HandTroller

A robust and accurate static hand gesture recognition system, which allowing user to input their hand gestures in a complex background with various orientations, for controlling various PC game

Carnegie Mellon University

5000 Forbes Avenue
Pittsburgh, PA 15213

(412)268-2464

(412)268-6345

12/8/2008

Bingran Liu

Ke Wang

Fu-Chieh Chuang

Contents

1. Introduction

The interaction between human and computer develops dramatically in recent years. Mouse and keyboard have been widely used and well accepted as standard input devices. However, hand gesture, as a more convenient way to input instructions, has been well implemented into computer language. In out project, we want to use a webcam to recognize human hand gesture in a complex background with satisfactory matching rate. And then use this input signal to control a pc-based mini snake game.

We are going to implement a relatively robust and accurate static hand gesture recognition system allowing user to input their hand gestures in a complex background with various orientations. In order to do that, hand image extraction, feature extraction and image matching are required. Detailed discussion will be listed below.

2. Previous Projects and Candidate Algorithms

Among previous 18-551 projects, the most relevant one is G3 2003 'Handslation' (They also give us ideas on our project name). The aim of the previous project is to recognize American Sign Language alphabet to facilitate the communication of people with hearing problems to outside world. To realize it, they use webcam to acquire raw data of each letter and build 26 MINACE filters (one for each letter).When a new letter is given, they compare the correlation results of each filter and the given image to recognize the new letter as the filter generating the highest correlation value. At this point, we can see that there is a mint of pattern recognition notion in their project. We adopt the same method of acquiring data. But we followed the basic concepts of pattern recognition through our project from feature extraction to classifier training to tackle this problem. As a result, by delicate feature selection, our algorithm only needs one 5-dimension feature vector to represent each gesture, which will save substantial amount of computing time as well as energy and eventually make our algorithm robust in embedded real-time scenario.

Gradient processing is a useful way to get good feature of an image, which has wide application in texture-rich image recognition. A hand gesture image is generally considered scarce in texture. However hand gesture image is rich in orientation information at the boundaries between foreground and background. Global orientation histogram used as hand gesture features is brought forward firstly in [2]. The algorithm is basically pretty simple. It calculates the gradient of each pixel and records it in an orientation histogram vector whose dimension is determined by how large the angle been used. However, due to its lack of concentrating on local features, the feature vectors cannot be distinguished between certain different gestures, thus it has a limited alphabet for hand gestures. [3] reported the correctness is around 75%. By augmenting local vector features as well as increasing the dimension of feature vector space, hand gesture recognition is successfully implemented in [1]. However, the algorithm in [1] is not

friendly to DSP implementation considering the large volume of data. In our project, due to some inconsistency in our rotation algorithm, we didn't adopt local orientation histogram to get hand gestures' features. And basically global information is adequate to make our project robust.

3. Algorithm Design

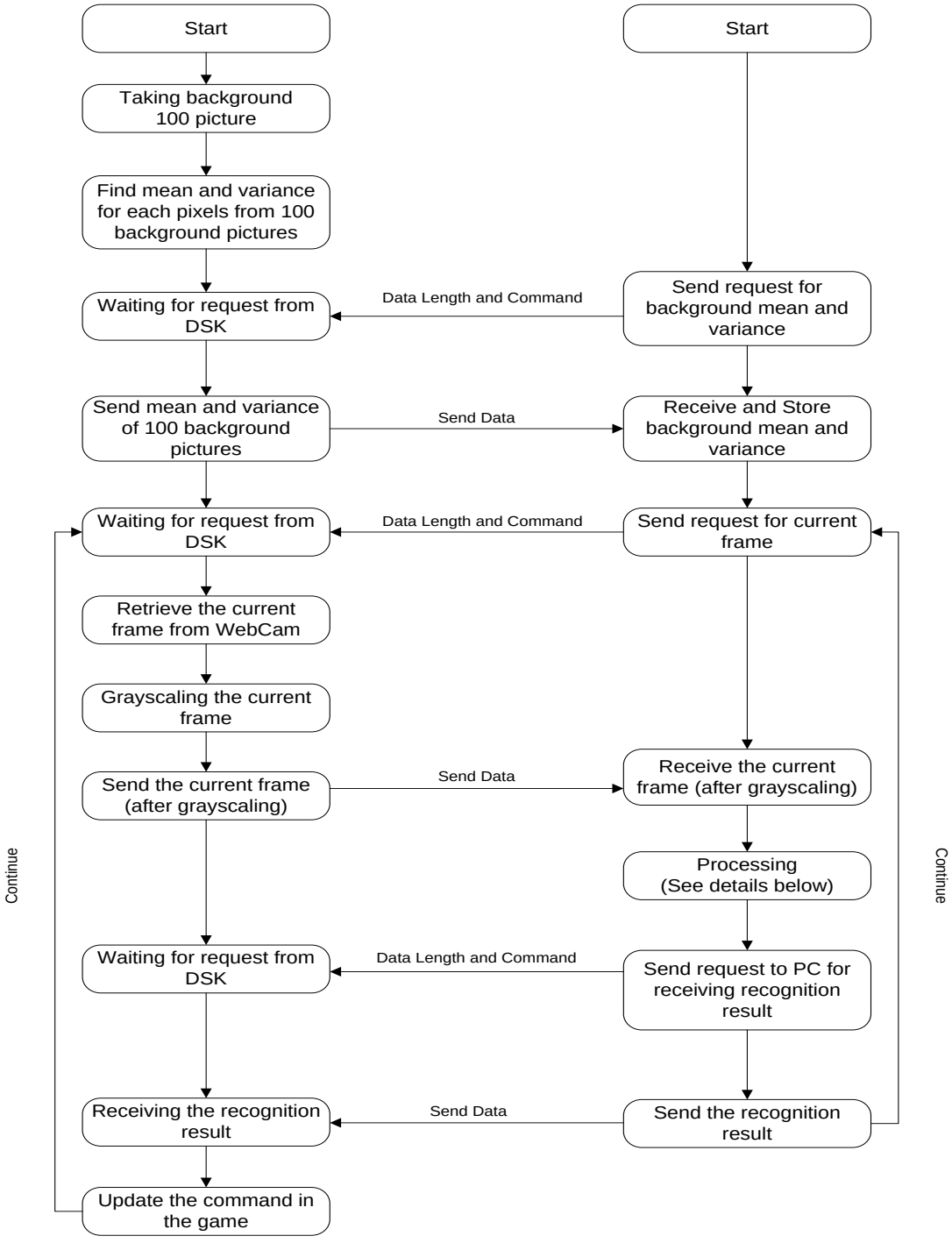
Decided on adopting Global Orientation Histogram to extract feature of the raw image, we first consider the resolution of the raw data. In previous project, the resolution is 64*64, and in [1] 80*100 resolution for hand region is used. The image captured by webcam is generally set to 320*240, and we think resample it to 160*120 will be a good choice since according to our demonstration settings, hand gesture is usually located in a square with side length varied from 80 to 100 around the center of the image.

Generally, for a given gesture there are mainly three different variances: in-plane translation, in-plane rotation and out-plane rotation. In our settings, we assume that user will place his or her hand on desk to give instructions and thus can ignore the effect of out-plane rotation. Histogram is by nature immune to in-plane translation, however we still have processes in the algorithm to find the center of the palm and place it at the center of the image in order to ease following rotation process as well as give us the potential to use local histogram features. Last but not least, as for in-plane rotation, we use rotation adjusting algorithm to offset its great influence on orientation histogram features.

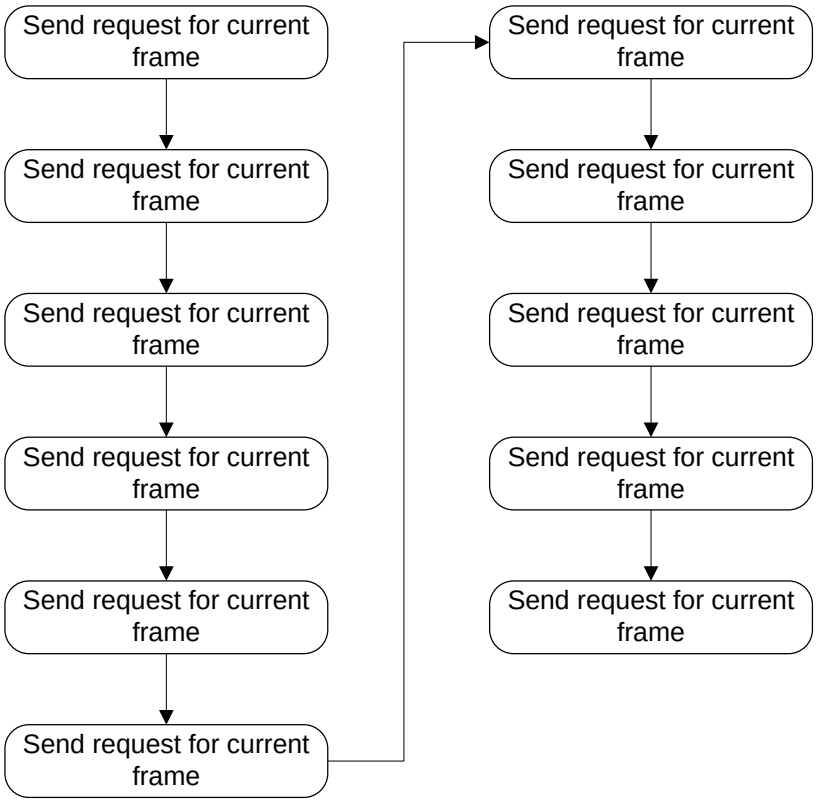
Since the both centriod finding algorithm and rotation adjusting algorithm are sensitive to how much the wrist shows in the image, for some gestures we get different variants. It is common in pattern recognition field to treat these variants as different gestures and have different training sets for each of them, which will be quite tedious and take substantial amount of time. Instead, we simply tune our matching algorithm to consider the variants.

4. Data Flow Chart

4-1. Overall Data Flow



4-2. Detailed Flow Chart on DSK



5. Detailed Algorithm

5-1. PC-Side Processing

Due to illumination noise and limited sensitivity of the camera, there will be variance for each pixel. Since back ground image is critical to our following processing, we need relatively accurate data. So on PC side we will take 100 pictures to get the mean of the background. We prefer to do this process on PC side because the memory requirement is high (19.2k*100 =1.92M), but the computation is not intensive. On GUI, the camera will white balance the image first, so the system will wait for 8 seconds to take the background images. Then, each frame (after re-sampling and grayscale) is transferred to the DSK to do the further processing.

5-2. DSK-Side Processing

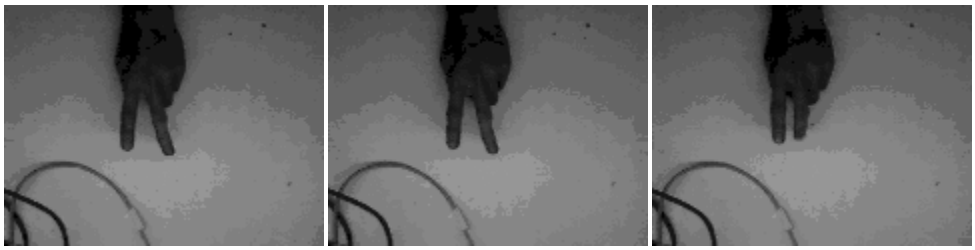
5-2-1. Details on Pre-Processing and Image Extraction

- Judge current frame
  The purpose of this process is to determine whether the current frame needs to be processed.

  The first step is to determine whether the current frame is different from the precious one. The method we applied here the subtract one from the other, and calculate the square sum of all the elements. The threshold we use here is 1650000. In other words, if it is greater than 1650000, the system will consider then as different frames. Otherwise, they will be considered as the same. 1650000 is not critical value. However, if it is too large, it is very hard to trigger the next gesture recognition which will be discussed later. If it is too small, some intermediate pose will be taken as input gesture. From our application, we think 1650000 are quite appropriate. One thing we need to take note is that it is highly depend on the portion of the hand in the picture as well.

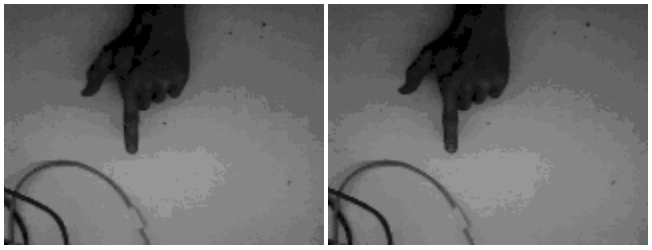
  If the two frames are different, the system will wait until any two adjacent frames are the same.

  The second step is to determine whether the previous frame was judged as the input gesture. If it is, then the current frame does not need to be processed because it makes no sense to recognize the same gesture for more than one time. In reality it means that if the user pose one gesture and hold it for seconds, the system will only recognize the very first frame and ignore the following.
  Base these two criteria, the system will judge whether the current frame needs to be processed. We can see that in order to let the system recognize the next gesture, two adjacent frames need to be different so that the system will think the user is changing the gesture. So if the threshold is too large, no change can be detected.
  The following is the example:



Change:	-	No	Yes
Input gesture:	Yes	No	No

Change:	Yes	Yes	Yes
Input gesture:	No	No	No



Change:	No	No
Input gesture:	Yes	No

So only No.1 and No.7 will be judge as input gesture and further processing will be taken.

- Subtract the background
  Subtract the background from the Get the grey scale image of the hand

- Convert to binary image
  We use Otsu Algorithm to binaries the grayscale image, the detailed algorithm is described as below:
  Assumed that the gray scale is H = {0, 1, …, L}, and the total numbers of pixels, of which gray value is I, are ni , then the total numbers of pixels is $N = \sum_{i=1}^{n} ni$. The probability of gray value of the i is $Pi = ni / N$ and the threshold value is t, then H is divided into two categories: C0 : {0, 1, … , t}, C1: {t+1,t+2, ….. ,t}. The probability of C0 and C1 are

$$P_i(t) = \sum_{t=0}^{t} P_i$$

$$P_i(t) = \sum_{i=t+1}^{L} P_i = 1 - P_0(t)$$

$$u_0(t) = \sum_{i=t+1}^{L} i P_i \Big/ P_0(t)$$

$$u_1(t) = \sum_{i=t+1}^{L} i P_i \Big/ P_1(t)$$

$$u_T(t) = \sum_{i=0}^{L} i P_{Pi}$$

$$ó_B^2 = P_0(t)\ P_1(t)[\ u_1(t) - u_0(t)\ ]2$$

$$t^* = \arg\max_{0<t<l-1} \sigma_B^2(t)$$

Since it is very computational intensive, thus only applies when it is absolutely necessary. In other words, we will convert those “input gesture” images into binary images.

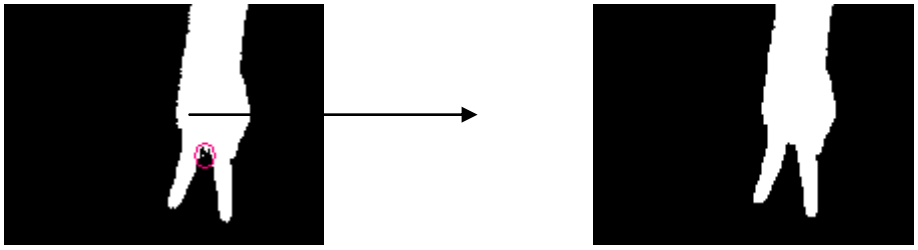
The following is the binary image we get from testing:

- 3*3 erosion

  Erosion is used to get rid of the noise dots. Noise dots are most likely in single form because by using Otsu Algorithm, the threshold for binary image is around 36, so in order to make one noise dots appears, the difference for background needs to be greater than 36. For our case, the background variance is only about 3 on average. So the probability for a noise dot is actually very low. However, the noise dot will affect the rotation heavily. So in order to make sure we can get rid of the noise dots, we choose to apply a 3*3 erosion. For the code, TI image library is used (.asm code)

- 3*3 dilation

  It is use to restore the boundary information, it can be found in TI image library as well.



- Find orientation

  First of all, we have to hind the center of the palm. Get the projection of the binary hand image, and then find peak value of the projection. Use peak value/1.1 as the threshold value to get rid of the wrist and the fingers. For the remaining region, find the center of the region, which should be the center of the palm. This method is quite simple but not robust. In our test, if the hand is vertical or almost vertical, it is very good at telling the palm region of the whole hand image. The results will be +/- 10 pixels compare to the manual result. However, if rotation is more than 45 degrees, it will give us a lot of variance. Since our application is a snake game, thus most likely the user won't put their hand in such a huge angle which is not comfortable at all. Besides, this simple method gives us a satisfactory accuracy and computation cycles. Second, we are going to find the farthest point from the center of the palm. Basically, we just go through the image, find all the 'on' pixels and calculate the distance between them and the center we find. After we find both the center and the farthest point, the line they defined will be considered as the orientation of the hand gesture.

- Linear transformation

  This process is used to move the hand in the center of the frame and cut off the wrist. Since we are using orientation histogram for image matching, whether the hand is at the center does not matter. However, in order to make sure the following rotation won't move any part of the hand out of the picture, this process is necessary. After transformation, the 1-45 rows will be ignored (wrist part).

- Rotation

  The rotation is to make the line connecting the two points mentioned earlier vertical. We are using reversed rotation to make use that there is no "holes" in the hand region after the rotation. The algorithm is to use the rotation matrix $\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$ to determine its original position and fill the new pixel according to the original position. Basically, the rotation won't give us the only result. However, it is always in certain patterns.

  We also find some interesting phenomenon about webcam's input.

  When we place our hand to left side of the camera, we will have edge noises as indicating in the last line of above figure. We try two different webcams and find the same thing happen. We have no explanation for it. However, it is not hard to tackle the problem. All we need to do is ignore the last row of the image when we try to find the farthest point.

- 3*3 erosion

  The very last erosion is used to remove the orientation noise caused by the rotation.

5-2-2. Details on Orientation Histogram Feature of an Image

Generally, orientation histogram is derived from a grayscale image. In our project, considering we are going to implement it in an embedded environment, we choose to make the input binary image, which will save great amounts of floating point calculations and retains much of the boundary information we are interested in. However, when the input is binary image, we can only generate histogram of 4 different angles (45 degree, 90 degree, 135 degree and 180 degree) with respect to horizontal axis, which will yield quantization error in some sense and limit the recognition space to 4 dimensions. Below is a sample orientation histogram vector:

$$v = [57 \;\; 98 \;\; 71 \;\; 31]$$

To generate such a feature vector from a binary image, we count the frequency of each degree. For example, a subset of pixels has a pattern as $\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$ indicating there is a 45 degree edge inside this image. In implementation, we are using a kernel $\begin{bmatrix} 8 & 4 \\ 2 & 1 \end{bmatrix}$ to correlate with the image to decide each pixel's orientation. It can be implemented by using bit-shifting operations only which will make it even faster. Our detailed feature extraction is as follows.

Orientation	Pixel pattern
45 degree	$\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$ $\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$ $\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$ $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$
90 degree	$\begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}$ $\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$
135 degree	$\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$ $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$ $\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$ $\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$
180 degree	$\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$ $\begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}$
Ignored	$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ $\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$ $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

We examine some properties for our feature extraction algorithm. For example,

A one pixel wide horizontal line with length n will generate a feature vector:

$$v = [2 \quad 0 \quad 2 \quad n*2]$$

No matter what the relative position of the line.

And an image of a 45 degree line as below will generate a feature vector:

$$v = [n*2 \quad 0 \quad 2 \quad 0]$$

Where n is the projection length of the line in x or y axis.

Generally, a line with arbitrary orientation larger than 45 degree will yield non-zero number in both 45 degree and 90 degree and the same apply to orientation smaller than 45 degree. For example, the vector representing the image below is

$$v = [60 \quad 122 \quad 2 \quad 0]$$

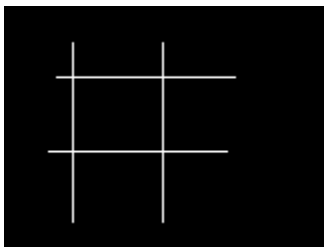
One can argue that the $\tan(\theta)$, where θ is the orientation value, can be approximated by the ratio of 90 degree and 45 degree, but it needs theoretical proof to clarify it. And obviously, if the ratio is not larger than 1, it will not generate legitimate value for $\tan(\theta)$.

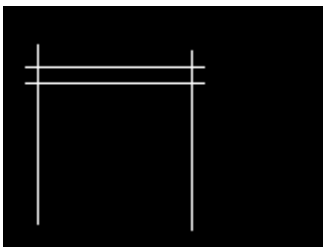
We also discover that this fashion of extracting orientation feature may incur some problems for images of periodic patterns. For example, a binary matrix looks like this:

$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

According to our algorithm, will not generate any 90 degree and 180 degree elements in the feature vector. (‘0’s represent the background and ‘1’s indicate the hand gesture region in our project.) We have two remarks to make on it. First is that we can assume that this kind of pattern doesn’t happen frequently in our hand gesture image, for under common circumstances we don’t have wholes scattering in our hand region. This assumption is bolstered by the fact that when we change our algorithm to get all the degrees included, for example, as for pixel pattern $\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$, it shall generate 45 degree, 90 degree as well as 135 degree, we didn’t see much change. Second, it indicates that 45 and 135 degree are more sensitive to threshold errors (wholes in hand).

One interesting property of feature extraction is that the good property of invariance to in-plane translation will largely limit the alphabet of our project. Considering the next two images, they will generate exactly the same feature vector.





Basically, if we have different objects in the image, any combination of them will not have impact on the feature vector we generate. That is the reason it will be hard to differentiate the two gestures we originally used in our project.

For these two gestures, when we rotate them to the 'standard' position, they will have neighboring values for the feature vector. To interpret it, the index finger will generate the same value as the little in the 90 degree, even though they are at the different position of the palm.

To deal with the different size of the user's hands, we attempt to normalize the feature vector by dividing each element with the sum of all the elements. But this substantially degrades our system performance, one reason for that is the sum of the vector may enjoy much more variance than one element and the normalization will render the system inconsistency. Fortunately, during our test, the scale problem seldom occurs and we suggest user training to prevent this problem occurring.

5-2-3. Details on Hand Gesture Design

Since for our project we need only 5 gestures to control the game, we attempt to choose the gesture sets that make our system robust. The final gesture sets are as follows.

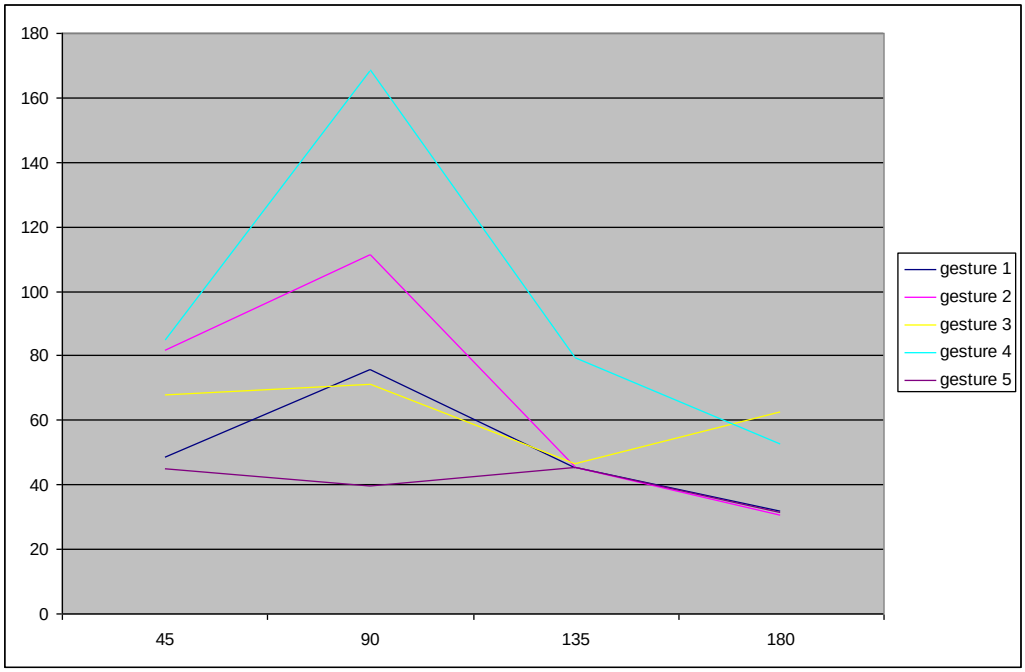
1 2 3 4 5

When we choose the gesture, we want them invariant to the errors in palm-center finding and rotation algorithms. And at the same time, each gesture should generate different feature vectors meaning they will be apart from each other in feature space. We show the mean of our database in the following table.

Gesture No.	45 degree	90 degree	135 degree	180 degree
1	48.8000	75.5750	45.2500	**32.0250**
2	81.7826	111.2609	45.3478	**30.7391**
3	67.9636	71.1636	46.8000	**62.4727**
4	85.0357	168.3929	79.3214	**52.6071**
5	**44.9388**	**39.5102**	**45.3469**	**31.5918**

Basically, gesture 5 should be included for it has the smallest orientation histogram vector values and can serve as a reference value for other gestures.

As we can see, for gesture 1, we have index finger pointed out (actually any one finger pointed out), we will see an increase in 90 degree element. And for gesture 2 we will get increment from gesture 1 for 90 degree as well as more projection on 45 degree. Same analysis can be done for gesture 3. We see significant increment in 45, 90 as well as 135 degree for 3 is a complex gesture and the thumb will generate more projection on 45 and 180 degree. We hope the following figure will show it more clearly.



5-2-4. Details on Matching

Basically, we calculate the Euclidean distance between each gesture and the given vector to assign the gesture vector generating the smallest distance to it.

6. Training and Testing Sets

Our database is generated by one of our teammate (Ke Wang). We calculate the mean of 40 samples of each gesture and use the mean matrix in our matching algorithm.

Each group member makes a testing set. And the test is rigorous in that we generate 50 samples for every gesture. The table below shows the correct rate.

	Ke Wang(trainer)	Jeff	Ran
Gesture 1	90%	96%	**94%**
Gesture 2	100%	100%	**100%**
Gesture 3	98%	100%	**100%**
Gesture 4	100%	100%	**100%**
Gesture 5	100%	100%	100%

As we can see, through our delicate choice of hand gesture alphabet, we get good results.

7. Profiling and Optimization

Otsu algorithm	**614k**
Check frame change	**46k**
Subtract background	**161k**
Convert to binary according to threshold	**41k**
Feature extraction	**518k**
Find palm center	**71k**
Find farthest point	**205k**
Liner-move	**181k**
Rotation	**389k**
Image matching	**440**
Erosion	**7.7k**
Dilation	**7.7k**

- Optimization

  For Otsu algorithm, it has 614k cycles which is the largest among all the processes. So firstly, we change the iteration from 0-255 to 0-50, so the total cycles reduced to 329k. Furthermore, we change the iteration pace to 5, the cycle reduced to 309k. For other functions, there is no loop can be manually unrolled. From the feedback we get, almost all loops have 3 to 4 iterations on parallel except feature extraction. In our feature extraction function, due to our heavily usage of if statement, the program will keep flushing the pipeline and thus runs relatively slow.

The current gray-scaling code takes 40 to 45 milliseconds for conversion. After searching online, I found a better gray-scaling code online, which takes only 15 to 16 milliseconds for conversion.

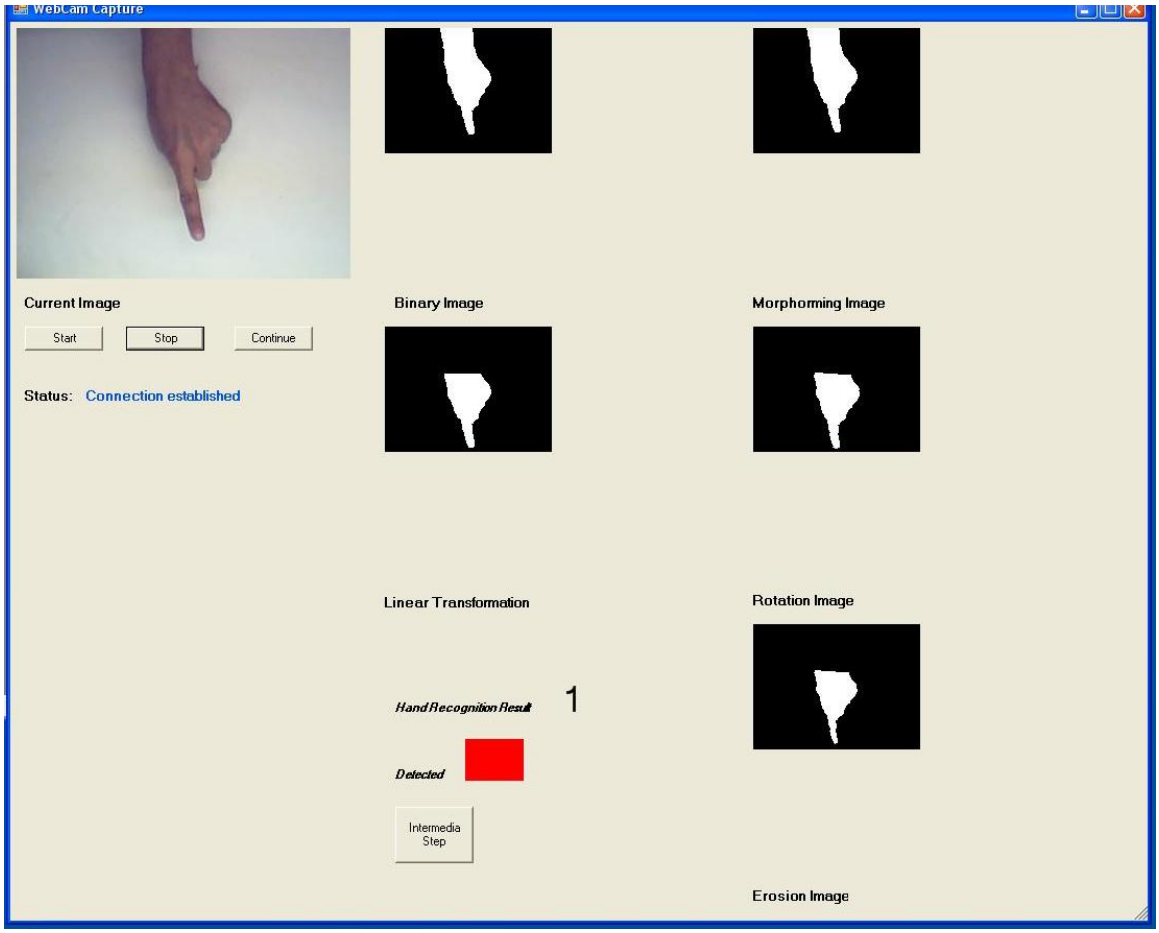
8. Schedule and Task

	Bingran Liu	***Ke Wang***	***Fu-Chieh Chuang***
Oct 13	Matlab code for hand gesture recognition (feature extraction)	Matlab code for hand gesture extraction (key parameter determination)	Create database for validation set
Oct 20	Matlab code for hand gesture recognition (image matching)	Matlab code for hand gesture extraction (linear move, rotation), Training set.	UI Design
Oct 27	Implementation of recognition algorithm (feature extraction)	Implementation of hand gesture extraction (compare frame, binary, orientation determination)	Implement webcams with UI and keyboard control
Nov 3	Implementation of recognition algorithm (image matching)	Implementation of hand gesture extraction (linear move, rotation)	Implement snake game
Nov 10	Subsystem test		Implement the control of the game from UI
Nov 17	Project combine		
Nov 24	Optimization		
Dec 1	Project Testing		

9. Demo

In the demo, our entire group member demonstrated on playing the game. And in our GUI we showed every intermediate steps of our processing on DSK. Both TAs enjoyed our game and Prof. Casasent raised interesting questions according to the image shown.

The figure below is our GUI. As it shows, we show 5 intermediate stages at the same time when a gesture is detected and recognized.



10. Conclusion and Future Work

In this project, we intend to build a robust system allowing user to use hand gesture to control computer games. It is arguable that general user would like to use hand gesture to substitute the traditional input devices such as keyboard and mouse, considering the possibility to have wrong instructions and adaption that must be made to use this kind of unconventional input method. However, as long as gamer is concerned, they will obtain novel experience by using our system, which will make the game more fun in some sense (you have to consider the inconsistency in the

input device). And generally, user of our system show great interest and appreciation to this innovative way of playing games.

Aiming at adding more fun experience, the future work would be adding custom-designed gesture components. It may have features like suggested gesture sets design, provided we have deeper knowledge of generating good gesture sets, and training functions. Also to make our system compatible with the main stream game, we propose to use DirectInput API.

Generally as a hand gesture recognition system, our project has much to improve. One simple alternation is to use local orientation histogram to generate more useful information for a given hand gesture and thus increase our system's maximum vocabulary. However, in order to do this, we need more robust algorithm to find the center of the palm. As stated in 5.2.1, the center will not be correctly found if the hand gesture's rotation is larger than some 45 degree.

11. Reference and Comments

[1] "Static Hand Gesture Recognition based on Local Orientation Histogram Feature Distribution Model" Hanning Zhou, Dennis J. Lin and Thomas S. Huang Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW'04)

This paper proposed a robust way of using local orientation histogram to recognize hand gestures. The algorithm requires huge amount of data access and is difficult to implement it as real-time recognition in embedded environment. Nonetheless, it indicates ways to generate new good features when global orientation histogram feature fails to distinguish gestures.

[2] "Orientation Histograms for Hand Gesture Recognition" William T. Freeman, Michal Roth, IEEE Intl. Wkshp. on Automatic Face and Gesture Recognition, Zurich, June, 1995

Classic Global Orientation Histogram Algorithm for gesture recognition was introduced by William T. Freeman in 1995. We make a slight change in the classic algorithm to fit it to binary input images.

[3] http://phd.serkangenc.com/orientation/orientation.php

The website contains source codes for hand gesture recognition system using global histogram. We are happy to see that such simple algorithm works pretty well in practice.

[4]"Hand Gesture Detection and Segmentation Based on Difference Background Image with Complex Background," Qiuyu Zhang, Fan Chen, and Xinwen Liu.

This reference gives detailed information about how complex background should be handled. So the mean and variance of background as well as the Otsu Algorithm is obtained from this paper.