**TECHNICAL REPORT ON**

**“HAND GESTURE RECOGNITION”**

**SUBMITTED BY:**

JAY PALAN G01036699

KRISHNA SINDHURI VEMURI G01024066

**UNDER THE GUIDANCE OF:**

PROF. Dr. ZORAN DURIC

DEPARTMENT OF COMPUTER SCIENCE



DEPARTMENT OF COMPUTER SCIENCE

VOLGENAU SCHOOL OF ENGINEERING

GEORGE MASON UNIVERSITY

FAIRFAX

VIRGINIA-22030

1. **Abstract**

Gestures are a powerful mode of communication and they play a major role in human computer interaction. Hand gesture recognition is one of the necessary technique used to build eco-friendly interfaces as it helps in human-to-human and human-to-computer communication. The main idea of this project is to create a vision-based system to detect some Indian hand gestures from video sequences. In our project we consider the computer camera as an input device for obtaining the hand gestures. The gestures are detected using the structure analysis and shape detectors. Using the vision concepts and OpenCV methods we plan to detect few of the many gestures.

1. **Introduction and Related Work**

It is a significant challenge for a computer to find gestures. In vision-based gesture recognition approach we require a camera ad taking the challenges like background invariant, lighting invariant and person and camera we need to achieve real-time results. Gesture Recognition is a part of wide variety of applications. Some examples include 3D design, tele presence, virtual reality, and sign language interpretation. Our goal was to design a system that could be used to interpret a few Indian Gestures using the structure analysis and shape detectors of OpenCV with Python.

The gestures we have chosen are Namaste, Lapet, Khoobsurat, and Victory. Namaste is a gesture that is used to greet each other with respect. The gesture Victory is widely used by many politicians in India while campaigning for the elections when the address a large amount of crowd. Lapet is a fun gesture used when you want the other person to know, that you have figured that the person is faking or bragging about himself. The gesture Khoobsurat is used when you see something beautiful or perfect.

a.Beautiful b.Namaste

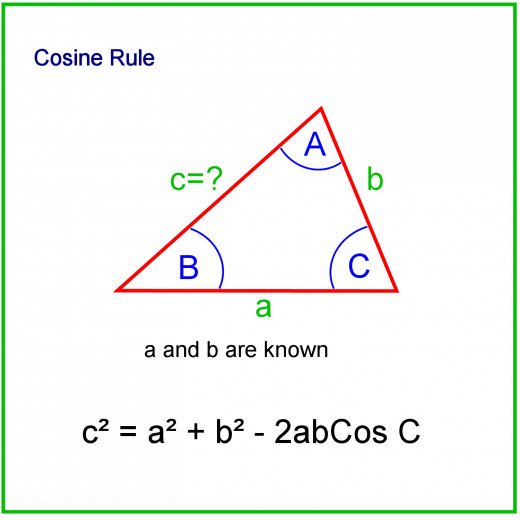
 

c.Lapet d.Victory

1. **Technical Approach**

Our approach was to use the laptop camera as an input. We then consider a 200\*200 window which is essentially used to recognizing the gestures in that window. Once the hand is placed in the window, we perform certain operations to recognize the gestures. They are given as follows.

* Convert the RGB window frame to gray.
* Blur the gray image frame using Gaussian Blur technique so that all the unwanted noise is removed.
* Depending on the background you need to choose a thresholding technique such that the hand is well segmented from the background. If the back ground is dark we choose the thresholding cv2.THRESH\_BINARY+cv2.THRESH\_OTSU, if the background is light we choose the thresholding as cv2.THRESH\_BINARY\_INV+cv2.THRESH\_OTSU. This is mainly used to segregate the hand from a uniform background.
* Once the thresholding is done we need to find the contours of the gesture.
* The area of the maximum contour is calculated
* The convex hull of the image is calculated along with its area.
* We need to then find convexity defects of these gestures
* Using the Cosine rule, we calculate the number of triangles formed by the gesture. Since the gestures chosen deal with fingers that are straight, this technique can be used to get the angles between the fingers. The convexity defects and the angle between them can determine the gestures.



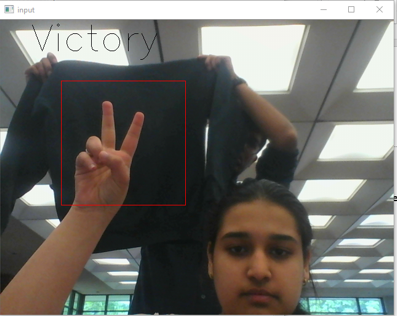
* We determine the normalized area as normalized\_area as hull\_area/max\_area

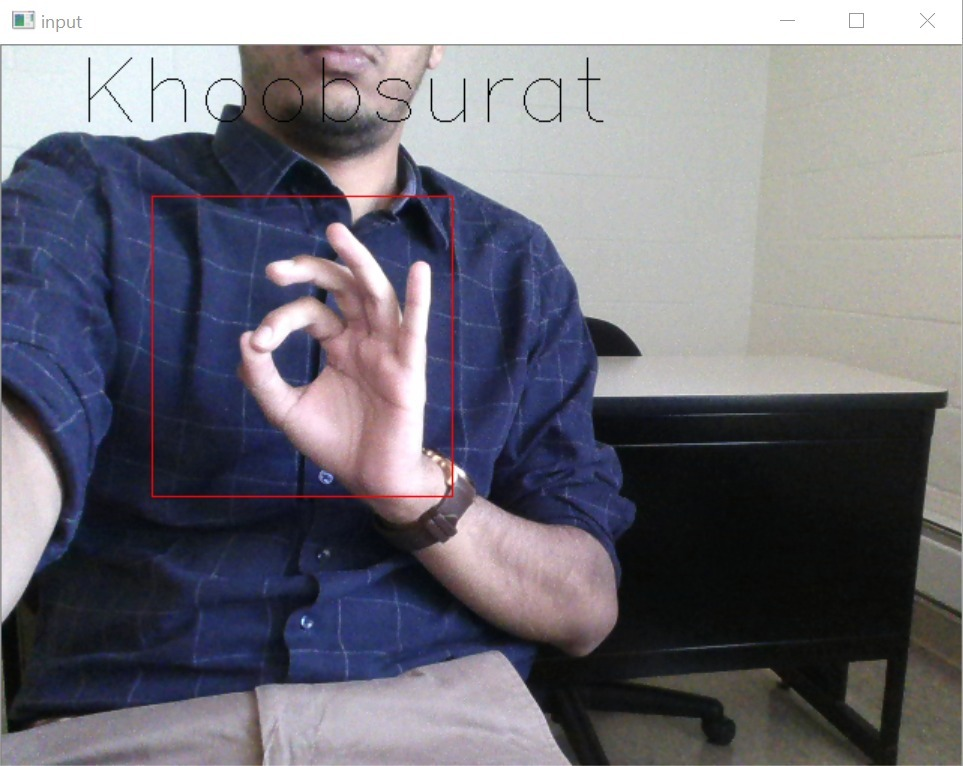
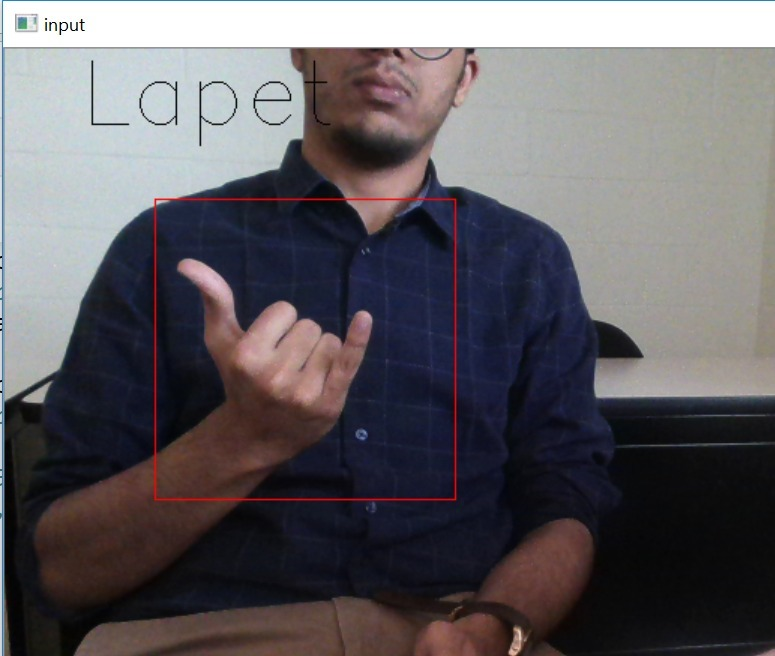
The gestures are categorized into the following categories.

* normalized\_area>1.7 and normalized\_area<1.8 and count\_defects ==0🡪Namaste.
* normalized\_area>1.47 and normalized\_area<1.55 and count\_defects==0🡪 Lapet
* normalized\_area> 1.25 and normalized\_area <1.35 and count\_defects==3 🡪Khoobsurat
* normalized\_area>1.45 and normalized\_area<1.55 and count\_defects==1🡪 Victory

1. **Results and Discussions**

Our program was able to determine the gestures on both of our hands. The following are few results.





Coming to the discussions we were trying to incorporate data mining concepts such as K-nearest neighbors and tried some template matching. Since we have very little knowledge in the field of the Data Mining it was very challenging for us to implement it in the given time constraints.

The python code of the program is given here,

*import cv2*

*import numpy as np*

*import math*

*# open the camera*

*cap = cv2.VideoCapture(0)*

*# used this variable to save the outputs*

*count = 0*

*#while the camera is open perform the following operations*

*while( cap.isOpened() ) :*

*#read input from camera*

*ret,img = cap.read()*

*cv2.imwrite('ProjectOutput/image%3d.jpg' % count,img)*

*#define a frame for the gesture recognition*

*cv2.rectangle(img, (301,301), (99,99), (0,0,255),0)*

*#create a new window of the frame*

*mask\_window = img[100:300, 100:300]*

*#cv2.imwrite('ProjectOutput/''mask\_window%3d.jpg' % count,mask\_window)*

*# convert the above window to a gray scale image to perform thresholding*

*grey = cv2.cvtColor(mask\_window, cv2.COLOR\_BGR2GRAY)*

*blur = cv2.GaussianBlur(grey,(5,5),0)*

*# apply thresholding on the image so that foreground is extracted from the background*

*ret, thresh = cv2.threshold(blur,200,255,cv2.THRESH\_BINARY\_INV+cv2.THRESH\_OTSU)*

*#used to store the contour area of the largest contour*

*max\_area=0*

*max\_cnt\_number =0*

*#find all the contours for the thresholded image*

*image,contours, hierarchy = cv2.findContours(thresh,cv2.RETR\_TREE,cv2.CHAIN\_APPROX\_NONE)*

*# if contours exist then perform the following operations*

*if contours is not None:*

*# check all the contours and find the contour with largest area using the cv2.contourArea() function*

*for cnt in range(len(contours)):*

*current\_cnt=contours[cnt]*

*current\_area = cv2.contourArea(current\_cnt)*

*if(current\_area>max\_area):*

*max\_area=current\_area*

*max\_cnt\_number = cnt*

*max\_cnt = contours[max\_cnt\_number]*

*#find the convexity hull of the image*

*hull = cv2.convexHull(max\_cnt)*

*#find the hull's area*

*hull\_area = cv2.contourArea(hull)*

*#we are using the defects area for recognizing the gesture as we have gestures with similar hull area values*

*defects\_area = hull\_area - max\_area*

*#display the contours and conexity hull on the blank image contourdisplay*

*contoursdisplay = np.zeros(mask\_window.shape,np.uint8)*

*cv2.drawContours(contoursdisplay,[max\_cnt],0,(0,255,0),2)*

*cv2.drawContours(contoursdisplay,[hull],0,(0,0,255),2)*

*#find the convexity defects of the image window*

*hull = cv2.convexHull(max\_cnt, returnPoints=False)*

*defects = cv2.convexityDefects(max\_cnt, hull)*

*count\_defects = 0*

*cv2.drawContours(contoursdisplay, contours, -1, (255, 0, 0), 3)*

*#if defects exist*

*if defects is not None:*

*#find the start,end and farthest points for the defects*

*for i in range(len(defects)):*

*s,e,f,d = defects[i,0]*

*start = tuple(max\_cnt[s][0])*

*end = tuple(max\_cnt[e][0])*

*far = tuple(max\_cnt[f][0])*

*# use the cosine rule for finding the angles between*

*a = math.sqrt((end[0] - start[0])\*\*2 + (end[1] - start[1])\*\*2)*

*b = math.sqrt((far[0] - start[0])\*\*2 + (far[1] - start[1])\*\*2)*

*c = math.sqrt((end[0] - far[0])\*\*2 + (end[1] - far[1])\*\*2)*

*angle = math.acos((b\*\*2 + c\*\*2 - a\*\*2)/(2\*b\*c)) \* 57*

*#check if the angle is less than 90 and then increments the number of triangles in that image*

*if angle <= 90:*

*count\_defects += 1*

*#display the triangular points on the contourdisplay image*

*cv2.circle(contoursdisplay, far, 1, [255,0,0], -1)*

*cv2.line(contoursdisplay,start, end, [0,255,0], 2)*

*#cv2.imwrite('ProjectOutput/contour%3d.jpg' % count,contoursdisplay)*

*shape\_str = ''*

*normalized\_area=hull\_area/max\_area*

*# categorize the gesture*

*if(normalized\_area>1.7 and normalized\_area<1.8 and count\_defects ==0):*

*shape\_str = 'Namaste!'*

*cv2.putText(img,shape\_str, (50, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 2, 2)*

*#cv2.imwrite('ProjectOutput/'+str(shape\_str)+'%3d.jpg' % count,thresh)*

*elif(normalized\_area>1.47 and normalized\_area<1.55 and count\_defects==2):*

*shape\_str = 'Lapet'*

*cv2.putText(img,shape\_str, (50, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 2, 2)*

*#cv2.imwrite('ProjectOutput/'+str(shape\_str)+'%3d.jpg' % count,thresh)*

*elif(normalized\_area>1.25 and normalized\_area<1.35 and count\_defects==3):*

*shape\_str = 'Khoobsurat'*

*cv2.putText(img,shape\_str, (50, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 2, 2)*

*#cv2.imwrite('ProjectOutput/'+str(shape\_str)+'%3d.jpg' % count,thresh)*

*elif(normalized\_area>1.45 and normalized\_area<1.55 and count\_defects==1):*

*shape\_str = 'Victory'*

*cv2.putText(img,shape\_str, (50, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 2, 2)*

*#cv2.imwrite('ProjectOutput/'+str(shape\_str)+'%3d.jpg' % count,thresh)*

*# else:*

*# cv2.putText(img,str(normalized\_area), (50, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 2, 2)*

*# cv2.putText(img,str(count\_defects), (50, 100), cv2.FONT\_HERSHEY\_SIMPLEX, 2, 2)*

*#Display all the windows*

*cv2.imshow('Thresholded',thresh)*

*cv2.imshow('input',img)*

*cv2.imshow('Contour',contoursdisplay)*

*k = cv2.waitKey(10)*

*if k == 27:*

*break*

*count = count + 1*

*cap.release()*

*cv2.destroyAllWindows()*

1. **Conclusions**

Our approach works only for few gestures and has many limitations. The first limitation is the detection of the hand, we could use any object in the window and get faulty results. For example, a pen in the window can show ‘Namaste’ as it has no convexity defects. There could be many false positives and false negatives. The second limitation would be background subtraction. If we could detect a hand properly using good foreground extraction or background subtraction techniques, then we can eliminate a few false positive and false negative cases. The distance between the hand and camera determines the defects area and we need to normalize the results. Also the gestures Victory and Lapet have same count defects and it arises in false positives.

1. **References**
2. OpenCV –Structure Analysis and Shape Detectors
3. Robust Hand Gesture Recognition with Kinect Sensor Zhou Ren Jingjing Meng Junsong Yuan Zhengyou Zhang
4. Vision based hand gesture recognition for human computer interaction: a survey Siddharth S. Rautaray · Anupam Agrawal
5. https://github.com/mahaveerverma/hand-gesture-recognition-opencv/