A Real-time Hand Gesture Recognition System

Arun Ganesan
University of Michigan, Ann Arbor

Caoxie (Michael) Zhang
University of Michigan, Ann Arbor

Abstract

Hello World

1 Introduction

Natural user interface (NUI) is a new way for human to interactive with machines. Among numerous NUIs which include multi-touch screen, eye tracking and many others, hand gesture seems to be one promising candidate. In this paper, we design and evaluate a novel hand gesture recognition system to demonstrate that we are close to an actual production-level system. The reader should be noticed that we do not claim hand gesture is THE future user interface, and in fact there are some limitations for using hand gestures such as users may feel fatigues (in the movie Minority Report, Tom Cruise has to take breaks many time due to fatigue). Similar to many other new UI systems, we propose our system as one (interesting) way to interact with the computer. We do not claim that the system would replace mouse and keyboard and we leave the usability problem to future research as building hand gesture recognition system alone is quite challenging.

1.1 Design Goals

Our system is designed to maximize user experience. Moreover, our system differs from other existing systems in the following ways.

Just hands. The users do not need to attach any additional physical objects to use our systems. They just need to show up their hands. Many existing system such as [SixSense, MIT Minority Report, MIT color glove] require users to wear gloves or markers for the RGB camera to capture. We eliminate this since the user should not do anything more than showing their hands.

Real-time. Our system should run smoothly on a modern machine with a graphical card not just on high-end machines. The system should also recognize hand gesture at a high frame rate. Our desired frame rate 30Hz, although the current version has around 10Hz. In our design and implementation, one driving goal is to squeeze every milliseconds as possible.

No calibration. Our system should not require a new user to do anything to calibrate the system to be used to the user.

Robust and Accurate. Our system should have an accurate estimation of where the users' hands are and what gestures they use with low false positive. Moreover, the system should be insensitive to various background, user's location, camera position and other noise.

Arbitrary gestures. Our system should be able to easily incorporate new types of gestures that any developers would like to add. By training new gestures, the system can recognize arbitrary gestures, for example the American sign language.

1.2 Main Ideas

Our system would not be possible without the use of Microsoft Kinect for PC, which we are probably among the first to obtain it in February 2012. Kinect is a multi-purpose sensor including RGB camera, depth camera and audio sensor. The Kinect SDK has offered skeletal recognition, which is however far away from recognizing hand gesture. The SDK also provides raw pixels for the RGB image and depth image at a maximum frame rate

30Hz. We use the depth image for gesture recognition and both RGB and depth image for generating training samples. The depth image is the key factor that distinguishes our system from most existing systems that use RGB camera. The advantage of the depth image is that it offers an addition dimension, i.e. depth of each pixel that is not present in the RGB image. An illustrating example would be an object and its background has similar color but the depth of the two is drastically different.

Our system adopts a data-driven approach: machine learning as opposed to hand-crafted rule-based systems. The adoption of machine learning transfers the human intelligent efforts from design rules/algorithms to design informative features. With the features on labeled data, machine learning allows computers to learn the rule/algorithm automatically. The key advantages of using machine learning in our system are (1) easy to incorporate developer-defined gestures: developers just need to feed the system with the gesture images to be trained rather than deriving new algorithms (2) robust to various environments such as camera position, background, various size of the hands: developers just need to generate the gestures on various environments without worrying about anything more.

In a high-level overview, the system is separated into two parts: training and real-time prediction. Only the real-time prediction component is seen by the end users. In the training component, we use color gloves to generate massive labeled data of the depth image. Random forest is trained to achieve both real-time performance and high accuracy. In the real-time prediction component, each pixel in the depth image is predicted the type of the gesture using GPU and then the prediction outputs are pooled to propose the final position and type of a gesture. Notice that we do not use any temporal or kinetics information as the current simple design suffices for the hand gesture tracking. [Need to elaborate more?]

1.3 Contributions

We summarize our contributions as follows:

A system for real-time hand gesture recognition. We design and implement a complete real-time hand gesture recognition system based on machine learning. The system can be used as API for other applications to read the gesture in real-time.

An inexpensive way to generate labeled data. We color gloves to easily generate labeled data through the use of alignment of RGB and depth images. In [MSR], the authors use sophisticated computer graphics to generate training samples. We found this expensive and through the use of color gloves, developers can generate their customized gesture without difficulty. Notice that the end users do not need to wear any color gloves.

An computational insight about random forest and support vector machine (SVM). To the best of our knowledge, there seems to be no literature in comparing SVM and random forest from a computational perspective. We provide an in-depth comparison in the angle of performance rather than merely predictability as done in most machine learning literature.

Extensive experimental evaluations of the system. We conduct extensive experiments evaluation on the effectiveness of the machine learning approach, i.e., random forest we use in a large space of parameters. Interesting results reveals a deeper understanding of random forest.

Our work is made public at: `https://github.com/arunganesan/hand-gesture-recognition`.

2 System Overview

From the user point of view, our system is divided into two components: developers mode for training gestures and end-user mode for real-time prediction. In a high-level overview, developers using the developer mode supply the system with labeled data by wearing color gloves. Hence the developers hand the labeled image for the system to train. In the end-user mode, the users offer the raw image by showing up their hands, and the system will make predictions on every pixel using GPU and then pool every prediction to get an estimation of the gesture. Between the two modes, they share a component: feature extraction, which obtains the features for each pixel in the depth image. The implementation of feature extraction in the two modes are however different one is using CPU in an off-line fashion and the other is using GPU (end-user mode) in an on-demand fashion.

2.1 The Kinect Sensor

The Kinect sensor provides both raw depth image and color image at a maximum frame rate of 30Hz and with a resolution 640 times 480. Each pixel in the depth image represent the distance from the object to the cam-

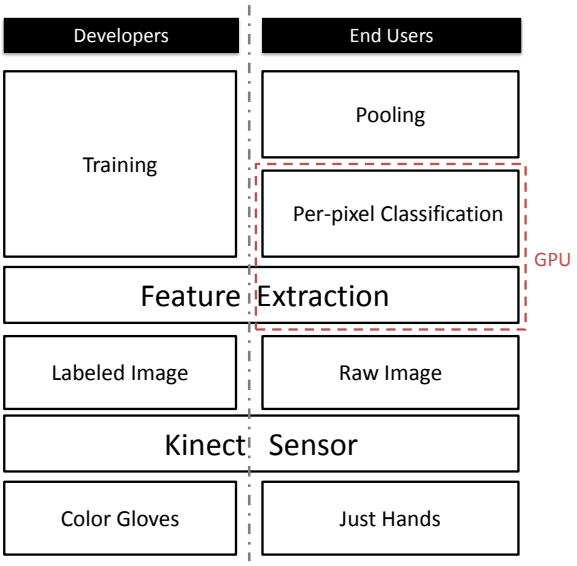

Figure 1: An overview of the system architecture

era. The error of the depth can be with several millimeters[TO VERIFY]. However, objects can have a shadow where some parts near the object do not have depth value at all.

2.2 Outline

In the following, we go through feature extraction and per-pixel classification in Sec. X, generating training samples in Sec. Y, pooling in Sec. Z, implementation in Sec. YY, our experience in Sec.Zz, experiment in Sec. XX and conclude in sec. XX.

3 Feature Extraction and Per-pixel Classification

In this section, we describe the main classification algorithm used in the system. At each frame, the system looks at the depth image, and predicts each pixel's type of gestures (e.g., open hand, close hand or background). The prediction result on every pixel will be fed into a pooling stage to propose the final gesture location and type. The driving reason for us to choose per-pixel classification is that it allows massive parallelism using GPU since each pixel is using the same predicting algorithm.

Figure 2: Offset pairs for a given pixel. In the left figure, the reference pixel is the center of the palm. In the right figure, the reference pixel is the center of the palm of the person that stands.

3.1 Feature Extractions

For each pixel $\mathbf{x}$, we extract a group of features. Each group say $\theta = \{\mathbf{u}, \mathbf{v}\}$ corresponds to the depth difference between two points offset from $\mathbf{x}$ normalized by its depth:

$$f_\theta(x) = d\left(\mathbf{x} + \frac{\mathbf{u}}{\mathbf{d}(\mathbf{x})}\right) - d\left(\mathbf{x} + \frac{\mathbf{v}}{\mathbf{d}(\mathbf{x})}\right) \qquad (1)$$

This feature extraction method is also used by [1]. The offsets $\theta = \{\mathbf{u}, \mathbf{v}\}$ are obtained randomly. In our design, we make those offsets to be uniformly sampled from a bounded circular area. As we can see from Figure 2, the offset pairs are located within the neighborhood of the palm and is adaptive to the depth. [[TODO: DO WE NEED MORE EXPLANATION?]]

3.2 The Classifier: SVM or Random Forest?

In the training samples each pixel is labeled (we would deal with how to generate massive labeled sample in Section 4) and we have to use a supervised learning methods. Two methods come into our decision of choice: linear SVM and random forest. Let us make the notation as follows: the number of training samples: l, the number of features: n, the number of classes (types of gestures): c. Our main criteria for choosing the best algorithm are (1) prediction time and (2) accuracy.

Linear SVM. In linear SVM, a model that is c number of weight vector length of n is trained. In prediction, the prediction is based on the dot product between the trained weight vectors and the feature vector. Therefore, the run-time complexity for predicting each pixel is $O(n \times c)$.

Random Forest. Random forest is built on an ensemble of decision trees that are trained on a bootstrap the training samples. Each node in the decision tree has one feature and a threshold to determine which branch to go to. Without pruning the decision tree, the depth of the tree is approximately as $O(\log l)$. Therefore $O(\log l)$ queries of features are made in a single decision tree when prediction. The run-time complexity for random forest is then $O(\log l \times n_{\text{tree}})$. In fact, we can prune each decision tree to limit its depth. [TO DO(how to prune)] Say the maximum depth is d_{tree}, the prediction complexity can go down to $O(d_{\text{tree}} \times n_{\text{tree}})$.

[[TODO: Show RF algorithm]]

Comparison. Let us do a simple calculation in which we use a realistic parameter setting. Suppose we have 2000 features, 3 classes, and using 3 trees with maximum depth of 20, the random forest is 100 times faster than linear SVM! Moreover, in our experimental evaluation, random forest has proven to be far superior than linear SVM in accuracy. Although linear SVM might use some advanced feature learning technique such as deep learning to achieve similar accuracy as to random forest, SVM is still too slow for us to adopt in the system. Inheriting from decision tree, random forest allows the system to extract features **on-demand**, which has been extremely crucial for real-time application as in the case of our system. The second author [[TODO: HOW ABOUT THE FIRST AUTHOR?]] is really surprised by this analysis since he used to believe linear SVM is unbeatable in practice.

Training. Training in linear SVM can be very fast as it can has a run-time complexity of $O(n \times l)$ and there exists technique to scale the training to distributed system [[TODO: Cite Michael's paper]]. Training in random forest, however, does not have an optimization-based foundation. We use brute force to determine the right feature and right threshold for each node in the decision forest. In determining the right threshold, we use grid search. In the training of random forest, it is highly recommended to put the training data in the main memory.

3.2.1 GPU For Real-time Prediction

There are $307,200$ (640×480) pixels in a frame. Each pixel will undertake $O(d_{\text{tree}} \times n_{\text{tree}})$ operations. We first tried to implement the random forest prediction using CPU. It takes about 1 minute to process a frame! GPU has to be used. In fact GPU is more suitable for high-parallelism and high-latency jobs, while CPU is better fit for low-parallelism and low-latency jobs. In the system, each pixel has fired a thread to make a prediction since the algorithm for each pixel is the same (i.e., the random forest is a sequence of branch operations).

Running the prediction algorithm using the CPU proved to be very slow. For a 640×480 image, our system took 2.5 minutes to classify all the pixels. Given that this problem is highly parallelizable, we turned to the GPU. Re-implementing the prediction algorithm with the GPU reduced the prediction time from 2.5 minutes to 400 milliseconds - a 99.7% speedup!

3.3 GPU performance enhancements

4 Generating training samples

Color glove approach. To simplify the data acquisition step, we used cropping, depth thresholding, and a single colored glove. We used 4 gestures, giving a total of 5 classes including the background. We collected 400 training samples evenly spread across the gestures.

5 Pooling

- mean versus median – mean is prone to outliers messing up the location. median is more resistant to that.
- majority gesture – if the hand is far away, this leads to the noisy labels being
- k-meniods – can be used to support multiple hands at once. and can be used to filter out the noise.

6 Implementation

* OpenCL

* Show actual system picture here

6.1 Refinements

* Changing double to float saves a lot of space * Use EC2

7 Experimental results

7.1 Classification algorithm experiments

8 Experience

8.1 Limitations

Our current system is not without limitations. First, the real-time prediction component cannot achieve a frame rate at 30Hz but at about 10Hz.

9 Related Work [[TODO: Need more]]

There are two main techniques in hand gesture recognition - appearance based and model based. These are akin to probabilistic models of classification and generative models of classification respectively. **Appearance based** approaches read the pixels from the camera and build classifiers to label that as belonging to a finite set of classes. The main limitation of this technique is that the set of labels is finite and fixed ahead of time. The advantage of appearance based approaches is their implementation is often extremely fast and therefore suited for situations where live classification is important. Examples of appearance based techniques can be found in [1, 2]. **Model based** approaches start with a set of hypotheses of the final classification based on rules of the object being classified. For instance, in the case of hand gestures a hypothesis can be a particular orientation of the joints. An advantage here is that the hypotheses can be generated from an infinite space of possible classifications. The main disadvantage of model-based techniques is that they are often computationally expensive. In addition, model-based approaches tend to be very complex. An example of a model based technique can be found in [3].

10 Conclusion

References

[1] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, A. Blake. Real-time human pose recognition in parts from single depth images. CVPR, 2011.

[2] R. Wang and J. Popović. Real-time hand-tracking with a color glove. In Proc. ACM SIGGRAPH, 2009.

[3] I. Oikonomidis, N. Kyriazis, and A. Argyros. Efficient model-based 3D tracking of hand articulations using kinect. In BMVC, Aug 2011.

[4] V. Lepetit, P. Lagger, and P. Fua. Randomized trees for real-time keypoint recognition. In Proc. CVPR, pages 2:775-781, 2005.

[5] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification Journal of Machine Learning Research 9(2008), 1871-1874.

[6] B. Schneiderman. The eyes have it: a task by data type taxonomy for information visualizations. Visual Languages, 1996.

[7] D.A. Keim. Information visualization and visual data mining. Visualization and Computer Graphics, IEEE Transactions. Vol 8, no.1, pp. 1-8, Jan 2002.

[8] W. Stuerzlinger, C Wingrave. The value of constraints for 3D user interfaces. Dagstuhl Seminar on VR, 2010.

[9] M. Hoffman, P. Varcholik, and J. LaViola. Breaking the status quo: improving 3D gesture recognition with spatially convenient input devices. IEEE VR, 2010.

[10] V. Bystritsky. ALGLIB. 14 Aug 1999. Web. `http://www.alglib.net`.