A Project Report on

Finger Tips Detection and Gesture Recognition

Ankit Gupta
Roll no. Y7061
Kumar Ashis Pati
Roll no. Y7203

Course Instructor: Prof. Simant Dube
CS676

Indian Institute of Technology, Kanpur

November 12, 2009

Acknowledgements

We would like to thank Prof. Simant Dube and Prof. Amitabha Mukerjee for their help and guidance in this project. Without their valuable suggestions, the objectives of this project would never have been fulfilled. We also thank everbody else who helped us to accomplish this task.

Contents

1 Problem Statement

Computer vision based hand tracking can be used to interact with computers in new innovative ways. Our task was to use hand tracking and finger tip detection to create a simple user interface to browse and edit photos.

2 Problem motivation

The availability of low-cost webcams has opened up avenues for a lot of applications in interactive gaming and applications as hand mouse and in the creation of natural user interfaces. Hand gesture recognition is a current field of research and a lot of work is being done on it. Numerous applications have been designed based it. Applications such the photo browsing in an I-phone or other systems using touch-sensitive surfaces can also be implemented using image processing thus giving an alternative to the use of such surfaces as well reducing the cost therein. The various possible applications are:

1. Editing platforms using hand gestures.
2. Implementation of computer mouse using fingers
3. Gesture identification for behavior recognition
4. Hand tracking using low-cost webcams find a lot of applications in interactive gaming and applications as hand mouse and in the creation of natural user interfaces.
5. Gesture recognition is a current field of research and a lot of work is being done on it.
6. Applications such the photo browsing in an i-phone or other systems using touch-sensitive surfaces can also be implemented using image processing thus giving an alternative to the use of such surfaces.

3 Outline of work

The various steps in the entire problem can be divided into the following broad headings:

1. Hand detection in a video
2. Identifying the essential features such as finger tips and orientation
3. Tracking of those features in subsequent frames
4. Adequate description of gestures to be used
5. Accurate identification/recognition of those gestures in a real-time video

For all the above methods we used a low cost webcam in the ambient room light .

4 Condition Assumed

In any problem related to hand tracking or gesture recognition, there usually a few constraints regarding the background, the object, speed of motion etc. The system designed by us works on the following basic assumptions:

1. The camera (webcam) is supposed to be fixed and motionless.
2. The background must have a uniform texture.
3. The background should contrast with the general skin tones.
4. The only features moving in a frame should correspond to the hand i.e. no other object moves in the background.
5. Shadows should be kept to a minimum

5 Hand Detection

Major steps are:-

1. Smoothing for noise removal
2. Color conversion
3. 1st frame is taken as background
4. Background subtraction
5. Thresholding
6. Smoothing for additional noise removal

This deals with identifying the hand in a general video frame and distinguishing it from the background. The pixels in a frame corresponding to the hand need to be segmented and separated so as to use it for further processing.

We use background subtraction for achieving the result. Before background subtraction we use a smoothing function to remove noise. For this we use the *cvSmooth* function with a Gaussian filter. Then color conversion of the input image from RGB to grey-scale is done.

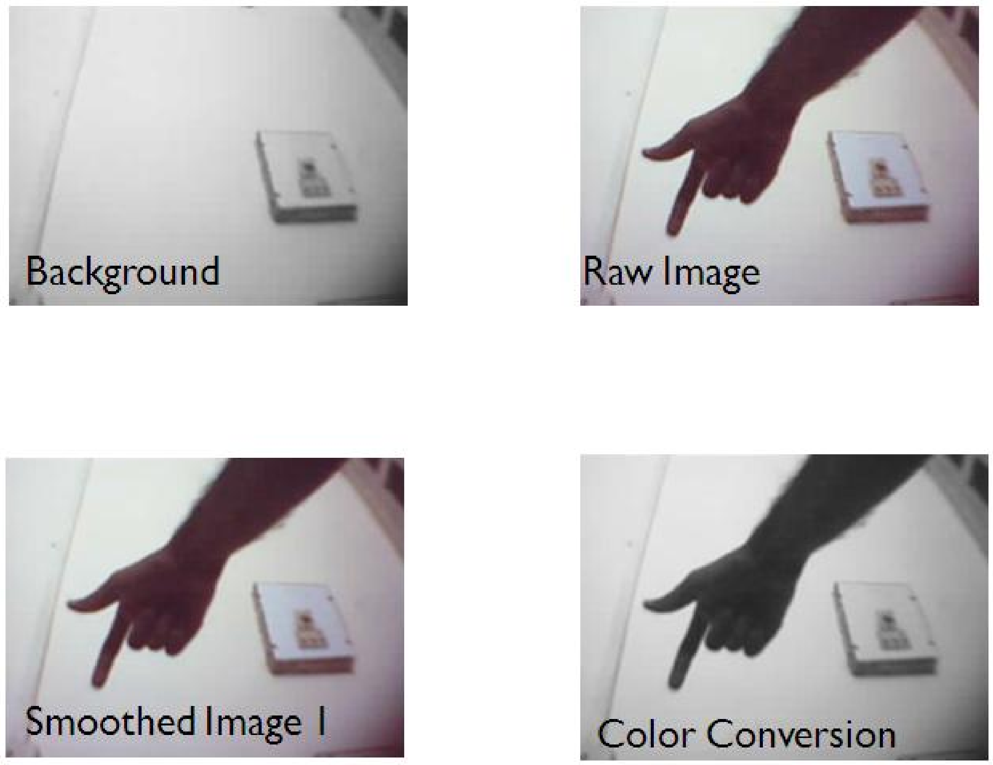


Fig 1: Image capture and noise removal

When the system is started we assume that there are no hand pixels in it and thus the first frame when the system starts is taken as the background and stored in memory. The background can be modeled in time to enhance robustness.

Based on assumption 4 then we can subtract the subsequent frames from the background to detect any motion and identify the hand pixels accordingly. For background subtraction we use the *cvAbsDiff* and then apply simple binary threshold to create a binary image in which the background is black and the hand pixels are white. The new binary image is thus classified as

$$I_{binary} = \begin{cases} 255 & \text{if } |I_{fore} - I_{bac}| > \sigma \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The threshold σ was chosen as 15 after repeated experiments to get the best results. After this the binary image is once again passed through a smoothing function to get remove any additional noise.

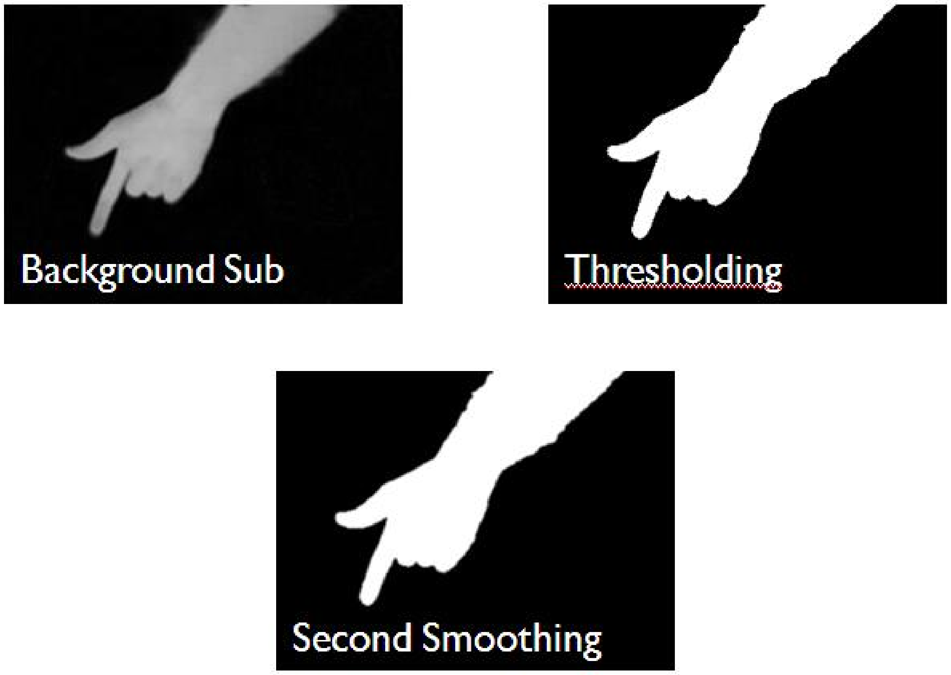


Fig 2: Background subtraction and noise removal

6 Region and Feature Identification

After the segmentation of the hand, we next move to extracting the hand region from the segmented image. We use the *cvFindContour* command to extract all the contours corresponding to probable hand regions. These contours trace the outline of each region. We then compute the contour having the largest area among those. As the largest object moving in the image should correspond to the hand, we conclude that the largest contour must denote the hand.

Once the largest contour is found out we use curvature determination to find out the peaks and valleys in the hand. Let the contour corresponding to the hand be denoted as C and it has n points, and each point is denoted by C_i. We find the curvature by finding the dot product between two vectors [C_i, C_{i-k}] and [C_i,C_{i+k}]. As can be seen from the figure below the points corresponding to the peaks and valleys will have a dot product close to zero for a particular value of k i.e. in the figure below β and ϕ will be close to 90 degrees whereas α will not. Experimentally, we take the threshold dot product value as 20 and the k as 5.

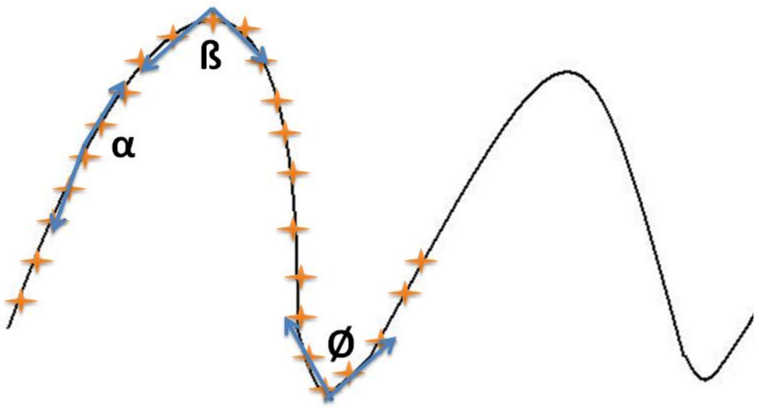


Fig 3: Peak and valley detection

Once the peak and valleys are determined, the next task is to differentiate between the peaks and valleys. This is done by imagining the vectors as 3-D vectors in the X-Y plane and taking their cross products. The direction of the cross product is in the positive Z-axis for peaks and negative Z-axis for valleys. Thus, the finger tips are determined from the hand image. These tips can afterwards be used for making gestures.

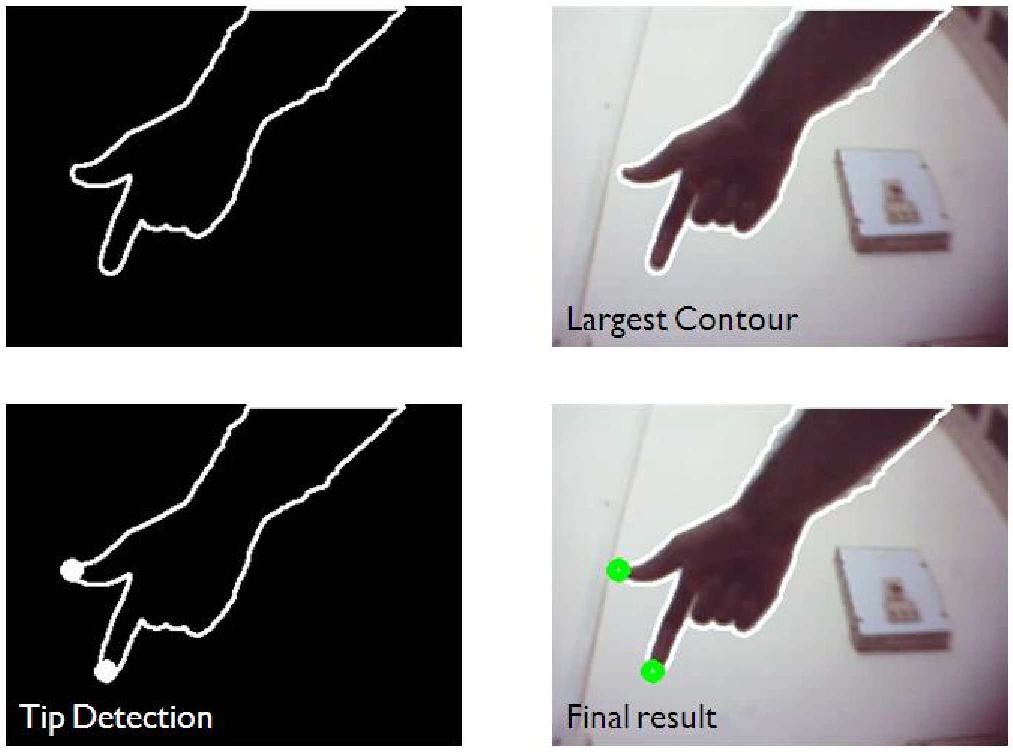


Fig 4: Contour to tip detection steps

7 Use of the identified features

The next task is to use the identified features *i.e.* the tips to create an interface for browsing and editing though photos. For this we tried resizing a given photo using finger tip detection and following it in subsequent frames.

The distance of the identified tip is measured from the top-left corner and x and y coordinates corresponding to the finger tip are used along with the knowledge of the original image size to resize the image. If image size is $X \times Y$ and the size of the webcam frame is $a \times b$ and the coordinates of the tip are (x,y), then the new resized image size is calculated as$X_{new} \times Y_{new}$ where

$$\mathrm{X}_{new} = \frac{X}{a} \times 2 \times x$$
$$\mathrm{Y}_{new} = \frac{Y}{b} \times 2 \times y$$

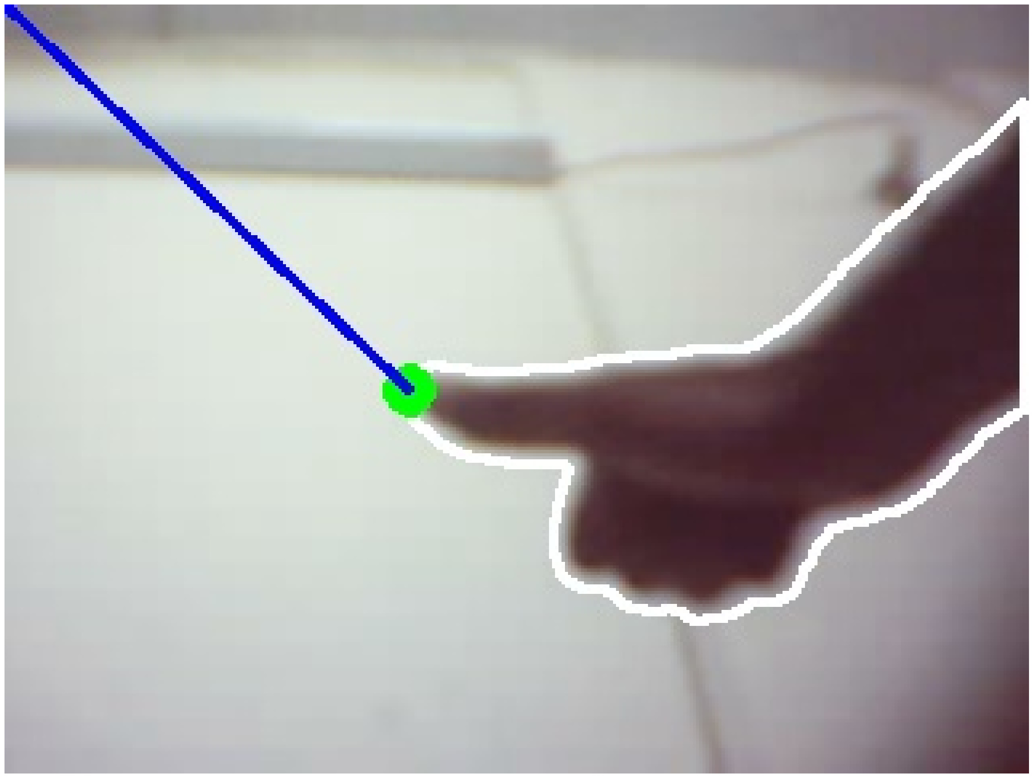


Fig 5: Detection of single peak and location of its coordinates with respect to origin

This data is then used to create a resized version of the given image and display it in real-time.

8 Results

The system was tested using a Zebronics-460WC webcam and an Intel Centrino Duo processor running at 1.67GHz. The entire coding was done on Microsoft Visual Studio utilizing the OpenCV platform created by Intel. The results can be summarized as follows:

8.1 Tip Detection:

The tip detection algorithm performed reasonably well as is seen from the set of images in *Fig 6* but failed at times as is seen from the set of images in *Fig 7(a) and (b).* The peaks were not always detected and if detected were not always tracked in subsequent frames. The primary reason was the change in the curvature of the fingers with the orientation of the hand. Again, because we used curvature to track down the peaks, the distance from the camera affected the curvature and hence hindered peak detection. Also detection of all the 5 fingers was not very accurate. The following images will illustrate it clearly.

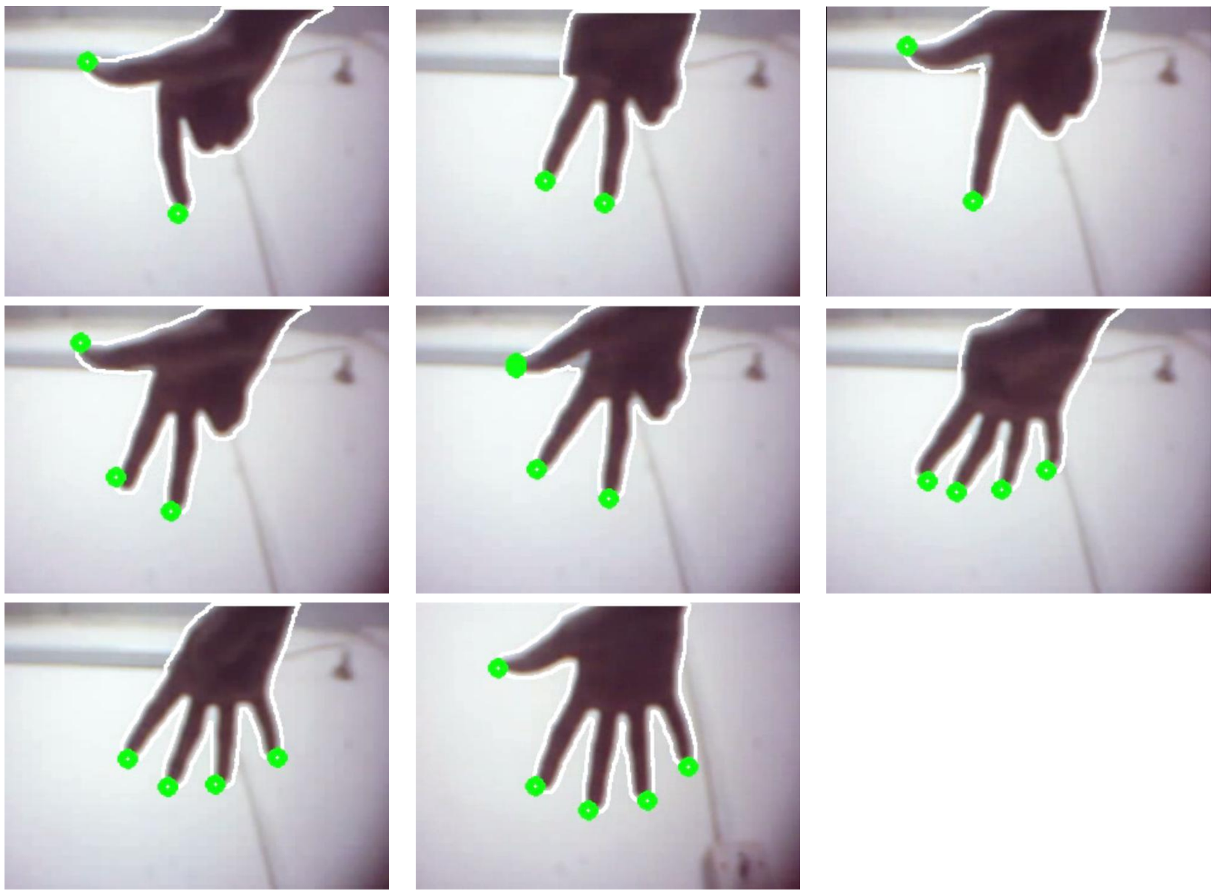


Fig 6: Some of the correct results

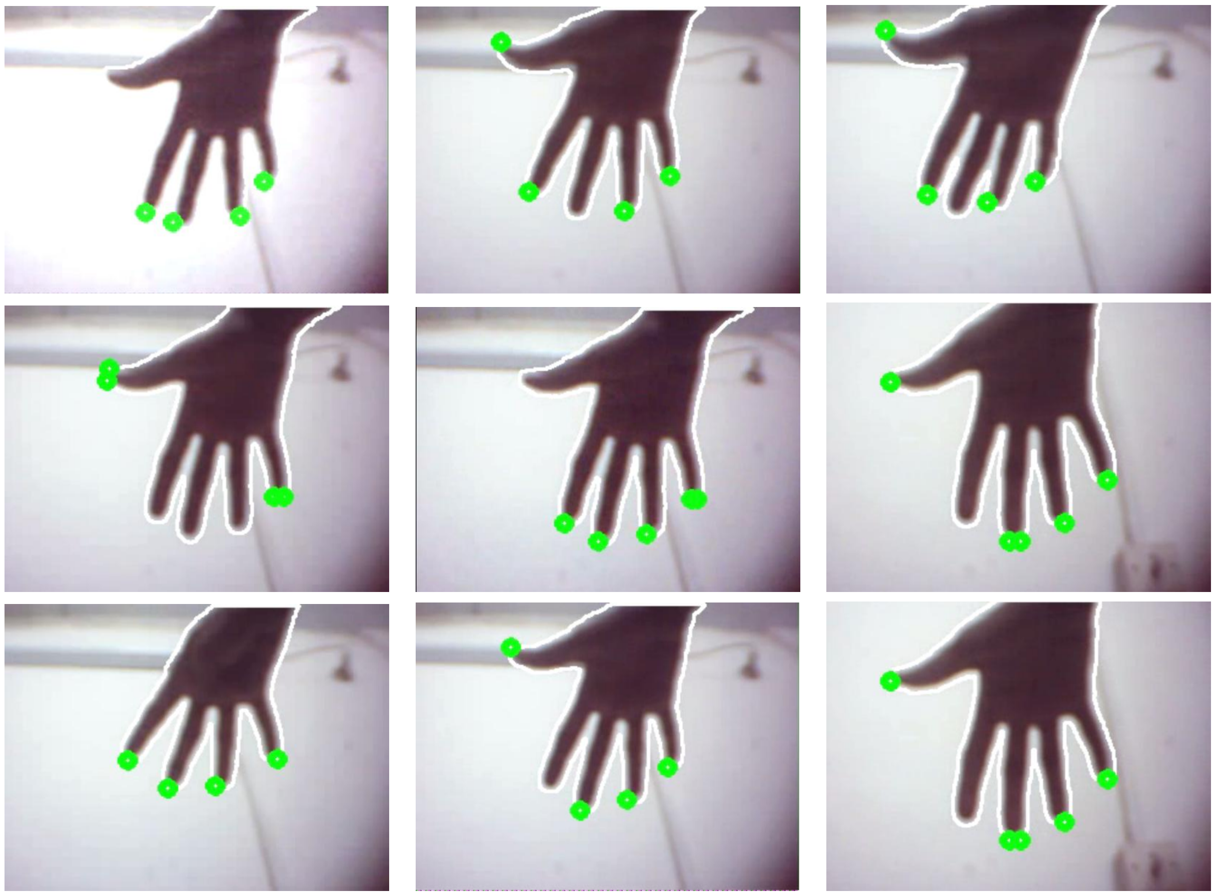


Fig 7(a): Some of the incorrect results (less peaks)

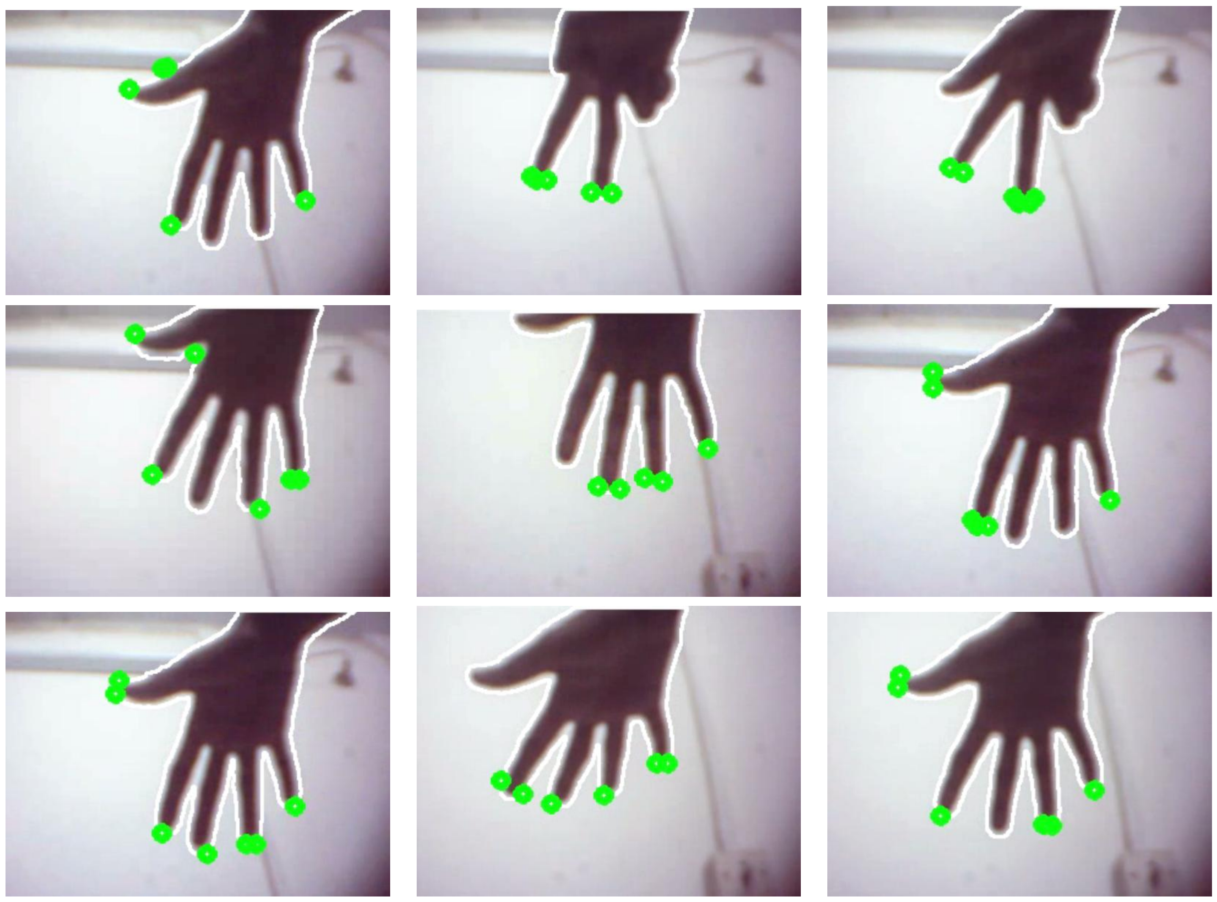


Fig 7(b): Some of the incorrect results (false positive peaks)

False peak detection was not usually a problem unless the hand moved with a greater speed. The tip detection was also a little sensitive to the separation of the fingers i.e. if the fingers were very close. In one of the images below a valley is detected as a peak. The reason for it is that certain portions of the background not very different from the hand pixels. Thus, it is important to ensure that background contrasts with the hand

8.2 Tip Classification:

As the algorithm depends upon curvature calculation from the contour, thus, although the algorithm is immune to rotation, any single peak will always be classified as a thumb. Also, in case the left hand is used the system will treat it like a right hand. In other words the system has no way to determine whether hand is a left or right hand and to distinguish between the fingers.

8.3 Photo resizing:

The algorithm works very well when there is only one tip in the image. In case the user shows two fingers, disturbance may occur. Also any single finger tip is detected as the thumb and the image is resized according to the motion of that finger. The reason for this is explained in the previous heading. The video attached will clearly illustrate this. Again if the finger moves too close to any of the corners then the false tips may result. Tracking of an identified tip is an issue. Another constraint is that the hand shouldnt move very swiftly.

Fig 8: The image in the 'display' window shows the original image which is resized

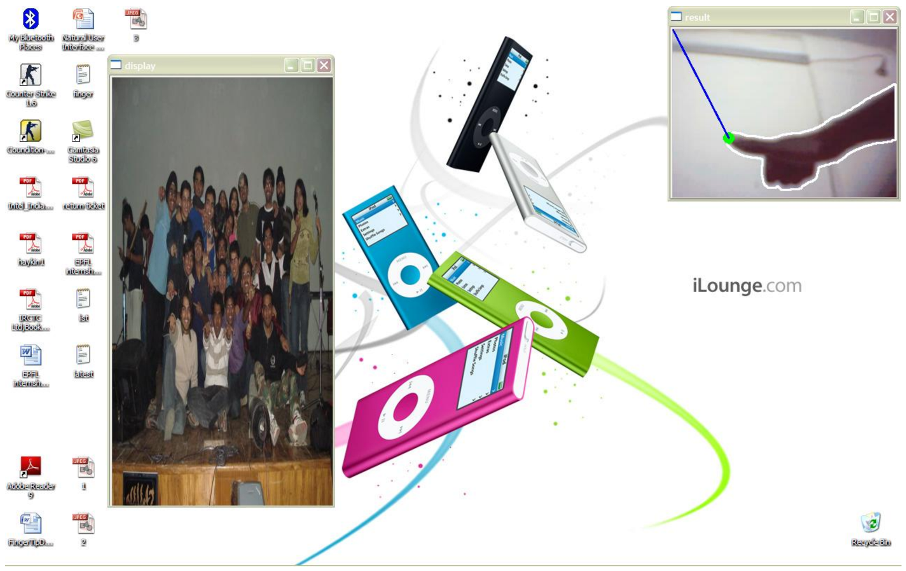
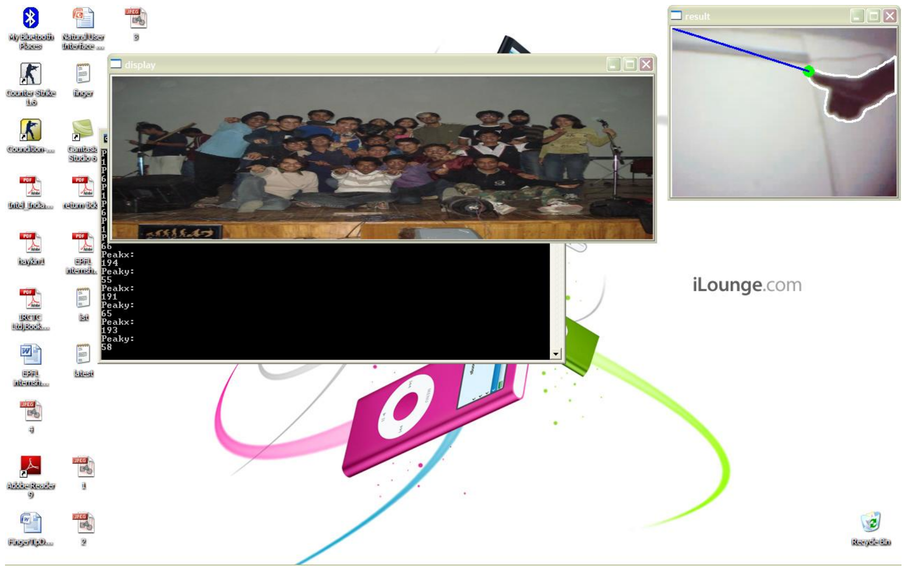


Fig 9: The image in the 'display' window is resized according to the thumb observed in the 'frame' window.



```
66
Peakx:
194
Peaky:
55
Peakx:
191
Peaky:
65
Peakx:
193
Peaky:
58
```

Fig 10: The image in the 'display' window is resized according to the thumb position in the 'frame' window.

9 Code

```
#include "stdafx.h"
#include <stdio.h>
#include "cv.h"
#include "highgui.h"
#include "ml.h"
#include "cxcore.h"
#include "ml.h"
#include "math.h"
#include <stdlib.h>
#include <iostream>
#include <string>
#include <sstream>
using namespace std;


CvHaarClassifierCascade *cascade;
CvMemStorage *storage1;
CvMemStorage *storage2;
CvMemStorage *storage3;
CvMemStorage *storage4;

int main( int argc, char **argv )
{
    CvCapture *capture = 0;  //capturing from camera


    IplImage  *frame = 0;
IplImage *smooth = 0;
IplImage *image = 0;
IplImage *dummy = 0;
IplImage *dummy2 = 0;
IplImage *dummy1 = 0;
IplImage *cloneImage = 0;
IplImage *lastImage = 0;
IplImage *diffImage = 0;
IplImage *bitImage = 0;
IplImage *displayImage = 0;
IplImage *des = 0;
int       key = 0;

storage1 = cvCreateMemStorage( 0 );
storage2 = cvCreateMemStorage( 0 );
storage3 = cvCreateMemStorage( 0 );
```

```
/* initialize camera */
    capture = cvCaptureFromCAM(1);
assert( storage1 && capture );


    if ( !capture ) {
        fprintf( stderr, "Cannot open initialize webcam!\n" );
        return 1;
    }

cvNamedWindow("result", CV_WINDOW_AUTOSIZE );
cvNamedWindow("display",CV_WINDOW_AUTOSIZE);

//image to be resized
displayImage = cvLoadImage("D:/My Memories/ME pics/DSC00685.jpg",1);


//variables for frame count and storing the peaks
int peakx1 = 0;
int peaky1 = 0;
int framec = 0;

while( key != 'q' ) {

/* get a frame */
frame = cvQueryFrame(capture);

if( !frame ) break;
cvSmooth(frame,frame,CV_GAUSSIAN,3,0,0,0);
//If is the first frame
if(!image){
//Create image header same as frame but with 1 channel to gray
image=cvCreateImage(cvGetSize(frame),frame->depth,1);
bitImage=cvCreateImage(cvGetSize(frame),frame->depth,1);
}

//Convert frame to gray and store in image
cvCvtColor(frame, image,CV_BGR2GRAY);

//If is the first frame
if(!lastImage){
//If no lastImage clone actual image;
lastImage=cvCloneImage(image);
}

if(!diffImage){
```

```
//Create image header same as frame but with 1 channel to gray
diffImage=cvCreateImage(cvSize(frame->width,frame->height),frame->depth,1);
}

//background substraction, thresholding and smoothing
cvAbsDiff(image,lastImage,diffImage);
cvThreshold(diffImage,bitImage,15,255,CV_THRESH_BINARY);
lastImage=cvCloneImage(lastImage);
cvSmooth(bitImage,bitImage,CV_GAUSSIAN,3,0,0,0);

//initialising seq variab;es for contours, largest contour and variables for largest area
CvSeq *seq = 0;
CvSeq *larseq = NULL;
CvSeq *domp = 0;
double area;
double largestArea = 0;

//finding the contours
cvFindContours(bitImage,storage2,&seq);

dummy = cvCreateImage(cvSize(frame->width,frame->height),frame->depth,1);
cvZero(dummy);

//for loop to find largest area
while (seq != NULL){
area = fabs(cvContourArea(seq));
   if(area > largestArea){
largestArea = area;
larseq = seq;
}
seq = seq->h_next;
}


cvDrawContours(frame,larseq,cvScalarAll(255),cvScalarAll(255),100,2,8);

//to find out the peak
if(larseq != NULL){
for( int i=5; i<larseq->total-5; ++i ) {
//p,q,r points for defining vectors
CvPoint* p = (CvPoint*)cvGetSeqElem( larseq, i );
CvPoint* q = (CvPoint*)cvGetSeqElem( larseq, i-5);
CvPoint* r = (CvPoint*)cvGetSeqElem( larseq, i+5);
//cur variable for dot product
int cur = (q->x - p->x)*(r->x - p->x) + (q->y - p->y)*(r->y - p->y);
if((cur < 20) && (cur > -20)){
```

```
//check for peak by direction of cross product
if (((q->x - p->x)*(r->y - p->y)-(q->y - p->y)*(r->x - p->x)) > 0){
if((abs(p->x-peakx1)<40)||(abs(p->y-peaky1)<40)){
//storing the value in the point
peakx1 = p->x;
peaky1 = p->y;
break;
}
}
}
}
}

//drawing the point in every 10th frame
if (framec % 10 == 0){
//proportionality constant for resizing
double ratioy = (displayImage->height)*2/frame->height;
double ratiox = (displayImage->width)*2/frame->width;

//defining size for the final resized image
int desy = (int)peaky1*ratioy;
int desx = (int)peakx1*ratiox;

//checking if there is a peak or not
if((peakx1 < 10)||(peaky1 < 10)){
cvShowImage("display",displayImage);
}

//resizing the image
else{
des = cvCreateImage(cvSize(desx,desy),displayImage->depth,3);
cvSet(des,CV_RGB(255,255,255));
cout << "Peakx:\n" << peakx1 << endl;
cout << "Peaky:\n" << peaky1 << endl;
cvResize(displayImage,des,CV_INTER_LINEAR);
cvShowImage("display",des);
}

}

//drawing the peak, the line connecting the peak with the origin and showing them
cvCircle(dummy,cvPoint(peakx1,peaky1),5,cvScalar(255,255,255),5,8);
cvCircle(frame,cvPoint(peakx1,peaky1),5,cvScalar(0,255,0),5,8);
cvLine(frame,cvPoint(peakx1,peaky1),cvPoint(0,0),cvScalar(255,0,0),2,8);
cvShowImage("result",frame);
cvSaveImage("result1.jpg",frame);
```

```
framec++;

//clearing the memory
cvClearMemStorage(storage1);
cvClearMemStorage(storage2);
cvClearMemStorage(storage3);

/* exit if user press 'q' */
key = cvWaitKey(1);
cvReleaseImage(&des);

}

cvDestroyWindow( "result" );
cvDestroyWindow( "display");
cvReleaseCapture( &capture );
return 0;
}
```

10 Future Scope

The following things were planned but couldnt be completed due to paucity of time.

10.1 Robustness of tip detection

The tip detection can be made more robust by incorporating a background model. Also, we thought of including the skin color in segmentation.

10.2 More gestures

We planned on implementing more functions like rotating an image using two hands, zooming on specific areas of an image using two finger tips, browsing through a set of photos using motion of a fingertip.

10.3 Integration

We also thought of integrating all these functionalities into a single software to create a more holistic experience of basic photo browsing and editing. The package would include a set of gestures for different functions and would serve as a natural user interface

The following can be done to improve the algorithm:

1. The curvature K can be made adaptive so as to ensure that the tip detection is not hindered by distance of the hand
2. Finding the orientation of the fingers to create a model of the hand
3. This model can be used as a feedback to ensure that tips are stable and accurate.
4. A set of gestures can be modelled for recognition using matching

11 Conclusions

We were able to detect the motion of the hand in a real-time scenario, track it, identify its features such as finger tips and use those to create a simple application. We were unsuccessful at determining the hand-pose as well differenciating between various fingers. Thus, we were restricted in the number of gestures we could use for further processing. The curvature method is rotation invariant and thus is a good option for determining finger tips but it has shortcomings in the sense that curvature depends on a lot of factors such as hand-orientation and distance. We can also conclude that a lot of applications can be created by using just the finger tips of a hand provided we are able to distinguish between the fingers and are able to track effectively.

References

[1] Shahzad Malik, *Real-time Hand Tracking and Finger Tracking for Interaction*,CSC2503F Project Report,December 18, 2003

[2] Cristina Manresa, Javier Varona, Ramon Mas and Francisco J. Perales , *Real Time Hand Tracking and Gesture Recognition for Human-Computer Interaction*,Electronic Letters on Computer Vision and Image Analysis 0(0):1-7, 2000

[3] Gary Bradski and Adrian Kaehler, *Computer Vision with the Open-CV Library*,Published by O'Reilly Media,September 2008

[4] *http://nashruddin.com/tag/opencv*

[5] Various articles from *www.wikipedia.org*