Virtual Equipment System: Waist Location, 3D Gesture, and Peripersonal Socket Locations Using Machine Learning

Category: Research

In this paper, we use machine learning to run 3 experiments intended to improve our Virtual Equipment System (VES). The first experiment is about predicting the position and rotation of the user's waist. The second experiment is about classifying the user's controller interaction with a piece of virtual equipment into different equipment gestures. The third experiment involves predicting locations for equipment to be placed in personal and peripersonal space.

1 Introduction

Body tracking is an important task in virtual reality. Most extended reality (XR) kits utilize a headset and 2 hand controllers to interact in XR. In order to track additional parts of the body, the user must purchase additional trackers or a camera capable of tracking the entire body such as the Microsoft Kinect. Relying on extra equipment is undesirable since adding additional equipment is costly and adds another barrier to entry for using VES. In this paper, we attempt to predict the user's waist using only a headset and 2 hand controllers.

Gesture recognition is a difficult problem in XR. Most gesture recognition techniques are not truly three dimensional and require mapping to a two-dimensional space before processing. In addition, current available gesture recognition applications are not complex enough for use in VES. In this paper, we propose to use machine learning to accurately classify three-dimensional gestures.

2 Related Work

2.1 Waist/Torso Tracking

Two of the common ways to do body tracking is with a hardware tracker (e.g. as part of a vr headset) or an external camera-based tracker.

For hardware trackers, some VR platforms such as HTC Vive have their own Vive Tracker that can be used. However, they cost $99 each. This adds additional cost to the already pricey VR setup ($800 for HTC Vive, $300 for Oculus Quest). They also require additional setup on the user's part.

For Camera-based tracking solutions, there are solutions such as OpenPose, Kinect, or Google's MediaPipe. Once setup, it can track multiple body parts, but no simple solution exists. The user would have to learn to setup these solutions and they would only work with VR experiences that support them.

We wanted to utilize just the headset and the two motion controllers, something that's available in most commercial VR solutions, to see if we can predict the position and direction of the waist. While we suspect there may not be enough information encoded in those 3 devices to figure out the user's body data, we wanted to give it a try. We were only able to find one work that is related to our approach. This paper, by Lee et al., showed that it was possible to classify quite accurately whether the torso was facing the front, the left, or the right, using just headset data.

2.2 VR Gestures

For Experiment 1, we wanted to be able to perform gestures where the system adapts to the user rather than the user adapting to the system. We looked into VR gesture support in the game engine Unity. There is a plethora of work done in terms of hand tracking. Hand tracking is now officially supported on platforms such as Oculus Quest. However, there is very little support for VR gestures in Unity's asset store. The available gestures are hand gestures, head gestures, body gestures, or a generic 3d gesture.

What we want to do most closely resemble a generic 3d gesture. The user is to grab some piece of equipment and move it around in 3D space to perform a gesture. A common approach is to collect the points in 3d space, map them onto a 2d plane, then compare the result to a trained result by looking at the distance difference (e.g. Levenshtein distance). The downside to this approach is that the users are required to match the initial pattern, which can be difficult when the user is in a different orientation than when they first produced the initial pattern.

3 Experiment 0: Waist Tracking

3.1 Dataset

The waist tracking dataset consists of a variety of movements to capture a wide range of waist movement possibilities. The user was asked to simulate various activities such as playing a sport or a game.

3.2 Machine Learning Architecture

We used PyTorch and Unity's Barracuda to prototype our initial models. However, we found the Unity ML-Agents package to be more intuitive and had more in-built support for complex architectures such as RNNs and LSTMs which are not supported as of this writing by the Barracuda package.

3.2.1 PyTorch - MLP

Multi-layer perceptrons are easy to design and implement, and for our purpose we developed a three layer perceptron which takes as inputs the headset and controller positions and rotations for both hands. It outputs the position of the waist tracker relative to the headset and its rotation in quaternions. The model makes use of ReLU activation to introduce non-linearity and batch normalization to prevent overfitting the training set.

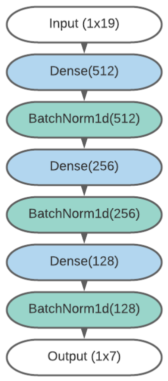

Figure 1: MLP Architecture

3.2.2 PyTorch - LSTM

LSTMs are useful for working with data where predictions are related to their timestamps. We intended to make use of the sequential nature of our data through training an LSTM for the regression task. We made several changes to the model proposed by J. Lee et. al. [1]. We removed the batch normalization layers and changed the hidden dimension for the first LSTM layer to 512 and the second LSTM layer to 128. The batch normalization layers had issues translating to ONNX in Unity with the LSTM and the end model was not supported in Unity.

3.2.3 ML-Agents

The Unity Machine Learning Agents Toolkit is an open-source project that allows for training agents in a game and simulation environment. In our experiment, we utilize ML-Agents to train agents to find the waist's position and rotation using reinforcement learning and recurrent neural networks.

3.3 Observations

The Agent will have access to its own position and rotation as well as the positions and rotations of the headset and controllers. The observations are setup as stacked Vectors so past positions and rotations are utilized. Another configuration does not use stacked Vectors, but utilizes the memory settings in the training configuration to use recurrent neural networks.

3.3.1 Actions

The agent has two possible sets of actions. If the agent is setup to use stacked Vectors then the agent will perform continuous actions in the x, y, and z direction. If the agent is setup to use recurrent neural networks then the agent can perform a set of 7 discrete actions. The agent has the option to do the following: do nothing, +x, -x, +y, -y, +z, or -z.

3.3.2 Rewards

The current model gives the agent a reward of 1.0 if its current distance to the target is less than its previous distance to the target. The agent receives a reward of -1.0 otherwise.

3.4 Preliminary Results

3.4.1 PyTorch MLP

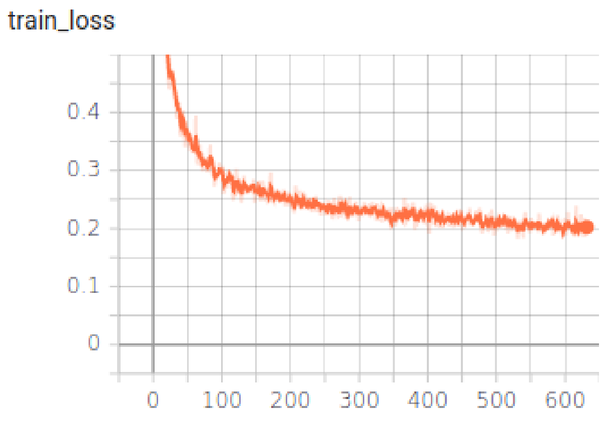

Figure 2: MLP Train Loss

The MLP model performs well in predicting the waist tracker position close to a radius of 2.5 cm and the rotation close to the ground truth waist tracker. We trained with a batch size of 128 with over 34k data points from a single user. The model was able to learn the tracker prediction function fairly quickly in under 600 epochs shown in Fig. 2 and 3 above and converges to a best L2 loss of 0.14 after 1500 epochs. We can see some of the predictions from the MLP in Fig. 4

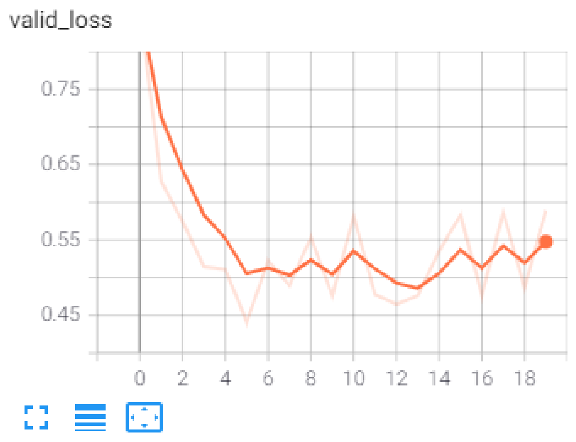

Figure 3: MLP Valid Loss

3.4.2 PyTorch LSTM

The LSTM model had a higher loss as compared to the MLP model and was unable to generalize the training set, resulting in the unstable tracker position and rotation predictions shown in Fig. 5 and 6.

3.4.3 ML-Agent

Our best preliminary result at the moment is an average of 8cm from the actual tracker to the predicted tracker, compared to the naive model's average of 10cm.

3.5 Conclusion

4 Experiment 1: 3D Gesture Recognition

4.1 Dataset

The gesture recognition dataset consists of multiple users performing simple unidirectional hand movements around the user's headset. For the time being the dataset is limited to 4 different classes of gestures based around the idea that the user will have a pair of virtual headphones. The users could control the audio settings with those gestures in the following ways:

- Up gesture to increase volume
- Down gesture to decrease volume
- Forward gesture to skip songs
- Backward gesture to rewind songs

The gestures or movements are represented by positional samples (x, y, z), and rotation samples (Euler and quaternion) across time. For this dataset we use 10 samples across the 1 second period prior to the end of the gesture. The positional samples are also made relative to the headset by subtracting the hand positions from the head position. This is done so that the trained model is not reliant on the user being in any given region of the unity world.

4.1.1 Features

All together the model trains on a dataset consisting of 150 features:

- 10 right hand relative position samples (x, y, z)
- 10 left hand relative position samples (x, y, z)
- 10 headset Euler rotation samples (x, y, z)
- 10 right hand Euler rotation samples (x, y, z)
- 10 left hand Euler rotation samples (x, y, z)

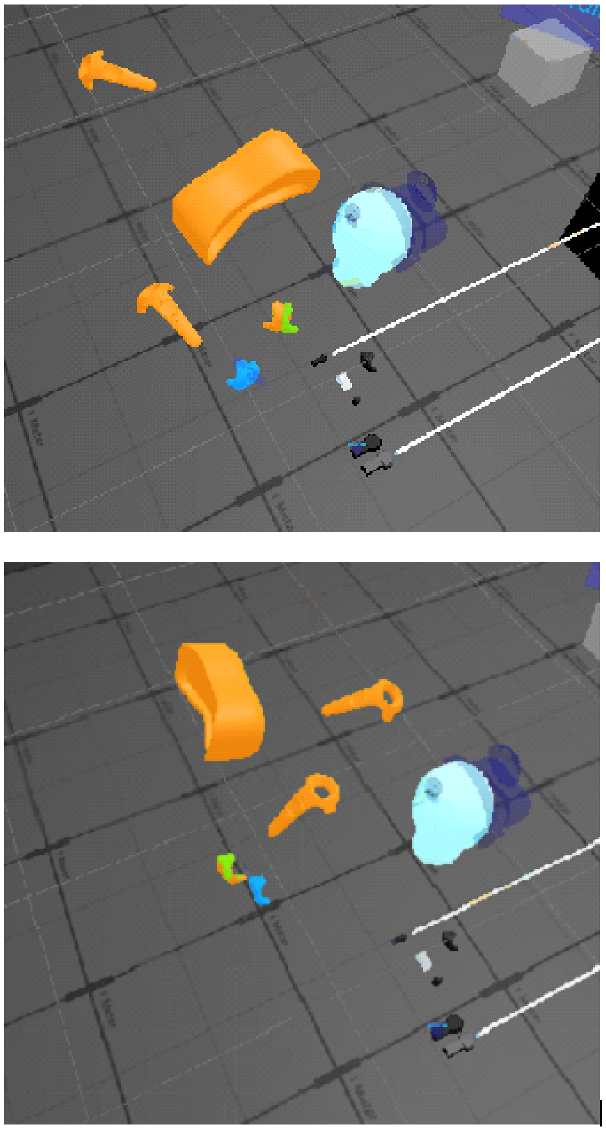

Figure 4: Predictions from MLP [Orange: Ground Truth, Green: MLP prediction, Blue: Naive prediction]

4.2 Machine Learning Architecture

Our Neural Network Classifier is a fully connected perceptron network with 4 layers. We use batch normalization between layers to avoid overfitting and a standard ReLU activation function. The output of the network is 4 confidence scores representing the probability that the input belongs to the indexed class.

We use Cross Entropy loss in training our neural network weights. Cross Entropy loss increases as the class probability output by the neural network diverges from the actual labeled class. We additionally apply L2 regularization to the loss in an effort to avoid overfitting on the training data. The network was trained for 20 epochs, with a learning rate of 1e-3.

4.3 Machine Learning Results

The training loss reaches 0.1 and training accuracy reaches above 95% correct classification, as shown by figures 7 and 8. However, the test or validation loss only reaches 0.55 and validation accuracy only reaches 85% correct classification, as show by figures 9 and 10. We will continue to gather more data and tweak the network

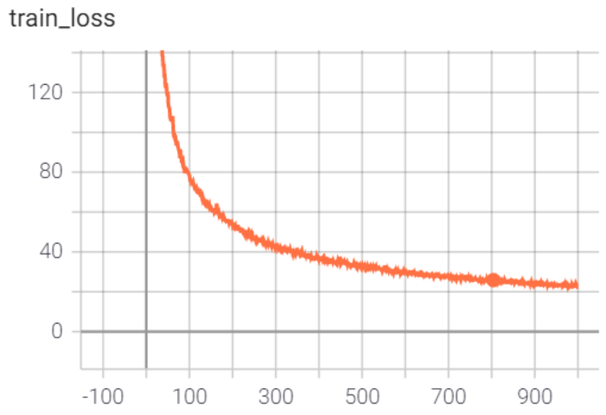

Figure 5: LSTM Train Loss

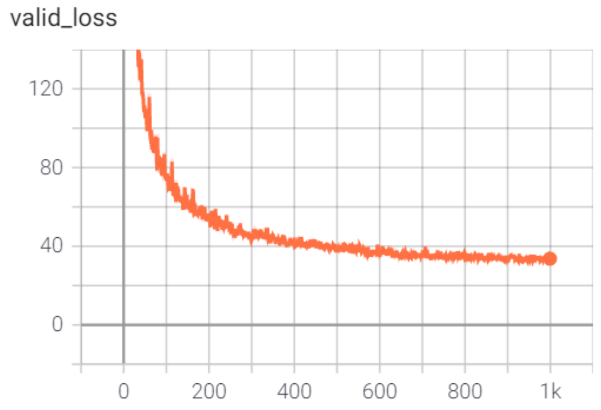

Figure 6: LSTM Valid Loss

hyper-parameters in order to bring the validation loss and accuracy in line with those of the training data.

Once deployed in Unity the trained model is able to correctly classify gestures near the user's ears. However, the classification performance noticeably decreases when the user is walking, crouching, lying down or in an otherwise unorthodox body position. This dip in performance is expected for the time being because the training data was gathered while users were stationary standing or sitting down.

4.4 ML-Agent

We are planning to try ML-Agent for Experiment1.

4.4.1 Observations

The Agent will have access to the headset's position and rotation as well as the controller's position and rotation. The observation will be set up as stackedVectors so we can utilize the history of the positions and rotations.

4.4.2 Action

The Agent will perform discrete actions that map to the enumeration of the Equipment Gestures (Up, Down, Forward, Backward).

4.4.3 Rewards

The agent will receive a reward of 1.0 for making the correction action (prediction).

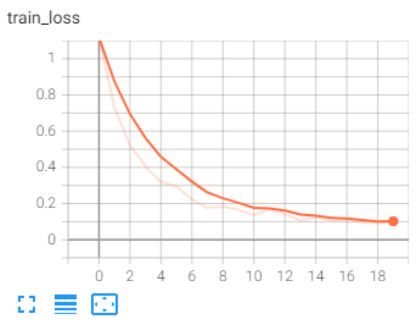

Figure 7: Gesture Recognition Training Loss

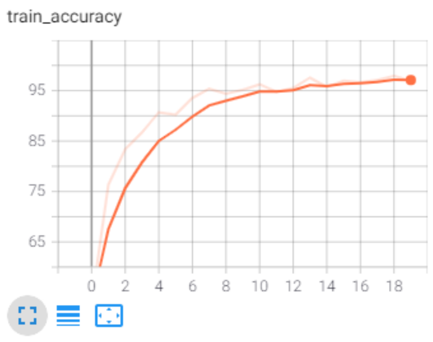

Figure 8: Gesture Recognition Training Accuracy

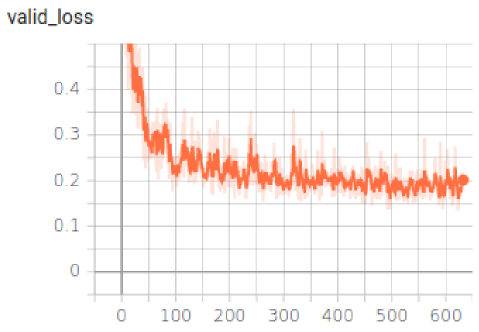

Figure 9: Gesture Recognition Validation Loss

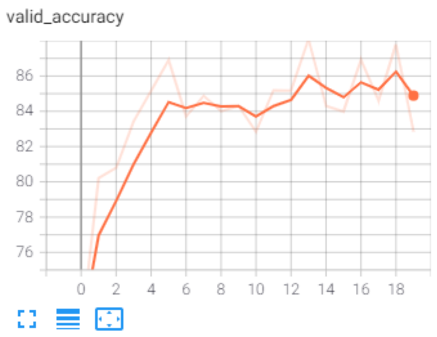

Figure 10: Gesture Recognition Validation Accuracy

4.5 Results

(We'll update with Experiment 1 Results using ML-Agent in the final draft.)

4.6 Conclusion

(We'll update with Experiment 1 Results using ML-Agent in the final draft.)

5 Experiment 2: Peripersonal Equipment Location

(We'll update with Experiment 2 using ML-Agent in the final draft.)

6 Acknowledgements

References

[1] J. Lee, A. Pastor, J.-I. Hwang, and G. J. Kim. Predicting the torso direction from hmd movements for walk-in-place navigation through deep learning. In *25th ACM Symposium on Virtual Reality Software and Technology*, VRST '19. Association for Computing Machinery, New York, NY, USA, 2019. doi: 10.1145/3359996.3364709

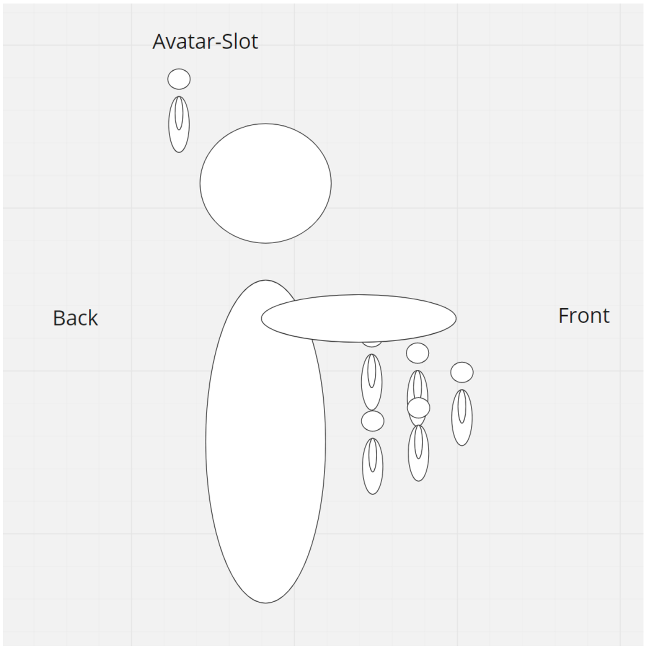

Figure 11: Avatar Dolls around the user's waist