**Image Processing**

**Aim:** To detect motion and hence recognize the number of faces in order to compare them.

**Theory:**

We have used opencv2 for image processing.

**Why OpenCV Over Matlab:**

* **Speed:**

Matlab code is interpreted first into java and then it is converted into C/C++ code. Hence time required for it to get converted into machine level language is significant .In comparison, OpenCV code is directly based on C/C++ and thus time required to get converted into machine level language is relatively less.

* **Resources:**

Matlab consumes more RAM than Opencv.

* **Cost**:

Opencv library is readily available and free of cost but Matlab is not.

**Procedure:**

**Step 1:**

The first step is to continuously capture frames.This is done with the help of a webcam which is already installed on every machine. Here we keep track of three frames:

Previous Frame

Current Frame

Next Frame

All three frames are converted into grayscale and then the absolute difference of previous and current along with of current and next is taken. The two differences are logically anded. This basically eliminates all the similar pixels and thus detects the presence of any new pixels(movement ) in the current frame. The area of movement will have white pixels and thus their intensity can be measured. Thus by comparing the intensity of each pixel with a certain threshold value, motion can be detected.

**Step 2:**

The second step involves detecting faces in the frame just captured in which motion was detected.

This involves using a machine learning algorithm known as haar cascades for face detection. For our project , in order to achieve accuracy, two haar cascades (frontal face alt and frontal face alt2) are used. These two are used for detecting faces in the captured frame. As soon as a face is detected, the image is cropped to the size of the face captured and is written in memory.

**Step 3:**

Third Step involves comparing two different faces to see if they match or not. For this , we have used template matching. Steps 1 and 2 are executed twice to capture and save two different images of the same person. These two images are then compared using the template\_match function.This function compares every pixel and determines the value of the pixel that matches the highest. This method also returns the x and y coordinates of that pixel. Thus a rectangle or region of Interest(ROI) is drawn. By comparing the areas of the original image and the rectangle and also comparing the value of the returned pixel with a threshold value, it becomes possible to check if the two faces are belonging to the same person.

Code for Motion detection and face detection:

import cv2

import numpy as np

face\_cascade1 = cv2.CascadeClassifier('C://Users/Admin/Downloads/opencv/build/etc/haarcascades/haarcascade\_frontalface\_alt.xml')

face\_cascade2 = cv2.CascadeClassifier('C://Users/Admin/Downloads/opencv/build/etc/haarcascades/haarcascade\_frontalface\_alt2.xml')

capture =True

cap =cv2.VideoCapture(0)

ret,pf = cap.read()

ret1,f = cap.read()

ret2,nf = cap.read()

prevFrame = cv2.cvtColor(pf,cv2.COLOR\_BGR2GRAY)

frame = cv2.cvtColor(f,cv2.COLOR\_BGR2GRAY)

nextFrame = cv2.cvtColor(nf,cv2.COLOR\_BGR2GRAY)

faces = []

def diffImg(t0, t1, t2):

d1 = cv2.absdiff(t2, t1)

d2 = cv2.absdiff(t1, t0)

return cv2.bitwise\_and(d1, d2)

detected = 0

while capture:

anded = diffImg(prevFrame,frame,nextFrame)

height, width= anded.shape

for i in range(0, height):

for j in range(0,width):

v = anded[i,j]

if v>100:

print "Motion Detected"

detected= 1

break

if detected == 1:

break

if detected==1:

faces1 = face\_cascade1.detectMultiScale(f, 1.1, 7)

faces2 = face\_cascade2.detectMultiScale(f, 1.1, 7)

print "detected"

crop\_img=0

for (x,y,w,h) in faces1:

print "In face"

cv2.imshow('img',f)

crop\_img = f[y:y+h, x:x+w]

cv2.imwrite('7.png',crop\_img)

for (x,y,w,h) in faces2:

print "In face"

cv2.imshow('img',f)

crop\_img = f[y:y+h, x:x+w]

cv2.imwrite('8.png',crop\_img)

break

print "Not detected"

cv2.imshow("Frame",anded)

prevFrame = frame

frame = nextFrame

f=nf

ret2,nf = cap.read()

nextFrame = cv2.cvtColor(nf,cv2.COLOR\_BGR2GRAY)

key = cv2.waitKey(10)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

cap.release()

cv2.destroyAllWindows()

//Code for template matching

import cv2

import numpy as np

import math

import time

capture = True

img = cv2.imread("parth.png")

img1 = cv2.imread("4.png")

W,H=img.shape[:2]

w,h = img1.shape[:2]

res = cv2.matchTemplate(img,img1,cv2.TM\_CCOEFF\_NORMED)

min\_val, max\_val, min\_loc, max\_loc = cv2.minMaxLoc(res)

top\_left = max\_loc

bottom\_right = (top\_left[0] + w, top\_left[1] + h)

a1 = W\*H

a2= (bottom\_right[1]-top\_left[1])\*(bottom\_right[0]-top\_left[0])

print a1

print a2

if(((a1-a2)/a1)<0.1 and max\_val>0.5):

print 'match'

cv2.rectangle(img,top\_left, bottom\_right, 255, 2)

cv2.imshow('Image',img)

cv2.waitKey(0)

cv2.destroyAllWindows()

print max\_val

print a1-a2

print max\_loc

**Conclusion:**

The code works fine for standard test cases.However refinement might be required when the model is tested in different working environments.