Neuranimation: Reactive Character Animations with Deep Neural Networks

Sebastian Silva, Sergio Sugahara and Willy Ugarte[a]
Universidad Peruana de Ciencias Aplicadas (UPC), Lima, Peru

Keywords: Neural Networks, Locomotion, Human Motion, Character Animation, Character Control, Deep Learning.

Abstract: The increasing need for more realistic animations has resulted in the implementation of various systems that try to overcome this issue by controlling the character at a base level based on complex techniques. In our work we are using a Phase Functioned Neural Network for generating the next pose of the character in real time while making a comparison with a modified version of the model. The current basic model lacks the ability of producing reactive animations with objects of their surrounding but only reacts to the terrain the character is standing on. Therefore, adding a layer of Rigs with Inverse Kinematics and Blending Trees will allow us to switch between actions depending on the object and adjust the character to fit properly. Our results showed that our proposal improves significantly previous results and that inverse kinematics is essential for this improvement.

1 INTRODUCTION

Due to the constant growth of the entertainment software industry and the further development in computer graphics and artificial intelligence, big companies, such as EA [1] and Ubisoft [2], are investing in Machine Learning driven systems to generate complex animations. Because of this, machine learning methods have been applied to various activities as walking, running and climbing. Nevertheless, those techniques are only attainable by great corporations that possess a motion capture studio. Thus, smaller independent studios recur to traditional methods that mostly do not have a significant presence in the creation of fluid animations accordingly to the objects in the environment in which the character is.

The field of computer animation dedicated to character animations is rapidly increasing the development of animation systems that can help with the development of complex 3D worlds in the video game industry. In addition, the entertainment software industry is constantly growing, just in the United States, according to the Entertainment Software Association (ESA) [3], the American software industry revenue improved a 2% from 2018 to 2019.

This conditions make the animators more valuable for the development of a game of great quality in important companies standards. Therefore, the independent game developers and startups will have to pay salaries up to $33,000 a year [4] for a single animator.

Evenmore, animating characters could be a hard problem since humans exhibit a fluidity in the actions they perform that is difficult to replicate by computer in real time, resulting in simple and unrealistic transitions between actions. Typically, animations for humans are captured from real humans using techniques such as motion capture or manually edited using key frames and interpolation curves, this methods as expensive and time-consuming. Furthermore, the animations of humanoid characters are especially difficult to control, since they present complex dynamic movements, not a clear start and a high index of degrees of freedom(DoF). Therefore, one of the goals of computer animation and the computer graphics area is to be able to generate animations with algorithms.

Many solutions have arisen proposing different methods to control a character by changing their joints properties to select the next pose accordingly to the action intended, methods such as physics-based models (Park et al., 2019; Hwang et al., 2018), motion matching (Holden et al., 2020) or time-series models (Starke et al., 2019; Holden et al., 2017). Although

[a] https://orcid.org/0000-0002-7510-618X

[1] Frostbite presents at GDC and SIGGRAPH - EA

[2] Ubisoft La Forge - Ubisoft

[3] Video Games Generated $35.4 Billion in Revenue for 2019 in The U.S. - ESA

[4] 3D Animation Salaries - Glassdoor

Silva, S., Sugahara, S. and Ugarte, W.
Neuranimation: Reactive Character Animations with Deep Neural Networks.
DOI: 10.5220/0010896500003124
In *Proceedings of the 17th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2022) - Volume 1: GRAPP*, pages 252-259
ISBN: 978-989-758-555-5; ISSN: 2184-4321

many methods had achieved fluid motions, like walking and running (Bergamin et al., 2019; Harvey and Pal, 2018), they respond mostly only to the terrain the character is standing on. On the other hand, some deep learning models are able to recognize environment objects and interact with them (Starke et al., 2019; Holden et al., 2020), but the actions are incorporated in the model so whenever new actions are needed the model needs to be retrained every time. Thus, we propose a mixed approach of deep learning with traditional methods to generate character animations and interactions.

To overcome this issue and make it more accessible, we use a Phase Functioned Neural Network (PFNN) (Holden et al., 2017) for basic states of locomotion, since it brings more freedom to the user, and an animation blending process on top of the network for animation transitioning to different custom actions. Our work is limited to adding animations for the character to do, depending on the object it interacts with, by switching or mixing the PFNN output with a pre-recorded animation, we utilize Unity3D built-in components for physics, collisions and animations control, as well as an animation rigging library, for ease of use.

Our main contributions are as follows:

- We develop an implementation of PFNN with an interaction system on top with assignable animations as needed.
- We propose the use of animation blend transitions with inverse kinematics for pose correction.
- We present an analysis of our method and a comparison with state-of-the-art approaches.

This paper is organized as follows. Section 2 discusses related works. Section 3 introduces relevant concepts, defines the problem and presents our approach. Section 4 shows a experimental study to prove the feasibility of our approach, then, we conclude the paper.

2 RELATED WORKS

In (Holden et al., 2017), the authors propose a novel framework for the synthesis of movements called Phase-Functioned Neural Network. In contrast to other movement synthesis networks, this uses a particular time variable called Phase that is represented by a Phase function as seen in equation 1. In this article they used the Catmull-Rom Cubic Spline function and changed the values of the weights and biases of the network depending on the current phase.

On the other hand, our work utilizes the PFNN to generate basic motion in real time. Nevertheless, we added a simple to use interaction generation system based on Inverse Kinematics (IK) to extend the reach of the PFNN in a simple matter.

Motion Matching is a character animation technique that dependends on a lot of data, and a set of algorithms that search the best suited animation for the next frame. (Holden et al., 2020) propose a state of the art mixed system, using the base of Motion Matching process, including the following algorithms:

1. Compressor: overcomes the need of storing the rotation and translation of the articulations of character, by generating them using only the parameters of the articulations of the character.
2. Stepper: generates a delta that aids the production of the next frame.
3. Projector: Finds the next most suitable step for the animation using K nearest neighbours.

In the article, the authors describe the usage of 4 different neural networks to replace certain steps of the algorithms, concluding in a more efficient Neural Network approach to Motion Matching. Our method describes a mixed approach such as Learned Motion Matching, with a clear difference, the integration of our method can be described as superficial, adding a layer of interactivity to the known PFNN. In contrast with Learn Motion Matching, our method does not need to train to add more interactions neither store the animation database in the application to generate the animations.

In (Zhang et al., 2018), the authors propose the usage of the output of one network (named Gating Network) as blending coefficients of expert weights to determine the dynamic weights for the Motion Prediction Network, in contrast to the PFNN which weights are calculated with a phase function. This gating networks allows the character to switch or blend different locomotion phases according to the user input and terrain variations the character is standing on. However, our method uses the PFNN since it require less data to train and can be stored in little space. In addition to this, the MANN needed the expert weights of each desired action so that they could blend.

In (Starke et al., 2019), the authors use a neural network to determine which action (states), or blend of actions, is needed in the next frame by detecting the surrounding through many voxels around the character in a cylindrical area, as well as the interaction to objects being a voxelization projection of its shape, adding to the network inputs along side the desired motion and character pose given by the user. Similar

to MANN, it uses a Gating Network and Motion Prediction Network in sequence. In contrast, our method controls the interactions on top of the network's output by blending the generated pose with a new assigned action by correcting the pose thanks to various animation rigging calculations, instead of detecting the object before hand and acknowledging them to the network, allowing the user to add as many actions as needed without the need of training a new model with another state.

3 KINEMATICS AND DEEP LEARNING FOR ANIMATION

3.1 Preliminary Concepts

The generation of animations of complex characters, like humans, presents different challenges to computer graphics. This problems include, but are not limited to the management of multiple degrees of freedom, the natural motions required for a animation, the generation in real time of animations, and others. In the following sections, we present some of the approaches other took.

3.1.1 Character Animation

Key Frame Animation. According to Lever (Lever, 2001), animation appears as a series of static images changing rapidly, with the purpose of giving a sensation of movement of the presented images. When animating in modern software, we present a character that can be moved in a virtual space, this movements are made by the animator.

Most of the times, this movements can be created by interpolating two different positions and rotations of a articulation of the character. The process of creating positions and rotations for interpolation is known as Key Frame Animation.

A basic usage of Key frame animations can be seen with the Unity 3D animator when creating a new animation, the user can access to the animation timeline, adding key frames that reference a value, like position or rotation. When the user creates multiple key frames, the engine does the interpolation between these key frames, resulting in a simple animation.

MoCap Animation. According to Kitagawa and Windsor (Kitagawa and Windsor, 2020) Motion Capture or MoCap can be defined as the sampling an recording of human motion, animals or inanimate objects as three dimensional data. We can use motion capture data to model the movements that out character will present in the animation. This movements can be incorporated by adding the animation or using a more complex approach such as Motion Matching.

Holden, Kanoun, Perepichka and Popa (Holden et al., 2020), showed that in Motion Matching, for every N frames the algorithm searches the database containing the Motion Data and finds the motion that best matches the current state and animation of the character. If an animation that has a lower cost than the current one is found, then a transition is inserted between the current step of the animation an the next animation.

Animation Blending. In (Ménardais et al., 2004), the authors mentioned the process to blend animations from short clips to get new longer animations by combining and looping said clips. This process, animation blending, is able to generate more fluid transition between different actions or a fusion of them (in case they can be overlapped). This is accomplished by having different animations and coefficients which determine how much of each animation it takes and thus generating the intermediate poses accordingly.

3.1.2 Character Kinematics

Kinematics describe the rotations and translations of points, objects or group of objects without considering what causes the motion nor physics properties like mass, forces, torque or any other reference.

Most virtual articulated models are complicated consisting in many joints, thus having a high number of degrees of freedom (DoFs) that have to undergone many transformations to achieve a desired pose.

In addition, they are required to satisfy a number of constraints that include joint restrictions so they act naturally, as well as, target nodes or end effectors to indicate where is aimed to end.

Forward Kinematic (FK). Since the characters joints are composed in a hierarchical manner, a way to handle this complexity to create a pose and ensuring some coherence, is by manually adjusting all the DoFs by carefully modifying their rotations so one joint moves its children joints accordingly, this process is also known as Forward Kinematics (FK) where each joints transformation is adjusted, most likely in a local space of the previous joint.

For instance, to animate an arm with a fixed shoulder, the position of the tip of the thumb would be then calculated from the angles of the shoulder, elbow, wrist, thumb and knuckle joints, taking into account all of their DoFs (Kucuk and Bingul, 2006).

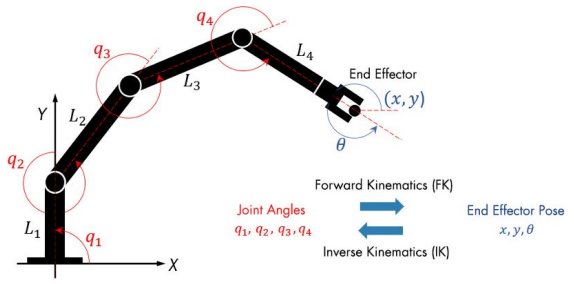

Figure 1: Forward and Inverse Kinematics Operations[5].

Inverse Kinematics (IK). It has become one of the main techniques to manipulate motion data as finding more efficient ways to manipulate the articulated models had become a necessity. IK allows to animate a model by adjusting only the end effectors (usually end effectors are control points, commonly placed at the distal portion of the limbs, such as feet or hands), this end effectors position and orientation are often determined by the animator or a MoCap reference.

Thanks to IK the rest of DoFs from parent nodes are automatically determined following different criteria (specified by the model constraints) according to the position of the end effector in world space, saving lots of work to the animator while still having control of it and creating a coherent pose (Aristidou et al., 2018). Fig. 1 depicts an example of FK and IK.

3.1.3 Deep Learning Animation Generation

Mixed Methods. This methods depend on different algorithms or methods for animation. Some examples for these methods are the following:

- *Learned Motion Matching:* Holden et al. (Holden et al., 2020) describes the usage of four Deep Neural Networks inside of Decompressor, Stepper and Projector Algorithms that are used in standard Motion Matching System for animations.
- *DeepLoco:* Peng, Berseth, Yin and Van De Panne (Peng et al., 2017) developed a Deep Reinforcement Learning approach to the animation of bipedal locomotion that highly depends of the Physics constraints added by the physics system.

Deep Neural Networks. In deep neural network approaches, the motion is generated as an output of one or more neural networks, each one of them executing a precise task depending on the network.

(Holden et al., 2017) implemented a single network with a phase function, this approach allows the character to monitor and change the weights of the network depending on the current phase. The phase function provides a simple solution to the reactive animation generation. This approach is known as Phased-Function Neural Network (henceforth noted by PFNN).

(Zhang et al., 2018) provided a method with no phase function that presented good results with quadruped characters. This approach depends on a Gating Network that finds a animation suited for the next step in a group of animations.

This Gating network can be seen in the work of Starke, Zhang, Komura and Saito (Starke et al., 2019), where the Gating Network is used in conjunction with a motion prediction network and four encoders that help the animation to generate a interaction with an object.

3.2 Animation Package Development

For the development of a Unity package capable of switching between animations with every user input or character-object interaction we proposed a character controlled PFNN for movement in every direction adapting to terrain variations and different types of movement (i.e. crouching or running) and applying the animation rigs necessary when interacting with object to correct the pose of the character.

3.2.1 PFNN for Character Basic Animations

In this section, we first describe the Phase Function, the layers used in the model and the training process. This Model is based and shares the Phase Function with the work presented by (Holden et al., 2017).

Phase Function. The phase function, as described by (Holden et al., 2017), computes a set of values called α that will be used by the network to generate the next pose in each frame. The phase function is represented by $\alpha = \Theta(p, \beta)$ (Holden et al., 2017), where p is the current Phase and β are the parameters.

The phase function can be any type of function or even another Neural Network, but for this project we used the Cubic Catmull-Rom Spline as a cyclic function, this requires that the start and the end control points be in the same place, as seen in the Fig. 2.

We use the Catmull-Rom Spline as the phase function, because of the cyclic behaviour described before, and, as pointed by (Holden et al., 2017), the function with the best performance for PFNN was Catmull-Rom Cubic Spline, which presents four control points.

[5]Inverse Kinematics

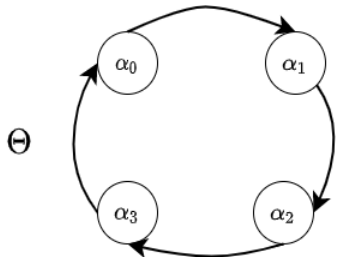

Figure 2: Graphic representation of Catmull-Rom Cubic Spline as a Cyclic Function.

Each control point α_k represents a set of weights in the neural network, this control points are use as in the Phase Function as $\beta = \{\alpha_0, \alpha_1, \alpha_2, \alpha_3\}$. The generation of the values to be used in this frame for a arbitrary p can be express as follows:

$$\begin{aligned}\Theta(p;\beta) = \quad & \alpha_{k_1} \\ & + w(\frac{1}{2}\alpha_{k_2} - \frac{1}{2}\alpha_{k_0}) \\ & + w^2(\alpha_{k_0} - \frac{5}{2}\alpha_{k_1} + 2\alpha_{k_2} - \frac{1}{2}\alpha_{k_3}) \\ & + w^3(\frac{3}{2}\alpha_{k_1} - \frac{3}{2}\alpha_{k_2} + \frac{1}{2}\alpha_{k_3} - \frac{1}{2}\alpha_{k_0}) \\ \text{Where} \quad & w = \frac{4p}{2\pi}(mod\,1) \\ & k_n = \left[\frac{4p}{2\pi}\right] + (n-1)(mod\,4)\end{aligned} \tag{1}$$

Model Structure. The PFNN model depends on the quantity of control points that are present in the subsection 3.2.1. Therefore, every layer in the model will present a configuration of four sets of weights and bias. The structure we used can be represented as a three layer model with 512 units for layers zero and one, and 311 for layer two.

For the activator of the layers we used Exponential rectified linear function activator(ELU). The neural network Θ (Holden et al., 2017) can be represented as:

$$\Phi(x,a) = W_2\text{ELU}(W_1\text{ELU}(W_0x + b_0) + b_1) + b_2 \tag{2}$$

where :

$$\text{ELU} = max(x,0) + exp(min(x,0)) - 1 \tag{3}$$

Where W_k and b_k is the network parameters returned by the phase function Θ as seen in Equation 1.

Training. For the training we take each frame x and its next frame y and the current phase p and create three matrices as $X = [x_0, x_1, ...]$, $Y = [y_0, y_1, ...]$ and $P = [p_0, p_1, ...]$. We calculate the mean and standard deviation of X and Y and normalized the data.

For the loss function of the model we used Mean Square Error and for the optimization, stochastic gradient descend algorithm (Kingma and Ba, 2014). For the construction and training of the model we used Tensorflow 2, with keras custom layers and models. We included Dropout layers with a retention probability of .7 and trained using batches of size 32. The training was performed with 10 epochs in about 12 hours in a Tesla P100.

3.2.2 Character Animation Rigging

For having a better control over the character we applied motion rigs to its skeleton defining many referential constraints between different parts of the limbs, torso, root node, etc. for the purpose of maintaining a correct anatomy and avoiding muscles contracting to the wrong direction.

As well as, adding two-bone IK constraints to the limbs, specifying the three nodes that compose them (i.e. shoulder, elbow and wrist for arms) and assigning new target nodes for a simple re-positioning of the distal portion of the limbs (hands and feet) by reorientating the two upper nodes for the distal node to match the target.

Thus allowing us to override the limbs animations for a pose correction or to move them independently to do a certain action (i.e. grabbing a door knob to open it). In Addition, by having IK constraints only applied to the limbs, the root node of the character (most likely the Hips) can be controlled separately and, thanks to the pose correction, have a full body control.

3.2.3 System Overview

In this section we will discuss how all of this was integrated together in Unity 3D thanks to its built-in properties and how our system works step-by-step as seen in Fig. 3.

Incorporation to Unity 3D. To accomplish the task of generating reactive animations we utilize Unity 3D because of its various embedded systems, including hierarchical Game Objects, Transformations, Rigid bodies, collisions and the Animation Rigging package developed for it.

Thanks to the mentioned systems we had the capability of building a hierarchical skeleton composition making possible the use of FK to create traditional animations for specific actions and to use the skeleton joints Transformation properties as input for the

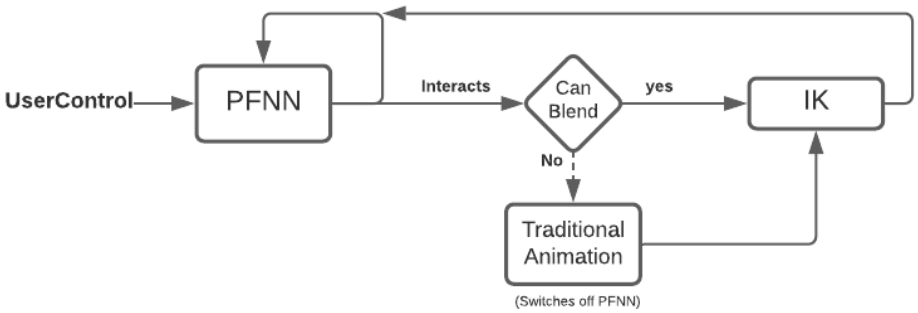

Figure 3: Graphic representation of how our system works.

PFNN which is built from a collection of binary files containing the weights and biases.

Furthermore, the rigid body and collisions system allows the detection and identification of the objects the character is interacting with. By this method, it can be determined which action to perform in each scenario since not all animations where incorporated within the possible PFNN outputs.

A way of switching back and forth between traditional animation and generating poses from a Deep Learning model to adjust to the action needed. Therefore, interactions that cannot be blended with the generated poses can still be performed by the character (i.e. sitting on a chair) by defining which animation to play when colliding with each object. Because of this, animation blending was integrated to take the output of the network and the corresponding action to play making a transition between pre-recorded animations and the model generated ones.

In addition to this, a whole scenario was created consisting different types of objects (a door and two chairs), a rugged terrain section and a movable object to test the performance of the character to different interactions. Notably, the PFNN excels at generating free movements in different locomotion phases, moreover, it adapts to the variations of the terrain its standing on (i.e. hiking a steep slope or trotting down hill).

Frame Pose Calculation. In each frame, if the PFNN is active, the script takes the joints properties, trajectory and current Phase for the following calculations. Then, the joint properties and trajectory are normalized. Afterwards, we use the Phase p to generate the index of the weights that are going to be use, described as follows:

$$\text{index} = 50\left(\frac{p}{2\pi}\right) \tag{4}$$

After that, we run the neural network with the weights and biases that correspond to the calculated index. Finally, we re-normalize the result and update the model and the phase. If the humanoid character interacts with an object, the rigs that correspond to said object are activated and the pose correction is done by the IK functions described in section 3.2.2.

4 EXPERIMENTS

In this section we will discuss the experiments our project has undergone, as well as, what is needed to replicate said experiments and a discussion of the results obtained after this process.

4.1 Experimental Protocol

To recreate the process of building, training and testing the model utilized in our project, we begin to describe what was needed to accomplish such task.

4.1.1 Development Environment

The environment used as our main platform where all our models were trained was Google Colaboratory Pro which provides us a total RAM of 25GB and a Tesla T4 or a Tesla P100 GPU, including the Tensorflow 2 and Keras libraries. Also having Google One, the first tier subscription, for up to 100 GB of storage space in Google Drive. Additionally, the game engine Unity 3D 2020.3.16f1 for the different objects (hierarchy and orientation properties) and physics systems (collisions and rigidbodies).

4.1.2 Dataset

The dataset was built from the scripts provided by Holden along side his PFNN article (Holden et al., 2017) which process many motion capture clips to obtain the joint properties of the person recorded and generating three numpy arrays, corresponding to the input data as X, the output data as Y and the current phase as P.

This numpy arrays are compressed and stored in a .npz file as a collection, in which the X is saved as

Table 1: RAM Consumption of the different projects that generate reactive animations.

Project	RAM
PFNN Adam (Holden et al., 2017)	1,034 MB
PFNN Original (Holden et al., 2017)	1,545 MB
NSM (Starke et al., 2019)	1,657 MB
Neuranimation (Ours)	1,237 MB

Xun, Y, as Yun and P, as Pun. The size of his .npz file is about 6.80 GB, containing 4,353,570 tuples of 342 features in Xun, 311 features in Yun and one feature in Pun.

4.1.3 Models Training

All model architectures were trained in Google Colab Pro utilizing Tensorflow 2 in a virtual GPU environment with extended RAM as seen in Section 4.1.1, with a 24 hour maximum run-time, for 10 epochs using batches of 32 tuples from the dataset, with the Adam optimizer and trying to minimize the loss value obtained with a Mean Square Error function, averaging 52 minutes per epoch. By having all models undergone the same training process we can analyze how they behave and the learning tendencies they take, all model took around nine hours to train.

4.1.4 Testing Environment

To qualify the performance of the models, we consider the loss function as the main metric to minimize during training. To preview the results of the generated poses we have created a Unity 3D demo playground scene, where a model can be loaded from their weights and biases saved in binary files. There we determine how realistic the model is generating poses in a qualitative manner, while measuring running performance (in RAM) to compare it with similar Unity 3D methods.

4.1.5 Source Code

Our code and dataset are publicly avaiable at https://github.com/SergioSugaharaC/Neuranimations, specifically the Unity 3D project and its development versions for testing the models and their added reactivity features with the environment's objects and terrain, as well as, our models and script notebooks had been uploaded at https://drive.google.com/drive/folders/14Xq5KwYPzx_vGre878BQq51GxALuzsfM?usp=sharing

4.2 Results

To compare our method of generating animations and interactions we used different projects based in Unity 3D and Deep Neural Networks that generates reactive animations and interactions (Starke et al., 2019) (Holden et al., 2017) (Zhang et al., 2018). For the comparison, we took the Unity 3D projects of each article and run them to see their performance in terms of RAM usage as seen in Table 1 where we can observe the different performance each Neural Network has in the projects. This tests were made in an i7-9700 with a total RAM of 16GB and a Nvidia 1050Ti Graphics Card.

In Table 1 we analyze our method presents slightly better performance than the PFNN Original (Holden et al., 2017) project, which presents the basic PFNN animations, and the NSM (Starke et al., 2019), a project containing different interactions that are generated by the corresponding Deep Neural Network. On the other hand, the PFNN Adam Project has better performance than ours, due to this being an optimization on the Original PFNN resulting in a better performance difference as expected.

As seen in Table 2, our model does not need to be retrained whenever a new action is desired to be added, since it works on top of the networks output by blending it with a pose correction controlled by IK. Hence, the network's independence allows us to increase the capabilities of performing different actions by adding hand crafted animations or blending them to interact with said objects accordingly.

4.3 Discussion

As presented in section 4.2, our method presents better performance than similar implementations of animation generation in Unity 3D while adding interactions to the framework with the usage of inverse kinematics. In Table 1, we compared our RAM consumption with two PFNN implementations made in Unity 3D and a NSM implementation, also in Unity 3D, because of the difference to manage interactions. With this results we can conclude that our method presents a great alternative to other Neural Network only approaches, maintaining a good performance while adding the animations framework using animation rigs.

Furthermore, we presented the Table 2 that describes the needs that are present when adding a new interaction to their projects. Our method excels at adding animations to the project, by using an IK system with targets and a set of personalized intractable objects. In contrast to NSM (Starke et al., 2019) that

Table 2: Traning comparison of NSM and Neuranimation.

Interactions	NSM			Ours		
	Min Animations	Re-training	Input Features	Min Animations	Re-training	Input Features
1	5	Yes	n	1	No	311
2	10	Yes	$n+14$	2	No	311
3	15	Yes	$n+28$	3	No	311
4	20	Yes	$n+42$	4	No	311

describes the state of the art Neural Network model for the generation of animation of interactions while correcting pose, needs to retrain and restructure input and output data to add a new interaction to the model.

5 CONCLUSION

We conclude that by adding pose correction with IK calculations to a Neural Network approach of animation generation, we could add interactions by using traditional animation for said interactions and the Neural Network for the generations of the basic animations. This mixed approach does not represents a performance reduction while making the character more reactive to the environment desired.

By adding animation riggings, we conclude that the generation of new animations can be highly enhanced by this method, adding constraints of movement and simplifying the calculations of bone position an rotation while maintaining a correct anatomical structure. Also, simplifying the generation of reactive animations using target for the rigs to move at.

In future works, we aim to improve the model as a part of the PFNN instead of our current two step approach, or even summarize events on the fly (Chancolla-Neira et al., 2020).

REFERENCES

Aristidou, A., Lasenby, J., Chrysanthou, Y., and Shamir, A. (2018). Inverse kinematics techniques in computer graphics: A survey. *Computer Graphics Forum*, 37(6).

Bergamin, K., Clavet, S., Holden, D., and Forbes, J. R. (2019). Drecon: data-driven responsive control of physics-based characters. *ACM Transactions On Graphics*, 38(6).

Chancolla-Neira, S. W., Salinas-Lozano, C. E., and Ugarte, W. (2020). Static summarization using pearson's coefficient and transfer learning for anomaly detection for surveillance videos. In *SIMBig*.

Harvey, F. G. and Pal, C. J. (2018). Recurrent transition networks for character locomotion. In *ACM SIGGRAPH Asia Technical Briefs*.

Holden, D., Kanoun, O., Perepichka, M., and Popa, T. (2020). Learned motion matching. *ACM Transactions on Graphics*, 39(4).

Holden, D., Komura, T., and Saito, J. (2017). Phase-functioned neural networks for character control. *ACM Transactions on Graphics*, 36(4).

Hwang, J., Kim, J., Suh, I. H., and Kwon, T. (2018). Real-time locomotion controller using an inverted-pendulum-based abstract model. *Comput. Graph. Forum*, 37(2).

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kitagawa, M. and Windsor, B. (2020). *MoCap for artists: workflow and techniques for motion capture*. CRC Press.

Kucuk, S. and Bingul, Z. (2006). *Robot Kinematics: Forward and Inverse Kinematics*.

Lever, N. (2001). *Real-time 3D character animation with Visual C++*. Routledge.

Ménardais, S., Kulpa, R., Multon, F., and Arnaldi, B. (2004). Synchronization for dynamic blending of motions. In *Symposium on Computer Animation*.

Park, S., Ryu, H., Lee, S., Lee, S., and Lee, J. (2019). Learning predict-and-simulate policies from unorganized human motion data. *ACM Transactions on Graphics*, 38(6).

Peng, X. B., Berseth, G., Yin, K., and Van De Panne, M. (2017). Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics*, 36(4).

Starke, S., Zhang, H., Komura, T., and Saito, J. (2019). Neural state machine for character-scene interactions. *ACM Trans. Graph.*, 38(6).

Zhang, H., Starke, S., Komura, T., and Saito, J. (2018). Mode-adaptive neural networks for quadruped motion control. *ACM Transactions on Graphics*, 37(4).