university of groningen / faculty of science and engineering

Autoencoder-based Deep Reinforcement Learning for ground-level walking of Human Musculoskeletal models

Bachelor's Project Thesis

Massimiliano Falzari, s3459101, m.falzari@student.rug.nl,
Supervisors: Prof. Dr. R. (Raffaella) Carloni

Abstract: This paper proposes using autoencoder-based deep reinforcement learning (AE-DRL) architectures for ground-level walking of human musculoskeletal models. It compares undercomplete autoencoder (AE) and variational autoencoder (VAE) in the context of physics-based simulations. The used deep reinforcement learning (DRL) is Proximal Policy optimization with Imitation Learning (PPO+IL). The architectures are trained with a two-phase approach. First, the autoencoder-based latent space is learned using gathered simulated data. Then, the DRL agent with the pre-trained encoder is trained to learn a walking policy. The results show that AE-DRL methods are more efficient in learning with the same observation space than the standard DRL. Compared to the baseline (i.e. PPO+IL), AE-PPO+IL had a 131% longer mean duration of an episode and a 23% higher mean cumulative reward. VAE-PPO+IL, on the other hand, had a 102% longer mean duration of an episode and a 9% higher mean cumulative reward. Generally, AE showed better results than VAE with respect to reconstruction error (measured by the mean square error(MSE)) and DRL mean cumulative reward. VAE, in contrast, performed better in terms of root MSE from the imitation data.

1 Introduction

In recent years Deep Reinforcement Learning (DRL) has provided a reliable and effective approach for learning complex policies in continuous state and action space. Particularly, trust region policy gradient methods (Peters & Schaal (2008) Schulman et al. (2015) Schulman et al. (2017)) showed promising results in a large variety of robotics tasks (Melo et al. (2021) Melo & Máximo (2019) Teixeira et al. (2020)). Nevertheless, these methods, like many others (e.g. policy iteration methods), struggle to learn efficiently in high-dimensional state-space. This phenomenon happens for a variety of reasons. One of which is that, in order to generalize across states effectively, the agent needs to learn an in-between representation (e.g. through feature extraction) (Higgins et al. (2017)). This process can be quite data and time intensive, resulting in low sample efficiency and huge training time. It is especially problematic when dealing with raw pixels or multiple input sensors.

A common approach to improve sample efficiency is to use model-based DRL methods. These methods learn a model of the environment (i.e. the environment's dynamics) while acting, which can be used to sample simulated data (Luo et al. (2022)). However, they are significantly more computational intensive then model-free methods. Their performance is highly dependent on the goodness of the learned environment model. Moreover, they still struggle in high-dimensional state-space. For these reasons, this paper focus on autoencoder-based DRL. The idea is to use an autoencoder (AE) to learn a low-dimensional representation of the environment, which the DRL agent will then use (Abbasi et al. (2021) Higgins et al. (2017) Lončarević et al. (2021) Q. Wang (2022) Prakash et al. (2019) Andersen et al. (2018) Igl et al. (2018)).

Furthermore, this study test the effectiveness of the aforementioned technique on an open-source physics-based simulation (OpenSim) of a healthy musculoskeletal model, in which the goal is to per-

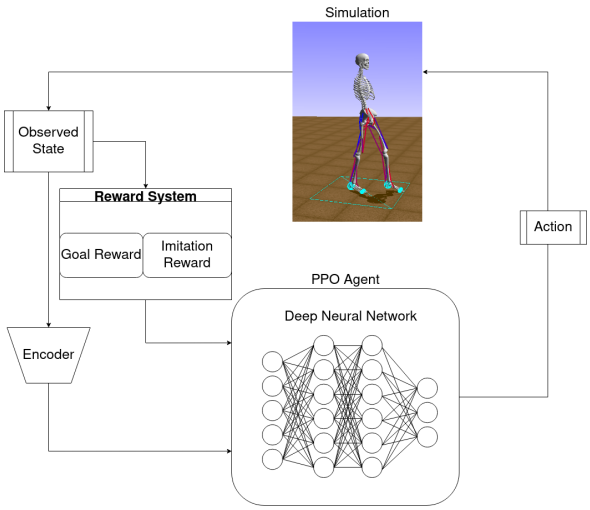


Figure 1.1: The proposed architecture. The particularity, as opposed to a standard DRL architecture, are two. First, the reward system has not only the usual goal reward but also the imitation reward. Second, the DRL agent does not get the observation directly. Instead, a pre-trained encoder compresses them before giving them to the agent.

form groud-level walking. The model consists of 22 muscles that control 14 degrees of freedom. The DRL algorithm of choice is PPO with Imitation Learning (PPO+IL). This algorithm showed encouraging results with a different model (De Vree & Carloni (2021) Surana (2021) Adriaenssens (2021)). The aim of the study is, therefore, to compare PPO+IL to the proposed autoencoder-based architecture (AE-PPO+IL). Figure 1.1 shows the proposed architecture. The main novelty with respect to standard PPO+IL is the pre-trained encoder. It will compress the high-dimensional observation into a low-dimensional representation.

To summarise, the main objectives of this paper are:

- Show the effectiveness of autoencoder-based architecture on non-pixel-based state-space.
- Compare the performances of AE-PPO+IL and VAE-PPO+IL to PPO+IL

The rest of the paper is organized as follows. Section 2 presents the necessary background on PPO and AE architectures and states the motivations for using such architectures. Section 3 dives into the methodology used in the study. It exposes all the technical details and implementation decisions with their rationale. Section 4 describes the results of the study. Lastly, Section 5 shows the limitation and future prospectives of this work.

2 Theoretical Background

This section dive into the mathematical details of the PPO algorithm, the autoencoder architectures used, and the motivation for using such architectures as opposed to classical statistical approaches (e.g. Principal Component Analysis)

2.1 Proximal Policy Optimization

The PPO algorithm was first introduced by Schulman et al. (2017). The primary motivation for introducing such an algorithm was to improve the Trust Region Policy Optimization (TRPO) method. TRPO was too complicated and incompatible with standard architectural optimization (e.g. parameter sharing between policy and value function).

Generally, trust region/natural policy gradient methods take their names because they constrain the policy update to be somewhat near the old policy (hence the name trust region). Doing so removes potentially destructive updates, making the learning process safer and more stable. In TRPO, the update is constrained based on the Kullback–Leibler (KL) divergence between the new and old policy. However, this makes the learning process a constrain-satisfaction problem which is notoriously computational expensive.

The method used to constrain the policy update is the main difference between TRPO and PPO. The clipped surrogate objective is, indeed, the core contribution made by Schulman et al. (2017). This new objective partially removes the need to constrain the update using the KL-divergence. Resulting in a more efficient and simpler algorithm.

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

$r_t(\theta)$ is the ratio between the old and new policy at time t. $\hat{A}_t$ is the advantage estimation (i.e. the difference between the expected and real reward at

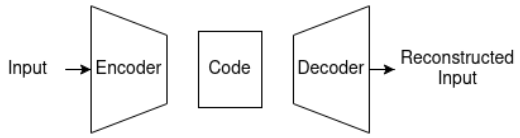


Figure 2.1: Autoencoder architecture. It is composed of an encoder which, given an input x, extracts its information into code h and a decoder which, given a code h, reconstructs the input $\hat{x}$

time t). $\hat{A}_t$ is normally computed using the Generalized Advantage Estimation algorithm (GAE).

The PPO algorithm gained much popularity thanks to its simplicity and effectiveness. Nevertheless, the number of interactions it needs to have with the environment can be significantly high. In particular, complex tasks take millions if not billions of iterations to achieve good results. Melo et al. (2021) ran the algorithm for 200 million steps, achieving state-of-the-art results after 75 million steps. As stated by the authors, this result is already an improvement compared to Abreu et al. (2019). However, it is not always possible to have so many interactions with the environment for different reasons. It can be too computational intensive in highly detailed physics-based simulation. Moreover, gathering so many experiences outside of simulations (i.e. in the real world) is, at best, impractical and potentially risky for the agent/robot.

2.2 Autoencoder

An autoencoder is a simple network which is trained to obtain as output the input (Goodfellow et al. (2016)). An autoencoder can be seen as two separate entities: an encoder and a decoder (Figure 2.1). Therefore, it must have a hidden layer which represents what is known as code or latent representation. The encoder takes the input x and returns the code (i.e. $encode(x) = h$). The decoder, on the other hand, takes the code and returns the reconstructed input (i.e. $decode(h) = \hat{x}$).

At first glance, this network does not seem to be useful if $decode(encode(x)) = x$ (which is the objective of the network). However, most autoencoder architectures are designed so that this objective cannot be reached. These restrictions force the network to learn valuable properties of the data. Properties that are represented by the code

One possible restriction is to force the code to be significantly smaller than the input. Therefore making the perfect reconstruction of the output almost impossible. In this way, the network is forced to learn the essential features of the data. This type of autoencoder is known as undercomplete autoencoder. For convenience, we will always refer to it as autoencoder. The loss function $L(x, decode(encode(x)))$ for an autoencoder is usually a distance or divergence metric (e.g. mean squared error (MSE))

A notable property of such a network is that when the loss is the MSE and the decoder is linear, the learned code has the same subspace as principal component analysis (PCA). As stated by Goodfellow et al. (2016), a nonlinear autoencoder is a better nonlinear generalization of PCA. Many empirical studies that compared PCA to AE ,in a variety of different field,have also proven this theoretical finding (Y. Wang et al. (2016),Lončarević et al. (2021),Almotiri et al. (2017),Siwek & Osowski (2017)). Specifically, Lončarević et al. (2021) showed that AE-based latent representation outperforms PCA-based latent representation when used by an RL agent. AE showed a smaller reconstruction error with respect to PCA. The RL agent also converged faster when an AE-based representation was used instead of a PCA-based one. This result is linked to the fact that AEs cannot only perform dimensionality reduction but also, as stated by Y. Wang et al. (2016), find repetitive structure. This property is crucial in robotics and, more generally, when aggregating multiple input sensors, which may or may not encode similar information.

2.3 Variational Autoencoder

A Variational Autoencoder(VAE)(Kingma & Welling (2013)) has an almost identical architecture to an autoencoder(Figure 2.2). However, its goal and learning process is significantly different. As opposed to an AE (discriminative model), a VAE is a generative model. It, therefore, aims to learn the joint distribution over the latent variables. There are many reasons why learning a generative model can be more useful and generalizable than learning a discriminative model (Kingma et al. (2019)). A generative model spontaneously learns causal relationships and is robust

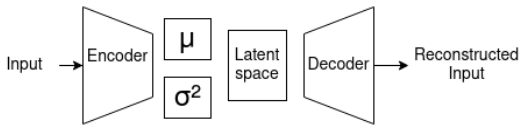


Figure 2.2: Variational Autoencoder architecture. The encoder maps an input into a mean and variance, which defines an isotropic multivariate Gaussian distribution. This distribution represents the code/latent space. The decoder samples from it and reconstructs the input.

against nuisance variables. These properties make generative models more suitable for learning good representations.

The VAE architecture is composed of a probabilistic encoder and decoder. This means that the encoder, instead of mapping the input into a single vector (like an AE), it maps the input into a distribution over the latent variables. On the other hand, the decoder is the generative model which samples from the latent space and produces an output (as similar as possible to the input).

Formally, given a dataset x and a set of random variables z, that represent the code or latent space, we want to know the posterior probability distribution of the latent variables given the dataset. In other words, we want to find $p(z|x)$. We must assume two points to rephrase this problem within a Bayesian framework. First, the code is sampled from some prior distribution $p(z)$ (usually, in standard VAE, the prior is an isotropic multivariate Gaussian distribution). Second, the data x is sampled from some conditional distribution $p(x|z)$. Note that these two assumptions are quite soft and natural in the context of generative models. We can therefore rewrite $p(z|x)$ as:

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}$$

However, calculating the evidence integral $p(x) = \int p(x,z)dz$ is often intractable or too expensive to compute. Therefore, Variational Inference (VI) can be used to approximate the posterior (for an extensive explanation of VI, see Blei et al. (2017)). The core idea of VI is to reframe the problem into an optimization problem. Using the KL-divergence to estimate the goodness of the approximation. By doing so, it derives the Evidence Lower Bound (ELBO). Maximizing the ELBO results in a better approximation. Therefore it can be used as a loss function with an inverted sign. The ELBO has different forms, but the one used in the VAE is:

$$ELBO(q_x) = \mathbb{E}_{z\ q_x}[log(p(x|z))] - KL(q_x(z)||p(z))$$

$p(x|z)$ is the probabilistic decoder, and q_x is the approximated posterior distribution. q_x is given by the following formula, where the probabilistic encoder approximates g and h:

$$q_x(z) = \mathcal{N}(g(x), h(x))$$

Finally, the VAE architecture outperformed PCA and other standard techniques in different dimensionality reduction tasks (Portillo et al. (2020),Lin et al. (2020)). It also had some promising results when used in combination with RL. Prakash et al. (2019) showed impressive results with respect to their baseline (which was without the VAE). It is important to notice that the observation space of the agent was an image. Which partially explains the drastic improvement from the baseline. However, other studies like Q. Wang (2022) showed that improvement upon other methods can be achieved even if the observation space is not pixel-based.

3 Methods

This section presents all the technical details needed to replicate the study. From the simulation software to the implementation details of the learning process. It concludes by explaining the hyperparameters for the different learning techniques.

3.1 Simulation

The simulation program of choice for this study is Opensim version 4.3-2021-04-14-dbde45530. Opensim is open-source physics-based simulation software that allows to create and analyze the dynamics of complex musculoskeletal model (Delp et al. (2007)). A variety of reasons led to choosing this system. First of all, multiple competitions such as "NeurIPS 2019: Learn to Move - Walk Around" and "NIPS 2017: Learning to Run" decided to use Opensim as their simulation software. It appears

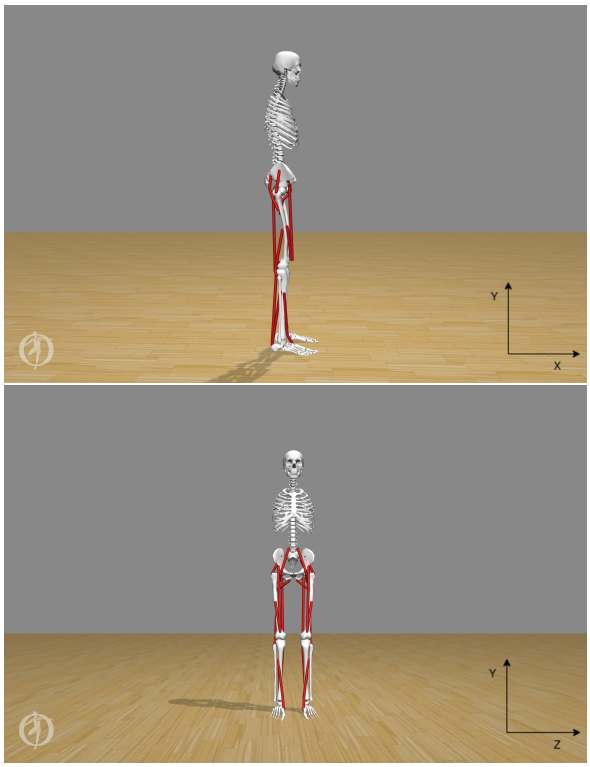


Figure 3.1: Overview of the Opensim model used in the study. The red colour identifies the 22 muscles.

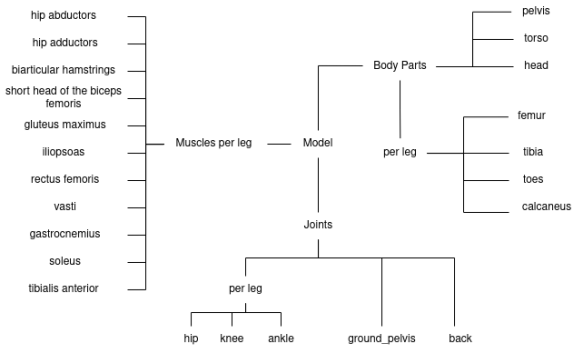


Figure 3.2: Overview of the different parts (taken into consideration)that compose the model.

3.2 Model

The model used in this study is a simplified version of a human musculoskeletal model (see figure 3.1). It was developed by Ajay Seth and adjusted by Carmichael Ong (Seth et al. (2018)). The model has 11 muscles per leg. For a total of 22 muscles. Moreover, it has 14 degrees of freedom: four at the knee and ankle joints (2 per leg), six at the pelvis (tilt, list, rotation,x,y,z) and four at the hip joints (flexion and adduction for each leg). A Hill-type muscle model simulates the muscles. This type of muscle model is not the most accurate in mimicking human muscles. However, its state equation is notorious for being fast to compute. This property makes it a perfect alternative to more realistic but complex systems. Moreover, the model gives information about different body parts, joints and the centre of mass (x,y,z) (Figure 3.2). It also takes into account the ground forces on the feet. Specifically, there are three components (x,y,z) for both the force and the torque applied to each foot. Resulting in a total of 12 values.

Finally, table 3.1 shows the exact information used by the AE architecture. Even though Opensim gives access to the accelerations, this study does not consider them. The two main reasons for this decision are: First, the accelerations, particularly during training, had many outliers, making learning harder. Second, the reward system (more about it in section 3.3) does not explicitly model acceleration.

to be the primary simulation in most human locomotion studies. Showing its reliability and accuracy. Finally, De Vree & Carloni (2021) used this software. Their work highly inspired this study. In their paper, they proposed two models, a healthy one and a transfemoral amputee one. They used Opensim to simulate these models' dynamics and a PPO agent to learn ground-walking in both situations.

One last feature of this software that has proven to be quite valuable is the ability to run motion files. Motion files are usually generated from data gathered from actual human motion. This feature was crucial in analyzing, processing and checking the imitation data used in the reward system for the DRL agent (more on the imitation data in section 3.3)

3.3 Reward System

The most vital component in a reinforcement learning framework is the reward system. This system is responsible for giving some kind of feedback to the agent, which can be positive or negative. The agent, on the other hand, has the objective to maximize the reward intaken.

Designing a good and correct reward is not, by any means, trivial. There are several risks when constructing it. A misspecified rewards system can lead to adverse side effects on the final agent behaviour. As explained by Hadfield-Menell et al. (2017). They can also lead to what is known as reward hacking. The work done by Amodei & Clark (2016) is an excellent example. The wanted goal was to win a racing game. However, due to poor reward specification, the agent ends up looping. It collects points in a circle without actually winning the race. Even if this was not the intended behaviour, it was the optimal/best behaviour found to maximize the reward intaken.

In this study, the reward system is composed of two main components. The goal reward should encourage the agent to move forward without falling. And the imitation reward, which should guide the forward movement to reassemble as close as possible a human-like movement. The designing decisions on the reward are highly based and influenced by De Vree & Carloni (2021) and Peng et al. (2018)

The goal reward is the MSE between the agent and the desired velocity. In particular, the velocities on the x and z axes are used. It is finally scaled using an exponential function.

$$reward_{goal} = e^{-8*(diff_{vel_x}+diff_{vel_z})}$$

Surana (2021) inspired the values of the scaling factors, which were found experimentally.

The imitation reward, on the other hand, is slightly more complex. It is composed of two parts. The difference in position and velocity between the agent and the imitation data(more on imitation data in section 3.4). MSE is used to calculate each difference. The joints and body parts considered are: knees, hip (adduction and flexion), ankles and pelvis (rotation, tilt, list). The total imitation reward is calculated as follows:

$$reward_{imi} = 0.75 * e^{-2*diff_{pos}} + 0.25 * e^{-0.1*diff_{vel}}$$

Note that the difference in position concerning the imitation data is more important than the difference in velocity. Two main reasons motivated this imbalance. First, the position is more determinant than velocity when it comes to imitation. Second, the velocity is already partially modelled by the goal reward.

Finally, the two rewards are weighted and summed together to form the final reward that the agent will use.

$$reward_{final} = 0.6 * reward_{imi} + 0.4 * reward_{goal}$$

The weights used are taken from De Vree & Carloni (2021) and were found to be the best experimentally.

Table 3.1: Number of selected dimensions for each component

Components	Number
Body parts (pos,vel,pos_rot,vel_rot)	$39 + 39 + 39 + 39$
Muscles (force,length,vel,activation)	$22 + 22 + 22 + 22$
Joints (pos,vel)	$17 + 17$
Centre of Mass (pos,vel)	$3 + 3$
Ground forces (force,torque)	$6 + 6$
Total	296

3.4 Imitation Data

Human data were used in this paper to calculate part of the reward. Therefore, using similar imitation data to replicate the study is vital. In this sub-section, the pre-processing made on the data will be presented.

First of all, the data was collected from a public database named Camargo Dataset. This dataset includes a variety of human data. The subject used in this study was AB06. The starting position was shifted to 12.8 seconds when the actual trial started. The data were trimmed when the subject started to walk in a circle (which was the trial objective). It was trimmed precisely after 3.9 seconds from the start of the trial. Then it was translated

and transformed to comply with the OpenSim environment. Specifically, it was translated to start from the origin and rotated by -90 degrees on the Y axis. Finally, the Opensim software scaled the data to fit our model and created the inverse kinematics, which will be used in the imitation reward (for more information about process see Ajay et al. (2022)).

3.5 Training Process

There are two main approaches in the literature used to train this autoencoder plus reinforcement learning architecture.

The first one can be seen as an online approach. It usually consists in training the AE and RL at the same time. However, this approach can result in training instability and bad results if done naively. This can happens because there is a feedback loop between the two learning processes. The autoencoder shapes the observation, which influences the RL. Meanwhile, the RL takes action based on the observation, influencing the following observation. Hence it biases the training of the AE. Therefore, most studies that take this approach have to design custom architecture. Which aims to control better the interaction between the two parts.

For instance, the work done by Yarats et al. (2021) is an excellent example. Their aim is similar to the one of this study: improving sample efficiency. To control the issues mentioned above, they create a custom architecture. This architecture uses three different gradients to update different system parts. For instance, the encoder is updated with two gradients. One coming from the reconstruction error (as in standard AE) and one coming from the soft Q-Learning. There is another gradient which is responsible for updating the policy.

However, since these online methods are relatively new, this study decided to take a "safer" and more conventional approach. This second method can be seen as an offline one. It consists of two phases (using the terminology from Higgins et al. (2017)): learn to see and learn to act. During the first phase, observations are gathered from the environment and used to train an AE in a classical unsupervised fashion. In the second phase, the pretrained encoder compresses the observation, which the learning agent uses. This approach completely avoids the problems that can arise using the first method. However, it has a few downsides.

Firstly, it needs to acquire observations from the environment. This process can be problematic, mainly if gathering them is not a trivial task (i.e. in the real world). On the other side, this can also be useful if data on the environment is already available. Particularly true for pixel-based input. Secondly, the gathered observations should cover as much as possible the entire observation landscape. This property is necessary to learn a good representation with the AE. VAE architectures, thanks to their probabilistic nature, can partially alleviate this problem.

This study uses simulation software as the environment. Therefore gathering observation is not an issue. However, if this is a concern, some statistical approaches (e.g. Gaussian Process Regression (GPR)) can be used to augment the datasets (see Abbasi et al. (2021) and Lončarević et al. (2021)).

In the literature, fixed policies are generally used to gather observations during the first phase. However, in this paper, creating such a fixed policy was not trivial due to the complexity of the task. Furthermore, using a random policy resulted in poor coverage of the observation landscape. For this reason, an RL agent was used. Once collected, the outliers were removed, and the dataset was normalized (using the z-score approach). Finally, the dataset was reduced to having only uncorrelated data points. All these pre-processing operations were done on the base of LeCun et al. (2012) which, as stated, should speed up and improve the learning process (e.g. better generalization properties).

During the second phase (i.e. Learn to act), the PPO agent was trained on the environment using the pre-trained encoder to create the compressed representation. It is essential to note that no gradient updates will modify the encoder during this phase.

3.6 Implementation details

The PPO algorithm was not implemented from scratch. Stable-baselines3 (SB3) by Raffin et al. (2021) was used. This library offers a variety of functionalities which were crucial for the development of the study.

Table 3.2 shows the hyperparameters used in the PPO algorithm. The values were found experimentally, based on previous studies (De Vree & Carloni

(2021),Surana (2021)). No hyperparameter search algorithm was applied.

The PPO algorithm contains two networks. The value function network and the policy network (both implemented by a MultiLayer Perceptron (MLP)). The networks do not share any parameters. They are composed of three hidden layers of 312 neurons. The dimensionality of the selected latent representation defined the width of the input layer. The output layer of the policy network had 22 neurons, each corresponding to a different muscle in the model. Lastly, the tanh function was used to squash the output. (in SB3, the parameter is called squash_output). Squashing the output is a common practice in DRL. The reason is that the policy network often does not output sensible values for the action-space boundaries.

On the other hand, the autoencoder networks were implemented using the library Pytorch developed by Paszke et al. (2019). The implementation does follow the standard theory for both the autoencoder and the variational autoencoder.

Table 3.3 presents the hyperparameters used for the two autoencoders. This study also defines a custom function to decide the number of neurons per layer. The function is defined by the following system of equations.

$$\begin{cases} n_1 = I \\ n_N = Z \\ n_i = n_{i+1} * \lambda \end{cases} . \tag{3.1}$$

The first two equations constrain the fact that the dimensionality of the input layer and the output layer is known (I input neurons and Z output neurons). The last equation force a constant shrinking of the number of neurons (assuming $Z >> I$). This system of equations can be rewritten in a single formula:

$$n_i = Z * \left(\frac{I}{Z}\right)^{\frac{N-i}{N-1}}$$

Where i is the layer index, and N is the number of layers.

Table 3.2: Selected Proximal Policy Optimization Hyperparameters

Parameter	SB3 name	Value
Seed	seed	42
Parallel environments	num_envs	20
Steps per worker	n_steps	1024
Epoch per update	n_epoch	4
Minibatch size	bach_size	512
Discount factors	gamma	0.999
Bias vs variance (GAE)	gae_lambda	0.9
Clip range (ϵ)	clip_range	0.2
Entropy coefficient	ent_coef	0.01
Learning rate	learning_rate	0.001

Table 3.3: Selected AE and VAE hyperparameters

Parameter	Value
Epochs	750
Hidden layers	6
Latent/Code dimensionality	66
Learning rate	0.001
Optimizer	AdamW

4 Results

This section first analyzes and compares the latent spaces generated by the autoencoders. The comparison is based on the reconstruction error. Then, it dives into the actual performance of the reinforcement learning algorithm. The performance of the architecture will be compared to the canonical PPO algorithm.

4.1 Latent space and Reconstruction Error

The different latent spaces are generated from the 296-dimensional observation space (as shown in Table 3.1). Each of them has 66 latent dimensions. Therefore, it is impossible to visualize them fully without losing meaningful information. For this reason, a 2-dimensional VAE and AE were trained.

The figures 4.1 show the two latent space distributions. The most important difference between the two is their shape. In the VAE latent space, we can see that the overall shape is close to an isotropic multivariate Gaussian distribution. Moreover, the distribution has an almost identical range for the two latent dimensions (i.e. θ^1_{VAE} and θ^2_{VAE}). These properties are perfectly in-line with the objective of the learning process. On the other hand, the AE latent space was not forced to comply with any prior

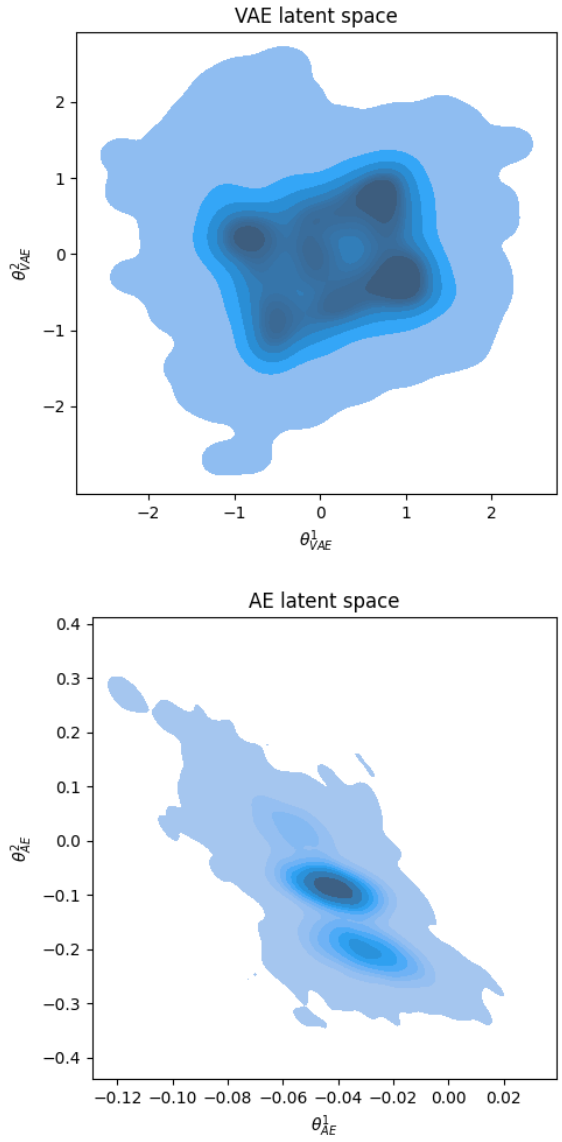


Figure 4.1: Variational Autoencoder and Undercomplete Autoencoder Latent spaces. The figures represent the projection of the testing data onto the latent space. The intensity of the colour represents the density. A Kernel Density Estimation (KDE) was used to represent the distribution.

distribution. Therefore, the AE found the (pseudo-)optimal distribution given its objective (i.e. minimize the reconstruction error). Finally, as opposed to the VAE latent space, the ranges of the two latent dimensions are significantly different.

That said, by using the reconstruction error, it is possible to have a rough indication of the "goodness" of the learned representation. The reconstruction metric of choice was the MSE. The baseline was a fitted Principal Component Analysis. In table 4.1 we can see the results. In general, the autoencoders performed slightly better than a standard PCA. This result shows again that autoencoders are generally better than PCA, thanks to their ability to perform nonlinear operations. The other notable outcome was that AE had a lower reconstruction error than VAE. It is not entirely unexpected. The VAE implicitly makes a tradeoff between the similarity of the latent distribution to the prior (i.e. by having the KL-divergence in the objective) and the reconstruction error. This can explain why the AE perform better under this metric.

4.2 Reinforcement Learning performance

The reconstruction error does not entirely assess the "goodness" of the learned representation. Therefore, we can compare the RL result using those representations. Figure 4.2 shows the learning curves of the different architectures. The first five million steps of the training process are shown .

Figure 4.2 compares the two proposed architectures and the baseline PPO+IL. It clearly shows that the autoencoder methods perform significantly better regarding mean episode duration. Specifically, AE-PPO+IL has a 131% increase in mean episode duration than PPO+IL. Meanwhile, VAE-PPO+IL improve over PPO+IL by a 102%. This

Table 4.1: Reconstruction error based on testing data gathered during the first phase of the training process (see Section 3.5)

Method	Reconstruction Error
VAE	0.0013 ± 0.0070
AE	0.0004 ± 0.0042
PCA	0.0023 ± 0.0065

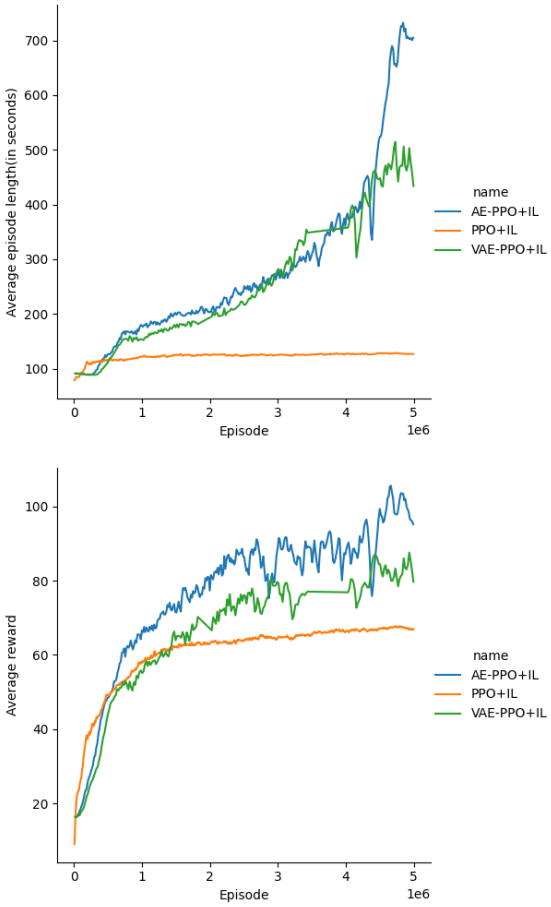
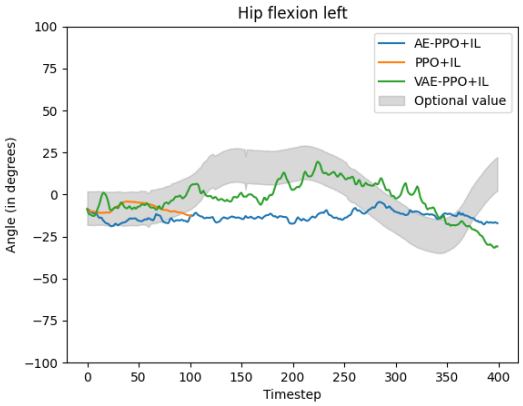


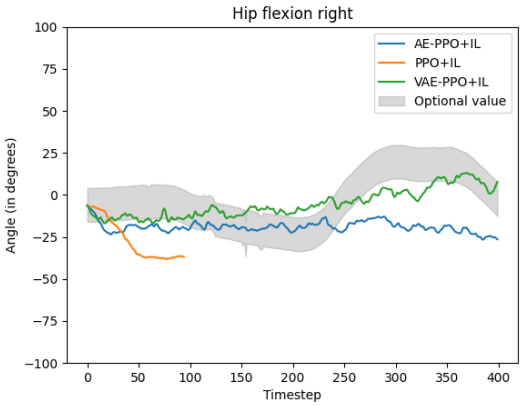
Figure 4.2: The learning curves of the three DRL architectures (AE-PPO+IL, VAE-PPO+IL, PPO+IL). The top plot shows the average episode length (in seconds). The bottom one presents the average episode rewards.

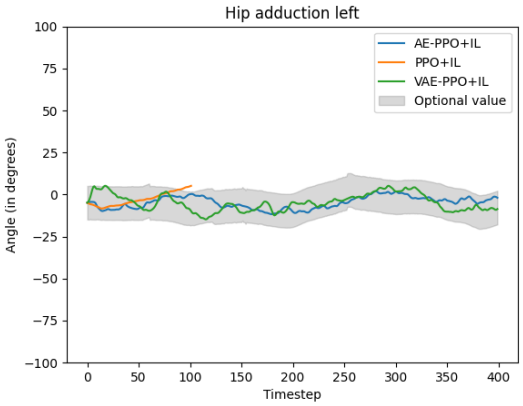
discrepancy in mean episode duration shows that the two autoencoder methods learn better to avoid falling compared to the baseline. Moreover, since they survive more in the environment, they can explore the state space more, resulting in better local optimal. On the other hand, they perform only marginally better than PPO+IL in terms of mean cumulative reward. AE-PPO+IL and VAE-PPO+IL improve over the baseline by 23% and 9%, respectively. This difference between the improvement in mean episode duration and mean cumulative reward could signal reward misspecification since surviving longer in the environment should result in significantly higher rewards. In fact, the AE-PPO+IL and VAE-PPO+IL learn to balance the model while trying to be as close as possible to the imitation data in order to get better rewards. To avoid such behaviour, a change in the reward system is needed.

Figures 4.3 and 4.4 compares the different architectures with respect to their gait pattern. The figures show roughly two seconds (200 timesteps = 1.05s), which, as described by Murray et al. (1964) should correspond to two full gait cycles. The grey area represents the imitation data with a standard deviation of 10 degree. The algorithms did not learn to follow the imitation data for both knee angles properly. This is in line with the fact that the agent does not take a step. Moreover, the knee angles are more challenging to follow than the rest of the imitation data because they are a broader range of motion/value. Generally, besides the knee angles, the agent is able to stay almost always within 10 degrees from the imitation data. Table 4.2 shows the total root mean squared error over all the 14 degrees of freedom. VAE-PPO+IL has the lowest root mean squared error over all the 14 degrees of freedom. A possible explanation for the better performance of the VAE compared to the other architecture is the ability to learn causal relationships (as explained in 2.3). This could have helped the RL agent better understand the environment's dynamics, making it easier to follow the imitation data while balancing in place.

The above result also translated into lower muscle forces and activation for the VAE-PPO+IL compared to the AE-PPO+IL. Table 4.3 and 4.4 summarises the activations and fiber forces of the muscles around the knee and the ankle joints. VAE-PPO+IL has in total 6.8% less activation and







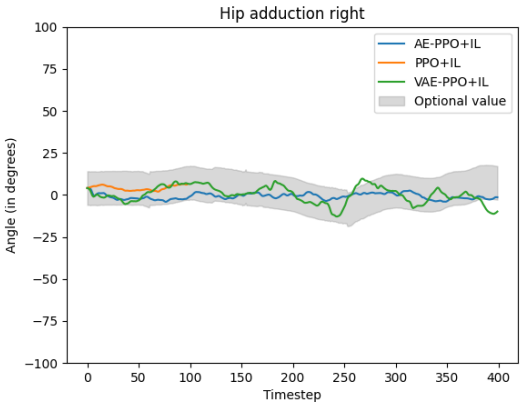
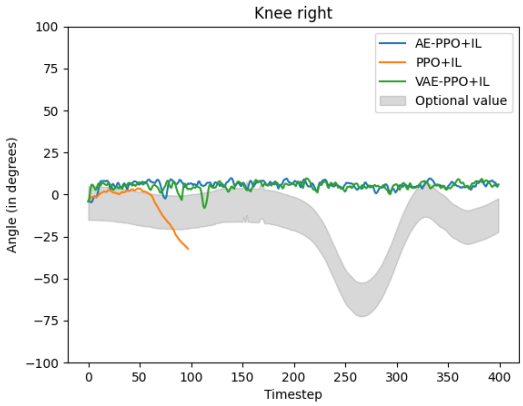
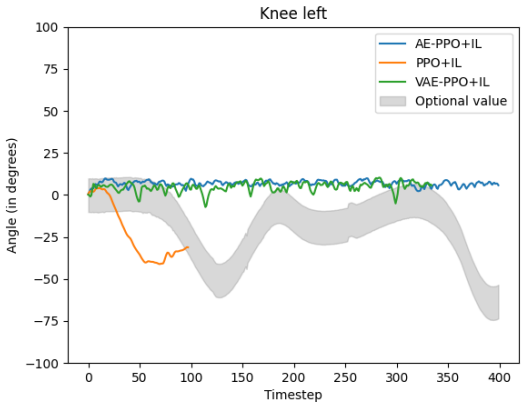
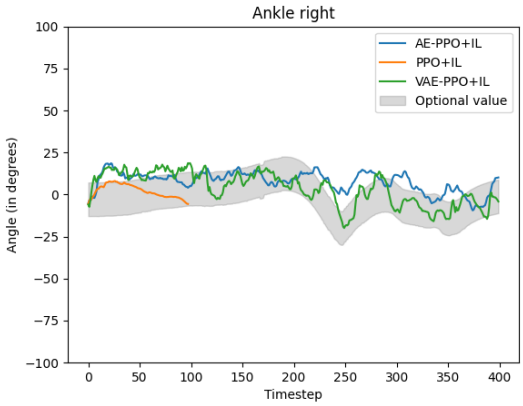


Figure 4.3: Gait cycles of the hip joint flexion and adduction. The angles are expressed in degrees. The grey area represents the imitation data with 10 degrees of standard deviation.







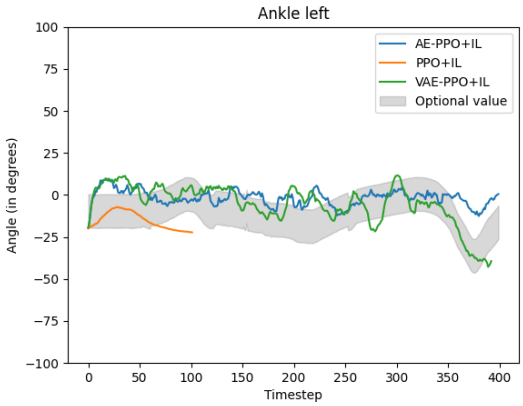


Figure 4.4: Gait cycles of the knee and ankle joints. The angles are expressed in degrees. The grey area represents the imitation data with 10 degrees of standard deviation.

11.1% less fiber force compared to AE-PPO+IL. This shows that VAE-PPO+IL is more energy-efficient. Figure 7.1 and 7.2 in the appendix show in more detail the fiber force of the muscles around the knee and the ankle joints for 2 gait cycles.

Lastly, the value loss during the training of the AE-PPO+IL and VAE-PPO+IL was relatively high. The value loss is the ability to predict each state's value (reward). It usually increases with the increase in reward and stabilizes at convergence. However, this pattern was not noticed with the other tested methods. It can signal a high exploration rate. This phenomenon can be associated with low-dimensional observations facilitating the agent's exploration.

5 Conclusion

This paper aims to propose and compare two different autoencoders with DRL architectures. The DRL of choice was the PPO+IL which, as shown in previous studies with similar structures, had promising results. The proposed autoencoders were an under-complete autoencoder and a variational autoencoder. They have many differences, as presented in section 2. However, the core difference is the constraints they force on their latent space. The approach used to train the whole architecture was offline. This method is divided into two different phases. In the first one, the autoencoder is trained using pre-gathered data. The DRL is trained during the second phase, using the pre-trained encoder to compress the observations. This study used an RL agent to gather the data. The collected trajectories were normalized to make the autoencoders learn more efficiently. Finally, they were evaluated based on the reconstruction error and the DRL performance. It was shown that they improved the DRL's performance and efficiency. The AE performed slightly better on reconstruction error and RL performance (mean cumulative reward and mean episode duration) than the VAE. In contrast, VAE had a lower RMSE from the imitation data. Nevertheless, both agents learned to achieve relative high rewards while not performing the wanted task. This showed that a modification of the reward system is needed.

Table 4.2: Cumulative RMSE from the imitation data over all the 14 degrees of freedom (see section 3.2)

Algorithm	RMSE
VAE-PPO+IL	148.8280
AE-PPO+IL	159.9797
PPO	171.9396

Table 4.3: Activation for the muscles around the knee and ankle joints. Each entry is the sum of the right and left muscles.

Muscle	AE-PPO+IL	VAE-PPO+IL	diff
bifemsh	0.99	0.84	0.14
vasti	0.69	0.45	0.24
soleus	0.34	0.43	-0.08
tib_ant	1.21	1.3	-0.09
Total difference			0.20

Table 4.4: Fiber force for the muscles around the knee and ankle joints. Each entry is the sum of the right and left muscles.

Muscle	AE-PPO+IL	VAE-PPO+IL	diff
bifemsh	688.68	585.92	102.75
vasti	382.27	219.13	163.13
soleus	147.71	173.05	-25.33
tib_ant	665.79	717.08	-51.29
Total difference			189.26

5.1 Limitation and Future work

This study has some significant limitations which can be improved upon.

First of all, the method used to gather data is questionable. Mainly, if the aim is to generalize to the real world, using another RL agent to gather data is risky and impractical. A possible solution is to use some statistical methods to generate data starting from a few data points, as suggested by other studies (see Abbasi et al. (2021) and Lončarević et al. (2021)). In this study, we used imitation data to improve the DRL performance, this data can be used as a starting point for the data generation.

Another possible approach is to design a fixed policy to explore the state-space. However, these approaches could fail to cover enough of the observation landscape. This fallacy is especially problem-

atic if the DRL agent is allowed to explore the state-space unconstrained (e.g. extremely high muscle activation can result in unrealistic position and acceleration). Therefore, to solve this problem, the best solution is to either constrain the action space or to use different AE architecture which are focused on learning a disentangled representation (e.g. Higgins et al. (2017)). An example of constraining the action-space can be seen in Lončarević et al. (2021). In that study, they used Dynamic movement Primitives (DMP) to reduce the action-space to only sensible actions.

Another fallacy of this study was the reward system. Given the results, the agents clearly learn to maximize the reward while not performing the actual task. This is a clear case of reward misspecification as explained in section 3.3. A possible solution is to give a higher weight to the difference in velocity. Particularly, the weight of -0.1 on the difference in velocity from the imitation data should be changed. This should theoretically be an incentive for the agent to move. Another possible option is to give a higher weight to the goal reward, which is the part responsible for the x and z velocities.

Finally, the vanilla PPO algorithm can be changed/improved. New, improved versions of the algorithm were designed. A perfect example is Hsu et al. (2020). In their paper, they highlight quite some fallacies in the PPO algorithm. The most notorious is the algorithm's dependence on the initial network weights. This relation is highly problematic when trying to replicate a study which did not specify the seed and approach used to initialize the networks. Therefore, to improve the study, different DRL techniques could be tested.

6 Acknowledgements

Massimiliano Falzari (author of the thesis) would like to thank his supervisor, Raffaella Carloni (Professor, University of Groningen), for the feedback and advice needed to conduct this study. A special thanks to Rutger Luinge, Carl Lange, Elena Poeltuijn and Robin Kock for contributing to the processing of the imitation data, the development of the techniques used in this study and lastly, for creating a motivated and creative working environment.

References

Abbasi, M., Karami, M., Koushki, A., & Vossoughi, G. (2021). Autoencoder-based safe reinforcement learning for power augmentation in a lower-limb exoskeleton. In *2021 9th rsi international conference on robotics and mechatronics (icrom)* (pp. 138–143).

Abreu, M., Reis, L. P., & Lau, N. (2019). Learning to run faster in a humanoid robot soccer environment through reinforcement learning. In *Robot world cup* (pp. 3–15).

Adriaenssens, A. (2021). *Testing for generality of a proximal policy optimiser for advanced human locomotion beyond walking* (Unpublished doctoral dissertation).

Ajay, S., Jennifer, H., Sam, H., Matt, D., Jill, H., Brian, K., ... Chris, H. (2022). *Simulation with opensim - best practices.* Retrieved 2022-08-30, from `https://simtk-confluence.stanford.edu:8443/display/OpenSim/Simulation+with+OpenSim+-+Best+Practices`

Almotiri, J., Elleithy, K., & Elleithy, A. (2017). Comparison of autoencoder and principal component analysis followed by neural network for e-learning using handwritten recognition. In *2017 ieee long island systems, applications and technology conference (lisat)* (pp. 1–5).

Amodei, D., & Clark, J. (2016). *Faulty reward functions in the wild.* Retrieved 2022-08-30, from `https://blog.openai.com/faulty-reward-functions/`

Andersen, P.-A., Goodwin, M., & Granmo, O.-C. (2018). The dreaming variational autoencoder for reinforcement learning environments. In *International conference on innovative techniques and applications of artificial intelligence* (pp. 143–155).

Blei, D. M., Kucukelbir, A., & McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American statistical Association*, *112*(518), 859–877.

Delp, S. L., Anderson, F. C., Arnold, A. S., Loan, P., Habib, A., John, C. T., ... Thelen, D. G.

(2007). Opensim: open-source software to create and analyze dynamic simulations of movement. *IEEE transactions on biomedical engineering*, *54*(11), 1940–1950.

De Vree, L., & Carloni, R. (2021). Deep reinforcement learning for physics-based musculoskeletal simulations of healthy subjects and transfemoral prostheses' users during normal walking. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, *29*, 607–618.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. (`http://www.deeplearningbook.org`)

Hadfield-Menell, D., Milli, S., Abbeel, P., Russell, S. J., & Dragan, A. (2017). Inverse reward design. *Advances in neural information processing systems*, *30*.

Higgins, I., Pal, A., Rusu, A., Matthey, L., Burgess, C., Pritzel, A., ... Lerchner, A. (2017). Darla: Improving zero-shot transfer in reinforcement learning. In *International conference on machine learning* (pp. 1480–1490).

Hsu, C. C.-Y., Mendler-Dünner, C., & Hardt, M. (2020). Revisiting design choices in proximal policy optimization. *arXiv preprint arXiv:2009.10897*.

Igl, M., Zintgraf, L., Le, T. A., Wood, F., & Whiteson, S. (2018). Deep variational reinforcement learning for pomdps. In *International conference on machine learning* (pp. 2117–2126).

Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kingma, D. P., Welling, M., et al. (2019). An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, *12*(4), 307–392.

LeCun, Y. A., Bottou, L., Orr, G. B., & Müller, K.-R. (2012). Efficient backprop. In *Neural networks: Tricks of the trade* (pp. 9–48). Springer.

Lin, E., Mukherjee, S., & Kannan, S. (2020). A deep adversarial variational autoencoder model for dimensionality reduction in single-cell rna sequencing analysis. *BMC bioinformatics*, *21*(1), 1–11.

Lončarević, Z., Gams, A., Ude, A., et al. (2021). Robot skill learning in latent space of a deep autoencoder neural network. *Robotics and Autonomous Systems*, *135*, 103690.

Luo, F.-M., Xu, T., Lai, H., Chen, X.-H., Zhang, W., & Yu, Y. (2022). A survey on model-based reinforcement learning. *arXiv preprint arXiv:2206.09328*.

Melo, L. C., & Máximo, M. R. O. A. (2019). Learning humanoid robot running skills through proximal policy optimization. In *2019 latin american robotics symposium (lars), 2019 brazilian symposium on robotics (sbr) and 2019 workshop on robotics in education (wre)* (pp. 37–42).

Melo, L. C., Melo, D. C., & Maximo, M. R. (2021). Learning humanoid robot running motions with symmetry incentive through proximal policy optimization. *Journal of Intelligent & Robotic Systems*, *102*(3), 1–15.

Murray, M. P., Drought, A. B., & Kory, R. C. (1964). Walking patterns of normal men. *JBJS*, *46*(2), 335–360.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc. Retrieved from `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`

Peng, X. B., Abbeel, P., Levine, S., & Van de Panne, M. (2018). Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions On Graphics (TOG)*, *37*(4), 1–14.

Peters, J., & Schaal, S. (2008). Natural actor-critic. *Neurocomputing*, *71*(7-9), 1180–1190.

Portillo, S. K., Parejko, J. K., Vergara, J. R., & Connolly, A. J. (2020). Dimensionality reduction of sdss spectra with variational autoencoders. *The Astronomical Journal*, *160*(1), 45.

Prakash, B., Horton, M., Waytowich, N. R., Hairston, W. D., Oates, T., & Mohsenin, T. (2019). On the use of deep autoencoders for efficient embedded reinforcement learning. In *Proceedings of the 2019 on great lakes symposium on vlsi* (pp. 507–512).

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., & Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, *22*(268), 1-8. Retrieved from `http://jmlr.org/papers/v22/20-1364.html`

Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015). Trust region policy optimization. In *International conference on machine learning* (pp. 1889–1897).

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Seth, A., Hicks, J. L., Uchida, T. K., Habib, A., Dembia, C. L., Dunne, J. J., ... others (2018). Opensim: Simulating musculoskeletal dynamics and neuromuscular control to study human and animal movement. *PLoS computational biology*, *14*(7), e1006223.

Siwek, K., & Osowski, S. (2017). Autoencoder versus pca in face recognition. In *2017 18th international conference on computational problems of electrical engineering (cpee)* (pp. 1–4).

Surana, S. (2021). *Evaluating deep reinforcement learning algorithms for physics-based musculoskeletal transfemoral model with a prosthetic leg performing ground-level walking* (Unpublished doctoral dissertation).

Teixeira, H., Silva, T., Abreu, M., & Reis, L. P. (2020). Humanoid robot kick in motion ability for playing robotic soccer. In *2020 ieee international conference on autonomous robot systems and competitions (icarsc)* (pp. 34–39).

Wang, Q. (2022). Varl: a variational autoencoder-based reinforcement learning framework for vehicle routing problems. *Applied Intelligence*, *52*(8), 8910–8923.

Wang, Y., Yao, H., & Zhao, S. (2016). Auto-encoder based dimensionality reduction. *Neurocomputing*, *184*, 232–242.

Yarats, D., Zhang, A., Kostrikov, I., Amos, B., Pineau, J., & Fergus, R. (2021). Improving sample efficiency in model-free reinforcement learning from images. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 35, pp. 10674–10681).

7 Appendix

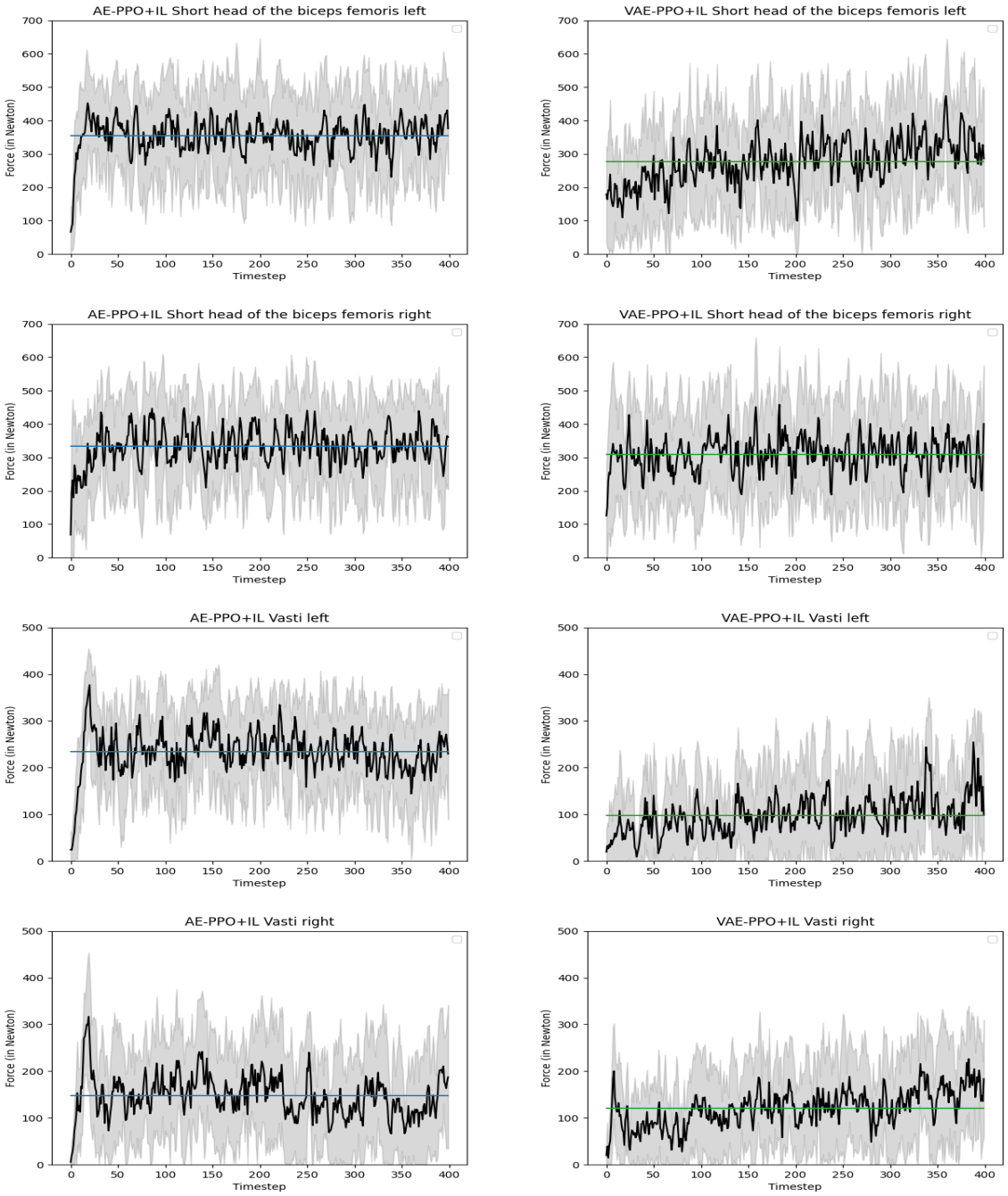


Figure 7.1: Fiber forces for the short head of the biceps femoris and vasti over 2 gait cycles for the two autoencoder architectures (left AE-PPO+IL, right VAE-PPO+IL). The blue and green lines represent the mean fiber force.

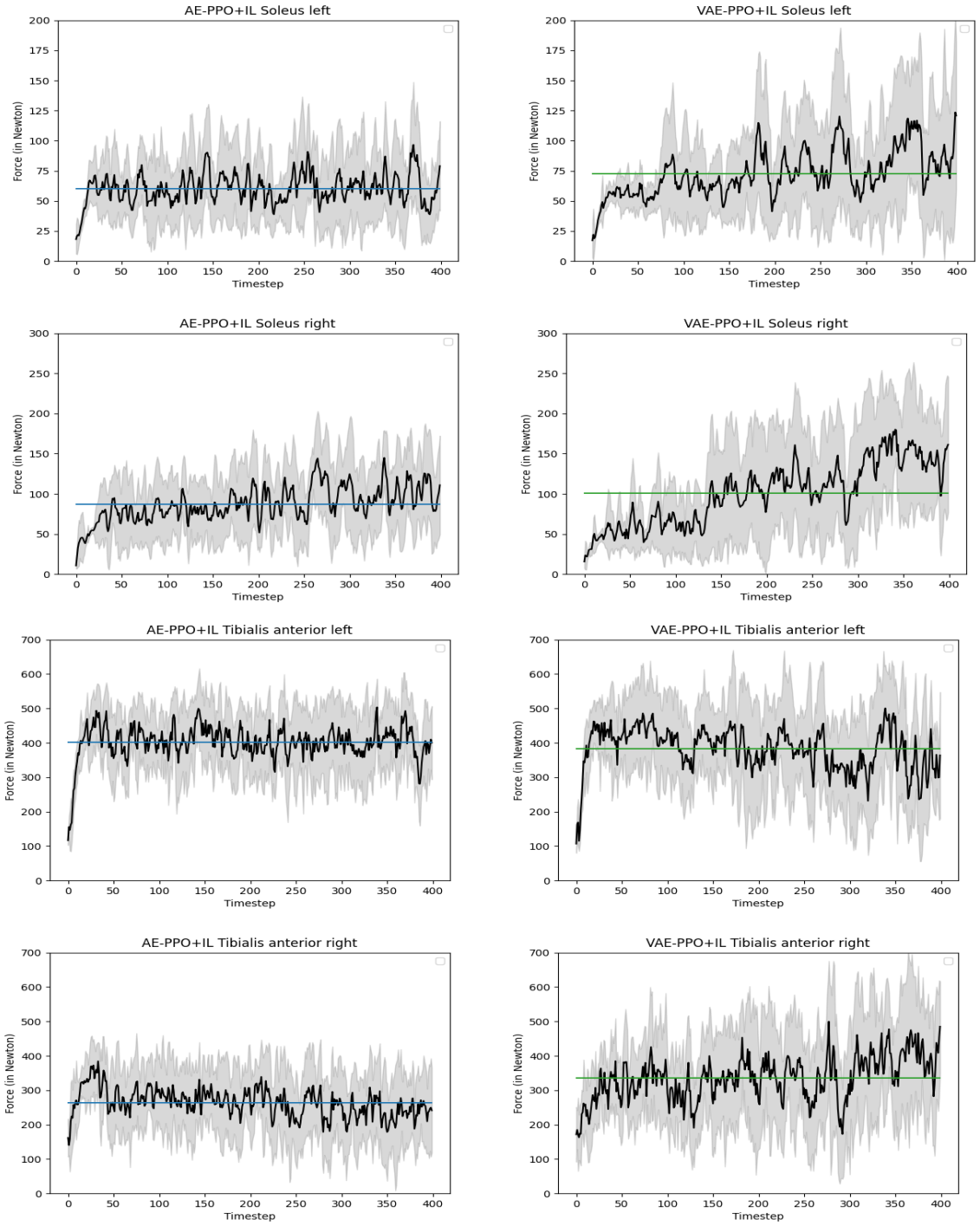


Figure 7.2: Fiber forces for the soleus and the tibialis anterior over 2 gait cycles for the two autoencoder architectures. (left AE-PPO+IL, right VAE-PPO+IL). The blue and green lines represent the mean fiber force.