TÉCNICO LISBOA

Character Locomotion using Imitation Learning from Observations with Wavelets

João Manuel da Silva Carias

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Pedro Alexandre Simões dos Santos
Prof. João Miguel de Sousa de Assis Dias

Examination Committee

Chairperson: Prof. José Luís Brinquete Borbinha
Supervisor: Prof. Pedro Alexandre Simões dos Santos
Member of the Committee: Prof. Manuel Fernando Cabido Peres Lopes

January 2021

Acknowledgments

I would like to thank my two supervisors, Professor João Dias and Professor Pedro Santos, for all the guidance, support, motivation, invested time and attention present throughout this work, as well as their counseling and patience when guiding me through the scientific communication procedure, all of which made this dissertation possible. To Professor João Dias, I'm also thankful for the possibility of exploring several topics that greatly stirred my interest and curiosity, and for the guidance in the choice process.

I am grateful to my parents and my sister for their love and support throughout my life, for the encouragement towards growth and for believing in me. To my late grandparents for their love, and in particular to my grandmother Catarina for her caring, generosity, and for setting an example of hard work and perseverance. To my uncles, aunties and cousins for their love and presence.

I would like to thank my friends for the companionship, affection and embrace that helped me grow as a person.

Abstract

This dissertation presents a novel extension to the paradigm of Imitation Learning from Observations by integrating frequency information into the learning structure, in the context of 3D physically-based running animation. This kind of animation is very repetitive, thus it may be possible to improve the system by introducing frequency information. In this thesis, we proposed a Model for Wavelet Augmented Imitation Learning from Observations, by implementing a Wavelet Transform extraction component on top of DeepMimic, and conducted empirical tests to further develop and test our approach against the DeepMimic framework. Results show that the implemented model offered no benefit to the trained policy. Moreover, there was a dramatic increase in training time and the obtained policy had similar performance but was less resilient to external forces, compared to the base system. The code is available at https://github.com/jocarias/DeepMimic.

Keywords

Imitation Learning from Observations; Continuous Wavelet Transform; Convolutional Neural Network; Reinforcement Learning; DeepMimic.

Resumo

Esta dissertação apresenta uma extensão ao paradigma de Aprendizagem por Imitação de Observações através da integração de informação de frequências na estrutura de aprendizagem, no contexto de animação de corrida 3D com base em simulação física. Este tipo de animação é muito repetitiva pelo que deverá ser possível melhorar o sistema ao introduzir informação de frequências. Nesta tese, nós propusemos um Modelo para Aprendizagem por Imitação de Observações Aumentada com *Wavelet*, através da implementação do componente de extracção de Transformada de *Wavelet* tendo o DeepMimic como base, e realizando testes empíricos para desenvolver e testar a nossa abordagem em relação ao DeepMimic. Os resultados mostram que o modelo implementado não apresenta qualquer benefício para a política treinada. Além disso, houve um aumento drástico no tempo de treino e a política obtida teve um desempenho semelhante com menor resiliência a forças externas, comparando com o sistema base. O código é disponibilizado em https://github.com/jocarias/DeepMimic.

Palavras Chave

Aprendizagem por Imitação de Observações; Transformada de Wavelet Contínua; Rede Neural Convolucional; Aprendizagem por Reforço; DeepMimic.

Contents

List of Figures

Acronyms

CNN	Convolutional Neural Network
COI	Cone Of Influence
CPU	Central Processing Unit
CWT	Continuous Wavelet Transform
ET	Early Termination
FIFO	First In First Out
GPU	Graphics Processing Unit
ILFO	Imitation Learning from Observations
Mocap	Motion capture
MNIST	Modified National Institute of Standards and Technology
MPI	Message Passing Interface
PD	Proportional-Derivative
PPO	Proximal Policy Optimization
ReLU	Rectified Linear Unit
RSI	Reference State Initialization
WAILFO	Wavelet Augmented Imitation Learning from Observations

1

Introduction

Contents

1.1 Motivation

In the context of 3D computer games, virtual characters play a fundamental role in the quality of immersion experience provided to the player. In order for a character to blend in with its surroundings, and thus providing a believable setting, it must show a natural and realistic animation - often as common forms of locomotion like walking and running.

Traditionally, these kinds of animations require specialized skills and are time-consuming to produce. Moreover, they are not able to react to unforeseen or unprepared interactions, which are bound to happen in complex scenarios, degrading the player's experience.

Physically-based animation [7–10] promotes the correct behavior of objects and characters by exposing them to the laws of physics and thus, providing a realistic experience. A simple example is Ragdoll Physics, commonly used to animate a character's body in specific situations, like unconscious/death animation.

In recent times, important developments in Deep Reinforcement Learning and to the hardware which it is run on are allowing new efforts to emerge leading to the creation of a controller that is able to command the members of the simulated body, orchestrating a set of synchronized actions that produces an intended animation, reacting naturally and intelligently to unplanned interactions. In this context, locomotion animations, like walking and running, are an active area of intensive research and several approaches have been presented, which can be divided by the ones that need motion data [7, 8] and the ones that don't [9, 10]. In terms of the former, while achieving impressive results, some limitations need to be addressed, such as improving the flexibility of the animation time-wise and the robustness, given external forces.

It's also worth noting that improving locomotion of a virtual agent in a simulation may offer benefits, in some cases, to the control of robotics systems in the real world.

1.2 Problem

This work will focus on the development of a system that, on a human-like 3D character in a physics-based simulation, will be capable of generating realistic and robust running animation, from motion data.

The system characteristics we seek to improve are at the level of robustness, the ability to overcome external random forces, and the time it takes for the system to reach a realistic animation, one that agrees with a person's expectation. A relevant benefit of contextualizing this problem in a physics-based simulation is the inherent possibility of finding policies capable of recoveries from unexpected forces, making the locomotion more realistic.

1.3 Hypothesis

Motion data can be a flexible source of different locomotion types and styles. Imitation Learning from Observations - learning a policy with data from another agent performing a task - appears to us as the ideal framework from which a system can be based on, in order to learn simple or even complex motions. For these types of systems in this context, the input information is often a set of state variables (position, angles, velocities) with the current configuration of the character's body (a set of members/links connected by joints).

In this work, we will explore a new idea based on the observation that the animation that is our focus - running on even ground - is very repetitive, meaning every so often the same movement is executed just with minor adjustments that are needed for a stable locomotion. Given the periodicity of the various movements, it is our belief that it can be processed in the frequency domain along the traditional time domain, adding otherwise hidden information that may help our objectives.

The Fourier Transform and its close variants have been used to better analyze periodic signals though it suffers from the inability to localize, with some precision, frequency in time, which will be a necessity for the types of signals that change abruptly. Wavelets, a more recent approach, have both good time and frequency localization. This rationale has guided good results, both in images and sound processing and particular speech recognition, though these types of signal are quite different among themselves and from the task at hand. Additionally, Wavelets have been used to analyze walking patterns [11, 12] and even to compress Human Motion Capture [13].

We believe that the use of Wavelets may bring additional unexplored advantages to solve our problem so it is our interest to explore the possibility of using this mathematical tool to transform how the system processes data and understand its effects on tasks previously mentioned, advantages, if any, and its limitations.

1.4 Document structure

The next chapter exposes relevant foundations, Chapter 3 presents previous related works, Chapter 4 contains an overview and both the implementation and evaluation of the main contribution and, lastly, the conclusion of this thesis in Chapter 5.

2

Theoretical Background

Contents

In this section we provide a brief explanation of several concepts important for this thesis. First, an introduction to Wavelets theory is presented in order to help the understanding of the main idea of this work. Next, Imitation Learning from Observations and its context is given, being this the main learning paradigm. Lastly, we'll explain the proximal policy optimization algorithm given its frequent use in Reinforcement Learning problems, including locomotion.

2.1 Wavelets

Wavelets [14], as a mathematical tool, allow the analysis of a signal both in time and frequency. It consists of using a particular wavelike function (mother wavelet) as a basis for obtaining a set of modified wavelet functions that are used to transform the signal (wavelet transform) from continuous-time to time-frequency representation (see Fig. 2.1). The result can be observed in a Scalogram - a representation of the obtained coefficient values in time and scale.

$$T(a,b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} \Psi^* \left(\frac{t-b}{a} \right) x(t) dt$$

Above is the formula for the Continuous Wavelet Transform where **a** corresponds to scale, **b** is time and * is the complex conjugate of the mother wavelet Ψ. It is possible to recover the original signal $x(t)$ using an inverse transform.

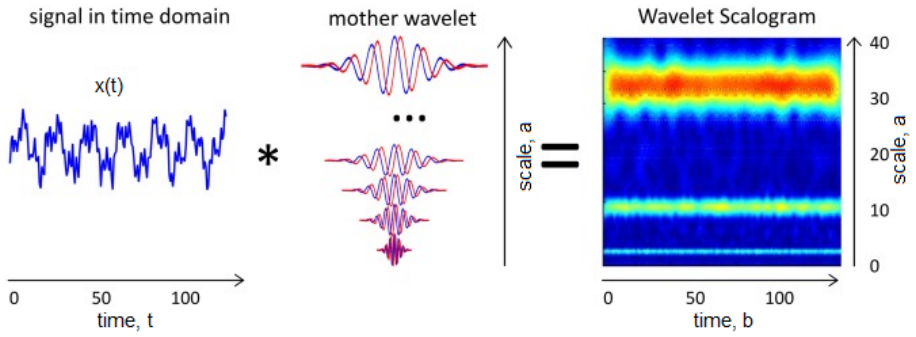

Figure 2.1: The signal $x(t)$ is transformed by innumerous scale iterations of a (Morlet) mother wavelet and by innumerous translations along the time axis resulting in the separation of frequencies through time, shown in the Scalogram. Warmer colours represent higher coefficient values and higher scales represent lower frequencies (figure adapted from [1]).

The mother wavelet can be altered by the scale factor **a**, resulting in a squeezed (useful for detecting high frequencies) or stretched function (for low frequencies), and the time shift factor **b** that moves the squeezed/stretched function throughout the signal, performing what can be seen as a convolution and resulting in a set of coefficients.

In a computational setting, depending mostly on how the scale and translation factors are discretized, we can have a Continuous Wavelet Transform (CWT) or a Discrete Wavelet Transform (DWT), each

having their advantages and disadvantages. By having a sparser scale discretization, the DWT allows a more compressed representation of the signal while demanding less computation.

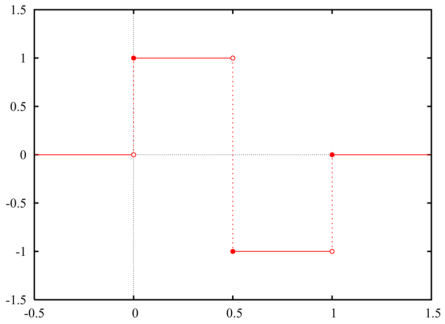

Figure 2.2: Haar wavelet (figure from [2]).

The first, and also the simplest, known mother wavelet is the Haar wavelet (shown in Fig. 2.2) which is often used as the introductory wavelet on studies on this object but also is found, among other applications, in image compression. It has the following formula:

$$\psi(t) = \begin{cases} 1 & ,0 \leq t < \frac{1}{2} \\ -1 & ,\frac{1}{2} \leq t < 1 \\ 0 & elsewhere \end{cases}$$

When transforming a signal with the DWT, after the initial convolution, one obtains detail coefficients, related to high frequencies, and approximation coefficients, related to low frequencies. The latter ones can be sub-sampled and subjected to the same type of convolution, from which point, the process is repeated as much as required, depending on the signal and requirements of the application.

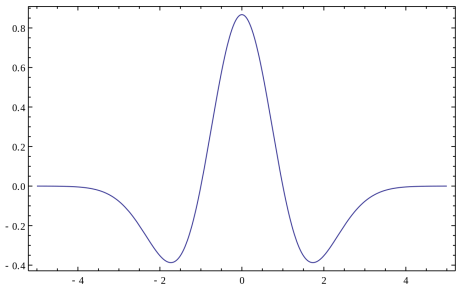

Figure 2.3: Mexican Hat wavelet (figure from [3]).

The Mexican Hat, or Ricker, wavelet (shown in Fig. 2.3) can be used in the CWT as a mother wavelet, which formula is defined as:

$$\psi(t) = \frac{2}{\sqrt{3}}\pi^{-\frac{1}{4}}\left(1-t^2\right)e^{-\frac{t^2}{2}}$$

In the process of a wavelet transform, a mother wavelet is contracted and extended before the convolution with a signal. At the edges of the signal (the start and the end), the contracted/extended wavelet will not be able to overlap completely with the signal and thus, originate boundary effects, which are more pronounced as the scale of the wavelet increases - lower frequencies. While there are different strategies to minimize these effects, the wavelet coefficients at these locations must be taken as inaccurate. The scalogram region that contains these boundary effects is known as Cone Of Influence (COI) and an example can be seen in Figure 2.4.

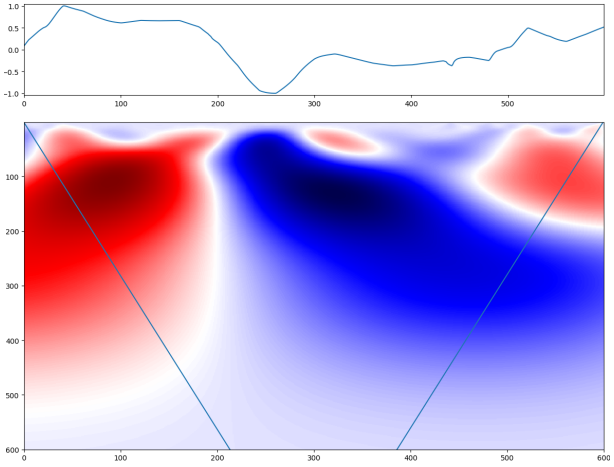

Figure 2.4: Cone Of Influence in a scalogram from a CWT with Mexican Hat as mother wavelet. Values are in the interval [-1, 1] - blue represents negative values and red positive values.

2.2 Imitation Learning from Observations

Imitation Learning (IL) [4], in the Machine Learning context, is an agent's process of learning a policy that executes a task with data previously generated from a different agent exemplifying that same task. Usually, both state and action data over time are provided which could be the joint angles of a robot and its torque commands, respectively.

Generally, there are two main classes of IL: behavior cloning and inverse reinforcement learning. In behavior cloning, a policy is learned directly, by supervised learning, from another policy. In inverse reinforcement learning, the process alternates between finding a hidden reward function from the given data and using reinforcement learning (RL) to learn the a policy that imitates the example data task.

For some challenges like the imitation of human movement, there are plenty of sources that only provide state data (e.g. videos of human body activity) and since action data is absent, the previous mentioned methods are of no use. To tackle this problem, the agent needs to learn how to execute a task only from state demonstration information generated by an expert, and here we are in the realm of imitation learning from observations (ILFO). There are several approaches to learn the imitation policy,

which can be organized into model-based and model-free groups, as seen in Fig. 2.5.

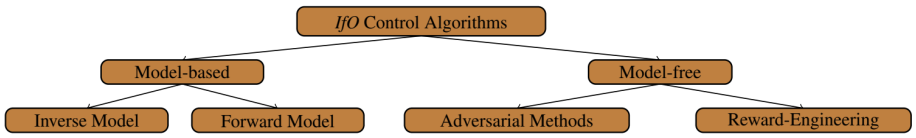

Figure 2.5: A diagram of the different approaches to Imitation Learning from Observations [4].

Model-based group consists on learning a particular type of dynamics model, which can be an inverse dynamics model (a mapping from current and next states (s_t, s_{t+1}) to actions (a_t) or forward dynamics model (a mapping from state and action pairs (s_t, a_t) to the next state (s_{t+1})). With Model-free group, there are no models learned during the training of the imitation policy, and can be separated into adversarial methods and reward engineering.

Adversarial methods are based on generative adversarial imitation learning (GAIL), which itself is based on generative adversarial networks, a technique with proven results in deep learning. Reward engineering consists on reinforcement learning with a handcraft reward function to obtain a imitation policy, an approach used in this work by rewarding the agent to follow closely a sequence of states given by the motion data.

2.3 Proximal policy optimization

Proximal policy optimization (PPO) [15] is based on a Reinforcement Learning family of algorithms denominated by Actor-Critic, which consists on joining two different RL approaches - value based and policy based [5]. The Actor represents the policy based approach and has the objective of learning the optimal policy, outputting the best action, given the state of the environment. The Critic represents the value based approach and has the objective of learning the value function which determines the quality of the state reached by following the action chosen by the Actor. Both Actor and Critic learning is driven by the temporal difference (TD) error given by the Critic (seen in Fig. 2.6).

Exploring further this idea, Mnih et al. work [16] represents a relevant standing in this area with the Asynchronous advantage actor-critic (A3C). Here, the Critic is improved by focusing on the advantage function, which compares the expected future reward of taking the chosen action in that state with the value of that state, realizing the unexpected benefits of said action. Additionally, an important change to the learning process was made by having multiple agents train asynchronously (as seen in Fig. 2.7), in their own independent environment, sharing later the knowledge learned.

Continuing on this line of progress, proximal policy optimization [15] is a recently developed algorithm that brought some of the benefits of trust region policy optimization (TRPO, a popular policy search algorithm) such as data efficiency and reliability, while being much simpler to implement and more general.

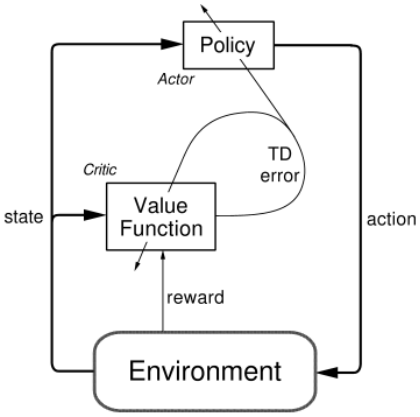

Figure 2.6: Actor-Critic model (figure from [5]).

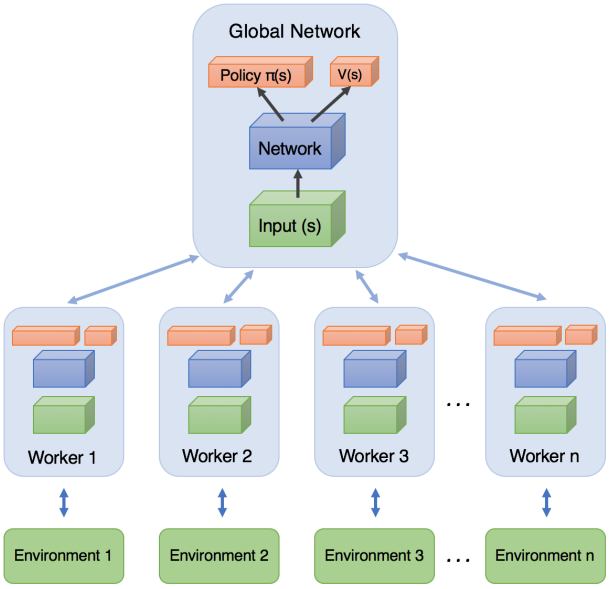

Figure 2.7: Asynchronous advantage actor-critic (A3C) training model (figure from [6]).

It introduces an imposed limit to the amount of policy change in each step to avoid instability during learning, something common for policy gradient algorithms.

The two main contributions are the Clipped Surrogate Objective and the use of multiple epochs of stochastic gradient descent (SGD) in each policy update. The new main objective, to be maximized each iteration, is given by the following formula:

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

where r_t is the probability ratio between the new policy and the old policy, ϵ is a hyperparameter (usually 0.2), $\hat{A}_t$ is an estimator of the advantage function at timestep t and, finally, the function $clip(arg1, arg2, arg3)$

which limits $arg1$ to the values between $arg2$ and $arg3$.

Moreover, the advantage estimator is given by:

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + ... + (\gamma\lambda)^{T-t+1}\delta_{T-1},$$

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

where t is the time index in the interval between 0 and T on a given trajectory segment with length of T, $V(s_t)$ is the learned value function, γ is the discount factor and λ is the generalized advantage estimation (GAE) parameter.

Algorithm 1 PPO, Actor-Critic Style

for iteration=$1, 2, \ldots$ **do**
  for actor=$1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

Figure 2.8: Proximal policy optimization algorithm

The algorithm can be seen in the Fig. 2.8. It is inspired by Advantage Actor Critic (A3C), in that it uses several parallel agents to explore and test the environment, improving the learning time. This was possible due to the use of Clipped Surrogate Objective.

3

Related work

Contents

Here, we present some of the relevant work in two different areas that are pertinent to this work. The Character Locomotion Animation section explores some of the work done in this area and pays particular attention to Deep Mimic, a state-of-the-art approach. Wavelets in Gait Analysis section presents some of the work done that is believed to provide a better understanding of their uses and capabilities in this work's context.

3.1 Character Locomotion Animation

Being an area of intense past and present research, much ground has been laid to this day. In the context of physics-based biped character, a diverse number of techniques are presently available to tackle this challenge.

Due to several recent developments in the field of Machine Learning - particularly in Deep Reinforcement Learning - some approaches have been gathering notoriety due to their impressive results. In [9] a character learns, using the PPO algorithm, a large set of locomotion styles with environment awareness. While functioning and relatively robust locomotions are obtained for different kinds of environments, it offers mostly unpredictable animations, often showing motion artifacts like inefficient gait and unnatural arm movement, which are then difficult to perfect making it not suitable for most contexts.

Muscle-based locomotion [17] provides, to a character, a realistic foundation along with the proper range and reaction of movement of body members. For instance, in [18], the authors developed a generic control method compatible with different bipedal simulated creatures, where no motion data is needed. It consists of the optimization of muscle routing and control parameters resulting in the ability of moving in a target direction at a target speed on irregular ground while being the focus of external perturbation. The resulting locomotions, at the level of lower body motion, where considered by the authors to be close to the state-of-the-art. In terms of limitations, it required quite a lot of work for the manual parametrization of the system and to correctly create a muscle model for a character in order to obtain a correct motion.

A data-driven approach provides direct guidance to an intended locomotion, something that has been researched for some time [19]. A recent work [20] shows promising new capabilities and fidelity in imitating different complex motions. Here, the authors created a system capable of a large set of locomotion skills with unorganized and mostly unlabeled data. It contains a recurrent neural network (RNN) that generates "future" physic simulated motions that are used to evaluate future states and actions in order to improve the policy learning process. The authors point to some limitations such as the long learning time for each motion/skill and the stiffness produced by PD controllers.

Deep Mimic

Deep Mimic [7] is a framework for synthesizing physics-based character animation that fuses two

distinct learning strategies - Imitation Learning from Observation and goal-direct Reinforcement Learning.

It is composed of several pre-existing components from different sources combined in such a way that, for a character model, it produces robust animations that closely follow the given motion data while able to achieve a goal, defined by a given reward function.

The system handles motion data from two different sources, motion capture (mocap) and keyframes. Mocap data, which has a duration between 0.5 and 5 seconds, is obtained from Carnegie Mellon University[1] and Simon Fraser University[2] Motion Capture Databases while keyframes data comes from artist-authored animations. The data was manually processed and retargeted to the different characters.

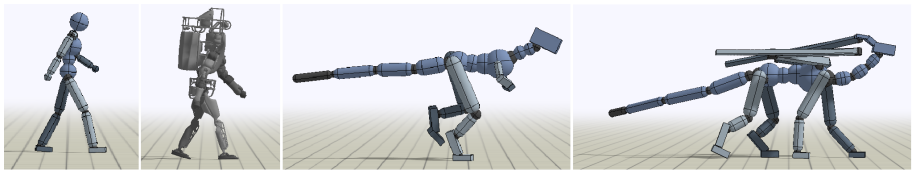

Figure 3.1: From left to right: humanoid, Atlas robot, T-Rex, dragon.

Several models were included in their work, as seen in Fig. 3.1. Comparing the first two models, they differ mostly only in their body mass distribution and actuators, both being animated with mocap data. The last two have little resemblance among themselves and with the ones already mentioned and their motion data source was keyframed animation. Each character's model is a particular setup of rigid bodies, where each link (member) is connected to its parent link with a 1DOF (degree of freedom) revolute joint for knees and elbows and 3DOF spherical joint for the rest. PD controllers [21] were placed at each joint with handcraft gains.

Following the authors's convention, the motion data is a sequence of target poses $\{\hat{q}_t\}$ and the control policy $\pi(a_t|s_t, g_t)$ is the mapping of the character's state s_t and goal g_t to an action a_t.

The state s represents the current configuration of the character's body, defined by several properties for each joint - position (x,y,z), quaternion rotation (w,x,y,z), linear and angular velocities (both x,y,z). Properties are computed in the local coordinate frame of the character, being the root (pelvis) at the origin and the facing direction being along the x-axis. There is also a phase variable ϕ due to the fact that the motion data's target poses change with time. Its value sits between 0 and 1, where 0 indicates the start and 1 the end of the motion where, in the case of a cyclic motion like walking and running, ϕ is reset to 0. An instance of the state vector starts with the phase variable followed by the root y value, then a sequence of position and rotation of each joint for all 15 joints, and ending with a sequence of the linear and angular velocities of each joint for all joints, totaling 197 variables.

[1] http://mocap.cs.cmu.edu
[2] http://mocap.cs.sfu.ca

The action contains a set of target angles that are transformed by proportional-derivative (PD) controllers into torques, which are applied to the model's joints. Targets for spherical joints and revolute joints are represented as axis-angle form and scalar rotation angles respectively.

The policy is materialized as a neural network (see Fig. 3.2), trained using PPO and queried at 30 Hz. In some scenarios, a heightmap H is used to provide visual information of the surrounding terrain/obstacles, but in the cases where such information is not necessary, the network is just composed by the layers 5 through 7. In the simplest cases where it's used, the heightmap is a 1D heightfield containing 100 samples along 10 meters but for more complex scenarios, it's a 32 x 32 sampled square region of 3.5 meters per side centered, in both cases, on the character.

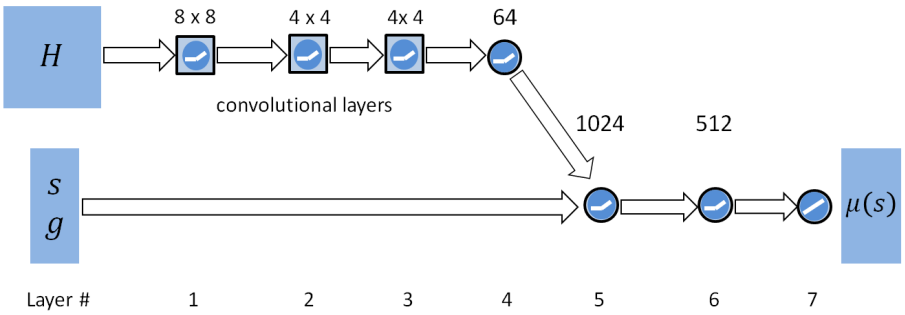

Figure 3.2: Illustration of the policy neural network. A heightmap H is directed through a sequence of 3 convolutional layers - a layer of 16 8x8 kernels and two layers of 32 4x4 kernels. Afterwards, a fully-connected layer of 64 units processes the obtained feature maps and concatenates with the state s and the goal g, serving as input for two fully connected layers, with 1024 and 512 units, and finally a layer of linear units producing the output $\mu(s)$. All hidden units use ReLU for activations.

From motion data, the system computes an imitation reward $r^I(s_t, a_t)$ and the goal defines a reward $r^G(s_t, a_t, g_t)$ for fulfilling a task. So for each timestep t, the total reward r_t is given by the expression:

$$r_t = w^I r_t^I + w^G r_t^G$$

Where w^I and w^G correspond to the weights of the imitation reward and goal reward, respectively. The authors used the following values: $w^I = 0.7$ and $w^G = 0.3$.

In Fig. 3.3 we can see an example of the importance of using both rewards, in order to complete a task, instead of using just the goal reward. On top, the character finds an accurate policy in terms of completing the task but the motion is unnatural, while on the bottom, both the movement and the goal are as expected. In the case of using just the imitation reward (not shown in the Fig. 3.3), the problem would be the lack of aim of the character. In situations where there is only the need to imitate the reference motion, the goal reward is left unspecified.

In respect to a goal, encoded in a reward function r_t^G, several examples were given. With the Target

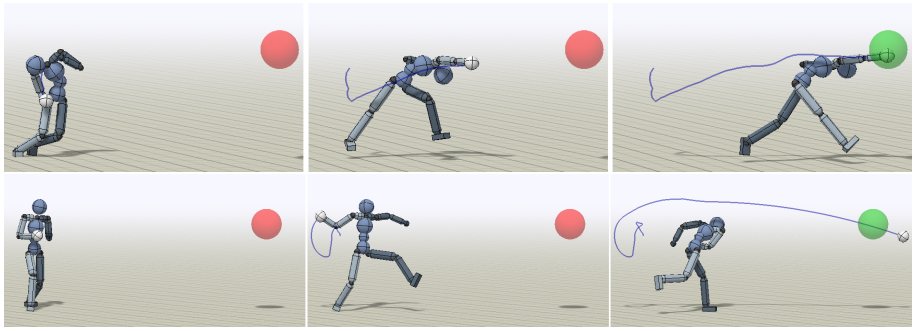

Figure 3.3: Throwing the ball without reference motion (top) and with reference motion (bottom).

Heading goal, it becomes possible to steer the character while executing a walking/running motion:

$$r_t^G = exponential\left[-2.5\ max(0, v^* - v_t^T d_t^*)^2\right]$$

where v^* is the intended velocity along a direction d_t^* (2D unit vector on the horizontal plane) and v_t is the character's center-of-mass velocity. The policy is provided with the goal $g_t = d_t^*$ and, while training, this direction is randomly changed multiple times in each episode, giving the ability, in runtime, of manually steering the character by selecting the direction value.

With the Strike goal, the character can hit a spherical target, within 0.2 meters of target's position, on a random position using the feet or others specific links:

$$r_t^G = \begin{cases} 1 & \text{, target was hit} \\ exponential\left[-4||p_t^{tar} - p_t^e||^2\right] & \text{, otherwise} \end{cases}$$

where p_t^{tar} is the target's location and p_t^e is the position of the specified link to hit the target. Some limits were imposed on the variability of the target distance (between 0.6 and 0.8 meters), height between 0.8 and 1.25 meters) and relative direction (-+2 rad). The policy receives as input the goal $g_t = (p_t^{tar}, h)$, in which h is a binary variable that informs if the target was hit in a previous timestep.

With the Throw goal, shown in the Fig. 3.3, the character learns to throw a ball to a target. Initially, the ball is connected to the hand of the character with a spherical joint, which is then released at a fixed time point. Both the g_t and r_t^G are the same for the Strike goal though with a difference in the character state s_t, given the inclusion of some of the ball's properties - position, rotation, linear and angular velocity. The target can vary between a distance of 2.5 to 3.5 meters, a height between 1 to 1.25 meters and a direction between 0.7 to 0.9 radians.

With Terrain Traversal goal, the character is able to travel across a path filled with obstacles. Both the g_t and r_t^G are analogous to the ones in the Target Heading goal except the heading is fixed in the forward direction. There are 4 different obstacle courses containing distant floor gaps, narrow irregular

floor, irregular stairs and lastly a mix of variable floor gaps and heights. It's relevant to notice that here, the authors employed Curriculum Learning [22] in order to achieve a faster training. Specifically, the neural networks without the heightmaps, are trained with the intended motion on flat terrain, then these are augmented with the heightmap and the respective convolutional layers and finally trained on a specific terrain.

In the process of learning a reference motion, the work emphasizes two concepts: reference state initialization (RSI) and early termination (ET). RSI corresponds to the strategy of varying the initial state of the motion and learning the sequence after this point, something fundamental for complex motions like a backflip. The author's argument is twofold: learning a future phase of the motion can be a requirement for a high reward in a previous phase and the policy needs to visit a state for it to take the (possibly high) reward of that state into consideration. ET consists of terminating a learning episode as soon as an undesirable state is encountered. It's another way of complementing the reward function and avoid having the network learn how to recover from states very different from the intended motion. In the case of walking or running, ET is triggered when certain parts of the character's model touch the ground or fall below a height threshold.

The use of these strategies is usually necessary for obtaining a successful policy and, in any case, the learning time improved significantly, though often two days were required for each motion to be learned on a 8-core CPU - GPU acceleration was not used.

Besides the ability to learn a particular reference motion (check Annex section for set of examples), Deep Mimic also offers alternative ways to combine several different motions. With Multi-Clip Reward, various reference motions are used during training and the final imitation reward is modified to accommodate this change by selecting the maximum reward value among the set of the individual motion imitation rewards. A different option is Skill Selector, where the user is given the control to choose which motion to execute at any point in time. The policy learns a set of different motions and the association between an individual motion and the command to execute it, given by a one-hot vector. In this case, there is no additional goal reward function r_t^G; also, during training, for each learning episode, the reference motion is selected at random and so the policy will learn the transition between all the motions. Finally, in Composite Policy, instead of one, there are multiple policies being trained with different reference motions and later combined together. The value function corresponding to each policy estimates the value of a state on that policy, thus the set of all value functions can be used to find the most appropriate policy for any state.

In the discussion of their work, the authors point to the need of addressing some current limitations. One to notice is a phase variable (synchronized with the reference motion) required by the policies that reduces their capability to adapt the timing of the motion, which once solved, could allow for more flexible and realistic recoveries.

In terms of implementation, the framework has 2 parts - DeepMimicCore is responsible for low level tasks such as interfacing with the physics engine, PD controllers and rendering while DeepMimic is responsible for high level tasks that allow the learning of imitation policies to take place. Even though the high level part is named the same as the complete system, and unless stated otherwise, any DeepMimic mention relates to the full framework.

3.2 Wavelets in Gait Analysis

Human gait analysis, the study of locomotion, is performed taking in consideration the characteristics of the data that supports it - low frequency shape and high frequency discontinuities [23].

In the mentioned work, the authors, having the objective of classifying human motion with data obtained by motion capture and accelerometers, presented Wavelets as a possible tool. Specifically, histograms of the wavelet coefficients, at every frequency band, were processed by a Support Vector Machine (SVM) classifier. In human motion data, variations in the low frequency are represented more intensely than the temporal discontinuities in the high frequency, which results in the latter having the respective wavelet coefficients with lower magnitude. This is a problem when using a histogram adapted to low frequency, since most high frequency will end up in the zero bin. To deal with it, the authors scaled down the wavelet coefficients by a constant (4), containing most in the interval -1 to 1. For each motion, a vector of fixed size was built with the result of concatenating histograms of different dimensions of the signals and then trained on a SVM classifier. This resulted in the ability of distinguishing between walking at different rhythms (slow, normal fast).

For human muscle activity signals, wavelet analysis has also been extensively used for diagnostics in a clinical context [24]. Moreover, the differentiation of gaits from different people, is a useful feature in several different applications; for instance, the detection of early stages of Parkinson's disease [11].

Given the variability of options in terms of Wavelet analysis characteristics, it's important to find the appropriate ones for the task at hand and some studies have tried to shed some light on this, particularly for scale [25] and Mother wavelet [12]. The later work serves the purpose of investigating, in the context of detecting gait events using Continuous Wavelet Transform (CWT), the differences in performance of different mother wavelets for both hemiplegic and healthy individuals. The authors argue that being able to detect gait events is essential for several applications in the Human healthcare area like control mechanisms in drop foot correction devices, recognizing human activity and aiding the decision on rehabilitation strategies. When walking, two gait events - heel strike (HS) and toe off (TO) - are commonly regarded as the most relevant ones in a normal gait cycle, providing swing, stance and stride parameters information. Hence, these were the gait events with this type of locomotion that this work focused on. The source of the data for this study was provided by 16 individuals (3 of them were hemiplegic patients)

using a wireless tri-axial accelerometer device on one of the lower legs, just below the knee.

As explained by the authors, the walking cycle can be divided into a double repetition of a sequence composed of a stance and a swing phase, which beginnings are indicated by HS and TO gait events, respectively. These events can be detected using CWT to process gait data, given the time-frequency connection between gait event and gait cycle, proved by previous studies that showed the effectiveness and stability of the CWT, even when subjected to disturbances.

In order to find the most appropriate mother wavelet for gait event detection, the authors of this work first constructed a general CWT algorithm.

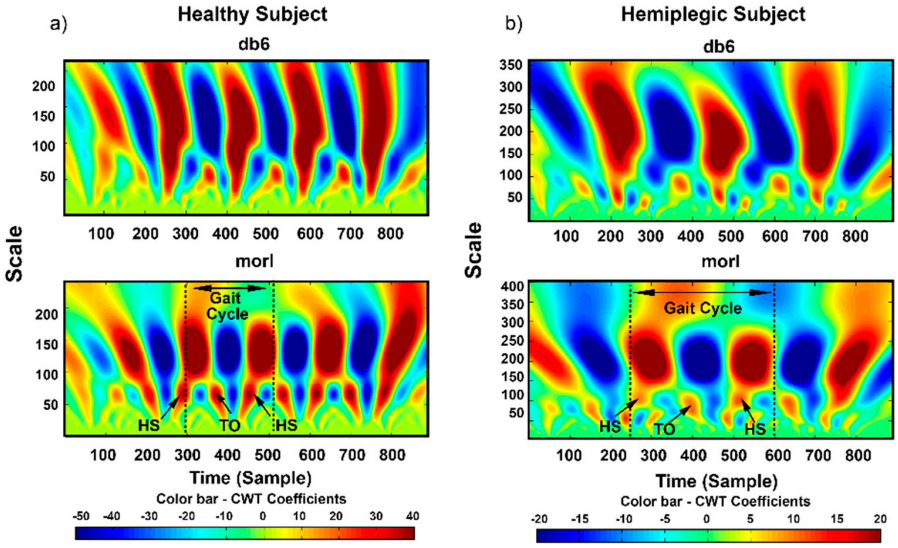

Figure 3.4: Scalograms of walking acceleration data processed by CWT. Two mother wavelets (”db6” and ”morl”) are shown side by side comparing the gait cycle of a healthy individual to a hemiplegic patient.

Wavelet Family	Order N	Orthogonality	Symmetry	Explicit Expression
Haar	db 1	Orthogonal	Symmetric	$\psi_{Haar}(t) = \begin{cases} 1 & 0 \le t < 1/2 \\ -1 & 1/2 \le t < 1 \\ 0 & otherwise \end{cases}$
Daubechies	db 2–10	Orthogonal	Asymmetric	No
Coiflets	coif 1–5	Orthogonal	Near symmetric	No
Symlets	sym 2–8	Orthogonal	Near symmetric	No
Gaussian	gaus 1–8	No	Symmetric	$\psi_{Gaussian}(t) = -\frac{1}{\sqrt{2\pi}} t e^{-\frac{t^2}{2}}$
Morlet	morl	No	Symmetric	$\psi_{Morlet}(t) = \frac{1}{\sqrt[4]{\pi}} e^{j2\pi f_c t} e^{-\frac{t^2}{2}}$
Meyer	meyr	No	Symmetric	$\psi_{Meyer}(f) = \begin{cases} \sqrt{2\pi} e^{i\pi f} \sin\left[\frac{\pi}{2} v\left(3\lvert f\rvert - 1\right)\right] & \frac{1}{3} \le \lvert f\rvert \le \frac{2}{3} \\ \sqrt{2\pi} e^{i\pi f} \cos\left[\frac{\pi}{2} v\left(\frac{3}{2}\lvert f\rvert - 1\right)\right] & \frac{2}{3} \le \lvert f\rvert \le \frac{4}{3} \\ 0 & \lvert f\rvert \notin \langle \frac{1}{3}, \frac{4}{3} \rangle \end{cases}$

N is the order of the mother wavelet. ψ is the mother wavelet function if the wavelet has the explicit expression. f_c is the wavelet central frequency, t denotes the time, and f denotes the frequency.

Figure 3.5: Summary of the investigated mother wavelets.

After developing the algorithm, the performance of 32 mother wavelets in detecting gait events was analyzed. The mother wavelets under consideration, as shown in Fig. 3.5, are as follows: ten

Daubechies (“db1” to “db10”), five Coinflets (“coif1” to “coif5”), seven Symlets (“sym2” to “sym8”), eight Gaussians (“gaus1” to “gaus8”), Morlet (morl) and Meyer (meyr). Some core properties of a mother wavelet are generally considered relevant in the selection process. The Orthogonality allows the assumption that the decomposition of the signal will avoid the overlapping of sub-frequency bands. The Symmetry contributes to the avoidance of phase distortion. Though the most deciding factor in the selection of a mother wavelet is recommended to be the performance measured both in terms of accuracy for the specific application. Time-error and F1-score - and quantitative criteria - cross-correlation coefficients (Xcorr), energy-to-Shannon entropy ratio (ESER) were investigated following that reasoning.

Particularly, a mother wavelet having the minimum time-error and maximums F1-score, Xcorr and ESER on the HS and TO events are the desired properties that are believed to indicate the most appropriate one. In terms of the accuracy criteria, the results shows “db6” to be among the group that shows lower average time-error, both for healthy and hemiplegic individuals, and “db5” and “db6” the ones with the best result for F1-scores. For the quantitative criteria, the results were quite varied and offer no clear answer. Moreover, upon further investigation of this criteria, it was found that it does not offer a proper indication for the ability of a mother wavelet to help in the detection of HS and TO gait events.

The authors conclude that “db6” sets itself apart in this application for the most appropriate mother wavelet to allow a better detection of the intended gait events.

This work, along with others, gave us confidence on the applicability of the wavelet transform for our case and contributed to our understanding of the problematic of choosing the best mother wavelet for our goals even though the result cannot be transposed directly to our work given the multitude of constraints that are present in our context - mainly processing requirements and a bigger variation of signals (full simulated body joint data with different locomotions). One question that arose here is the possibility of reducing, for each joint, the amount of properties to be considered.

4

Towards a Model for Wavelet Augmented ILFO

Contents

In this section we present a Model for Wavelet Augmented Imitation Learning from Observations (WAILFO) in both its broad and detailed view. Additionally, as the development proceeded in an iterative way, the details and the respective evaluation are presented in a sequential order. Finishing the chapter, there's a discussion of the results.

4.1 Overview

DeepMimic [7] applies Imitation Learning from Observations to obtain similar animations as those shown by the given motion data while under the constraints of a physics-based simulation. With the objective of developing a system capable of imitating, robustly, motion data of a running animation, with the same constraints, we based our system on DeepMimic, a fully functioning system.

The gist of our work is the integration of the Continuous Wavelet Transform coefficients, obtained from State data, into the system. By making use of the structure already in place, capable of finding good policies, the implementation complexity is reduced and the chances of success grow. Thus, the existing neural networks are the target of the set of modules developed/used in this work.

The added modules are:

- a Memory Buffer, to save State data;
- a CWT, to transform several instances of the State data;
- a CNN, that outputs scalogram features.

These are connected by the following pipeline: a sequence of DeepMimic's State variables are saved in the Memory Buffer, which feeds into a Continuous Wavelet Transform (CWT) that, for each state variable, returns the respective coefficient matrix. Each coefficient matrix, like an image, is the input of a Convolutional Neural Network (CNN) that outputs high level scalogram features to DeepMimic's existing network, alongside the State Vector. The complete network can be seen in Figure 4.1.

It was expected that the new additions would represent a significant computational weight to the CPU on an already demanding load - state variables may need to be acquired more frequently, i.e. among policy updates; then stored/disposed in a first-in first-out (FIFO) buffer; the history of each variable is transformed by the CWT into a scalogram; all the scalograms are processed by the CNN. Of notice is the ability of DeepMimic to train with several parallel agents using different CPU threads. Since the state is unique to each agent, all the calculations mentioned previously have to be performed in each thread.

The humanoid model is used to provide the embodiment for the system in the environment whose state information (this model's joints and phase variable) are fed to a network of 2 fully connected layers. By using the PPO algorithm, 2 similar networks are created in order to find the best policy - one used as actor, outputting the target angles for the PD controllers of the joints of the humanoid model, and

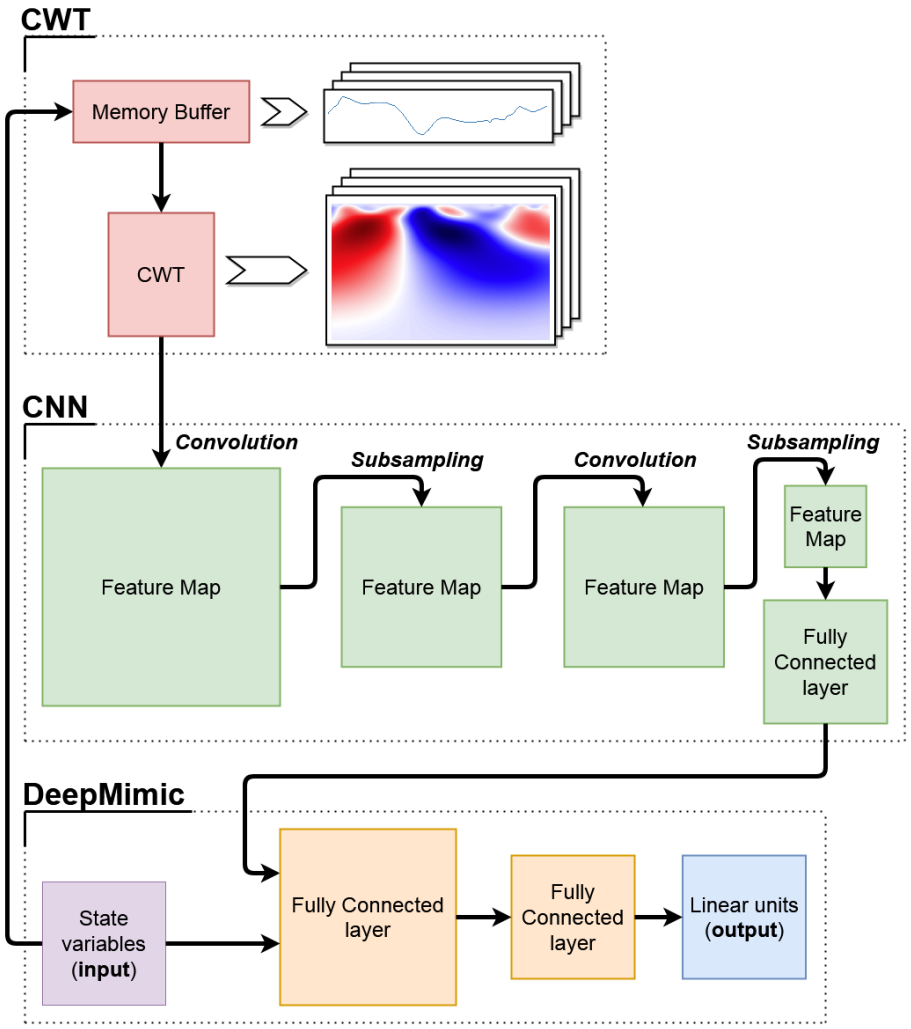

Figure 4.1: Architecture of the policy network of the proposed WAILFO model. The modules of the new pipeline are in red (Memory Buffer and CWT) and green (CNN). Its output, alongside state variables, provide the input for the sequence of two fully connected layers. The final layer contains linear units that outputs target angles for the PD controllers of the joints. In the scalogram, which shows the CWT coefficients of the signal above it, the red color represents positive coefficients, blue color represents negative coefficients and white represents values close to zero.

one used as critic, returning the believed value of the current state. While being part of DeepMimic's capabilities, the goal input is ignored for this work alongside the ability to add terrain height information, since only regular terrain is used. The running animation, which is the target of the policy, is the one provided by mocap data already included in DeepMimic. In terms of Wavelet Transform type, the choices depend on the application. For our case - the need to analyze state signals - Continuous Wavelet Transform was chosen. This had the predicted drawback of being more computational intensive.

Given this and in order to implement the WAILFO model, an iterative approach was used: ablation studies were performed to understand the impact of DeepMimic's properties/capabilities and relevant DeepMimic modifications had to be made to accommodate the new modules, following is a study on the CWT process that allowed the development of a better intuition of the topic and ways to be more performant in this area and lastly, the development of the CNN.

4.2 DeepMimic modifications

The calculations for both the CWT and CNN were expected to be expensive on the CPU, thus it would be helpful to find ways to lighten the CPU load. Since DeepMimic supports motions more complex than running, it was likely that the reduction of some of its capabilities would bring better performance without losing the ability to learn the intended motion. With ablation studies, several approaches were investigated - it's relevant to underline that the results and conclusions cannot be extrapolated to other supported motions, which can be very different and more complex.

4.2.1 State array tests

When using the humanoid character model, in a State query (DeepMimic's high level module calls a method in DeepMimicCore) there are 197 variables returned in an array; of those, 195 contains the 13 different properties of the 15 joints, namely position, angle, linear and angular velocities; all joints have their property values in relation to the root joint, except for the latter.

The benefits of reducing the number of variables are twofold. Firstly, the State query happens faster, particularly significant when there are dozens of possible new queries between two policy updates, and there are several millions steps in a complete training session. Secondly, having fewer variables allows for a faster neural network process. Moreover, at this point in time and with commercial off-the-shelf hardware, it's not reasonable to expect a real-time usage of the final system when all of the 197 signals are to be processed, thus a study on the contribution to the training of the State variables becomes even more valuable. Starting by testing the contribution of each state property type, we substituted every other properties values with zeros (with the exception of the phase variable, which was always present).

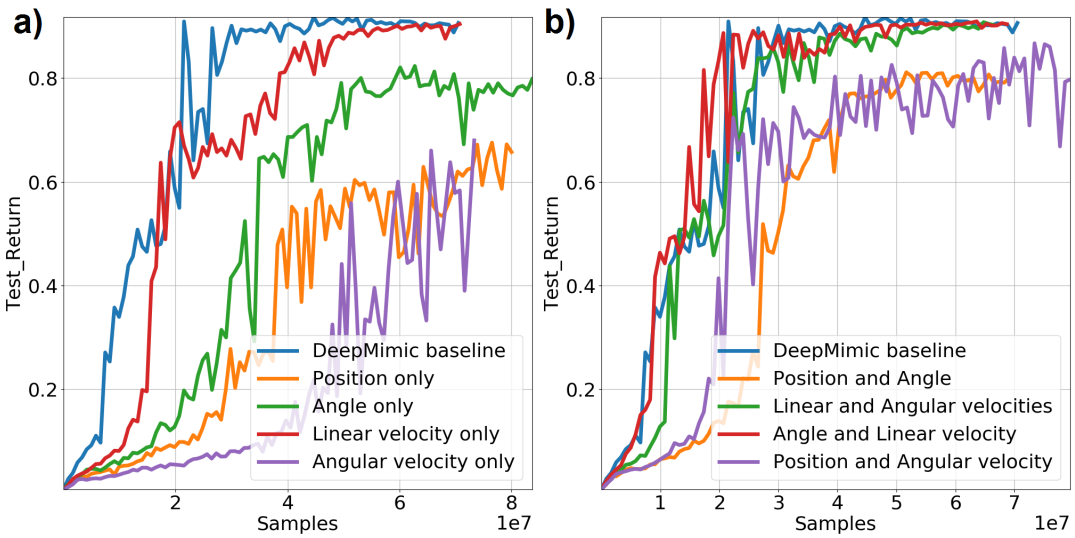

Figure 4.2: Performance comparisons between single State array properties **a)** and combinations of two State array properties **b)**.

As the Figure 4.4 **a)** shows, with only position or angular velocity variables, the training does not result in a competent policy while with angle or with linear velocity the inverse happens, being the latter close to the use of all variables. In Figure 4.4 **b)**, with this reduced set of possible combinations, it's already conclusive that position properties do not offer much benefit and linear velocity in conjunction with angle (or angular velocity) variables results in a policy with performance as good, or even better, as the full array of variables.

4.2.2 Initial changes

As the previous study shows, it was possible (when learning the running motion) to remove some properties from the state array and maintain an acceptable performance. With the intention of, in the future, using only the CWT data as the input for learning (from the signals contained in the mocap file), we decided to reduce to a minimum the state array information. The reduced state was obtained by reimplementing the method, in DeepMimicCore, that returns the State vector (*RecordState*) and the corresponding method that returned its size (*GetStateSize*) - the reduced state contains the phase variable (1D), root linear velocity (3D), and the rotations from the right (4D) and left (4D) ankles, root (4D), right hip (4D), right knee (1D), right shoulder (4D), right elbow (1D), left hip (4D), left knee (1D), left shoulder (4D) and left elbow (1D) for a total of 36 variables.

Additionally, based on empirical tests, and in order to lower the computational efforts, we changed DeepMimic's first existing layer by reducing it by half (from 1024 to 512).

4.2.3 Replay buffer size

The replay buffer keeps different information in memory in order for the system to learn from past experiences (by random sampling from it). It includes N instances of the State array, N related goals (when used), N actions, N rewards, where N is a parameter with the default value of 500k. Since the CWT module is external to the CNN module, the CWT coefficients (scalograms) of the chosen signals must be saved, side by side with the corresponding State array in order to avoid expensive redundant CWT calculations. This introduces a memory restriction to the size of each scalogram and to the number of scalograms (channels). As an alternative, we could have added the calculations of the CWT to the CNN - the CNN block would receive an array of state variables and preprocess it with the CWT before the first CNN convolution. This wouldn't require any more memory, since the same State array would feed both the CNN and the first layer of DeepMimic's networks. Unfortunately, the redundant CWT processing would greatly delay the training.

Due to the new memory restrictions, there was a need to understand the effect of reducing the replay buffer size. The default size of 500k is quite large so a significant reduction was tested.

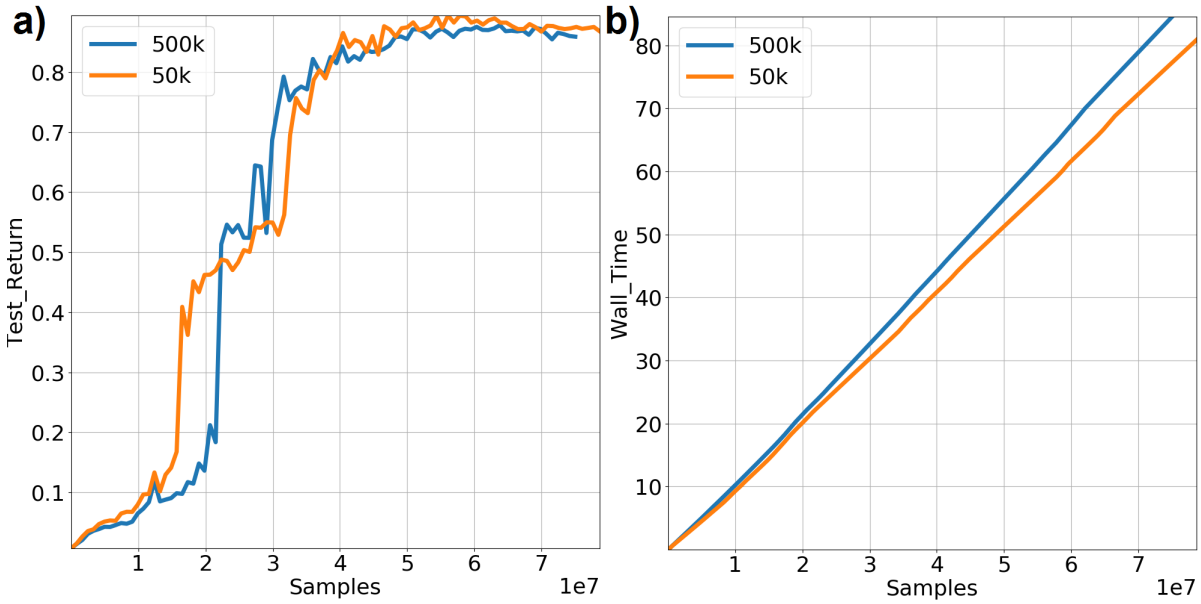

Figure 4.3: Performance comparison **a)** and time comparison **b)** between replay buffer sizes. *Wall_Time* in hours.

Figure 4.3 shows that, in this context, reducing by 10 times the size of the replay buffer still allows for a correct policy to be learned and even better performance, with a small reduction of training time. Unfortunately, when testing the resilience to external random forces, the character shows a lack of robustness in its movement suggesting that it had some catastrophic forgetting on how to recover from less stable positions. We concluded that the size of the replay buffer could be reduced in order to allow the saving of CWT scalograms in the replay buffer, with the added bonus of some gain in performance

and reduced training time, so the value of 100k was chosen - in order to avoid catastrophic forgetting.

4.2.4 Policy update frequency tests

Policy update is the process by which the system reads the state of the character and outputs the target angles. By default, its value is 30Hz - meaning there is an interval, on average, of 33,3 milliseconds between each set of actions. Since several different motions are provided and supported by the system, a more complex motion may require a higher value than the task of running, reducing/increasing the default value may be possible and beneficial for the intended motion while not advisable for others. Different values were tried in order to understand how it affected learning and to possibly find a better value. To note that the policy update frequency used when training the policy should be the same as when executing the policy.

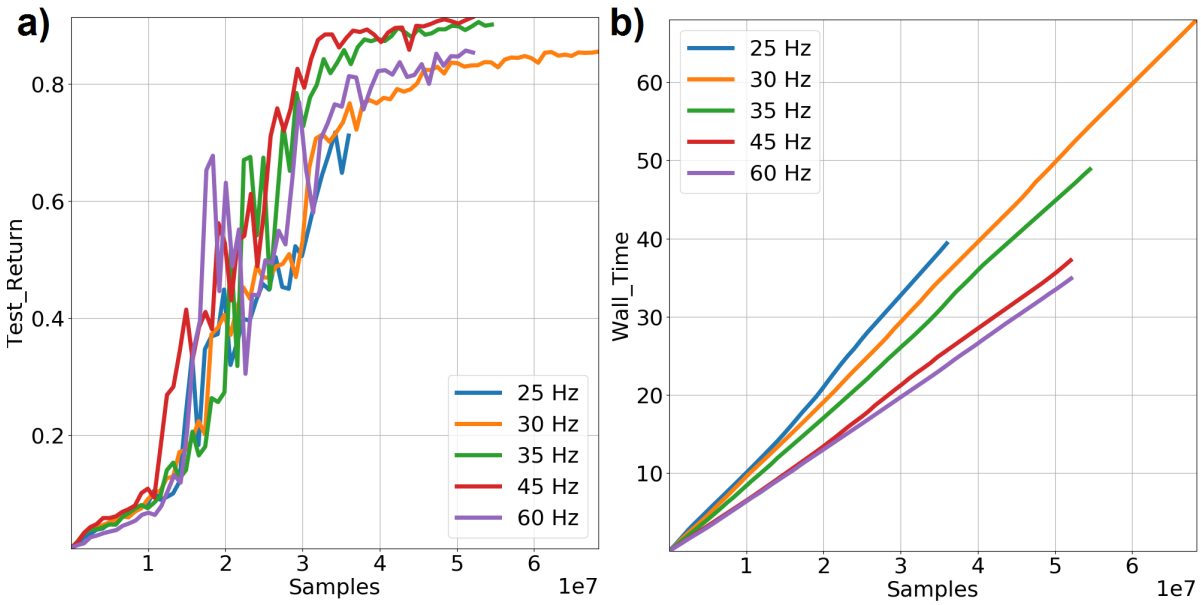

Figure 4.4: Performance comparison **a)** and time comparison **b)** between policy update frequencies. *Wall_Time* in hours.

This comparison shows, among the tested values, that 45 Hz is the most beneficial frequency value for the running motion - the performance is on par with even the default system (State array with complete variables) while taking considerable less time to train. We understood that, with a frequency too small, the learned policy isn't able to react in time and correctly imitate the motion - the CPU spends more time the physics simulation; with a value too large, the training is overwhelmed with very similar inputs and the policy ends up with a meager performance.

In our case, an enormous amount of additional computations were to be added (detailed in the next sections) in each policy update, with influence in training time and, possibly, during runtime, so it was

decided that the frequency value would remain at 30 Hz.

4.2.5 Baby Walker

As the new system required a continuous sequence of state data, a possible problem came to our attention. DeepMimic uses a strategy called early termination (ET) - stopping the learning episode when the character falls on the ground. While the author of the original work argued its usefulness, with the new modules added, it may now represent an impassable obstacle: initially during training, the character falls immediately on the ground not giving enough time to fill the memory buffer with valid data. To overcome this, a new strategy, named Baby Walker, was introduced that allowed the character to remain in a straight position and floating in the air, by applying forces to the root joint, as seen in Figure 4.5.

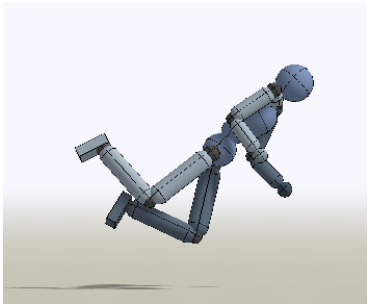

Figure 4.5: Baby Walker supports the character in the air by the application of forces to the root joint (pelvis).

During the training of a policy, the system starts with Baby Walker but stops using it after a performance threshold is surpassed - 0.8 on an interval [0,1]. From Figure 4.6, we can conclude that not only this strategy is not a detriment to the performance of the default system, it may offer a performance advantage.

4.3 CWT Module

Initially, we wanted to develop a better understanding of the results of the CWT applied to the state signals. The PyCWT library [26] was used due to its performance, parametrization and the ability to show the Cone of Influence (COI).

As explained before, the COI represents a region in the scalogram with boundary effects - the result of the signal's transformation with a segment of the resized mother wavelet, and thus represents inaccurate information. At this point there wasn't any indication that this could interfere with the training and so, it was decided that this region would remain in the scalogram feeded to the CNN.

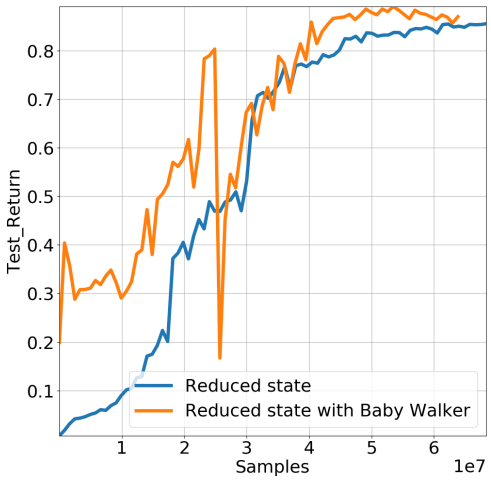

Figure 4.6: Performance comparison between Baby Walker ON (orange) and OFF (blue). When using Baby Walker, the support is disabled when the performance reaches 0.8.

Throughout this work, the scalograms are shown with lower scales (higher frequencies) at the top and higher scales (lower frequencies) at the bottom - this implies that the quickest signal changes will be represented by the lowest scale.

4.3.1 Memory Buffer

The Memory Buffer is the structure that holds the State's values from the different joints. It's implemented as a list of First In First Out (FIFO) lists and receives 2 main initial parameters: an array with the names of the signals - that indirectly informs about the number of channels (signals) - and the Buffer length. Initially filled with zeros, the Memory Buffer will keep all values until full, at which point it discards the oldest saved set of values when a new set of values arrive. During training, when the current episode ends, either by reaching the maximum episode time or when the character falls on the floor, the Buffer is reset - filled with zeros - in order for it to contain only valid data on each episode.

4.3.2 Mother wavelets

There are several different mother wavelets to use with a CWT, which may be more or less appropriate, depending on the signal to analyze. Two common choices are Morlet and Mexican Hat wavelets. In the Figure 4.7, the signal (with a time window of 0.5 seconds) from DeepMimic, obtained from a trained policy, is transformed into scalograms by a CWT with Mexican Hat as the mother wavelet and by a CNN with Morlet.

The scalogram on the left shows simple visual features and a clear localization of where, in time, the big changes of the signal are, compared with the one on the right. This induced us to choose the

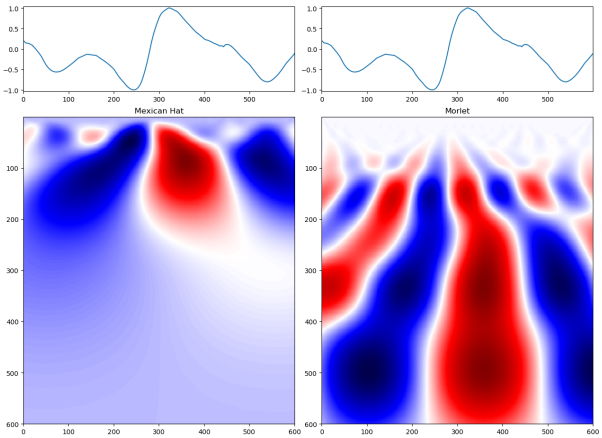

Figure 4.7: Comparison of mother wavelets. Scalograms obtained from a CWT with Mexican Hat on the left and with Morlet on the right.

Mexican Hat as mother wavelet.

It's important to underline that the colors in the scalograms, throughout this work, exist to allow a better understanding of its content. These scalograms could be seen as a grayscale image since they only have one channel of information. All their values are in the interval [-1, 1] and, in the images, the blue and red colors represent negative and positive values, respectively.

4.3.3 Linear and logarithmic scales

A common way of visualizing scalograms is using a logarithmic scale instead of a linear one. In that case, the lower scales will be overrepresented in the overall scale distribution, which represents additional high frequency details. This can be observed in the Figure 4.8.

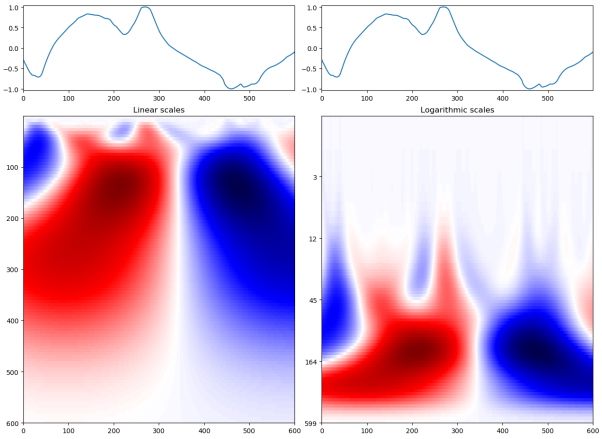

Figure 4.8: Comparison of scale distributions. On the left, the scalogram with linear scale distributions and on the right the scalogram with logarithmic scale distribution.

With a logarithmic scale, the lower scales - higher frequencies - are more detailed while the higher

scales are more compact. To note that, because higher scales (lower frequencies) are more expensive to compute - fewer none-zero calculations from the edges of the mother wavelet - the CWT of a logarithmic scale will take less time to compute. The logarithmic scale distribution is an option in cases where there is a sampling procedure. For our use case, it was not clear, at this point, the need to undersample the signal thus, the linear scale distribution was chosen.

4.3.4 Module Details

There are several key parameters that help define the obtained scalograms. As previously detailed, the mother wavelet was set to be ”Mexican Hat” and the scales had a linear distribution.

Another important aspect is the signal's time window, the period of time from which the CWT coefficients will be calculated. A time window too small wouldn't allow the system to gather relevant frequency information. A time window too big and the computational requirements increase or the detail of the frequencies is drastically reduced. A time window of 0.5 seconds was chosen in order to strike a balance between the both cases - this choice took into consideration that the running motion used, which is included in DeepMimic, has a cycle of 0.8 seconds, thus only 2 sequential time windows are needed to observe the whole motion. Since the policy update has a default frequency of 30Hz, there are 15 updates during the defined time window, which results in a scalogram of size 15x15.

Each scalogram requires both memory and computational resources so there are strong constraints in the amount of channels that the system can support. The mocap file contains rotation values of each joint at specific motion timestamps. From these, 2 joints of each arm (shoulder in 4D and elbow in 1D) and each leg (hip in 4D and knee in 1D) were chosen, totaling 20 channels. As a final improvement, with a 100k replay buffer size and 20 channels, it was possible to have a modest increase in the scalogram size - 16x16 - and this reached the limits of the memory constraints.

In terms of pipeline, the CWT process sits between 2 normalization operations, with values in the interval [-1,1]. The first normalization was necessary in order for the scalogram to show relevant features - if the difference between the maximum and minimum of the signal is small compared to the absolute minimum value, the signal's frequency changes won't show in the scalogram. The second normalization is a common procedure when feeding CNNs, in order to align the range of the different signals, which may vary greatly.

4.4 CNN Module

The purpose of a CNN in this work is to learn the most relevant scalogram features for the task at hand and convey this information to existing DeepMimic's neural networks in order for the system to perform better, in terms of imitating the running animation and be more resilient to external forces. The

chosen approach consisted of adding a CNN that could learn how to identify characters from images, and integrate it within the system. Since we are using the CPU for all the calculations (instead of the GPU), a network with reduced complexity was advisable toward keeping the number of computations low. For this purpose, the CNN structure from [27], capable of learning how to identify MNIST digits [28] was adapted and used.

In order to adapt CNN's structure to better fit the current problem, an intermediary step was taken. As a standalone project, the CNN structure and context code changed, from identifying digits, to predicting the value of the phase variable, a real value in the interval [0,1]. The phase variable identifies the timestamp of the current motion State to be imitated and is the first variable in the State array. The rationale of this approach is that if the CNN can predict, given the input of several 2D scalograms, what is the current motion timestamp, it must have learned several scalogram features that are useful for the main problem. The input data obtained directly from the mocap file is mostly quaternions and 1-dimensional rotation data. Empirical tests showed that the CNN could predict, albeit unstable, the corresponding phase variable so the last layer before the output was changed from a ReLU (Rectified Linear Unit) activation function to sigmoid, which solved the issue (though it made the system slower). Other changes include removing the dropout steps, reducing the last layer to 128 units, reducing both the number and the size of the filters on the two convolutional layers.

Often, the input of a CNN is a color image with 3 parallel channels (Red, Green and Blue, or a different color model). Here, we adopted the perspective that the different scalograms are different channels from the same "image", thus in our implementation, the CNN has a total of 20 channels (as mentioned in the previous section).

4.5 Model evaluation

In Figure 4.9 **a)** we can observe the performance difference between the base DeepMimic system and the WAILFO model. The base system reaches 0.91 while the new model does not surpass 0.88; as an additional comparison, the reduced state with Baby Walker system passes over 0.89. Figure 4.9 **b)** shows the training times of each system. The WAILFO model takes close to 2.4x more time than DeepMimic (or the reduced state) without bringing any improvements. Additionally, empirical tests show that the character using the policy is less resilient to external forces.

All the tests were made with 2 agents (2 threads), with an Intel Pentium G4560 CPU (Hyper-threading enabled) and 8GB RAM.

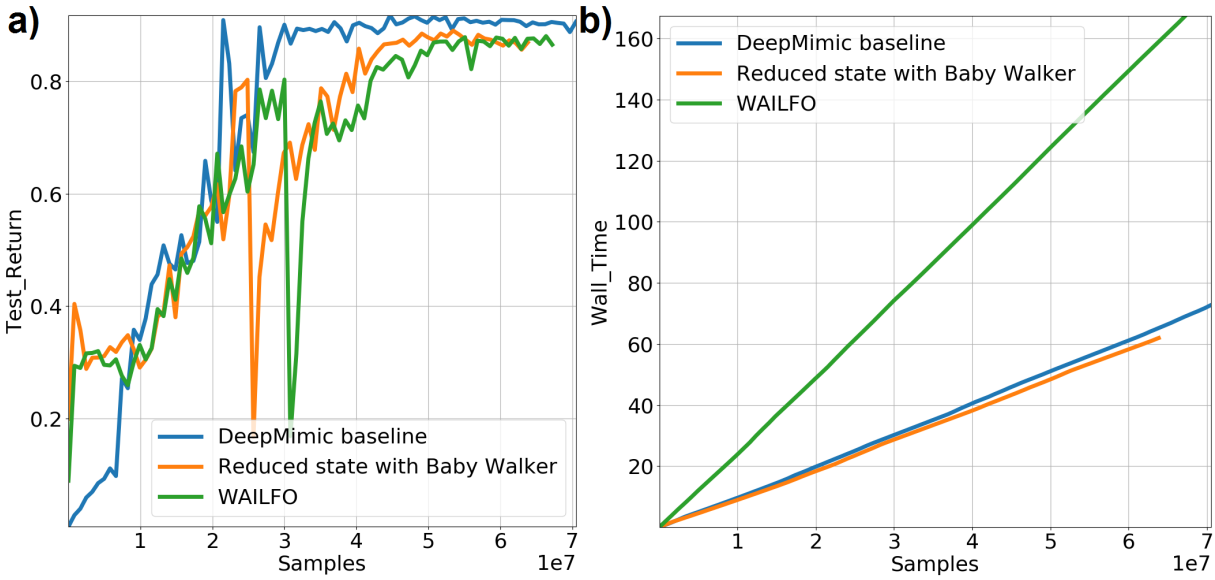

Figure 4.9: Performance **a)** and training time **b)** comparison between DeepMimic baseline (blue), Reduced state with Baby Walker (orange) and the implementation of WAILFO model (green). *Wall_Time* is in hours.

4.6 Discussion

The system using a reduced state with Baby Walker can be seen as a simplified version of DeepMimic and it is the actual base for the WAILFO. As such, it's with this system that the new model was compared to. As it is shown in Figure 4.9 **a)**, both systems show similar performances, albeit with a small advantage to the base system in terms of performance per sample. Moreover, by testing the trained policies during run time, it was evident that the addition of a new model, with the current implementation, reduced the character's resilience to external forces. Both outcomes lead us to conclude that the new model, as implemented, was not beneficial to the system.

The time window of the signals that are processed in the CWT may be too small or too big (as already mentioned in the CWT's *Module Details* section). Supplementary tests, with different time windows of different lengths will shed some light on the effects of this parameter on the system.

It is possible that the frequency information from the scalograms and processed by the CNN is being used by the system, but a specific test should be made to better clarify this - training the WAILFO system without the duplicated state variables (the ones that are saved in the Memory Buffer) in the state array.

The use of a linear scale distribution may have impaired (or delayed) the system from learning important scalogram features from higher frequencies. The use of a logarithmic scale distribution emphasizes the signal's faster changes and reduces the area of the scalogram occupied by lower frequencies, which, along with increased signal sampling, could help with this issue. Other practical problems, due to hardware restrictions, are the small size of the scalograms and associated lack of detail - these can be solved by a more capable hardware system though the decoupling of the CNN from the existing

DeepMimic networks provide a better outlook.

Regarding the CNN, an unresolved problem happens at the beginning of each training episode, when the Memory Buffer is not full. The resulting scalograms represent fake data which may hurt the CNN learning process - the Baby Walker strategy provided assistance to the base system and may have helped lessen the issue, though a test, running WAILFO without that support, has to be conducted to confirm this. An additional improvement to the CNN would be to train it beforehand with mocap - pre-training with the intended motion data would allow the CNN to output useful features right from the start of the policy training.

In terms of training time, the WAILFO implementation took more than 2.4x to reach the same number of steps as the base system. The main contributors are the CWT calculations, with 2 normalization operations for each channel, and the CNN processing and training. The use of logarithmic scale distribution would accelerate the calculations since smaller scales (higher frequencies) are faster to compute. The CNN, as already discussed, could be separated from the existing networks and pre-trained, removing the need to train them during policy learning. And as the neural networks complexity grows, so does the need for faster computations - the introduction of GPU support would allow for faster training with networks of higher complexity.

5

Conclusion

Contents

5.1 Conclusions

The WAILFO model is a first instantiation of the novel approach, for this context, of introducing frequency information to the learning structure. The results are clear - the implemented model offered no observed benefits at the cost of training time and resilience to external forces. Some limitations may explain the results - the CWT calculations are demanding and the resulting scalograms, due to memory constraints, needed to be small and fewer; the CNN also increased the CPU load and there was the need for a simple network since it was not possible to use the GPU for the calculations. Despite this, we learned that some aspects of the implementation aren't advisable, namely saving scalograms in the replay buffer and calculating higher CWT scales (which we can consider wasted computational resources) or using a logarithmic scale distribution. We conclude that the broad approach is worth further investigation, whether with this particular model or other, but always with more computational power. Some compelling alternatives are introduced in the next section.

During this work, some realizations about the base system were obtained. DeepMimic is a marvellous framework with advanced capabilities, very complete and optimized; the lacking of GPU support and the inability to resume training are the main shortcomings of an otherwise great system. In terms of learning assistance and as an alternative to early termination (ET), the Baby Walker strategy seems a simple and effective way to help the learning process and can be considered a valid Curriculum Learning procedure - learning a task with increasing steps of difficulty.

5.2 Future Work

5.2.1 Model improvements

The most pressing change to the model's implementation is separating the CNN and the existing neural networks. The CNN is trained, independently, with an augmented mocap data (synthetic intermediary data with small random variations is introduced) to predict the respective value of the phase variable. Then the weights are saved and locked from further learning. When building the pipeline with DeepMimic, the output unit (along with its connections) is discarded and the last fully connected layer values are fed into the input placeholder of the existing networks. This brings several advantages: the CNN learns more useful features since it trained with small variations of a perfect running animation; during policy learning, the computational load is reduced since the CNN only processes the input and does no longer learn; an array with the values of the last fully connected layer is saved in the replay buffer instead of a set of scalograms (eliminating the memory limitations of the current implementation), allowing for more channels and a more detailed scalograms.

To further improve this new implementation, logarithmic scale distribution will bring faster CWT pro-

cessing and more detailed to higher frequencies. It may also be worth checking if discarding (and, if possible, preventing the processing of) the COI area from the scalogram helps the learning. Finally, looking into other mother wavelets might reveal one more appropriate (with better performance) for this problem.

5.2.2 Other changes

Taking a step back, instead of considering a set of states as the input of the system, it's worth exploring the possibility of using a list of set of states as the sole input of the system (e.g., by using the Memory Buffer). This would change the proposed model by dropping the CWT module and switching to a 1D CNN; moreover, more observations need to be performed between policy updates. This would, possibly, result in higher performance with fewer learning steps since the CNN would recognize known useful movements though, perhaps, at the cost of resilience to external forces and training time. If this new approach emerges as valid, a more computational restrictive solution of adding frequency data is to check the mocap data for the most relevant scale for each joint and use a 1D CNN to process it - this would reduce drastically the number of calculations though its benefits need to be investigated.

DeepMimic supports parallel processing via MPI (Message Passing Interface) - this allows for quicker learning by training multiple agents in parallel. This works well when dealing with simple neural networks but as the complexity increases, so does the need for specialized parallel hardware, like a GPU. Hence, a valuable improvement would be adding GPU support while maintaining the MPI compatibility.

Lastly, in terms of a running animation, the policy update frequency is a subject with promising benefits. Empirical tests have shown that an increase of this frequency, until a certain point, leads to a system that performs better and learns quicker. The ideal solution would be for the system itself to manage the policy update frequency - increasing it to learn faster and when needing a greater accuracy in the motion (e.g., to recover), and decreasing it (during predicted motion parts) to save computational resources when running the policy.

Bibliography

[1] E. Kirilina, N. Yu, A. Jelzow, H. Wabnitz, A. Jacobs, and I. Tachtsidis, "Identifying and quantifying main components of physiological noise in functional near infrared spectroscopy on prefrontal cortex," *Frontiers in human neuroscience*, vol. 7, p. 3, 12 2013.

[2] W. Commons, "File:haar wavelet.png — wikimedia commons, the free media repository," 2019, [Online; accessed 5-January-2020]. [Online]. Available: https://commons.wikimedia.org/w/index.php?title=File:Haar_wavelet.png&oldid=356892702

[3] J. McLoone, "Mexican hat wavelet: Mathematica source code. own work this w3c-unspecified diagram was created with mathematica, cc by-sa 3.0," 2012, [Online; accessed 8-December-2020]. [Online]. Available: https://commons.wikimedia.org/w/index.php?curid=18679602

[4] F. Torabi, G. Warnell, and P. Stone, "Recent advances in imitation learning from observation," *CoRR*, vol. abs/1905.13566, 2019. [Online]. Available: http://arxiv.org/abs/1905.13566

[5] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. A Bradford Book, 1998.

[6] A. Juliani. (2016) Simple reinforcement learning with tensorflow part 8: Asynchronous actor-critic agents (a3c). [Online]. Available: https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c-c88f72a5e9f2

[7] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne, "Deepmimic: Example-guided deep reinforcement learning of physics-based character skills," *CoRR*, vol. abs/1804.02717, 2018. [Online]. Available: http://arxiv.org/abs/1804.02717

[8] S. Park, H. Ryu, S. Lee, S. Lee, and J. Lee, "Learning predict-and-simulate policies from unorganized human motion data," *ACM Trans. Graph.*, vol. 38, no. 6, 2019.

[9] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. A. Riedmiller, and D. Silver, "Emergence of locomotion behaviours in rich environments," *CoRR*, vol. abs/1707.02286, 2017. [Online]. Available: http://arxiv.org/abs/1707.02286

[10] T. Geijtenbeek, “Animating Virtual Characters using Physics-Based Simulation,” PhD Thesis, Utrecht University, 2013.

[11] Y. Castaño-Pino, A. Navarro, B. Muñoz, and J. Orozco, *Using Wavelets for Gait and Arm Swing Analysis*, 03 2019.

[12] N. Ji, H. Zhou, K. Guo, O. Samuel, Z. Huang, L. Xu, and G. Li, “Appropriate mother wavelets for continuous gait event detection based on time-frequency analysis for hemiplegic and healthy individuals,” *Sensors*, vol. 19, p. 3462, 08 2019.

[13] P. Beaudoin, P. Poulin, and M. van de Panne, “Adapting wavelet compression to human motion capture clips,” in *Graphics Interface 2007*, May 2007, pp. 313–318.

[14] P. S. Addison, *The Illustrated Wavelet Transform Handbook*. CRC Press, 2016.

[15] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[16] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” *CoRR*, vol. abs/1602.01783, 2016. [Online]. Available: http://arxiv.org/abs/1602.01783

[17] A. Cruz Ruiz, C. Pontonnier, N. Pronost, and G. Dumont, “Muscle-based control for character animation,” *Computer Graphics Forum*, vol. 36, no. 6, pp. 122–147, 2017. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12863

[18] T. Geijtenbeek, A. F. van der Stappen, and M. Panne, “Flexible muscle-based locomotion for bipedal creatures,” *ACM Transactions on Graphics*, vol. 32, pp. 206:1–206:11, 11 2013.

[19] V. Zordan and J. Hodgins, “Motion capture-driven simulations that hit and react,” 01 2002, p. 89.

[20] S. Park, H. Ryu, S. Lee, S. Lee, and J. Lee, “Learning predict-and-simulate policies from unorganized human motion data,” *ACM Transactions on Graphics*, vol. 38, pp. 1–11, 11 2019.

[21] X. B. Peng and M. van de Panne, “Learning locomotion skills using deeprl: Does the choice of action space matter?” *CoRR*, vol. abs/1611.01055, 2016. [Online]. Available: http://arxiv.org/abs/1611.01055

[22] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, “Curriculum learning,” in *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, 2009, pp. 41–48. [Online]. Available: https://doi.org/10.1145/1553374.1553380

[23] K. Quennesson, E. Ioup, and C. L. Isbell, “Wavelet statistics for human motion classification,” in *AAAI*, 2006.

[24] I. Koenig, P. Eichelberger, A. Blasimann, A. Hauswirth, J.-P. Baeyens, and L. Radlinger, “Wavelet analyses of electromyographic signals derived from lower extremity muscles while walking or running: A systematic review,” *PLOS ONE*, vol. 13, pp. 1–15, 11 2018. [Online]. Available: https://doi.org/10.1371/journal.pone.0206549

[25] C. Caramia, C. De Marchis, and M. Schmid, “Optimizing the scale of a wavelet-based method for the detection of gait events from a waist-mounted accelerometer under different walking speeds,” *Sensors*, vol. 19, p. 1869, 04 2019.

[26] S. K. et al., “Pycwt: spectral analysis using wavelets in python,” 2017, [pycwt 0.3.0a22]. [Online]. Available: https://pypi.org/project/pycwt

[27] A. Damien, “Convolutional neural network,” 2017, [Online; accessed 30-December-2020]. [Online]. Available: https://github.com/aymericdamien/TensorFlow-Examples/blob/master/examples/3_NeuralNetworks/convolutional_network_raw.py

[28] Y. LeCun, C. Cortes, and C. Burges, “Mnist handwritten digit database,” *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.