Journal, Vol. XXI, No. 1, 1-5, 2013
Additional note

Gesture-Recognition; Character Control; Virtual Reality; Machine Learning; Real-Time; User-Defined

Abdesselam Guerroudj[1]*, Sheldon Andrews[1]

Abstract
This paper presents the initial work towards developing Gesture-Based Character Control, a gesture-recognition, and control system that allows users to record custom gestures using the Virtual Reality controllers and use it to control a character's corresponding actions in real time. Firstly, a neural network is trained offline on an open Mocap database to distinguish between gestures and idle actions that the user may perform in real time. Then, during runtime, the neural network is tuned to improve detecting gestures each time the user performs them. In this report, we present the first phase of the project. The offline training of the system to efficiently distinguish between gestures and idle action.

Keywords
Gesture-Recognition – Character Control – Virtual Reality – Machine Learning – Real-Time – User-Defined

[1] *Software Engineering Department, École de Technologie Supérieure , Montreal, QC, Canada*

Introduction

Gesture-based control of 3D characters is an intriguing problem wherein the overall goal is to provide expressive and interactive control of an animated character using only simple gestures and tracking devices. This type of technology has numerous benefits for virtual reality and video game applications. Using gestures to control 3D characters in virtual reality will significantly improve immersiveness. However, different users may identify different gestures to be more natural for a given action than others.
For example, to fly an eagle in Virtual Reality Figure 1, some players will find that flapping their entire arms to feel more immerse while controlling the eagle. Other players, maybe lazier and settle for moving only their hands in a flapping motion to be a more convenient way to control the flapping of the eagle's wings. The accommodating solution is to build a system that gives users the ability to record what they deem as a suitable gesture and assign it to the various virtual character actions. The ideal system will satisfy the following criteria:

- Identify the custom gesture and use it to drive the corresponding action or motion of the virtual character in real time.
- Distinguish between the various custom gestures being performed in real time.
- Discriminate between the active gestures and the idle passive movements that the user may perform while not intending to control the character.

1. Approach

Machine learning methods are well suited to the task of gesture recognition. In particular, recurrent neural networks (RNNs) are capable of encoding both temporal and spatial aspects of the mapping between gesture and character motion. We propose training a long short-term memory (LSTM) to recognize bespoke gestures that are defined by the end-user. We specifically target control of 3d characters for real-time video games using HTC-Vive trackers. Training will be split into two stages :

- **Learning the null class of motions** These are movements that do not map to any gesture, such as idling, hand gestures during conversation, itching or scratching, and other motions we want the system to reject as valid gestures.
- **Learning active gesture class of motions** These are movements the user does intend to use to control the 3d character, and they are customized by

Figure 1. Example User-specified motion control of virtual character using VR controllers

the user during a calibration phase before online play.

Due to the sequential nature of the gestures and movements involved in real-time control, Hidden Markov Models (HMMs) [1] and Recurrent Neural Nets (RNNs) are the most suitable training algorithms to use. However, HMMs need strong assumptions to be made about the model such as the dependency of the current state on only the previous and next states. In addition, HMMs are not complex enough to discriminate between a user idling and performing a gesture. This makes RNNs much more suitable due to their discriminative nature, and their complexity. Since we have access to large datasets, RNNs was our choice. Furthermore, the importance of sequence in motion requires us to choose LSTMs to handle learning the long-term dependencies and avoid the potential vanishing gradient problem.

2. Experimental Setup

For starters, both classifiers will be trained by leveraging the large existing database of human motion, CMU [2] dataset. The hand positions from a set of motion files will be extracted, localized, then perturbed slightly.
More specifically, we manually selected two sets of motion files from the database: movements, and gestures. We modified the motion skeletons by adding a single point at each hand mimicking the position of a VR controller.
The positions of the newly added points are tracked, then localized relative to the root bone. This effectively eliminates locomotion and adds more generalizability to our data. To add more robustness, we added per-frame white noise on the positions of the two points. See Figure 2
For each motion file, we generate 20 different iterations with white noise added at each frame generating perturbed motions. Perturbations are to mimic real-time controller movement noise. This results in a much more robust model. We used 56 motion files for each class with 20 different iterations for each. Resulting in a dataset of 6 columns by 6,373,900 rows. Columns contain positional coordinates (X, Y, Z) for both hands. Rows correspond to total frames of all motion files used. See Figure 3a
As an example, a gesture motion file with 6 seconds of length is 720 frames long recorded at 120 frames per second. A frame of data consists of the positional information of both hands, 6 coordinates. We add noise to each hand coordinates separately, at each frame. We do this 20 times, resulting in $720 \times 20 = 14400$ frames of data.
To train on the data, we chose to train an LSTM network using Keras [3] with TensorFlow [4] in Python. One more necessary step to help with the training was changing the data from the original shape of (6373900, 6) to (84984, 360). By concatenating each 3 seconds of data, i.e. 360 frames/rows into one row. See Figure 3b

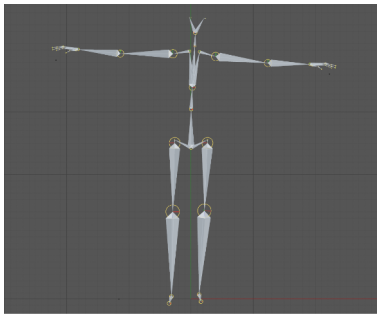
3. Results

During training, we noticed several parameters affected the training results. We performed a parameter sweep. Firstly, 3 layers seemed to show the right compromise of speed and accuracy. Then, we sweeped over the rest of parameters as follows:

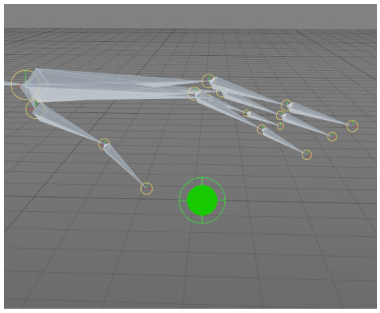
- Number of nodes: [16, 32, 128, 256, 512]
- Dropout: [0.1, 0.2, 0.3, 0.4, 0.5, 0.6]
- Activation Function: [ReLu, Sigmoid]
- Learning rate: [0.1, 0.01, 0.001, 0.0001]

Out of the parameters above, the best training results were obtained using: 256 nodes, a dropout of 0.2, with ReLu (rectified linear unit) as activation function, and a learning rate of $1e^{-3}$ We added a decay of $1e^{-6}$ to the learning rate as well. The final network structure was made of two LSTM layers with 256 nodes each, and a dense connected layer with 256 nodes as well.
The output, had two classes labeled as 0 or 1 with 0 being a non-gesture and a 1 being a gesture. This also made the loss function to be a categorical cross entropy seeing how this is a binary output. See Figure 4

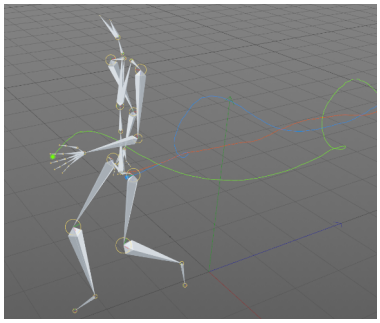
Then, final training was performed with added 5 fold cross validation for further inspection of the best possible combination of training and testing data. See Figure 5
The results are very positive so far. The training was considerably fast.



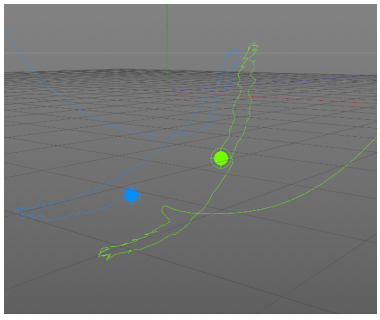
(a) Original skeleton from CMU database



(b) Added virtual markers on each hand

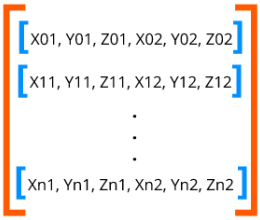


(c) Virtual markers positions tracking

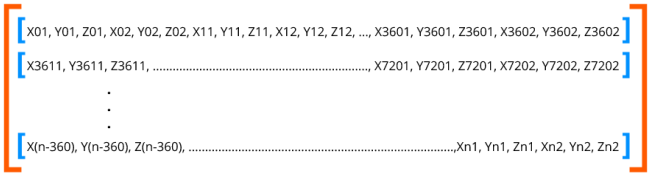


(d) Localization of virtual markers positions relative to root bone and perturbation

Figure 2. Data preparation steps



(a) Original Shape



(b) Reshaped

Figure 3. Reshaping training data to longer arrays to preserve the temporal information of the motion. n =6,373,900

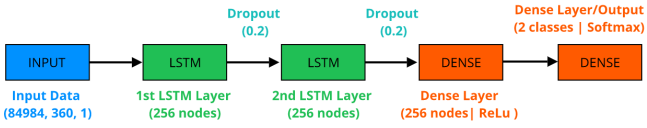
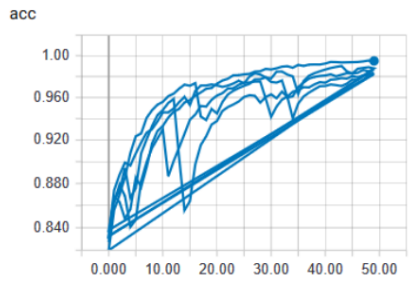
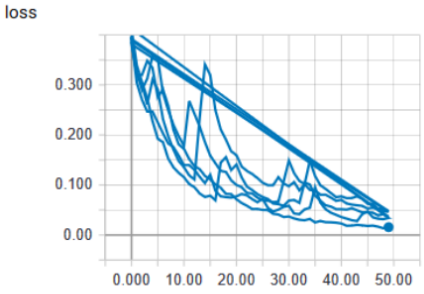


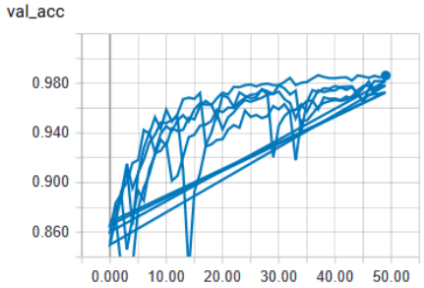
Figure 4. LSTM network model. Loss function: Categorical Cross-entropy. Optimizer: Adam. Learning rate: e^{-3} decay e^{-6})



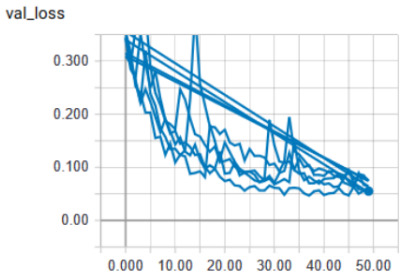
(a) Training set accuracy. Best value: 99.50%



(b) Training set loss. Best value: 1.57%



(c) Testing set accuracy. Best value: 98.64%



(d) Testing set loss. Best value: 5.52%

Figure 5. Training Results

4. Conclusions and Future Work

The training results obtained are backing our expectations of this system being able to recognize user gestures in real-time.

An immediately required addition would be to normalize the data to be independent of the player's height. Furthermore, we are planning to work on the second phase of building this system. In the upcoming phase, the system will take user-input and train on it so that it recognizes the gesture every time the user performs it. Since it is highly unlikely that the user will perform the motion exactly the same each time they attempt to control the character. We, therefore, plan to generate a robust training set using only a small number of examples using a similar idea to the aforementioned technique. However, rather than generating variations of skeleton motion, we plan to fit parametric curves to trajectories from the HTC Vive trackers and perturb the motions in the parameter space of the curves. This includes not only spatial but also temporal variations of the motions. In this way, only a small number of example motions will be used to produce hundreds.

Acknowledgments

The main idea of this project was extended upon a research project that I worked on under the supervision of my supervisor Dr. Sheldon Andrews during the summer. The original project was an extension of Master of Puppets paper [1] using HMMs My supervisor, came up with the idea of training a neural network instead of HMMs. I attribute this work to his continuous supervision and guidance, and all the help he's given me.

References

[1] Yaoyuan Cui and Christos Mousas. Master of puppets: An animation-by-demonstration computer puppetry authoring framework. *3D Research*, 9(1):5, 2018.

[2] CMU MoCap. The data used in this project was obtained from mocap. cs. cmu. edu. the database was created with funding from nsf eia-0196217. *City*, 2003.

[3] François Chollet et al. Keras. `https://keras.io`, 2015.

[4] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin,

Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.