Real-time Gesture Pattern Classification with IMU Data

Alex Fu
Stanford University
Computer Science Department
alexfu@stanford.edu

Yangyang Yu
Stanford University
Electrical Engineering Department
yyu10@stanford.edu

Abstract

Hand gesture recognition is an important task for virtual reality applications. In this report, we propose a neural network based classifier for real-time gesture recognition through IMU data inputs. In addition, we collect a training and testing data set from IMU and build a simple Unity scene for visualization of the classifier performance. We evaluate the recognition accuracy in real-time both quantitatively and qualitatively.

1. Introduction

Human hand gestures is a natural way of communicating ideas in daily life, especially when other natural communicating cues like speaking or listening fails. In this case, the movement and pose of a person's hands provide valuable information for the viewer to interpret what this person means.

Similarly, we can also use gesture recognition as a new interface between human and computer. Gesture recognition determines the user intent through the recognition of the hand gestures. In the past decades, many researchers have strived to improve the hand gesture recognition technology. Hand gesture recognition has great value in many applications such as sign language recognition, augmented reality (virtual reality), sign language interpreters for the disabled, and robot control.

In this report, we specifically look at how a real-time hand gesture classification could be applied to virtual reality applications. Current on-the-market virtual reality headsets often come with hand hold controllers that contain position and pose tracking. For example, the simplest version is the Google Daydream View Controller, as in Figure 1. A more complex version is Oculus Touch, shown in Figure 2. These hand controllers usually come with multiple buttons for specific controlling tasks, like volume, clicks, menu controls, etc. They also have internal position tracking and pose tracking. Even through these tracking mechanism varies in their implementation and complicity, they are in general fairly accurate and allow a third-party application developer to simulate a virtual controller corresponding to the real controller movements, as shown in Figure 3.

Figure 1. Google Daydream View controller, used with Daydream headsets.

Figure 2. Oculus Touch VR Hand Controller, used with Oculus Rift headsets.

Given these inputs from the hand controller, we can interpret the user intentions based on complex gesture patterns that are more than just a button click. Especially in

Figure 3. Google Daydream View controller, used with Daydream headsets.

Figure 4. Spells patterns defined in Harry Potter series.

many fiction and game applications, we may want to perform different tasks based on the user's hand gesture patterns. For example, in the world of Harry Potter, a wizard can draw specific patterns in the air as casting spells. Each pattern as shown in Figure 4 is a pre-defined spell and will trigger a specific task to be performed. In a virtual reality setting, we can utilize our hand controller as the wand. Given the position and pose tracking data from the controller, we can classify the gesture pattern performed by the user to determine what the user's intention is.

With the recent development in deep learning and neural networks, we propose a long-short term memory based model to classify gesture pattern inputs in real-time. We collect raw training data sets using Inertial Measurement Unit (IMU) to train this neural network model. We also build a simple Unity visualization scene to demonstrate this classifier. Finally, we evaluate on the prediction accuracy in real-time both quantitatively and qualitatively.

2. Related Work

Major previous research on gesture recognition utilizes Hidden Markov Model (HMM), Fuzzy Systems or neural networks. Hidden Markov Model [1] [3] is a mainstream statistical model. It is capable of modeling spatio-temporal time series where the same gesture can differ in shape and duration. The major problem is we cannot predict the start and ending time in real-time gesture recognition system for continuous gestures to extract isolated gestures.

There are also many research on fuzzy system based gesture recognition [4]. This approach employs a powerful method based on hyperrectangutar composite neural networks (HRCNNs) for selecting templates. Templates for each hand shape are represented in the form of crisp IF-THEN rules that are extracted from the values of synaptic weights of the corresponding trained HRCNNs. Each crisp IF-THEN rule is then fuzzified by employing a special membership function in order to represent the degree to which a pattern is similar to the corresponding antecedent part. When an unknown gesture is to be classified, each sample of the unknown gesture is tested by each fuzzy rule. The accumulated similarity associated with all samples of the input is computed for each hand gesture in the vocabulary, and the unknown gesture is classified as the gesture yielding the highest accumulative similarity.

Recent advance in deep learning provides us with new tools to solve gesture recognition problems. Neural network based models can capture complex patterns in the training data and models like long-short term memory [2] can preserve temporal information. [5] uses a 3D convolution LSTM to better capture gesture information and is able to obtain a high recognition accuracy.

3. Methods

In the absence of an on-the-market controller, we use an IMU as our data input source. Our project includes the following stages: collecting training and testing data, extracting features from IMU inputs, building neural network based classifier models, building test-time visualization in Unity.

3.1. IMU Data Readings

We use the VRduino, shown in Figure 5, as provided in this course as our data input source. VRduino contains an IMU that provides 9 degree of freedom sensor readings including gyroscopes, accelerometers and magnetometers. VRduino uses Arduino interface and is connected to PC through USB using Teensy.

In our project, the user can hold the VRduino chip in his/her hand and perform any hand gesture patterns. The sensor inputs from IMU will be streaming into local PC. We record the raw sensor inputs as training data set and use the raw data stream directly in real-time testing.

3.2. Training Data Collection

Using the VRduino chip as described above, we collect 1040 sets of data samples manually. Each data sample

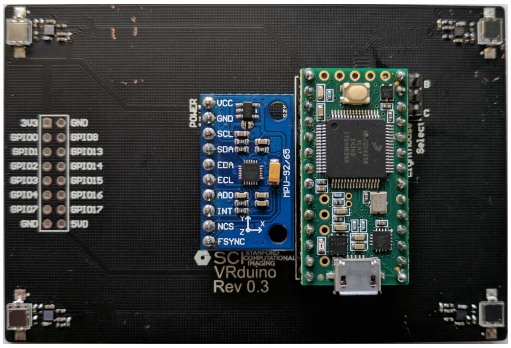


Figure 5. VRduino.

Features	Dimension	Accuracy
quaternion + gyro + acc	10	77%
quaternion + acc	7	61%
gyro + acc	6	75%
euler + gyro + acc	9	42%

Table 1. Different feature combinations. (gyro: gyroscope readings, acc: accelerometer readings, quaternion: quaternions calculated from raw IMU inputs, euler: euler angles calculated from raw IMU inputs)

is a sequence of raw IMU sensor readings that has a predefined start and ending time. Specifically, we record the data stream from VRduino into a text file after a keyboard input "start" and a keyboard input "end". We also include a label describing the gesture pattern for each data sample. Among the 1040 sets of data samples, we labels of circle, horizontal flick and vertical flick, each taking 30% of the total number of data samples.

3.3. Feature Extraction

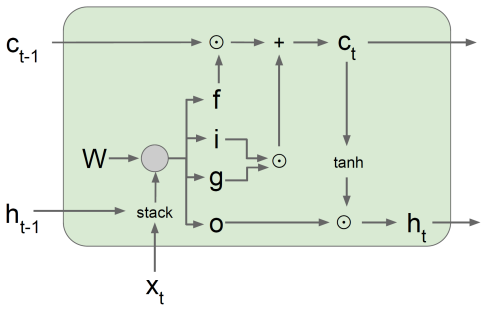
To classify a specific gesture pattern accurately, we need to take into account of both the pose and the position of the VRduino chip. We pre-process the raw sensor inputs to calculate the quaternions in order to track the rotation of the VRduino. We also normalize the gyroscope and accelerometer inputs.

We experiment with several combinations of quaternion and raw sensor inputs to determine which features work best with our model.

From Table 1, we can see the best performing features are the combination of quaternion, gyroscope and accelerometer data. This combination gives use 10 features to be used at each time step in the LSTM model.

3.4. LSTM Classifier

Since we want to preserve the sequential information we get from the input data, we choose to use standard LSTM cells in our model. As shown in Figure 6, each LSTM cell



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Figure 6. LSTM cell structure.

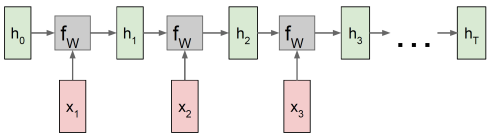


Figure 7. LSTM layer structure.

takes in the input x_t of all features at a specific time step, then the input x_t flows through four gates defined in this cell (t, f, o, g) with corresponding weights.

First, we process all our input sequences to be the same length by sampling within the start-end window. Then, each of our input data sample is a sequence of 10 dimensional feature arrays representing the status of VRduino at a specific time step. As shown in Figure 7, the LSTM layer is a sequence of LSTM cells with each cell taking its own data input.

We implement our model using Keras on top of Tensorflow, as in Figure 8. We first put together two identical LSTM layers. Then we use a Dense layer with ReLU activation function.

$$f = \text{ReLU}(Wx + b) \tag{1}$$

At the very end, we use Dense layer with softmax as a softmax classifier.

$$f = \text{softmax}(Wx + b) \tag{2}$$

Finally, we compute the cross-entropy loss. We train our model using Adam optimizer.

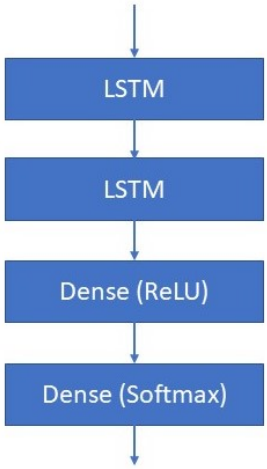


Figure 8. LSTM layer structure.

3.5. Test-time Unity Visualization

We implement a simple visualization in Unity to show how our classifier perform in real-time on IMU input data stream.

We stream IMU testing data from VRduino directly into Unity engine through serial port. We also take in keyboard input through another serial port into Unity engine to navigate through the scene and record the start and end time of a gesture performed by the user.

During test time, if the user's gesture is classified as "circle", the scene changes to red. If the user's gesture is classified as "horizontal flick", the scene changes to blue. If the user's gesture is classified as "vertical flick", the scene changes to green, as shown in Figure 9.

4. Evaluation

As shown in Figures 10, there is still a gap between the training accuracy and validation accuracy. Our model is likely over-fitting.

Thus, we introduce Dropout with a dropout rate of 50% into our model. Our final model achieve 77% accuracy on validation set. On average, we achieve 60% accuracy during test time.

Qualitatively, our classifier can predict the user's gesture with a good accuracy in real-time. However, the first several attempts are usually much more accurate then the later ones.

5. Discussion

There are several places we can improve on our current result.

First, we train and test on a very small data set due to the time and capacity limit we have on collecting the data ourselves, which leads to high over-fitting in our model. If

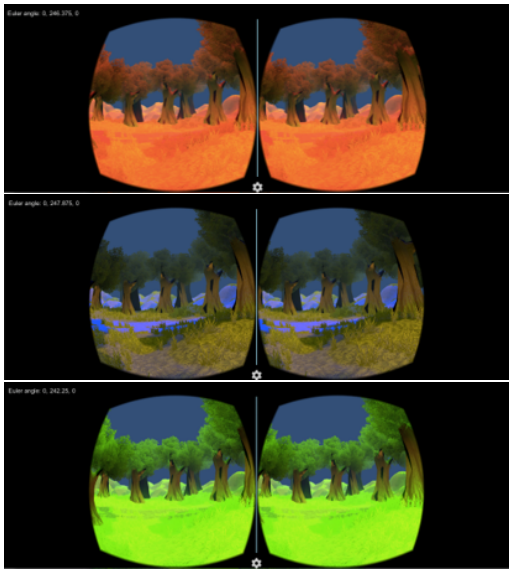


Figure 9. Example of visualization when a certain gesture is recognized. Different gesture will trigger different tasks.

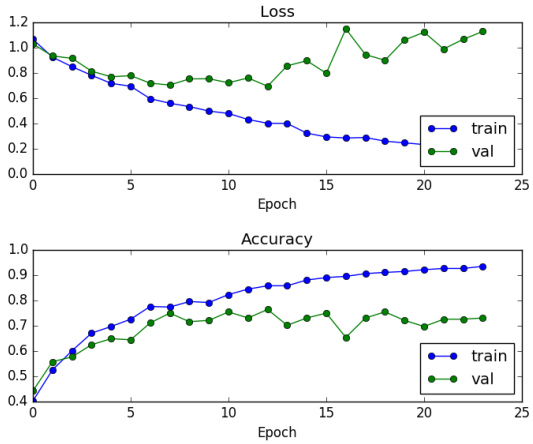


Figure 10. Loss and validation plot.

we could have a better system collecting training data, for example, build a real application that asks users' to perform certain gestures and obtain labels at the same time, we could have a much larger data set to train on.

Besides, our data set is also very uniformed and does not contain enough noise. This is because we generate the data set with only two persons and we tend to perform the same gestures over different attempts. The users would have a larger range of motions that could be understood as "flicks" or "circles" but our data set does not capture these variations. Our training set does not represent the testing input well. This is also a likely reason that leads to many misclassifications in test time.

Thirdly, during testing time, the classification results are significantly more accurate for the first few gesture inputs than the later ones. This is likely because of the drift in IMU readings. We could perform automatic reset after certain time intervals. Alternatively, we could correct for drifting during pre-processing time.

Fourth, currently, we use keyboard inputs to define the start and end time of a gesture. We could calculate the local gradients of input data and predict the start and end time of a gesture automatically. If this is implemented, we can then recognize different gestures in a continuous data stream, which would be much more applicable for applications like reading sign languages.

References

[1] M. Elmezain, A. Al-Hamadi, J. Appenrodt, and B. Michaelis. A hidden markov model-based continuous gesture recognition system for hand motion trajectory. In *2008 19th International Conference on Pattern Recognition*, pages 1–4, Dec 2008.

[2] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[3] H.-K. Lee and J. H. Kim. An hmm-based threshold model approach for gesture recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(10):961–973, Oct 1999.

[4] M.-C. Su. A fuzzy rule-based approach to spatio-temporal hand gesture recognition. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 30(2):276–281, May 2000.

[5] G. Zhu, L. Zhang, P. Shen, and J. Song. Multimodal gesture recognition using 3-d convolution and convolutional lstm. *IEEE Access*, 5:4517–4524, 2017.