DEVELOPMENT AND EVALUATION OF A SAFE FALL CONTROLLER FOR A LOWER LIMB EXOSKELETON USING REINFORCEMENT LEARNING

K. H. Vogelzang

FACULTY OF ENGINEERING TECHNOLOGY
DEPARTMENT OF BIOMECHANICAL ENGINEERING

Dr. E..H.F. van Asseldonk
W.F. Rampeltshammer, MSc
Dr. B. Rosic
Dr. A.Q.L. Keemink

DOCUMENT NUMBER
BW - 747

14 AUGUST 2020

UNIVERSITY OF TWENTE.

Summary

Lower limb exoskeletons allow people with paraplegia to walk again. Currently, this walking is rather limited because the pilots have to use crutches to maintain balance and steps are made by following pre-defined trajectories. Additionally, most exoskeletons are used in a clinical or research setting, generally with some safety mechanism in place in case of a disturbance. It is likely that the field of lower limb exoskeletons will continue to develop, allowing for more exoskeletons to be used without any external safety mechanisms. Not having these makes the exoskeleton rely more on a good balance controller to maintain balanced when facing disturbances and a good safe fall controller if this balance cannot be maintained and damage has to be minimised. Due to the assumption that another safety mechanism is in place to catch the human-exoskeleton system, the state of the art fall controllers do very little to actually break the fall, risking injury to the human pilot and damage to the exoskeleton. This work, therefore, aims to create a better controller for minimising damage when falling with a lower limb exoskeleton. This controller needs to lower both the amount of times the head is hit as well as the impact force on and impact velocity of the head if it does hit the ground. This is tested for the forward and backward fall using a three-dimensional, humanoid model.

To create this controller the Soft Actor-Critic reinforcement learning algorithm has been chosen as the implemented algorithm. This algorithm expands on the regular Actor-Critic by changing the deterministic policy into a stochastic one from which the actions are drawn. For the model of the human-exoskeleton system, a humanoid model is used which can only control its lower limbs. The upper limbs are kept at their initial position using high stiffness and damping to train for different methods of catching, making the controller more robust to different human pilot fall strategies. To get a different fall strategy for the forward as for the backward fall within limited training time, two controllers have been trained, differing in the trained fall direction. As the algorithm is trained for minimising impact, the reward mostly consists of a big, negative, instantaneous reward at the first major impact. The used value functions have trouble foreseeing this reward, which results in a higher return than correct. Therefore, after the episode ends the reward of every transition in the episode is modified to also include a discounted future reward. This results in propagating the rewards backwards to the actions responsible for the impact allowing it to train more quickly towards an adequate fall strategy.

The controllers were quantitatively and qualitatively evaluated using a set of nine test cases. The qualitative evaluation of the forward fall controller returned that the controller mostly exposes the distal limbs to floor contact, while mostly preventing impact on the head and body core. The controller does, however, depend heavily on the upper body to catch itself. The qualitative evaluation of the backward fall controller showed that the impact is on the distal limbs and the pelvis, the risk of head injury is low, the falls do not look severe and there is little dependence on the upper body to catch itself. The quantitative evaluation showed that the presented controllers have a far lower head impact, if any, compared to the state of the art controller. Altogether, for their respective fall direction, the created controllers successfully divert the impact on the head, torso and waist to the distal limbs, where the impact generally causes less damage.

Contents

1 Introduction **3**
- 1.1 Background . . . 3
- 1.2 Objective . . . 4
- 1.3 Outline . . . 5

2 Theory **6**
- 2.1 Fall detection . . . 6
- 2.2 Fall direction . . . 7
- 2.3 Fall strategy . . . 7

3 Reinforcement learning algorithm **9**
- 3.1 Selection . . . 9
- 3.2 Soft Actor-Critic algorithm . . . 10

4 Methods **13**
- 4.1 Simulation set-up . . . 13
  - 4.1.1 Soft Actor-Critic modifications . . . 13
  - 4.1.2 Humanoid model . . . 15
  - 4.1.3 Reward . . . 17
  - 4.1.4 Observations . . . 18
- 4.2 Evaluation method . . . 19

5 Results **22**
- 5.1 Forward fall controller . . . 22
- 5.2 Backward fall controller . . . 26
- 5.3 Generalisation to other falls . . . 30

6 Discussion **31**

7 Conclusion and Future work **34**

Bibliography **36**

Appendix **39**
- A Robustness to noise . . . 39
- B List of reinforcement learning terms . . . 41
- C Training parameters . . . 43

Chapter 1

Introduction

Lower limb exoskeletons are used more and more for rehabilitation, for instance to aid people with paraplegia to be able to walk again, also allowing them to take the exoskeleton home for personal use (Kandilakis and Sasso-Lance, 2019). Currently, the lower limb exoskeletons are mostly being used together with crutches or a walker to guarantee balance (Esquenazi et al., 2012), (Tefertiller et al., 2018). Although there is ongoing research into crutchless walking (Gurriet et al., 2018), hence, it is expected that crutches will not be needed in the future. Crutchless walking would result in people with paraplegia to be able to use the exoskeleton more freely. Not having crutches frees up the hands to be able to carry things around or perform activities of daily living more easily. The downside of not having crutches, however, is that this makes the total system more unbalanced, similar to humans. This means that there will always be a disturbance that is able to topple the human-exoskeleton system. If not caught by a balance controller the system will fall and a safe fall strategy should be performed which minimises the damage to the human pilot and the exoskeleton. The current safe fall controllers are developed with the crutches, walkers or other safety mechanisms in mind, resulting in a controller that either locks the joints or frees them completely allowing for the safety mechanism to catch the human-exoskeleton system without any erratic movement (He et al., 2017). However, these do not offer any safety in case no safety mechanisms are in place or if the direction of the fall cannot be caught by the safety mechanism, as is the case for a backward fall when using a walker. Locking all the joints during fall can result in head impact velocities of up to 5.2 m/s (Qiu et al., 2018), which are higher than some concussion inducing impact velocities in American Football (reported as low as 3.75 m/s) (Campolettano et al., 2018). So, in order to be able to use lower limb exoskeletons safely without the use of other safety mechanisms, a safe fall controller has to be developed which is able to lower the head impact velocity. This work aims to create such a safe fall controller, which is able to catch the human-exoskeleton system to limit the impact to the head.

1.1 Background

Currently, research into falling in an exoskeleton is very limited (Khalili, 2016), as most research for exoskeletons is in a clinical or research setting where safety mechanisms are in place to prevent the falling of the exoskeleton. With these safety measures in place the current fall control strategy of either locking or freeing the joints is sufficient (He et al., 2017). Here, locking means to add a high virtual stiffness and damping to the control scheme to keep the joints approximately in place, whereas freeing means to remove any control from the joint completely. This fall strategy, however, brings major safety issues if not coupled with other safety devices as it does little to mitigate the conversion of potential energy into kinetic energy (Khalili et al., 2017). Qiu et al. (2018) reported that a "locking" safe fall controller results in a head impact

of 4.8 m/s for the forward fall, 4.5 m/s for the backward fall and 5.2 m/s for the sideways fall, when performing fall experiments with healthy users.

Some research into new safe fall controllers for exoskeletons has already been done as Khalili et al. (2017) created a controller for the lower limb exoskeleton for the backward fall based upon a triple inverted pendulum model in the sagittal plane. This fall strategy consisted of squatting at the beginning of the fall and knee extension near the end of the fall. This strategy was also experimentally tested in Khalili et al. (2019) by a half-scale mechanical prototype. Both in the simulations and in the experimental testing the resulting hip impact velocity was just above 2 m/s (2.09 and 2.04 m/s, respectively) and head impact was negated.

1.2 Objective

In order to allow people with paraplegia to use exoskeletons more freely, it is not only required to create a better controller for walking or a better balance controller. It is also required to create a controller for handling the disturbances which cause such unbalance that a fall is inevitable. As soon as no other safety mechanisms are in place, the controller of the exoskeleton should be able to fall as safely as possible, reducing the damage to the human pilot by preventing head and torso impact. This work aims to create this controller, which minimises damage when falling with a full lower limb exoskeleton used by people with paraplegia.

There are two specifications on this research goal which elaborate on the lower limb exoskeleton and the safe fall controller. First, the chosen lower limb exoskeleton to create for is the Symbitron^{+} exoskeleton (Meijneke et al., 2017) as this is the exoskeleton used at the local institution. This exoskeleton is able to control the hip, knee and ankle flexion and extension, as well as the hip abduction and adduction. Second, the safe fall controller is found using optimal control, because this, theoretically, is able to find the best possible fall controller. The chosen optimal control strategy is reinforcement learning, this is because reinforcement learning has proven to solve complex problems in robotics (Pachocki et al., 2018). Moreover, reinforcement learning does not require knowledge of the model it is working with, which allows for generating a controller for a discontinuous model in which other optimal control strategies, such as model predictive control, cannot find a controller.

Several requirements were defined to test whether the final goal has been reached. The first requirement being that the controller lowers the head impact velocity as compared to the current locked joints controller for the forward and backward fall, preferably below 3.5 m/s. Improved further is to completely negate impact on the head as well as the torso and waist, as this typically results in the least damage overall (R1). To make sure the human pilot can predict what safe fall strategy will be performed and thus can act accordingly, each fall direction should result in a consistent safe fall strategy (R2). The created model should be based on the hardware of the Symbitron+ (R3).

Outside the scope of this work is the creation of a fall detection controller. It is assumed that another part of the total system is able to detect falls and if detected, activates the controller created in this work.

In this work, the Soft Actor-Critic reinforcement learning algorithm (Haarnoja et al., 2018) has been implemented. This algorithm is modified to work with the sparse and delayed rewards present in a fall. The initialisation of the algorithm also is improved such that it works better with inputs of different order of magnitude. The humanoid model as implemented by Roboti LLC (2018) is altered to only actuate the same joints as the Symbitron+ exoskeleton. Furthermore, the initial conditions of the model are changed such that it represents the start of a fall. The output of the model and thus the input to the reinforcement learning algorithm is changed such that it outputs realistic, measurable dynamic properties and sensor feedback. The returned reward is based upon the outcome of a literature review on the fall strategy methods for reducing

fall damage. The controller is evaluated both qualitatively and quantitatively, to evaluate among others the realism and severity of the fall.

The impact of this work is the creation of a controller for a safe fall in a lower limb exoskeleton that considerably lowers the impact of the head, torso and waist as compared to the current state of the art controller. Furthermore, this controller has been created using reinforcement learning, works for a 3D model and offers a solution for a range of initial conditions.

1.3 Outline

The outline of the report is as follows: First, the Theory, focusing on safe fall methods, is discussed in Chapter 2. Followed by the selection and description of the reinforcement learning algorithm in Chapter 3. Afterwards, the simulation set-up and the evaluation method are discussed in the Methods in Chapter 4. The results of the evaluation are presented in the Results in Chapter 5. The discussion on the methods and results is presented in Chapter 6. Finalising with the Conclusion and Future work in Chapter 7.

Chapter 2

Theory

As mentioned in the background, there is little research into falling with an exoskeleton. Luckily, however, research into human and humanoid robot falling does offer similarities with human-exoskeleton falling and therefore can offer insight into the topics despite lacking human-exoskeleton specific research. When looking at the field of humanoid falling, Ma et al. (2014) state three main aspects a safe fall controller consists of:

1. Fall detection
2. Fall direction
3. Fall strategy

The final two relate to the minimisation of damage to the environment and the minimisation of damage to itself, respectively, as discussed by Samy et al. (2017) and Yun et al. (2009). For humanoid falling, generally, the former is most important. This is because the fall direction has the chance to reduce the damage to human bystanders, which is not the case for the fall strategy. For the field of human-exoskeleton falling, however, the fall strategy has to reduce the damage to the pilot. Therefore, the priority for human-exoskeleton falling is more towards minimising the damage to itself and less towards minimising the damage to the environment. For that reason this related work focuses on fall strategies rather than fall directions. Note that, in certain cases fall direction might still be more important to, for instance, dodge a table. This, however, is circumstantial, whereas an adequate fall strategy always helps.

2.1 Fall detection

Before the fall strategy can be initiated, a fall has to be detected. Subburaman et al. (2019) give five requirements for fall prediction in a humanoid:

1. Generality to cope with disturbances applied in arbitrary directions and varied terrains
2. Robustness to different level of noises observed in various sensors
3. Agility to facilitate swift control actions
4. Reliability to minimise the failure rate
5. Versatility to handle different dynamic movements

The trade-off between reliability and agility of the detection offers a notable challenge. If the detection algorithm tries to be as reliable as possible it detects falls later, which results in less time to facilitate swift control actions. Focusing on agility, however, results in false positives which trigger the fall strategy even though balance could still be recovered. Implications for this work is that testing should include falls which could still be recovered from as well as falls which might make it harder to perform a safe fall strategy for.

As for the algorithm itself, there are a lot of different methodologies which can give some indication of fall probability. Examples given by Subburaman et al. (2019) are the Ground Projection of the Centre of Mass (GCoM), the Center of Pressure (CoP), the Zero Moment Point (ZMP), the Foot Rotation Indicator (FRI) point and the Zero Rate of change of Angular Momentum (ZRAM) point. Generally, a proper fall detection algorithm would include a combination of these variables. However, as mentioned in the introduction, the creation of a fall detection algorithm is outside the scope of this work and therefore will not be discussed further.

2.2 Fall direction

Looking into the fall direction, there are two main reasons for focusing on this aspect. First, is to avoid objects which would be in the direction in the fall and second, is to change the body orientation during the fall, to change a sideways fall into a forward fall, for instance. In general, exoskeletons themselves have no sensors which map the environment, this or some communication with the human pilot should be included to enable falling in a safe direction. The body orientation, however, can already be sensed by the exoskeleton or can be sensed by placing an IMU. Therefore, a safe fall controller could also be designed for changing the body orientation. Depending on where the controller wants to focus, this body orientation can be defined as either the pelvis orientation for better leg placement or torso orientation for better arm placement. Studies in humanoids give two main methods for changing the fall direction:

First, is changing the orientation or position of the leading edge of the base of support (Yun et al., 2009). When a humanoid starts to topple, its centre of pressure (point on a surface through which the resultant ground reaction force passes) touches the edge of the base of support. This is the leading edge or corner about which the humanoid rotates, thus changing the orientation or position of this leading edge therefore changes the direction in which the humanoid is falling. The easiest and most effective way to change the leading edge is generally by taking a step as this changes both the base of support as well as the centre of pressure.

Because the rotational inertia is dependent on the configuration, it can be changed to rotate the humanoid more towards the wanted direction (Lee and Goswami, 2011). This method, however, does little to change the fall direction as the rotational momentum stays the same. It can, however, allow the humanoid to spin about itself more quickly, enabling the humanoid to rotate a sideways fall towards a forward fall. This method, unfortunately, is hard to implement in a human-exoskeleton system as the rotational inertia is dependent on the complete configuration, including the joints controlled by the human pilot, which can oppose the wanted change in rotational inertia.

2.3 Fall strategy

Ma et al. (2014) and Lee and Goswami (2011) both discuss global methods for reducing the damage done by falling down for humanoids. Combining these findings result in three global methods to minimise the damage done by the ground impact after falling. A fourth method comes forward when taking the human pilot of the exoskeleton into account.

1. Minimise vertical kinetic energy
2. Fall on correct body segments
3. Maximise impact time
4. Reduce the risk of head impact

The methods used to minimise vertical kinetic energy has come forward in three ways. First, Fujiwara et al. (2003) used the increase of angular momentum to slow down the rotation about

the ankle. This usually was done by first flexing the knee joint, which decreases the moment arm and thus decreases the rotational acceleration. After a certain angle with the ground has been reached the knee joint would extend itself again increasing the angular momentum. This, however, should not be done too early as this does increase the moment arm and would therefore increase the rotational acceleration. The second method is to create a vertical reaction force from the ground (Robinovitch et al., 2000). This also happens during the extension of the knees and thus decreases the vertical kinetic energy even further. During the extension of the knee, however, slippage should be taken into account, because losing grip on the ground could worsen outcomes (Khalili et al., 2017). The final method is discussed by Yun and Goswami (2014), where the goal is to convert as little potential energy into kinetic energy by catching the body as high up as possible. If a wall or other object is within reach it is also possible to grasp towards that and stop the conversion of potential energy in that way as stated by Wang and Hauser (2017). Once the body has landed on either the knees or hips a final method is also possible, which is to create a counter-rotational torque in the knee or hip joint, respectively (Fujiwara et al., 2004). This is not possible earlier during the fall, because the ankle joint does not have the moment arm to compensate for the fall. If the ankle joint were able to generate the required moment there should not have been a fall, and the ankle balance strategy should have been implemented to prevent the fall (Horak and Nashner, 1986).

Falling on the correct body segments generally is a two-step process. First, the humanoid should be rotated such that the wanted body segments are orientated in the direction of the fall. For this the methods mentioned in the fall direction section can be used. Second, when the humanoid is falling in the correct direction, the wanted body segments can be rotated forward to catch the ground as early as possible (Yun and Goswami, 2014). The reason to do this as early as possible is for the the centre of mass to remain as high as possible, which in turn means that less potential energy is converted into kinetic energy. Note that, on the other side of this method is to prevent unwanted body segments from hitting the ground. Khalili et al. (2017) did this by rotating the unwanted body segments (in this case the head) as far away as possible, which is the inverse of the earlier mentioned rotating towards the ground of wanted body segments. This strategy of keeping the head from hitting the ground was quantified by Fujiwara et al. (2006) by trying to minimise the excess angular momentum of the trunk at knee-ground contact in forward falling.

The third method for minimising damage, maximising the impact time, presented itself in two ways in the found literature. Firstly, by adding padding to the designed impact locations, usually knees, hip and hands (Lee and Goswami, 2011). This padding will compress during impact, which lengthens the impact time as compared to the impact time between two completely rigid bodies. And second, by changing the control of the joints to be more compliant from the impact onwards (Fujiwara et al., 2004). Note that for a compliant controller to work, it is necessary for the body segments to not be in line with the ground reaction force as this does not create any moment to be compliant for (Samy and Kheddar, 2015). Both methods can be seen as different ways of adding a compliant structure to the humanoid either physically or by applying the correct controller.

The last method, reducing the risk of head impact, only comes forward when looking at falls with humans involved and is therefore not mentioned in any humanoid research. This method is quite similar to using the second method and uses a strategy which avoids falling on the head. If, however, the head still ends up close to the ground a slight difference in push, noise or fall strategy might have resulted in hitting the head. To prevent this, a margin should be implemented, which would allow for these differences and still result in no head impact. This is especially important in the case of a human-exoskeleton system as the separate systems do not communicate during a fall, which might result in conflicting fall strategies (Khalili et al., 2017).

Chapter 3

Reinforcement learning algorithm

In this chapter the selected reinforcement learning algorithm is discussed. Followed by the description of the base algorithm. The changes made to the base algorithm are discussed in the Methods (see Chapter 4.1.1). A list of reinforcement learning terms can be found in Appendix B.

3.1 Selection

For the selection of the algorithm a set of requirements has been created. These requirements are based upon the environment the agent learns in and the probability of the algorithm finding a working solution for the fall environment. The algorithm

P1. must work with continuous state and action spaces.
P2. must be able to output multiple action signals.
P3. should have a good solution to the exploration-exploitation challenge.
P4. should have a proven track record of functioning in complex, continuous environments.

Using these requirements, the choice was made to use the Soft Actor-Critic (SAC) algorithm (Haarnoja et al., 2018). The SAC algorithm is based upon a general Actor-Critic algorithm,

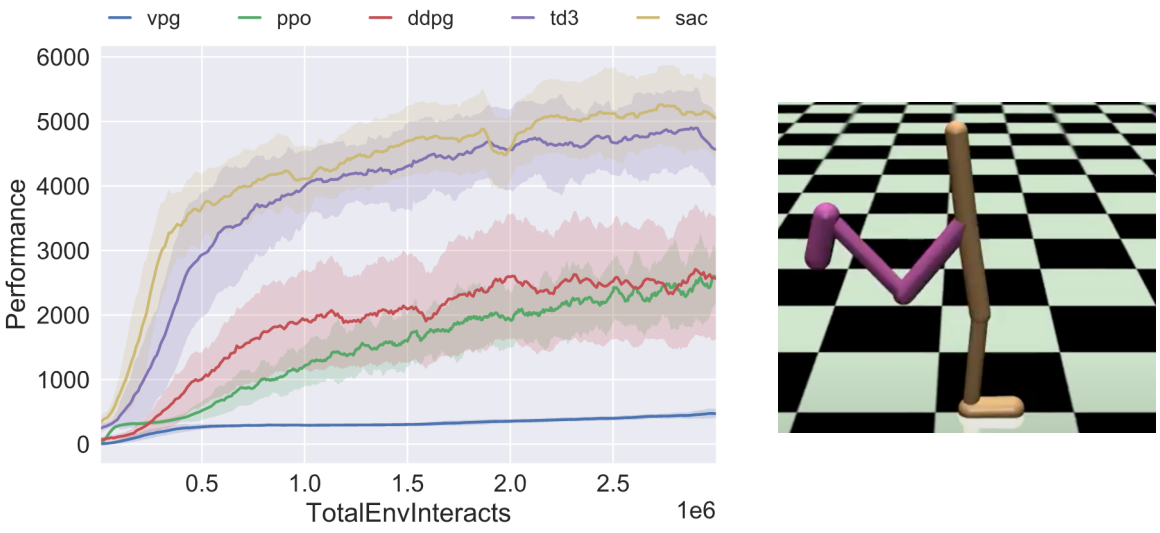

Figure 3.1: Left: Benchmark of various algorithms on the 2D Walker Gym environment, found in: *https://spinningup.openai.com/en/latest/spinningup/bench.html*. Right: Video frame of the Walker2D environment

however, instead of directly outputting a single action, the policy outputs a probability distribution over actions by providing the mean and standard deviation. The action is then drawn from a normal distribution with this mean and standard deviation. This allows for the exploration to be directly included in the policy. The loss function of the policy includes both the action value as well as a value related to the entropy (or randomness) of the action. This means that the agent is rewarded both for exploiting the environment, but also for exploring as much as possible. The algorithm, thus, has a solution to the exploration-exploitation problem by allowing the agent to limit its own exploration when it converges to a good solution (P3).

The left side of Figure 3.1 shows a benchmark performance by Open AI for various algorithms on the 2D Walker environment[1]. The right side of Figure 3.1 shows a video frame of the 2D Walker. The goal of this environment is to walk as far right as possible, while keeping the upper body upright. This is done by controlling the hip, knee and ankle joint. The selected algorithms for the benchmark were, Vanilla Policy Gradient (VPG), Proximal Policy Optimization (PPO), Deep Deterministic Policy Gradient (DDPG), Twin-Delayed Deep Deterministic Policy Gradient (TD3) and Soft Actor-Critic (SAC). The used implementations and documentation can be found at the Open AI website[2]. The benchmark shows that the SAC algorithm is performing best (P4). The TD3 algorithm is a close second, however, the TD3 algorithm was shown to perform worse on other environments, such as the walking humanoid (Haarnoja et al., 2018). The SAC algorithm is created specifically for continuous state and action spaces (P1) and is able to output multiple actions (P2).

3.2 Soft Actor-Critic algorithm

The implementation of the SAC algorithm used in this work is based upon the implementation of Vaishak V. Kumar (Kumar, 2019). This implementation has been created using the PyTorch library[3]. An overview of the learnable networks in this implementation is shown in Figure 3.2. Note that in this figure the inserted states and actions are excluded for clarity purposes. Furthermore, α (relative entropy weight) is not a learnable network, but a learnable scalar variable. All the neural networks feature two hidden layers with each 512 nodes, the nodes in the hidden layers all feature a rectifier linear activation (see Equation 3.1).

$$y = \max(0, x) \tag{3.1}$$

[1] https://gym.openai.com/envs/Walker2d-v2/
[2] https://spinningup.openai.com/en/latest/user/introduction.html
[3] https://pytorch.org/

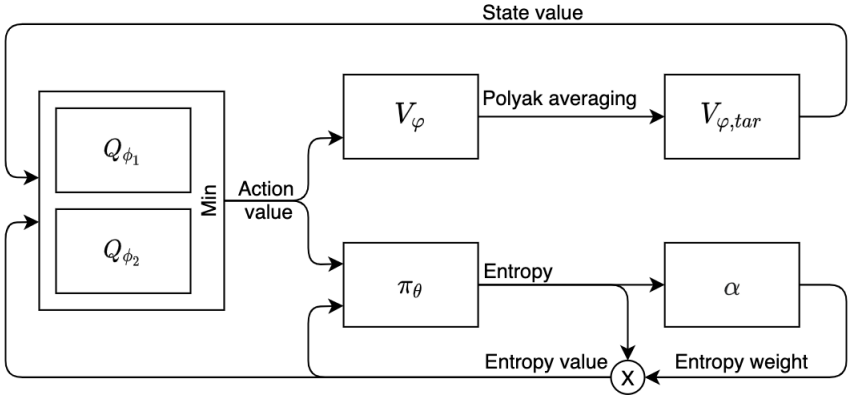

Figure 3.2: Overview of how the various networks influence each other. $Q_{\phi_{1,2}}$ refers to both action-value functions, V_φ and $V_{\varphi,tar}$ refer to the (target) state-value function, π_θ to the policy and α to the relative entropy weight.

The policy network outputs a deterministic mean and logarithmic standard deviation of the probability density function (PDF) to later draw the action from. The reason for not drawing the action value directly from a stochastic policy is because by giving the PDF the output of the policy network is deterministic, even though the actual taken action is not. This is known as the reparametrisation trick (Haarnoja et al., 2018). Being deterministic is required to be able to perform learning using stochastic gradient descent. The reason for a logarithmic standard deviation is because the logarithm is defined for all real numbers, whereas the regular standard deviation is undefined for all negative values, which can cause troubles with the gradient descent. Finally, to make sure the chosen action value is within the possible action range, a hyperbolic tangent function is used to map all real numbers to a value between -1 and 1. Therefore, the action range is normalised to map this range to the actually used values.

As is shown in Figure 3.2, there are two Q-values. The reason for this is so the minimum of the two can be taken to prevent overestimation of the Q-value. The sole difference between the two Q functions is in the initialisation, in which the weights for both Q functions are drawn separately. This difference is enough to severely limit the probability of overestimation of the action-value during the training phase (Haarnoja et al., 2018).

Training

The training of the algorithm is done by updating the weights of the various neural networks by use of stochastic gradient descent. In order to use stochastic gradient descent, the gradient of a loss value to the network weights has to be calculated. The gradient to the loss values are automatically calculated by the `backward` function of the PyTorch library. An update step is taken at each time step during training.

In this section, the Q-values are referred to as $Q_{\phi_i}(s, a)$, where i refers to the respective Q-networks and the subscript refers to the neural network weights. The state-value is referred to as $V_\varphi(s)$ and the target state-value is referred to as $V_{\varphi,tar}(s)$. Finally, the policy network is referred to as $\pi_\theta(s)$ and the chosen action is referred to as $\tilde{a}_\theta(s, \xi)$ where $\xi \sim \mathcal{N}(\mu_\theta(s), \sigma^2_\theta(s))$ in which $\mu_\theta(s)$ and $\sigma_\theta(s)$ are the mean and standard deviation given by the policy. The used states (s_k), actions (a_k), rewards (r_k), next states (s_{k+1}) and done (D_k) values are drawn from the replay buffer, where k is used to describe a single transition and K refers to a complete, randomly-selected batch of transitions.

First, the Q-value loss is calculated using the mean squared error of the Q-value with the Bellman equation.

$$Q_{i,loss}(s_K, a_K, r_K, s_{K+1}, D_K) = \frac{1}{K_{size}} \sum_{k \in K} (Q_{\phi_i}(s_k, a_k) - Q_{\varphi,tar}(r_k, s_{k+1}, D_k))^2 \tag{3.2}$$

where $Q_{\varphi,tar}(r_k, s_{k+1}, D_k)$ is given by the Bellman equation:

$$Q_{\varphi,tar}(r_k, s_{k+1}, D_k) = r_k + (1 - D_k)\gamma V_{\varphi,tar}(s_{k+1}) \tag{3.3}$$

with $\gamma = 0.99$ being a discount term.

Next, the loss of the value function is calculated using the mean squared error with a calculated target value (note, not the target state-value function):

$$V_{loss}(s_K, \tilde{a}_\theta(s_K, \xi)) = \frac{1}{K_{size}} \sum_{k \in K} (V_\varphi(s_k) - V_{target}(s_k, \tilde{a}_\theta(s_k, \xi)))^2 \tag{3.4}$$

Where the target value is calculated as follows:

$$V_{target}(s_k, \tilde{a}_\theta(s_k, \xi)) = \min_{i=1,2} Q_{\phi_i}\left(s_k, \tilde{a}_\theta(s_k, \xi)\right) - \alpha \log \pi_\theta\left(\tilde{a}_\theta(s_k, \xi) | s_k\right) \tag{3.5}$$

This target value can be explained as follows, the Q-value gives the value of the state (given the chosen action) and $\log \pi_\theta$ is the logarithm of the probability of choosing the action given the policy (as calculated in Equation 3.6). As a lower probability means that it is more random this value is subtracted (higher value for being more random). The relative weight between the value gained by achieving a high reward (Q-value) and the value for being more random is given by α. Note that later, this target value returns in the policy loss (taking the minus sign in the loss function into account).

The logarithm of the probability is calculated as follows (Kumar, 2019):

$$\log \pi_\theta \left(\tilde{a}_\theta(s_k, \xi)|s_k\right) = \log(p(\xi|\mathcal{N}(\mu_\theta(s_k), \sigma_\theta^2(s_k)))) - \log(1 - \tilde{a}_\theta(s_k, \xi)^2 + \epsilon) \tag{3.6}$$

Because an action is chosen for every actuator, multiple actions are chosen. This means that the above equation returns multiple probabilities, however a single value is required for the loss function. Joining the probabilities together is done by taking the product of the probabilities (assuming independence of drawn actions). Due to taking the logarithm of the probabilities this product changes into a sum (logarithmic multiplication rule). To make the solution independent of the amount of actuators and thus more general, the joined logarithmic probability is divided by the amount of actuators. Combining the sum of probabilities and the division by the amount of actuators together results in taking the arithmetic mean.

The loss of the policy is calculated using the mean error:

$$\pi_{loss}(s_k) = \frac{1}{K_{size}} \sum_{k \in K} \left(\alpha \log \pi_\theta \left(\tilde{a}_\theta(s_k, \xi)|s_k\right) - \min_{i=1,2} Q_{\phi_i} \left(s_k, \tilde{a}_\theta(s_k, \xi)\right) \right) \tag{3.7}$$

Note that, the higher the probability, the higher the loss (try to be as random as possible) and the higher the perceived reward as described by the Q-value, the lower the loss (try to maximize the reward). The relation between the two values is given by α.

This α (relative entropy weight), thus determines the focus of the policy on being more random or on achieving a higher reward, where a higher α means a higher focus on being more random. α is updated as follows, note that this is not a loss function, but the complete update as α is not a neural network, only a single variable (Haarnoja et al., 2018).

$$\alpha = \alpha + \mathrm{lr}_\alpha \frac{1}{K_{size}} \sum_{k \in K} \left(\log \pi_\theta \left(\tilde{a}_\theta(s_k, \xi)|s_k\right) - H_{min}\right) \tag{3.8}$$

Where lr_α is the learning rate of α (in literature typically also α) and H_{min} is the minimum required entropy (which is also the maximum required probability) and a hyperparameter. This function, therefore, increases α when the entropy is too low (it increases the value of entropy) and does the opposite when entropy is too high. This ensures a set amount of entropy throughout training. It is, however, possible to have a lower entropy (higher probability) at states which require precision and compensate with a higher entropy (lower probability) at less important states. The minimum required entropy is independent of the amount of actuators, because the total probability is divided by the amount of actuators.

Finally, the weights of the target value function is updated using Polyak averaging with the regular value function:

$$\varphi_{tar} = (1.0 - \tau)\varphi_{tar} + \tau\varphi \tag{3.9}$$

with $\tau = 0.01$.

To start the training and fill the replay buffer, the first actions are not given by the policy, but are randomly drawn from a uniform distribution ranging the whole action space. In this work, these observation frames have been selected to cover the first 1000 actions (5 episodes).

Chapter 4

Methods

4.1 Simulation set-up

A general reinforcement learning algorithm has an agent which interacts with an environment. In order to give a complete overview of the simulation, all elements of this agent-environment system (as given in Figure 4.1) are discussed. Starting with the modifications made to the earlier introduced Soft Actor Critic algorithm. Followed by, the humanoid model created in MuJoCo[1], a physics simulation software optimised for reinforcement learning, programmed in XML. Finalising with both the reward and the observation functions, that are created in an Open AI Gym's environment[2], which provides hooks to the MuJoCo simulation, programmed in Python.

4.1.1 Soft Actor-Critic modifications

Reward

The reward signal used to evaluate a fall mostly consists of sparse and delayed rewards as the reward is dependent on the ground impact to properly evaluate the fall. The reward signal is divided into two categories, discounted future reward and immediate reward. The discounted future rewards refer to reward signals which come forward in a future time step, but are influenced by previous time steps, such as the ground impact. Immediate reward refers

[1] http://www.mujoco.org/
[2] https://gym.openai.com/

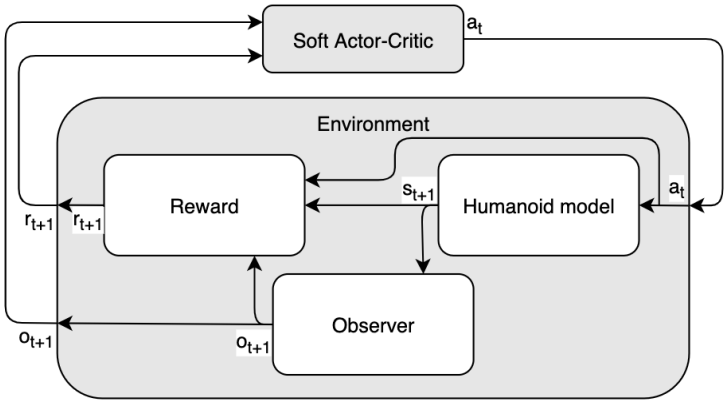

Figure 4.1: Agent-environment overview used in this work. The Soft Actor-Critic is the agent and the environment consists of the humanoid model, which is the dynamic process and a reward and observation function, which return a reward and an observation to the agent, respectively.

to the reward which is only influenced by the choices made during that time step, such as the actuation cost. There are two reasons for implementing a discounted future reward. First, the low-pass filtered action signal and the inertias present make the movement of the humanoid dependent on the previous action signals. And second, because of the delayed reward problem, which is a fundamental problem in reinforcement learning (Sutton and Barto, 2018), where the actual reward to a certain action is only known further in the future.

The discounted future reward is an adaptation to the forward view eligibility tracing solution as presented by Li et al. (2008) and is used to provide a solution to this delayed reward problem. Experience replay presents the problem that the correlation between transitions has to be learned by a state and/or action value function. In theory, when a value function has been learned correctly, the Bellman equation can be used to estimate the value of a transition taking into account the value of the next state, which also takes into account the value of its next state, etcetera. The issue here lies in the fact that for this to work the Bellman equation is dependent on the correct estimation of a value function. As long as these value functions cannot identify that the humanoid model is going to have an impact from the fall, the Bellman equation does not return a low value.

The forward view eligibility tracing solution given by Li et al. (2008) replaces the reward at a given time step with a reward containing the discounted sum of future rewards up to a time horizon, after which an estimator is used to determine the value of the future rewards. The goal of this forward view eligibility tracing method is thus, to estimate the value of the current state, because it uses an estimator to compensate for a finite time horizon. Another eligibility tracing implementation is by Daley and Amato (2019), who use learning with mini-batches where the values of the mini-batches are updated upon opening the mini-batch. Afterwards, the return is calculated based upon the updated values. The goal here aside from eligibility tracing is to be able to re-use old transitions by re-evaluating their value.

The goal of modifying the reward in this work is to propagate high impacts backwards to the actions responsible. Therefore, the estimator can be left out of the reward modifications. Additionally, because the value functions are slow to learn whether an impact is imminent, it is unwanted to use value functions for the reward modifications. Furthermore, a distinction is made between immediate and discounted future rewards, so this has to be split up as well. Combining these requirements, results in each reward being updated as follows:

$$r_t = r_{imm,t} + \sum_{k=0}^{T} \frac{\lambda^k}{\sum_{k=0}^{T} \lambda^k} r_{future,t+k} \tag{4.1}$$

Where λ is a hyperparamater controlling the discount factor of future rewards and the division by the sum of the discount factor is used to keep the reward in the same order of magnitude. T is the time horizon, which if zero makes it such that the regular reward is returned. If T is set to a large value, issues might be encountered with a begin-of-episode clip-off (as $r_{future,0}$ is only referenced by r_0, thus it might be preferable to get a low reward earlier as a less low reward later). It is also possible to implement this function in an environment with infinite time horizon. Then the return of a time step would be calculated after all T future time steps have been taken, basically inserting the transition into the replay buffer with a delay of T time steps. However, in this work the new rewards are calculated at the end of each episode.

The advantage of this method is that no extra calculations from the value function are required to calculate future values and the future rewards are completely accurate as these have been returned from the environment. The disadvantage of this method is that the achieved rewards are based upon an older version of the policy, thus a newer policy might achieve a higher reward for the same state, due to better actions later. To minimise this disadvantage the chosen replay buffer is of a smaller size (20,000 time steps or 100 falls) as compared to other research.

Initialisation

An important aspect of reinforcement learning is how everything is initialised, because a detrimental initialisation might lead to being stuck in a local optimum for the whole training. In the case of the humanoid an issue arises where the inputs are of different order of magnitude (over a factor 100 difference). If the same uniform distribution is used for the initialisation of all the weights, the lower inputs never affect the output significantly and therefore are not trained towards the right direction. To prevent this the initialisation of the input layer is dependent on the maximum value of the input, besides also being dependent on the amount of inputs. As described in the following equation:

$$w_{(i,j)} \sim U\left(-\frac{1}{\sqrt{n_u}\max(|u_i|)}, \frac{1}{\sqrt{n_u}\max(|u_i|)}\right) \tag{4.2}$$

where $w_{(i,j)}$ is the weight connecting input i to node j, n_u is the number of inputs and $max(|u_i|)$ is the maximum absolute value of input i. Because the weights are uniformly drawn from the distribution the total output towards each node of the first hidden layer is approximately between $-1/\sqrt{n_u}$ and $1/\sqrt{n_u}$. The initialisation of the other layers can therefore use the standard PyTorch initialisation:

$$w_{(i,j)} \sim U\left(-\frac{1}{\sqrt{n_u}}, \frac{1}{\sqrt{n_u}}\right) \tag{4.3}$$

4.1.2 Humanoid model

The dynamic process used in this program is based upon the Humanoid example provided by MuJoCo. The standard initial position of the humanoid model is shown in Figure 4.2. This standard initial position can be manipulated such that it represents the start of a fall.

There are a total of 21 joints defined in the humanoid model, which are listed in Table 4.1. Because the focus of this work is in falling in an exoskeleton, the simulation only has control over the same joints as the Symbitron$^+$ exoskeleton. These joints are for hip abduction/adduction and flexion/extension, knee flexion/extension and ankle flexion/extension for both the left and right leg. Because it is difficult to also model human behaviour, the remaining joints (abdomen, shoulders and elbows) are modelled as if the human pilot is co-contracting all muscle pairs resulting in a stiff spring-damper system (75 Nm/rad and 2.5 Nms/rad, respectively (Lee and Ashton-Miller, 2011)), so the initial positions are chosen as the joint positions throughout the fall. The remaining joints, the hip (external/internal rotation) and ankle (pronation/supination) muscles are passive as the human pilot and the exoskeleton typically have no control over them.

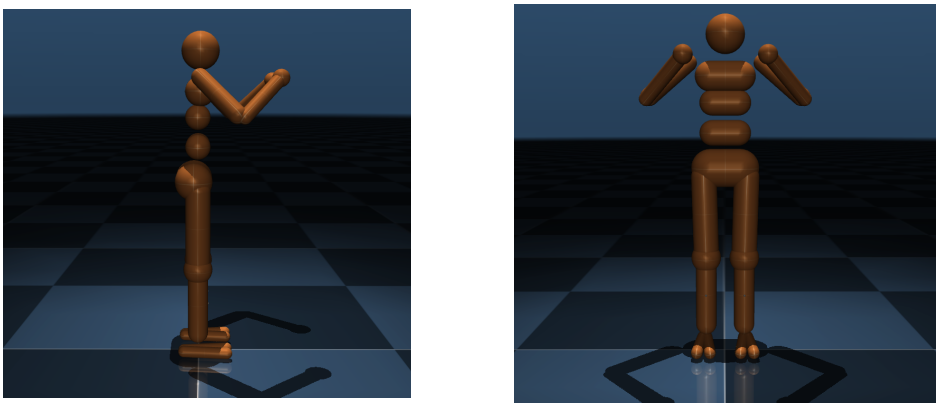

Figure 4.2: Side (left) and front (right) view of the standard initial position with no fall direction, initial rotation with ground or initial joint positions defined.

As there still is some passive stiffness and damping of the muscle tissue some minor stiffness and damping is still present, therefore the stiffness and damping of these are kept the same to the original humanoid model (5 Nms/rad damping and 10 Nm/rad stiffness for the hip external/internal joint and 4 Nm/rad stiffness for the ankle pronation/supination joint).

To increase realism of the torque sources the actuation signals are low-pass filtered using a cut-off frequency of 30 Hz, which approximates the implemented behavior of the controlled joint (Rampeltshammer et al., 2020). This creates a phase lag between the actuation signal and reaching the desired torque.

Finally, the initial positions of the humanoid model are defined. Because reinforcement learning is known to exploit initial positions, most initial positions are defined as a range from which the initial positions are uniformly drawn. Generating an initial position as used for the safe fall training is then defined as follows:

1. The direction of the fall is defined by the angle with the forward axis in the horizontal plane, which is used to create a forward ($0\pm15°$) or backward ($180\pm15°$) fall. The base frame of the humanoid is then rotated towards the ground by 5 to 15 degrees in the direction of the fall.
2. The initial translational velocity in the direction of the fall is set to 0.3 to 0.5 m/s and the initial rotational velocity about the rotational axis perpendicular to the fall is set to 0.6 to 1.0 rad/s (rotating in the direction of the fall).
3. The initial positions of all joints above the pelvis (abdomen, shoulder and elbow joints) are drawn from a uniform distribution spanning its entire range of motion. The spring reference point is then set to the initial position.
4. To also train for varying masses, the weight is chosen uniformly between 70 and 90 kg.

The first two items are to create the fall and are chosen empirically. The criteria for the initial configuration are such that the slowest fall will most likely not trigger the fall detection to ensure that the slowest fall will be included and the fastest fall will take about 0.6 s until impact when uncontrolled, which is approximately the average time until pelvis contact taken from start of descent (Choi et al., 2015). The extremer cases were left out, because the earlier mentioned

Joint	Movement direction	Range of Motion	Active	Maximum Torque
L/R Hip	Abduction/adduction	30°	Active	100 Nm
L/R Hip	Flexion/extension	140°	Active	70 Nm
L/R Hip	External/internal	95°	Passive	-
L/R Knee	Flexion/extension	160°	Active	70 Nm
L/R Ankle	Pronation/supination	100°	Active	100 Nm
L/R Ankle	Inversion/eversion	100°	Passive	-
Abdomen	Lateral flexion	70°	Co-contracted	-
Abdomen	Flexion/extension	90°	Co-contracted	-
Abdomen	Vertical rotation	105°	Co-contracted	-
L/R Shoulder*	Mostly abduction/adduction	145°	Co-contracted	-
L/R Shoulder*	Vertical & horizontal flexion/extension	145°	Co-contracted	-
L/R Elbow	Flexion/extension	140°	Co-contracted	-

Table 4.1: List of all the joints in humanoid model. *Because the shoulder joint only consists of two simple joints instead of three the movement axes are combined.

cases are more likely to occur and to ensure that a safe fall strategy would be possible for every configuration. The third item is to prevent exploitation of the initial position of the upper body. The joint-specific ranges are found in the original environment created by MuJoCo. The spring reference points are set to the initial position to also train for different methods of catching, making the controller more robust to different human pilot fall strategies. Two examples of initial positions for forward falls are shown in Figure 4.3.

Afterwards, the episode is run for 2 seconds, which is 200 time steps during training. The control frequency is set to 100 Hz during training, instead of the 1000 Hz used in the Symbitron$^+$ exoskeleton, to speed up training.

4.1.3 Reward

The reward is based upon the earlier identified methods for reducing the damage done by falls (see Chapter 2.3). To dissuade the agent from choosing an unnecessary action a term based upon actuation signal is also included.

Body impact

The first term is based upon falling on the correct body segment and minimising total impact force. It is calculated as follows:

$$R_{Fimpact} = -c_1 \sum_b w_{F,b} F_{c,b} \tag{4.4}$$

where c_1 is the weight of the contact force reward chosen empirically, $w_{F,b}$ is the body-specific weight (importance) of the body impact and $F_{c,b}$ is the impact force on the body. This term in theory solves for three of the identified methods, as a lower kinetic energy or a longer impact time both lower the impact force and the relative weight focuses on falling on the wanted body segments.

Vertical kinetic energy

The second term is based upon the vertical kinetic energy, returning the value of the vertical velocity times a weight based upon the relative importance of the body, all squared as shown in Equation 4.5.

$$R_{Ekin} = c_2 \operatorname{sign}(p_k - p_{k-1}) \frac{(p_k - p_{k-1})^2}{dt} \text{ , where } p_k = \frac{\sum_b w_{E,b} h_{b,k}}{\sum_b w_{E,b}} \tag{4.5}$$

where c_2 is the weight of the kinetic energy reward and p_k is the weighted centre of mass height at time step k. In which, $w_{E,b}$ is the weight of the vertical kinetic energy assigned to each body

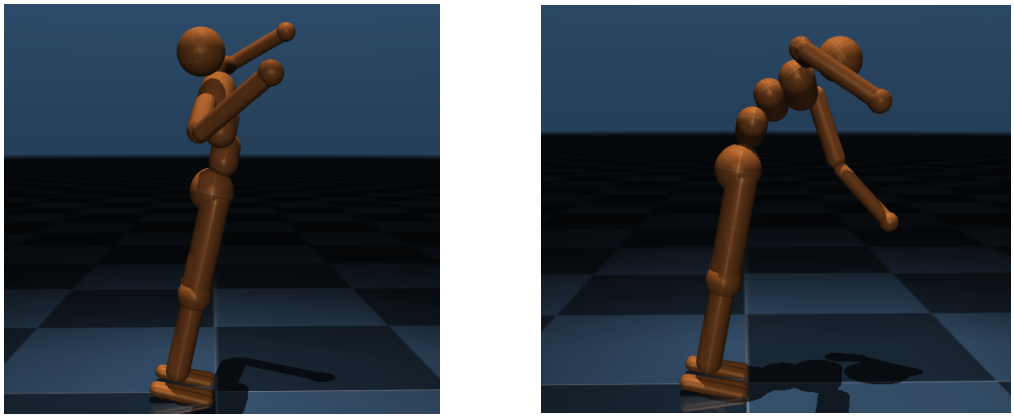

Figure 4.3: Two examples of initial positions of the "forward fall"-scenario.

and $h_{b,k}$ is the body height. The mass of the body is included in the weight of the body. Using the kinetic energy means that the velocity is squared, this is because the higher velocities are also exponentially worse. Squaring should therefore deviate the agent from learning high velocities, which might result in breaking. The advantage of this term with respect to the impact force term is that it gives a continuous reward, instead of the discrete impact.

Risk of head impact

The risk of head impact is given by the height of the head, because a higher head position would add robustness to small errors, and by the velocity of the head, as no speed but a low height still results in hardly any risk. The equation used, therefore, is defined as follows:

$$R_{head} = -c_3 \max\left(\min\left(h_{head,k} - c_h,\ 0\right) \cdot \left(\frac{h_{head,k} - h_{head,k-1}}{dt}\right),\ 0\right) \tag{4.6}$$

where c_3 is the weight of the head risk reward, $h_{head,k}$ is the height of the midpoint of the head at time step k and c_h is a constant height below which this reward is non-zero. The min function makes it such that this reward is zero for every height above c_h, meaning that the risk of head impact is low enough that it is insignificant to take into account. The max function makes it such that when the head is moving away from the ground no positive reward is given as there is no such thing as negative risk.

Actuation cost

In order to stimulate the agent to only give an action signal when necessary a negative reward on the actuation signal is given by a standard quadratic cost as shown in the following equation:

$$R_{act} = -c_4 \sum_m a_m^2 \tag{4.7}$$

where c_4 is the weight of the actuation reward and a_m is the activation signal (not applied torque) of motor m. This actuation cost is the only immediate reward as defined in the reward modification of the agent (see Section 4.1.1), the other rewards are all defined as discounted future rewards.

4.1.4 Observations

The observations returned from the environment to the agent are chosen to have realistic values, which can be returned by the sensors of the exoskeleton (possibly by adding some additional sensors) as well. These observations are:

- Actuator rotational position, rotational velocity and applied torque
- Total mass
- Sensor data at shins, wrists, upper and lower back, consisting of:
  - Translational accelerometer
  - Gyroscope
  - Translational velocity meter
  - Gravity sensor

Note that the velocity meter and gravity sensor are not actual, realistic sensors but the values for these sensors can be constructed from the other sensors. The observations used during training and testing are all noise free measurements. A test on the impact of input noise on the outcome is found in Appendix A.

4.2 Evaluation method

To judge the outcome of the reinforcement learning implementation, a two-fold evaluation is chosen. First, the outcome is evaluated qualitatively, based upon defined evaluation criteria. And second, the outcome is evaluated quantitatively, based upon the impact force and velocity on the body parts.

Test cases

To ensure that all outcomes are evaluated identically a set of initial test conditions have been created. These test conditions are based upon a standard initial position, which is the first test case and can be seen in Figure 4.4 for the forward fall. This standard case features the average of the initial positions, velocities and total mass used during training (T_0). The other test cases change a single feature of the standard case as follows:

- The mass is set to the minimum and maximum value (T_{M-} & T_{M+})
- The initial rotation toward the ground and its initial translational and rotational velocity are set to the minimum and maximum value (which should in theory result in the softest and hardest fall) (T_{F-} & T_{F+})
- The initial rotation about the vertical axis (the fall direction) is set to the minimum and maximum value, this results in falling more towards the left or right. (T_L & T_R)
- The upper body configuration is changed:
  - Arms kept to the side of the body (cannot catch upper body with the arms) (T_{AS})
  - Arms aimed backwards (T_{AB})

The tests are performed using only the mean value of the policy, because this is the action the policy on average thinks is best. Moreover, the control frequency during the tests is set to 1000 Hz, which is the control frequency used in the Symbitron$^+$ exoskeleton.

Qualitative evaluation

The qualitative evaluation will judge the outcome of the test cases based upon a set of evaluation criteria. These criteria can be scored with either a 1 (minimum), 2 (middle) or 3 (maximum). In Table 4.2, the criteria and the corresponding rubric can be seen. Note that some of these criteria have very strict indicators, such as E2. However, other criteria use a general idea to describe what to look for, such as E5.

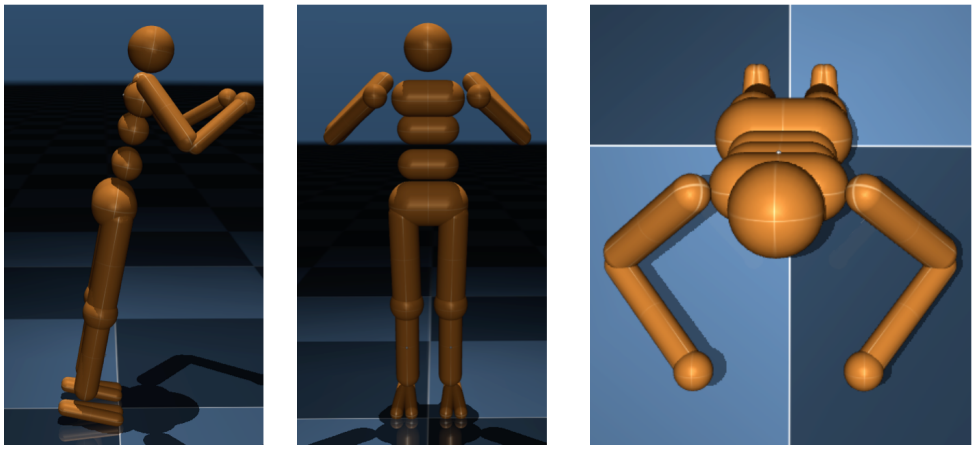

Figure 4.4: Side (left), front (middle) and top (right) view of the standard initial position of the forward fall.

Criterion	Minimum (1)	Middle (2)	Maximum (3)
E1. Severity of fall	• Individual limb impact • Body segments perpendicular to surface at impact • Does not use ground contact to slow down upper body rotation	• Shared limb impact at lower limb impact • Body segments perpendicular to the ground at impact • Slightly uses ground contact to slow down upper body rotation	• Shared limb impact at all impacts • No body segments perpendicular to ground at impact • Uses ground contact to (almost) stop upper body rotation
E2. Limbs exposed to impact	• Head is exposed to impact	• Torso or waist is exposed to impact	• Only distal limbs are exposed to impact
E3.. Risk of head impact	• Has head impact	• Head close to the ground	• Head not close to the ground
E4. Smoothness	• Jitter throughout the fall	• No jitter during the fall	• No jitter during and after the fall
E5. Realistic / natural fall	• Very fast, immediate movements • Joints are at their maximum range forcing into hyperextension or -flexion • Overall bad, unnatural impression	• Some fast, immediate movements • Joints are at their maximum range without forcing into hyperextension or -flexion • Overall natural impression as if a normal person is falling	• All gradual movements • No joints are near their maximum range • Overall natural impression as if a trained person is falling
E6. Dependence on upper body for safe fall	• Is not saved by upper body	• Is saved by upper body	• Does not rely on upper body

Table 4.2: List of qualitative evaluation criteria and their respective indicators.

Quantitative evaluation

The quantitative evaluation determines the quality of the falls by examining the impact force and impact velocity of the body parts of the humanoid model. In order to give meaning to these quantities the results are compared to the state of the art controller, which is achieved by applying a stiffness (75 Nm/rad) and damping (2.5 Nms/rad) to the actuator joints, similar to the used upper body stiffness and damping values. This was done by removing the actuation from the dynamic process and applying the mentioned values to the humanoid model. Note that, due to the difficulties of physical impact modelling, the impact force is an indication and only has relevance in comparison with the uncontrolled fall impact forces, it does not show actual impact forces.

The minimum goal is to lessen the impact on the head, preferably to negate the impact on the head. The impact refers to both the impact force and impact velocity, the reason for both values is to verify the outcomes. A better outcome would be less impact on the torso and waist as well. The best outcome would be a generally lower impact across all body parts.

Generalisation to other falls

To show how the found policies generalise to falls outside of the training range, both policies are tested on falls in all directions and with harder and softer falls. The fall direction is divided into sections of 5° and each direction is divided into eight increasingly harder falls. The hardness of the fall is increased by increasing the initial rotation towards the ground and the initial translational and rotational velocity. Starting with an initial rotation of 2.5°, initial translational velocity of 0.25 m/s and initial rotational velocity of 0.5 rad/s. Increased by 2.5°, 0.05 m/s and 0.1 rad/s for each step, respectively. This results in starting with a softer fall as T_{F-} and ending with a harder fall as T_{F+}. The maximum impact force on the head for each fall functions as the evaluated result. This test is run for the arms forward (T_0), arms sideways (T_{AS}) and arms backwards (T_{AB}) test cases.

Training set-up

Training is done on a remote server provided by the local institution. The server runs Ubuntu version 19.10, has 32 GiB RAM, 2 Intel Xeon CPU E5-2630 v4 processing units and an Nvidia TITAN Xp graphics card. The hyperparameters and weight values used during training can be found in Appendix C.

Chapter 5

Results

5.1 Forward fall controller

After having learned for 874,400 time steps (or 4372 falls), the generated safe fall strategy used by the forward fall controller is to first fall on the knees early, such that the upper body has to rotate over it. Followed by the upper body rotating forwards and being caught by the arms (see Figure 5.1). Due to the early knee impact the upper body generally is behind the knees, which slows down falling and makes it easier for the upper body to catch itself. A video of the forward fall controller performing all the test cases is included in the Appendix.

A better insight into the policy is given by the actuator torques during the fall with the T_0 initial position (see Figure 5.2). In the figure three time steps are indicated, these relate to the knee impact, hand impact and zero upper body velocity, respectively.

Looking at the hip abduction / adduction shows that the left leg moves outward during the fall, resulting in a slightly wider base of support after knee impact. After knee impact the ground friction on the knees is too large, restricting the sideways movement of the hips.

The hip flexion / extension figure shows that first a slight extension takes place, rotating the upper body upwards. After approximately 200 ms the hips start flexing. At this point there is little to no ground contact, which means that besides rotating the upper body forward, this mostly also rotates the legs forward as the mass of the legs is lower as the upper body mass. After knee impact the right hip provides an extending torque, counteracting the forward upper body rotation.

The knee joint starts off by flexing, resulting in the bending of the knees, which ensures knee impact. This amount of knee torque results in slightly lifting the feet off the ground, which allows the forward rotation of the upper leg by hip flexion. This combination ensures that the knees are in front of the upper body at knee impact, this allows for counteracting the forward upper body rotation.

The ankle joint first provides a slight extension torque, which increases the ground reaction force. This allows for the hip to first extend itself to rotate the upper body upwards. Afterwards,

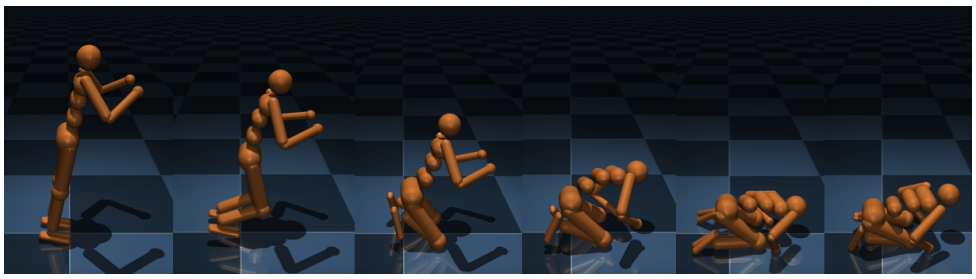

Figure 5.1: Trajectory of T_0 when using the forward fall controller on a forward fall.

the flexion of the ankle joint lifts the feet off the ground so the lower leg can rotate backwards and the upper leg forwards.

Even though after 750 ms the upper body is caught there is still some movement as the upper body moves back up again due to the stiffness present in the system. This is why the torques do not stay constant after the zero upper body velocity point. It can also be noted that the torques are far from zero after the body already has been caught, this is expected to be due to the relation between action signal and actuation cost not having been learned yet.

Qualitative evaluation

The qualitative evaluation of the forward fall can be seen in Table 5.1. The average results show the impacted limbs (E2) generally are the distal limbs and that the controller generally shows a smooth motion profile (E4), without too much unnecessary movements. The fall did look somewhat realistic (E5), however, in some cases the upper body topples to the side after being caught (T_{M+}, T_L and T_R). The forward fall controller does rely heavily on the upper body

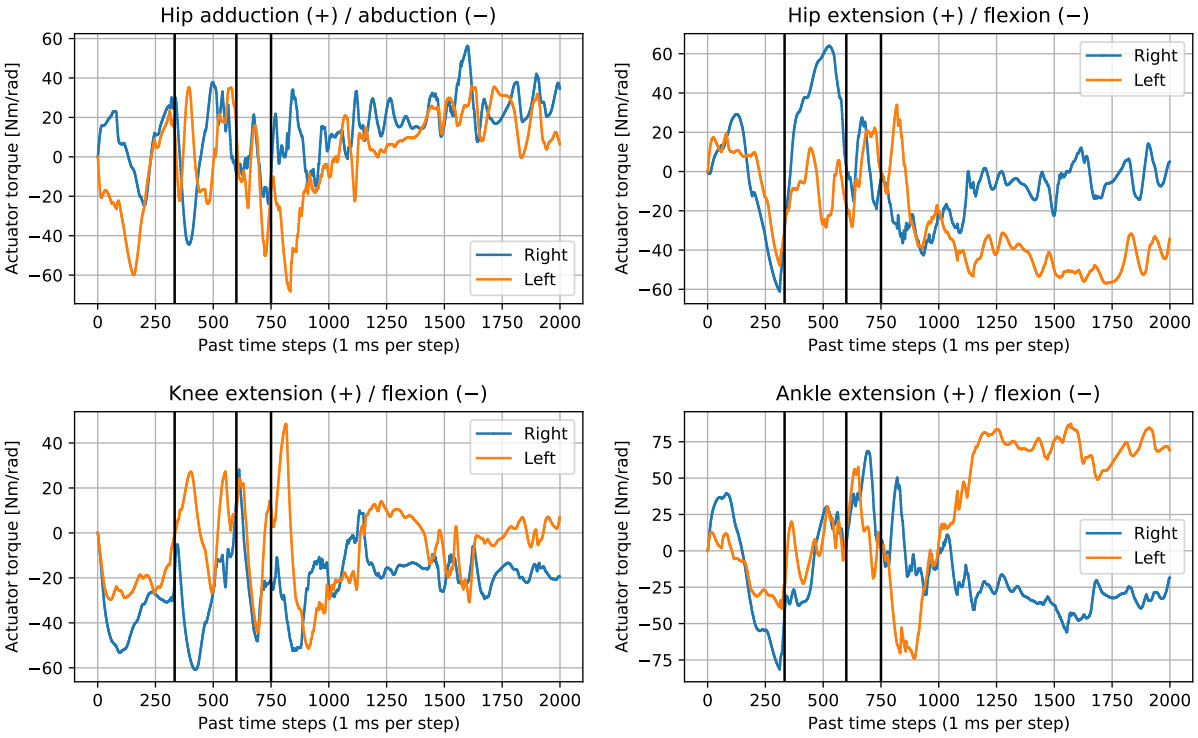

Figure 5.2: Actuator torques during T_0 for the forward fall following the policy of the forward fall controller. The vertical lines indicate knee impact (330 ms), hand impact (600 ms) and zero upper body velocity (750 ms)

Criterion	T_0	T_{M-}	T_{M+}	T_{F-}	T_{F+}	T_L	T_R	T_{AS}	T_{AB}	Avg
E1.	2	2	2	2	1	2	2	2	2	**1.9**
E2.	3	3	3	3	1	3	3	1	3	**2.6**
E3..	3	3	2	3	1	2	2	1	3	**2.2**
E4.	3	2	2	3	2	3	2	3	3	**2.6**
E5.	3	3	2	3	2	2	2	1	3	**2.3**
E6.	2	2	2	2	1	2	2	1	2	**1.8**

Table 5.1: Qualitative evaluation of the test scenarios of the forward fall as performed by the implemented forward fall controller.

to catch itself (E6). During T_{AS} the upper body was almost caught by solely the lower body. However, the controller was not able to maintain balance, resulting in the upper body falling backwards after being kept upright for some time.

The severity of the fall (E1) would be improved if the upper body rotation was slowed down more by the controller, this is too high before hand impact in all simulations. This further comes forth in the risk of head impact (E3) of T_{M+}, since the mass is higher during this simulation the upper body is barely able to stop itself before head impact. During T_L and T_R the risk of head impact is higher as the upper body is mostly caught by one arm, which results in the head being close to the ground before stopping the forward rotation. The severity of the fall could be further improved as the knee impact was done with the thighs perpendicular to the floor during T_{F+} and T_{AB}.

T_{F+} shows that the controller has difficulty when dealing with harder falls. This is because the upper body is completely above the knees during knee impact, instead of behind the knees which is the case during the other simulations. Because the upper body slows down by having to rotate over the knees, removing this from the safe fall strategy severely increases the forward rotation of the upper body as compared to the other simulations, resulting in the upper body not being able to catch itself before head impact.

Quantitative evaluation

The quantitative analysis has been divided into four categories, based upon the consequences of impact on the specific limbs. These categories are head (1), torso and waist (2), pelvis, thighs and upper arms (3) and shin, knees and lower arms (4). The pelvis was chosen as a category 3, even though impact here could be quite severe, because the pelvis is located within the exoskeleton. This means that if it is known that impact on the pelvis is likely, design considerations could be made to add padding to the exoskeleton, reducing the severity of landing on the pelvis.

In Figure 5.3 the impact forces and velocities of the T_0 scenario are shown. The bars are ordered as such that the left most bar is the most important, namely the head and the right most bar is the least important, namely the shins, knees and lower arms. The figure shows that during the uncontrolled fall the humanoid has a high impact on the head and an impact on the shins, knees and lower arms. During the controlled fall, the humanoid does not hit its head, but instead hits its pelvis, thighs and upper arms and has a harder impact on the knees. For the rest of the evaluation, the impact forces and velocities are grouped per category instead of per test in order to show the trends which come forward over all test cases.

The impacts grouped per category are shown in Figures 5.4 to 5.7. In each figure the maximum impact force or velocity is compared to an uncontrolled fall controller for each of the tests. The head impact (see Figure 5.4) shows that the generated controller reduces both the amount of times the head is hit as well as the impact force and impact velocity if the head is hit.

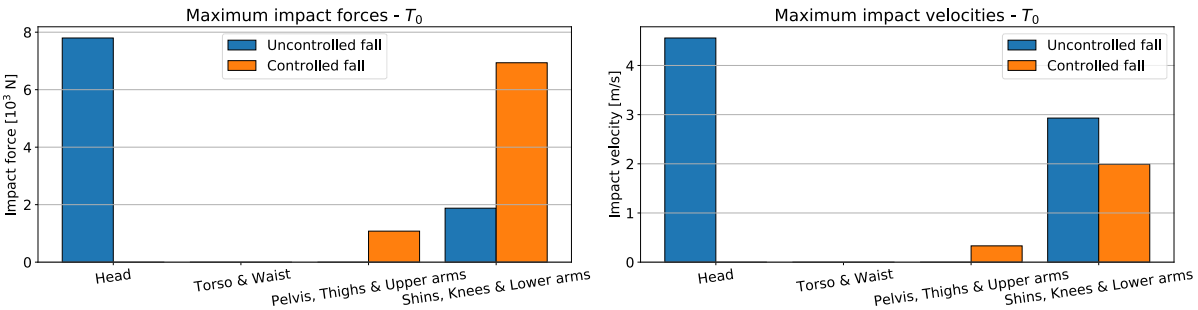

Figure 5.3: Comparison of maximum impact force (left) and velocity (right) during the T_0 scenario between the uncontrolled fall (blue) and controlled fall (orange) of the forward fall.

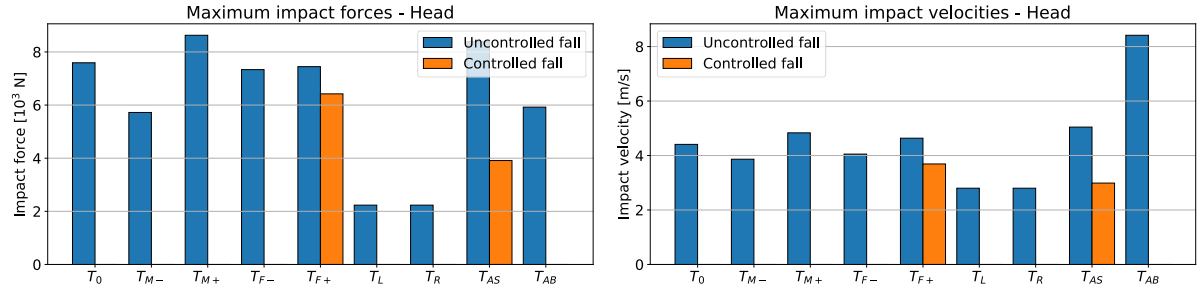

Figure 5.4: Comparison of maximum impact force (left) and velocity (right) of the head during forward fall between the uncontrolled fall (blue) and controlled fall (orange) for all test cases.

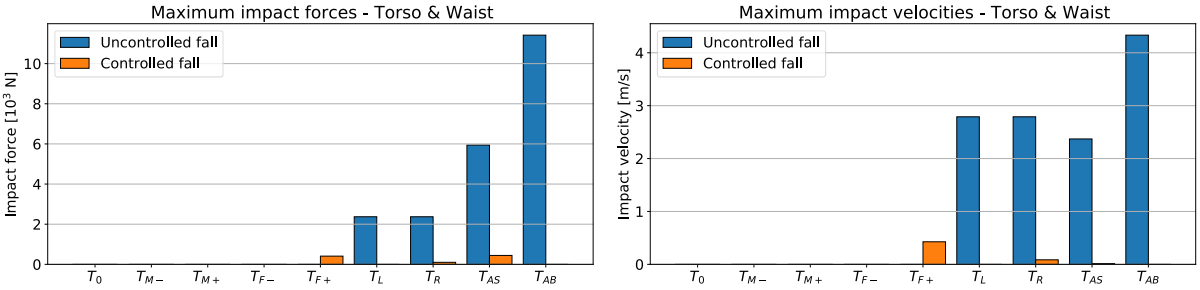

Figure 5.5: Comparison of maximum impact force (left) and velocity (right) of the torso and waist during forward fall between the uncontrolled fall (blue) and controlled fall (orange) for all test cases.

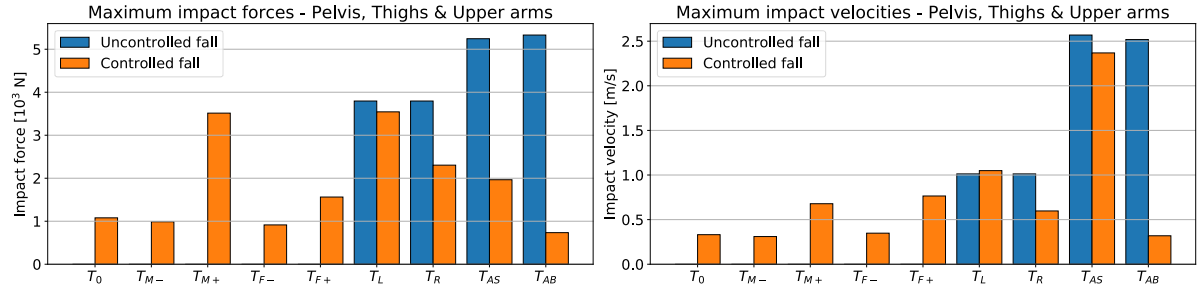

Figure 5.6: Comparison of maximum impact force (left) and velocity (right) of the pelvis, thighs and upper arms during forward fall between the uncontrolled fall (blue) and controlled fall (orange) for all test cases.

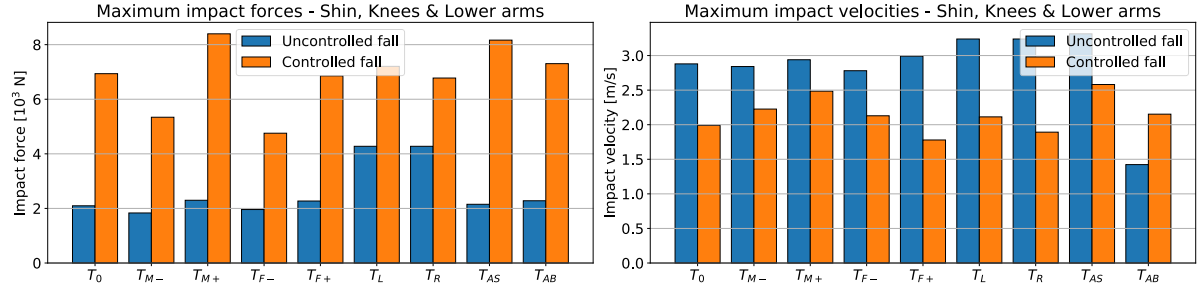

Figure 5.7: Comparison of maximum impact force (left) and velocity (right) of the shin, knees and lower arms during forward fall between the uncontrolled fall (blue) and controlled fall (orange) for all test cases.

The humanoid implementing the uncontrolled fall controller hits its head in all the test cases and the head impact velocities are approximately equal to the reported value of 4.8 m/s by Qiu et al. (2018), except for T_{AB} where the pelvis hit the ground first, which resulted in a whipping effect on the head, increasing its velocity. During the controlled case, the humanoid only hits its head in T_{F+} and T_{AS}. The T_{F+} case is because the upper body rotation has not been reduced enough and therefore the upper body cannot be fully caught by the arms. During T_{AS} the humanoid falls backwards after remaining upright for some time. As the arms are kept to the side, the humanoid is unable to catch itself. During T_{AB} the humanoid also falls backwards, however now the humanoid is able to catch itself as the arms are aimed backwards.

The impact on the torso and waist (see Figure 5.5) is almost completely negated. The impacts that are present are low enough to not result in injury or damage.

The impact on the pelvis, thighs and upper arms (see Figure 5.6) generally is due to the humanoid hitting itself during the fall. Either by hitting the thigh and/or pelvis with the feet or by landing on the thigh with an arm. During T_{M+}, T_L, T_R and T_{AS} the humanoid topples to the side after having caught itself, landing on the upper arm, resulting in the higher impact force and velocity in these cases.

Finally, in Figure 5.7, the control strategy of first falling on the knees clearly comes forward. This is recognised by the fact that the impact velocities are lower for the forward fall controller, however, the impact forces are tremendously higher. This difference is due to the humanoid using the generated controller landing on the knees first and trying to decelerate the body by this and the humanoid implementing the uncontrolled fall controller hitting the knees later due to falling forward as a plank and hitting the knees at approximately the same time as the rest of the body. The higher impact force, however, is to be expected as the controller is designed for a non-catchable fall situation and thus the lessened impact on the head, torso and waist has to be caught elsewhere.

5.2 Backward fall controller

After having learned for 1,714,400 time steps (or 8572 falls), the safe fall strategy implemented by the backward fall controller is to keep the upper body in front of the pelvis. This results in the upper body rotating forward after pelvis impact instead of backwards, which prevents the head from hitting the ground. The controller does this by rotating the upper body forwards and by stretching the legs, pushing the pelvis backwards (see Figure 5.8). A video of the backward fall controller performing all the test cases is included in the Appendix.

A more in-depth description of the policy is given using Figure 5.9. In this figure the actuator torques during the backward fall in the T_0 case are shown. In the figure two time steps are indicated, these relate to the pelvis impact and zero upper body velocity, respectively.

Starting at the hip flexion / extension, it is shown that the hips are flexing during the fall. This maintains slightly forward upper body rotation. Interestingly, it can be seen that after the impact there hardly is any balancing torque to maintain a forward rotated upper body.

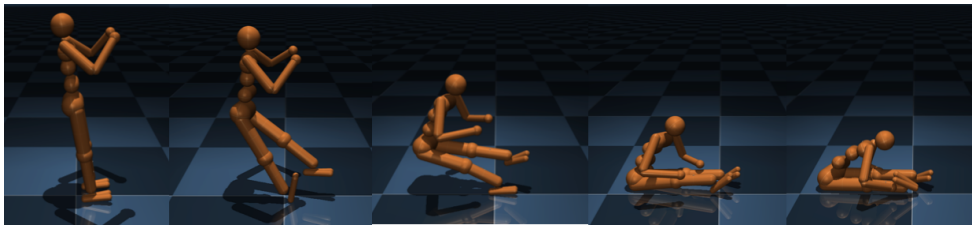

Figure 5.8: Trajectory of T_0 when using the backward fall controller on a backward fall.

The knees on the other hand are extending throughout the fall up until pelvis impact and after the upper body has stopped moving downward with a dip in torque in between. This extension ensures that the legs are always stretched, so that the pelvis is as far backwards as possible and the upper body does not hit the knees after ground impact.

The policy with regards to the hip abduction / adduction and the ankle flexion / extension barely lies in minimising ground impact. The applied torques result in the passive external / internal hip joint being rotated such that after the fall the left leg is rotated internally and the right leg is rotated externally. This results in the hip abduction / adduction joint to also apply force in the direction which normally would be the hip flexion / extension. This, unfortunately, is not realistic and is only true in the created simulation. The high hip abduction / adduction torque after having fallen is then used to push the upper body forward, preventing it from toppling. This also explains why the hip flexion / extension is barely used to balance the upper body.

Qualitative evaluation

The qualitative evaluation of the backward fall controller is shown in Table 5.2. Most of the evaluation criteria are evaluated to the (near) maximum score. The only limbs ever hitting the ground are the distal limbs (E2), the risk of head impact also is low in all the test cases (E3) and there is little to no jitter during and after the fall (E4).

During T_{F+} and T_{AS} the upper body is perpendicular to the ground at pelvis impact (E1). Furthermore, during T_{AS} the left hand impacts before the pelvis impact. This perpendicular, individual limb impact results in a minimum score. During T_{AB} the upper body falls backwards, which is why it is caught by the upper body (E6).

The fall itself does not look very natural (E5) as the legs are stretched throughout the whole fall, which is not what a trained person would do. A trained person would fall backwards using a squatting motion (Moon and Sosnoff, 2017). Additionally, during T_{F-}, T_{F+}, T_L, T_R and T_{AS} the humanoid also topples to the side after having fallen.

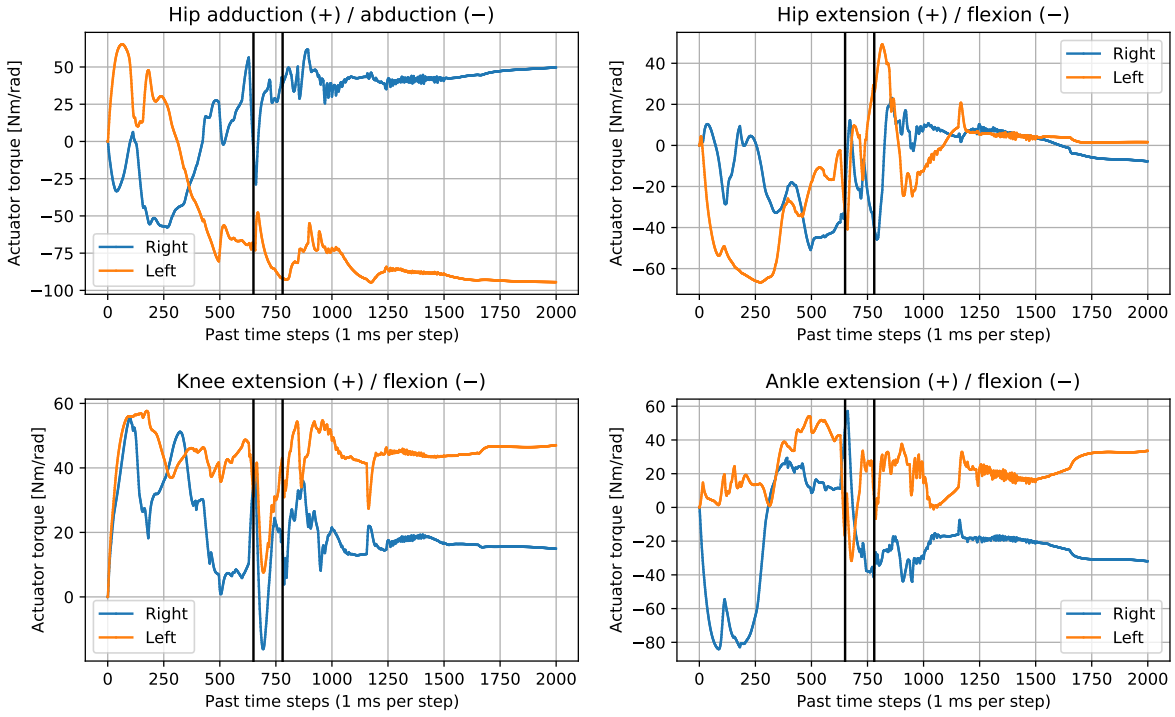

Figure 5.9: Actuator torques during T_0 during the backward fall following the policy of the backward fall controller. The vertical lines indicate pelvis impact (650 ms) and zero upper body velocity (780 ms).

Quantitative evaluation

In Figure 5.10 the impact forces and velocities during the T_0 case for the backward fall are shown. The impact forces on the head, torso and waist during the uncontrolled fall are negated with a slight increase in pelvis, thighs and upper arms impact force and velocity and an impact on shin, knees and lower arms. As mentioned in the forward fall evaluation, the increase in impact force and velocity in the distal limbs is to be expected, as the negated impact has to be covered by another part of the body (the total amount of converted kinetic energy is somewhat similar).

The impact forces and velocities over all the test cases show a similar pattern (see Figures 5.11 - 5.14). The head impact has been completely negated over all test cases (see Figure 5.11) and there is only torso and waist impact during T_L and T_{AS} (see Figure 5.12).

The impact force on the pelvis, shin and upper arms (see Figure 5.14) has only been majorly increased during the T_L, T_R and T_{AB} case and is lower during T_{AS}. The impact velocity is only increased during T_{AB}, whereas it is lower during T_L and T_R.

The impact force on the shin, knees and lower arms (see Figure 5.14) has two high peaks during the T_R and T_{AS}, where the high impact of T_{AS} is explained by the hand hitting the ground before the pelvis.

Criterion	T_0	T_{M-}	T_{M+}	T_{F-}	T_{F+}	T_L	T_R	T_{AS}	T_{AB}	Avg
E1.	3	3	3	3	2	3	3	1	3	**2.7**
E2.	3	3	3	3	3	3	3	3	3	**3.0**
E3.	3	3	3	3	3	3	3	3	3	**3.0**
E4.	3	3	3	3	3	3	3	3	3	**3.0**
E5.	2	2	2	2	2	2	2	2	2	**2.0**
E6.	3	3	3	3	3	3	3	3	2	**2.9**

Table 5.2: Qualitative evaluation of the backward fall as performed by the implemented controller.

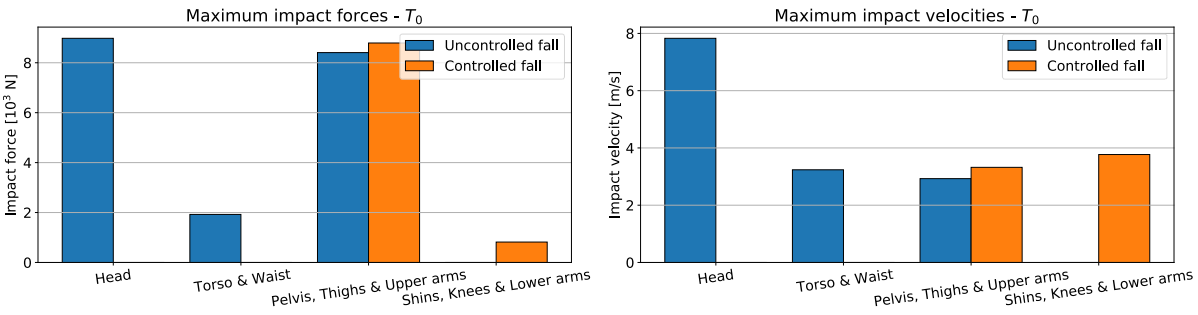

Figure 5.10: Comparison of maximum impact force (left) and velocity (right) during the T_0 scenario between the uncontrolled fall (blue) and controlled fall (orange) of the backward fall.

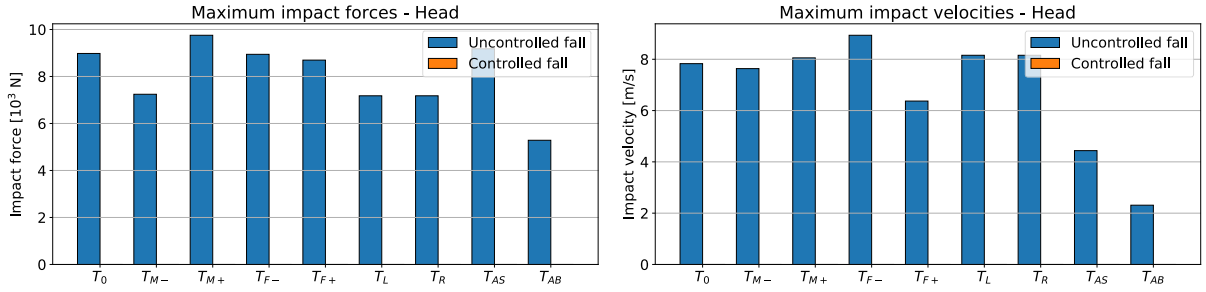

Figure 5.11: Comparison of maximum impact force (left) and velocity (right) of the head during backward fall between the uncontrolled fall (blue) and controlled fall (orange) for all test cases.

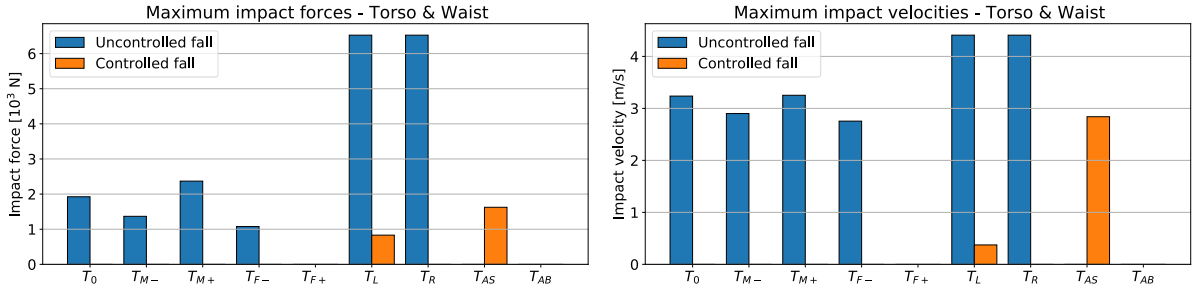

Figure 5.12: Comparison of maximum impact force (left) and velocity (right) of the torso and waist during backward fall between the uncontrolled fall (blue) and controlled fall (orange) for all test cases.

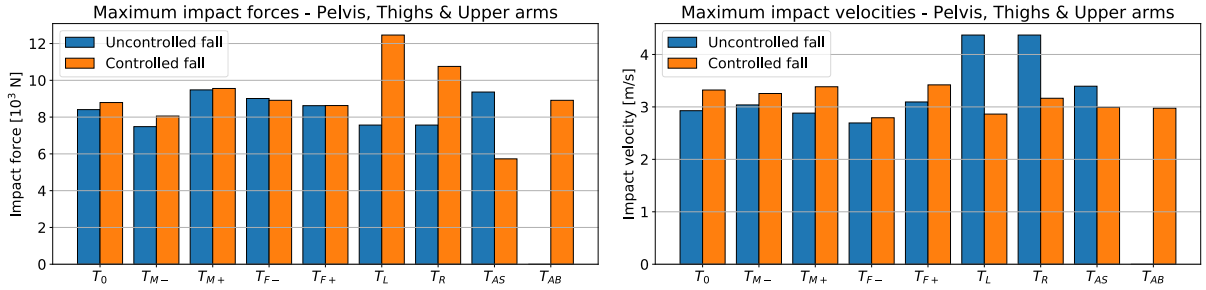

Figure 5.13: Comparison of maximum impact force (left) and velocity (right) of the pelvis, thighs and upper arms during backward fall between the uncontrolled fall (blue) and controlled fall (orange) for all test cases.

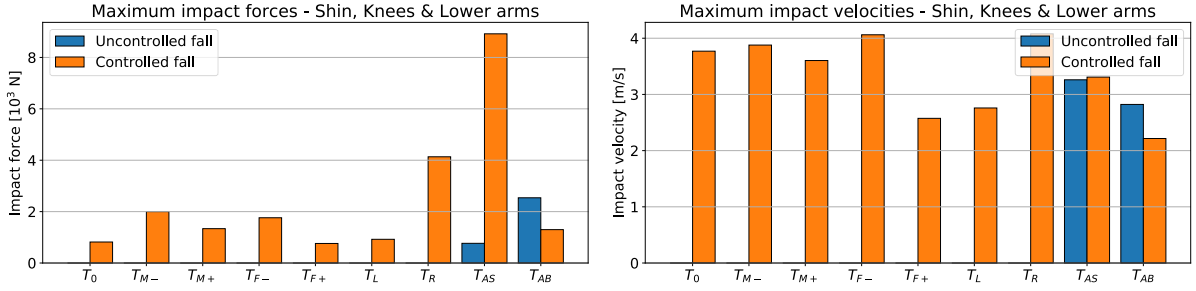

Figure 5.14: Comparison of maximum impact force (left) and velocity (right) of the shin, knees and lower arms during backward fall between the uncontrolled fall (blue) and controlled fall (orange) for all test cases.

5.3 Generalisation to other falls

Figures 5.15 and 5.16 show how the policies generalise to other falls. The plots indicate by color how high the maximum impact of the head is if the fall is started in the shown direction with the shown starting velocity and rotation. On the radial axis only the initial translational velocity is given, however, the initial rotational velocity and initial rotation are also increased accordingly. Starting with an initial rotation of 2.5° and initial rotational velocity of 0.5 rad/s at initial translational velocity of 0.25 m/s. Increased by 2.5°, 0.1 rad/s and 0.05 m/s for each step outward, respectively.

The forward fall controller (see Figure 5.15) has difficulty with generalising to harder falls as also comes forward in the evaluation of the T_{F+} case. It is also shown that it has a high dependency on the upper body to catch itself as it rarely prevents head impact when the arms are kept to the side. There is no fall direction and force for which the forward fall controller manages to prevent head impact in all three cases. It performs similar to the forward direction up to about 60° from the forward direction.

The backward fall controller (see Figure 5.16) hardly hits the head if the arms are facing forward. It manages to do so by pushing off hard from the ground jumping in the fall direction, this results in mostly horizontal velocity and less vertical velocity, which the upper body is able to catch. This, however, would be a bad strategy if the arms are unable to catch the upper body as is shown in the other two plots or if an obstacle would be in that direction. The backward fall controller generalises well to falls harder as it has trained for as there is no increase in head impact. The softer falls do have a slight increase in head impact in the arms forward case. The controller generalises well to falls up to about 45° from the backward direction.

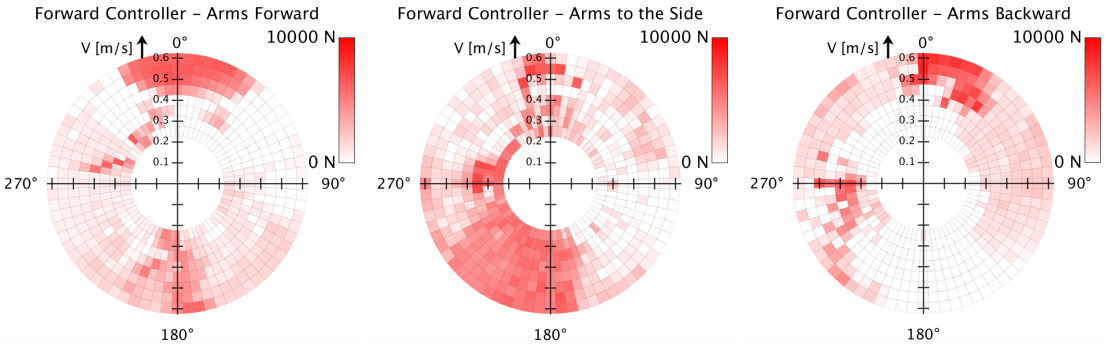

Figure 5.15: Polar plots of the maximum head impact force for falls in all directions and with harder and softer falls as was trained with, implementing the forward fall controller with the arms facing forward (left), with the arms to the side (middle) and with the arms facing backwards (right)

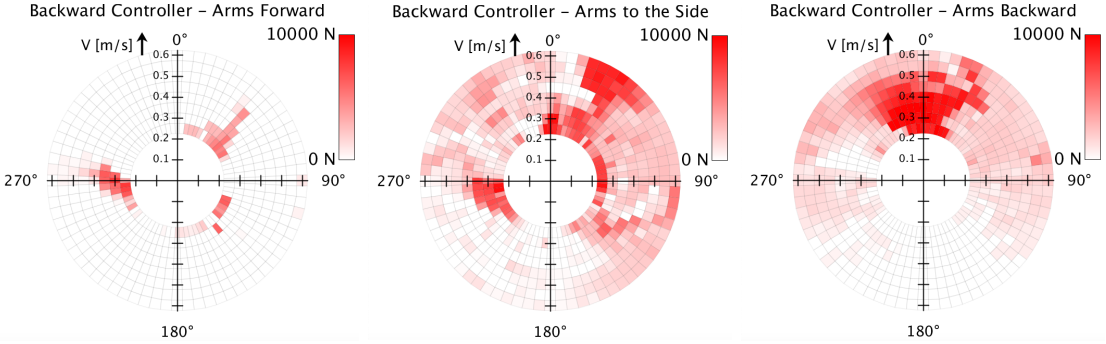

Figure 5.16: Polar plots of the maximum head impact force for falls in all directions and with harder and softer falls as was trained with, implementing the backward fall controller with the arms facing forward (left), with the arms to the side (middle) and with the arms facing backwards (right)

Chapter 6

Discussion

Using the presented method, two safe fall controllers are created. One is trained on the forward fall and one is trained on the backward fall. Both of these controllers are tested on a set of test cases for which is shown that they outperform the current state of the art controller, which locks the joints by adding virtual stiffness and damping to the joints. The forward fall controller lessens the impact on the head, torso and waist. The backward fall controller negates the impact on the head altogether during the test cases and lessens the torso and waist impact during most test cases. A test on the robustness to noise showed that the results of the forward fall controller remained similar to the earlier presented results when dealing with input noise (see Appendix A). Those test results, however, are based upon a single, representative instance of the noise. To better confirm that the controller is robust to input noise the test results over multiple runs should be given.

To achieve the results, the controllers had to learn a fitting safe fall strategy. The strategy performed by the forward fall controller consists of first hitting the ground with the knees, while trying to keep the upper body upright. After knee impact the upper body topples forward after which the upper body has to catch itself. The safe fall strategy performed by the backward fall controller is to keep the upper body slightly forward during the fall. This is done by stretching the legs as much as possible and thus pushing the pelvis backwards.

Lessening the impact on the head, torso and waist, does come at a cost however. For the forward fall this results in having a higher impact force on the shins, knees and lower arms and landing more often on the pelvis, thighs and upper arms. For the backward fall there is an increase in impact force for the pelvis, thighs and upper arms and the shins, knees and lower arms are landed on more often. As the lower limbs are inside the exoskeleton it might be possible to add padding to the exoskeleton at places where it is likely to land. The forward fall controller does also rely heavily on the upper body to catch itself. This also comes forward when looking into human fall strategies, but it would be better if the controller is able to catch itself fully with only lower limb movements. In one of the test cases it is able to fully stop the forward rotation of the upper body after landing on the knees. Afterwards, however, it is not able to keep the upper body upright and falls backwards. This might be solved by training the algorithm for longer as these situations in which the upper body remains upright might be unlikely cases. Not being able to keep the upper body upright could also be due to using the same stiffness and damping for all passive joints, where the abdomen typically should have a higher stiffness and/or damping to represent realistic values for co-contracted muscles.

During training and testing it was chosen to set the spring reference position to the initial position of the joint. As a result the upper body joints barely moved from their initial position. Consequently, the controllers are trained to be more independent of the upper body configuration as would be the case if the upper body limbs were to always keep the same spring reference position as the latter would result in the same upper body fall strategy for every fall. Not

being dependent on the upper body configuration during fall and during impact should majorly improve the resulting fall strategy as the resultant fall strategy relies less on the human pilot for a safe fall. Nevertheless, it is not trained and tested what would happen if the human pilot were to perform a consistent fall strategy as might happen if the pilot has undergone training for falling in an exoskeleton. This might result in a better controller specifically designed for that specific upper body fall strategy, however, it will most likely fail if the pilot is not trained or is not consistent in its fall strategy.

Continuing on the upper body, even though it might be beneficial for the controller to have information about the upper body configuration, this might not result in the best possible controller. This is because the sensor information of the upper body can be wholly influenced by the pilot. For instance, if the pilot were to perform an unlikely movement, it could trigger an unforeseen action by the controller, resulting in a hard impact. Besides this, it might be best for the controller to find a result, which works best regardless of the upper body.

The presented controllers are generated using reinforcement learning. Using reinforcement learning, however, also comes with some downsides, which also come forward in this work. First of all, the reinforcement learning method does not learn towards what it does not know. Therefore, learning more complex controllers such as first flexing the knees and then stretching the knees for the backward fall will probably not or hardly ever come forward in reinforcement learning, especially if the in-between steps are worse. Another downside is, it is hard to check whether the controller performs consistently for all falls within its range as some combinations of unlikely measurements might result in an unwanted action signal. Finally, the algorithm can exploit errors in the simulation environment, this comes forward in the balancing strategy for the backward fall in which the rotation of the hip joint allowed the hip abduction / adduction joint to also push the upper body forwards, which it normally is not able to do. This can also come forward in other optimal control methods, however due to the long training times of the reinforcement learning method and its dependence on a suitable random initialisation makes it harder to acquire another controller which performs adequately after a fix to the simulation is implemented.

The used Soft Actor-Critic reinforcement learning algorithm trains using experience replay. Due to this, the relation between transitions took too long to learn, therefore the action- and state-value did not recognise the upcoming fall and the Bellman equation did not offer the correct learning value to improve the value functions and the policy. The used solution was to update the reward after every episode to also include future reward in that transition. This resulted in a computationally efficient and accurate solution, as no extra iterations trough value functions had to be made (as is done in other solutions) and the used rewards were all presented by the environment and therefore accurate for that policy. This works for smaller replay buffers, however, for larger replay buffers the policy differs too much from the used policy, resulting in an incorrect value for certain states. Not being able to have a large replay buffer might hurt in training as edge cases might not come forward for long enough to also influence the policy. Additionally, by changing the reward to also feature future reward, the starting transitions are not represented in the reward as often as the other transitions. This means that if a too large time horizon is used, the agent will rather lower the reward if it can get it earlier instead of trying to increase the reward.

The reinforcement learning method used in this work, however, is not the only option of optimal control. Another option of optimal control would have been the Model Predictive Control method. When comparing reinforcement learning to this method some differences stand out. First, there are some non-continuous elements present in the used model, such as the ground impact. Furthermore, the cost on impact force is, as it is currently applied also discontinuous. There are solutions present for these discontinuities which have to be implemented first, before MPC is able to function. Another difference is that even though reinforcement learning has long

training times, it has short computation times during actual application. Model Predictive Control, however, has to perform more computations during actual application and might therefore be too slow to implement in a fall controller. Model Predictive Control on the other hand is not dependent on training to perform, this in theory makes it more robust to falls in all directions, whereas reinforcement learning might perform detrimental in falls outside of its training range.

In conclusion, the implemented controllers lessen the impact on the head for forward falls and negate it during backward falls when performing the test scenarios. The controllers keep the head impact velocity below 3.5 m/s in all but one test scenario. In about a quarter of the test scenarios it was able to prevent both the head, torso and waist from hitting the ground (R1). The applied safe fall strategy was performed with high consistency in both the forward and the backward fall (R2). The created environment the controllers are performing on is based upon the hardware specifications of the Symbitron^{+}, only being able to actuate the same joints with the same torque, applying similar motor dynamics and having the same update frequency (R3).

Despite the mostly positive requirements, there are still some difficulties to overcome before usage in an exoskeleton. First, the algorithm should be more independent of the upper body or some human pilot training should be done to ensure that the upper body movement is consistent. The controller can be made more independent of the upper body by removing the sensors on the torso and arms as these are influenced by the upper body movement. Furthermore, the stiffness and damping of the passive joints could be altered for each episode. This has as effect that the upper body is less reliable in catching itself, which should result in a controller which is less dependent on the upper body. In case of training the human pilot, the upper body movement should be changed to (mostly) perform the trained strategy and train a controller with these settings. Second, the current controllers have a high knee impact during the forward fall and a hard pelvis impact on the backward fall. Although this is an improvement from the current situation, these impacts are still hard enough to cause serious injury or damage if not handled properly. A solution for this would be to add padding to the exoskeleton or test on a soft floor when testing the fall controllers in a real-world environment. Third, the current implementation does not use any of the exoskeleton software, consequently the found controller still has to be implemented in the exoskeleton software. Additionally, the used sensors currently are not present in the exoskeleton, these have to be added to the exoskeleton for the system to function.

Finalising, this work improved on the current state of the art for exoskeleton safe fall controllers in multiple ways. It presented a controller which protects the head better than the state of the art controller. This is done by looking for a solution in three dimensions, whereas other research has only looked into two-dimensional solutions (Khalili et al., 2019). Furthermore, the controller is more broadly applicable as other research, which only tested for a single initial condition (Khalili et al., 2019). Additionally, this work implemented a safe fall controller using the reinforcement learning method, which has not been presented in research of safe fall controllers for exoskeletons before. In order to get the reinforcement learning method to work, a novel solution to the eligibility tracing problem has been created, which allowed for the backward propagation of the reward.

Chapter 7

Conclusion and Future work

In this work, two new controllers for minimising damage when falling in a lower limb exoskeleton have been created. For this, four methods for reducing fall damage were identified in literature. These methods were then used as the basis for training a reinforcement learning algorithm, which was chosen as the optimal control strategy. The implemented reinforcement learning algorithm was the Soft Actor-Critic algorithm, which tries to maximise reward as well as exploration, simultaneously. The algorithm trained on a humanoid model, in which it was only able to control eight actuators in the lower limbs, similar to the Symbitron^{+} exoskeleton. The earned reward consisted of an immediate reward based upon actuation cost as well as a discounted future reward based upon the four methods for reducing fall damage, which was used as an eligibility tracing method to propagate the low values (high impacts) backwards to the actions responsible.

To test the effectiveness of this method two controllers, one for the forward fall and one for the backward fall, were created. These controllers were evaluated both qualitatively and quantitatively. The qualitative evaluation of the forward fall controller returned that the controller mostly exposes the distal limbs to contact, while preventing impact on the head and body core. It also showed that the controller generally returns a smooth control strategy without too much jitter throughout the fall. The fall does still look quite severe, because the upper body rotation is not slowed down enough after knee impact. Furthermore, the controller depends heavily on the upper body to catch itself and prevent the core and head from impact with the ground. This means that if the controller would be implemented, the pilot is also responsible for a safe fall. The qualitative evaluation of the backward fall controller showed that the impact is on the distal limbs and with a low risk of head injury; the falls also do not look severe and there is little dependence on the upper body to catch itself. The falls do, however, not look like performed by a trained person as the legs are stretched throughout the fall instead of first performing a squatting motion. The quantitative evaluation showed that both the presented controllers have a far lower head impact, if any, as compared to the state of the art controller. Additionally, the impact on the torso and waist is lower. This is achieved by distributing the impact force to the shin, knees and lower arms (but with similar impact velocity). The impact on the pelvis, thighs and upper arms are impacted more often as compared to the original controller. It was furthermore tested how the controllers generalise to other falls by performing falls in all directions and with harder and softer falls as was trained with. This showed that the forward fall controller does generalise to softer falls, but does not to harder falls and it performs similar to the forward case for falls ranging up to 60° from the forward direction. The backward fall controller does generalise to both harder and softer falls and performs similar to the backward case for falls ranging up to 45° from the backward direction. The proposed requirements for a safer fall controller have been met. The controllers do show room for future improvement as the impact on the head, torso and waist are not yet completely negated.

A follow-up research should focus on testing the proposed method for more types of falls, such as sideways or during walking. Training without a stiff upper body and/or without upper body sensor information could result in a controller which is less dependent on the upper body for a safe fall. The test with noise input could be elaborated upon by checking the realistic values of noise in the Symbitron^{+} exoskeleton, by testing over more instances of noise and by also adding noise to the output of the controller. A further increase to the realism of the simulation can be achieved by adding a slight communication delay. If these changes are implemented and still result in a proper controller a test in a real-world setting should be performed to verify the simulated results.

Bibliography

Campolettano, E. T., Gellner, R. A., and Rowson, S. (2018). Relationship between impact velocity and resulting head accelerations during head impacts in youth football. In *Proceedings. International IRCOBI Conference on the Biomechanics of Impacts*, volume 2018, page 326. NIH Public Access.

Choi, W., Wakeling, J., and Robinovitch, S. (2015). Kinematic analysis of video-captured falls experienced by older adults in long-term care. *Journal of biomechanics*, 48(6):911–920.

Daley, B. and Amato, C. (2019). Reconciling λ-returns with experience replay. In *Advances in Neural Information Processing Systems*, pages 1133–1142.

Esquenazi, A., Talaty, M., Packel, A., and Saulino, M. (2012). The rewalk powered exoskeleton to restore ambulatory function to individuals with thoracic-level motor-complete spinal cord injury. *American journal of physical medicine & rehabilitation*, 91(11):911–921.

Fujiwara, K., Kajita, S., Harada, K., Kaneko, K., Morisawa, M., Kanehiro, F., Nakaoka, S., and Hirukawa, H. (2006). Towards an optimal falling motion for a humanoid robot. In *2006 6th IEEE-RAS International Conference on Humanoid Robots*, pages 524–529. IEEE.

Fujiwara, K., Kanehiro, F., Kajita, S., and Hirukawa, H. (2004). Safe knee landing of a human-size humanoid robot while falling forward. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 1, pages 503–508. IEEE.

Fujiwara, K., Kanehiro, F., Kajita, S., Yokoi, K., Saito, H., Harada, K., Kaneko, K., and Hirukawa, H. (2003). The first human-size humanoid that can fall over safely and stand-up again. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, volume 2, pages 1920–1926. IEEE.

Gurriet, T., Finet, S., Boeris, G., Duburcq, A., Hereid, A., Harib, O., Masselin, M., Grizzle, J., and Ames, A. D. (2018). Towards restoring locomotion for paraplegics: Realizing dynamically stable walking on exoskeletons. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2804–2811. IEEE.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. (2018). Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*.

He, Y., Eguren, D., Luu, T. P., and Contreras-Vidal, J. L. (2017). Risk management and regulations for lower limb medical exoskeletons: a review. *Medical devices (Auckland, NZ)*, 10:89.

Horak, F. B. and Nashner, L. M. (1986). Central programming of postural movements: adaptation to altered support-surface configurations. *Journal of neurophysiology*, 55(6):1369–1381.

Hui, J. (2018). Reinforcement learning terms. https://medium.com/@jonathan_hui/rl-reinforcement-learning-terms-242baac11907. [Online; accessed August 9, 2020].

Kandilakis, C. and Sasso-Lance, E. (2019). Exoskeletons for personal use after spinal cord injury. *Archives of Physical Medicine and Rehabilitation*.

Khalili, M. (2016). *Developing control strategies to mitigate injury after falling backward with a lower limb exoskeleton*. PhD thesis, University of British Columbia.

Khalili, M., Borisoff, J. F., and Van der Loos, H. M. (2017). Developing safe fall strategies for lower limb exoskeletons. In *2017 International Conference on Rehabilitation Robotics (ICORR)*, pages 314–319. IEEE.

Khalili, M., Van der Loos, H. M., and Borisoff, J. F. (2019). Studies on practical applications of safe-fall control strategies for lower limb exoskeletons. In *2019 IEEE 16th International Conference on Rehabilitation Robotics (ICORR)*, pages 536–541. IEEE.

Kumar, V. V. (2019). Soft actor-critic demystified. https://towardsdatascience.com/soft-actor-critic-demystified-b8427df61665. [Online; accessed June 24, 2020].

Lee, S.-H. and Goswami, A. (2011). Fall on backpack: Damage minimizing humanoid fall on targeted body segment using momentum control. In *ASME 2011 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 703–712. American Society of Mechanical Engineers Digital Collection.

Lee, Y. and Ashton-Miller, J. A. (2011). The effects of gender, level of co-contraction, and initial angle on elbow extensor muscle stiffness and damping under a step increase in elbow flexion moment. *Annals of biomedical engineering*, 39(10):2542.

Li, T., Zhao, D., and Yi, J. (2008). Heuristic dynamic programming strategy with eligibility traces. In *2008 American Control Conference*, pages 4535–4540. IEEE.

Ma, G., Huang, Q., Yu, Z., Chen, X., Hashimoto, K., Takanishi, A., and Liu, Y.-H. (2014). Bio-inspired falling motion control for a biped humanoid robot. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 850–855. IEEE.

Meijneke, C., Wang, S., Sluiter, V., and van der Kooij, H. (2017). Introducing a modular, personalized exoskeleton for ankle and knee support of individuals with a spinal cord injury. In *Wearable robotics: challenges and trends*, pages 169–173. Springer.

Moon, Y. and Sosnoff, J. J. (2017). Safe landing strategies during a fall: systematic review and meta-analysis. *Archives of physical medicine and rehabilitation*, 98(4):783–794.

Pachocki, J., Chociej, M., Petron, A., Zaremba, W., Powell, G., Welinder, P., Tobin, J., McGrew, B., Weng, L., Baker, B., Józefowicz, R., Sidor, S., Schneider, J., Ray, A., Plappert, M., and Andrychowicz, M. (2018). Learning dexterity. https://openai.com/blog/learning-dexterity/. [Online; accessed July 11, 2020].

Qiu, J., Chen, Y., Cheng, H., and Hou, L. (2018). Impact analysis on human body of falling events in human-exoskeleton system. In *Congress of the International Ergonomics Association*, pages 767–776. Springer.

Rampeltshammer, W. F., Keemink, A., and Van Der Kooij, H. (2020). An improved force controller with low and passive apparent impedance for series elastic actuators. *IEEE/ASME Transactions on Mechatronics*.

Robinovitch, S. N., Chiu, J., Sandler, R., and Liu, Q. (2000). Impact severity in self-initiated sits and falls associates with center-of-gravity excursion during descent. *Journal of biomechanics*, 33(7):863–870.

Roboti LLC (2018). Mujoco, advanced physics simulation. http://www.mujoco.org/. [Online; accessed July 6, 2020].

Samy, V., Bouyarmane, K., and Kheddar, A. (2017). Qp-based adaptive-gains compliance control in humanoid falls. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4762–4767. IEEE.

Samy, V. and Kheddar, A. (2015). Falls control using posture reshaping and active compliance. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 908–913. IEEE.

Subburaman, R., Kanoulas, D., Muratore, L., Tsagarakis, N. G., and Lee, J. (2019). Human inspired fall prediction method for humanoid robots. *Robotics and Autonomous Systems*, 121:103257.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Tefertiller, C., Hays, K., Jones, J., Jayaraman, A., Hartigan, C., Bushnik, T., and Forrest, G. F. (2018). Initial outcomes from a multicenter study utilizing the indego powered exoskeleton in spinal cord injury. *Topics in spinal cord injury rehabilitation*, 24(1):78–85.

Wang, S. and Hauser, K. (2017). Real-time stabilization of a falling humanoid robot using hand contact: An optimal control approach. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 454–460. IEEE.

Yun, S.-k. and Goswami, A. (2014). Tripod fall: Concept and experiments of a novel approach to humanoid robot fall damage reduction. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2799–2805. IEEE.

Yun, S.-k., Goswami, A., and Sakagami, Y. (2009). Safe fall: Humanoid robot fall direction change through intelligent stepping and inertia shaping. In *2009 IEEE International Conference on Robotics and Automation*, pages 781–787. IEEE.

Zychlinski, S. (2019). The complete reinforcement learning dictionary. https://towardsdatascience.com/the-complete-reinforcement-learning-dictionary-e16230b7d24e. [Online; accessed August 9, 2020].

Appendix

A Robustness to noise

To test the robustness of the found controllers to noise, a controller input with noise is created. For this, first the maximum, absolute values of all inputs are measured over 100,000 time steps using actions uniformly drawn from the action space. If the maximum value is known, such as maximum mass or actuator torque, the known maximum values are recorded. This maximum value is then used during simulation as all signals are drawn from a normal distribution as given in Equation 1.

$$\bar{u_i} \sim N(u_i \ , \ 0.02 \cdot \max(|u_i|)) \tag{1}$$

where u_i is input i. This equation shows that a Gaussian noise is added on top of the deterministic input. This Gaussian noise has a standard deviation of 2% of the maximum value of the input.

Testing the robustness to noise of the controllers is done by running the test cases with the forward fall controller and comparing the results of one instance of the noise with the results of the controlled noise-free fall (see Figures 1-4). Note that the figures feature only one instance of the noise, due to the stochastic nature of the noise other instances will give slightly other results. This instance was chosen, because it was the first outcome.

Figure 1 shows that the controller with noise also does prevent head impact during most test cases. In this instance of noise the head impact during T_{AS} has successfully been prevented, which was not the case during the noise-free controlled fall, however, the prevented head impact of T_L during noise-free testing could not be prevented within this instance of noise.

The torso and waist impact remain relatively low during both simulations as compared to impact forces and velocities on other parts of the humanoid (see Figure 2). For the impacts on the pelvis, thighs and upper arms the controlled fall with noise has a couple of harder and faster impacts as compared to the noise-free controlled fall and only significantly performed better in the T_{M-} test case (see Figure 3). When comparing the results of the impact forces on the shins, knees and lower arms hardly any difference comes forward (see Figure 4).

In general the results of the controlled fall with noise are similar to the results of the noise-free fall during the test cases.

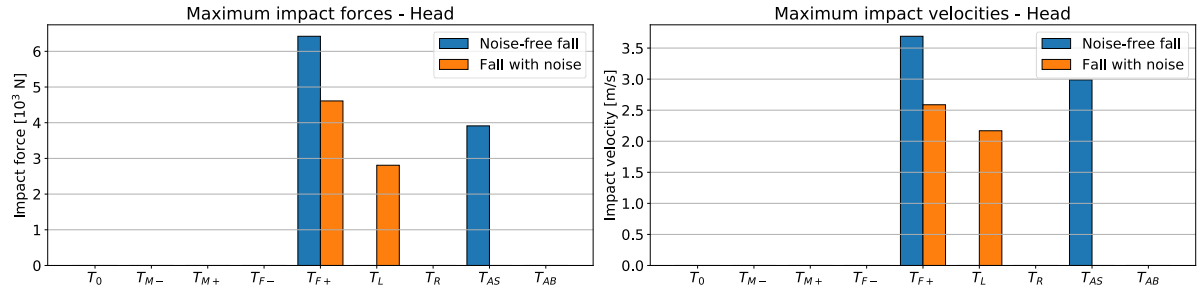

Figure 1: Comparison of maximum impact force (left) and velocity (right) of the head during forward fall between the controlled noise-free fall (blue) and controlled fall with noise (orange) for all test cases.

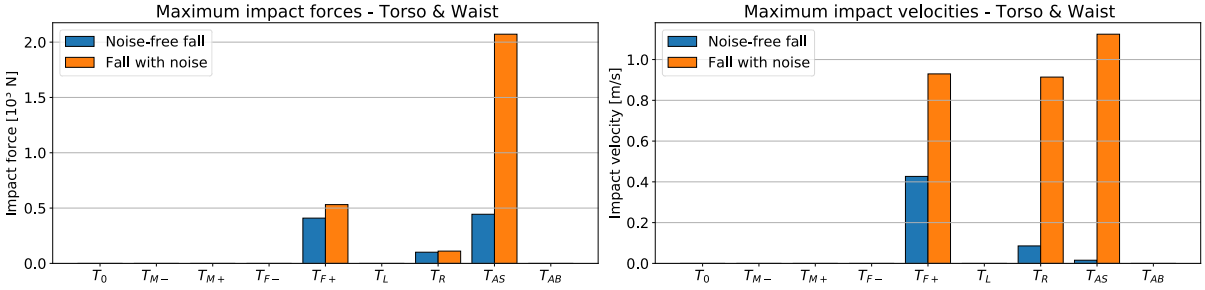

Figure 2: Comparison of maximum impact force (left) and velocity (right) of the torso and waist during forward fall between the controlled noise-free fall (blue) and controlled fall with noise (orange) for all test cases.

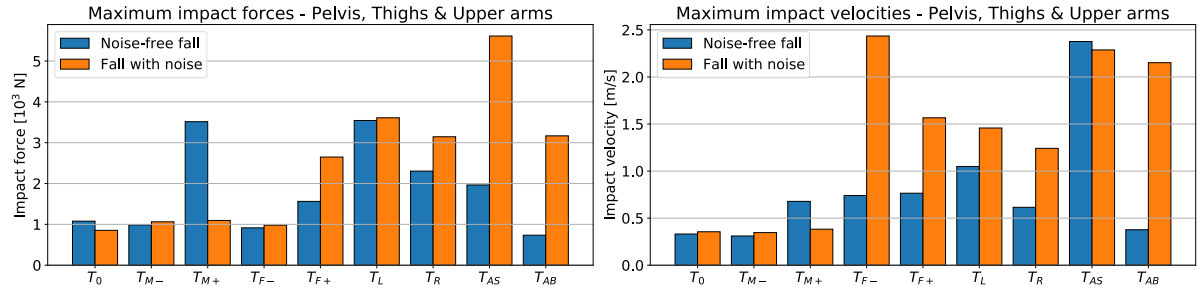

Figure 3: Comparison of maximum impact force (left) and velocity (right) of the pelvis, thighs and upper arms during forward fall between the controlled noise-free fall (blue) and controlled fall with noise (orange) for all test cases.

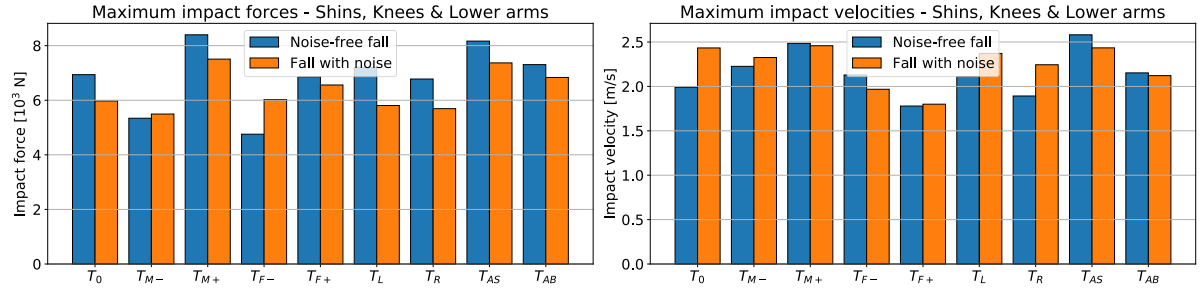

Figure 4: Comparison of maximum impact force (left) and velocity (right) of the shin, knees and lower arms during forward fall between the controlled noise-free fall (blue) and controlled fall with noise (orange) for all test cases.

B List of reinforcement learning terms

Based on Hui (2018), Zychlinski (2019) and `https://deepai.org/definitions`

Action Actions are the agent's methods which allow it to interact and change its environment, and thus transfer between states.

Action-value (Q-value) Usually denoted as $Q(s, a)$, the Q-value is a measure of the overall expected reward assuming the agent is in state s and performs action a, and then continues playing until the end of the episode following policy π.

Agent The learning and acting part of a Reinforcement Learning problem, which tries to maximize the rewards it is given by the environment.

Bellman equation The Bellman equation defines the relationships between a given state (or state-action pair) to its successors. The most common one usually encountered in reinforcement learning tasks is the Bellman equation for the optimal Q-Value, which is given by:

$$Q^*(s, a) = \sum_{s',r} p\left(s', r \mid s, a\right) \left[r + \gamma \max_{a'} Q^*\left(s', a'\right)\right]$$

where the asterisk sign indicates optimal value.

Discount factor (γ) The discount factor, usually denoted as γ, is a factor multiplying the future expected reward, and varies on the range of [0,1]. It controls the importance of the future rewards versus the immediate ones. It is found in the Bellman equation.

Entropy Entropy is a measure of the randomness in the information being processed. The higher the entropy, the harder it is to draw any conclusions from that information and the more random it is.

Environment Everything which isn't the agent; everything the agent can interact with, either directly or indirectly. The environment changes as the agent performs actions; every such change is considered a state-transition. Every action the agent performs yields a reward received by the agent.

Episode All states that come in between an initial-state and a terminal-state. In situations where there is no terminal-state, we consider an infinite episode. It is important to remember that different episodes are completely independent of one another.

Expected return Sometimes referred to as "overall reward" and occasionally denoted as G, is the expected reward over an entire episode.

Experience replay As reinforcement learning tasks have no pre-generated training sets which they can learn from, the agent must keep records of all the state-transitions it encountered so it can learn from them later. The memory-buffer used to store this is often referred to as experience replay or replay buffer.

Exploitation & Exploration Reinforcement learning tasks have no pre-generated training sets which they can learn from — they create their own experience and learn "on the fly". To be able to do so, the agent needs to try many different actions in many different states in order to try and learn all available possibilities and find the path which will maximize its overall reward; this is known as exploration, as the agent explores the environment. On the other hand, if all the agent will do is explore, it will never maximize the overall reward — it must also use the information it learned to do so. This is known as exploitation, as the agent exploits its knowledge to maximize the rewards it receives. The trade-off between the two must be balanced in order to allow the agent to both explore the environment enough, but also exploit what it learned and repeat the most rewarding path it found.

Initialisation At the start of training the weights of the neural networks have to be inserted. These weights are drawn from a distribution, which is typically dependent on the size of the

previous layer. For linear layers `PyTorch` typically uses:

$$w_{(i,j)} \sim U\left(-\frac{1}{\sqrt{n_u}}, \frac{1}{\sqrt{n_u}}\right)$$

An important aspect of initialisation for neural networks is to perform symmetry-breaking, otherwise all the neurons perform the same calculation and which would be no better as a neural network with a single node.

Loss function Loss functions are used to determine the error (the loss) between the output of the algorithms and the given target value. The loss function is used together with stochastic gradient descent to minimise the loss.

Neural network Neural networks are function approximators, which are particularly useful in reinforcement learning when the state space or action space are too large to be completely known. Neural networks consist of layers of nodes, with the first one being the input layer, the final one being the output layer and the in-between layers being the hidden layers. There exist weights connecting all the nodes to all the nodes of the following layer. The linear transformation of these weights and the values in the previous layers passes through a non linear activation function to produce the values of the next layer. This process happens layer to layer during forward propagation and by back propagation, the weights can be updated towards their optimum value using, for instance, stochastic gradient descent.

Policy (π) The policy, denoted as π (or $\pi(a|s)$), is a mapping from some state s to the probabilities of selecting each possible action given that state.

Polyak averaging Polyak averaging is a method to update a target network. The weights of the main neural network is slowly copied to the target value using:

$$\varphi_{tar} = (1.0 - \tau)\varphi_{tar} + \tau\varphi$$

In which φ denotes the weights of the main (φ) and target (φ_{tar}) network and τ is a measure for how quickly the weights are copied.

Replay buffer See Experience Replay.

Reward A numerical value received by the agent from the environment as a response to the agent's actions. The agent's goal is to maximize the overall reward it receives during an episode, and so rewards are the motivation the agent needs in order to act in a desired behavior.

State / Observation Every scenario the agent encounters in the environment is formally called a state. The agent transitions between different states by performing actions. State and observation are generally used interchangeably in reinforcement learning.

State-value function Usually denoted as V(s), the value function is a measure of the overall expected reward assuming the agent is in state s and then continues playing until the end of the episode following policy π.

Stochastic gradient descent Stochastic gradient descent is a method to find the optimal parameters or weights. It iteratively makes small adjustments to the neural networks to decrease the error of the network. This method decreases the error by approximating the gradient of the loss with respect to the neural network weights using a randomly selected batch from the replay buffer.

Target network A neural network that is a stable approximation of the main neural network. Enabling to train the main network on the values predicted by the target network. Which prevents the feedback loop that occurs when the main network trains on values predicted by itself. By avoiding this feedback, training stability increases.

C Training parameters

SAC hyperparameters

Hyperparameter	Value
Number of hidden layers	2
Number of nodes per hidden layer	512
Policy learning rate	1×10^{-4}
Action-value learning rate	1×10^{-4}
State-value learning rate	1×10^{-4}
α learning rate	3×10^{-4}
Minimum required entropy (forward fall)	0.0
Minimum required entropy (backward fall)	-0.5
Replay buffer size	$20,000$
Batch size	128
Bellman equation discount factor (γ)	0.99
Reward modification discount factor (λ)	0.98
Reward modification time horizon	20
Time steps per episode	200
Observation frames	$1,000$
Body hit weight (c_1)	3×10^{-6}
Vertical kinetic energy weight (c_2)	2
Head risk weight (c_3)	0.2
Actuation cost weight (c_4)	5×10^{-3}

Table 1: Hyperparameters used during training

Body impact & vertical kinetic energy weights

Body	Body impact value, $w_{F,b}$	Kinetic energy value, $w_{E,b}$
Pelvis	2	10
Thigh	10	5
Knee	2	3
Shin	5	3*
Foot	0.1	0.2
Lower waist	20	15
Torso	20	25
Head	100	25*
Upper waist	20	25*
Upper arm	10	5
Lower arm	5	3
Hand	2	3*

Table 2: Reward weights used during training. *This body is rigidly connected to the above body and therefore has the same kinetic energy value.