Project Report On

Interacting with software using gestures

"A dissertation submitted in partial fulfillment of the requirements of 7th Semester 2017 Project-I (CS-794) examination in Computer Science and Engineering of the Maulana Abul Kalam Azad University of Technology"

Submitted by

Aditya Jain (10200114003)
Pallab Kumar Ganguly (10200114025)
Sauradip Nag (10200114044)
Swati Ghosh Hazra (10200114056)

Under the guidance of

Mr. Tamojit Chatterjee
Software Engineer
Indeed

Department of Computer Science and Engineering
Kalyani Government Engineering College
(Affiliated to Maulana Abul Kalam Azad University of Technology, West Bengal)
Kalyani - 741235, Nadia, WB

কল্যাণী গভঃ ইঞ্জিনিয়ারিং কলেজ
KALYANI GOVERNMENT ENGINEERING COLLEGE
(GOVERNMENT OF WEST BENGAL)
KALYANI, NADIA, PIN - 741 235, WEST BENGAL

Memo No. :

Date :

Certificate of Approval

This is to certify that this report of B. Tech 7th Sem, 2017 project, entitled **"Interacting with Software using Gestures"** is a record of bona-fide work, carried out by **Aditya Jain, Pallab Kumar Ganguly, Sauradip Nag, Swati Ghosh Hazra** under my supervision and guidance.

In my opinion, the report in its present form is in partial fulfillment of all the requirements, as specified by the ***Kalyani Government Engineering College*** and as per regulations of the ***Maulana Abul Kalam Azad University of Technology***. In fact, it has attained the standard, necessary for submission. To the best of my knowledge, the results embodied in this report, are original in nature and worthy of incorporation in the present version of the report for the Project-I (CS-794) 7th Sem B. Tech programme in Computer Science and Engineering in the year 2017.

Guide / Supervisor

Kousik Dasgupta

Dr. Kousik Dasgupta
Assistant Professor
Department of Computer Science and Engineering
Kalyani Government Engineering College

Examiner(s)

Head of the Department
Computer Science and Engineering
Kalyani Government Engineering College

Phone : +91-033-25821309 (Office), +91-9830040558(Mobile) E-mail: kousik.dasgupta@kgec.edu.in

INSTITUTE LETTER HEAD (FOR EXTERNAL GUIDE)

Certificate of Approval

This is to certify that this report of B. Tech 7th Sem, 2017 project, entitled **"Interacting with Software using Gestures"** is a record of bona-fide work, carried out by **Aditya Jain, Pallab Kumar Ganguly, Sauradip Nag, Swati Ghosh Hazra** under my supervision and guidance.

In my opinion, the report in its present form is in partial fulfillment of all the requirements, as specified by the ***Kalyani Government Engineering College*** and as per regulations of the ***Maulana Abul Kalam Azad University of Technology***. In fact, it has attained the standard, necessary for submission. To the best of my knowledge, the results embodied in this report, are original in nature and worthy of incorporation in the present version of the report for the Project-I (CS-794) 7th Sem B. Tech programme in Computer Science and Engineering in the year 2017.

Guide / Supervisor

Tamojit Chatterjee

Mr. Tamojit Chatterjee

Software Engineer
Indeed

Table of Contents

Abstract

The long term object of the project is to develop software to improve Human-Computer Interaction (HCI) with the help of computer vision. Our aim is to provide inputs to a computer by means of hand gestures. For this purpose, a set of gestures is mapped to certain inputs, for instance, a swipe-down gesture could indicate minimising the active window.
The computer needs to therefore be able to detect these gestures accurately in order to provide a smooth experience to the user. The first thing would be to detect the hand, then to detect the change in position, attitude of the hand, and then to classify these changes as a gesture. Finally, this gesture has to be interpreted as input to the computer to realise our long-term objective.
We have decided to divide the entire project into phases. In the first phase, we intend to accurately detect and thereby extract the image of a hand from a picture as accurately as possible. For this purpose, we created our own dataset which is formed from some 2000 photographs of hands taken from volunteers and some 4000 images not containing hands, which are then permuted to create a training set. In addition to this we made a small test set to verify, and evaluate the performance of various classifiers.
On the aforementioned training set, we split it into three portions corresponding to three window sizes, i.e., corresponding to various sizes of hands. This is to enable the classifier to correctly detect hands of varying sizes. For each such training set, the classifier was trained using OpenCV in Python, using a HAAR-like cascade library provided by OpenCV. For each such training set, we obtain a XML file which contains the various weights for the classifier.
After the training is completed, the classifier was tested against a video feed, where initially we found unacceptable levels for false positive detection, i.e., a lot of negative portions were classified as positive. To remedy this, a different window size was used, and many of the parameters of the training as well as the detection functions were tweaked to improve the results.
Finally, for a quantitative description of results, the test set was run on the classifier. Various parameters were used, and the accuracy found out for each parameters. We then plotted the accuracy against these parameters to conclude on the values of the parameters to be taken. We are still in the process of perfecting this classifier.
In the next phase we intend to create a second set of classifier which can distinguish between an open palm and a closed fist. This report primarily deals with the work done in the first phase of the project, i.e., hand detection

Literature Survey

Online Human Gesture Recognition from Motion Data Streams by Xin Zhao, Xue Li, Chaoyi Pang, Xiaofeng Zhu, Quan Z. Shen:

The key concept for the work of this paper is that the movement of the human skeleton is sufficient for distinguishing different gestures performed by the human.

Learning Stage: Training dataset is captured by depth-cameras, having 3D joint positions of the human skeletons. This set is scanned to produce a motion data stream. The data streams are manually segmented to gesture instances which are further clustered into a dictionary of motion templates (1D time dependent function representing distance value of two joints). Each dimension is clustered separately for separate movements of the body, and the centroid of the cluster approximates the motion of that body part.
Prediction Stage: The human skeletons are captured using the depth-camera, and are transformed into motion data streams, and on to feature vectors. Then that feature vector is used by the classifier to get the label for the unknown gesture.

This approach also claims to have a much higher accuracy, even with a simple linear regression, and also claims to have a superior latency compared to existing systems.
This approach was found to be unsuitable for our purposes, it was not pursued any further.

Hand gesture recognition using a real-time tracking method and hidden Markov models, Feng-Sheng Chen, Chi-Ming Fu, Chung-Lin Huang:

In this paper, a hand gesture recognition system to recognize continuous gesture before stationary background is introduced. The system consists of four modules: a real time hand tracking and extraction, feature extraction, hidden Markov model (HMM) training, and gesture recognition.

Robust Real-Time Face Detection, Paul Viola, Michael Jones:
This paper describes a face detection framework that is capable of processing images extremely rapidly while achieving high detection rates. There are three key contributions. The first is the introduction of a new image representation called the “Integral Image” which allows the features used by our detector to be computed very quickly. The second is a simple and efficient classifier which is built using the AdaBoost learning algorithm to select a small number of critical visual features from a very large set of potential features. The third contribution is a method for combining classifiers in a “cascade” which allows background regions of the image to be quickly discarded while spending more computation on promising face-like regions.
This paper was found to be most suitable for our work.

Methods Pursued

Classification Problem

We recognise that our problem is essentially a classification problem. We are given an image and we have to find out whether it contains a hand (positive) or whether it does not (negative). For this problem we have to therefore collect some features on the basis of which we can classify the images. We have to decide which are the features that will classify the positives from the negatives as best as we can. We say that a feature is a set of characteristics which will enable us to distinguish the positives from the negatives. Typically, the features in a computer vision problem are the result of some kind of operation on an image. In the object detection framework, the features are HAAR-like features.

Object Detection using Viola-Jones Framework

First, a classifier (namely a *cascade of boosted classifiers working with haar-like features*) is trained with a few hundred sample views of a particular object (i.e., a face or a car), called positive examples, that are scaled to the same size (say, 20x20), and negative examples - arbitrary images of the same size.

After a classifier is trained, it can be applied to a region of interest (of the same size as used during the training) in an input image. The classifier outputs a "1" if the region is likely to show the object (i.e., face/car), and "0" otherwise. To search for the object in the whole image one can move the search window across the image and check every location using the classifier. The classifier is designed so that it can be easily "resized" in order to be able to find the objects of interest at different sizes, which is more efficient than resizing the image itself. So, to find an object of an unknown size in the image the scan procedure should be done several times at different scales.

HAAR Cascades

The word "cascade" in the classifier name means that the resultant classifier consists of several simpler classifiers (*stages*) that are applied subsequently to a region of interest until at some stage the candidate is rejected or all the stages are passed. The word "boosted" means that the classifiers at every stage of the cascade are complex themselves and they are built out of basic classifiers using one of four different boosting techniques (weighted voting). Currently Discrete Adaboost, Real Adaboost, Gentle Adaboost and Logitboost are supported. The basic classifiers are decision-tree classifiers with at least 2 leaves. Haar-like features are the input to the basic classifiers, and are calculated as described below.

HAAR-Like Features

Features for the classifier are calculated as follows:

Each of the HAAR-kernels are convoluted over the image, and for each position of the kernel,

Value = Σ (pixels in black area) - Σ (pixels in white area)

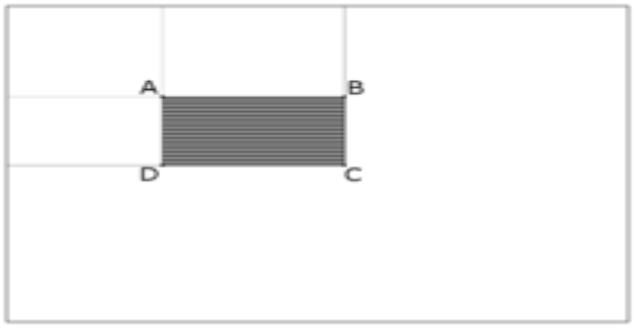
Now the number features generated thus as enormous. For even a 24x24 window, we can get almost 160000+ features. For each window, calculating the sum over the window can be extremely time-consuming. Therefore a algorithm called Integral Area Method is used to speed up computation.

Integral Area Method

Integral images can be defined as two-dimensional lookup tables in the form of a matrix with the same size of the original image. Each element of the integral image contains the sum of all pixels located on the up-left region of the original image. Any rectangle can be computed with just four lookups:

Sum = I(C) + I(A) - I(B) - I(D).

where points A, B, C, D belong to the integral image I, as shown in the figure.



Thus the time involved in calculating the sum of pixels is reduced substantially.

Adaptive Boosting

With over 160000 features, it would be extremely computationally expensive to train a classifier. So to reduce the number of features to a manageable number, the Adaptive Boosting (AdaBoost) algorithm is used. In this algorithm, initially all the training images are assigned an equal weight. Now for each feature, if the classifier classifies an image correctly, its weight is reduced, whereas if an image is incorrectly classified, its weight is increased. Thus we remove certain features and retain the ones which can classify trickier points.
The final classifier is a weighted mean of the selected features.

Implementation using OpenCV in Python

a. Choice of parameters for training classifier

1. **Choice of Dataset:** We considered *Standard ImageNet Database* as our Dataset for Training and Hand Samples were obtained from Key frames of Hand Videos taken from Smartphone Cameras.

2. **Image Size:** Initially, we trimmed *all the Negative Images to size 100 x 100*. We also trimmed the Hand Samples down to 30x30, 40x40 and 50x50, and we superimposed and observed that higher the Hand Sample Dimension, Higher the Accuracy, so we chose *50 x50 Hand Samples* and Superimposed on 100 x100 Negative Images to Get Positive Images. We gray scaled all the Positives and Negative Images and prepared them for Training.

3. **Number of Positives and Negatives:** We took around 3000 negative Images and we took 4-5 samples of Hand Image initially and superimposed each hand in different position, angles over the 600 negative Images out of 3000 Negatives, we did the same with the rest of the Hand Samples. So in Total we got ~3000 Superimposed Images with varying background. We consider this Superimposed Images as Positive Images. According to [4], t*he ideal ratio of Positive and Negative Image for Training the HAAR Cascade Classifier is 1:2*. So, we took *number of Positive Images as ~ 1500 and number of Negative Images as ~ 3000.*

4. **Number of Stages:** In our Observation, the Classifier has Higher Precision if we fix the *number of Stages > 15,* because higher the number of Weak Classifier, finer the features .Very Less and Correct features will be passed on to the next stages

b. Choice of parameters for testing classfier

Haar Classifier is tested by using the detectMultiScale() Object of OpenCV.Below we discussed some of the Testing Parameters of DetectMultiScale Object

1. **Input Image:** Input to this Object is gray scaled Key frames of Live Video Stream taken from Webcam of Laptop

2. **Scale Factor:** Scale Factor Parameter of DetectMultiScale specifies how much the image size is reduced at each image scale. According to our Observation, we concluded that if we increase the scale of the image > 1.05, we lose out on Accuracy. Scale Factor 1.x means we're using a small step for resizing, i.e. reduce size by x %, we increase the chance of a matching size with the model for detection.

3. **No of Neighbours :** This Parameter of DetectMultiScale() specifies how many neighbors each candidate rectangle should have to retain it. This Parameter depends on the size of Classifer used . We observed in our testing that , if minimum no of Neighbour Parameter value > 20 for a 30 x 30 Classifier , then Accuracy Drops . It gives best Result when t*his Parameter Value is 10*

4. **Window Size :** Window Size Parameter of DetectMultiScale() specifies the Possible size of Window the Classifier will scan over the Input Image in search of Hand Object.There are 2 variants of WindowSIze available :

   a. ***min_Size :*** This is a parameter which specifies that the detector window size will scan for Hand objects whose Minimum Size is > than min_Size else it will ignore it .In our testing we saw that if min_size=(60,60) I.e 60 x 60 window size, then Classifier can identify Hand far away in a Video captured using Web Cam of Laptop in room Environment

   b. ***max_Size :*** This is a parameter which specifies that the detector window size will scan for Hand objects whose Maximum Size is <than max_Size else it will ignore it . In our testing , we did not fix this parameter as in a Video Stream where Hand is close to the WebCam , it cannot be estimated ,so by not fixing this parameter , we are providing a flexibility to detect hands irrespective of distance from camera

Results

The performance metric used for results is accuracy. We have used accuracy because it is simple to compute, and because the test data (discussed below) was not skewed unfairly.

Results from various parameters

For evaluating the metric, we created ~200 samples using opencv_create samples. These samples were created from new data, never used before. The arguments to the function detectMultiScale were varied and accuracy plotted. The arguments were varied independently of each other. The results are summed up below:

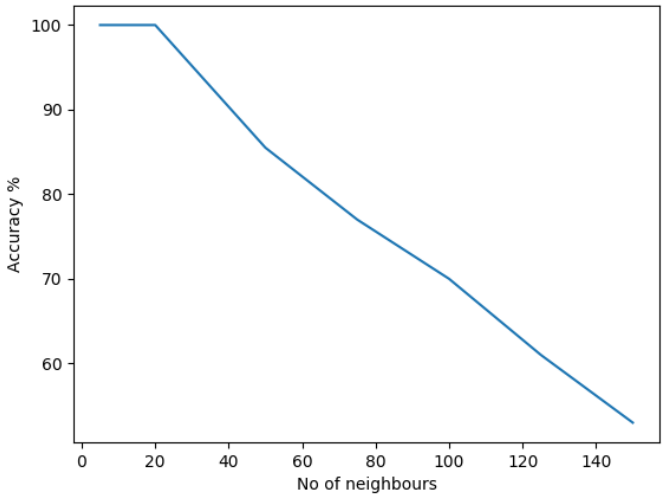


Fig: Plot of accuracy versus no. of neighbours

No. of neighbours	Accuracy %
5	100
10	100
15	100
20	100
50	82.5
75	74
100	67
125	58
150	50

Table: Accuracy vs. no. of neighbours

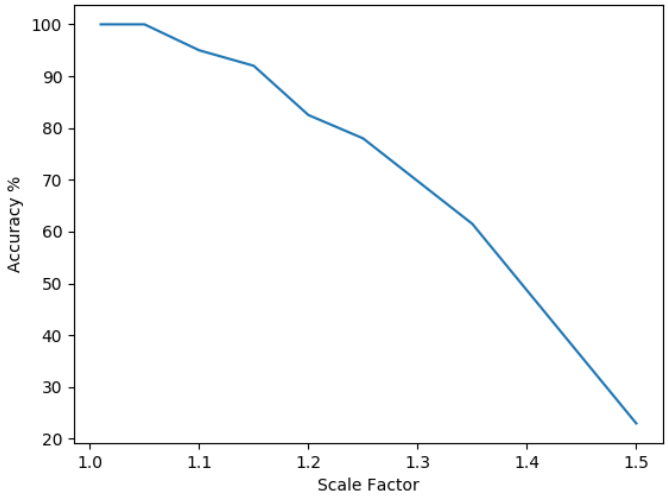


Fig: Plot of accuracy versus scale factor

Scale Factor	Accuracy %
1.01	100
1.03	100
1.05	100
1.1	93.5
1.15	90
1.2	81.5
1.25	75.5
1.35	60
1.5	22

Table: Accuracy vs. no. of neighbours

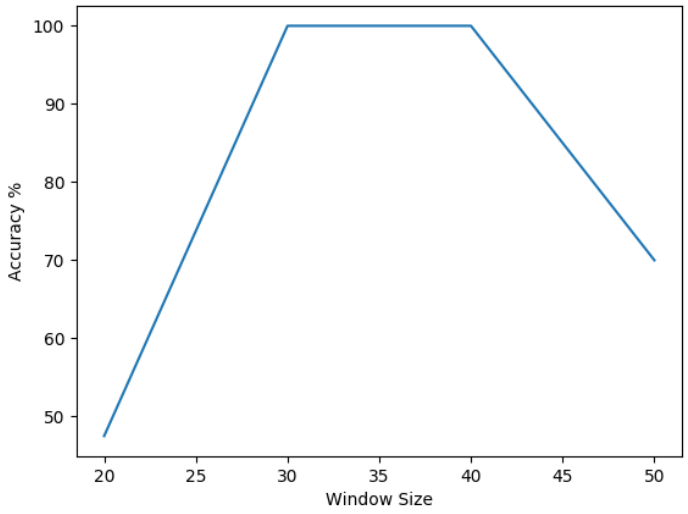


Fig: Accuracy versus Window Size

Window Size	Accuracy %
20x20	47.5
30x30	100
40x40	100
50x50	70

Table: Accuracy vs. Window Size

Explanation of results

From these figures, we can conclude that generally, increasing the scaling factor, and the number of neighbours leads to a reduction in accuracy. A possible explanation for this is when the number of neighbours is very large, the space between the two detected points is even larger than the size of the image itself.

Also, as we increase the extent to which the image is scaled, we lose out some windows because of the restriction on window size and therefore the accuracy falls off.

On increasing the window size, the accuracy generally should improve, because as the window size increases, more information is available to the classifier, and the probability that a hand will get detected should increase. However we observe that in our tests, the accuracy falls remarkably at windowSize=50. This is probably because the test set contained was composed entirely of 30x30 images, so the 50x50 classifier missed a sizeable proportion of them. We are still working to improve this.

We also tested the classifier against video input from a video camera. One such result is demonstrated here.

Fig: Result of video from camera

Work Ahead

The next work is to further refine the hand detection classifier to make it usable for subsequent stages in our project. This will include further testing of previous classifiers, by tuning the various parameters discussed above, as well as creating new ones from more general and cleaned data, to improve classification.

For gesture recognition from video stream, first we need to extract the hand from the video stream. Thereafter we plan to use a second classifier to distinguish between an open hand versus a closed one, count the number of fingers. Based on these classifiers we will be able to detect gestures performed by the hand, and associate that gesture with certain input to a computer.

References

1. P. Viola, M. J. Jones, "Robust Real-Time Face Detection", in *International Journal of Computer Vision 57(2)*, 137–154, 2004.ō

2. X. Zhao, X. Li, C. Pang, X. Zhu, Q. Z. Sheng, "Online Human Gesture Recognition from Motion Data Streams", in *Australian Research Council (ARC), Discovery Project DP130104614.*

3. F. S. Chen, C. M. Fu, C. L. Huang, "Hand gesture recognition using a real-time tracking method and hidden Markov models", in *Institute of Electrical Engineering, National Tsing Hua University, Hsin Chu 300, Taiwan, ROC*, March 2003

4. https://pythonprogramming.net/haar-cascade-object-detection-python-opencv-tutorial/