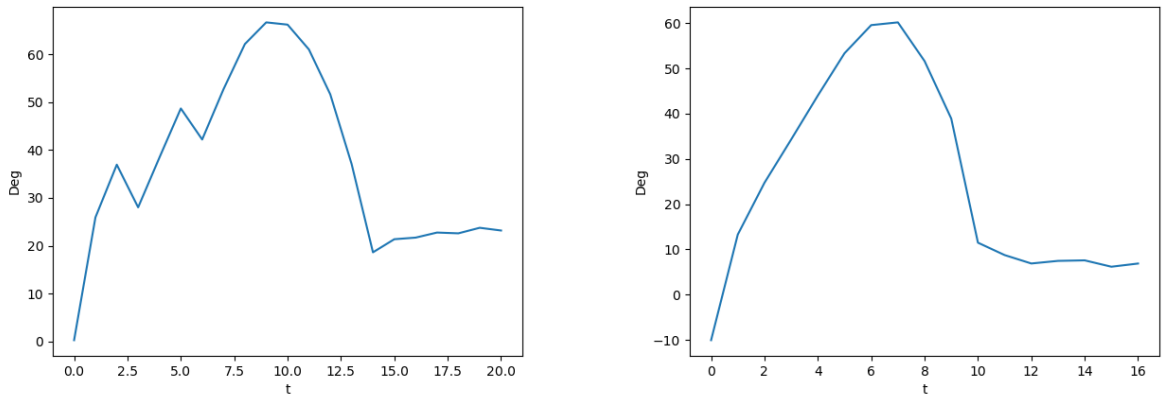

Figure 3.18: Average Joint variation of the hip angle of experiment (b)

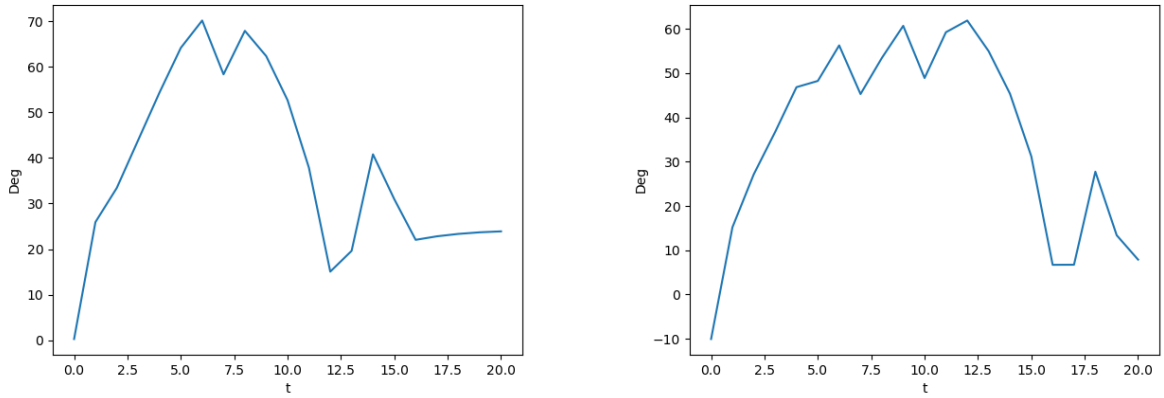

Figure 3.19: Average Joint variation of the hip angle of experiment (c)

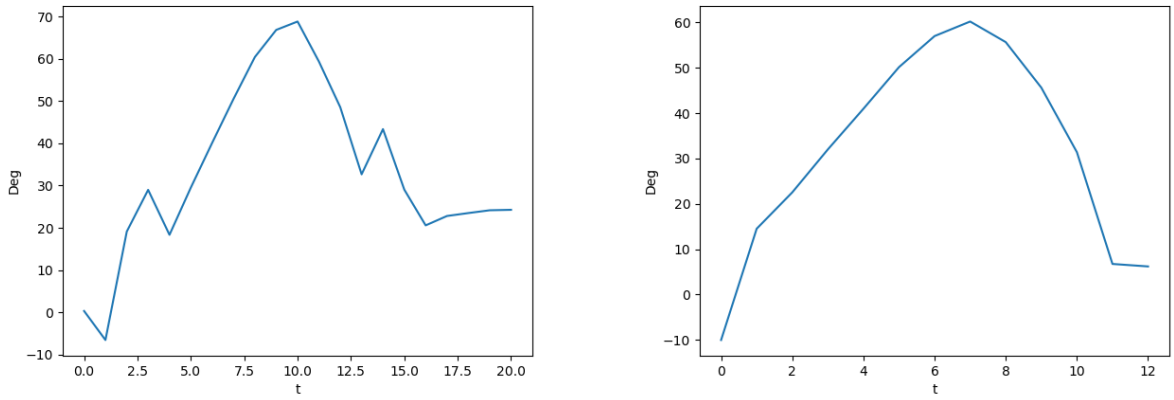

Figure 3.20: Average Joint variation of the hip angle of experiment (1)

initial configurations of the pivot foot, and how the model tends to forget what it learned, the final model has been trained over 50000 episodes;

- to introduce more variety and more complexity, the final model has

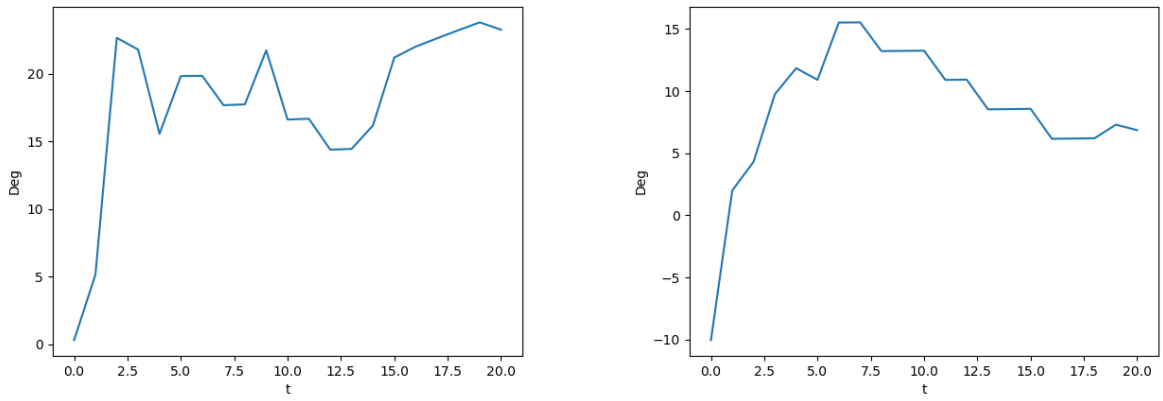

Figure 3.21: Average Joint variation of the hip angle of experiment (2)

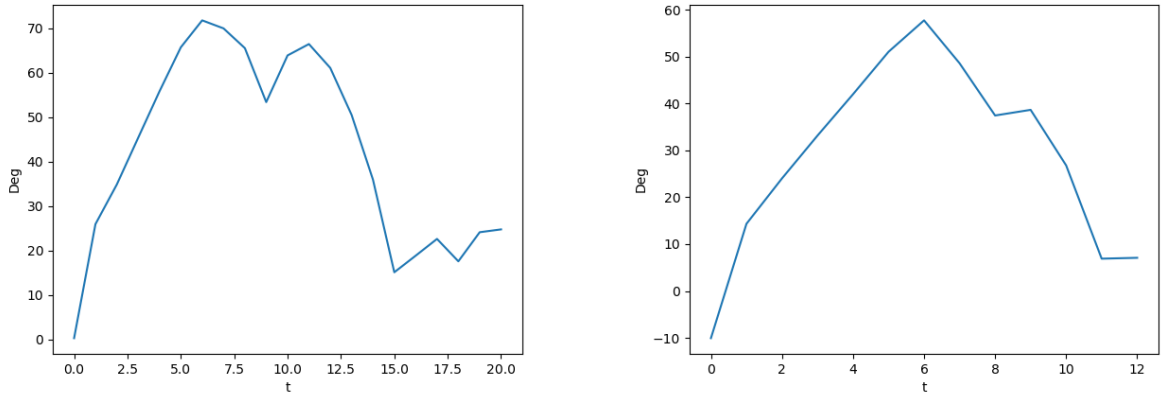

Figure 3.22: Average Joint variation of the hip angle of experiment (3)

	pivot aligned with agent	pivot not aligned
Average Generation Time (ms)	2.53	2.30
Average Knee Angle	-47.175	-52.136
Average Hip Angle	41.131	39.453

Table 3.3: Configurations for experiment **(a)**

been trained with 6 different starting scenarios: 3 of them are the scenarios regarding the pivot foot as done in the 6 previous experiments;

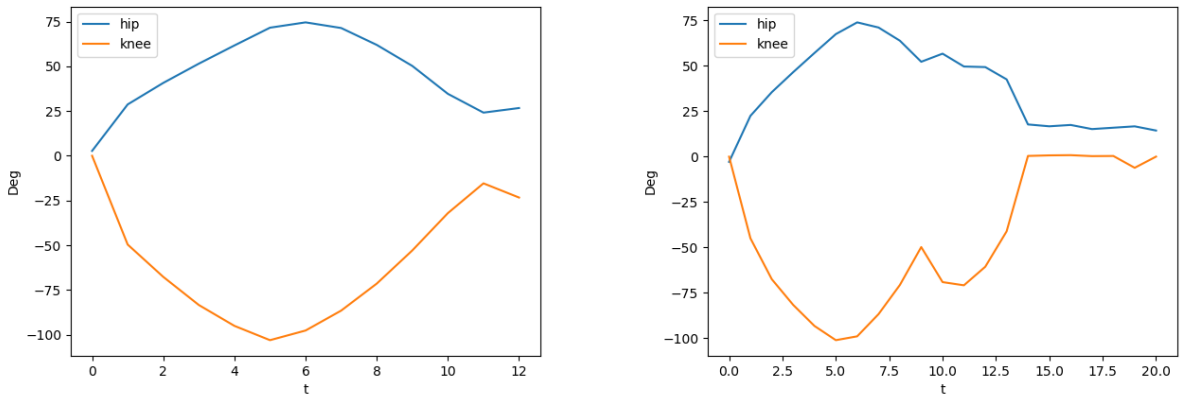

Figure 3.23: Angular variations of joint angles of experiment **(a)** in condition (b) and (c)

	pivot aligned with obstacle	pivot not aligned
Average Generation Time (ms)	2.28	2.27
Average Knee Angle	-52.334	-56.731
Average Hip Angle	29.733	38.730

Table 3.4: Configurations for experiment **(b)**

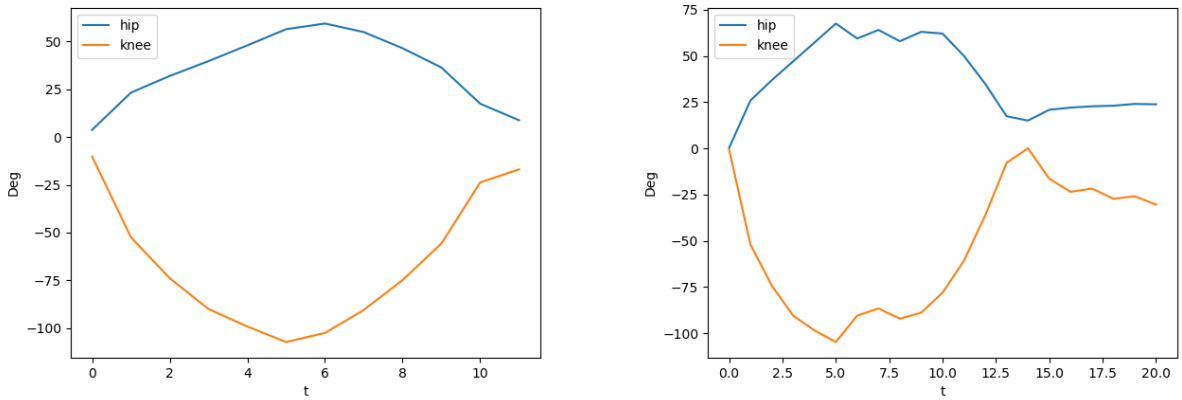

Figure 3.24: Angular variations of joint angles of experiment **(b)** in condition (a) and (c)

while the other 3 configurations are related to the agent. In other words, we change the starting position of the agent by going closer to

	pivot aligned with obstacle	pivot aligned with agent
Average Generation Time (ms)	2.55	2.06
Average Knee Angle	-46.396	-52.572
Average Hip Angle	28.419	41.442

Table 3.5: Configurations for experiment **(c)**

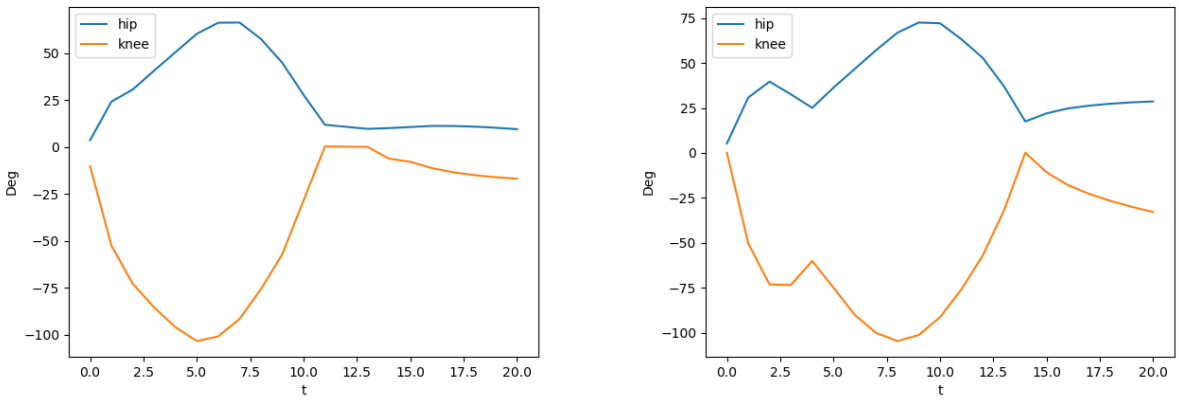

Figure 3.25: Angular variations of joint angles of experiment **(c)** in condition (a) and (b)

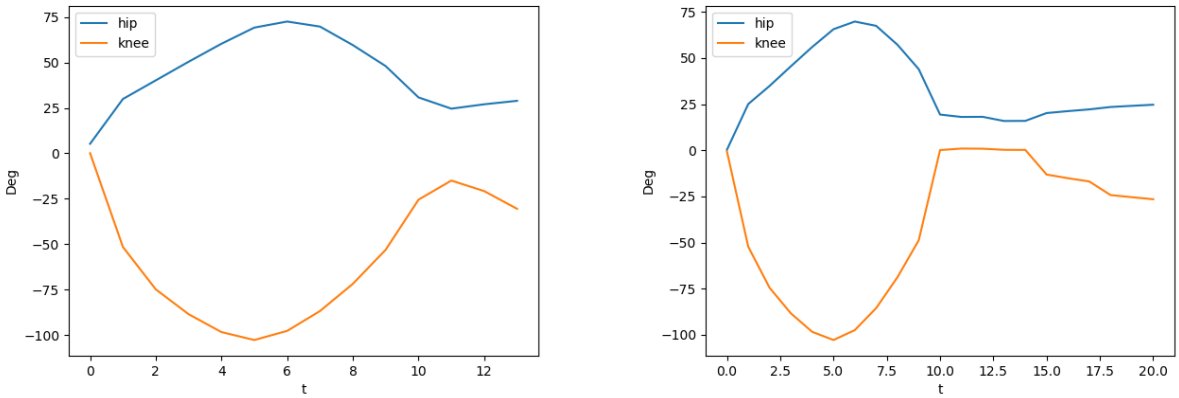

Figure 3.26: Angular variations of joint angles of experiment **(1)** in condition (2) and (3)

the obstacle;

- the obstacle used for training and testing are the same used for the

	pivot aligned with agent	pivot not aligned
Average Generation Time (ms)	2.46	2.19
Average Knee Angle	-51.204	-51.066
Average Hip Angle	41.345	37.621

Table 3.6: Configurations for experiment **(1)**

	pivot aligned with obstacle	pivot not aligned
Average Generation Time (ms)	2.49	1.23
Average Knee Angle	-8.377	-11.675
Average Hip Angle	11.367	16.849

Table 3.7: Configurations for experiment **(2)**

previous 6 models;

- the model has been tested over 1000 episodes. In other words, we generated 1000 trajectories with different obstacles and configurations.

As it can be seen from Figure 3.29, the trajectories are smooth and compliant to the type of obstacles, even thought some of them might hit the obstacle. After a qualitative check, the majority, around 94.7% of the generated trajectories do not hit the obstacle. In other words, out of the 1000 generated trajectories, 53 go through the obstacle region.
Now I will report, in Table 3.10 the accuracy of all the models and con-

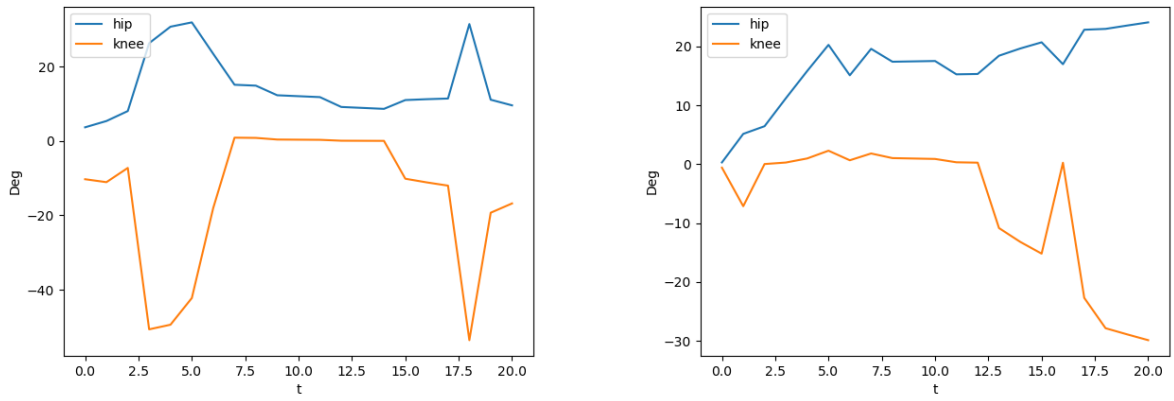

Figure 3.27: Angular variations of joint angles of experiment **(2)** in condition (1) and (3)

	pivot aligned with obstacle	pivot aligned with agent
Average Generation Time (ms)	2.14	2.52
Average Knee Angle	-49.266	-49.424
Average Hip Angle	29.942	41.001

Table 3.8: Configurations for experiment **(3)**

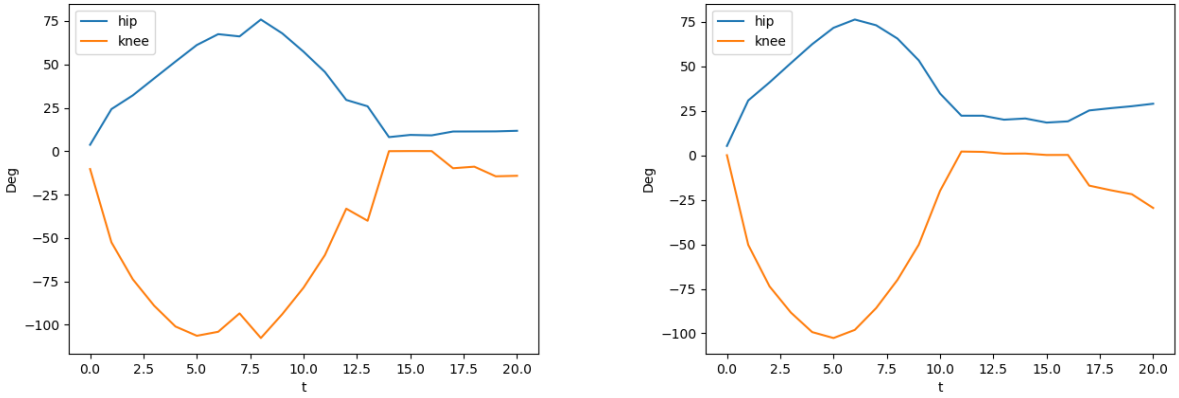

Figure 3.28: Angular variations of joint angles of experiment **(3)** in condition (1) and (2)

front them with the final model. From Table 3.10, we can see that the Final model has the highest accuracy. The increment in the accuracy compared

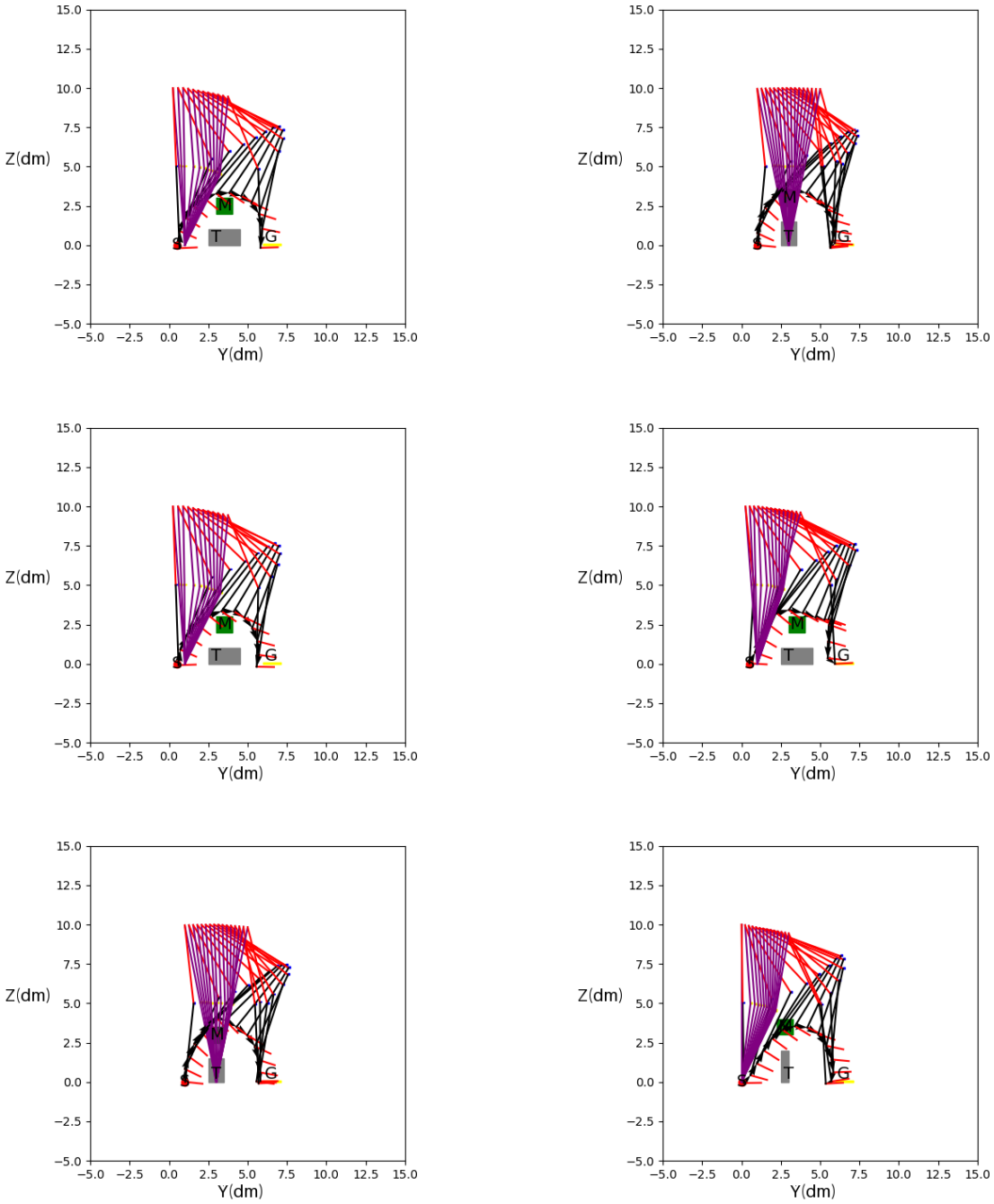

Figure 3.29: Trajectories generated by the final model

Average Generation Time (ms)	Average Knee Angles	Average Knee Angles
2.31	-56.604	37.532

Table 3.9: Tabular results of the final model

to the other models can be determined from the fact that the model has seen more initial configurations and it became more compliant to different

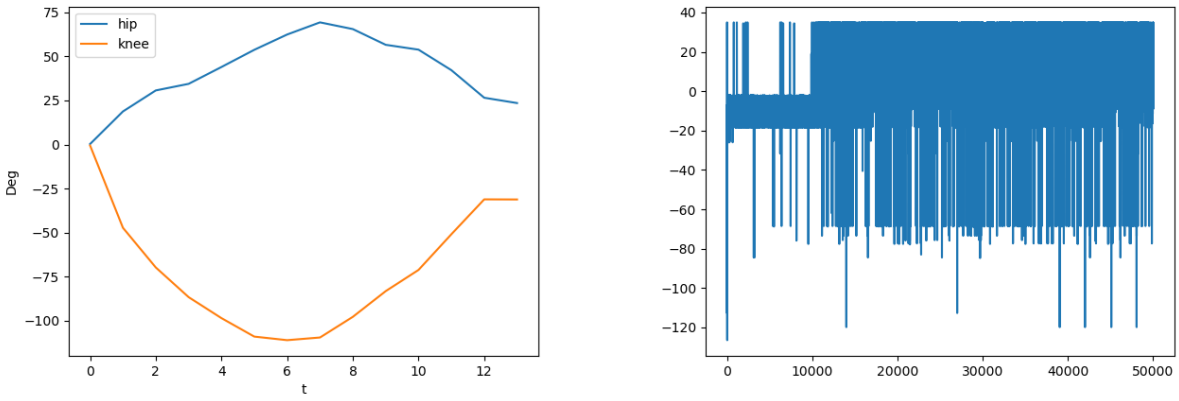

Figure 3.30: Average Joint variation of the hip and knee angles and training trend of the final model

scenarios.

It can also be seen from Figure 3.30 that the behavior of the hip angle and the knee angle follows the expected behavior, that is, the hip is extending and then flexing before and after, respectively, passing over the obstacle, while the knee is doing the opposite.

Now, I am going to report in Table 3.11 the comparison between the CFFTG [11] and the Final Model in order to understand if the proposed project can perform at least as the previous version. As it can be seen, the CFFTG has a generation time lower than the Final Model's one. This topic is gonna be discussed later on in the Discussion chapter.

Name	Accuracy (%)
Final Model	94.7
(a)	58.2
(b)	75.8
(c)	74.1
(1)	56.9
(2)	3.8
(3)	58

Table 3.10: Comparison of the final model with the 6 experiments

Obstalce	CFFTG Generation Time	Final Model Generation Time
Base 1	2.055 ms	3.001 ms
Base 2	1.974 ms	3.028 ms
Base 3	2.003 ms	3.002 ms
Triangle	1.982 ms	3.353 ms
Half-circle	1.958 ms	3.001 ms

Table 3.11: Comparison of the Final Model with the CFFTG

Chapter 4

Discussion

On a qualitative basis, experiments **(a),(b)** and **(c)** have a more natural and aligned behavior with respect to the hypothesis, i.e., generating a trajectory of the foot of the LLE that is aligned with a physiological step. On the other hand, these 3 experiments are subject to more errors in trajectory generations. Some of the trajectories have strange behaviors, such as moving slightly backwards or oscillating. This can be related to the number of episodes used for training.

While training the models, there was an important difference between the models trained over 10000 episodes and the ones trained over 25000. For the latter ones, it was noticeable that the agent preferred, in the long run, to hit the obstacle more than to take extra steps, making some of the trajectories going through the obstacle region. Taking this into account, it is reasonable that the Average Generation Time is shorter than in the other case. This situation is mainly present in experiment **(2)**, where the majority of generated trajectories suffers from the problem discussed earlier.

The experiments **(1),(2)** and **(3)** have a smoother trend on average, as shown in Figures 3.14, 3.15 and 3.16, but there are trajectories where the swing knee's angle varies significantly, as it can be seen in the second plot of Figure 3.15. That is related, as explained previously, to the fact that in the long run, the agent will start to prefer to make shorter trajectory through the obstacle region than making a longer trajectory.

As it can be seen specifically from the plots in Figure 3.15, the shape of the functions are in a V shape cause the majority of the generated trajectories

are flat, making the knee angle vary in a small interval of values.
As it can be seen from the 6 experiments proposed and from the Final Model, it is possible to train an agent with a DRL approach in order to generate trajectories. The single models pointed out the best environment and conditions for the agent to be trained. For example, the models **(b)** and **(2)** showed that training the agent by always considering it aligned with the pivot foot is not optimal in the long run, especially after 10000 episodes of training. Training the agent in different starting configurations of the agent and the pivot foot, and with different obstacles allowed us to obtain a model compliant to the different scenarios with a success rate of 94.7%.
The average generation time of the Final Model is 3 ms on average, which is acceptable considered that the code is written in python, which is not efficient in terms of execution time. Compared to the CFFTG, which takes 2 ms on average to generate a trajectory, we can say that the Final Model is comparable in terms of performances. Another advantage of the Final model is that, since it is a learned model, it can try to generate trajectories of new obstacles, or obstacles that pose problems for the CFFTG in a more reliable way. On the other hand, it is more likely to generate wrong trajectories, making the LLE colliding with the obstacles. In other words, the CFFTG is more reliable to find a trajectory, since at the end of the execution it is gonna find one, meaning that it can take a considerable amount of time in the real world case scenarios. The Final model, is more subject to errors, but it can find trajectories in a faster and more consistent way, meaning that it should be able to always generate a trajectory in a acceptable amount of time.
With regards to the training performances, the models from experiment **(a),(b)** and **(c)** took on average 12 minutes $\pm$ 3 minutes to train, while the other 3 scenarios took on average 29 minutes $\pm$ 5 minutes. The final model took 55 minutes and 27 seconds to train. The increase of training time, given that the experiments **(a),(b)** and **(c)** and the Final model have been trained on the same amount of episodes, is related to the fact that both the agent position and the pivot foot position change over episodes, making

it more challenging for the agent to learn. In terms of generation time of the 1000 testing trajectories, the Final model employed 1.764 seconds to generate all of them after the training phase.

The generated trajectories have been made compliant with the real exoskeleton, taking into account the whole kinematic chain and the real measurements of the links. That is, the generated trajectories also take into account the position of the joints and links, making the proposed solution suitable for real world implementations.

Chapter 5

Conclusions

The aim of the thesis, and the solution I proposed, was to use DRL techniques in order to allow Lower-Limb Exoskeletons to autonomously avoid low obstacles in different contexts and conditions, allowing the generation of the trajectories to be compliant to the current environment and LLE state. The proposed approach allows the LLE to overcome some of the limitations found in the CFFTG.

We can say that the proposed solution has been effective in allowing the LLE to avoid obstacles in different scenarios, due to the fact that the agent was able to complete the simulations tasks successfully. Taking into account that the model is learned and that the real world is different than the simulated one, the results showed that it is possible to implement a DRL solution to generate trajectories autonomously for exoskeletons. Considering this, the future work will be introducing more constraints and variables, such as different obstacles and configurations in order to have a model that is more complete and compliant with real world environments. To further validate the solution, the method will be implemented and tested on a real exoskeletons, to further assess the capabilities of the proposed solution.

Bibliography

[1] A. F. Ruiz-Olaya, A. Lopez-Delis, and A. F. da Rocha, "Chapter eight - upper and lower extremity exoskeletons," *Accademic Press*, pp. 283–317, 2019.

[2] D. Kawamoto, "Exoskeleton suits: 26 real-life examples, https://builtin.com/robotics/exoskeleton-suit," *builtin*, 2024.

[3] H. Hu and Y. Liu, "Blind adaptive gait planning on non-stationary environments via continual reinforcement learning," in *2021 IEEE International Conference on Unmanned Systems (ICUS)*, pp. 280–284, 2021.

[4] A. Dastider, S. J. A. Raza, and M. Lin, "Safe locomotion within confined workspace using deep reinforcement learning," in *2021 Fifth IEEE International Conference on Robotic Computing (IRC)*, pp. 111–114, 2021.

[5] X. Wang, H. Fu, G. Deng, C. Liu, K. Tang, and C. Chen, "Hierarchical free gait motion planning for hexapod robots using deep reinforcement learning," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 11, pp. 10901–10912, 2023.

[6] L. Bao, J. Humphreys, T. Peng, and C. Zhou, "Deep reinforcement learning for bipedal locomotion: A brief survey," *arXiv*, 2024.

[7] A. Luo, Androwis, "Robust walking control of a lower limb rehabilitation exoskeleton coupled with a musculoskeletal model via deep reinforcement learning," *Journal of NeuroEngineering and Rehabilitation*, 2023.

[8] M. C. B. Lowell Rose, "A model-free deep reinforcement learning approach for control of exoskeleton gait patterns," *Robotica*, 2021.

[9] S. Ruo Xi Yong, "Reinforcement learning: Implementing td(λ) with function approximation," *medium*, 2023.

[10] G. A. S., "Robotic exoskeletons: The current pros and cons," *World journal of orthopedics*, pp. 112–119, 2018.

[11] E. M. Edoardo Trombin, Stefano Tortora, "Low obstacles avoidance for lower limb exoskeletons," Master's thesis, Università degli studi di Padova, 2022.

[12] A. B. R. Sutton, *Reinforcement Learning: An Introduction*. The MIT Press, 2018.

[13] Y. Bai, "Relu-function and derived function review," *SHS Web of Conferences*, 2022.

[14] S. Mohan, "A brief overview of eligibility traces in reinforcement learning," *Nerd For Tech*, 2024.

Acknowledgements

I'd like to thanks my parents and my brother to support me during this years, through the difficulties and the good times. I want to thank Prof. Tortora, Dr. Trombin and the University of Padua for allowing me to work on this research. At last, but not least, I'd like to thanks my childhood friends Alessia, Stefania, Marco, Damiano and Carlo, and my colleagues from Computer Engineering Lorenzo L., Lorenzo G., Riccardo, Marco F., Marco R., Giovanni, Francesco and Lorenzo C.