Universiteit Leiden
The Netherlands

# Opleiding Informatica

Learning a NAO Robot to Balance Using Reinforcement Learning

Justin de Rooij

Supervisors:
Joost Broekens & Michiel van der Meer

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

25/08/2022

**Abstract**

The NAO robot can be programmed to walk, but does so with a bipedal gait that is not similar to the human bipedal and biphasic gait. Balancing the robot is dependent on the feet that are flat at the bottom. For the flat feet to be sufficient to remain in balance, the centre of mass needs to be directly between the feet. The robot can make steps with a maximum of eight centimetres to make sure it does not lose balance. To be able to make the robot walk faster and make bigger steps, we investigate whether a form of balancing, similar to what is used by humans, can be learned to enable the robot to stay in balance without only relying on its feet. This is needed to eventually create a more natural gait in future research. To enable the robot to tilt forwards and backwards while balancing, a rounded prosthetic is added to the bottom of the feet. Furthermore, we create two different algorithms that both use the arms of the robot to counter an imbalance. The first algorithm uses the angle of the torso and moves the arms in the opposite direction. The second algorithm is a Deep Q-learning algorithm that moves the arms either forwards or backwards. Both algorithms had difficulties staying in balance for a long time period. The first algorithm introduces an oscillation effect which eventually leads to losing balance. The second algorithm develops a bias which favors moving the arms forward, whether the robot itself is tilting backwards or forwards. This subsequently leads to losing balance quickly. The difficulties make both algorithms not yet able to be fully accepted as new and stable balancing algorithms for the NAO robot.

# Contents

# 1 Introduction

The NAO robot, created by Softbank Robotics, is a humanoid robot that can be programmed to walk, speak, see, hear and interact. The NAO robot comes with sensors and a predefined biped gait which makes it able to walk around. The robot is a planar bipedal robot [Eur] and has flat feet, which makes it limited in its walking capabilities. The standard built-in walking algorithm allows for walking on flat and stable surfaces, because if the feet are not placed down on the floor in a flat manner, the robot will easily lose its balance. This means that the robot can only be in balance whenever it is able to put its center of mass directly above and between its legs and feet. So despite having the appearance of humans, it does not have the ability to walk like a human would.

Whenever the walking algorithm is executed the robot is able to set steps in the forward and lateral direction. Considering the forward direction, the robot is able to make steps ranging from a minimum of one millimetre to a maximum of eight centimetres [Eur]. The steps are done by lifting the leg that should be put forward and placing it down forward according to the step size. This means that again, the robot keeps its balance by moving its point of mass directly above the leg that is still on the ground. However this is not equivalent to the way a human would take its steps. The human gait consists of losing balance in the forward direction when lifting one leg and restoring the balance by placing the lifted leg somewhere in front of the other one. Contrarily, the robot uses the support of the feet in the lateral direction to be able to take a step.
Thus to make the robot use a gait that is more similar to that of a human, its gait will need to be changed. The eventual goal is to enable the NAO robot to walk faster. We investigate the first step towards a gait that is more natural [Lov05][Lov07] and human-like in the form of a new way for the robot to balance itself. This leads to the following research question: To what extent is a NAO robot, by using curved feet, its arms and reinforcement learning, able to learn a more natural way of balancing to eventually be used in a more natural gait?

To enable the robot to have a smoother tilt forwards and backwards, a new set of feet are attached to the bottom of the existing feet. The shape of the feet will be similar to that of the default feet, except that the bottom side is rounded in the longitudinal direction. Having a round and smoother tilt in the longitudinal direction allows the robot to not lose balance immediately whenever there is a tilt, which is the case for the original feet. This also enables the usage of balancing algorithms and techniques that were not useful to apply with the original feet. The algorithms used in this research are examples for this. To learn to balance, the robot will be programmed to use its arms to move the body to the opposite side relative to the direction of the imbalance. The arms are used because the arms are the parts of the body that can be extended the farthest to be used as counterweight, while the torso can remain upright. The direction and magnitude of the imbalance is taken from a sensor in the torso of the the robot. This algorithm forms the ad-hoc algorithm and serves as baseline. The baseline is compared to an algorithm that implements a reinforcement learning algorithm. We investigate this balancing problem to serve as a basis for more natural walking behaviours for NAO robots.

## 1.1 Thesis overview

Section 2 discusses related work and what approaches concerning balancing for biped robots have already been considered. Then, section 3 shows the algorithms that are used to learn to balance, the state and action spaces the algorithms use and the setup of the learning environment for the NAO robot is explained. Section 4 describes the results of the performance of the two algorithms that are investigated. Section 5 discusses the results and the limitations during the research. Lastly, section 6 contains conclusions and considerations for future research.

# 2 Related work

A field where balancing and walking for NAO robots is a broadly research subject, is the field of sports and soccer in particular. An example is the annual robot soccer world cup that is held since 1998 [NSM+98]. Soccer requires a more dynamic locomotion rather than what is needed for walking in a straight line and in previous research it has already been tried to improve the default NAO locomotion to be more dynamic [SSC09]. The following section discusses walking speed in a virtual soccer environment, whereas the last section discusses balancing in a general environment.

## 2.1 Running using reinforcement learning

When playing soccer, running and balancing are two of the most important aspects. Abreu, Reis and Lau [ARL19] propose a way of leveraging the Proximal Policy Optimization, introduced by Schulman et al. [SWD+17], to enable the NAO robot to run faster. Using this technique, they create a stable sprinting and stopping behaviour which enables a NAO robot to run from one point to another, without falling over along the way. The authors make use of a simulator that is used for official RoboCup matches. The behaviours that they create are tested and executed on a virtual NAO robot, in the aforementioned simulator. The authors have conducted tests where over a thousand episodes are executed. In all these episodes, the NAO robot did not fall over. When they consider an entire trajectory being traversed, from start at 0m/s to a distance of 28 meters, the robot reaches an average speed of 2.37m/s to 2.38m/s.

Whether a physical robot is able to accomplish roughly the same results as in the mentioned research cannot be deduced directly from this study since a simulation environment is used. This is due to the reality gap. Simulations are limited to the elements that have been taken into account by developers of the simulations. Although there may already exist several successful simulators, they are still an approximation of the real world. Multiple techniques are already available to counter this problem, such as adapting the simulator with real world data [CHM+19], domain randomization [PAZA18] or meta-learning [FAL17][NCL+18], and could be used to improve the conversion of this research from simulation to reality. Furthermore the research was limited within a specific set of rules, used in the official RoboCup matches. These rules limit the authors in using data that the NAO robot itself would be able to provide, such as the value of the gyroscope from the inertial unit. The authors need to calculate this data themselves instead and this data may differ from the data that would otherwise be provided by a physical robot.

### 2.2 Balancing of a biped robot using neural networks and Kinematics

To make balancing more dynamic and adaptive, Kun and Miller implemented an "adaptive dynamic balance scheme" [KM96], meaning that the center of mass is allowed to move away from above the feet and the robot should be able to remain in balance by making a step. The authors write that CMAC neural networks[MSW95] were responsible for maintaining balance front-to-back and side-to-side. The research concludes that it is possible to create a balancing scheme using neural networks, although it often needed human supervision to not fall on the ground when it lost balance. Besides the notion the robot being a biped robot, it is not clear what robot is used and conducting this research on a NAO robot does not necessarily lead to the same outcomes.

Kofinas, Orfanoudakis and Lagoudakis present "a complete and exact analytical forward and inverse kinematics solution for all limbs of the Aldebaran NAO humanoid robot" [KOL15]. For this analytical method, the authors wrote a library with a set of functions to do "real-time kinematics computations on-board the robot". Kofinas et al. have tested their model by using it in their implementation for three different applications, namely center-of-mass balancing, pointing to a moving ball and human-guided balancing on two legs. Considering here the two balancing applications, the authors conclude that their implementation results in better performance than other possible techniques that are mentioned in the paper. This library that was written by the authors could be used for future research, when walking is considered.

## 3 Method

To investigate whether it is possible for a humanoid NAO robot to learn to balance in a natural way, similar to that of humans, two algorithms will be implemented and executed in an experimental setup. This experimental setup is explained in the next section, whereas the algorithms are discussed in section 3.2.

### 3.1 Materials

Balancing is one of the most import aspects of walking and humans can use a large set of muscles to keep the body in balance so that it does not fall over. The NAO robot may look like a human, but it does not have all the muscles that a human has. Balancing with the NAO robot happens mostly by keeping its point of gravity roughly in the center between its legs and by receiving support from the flat feet. The feet could also be one of the biggest causes of imbalance when the NAO robot, with its default gait, would take a bigger step then is allowed by the default walking algorithm. If the robot does not place its feet flat on the ground, it will fall over because it has no built-in mechanism to restore its balance while falling over. So the first step is to give the robot rounded feet, by making a NAO foot prosthetic, see figure 1, and placing it at the bottom of the feet. This allows for a round and smoother tilt forwards and backwards in contrary to the original feet, where a tilt resulted in imbalance. The feet match the original NAO feet in shape and are 1.6 centimetres thick in their thickest point. Figure 2 depicts the stand where the robot is placed. This to make sure that the robot does not fall on the ground too often where it eventually might break. The surface is a square of 50 centimetres and the poles on the side are roughly 70 centimetres in height, such that they are higher than the robot is.

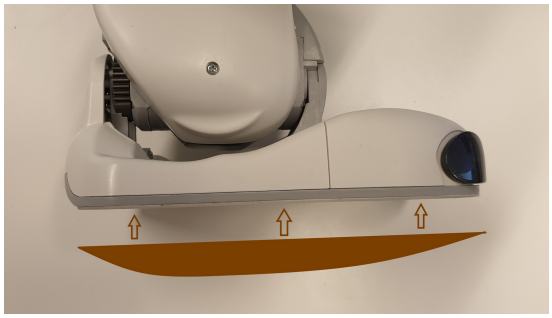

Figure 1: The prosthetic feet used for the robot.

Figure 2: Complete setup that is used while learning.

## 3.2 Algorithms

Both implementations use the movement of the arms over the shoulder joint to try to stay in balance. We define being in balance by having the robot have an angle of zero degrees, meaning the torso is in an upright position. Both algorithms will have the same conditions to work with, for example the initial position of the robot. This will be discussed in section 3.3. The algorithms are written in Python and use the Python SDK written for NAOqi[1] to be able to access the NAO robot's functionalities. Using the SDK, three types of values can be collected from the sensors of the robot, namely the angle of the torso in radians, the gyroscope in radians/second and the pitch of the right and left shoulder.

### 3.2.1 Ad-Hoc Algorithm

The goal for designing the first algorithm, is to investigate whether a simple approach to the balancing problem can result in the robot not losing its balance and to have a non-learning baseline. The pseudocode for the algorithm is below in algorithm 1 and will now be explained. Parameters, together with their value and a short description, can be found in table 1. The algorithm uses the angle of the torso to determine the position the shoulders should be in. The principle of the algorithm is to counter any imbalance the robot is experiencing. This means that, when the robot is tilting forwards, it will move its arms backwards to counter this. When tilting backwards the arms will be moved forwards. Furthermore, the speed at which the robot is losing balance is countered by faster or slower arm movements.

When executed, the algorithm starts with collecting the angle of the torso and gyroscope value from the inertial unit (line 12 in the pseudocode). Line 13 & 14 translate the angle the torso has to the angle the shoulder joints should be rotated to. Having the torso in an upright position, would result in an angle of zero degrees. This means that whatever angle the torso is in, it will be the

[1]Download page: https://developer.softbankrobotics.com/nao6/downloads/nao6-downloads-linux

| Parameter | Value | Description |
|---|---|---|
| angle | - | Angle of the torso in radians, taken from the inertial unit of the robot. |
| gyro | - | Gyroscope value in radians/second, taken from the inertial unit of the robot. |
| shoulder | 1.57 | Angle of the shoulders (in radians) when the arms are besides the torso, pointing downwards relative to the torso. |
| multiplier | 0.2 | Multiplier used together with the gyroscope value. |

Table 1: Parameters and their values, used in the ad-hoc algorithm.

angle that needs to be compensated to stay in balance. The shoulder parameter is the angle of the shoulders when the arms are besides the torso, see table 1. From this position, the arms can only be moved backwards by 30 degrees. Thus for this algorithm, the range of motion for the arms (over the shoulder joints) is limited between 60 and 120 degrees. The function that is called in lines 16 & 17 limits the new angle to this range.

The function that is called in line 18 adds the value of the gyroscope to the angle that is to be set (details of this function can be seen in lines 1 to 9). This value is defined as the speed the torso is rotating with, measured in radians / second. When the algorithm is executed and the robot is trying to remain in balance, the body receives momentum, coming from the movement of the arms. This makes the calculation of the angle, in which the arms should be placed, inaccurate. To counter this momentum, the gyroscope value is multiplied with the multiplier in table 1 and added to the angle to be set, whenever the torso is moving towards an upright position.

**Algorithm 1** Ad-hoc algorithm

**Input:** angle & gyroscope

1: **function** ADDGYRO(Diff, Angle, LeftShoulder, RightShoulder)
2: $Gyro \leftarrow gyro * multiplier$
3: **if** $Angle > 0$ & Diff$< 0$ **then**
4: Add Gyro to LeftShoulder, RightShoulder
5: **else if** $Angle < 0$ & Diff$> 0$ **then**
6: Add Gyro to LeftShoulder, RightShoulder
7: **end if**
8: return LeftShoulder, RightShoulder
9: **end function**
10:
11: **while** $Terminated = false$ **do**
12: Collect data from sensors
13: Diff$\leftarrow$ $angle - PreviousAngle$
14: $ScaledDiff \leftarrow$ Scale Angle to range of shoulder motion
15: $PreviousAngle \leftarrow angle$
16: $RightShoulder \leftarrow checkBounds(shoulder + ScaledDiff)$
17: $LeftShoulder \leftarrow checkBounds(shoulder + ScaledDiff)$
18: $LeftShoulder, RightShoulder \leftarrow addGyro(Diff, Angle, LeftShoulder, RightShoulder)$
19: Set new angles for the shoulders
20: **end while**

### 3.2.2 Reinforcement Learning Algorithm

In algorithm 2 below, the pseudocode for the Reinforcement Learning algorithm can be found. The pseudocode contains parameters that are shown in table 2, together with their value and a short description. The algorithm used is a Deep Q-learning algorithm [MKS+15] that uses the ε-greedy exploration strategy. This algorithm was chosen based on its good performance with a similar problem, that of the inverted pendulum problem [ÖVTU20]. The algorithm is implemented using Keras and TensorFlow for the neural network components. A visualisation of the neural network can be seen in figure 3. Here, the numbers represent the number of nodes in a layer. The neural network takes as input the angle and gyroscope values from the inertial unit, and the shoulder angle from the local sensor. The possible actions are the output of the network and can be two options, namely 0 and 1, meaning moving the arms backwards by five degrees and forwards by five degrees respectively (this happens in line 3 and 5 of the pseudocode).

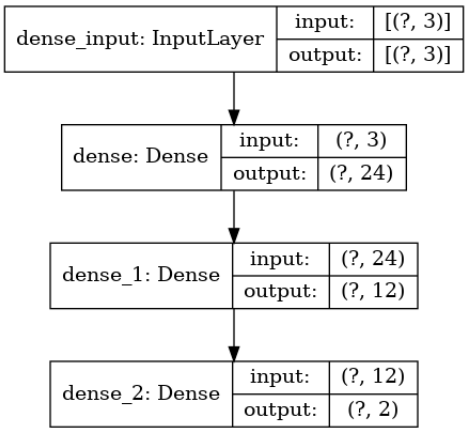


Figure 3: Visualisation of the neural network used in the algorithm.

An episode can be ended in two ways. Firstly, when the angle of the torso is above a certain value which cannot be recovered, this is counted as having lost balance completely. Secondly, when the number of steps done in an episode exceeds 750, this is also considered as the end of an episode. The ε-greedy exploration strategy allows to have an emphasis on exploration at the beginning of the algorithm, to then slowly move to an emphasis on exploitation towards the end of the algorithm. The initial $\varepsilon = 1.0$. Values of 0.8, 0.6 and 0.4 have also been used to determine when the algorithms performed the best. Following experiments of which the results can be seen in figure 9, an $\varepsilon$ of 1.0 seemed to perform the best. The decay of $\varepsilon$ is then defined as follows:

$$\epsilon = 0,01 + (1,0 - 0,01) * e^{(-0,01 * episode)} \tag{1}$$

where 1.0 is the initial value of ε. To visualise the decay of ε, figure 4 shows the decay for the four used values of ε. During testing, $\varepsilon$ had a value of 0, meaning that every action was determined by the neural network.
The training model is updated every 4 steps and this value is chosen because it is also used in the original work on deep reinforcement learning [MKS+15]. Training more often may result in

| Parameter | Value | Description |
|---|---|---|
| angle | - | Angle of the torso in radians, taken from the inertial unit of the robot. |
| gyro | - | Gyroscope value in radians/second, taken from the inertial unit of the robot. |
| shoulder | - | Angle of the shoulders in radians, taken from the local sensors of the robot. |
| training_model | Neural Network | A neural network that is trained after every 4 steps in the learning process. |
| target_model | Neural Network | A neural network that is occasionally updated with the weights of the training_model and forms the final network. |
| replay_memory | deque | Memory of past steps to train the training_model with. |
| ε | 1 | Parameter used to move from exploration to exploitation during the learning process. |
| steps_to_update_target_model | 0 | Parameter to keep track of number of steps that have already been done. |
| n_steps | 100 | Number of episodes done in the learning process. |

Table 2: Parameters and their values, used in the RL algorithm.

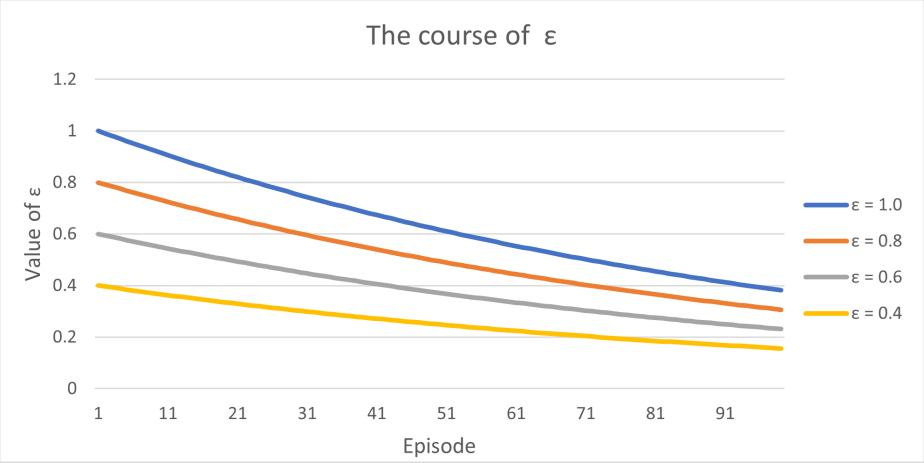


Figure 4: Course of the decay of ε.

overtraining the model. The model trains using experience replay. The model is trained on a dataset of past observation, action, reward triplets and thus makes use of offline learning. During the execution of the algorithm, the database is filled with triplets that are formed every step since the start of the algorithm. When the model is trained, a random batch of triplets is taken from

the database. The reward is the angle of the torso and negated when it is positive. This makes the upright position of the torso give the highest reward and losing balance completely results in the lowest reward. Specifically, when the angle of the torso is either 0.15 or -0.15 as example, the reward will in both cases be -0.15, such that the closer the value is to 0, the higher the reward is.

**Algorithm 2** Deep Q-Learning algorithm

```
    Input: angle, gyroscope & shoulder
 1: function STEP(action, step_count)
 2:     if action == 0 then
 3:         move_arms_forward()
 4:     else if action == 1 then
 5:         move_arms_backwards()
 6:     end if
 7:     Set new angles for the shoulders
 8:     Do a new observation
 9:     Determine reward and check if episode is done
10: end function
11:
12: for episode in n_steps do
13:     Bring robot to initial position
14:     observation ← [angle, gyroscope, RShoulder/LShoulder]
15:     done ← false
16:     while not done do
17:         steps_to_update_target_model+ = 1
18:         if random_number ≤ ε then
19:             Choose random action
20:         else
21:             Choose action with the highest Q-value in training_model
22:         end if
23:         new_observation, reward, done = step(action, step_count)
24:         Add [state, action, new_observation, reward] to replay_memory
25:         if steps_to_update_target_model%4 == 0 or done then
26:             Train training_model
27:         end if
28:         observation ← new_observation
29:
30:         if done == true then
31:             if steps_to_update_target_model ≥ 100 then
32:                 Copy weights of training_model to target_model
33:             end if
34:         end if
35:     end while
36:     Make ε decay slightly
37: end for
```

## 3.3 Experimental setup

At the start of training and testing, the robot is placed in the same spot in the middle of the stand, which is explained in section 3.1. The initial starting position of the robot is set as follows: Shoulders pitch is set to 90°, the right and left shoulder roll are set to -10° and 10° respectively, elbows yaw are set to 0°, the right and left wrist yaw are set to 2° and -2° respectively, both hands are closed completely, the hip yaw pitch is set to -7.8° and the head pitch is set to 1°. This position, together with the prosthetic feet, make for imbalance forwards and backwards around an angle of 0° (the upright position). This means that while testing and training, the robot was able to randomly

lose balance either forwards or backwards. While the algorithms are executed, no further support is given to the robot. Whenever an episode had ended while training, the robot was put back in the same initial position, to ensure that every episode would operate under the same circumstances.

The following tests have been conducted: Firstly, the performance of both the ad-hoc and RL algorithm has been tested by recording the angle of the torso and the shoulders for a period of roughly 16 and 17 seconds respectively. During the learning process of the RL algorithm, we have also recorded the time without falling for every episode. All these tests were conducted five times and will be averaged in the results. To show whether and how the model is learning, we have conducted a test where we collected the model after four different time periods and recorded the performance. Lastly, we have conducted a test to see which initial value of $\varepsilon$would result in the best performance.

# 4 Results

This sections presents the obtained results for the two algorithms, of which the implementation can be found in section 3.2. First, the performance of the ad-hoc algorithm will be discussed. Next the performance of the RL algorithm will be discussed, together with the time the robot was able to balance itself during the training process. Furthermore, the performance of the model at four different stages in the learning process is discussed. Lastly, a comparison of the performance of different initial $\varepsilon$ is discussed.

## 4.1 Ad-Hoc Algorithm

After the initial implementation of this algorithm was executed, an oscillation effect, added by the movement of the arms and body, could be observed visually. To counter this observed momentum, the gyroscope value, multiplied with a multiplier, was added to the angle to be set. This multiplier can be found in table 1 and its value is chosen because this is the best performing multiplier. Multiple multipliers have been tried to analyze whether this would have a positive effect on the algorithm. Multipliers of 0.15 and lower seemed to not have enough effect to counter the observed momentum. Multipliers of 0.25 and higher led to an effect that appear as shocking movements. It was observed that this effect led to momentum forwards and backwards alternately and this always led to the robot losing balance completely. So the multiplier used in the algorithm is as high as possible while not creating this shocking effect and this algorithm is used while obtaining the following results.
Figure 5 shows the execution of the algorithm for a time period exactly long enough to execute 300 actions, averaged over 5 runs. This means the algorithm has obtained an angle and successively set a new angle for the shoulder joints exactly 300 times, taking a total time of around 15.7 seconds. The figure shows clearly that when the angle of the torso is increasing, meaning that the torso is tilting forwards, the angle of the shoulders is also increasing, meaning that the arms are moving backwards. Conversely, when the angle of the torso is decreasing, meaning that the torso is tilting backwards, the angle of the shoulders is decreasing, meaning that the arms are moving forwards. The minima and maxima of the shoulder values seem to be more to the right, but only by a very small margin. This coincides with the visual observation that the movement of the arms seem to

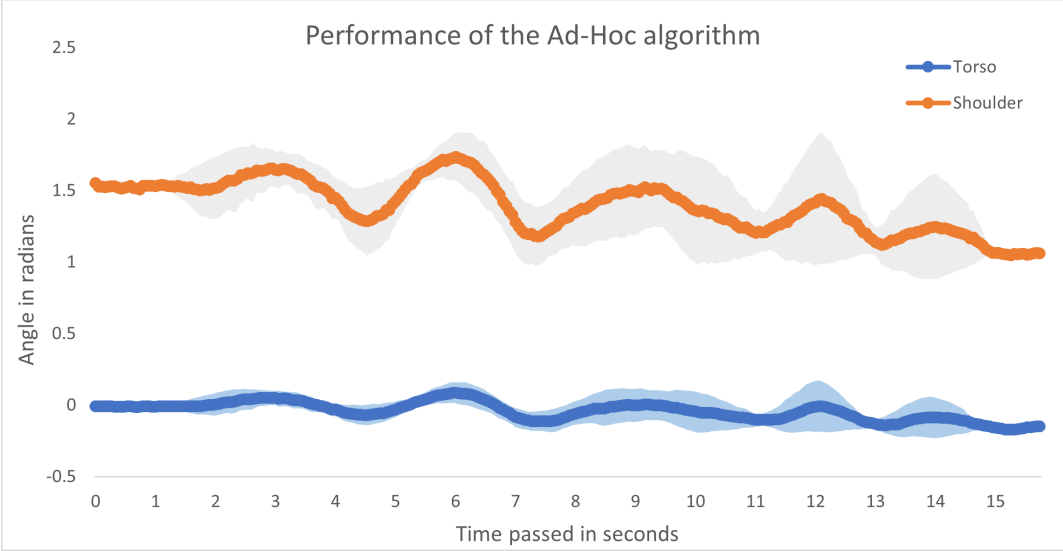


Figure 5: Averaged position of torso (in radians) compared with position of shoulder joints (in radians) with standard deviation in the form of error bands, during execution of the algorithm.

have a small delay compared to the angle of the torso. Upon starting losing balance, when the algorithm is executed, the robot starts to react to the angle the torso is making. During the time period, it can be seen from the figure that the robot is moving further away from an angle of 0 (the upright position) at every iteration. The shoulders follow the same behaviour and there is thus an oscillation effect which we were not able to remedy by tuning the possible parameters in the algorithm.

The standard deviation, indicated with the error bands around the average lines in the graph, shows the variation in the different runs. Because the robot can randomly fall either forwards or backwards when the algorithm starts, the 5 runs individually have very similar data points, but some are mirrored to others. The algorithm seems robust, meaning that the robot seems to perform the same each time, given a set of parameters.

## 4.2 Reinforcement Learning Algorithm

Figure 6 shows the performance of the Deep Q-learning algorithm during the training process, averaged over five runs. Throughout the whole training time, most episodes end under five seconds. When the training progresses, more and higher peaks can be seen in the data. Thus there is an increase in the number of seconds without falling, but the majority of the episodes still result in losing balance within four seconds. The algorithm is trained for 100 episodes. Higher number of episodes have been tried as well, but this did not seem to improve the performance of the algorithm. What did happen was that some elements of the behaviour observed after 100 episodes, became stronger and more visible. As example, a preference to move its arms in the forward direction is already present after training for 100 episodes, but after more episodes the arms were moved forwards faster and more often. The algorithm also seemed to be less reactive to the angles the torso was making and thus fell over faster.

The standard deviation, indicated with the error bands, show high variance between the five runs in the first 20 episodes. This is likely due to that there is much exploration at the beginning and thus can lead to losing balance very fast, or take the right actions randomly and lose balance slower. Between episode 40 and episode 60, the variance increasing together with the average and the algorithm seems to perform slightly better than it did before. The higher peaks in the variance could arise because an episode in one of the runs suddenly performed very well, but the next episode could again be a very bad performance.

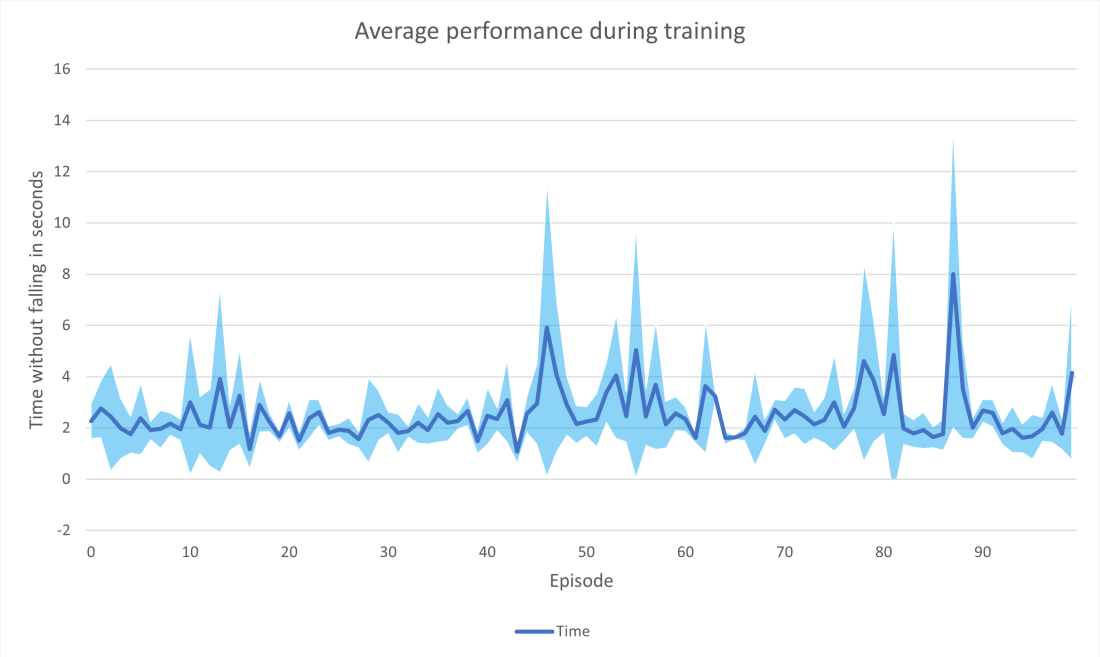


Figure 6: An averaged overview of the time in seconds the robot was able to balance itself while training, including the standard deviation in the form of error bands.

Figure 7 shows the execution of the algorithm for a time period roughly of the same length as the ad-hoc algorithm. The figure contains the average over 5 independent runs and includes the standard deviation. The reinforcement learning algorithm (RL algorithm) has obtained an angle and successively set a new angle for the shoulder joints exactly 200 times, taking a total time of around 17.3 seconds. Due to computational overhead, the ad-hoc algorithm was able to do 300 times within a slightly shorter time period. Almost immediately from the start, the robot starts moving its arms forward. This movement increased the imbalance forward and the robot was just able to prevent falling over by moving the arms backwards again. This behaviour happened repeatedly but with smaller movements, as can be seen after six seconds have passed. Whether the robot was moving backwards or forwards when the algorithm was executed, the robot seemed to have a preference for moving its arms forward and this preference could already be spotted very early on in the training process. The reward for this algorithm consisted of only negative rewards, using the angle of the torso. We tried to improve on this by adding a small positive reward whenever the torso was in a upright position, to be precise angles within $\pm 0,05$ of 0. This however did not lead to any improvements on the behaviour of the robot.

Again for this graph, the standard deviation shows the variation in the different runs, indicated by the error bands around the average lines. The variance seems to be high whenever the line is rising or falling, but seems to be stable in the maxima and minima. This is likely because the different runs do not always follow the same data points when the body of the robot is moving, but eventually all had the bias in the forward direction which explains the small variance in the maxima and minima. The algorithm is not robust, because it can perform very differently, even with the same parameters.

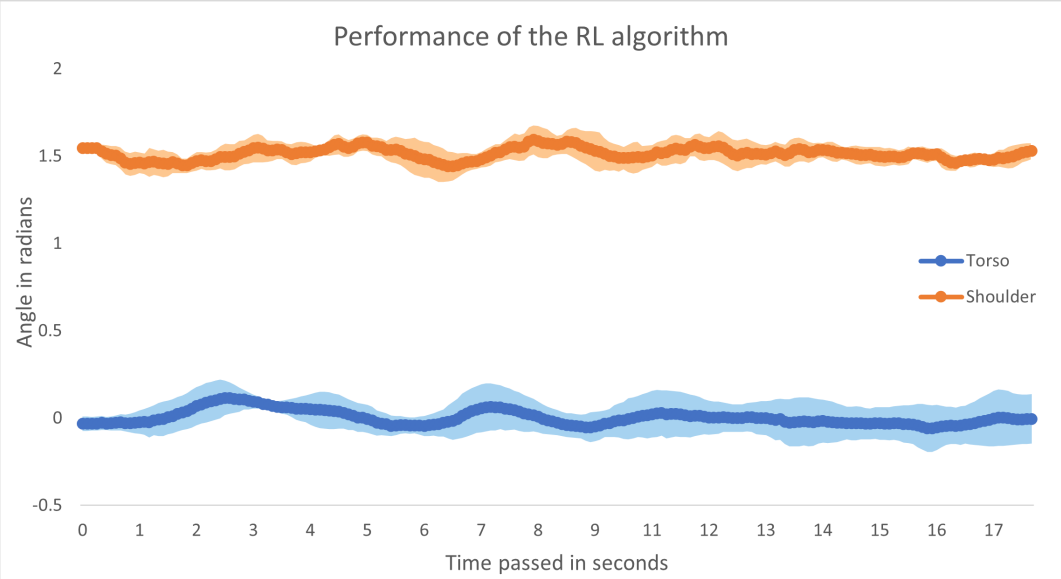


Figure 7: Position of torso (in radians) compared with position of shoulder joints (in radians) with standard deviation in the form of error bands, during execution of the algorithm.

To investigate whether our model does improve during the learning process, we have collected the models after a number of steps (actions, new angles being set) had been done during the learning process, namely after 500, 1000, 1500 and 2000 steps. The number of steps can be roughly translated to after 4.5, 9, 13.5, and 18 seconds respectively. This can be seen in figure 8. After having reached 2000 steps done, all 100 episodes were also completed indicating the end of the learning process. The figure shows that the model taken after 500 steps had been completed does not do much, except for moving its arms forward to the most extreme position, and then never reacting to the angle the torso is making. This results in falling over within three seconds. The model after 1000 steps have been completed also starts by moving its arms forward immediately and after three seconds have passed, the angle the torso is making means the robot has fallen over. Because of our setup, the robot bounced back from the pole it fell against and after three seconds have passed, the robot does seem to react to the angle the torso is making and starts moving its arms backwards. After 13 seconds have passed the angle of the torso again indicates the robot has fallen over, but this time backwards. Again, being in this extreme position makes that the robot finally seems to react to the angle of the torso and starts moving its arms forward, right before the run ends.
The model after 1500 steps had been done again starts by moving its arms forward, but seems

to react to the angel the torso is making, earlier than the previous models, see the data point after almost two seconds have passed. After 5 and 12 seconds have passed, the robot moves its arms backwards to an extreme position. The angle of the torso at those time periods indicate the robot has fallen over. So this model seems to react better to losing balance forwards than the previous models do, but all still do not move the arms forward fast enough to prevent losing balance backwards. The model after 2000 steps had been done still starts by moving its arms forward, but does this less far then the previous models did. Overall does this model not make the robot move its arms into extreme positions. The robot itself does go into an extreme position forwards after 8 and 13 seconds and backwards after more than 16 seconds have passed and hits the pole, where it slightly bounces back. However after 5 and 10 seconds have passed, the robot seems to take the right actions to prevent falling over again.

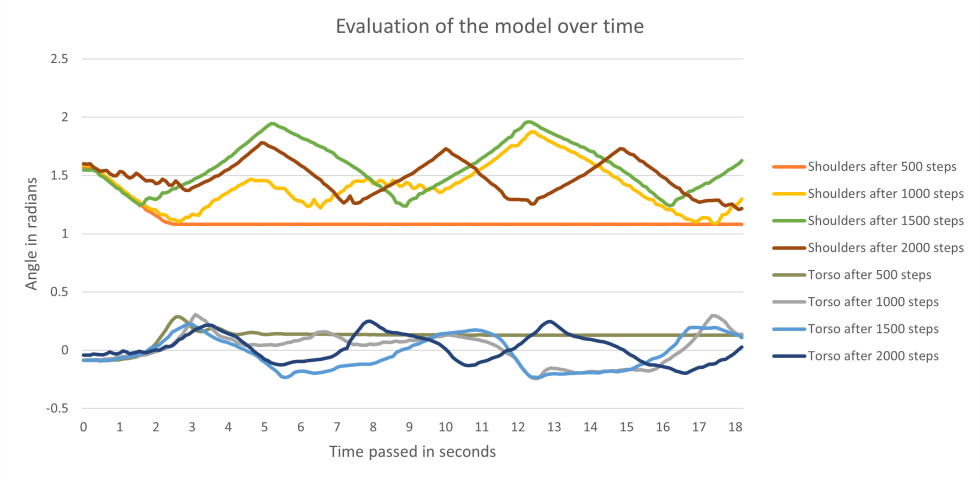


Figure 8: Execution of the model in four different stages during the learning process.

To increase on the performance discussed above, further experiments were attempted with different hyperparameters. Firstly, we tried to copy the weights of the training model to the target model more often, from every 4 steps to every 2 steps. This however lead to a faster development of the bias to move the arms forward no matter the angle of the torso and overall did not lead to any improvements. Furthermore we tuned the initial $\varepsilon$ parameter. As described earlier, an initial $\varepsilon$ of 1.0 is used in the reinforcement learning algorithm. However we have also used the value 0.8, 0.6 and 0.4 for $\varepsilon$. The results can be found in figure 9. It stands out that the training process when $\varepsilon = 1$ has higher peaks throughout the episodes, which means it is able to keep in balance for a longer period. The line for $\varepsilon = 0.4$ can more often be found at the lower side of the graph, not being able to keep in balance much longer than one second. The visual observation for these episodes is that the robot immediately moved its arms forward and loses balance. The figure together with observations make it seem that when less exploration is done at the beginning of the learning process, by lowering the initial $\varepsilon$, the robot is likely to do not much more than moving its arms forward in an episode. To investigate whether the foot prosthetic had any negative influence on the performance of the algorithms, both algorithms have been executed after removing the foot prosthetics and setting the stiffness of the ankle joints to 0. Moving only a few degrees from the upright position is made

difficult by friction. When trying to move further the friction has no influence anymore and the robot would fall very quickly, where the movement of the arms to counter would have a small to no effect. Thus using this as a comparison to the rest of the research is not possible.

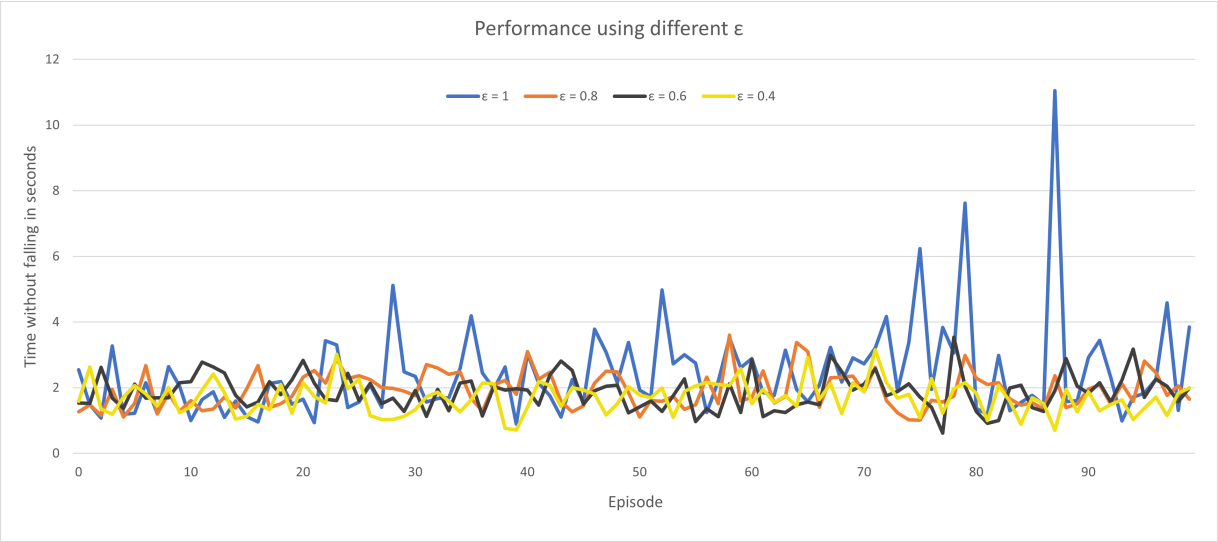


Figure 9: Performance of the algorithm with varying initial ε.

The most notable difference in the performance of the ad-hoc and RL algorithm seems to be that the ad-hoc algorithm reacts faster to the angle the torso is making, which then creates an extra momentum, whereas the RL algorithm seems to eventually be able to counter the imbalance but does so in extreme positions and with slower arm movements. The RL algorithm can also make the robot lose its balance in a short time when exploiting its bias to move the arms forward no matter the angle of the torso. The arms seem to move faster with the ad-hoc algorithm which results in more extreme positions for the shoulders in, which can be seen in figure 5, and this can be compared to the RL algorithm in figure 7.

# 5 Discussion

## 5.1 Limitations

The execution of this research was limited by a some aspects. Firstly there is a hardware limitation. During the execution of the algorithm, the robot retrieves data from its sensors and inertial unit multiple times. The developers of the robot and the associated software, state in the documentation that the memory element of the robot should not be accessed more than once per 50 milliseconds, otherwise the CPU might overheat. This is mostly a concern for the ad-hoc algorithm, because this algorithm is now deliberately slowed down to prevent the algorithm from accessing the memory unit too often. Secondly, when the angle for the shoulder is set and the new angle differs only

0,1° or 0,2° from the previous one, the robot sometimes does not move its arms. This made very small movements in the ad-hoc algorithm almost impossible. Lastly, the center of mass of the robot is around 2/3 of the feet, seen from front to back. Thus when making the prosthetic feet, the thickest part of the feet had to be put around 2/3 of the feet, but it was too difficult to determine precisely where this point is. This is why some of the positions of the limbs of the robot, as discussed in section 3.3, are altered to compensate for this inaccuracy. This means that the robot was able lose balance forwards and backwards while being in the upright position. However this does make the feet asymmetric. Thus tilting backwards is easier than tilting forwards. Furthermore this led to the front of the feet giving some resistance when tilting forwards, which led to a small momentum backwards. This influenced the ad-hoc algorithm in its execution, because this added momentum leads to bigger differences in the angle of the torso. It follows that this leads to faster arm movements and this again results in some momentum, as discussed earlier.

### 5.2 Results discussion

In section 4.2, it is discussed that the robot seems to have a preference for moving its arms forwards when executing the RL algorithm. It seems the algorithm reacts to the limitations discussed above, namely the fact that it is easier to fall backwards than it is to fall forwards. Whenever the robot started to fall backwards, it also moved its arms forwards in almost all the cases (but not all). When falling forwards, none of the times the algorithm was executed, the robot started by moving its arms further backwards than 90°. Whenever this algorithm would be used to balance the robot while it is walking, it is not ideal that falling forwards can almost never be prevented, only when it is losing balance forwards very slowly. To make sure this effect did not arise because every episode started with an angle of roughly 0, it has been tried to retry the learning process and now holding the robot at a variety of random angles. This also did not improve the behaviour. Figure 8 clearly shows that the algorithm improves over time. The first model just loses balance, while the other models seem to increasingly react to the angles the torso is making. However this is not enough to stay in balance for a longer period of time, as figure 6 is showing no ascending trend, and the robot also does not return to a state where the angle of the torso is only slowly moving around 0.
Doing less exploration leads to a worse performance of the algorithm, see figure 9. Hence an initial value for $\varepsilon$ of 1 seems to be the best choice out of the four options. During the learning process, an increasing number of episodes show peaks. However the expected ascending tendency in the graph, meaning the algorithm keeps improving throughout the process, stays out.
The ad-hoc algorithm has a response that matches more the movement of the torso. It correctly moves forward when falling backwards, and correctly moves backwards when falling forwards. The algorithm does however experience an oscillation effect which prevents the robot to return to a stable position.

## 6 Conclusions and Further Research

The ad-hoc and RL algorithms both have their shortcomings, which makes them not yet good enough to use in further research for a more natural gait. The ad-hoc algorithm shows arm movements that coincide with the expected behaviour, namely countering imbalance by moving arms in the opposite direction, but the arms continue to move faster until the robot loses its balance.

The algorithm misses an effect that allows for slowing down the movement to return to an upright position, without falling over. Furthermore it looks like that the small delay, that arises because the algorithm needs to react to reading a new torso angle in real time, makes it impossible to put the arms in the exact position needed to return to a stable position. For this, future research could be done to investigate whether this algorithm can be improved by adding some sort of prediction mechanism. As our algorithm only reacts to the angle of the torso, it could be that the slowing effect, that was mentioned before, can be obtained by predicting what needs to happen when a certain angle is registered. This can be in combination with the shoulder and gyroscope values. One way to do this, could be by recording the angle of the torso, the action performed by the algorithm together with the result of the action. It could be investigated if it is possible to create a general formula to add a small value when necessary to counter the oscillation effect. This allows the algorithm to avoid the small delay and may be just enough to cancel the extra momentum.

Regarding the RL algorithm, there need to be improvements to become stable. The preference to move the arms forward, even though the torso is already tilting forward, is currently the biggest shortcoming. A possible explanation could be that the feet make it easier to fall backwards than forwards. It might be that the feet make it necessary to have a preference forward which also prevails when the arms should be moved backwards, but this would not coincide with the notion that the bias can already be noted after only some episodes. Further research could investigate if other feet prosthetics encounter the same problem. It could be investigated if the feet can be adjusted to be less round and if it follows that the bias also becomes less present. Another possibility would be to investigate whether adjustments to the neural network itself could lead to different behaviour. Altering the number of nodes in a layer, the number of layers or the activation function are common techniques to find the best performing neural network for a problem and may lead to improvements or deterioration in the behaviour, for example the bias.

Both algorithms are currently not good enough to be used in a walking algorithm, although the ad-hoc algorithm seems to be close. The RL algorithm does seem to learn a somewhat correct balancing behaviour, but still does need to improve to be able to perform well enough to be used in a walking algorithm. To investigate whether the algorithms are unsuitable to solve the balancing problem on biped robots in general, it should be investigated if other biped robot suffer the same shortcomings when executing the algorithms.

# Acknowledgements

I would like to thank my two supervisors at LIACS, Joost Broekens and Michiel van der Meer, for helping me in setting up the project, giving advice throughout the research, the realization of this paper and lastly providing me with a NAO robot to conduct the research with.

# References

[ARL19] Miguel Abreu, Luis Paulo Reis, and Nuno Lau. Learning to run faster in a humanoid robot soccer environment through reinforcement learning. In *Robot World Cup*, pages 3–15. Springer, 2019.

[CHM+19] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8973–8979. IEEE, 2019.

[Eur] SoftBank Robotics Europe. Softbank robotics documentation - locomotion control. http://doc.aldebaran.com/2-8/naoqi/motion/control-walk.html. Last Accessed: 29-06-2022.

[FAL17] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[KM96] Andrija Kun and W Thomas Miller. Adaptive dynamic balance of a biped robot using neural networks. In *Proceedings of IEEE international conference on robotics and automation*, volume 1, pages 240–245. IEEE, 1996.

[KOL15] Nikolaos Kofinas, Emmanouil Orfanoudakis, and Michail G Lagoudakis. Complete analytical forward and inverse kinematics for the nao humanoid robot. *Journal of Intelligent & Robotic Systems*, 77(2):251–264, 2015.

[Lov05] C Owen Lovejoy. The natural history of human gait and posture: Part 1. spine and pelvis. *Gait & posture*, 21(1):95–112, 2005.

[Lov07] C Owen Lovejoy. The natural history of human gait and posture: Part 3. the knee. *Gait & posture*, 25(3):325–341, 2007.

[MKS+15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[MSW95] W Thomas Miller, Richard S Sutton, and Paul J Werbos. *Neural networks for control*. MIT press, 1995.

[NCL+18] Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. *arXiv preprint arXiv:1803.11347*, 2018.

[NSM+98] Itsuki Noda, Sho'ji Suzuki, Hitoshi Matsubara, Minoru Asada, and Hiroaki Kitano. Robocup-97: The first robot world cup soccer games and conferences. *AI magazine*, 19(3):49–49, 1998.

[ÖVTU20] Recep Özalp, Nuri Köksal Varol, Burak Taşci, and Ayşegül Uçar. A review of deep reinforcement learning algorithms and comparative results on inverted pendulum system. *Machine Learning Paradigms*, pages 237–256, 2020.

[PAZA18] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3803–3810. IEEE, 2018.

[SSC09] Johannes Strom, George Slavov, and Eric Chown. Omnidirectional walking using zmp and preview control for the nao humanoid robot. In *Robot Soccer World Cup*, pages 378–389. Springer, 2009.

[SWD+17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.