CS 512 Assignment 3: Report

Charu Saxena
Department of Computer Science
Illinois Institute of Technology
22nd oct ,2017

Abstract

In this report we will discuss about various implementation and usage of various opencv functions and python functions using python 3.6. Some of the implementations and usage discussed are basic computer vision and image processing problems including capturing images and live images, modifications on image. Particularly, the descriptions and implementation of corner detection is discussed in this report which will include implementation of Harris corner detection, localization for a image, and also for two images(one be with the different view point) feature detection and matching.

1. Problem Statement

a) To write a program to load and display images and perform following operations:
- To estimate gradients and apply own implementation of Harris corner detection.
- To get better localization for each corner.
- To compute feature vector for each corner point
- To display rectangles over the original image centered around detected corners.

b) Using the feature vectors match the feature points in both the images and number the corresponding points.

c) Introduce trackbar/button to control variance of gaussian,neighbourhood size of correlation matrix, weight of trace in harris corner detection and a threshold value.

2. Proposed Solutions

a) <u>Implementation of harris corner detection:</u>

As per theory, maximising the function E(u,v) will give corner where,

$$E(u,v) = \sum_{x,y} \underbrace{w(x,y)}_{\text{window function}} \underbrace{[I(x+u,y+v)}_{\text{shifted intensity}} - \underbrace{I(x,y)}_{\text{intensity}}]^2$$

which can be further solved as:

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

$$M = \begin{bmatrix} \sum_{S(p)} (dI/dx)^2 & \sum_{S(p)} dI/dx dI/dy \\ \sum_{S(p)} dI/dx dI/dy & \sum_{S(p)} (dI/dy)^2 \end{bmatrix}$$

Here I found I_x I_y are image derivatives using **cv2.sobel( ).**

After we get this we consider each window size to detect if there is a corner or not using equation:

$$R = det(M) - k(trace(M))^2$$

where

- $det(M) = \lambda_1 \lambda_2$
- $trace(M) = \lambda_1 + \lambda_2$
- λ_1, λ_2 are the non-sorted eigenvalues of M
- x_1, y_1 are the eigenvectors corresponding to λ_1
- x_2, y_2 are the eigenvectors corresponding to λ_2

Here I used **cv2.cornerEigenValsAndVecs(src, blockSize, ksize[, dst[, borderType]])** which will give four results $(\lambda_1, \lambda_2, x_1, y_1, x_2, y_2)$.

So we will find det(M) and trace(M) from this opencv function.

After this I am able to detect corner for a widow but now we have to choose the exact location of the corner in the window(localization) for which i have used opencv function:**cv2.cornerSubPix(image, corners, winSize, zeroZone, criteria)**.

After detecting the exact location for the corner I have to put rectangles around it which are centered at rectangle and numbers:

cv2.rectangle(img,(),(),()) and cv2.putText()

b) <u>Matching the feature points:</u>

After step a) is done for both the images, I will now try to match the corresponding feature points and numbers plotted. For this I have used the unique corners that we detected earlier and pass through:**cv2.keypoints()** to get keypoints , which is then passed to:

```
orb = cv2.ORB_create()
kp2, des1 = orb.Compute(img1,kp1)
```

Where the keypoints will be detected in kp1 and the description of those points will be saved as des1.
For matching the points I have used:
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
matches = bf.match(des1,des2)#to match description from both images des2 is for second image.
After this we will sort these matches to detect the best matches first using **sorted()** then I will use: **img3 = cv2.drawMatches(img1,kp3,img2,kp4,matches[:number of matches], flags=2)**
This will draw lines across the both the images to match the feature vectors.

c) **Introduce trackbar:**

After a) and b) I will use trackbar to manually change and see the results when certain parameters are changed.Parameters that are in trackbar are:

- Variance of gaussian()
- Window size
- Threshold for detecting the corner
- K value to detect R

I have used two function for this: **cv2.CreateTrackbar(trackbarName, windowName, value, count, onChange)** and **cv2.getTrackbarPos(trackbarname, winname).**

3. Implementation

The implementation is done in python 3.6 and spyder

Two images are captured by webcam in the interval of **3 seconds**

Problems faced and solutions:

- The very first was defining the termination criteria for cornerSubPix() which I used **criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 250, 0.1)** where I used epsilon value as 0.1 and max-iterations = 250.
- Also, deciding the initial values of k and threshold is an issue,
- One other is to draw color rectangles on gray-scaled image, so in the end I made again converted the rectangles to color and them drew them on gray-scaled image.

Instructions for the program:

Keyboard: Press ‘q’ to destroy all frames
Press esc to close
Trackbar:”threshold” to adjust threshold to determine corners.
“k-value” to adjust k-value for R
“w” to adjust window-size
“g” to adjust gaussian for derivative

4. Result and discussion

I. This contains comparisons between cv2.harriscorner vs self-implemented harris corner:

a)number of corners detected:

Observation:

Above discussed is the number of corners detected by cv2.harriscorner on the left and on the right I have self-implemented one. Red dot on the left image shows corners detected whereas in the right image corners are detected by rectangles with numbers on them. So we see that the corners are almost similarly detected. This shows a good implementation of the cv2 functionality.

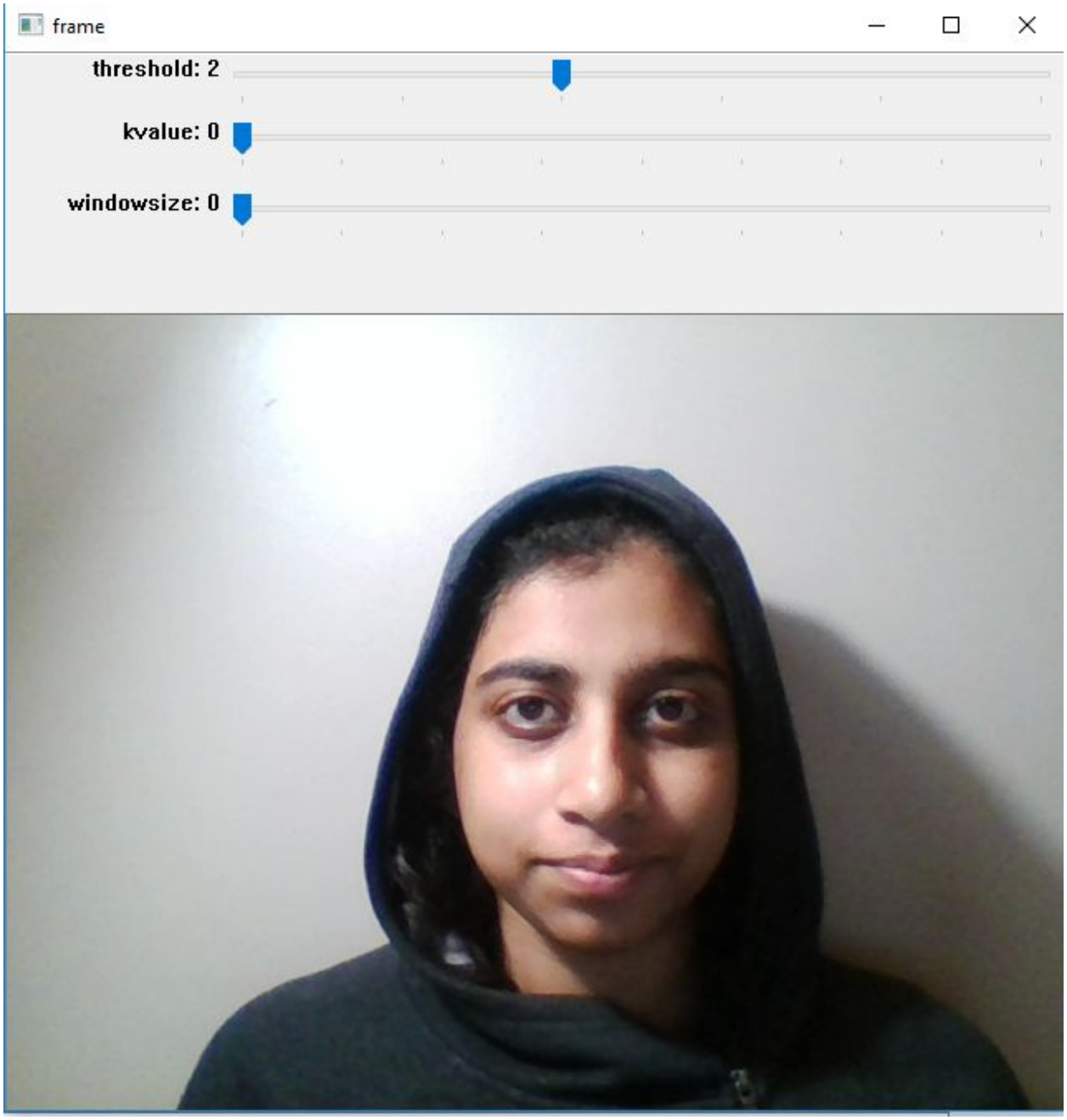
b) number of corners after blue affect:

Observation:

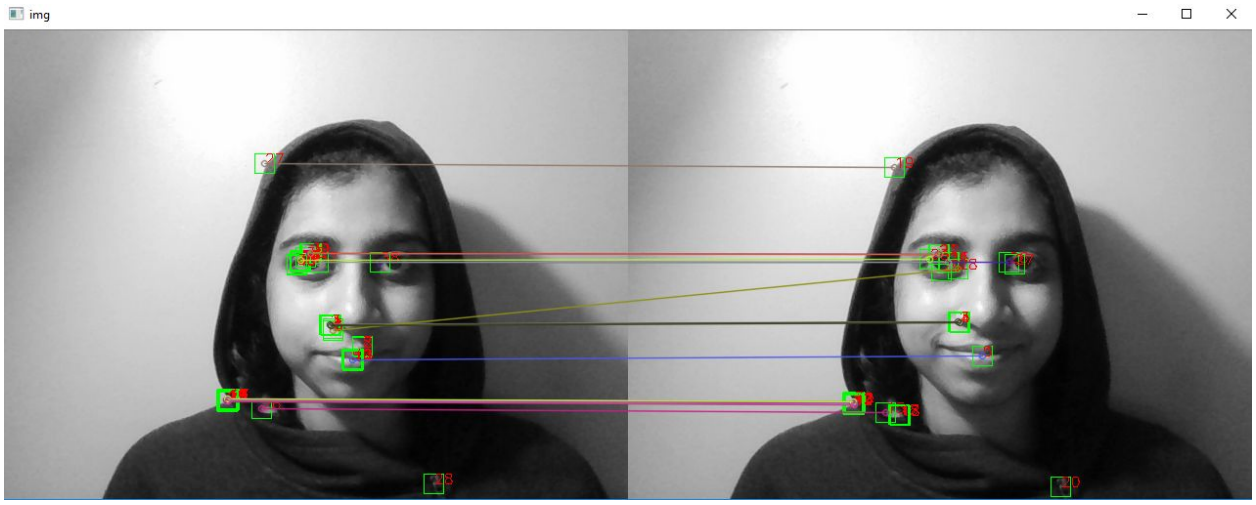
Above discussed is the number of corners detected by cv2.harriscorner on the left and on the right I have self-implemented one after the BLUR affect on both the implementation. Red dot on the left image shows corners detected whereas in the right image corners are detected by rectangles with numbers on them. So we see that the corners are almost similarly detected. This shows a good implementation of the cv2 functionality.

II. This contains result of parameter tuning in self implemented harris corner and matching

Below is the screenshot of my frame which will be having the track bars showing: threshold, k-value and window_size.



a)**comparing the thresholds value: here we have considering the effect of threshold on corners detected:**



Observations:

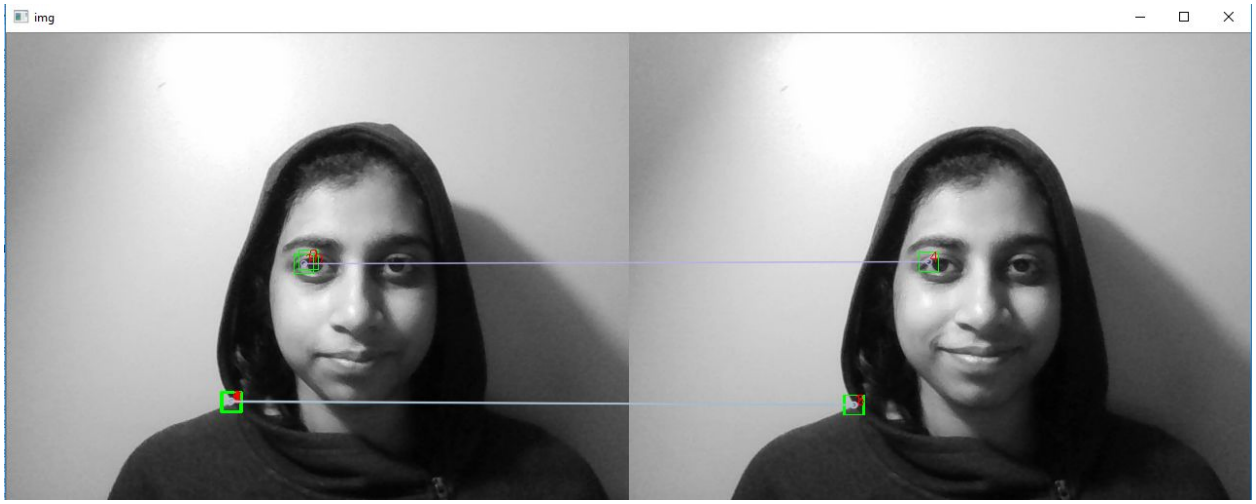
Here we see that when the threshold value is increased from 2 to 4 the number of corners detected and the matches decreased. Parameters were same like windowsize=3, k-value=0.2. This is because, when the threshold is increased we have lesser number of corners which will have product of eigenvalues greater than this increasing threshold resulting in lesser number of corners.

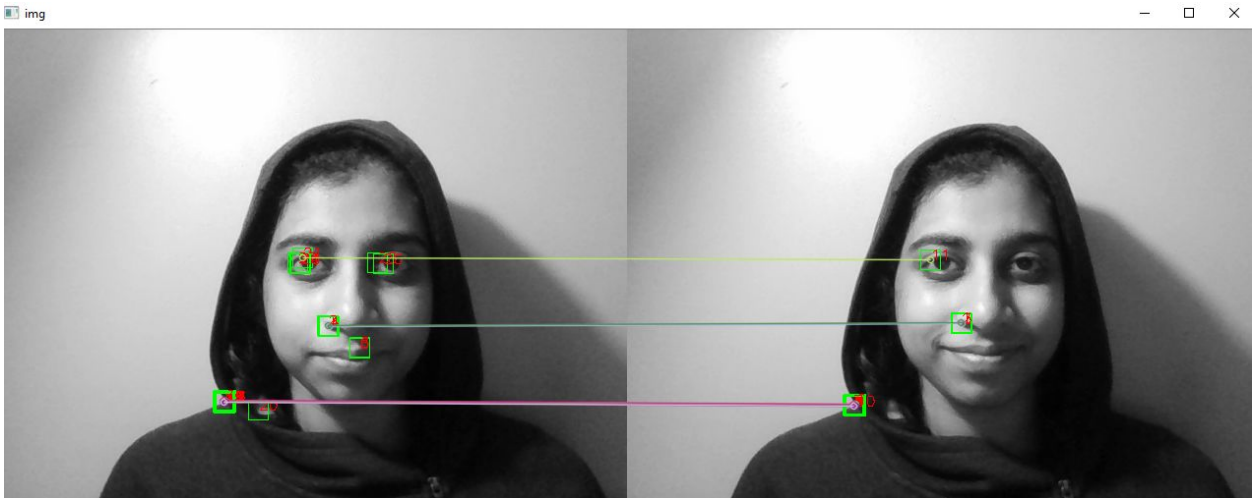
b)**comparing the k value: here we have considering the effect of k on corners detected:**

Observations:

Here we see that when the k value is increased from 3 to 6 the number of corners detected and the matches decreased. Parameters were same like windowsize=3, threshold=0.3. This is because when the k value is increased according to the equation R value decreases and then when it is compared to Threshold it won't be able to get detected thus lesser number of corners.

b)**comparing the window size value: here we have considering the effect of window size on corners detected:**





Observation:

Here we see that when the window size value is increased from 2 to 5 the number of corners detected and the matches also increased.Parameters were same like k=0.01, threshold=0.1. This is because when the window size value is increased the neighbourhood size increases thus now will be detecting more corners with increased size.

So we see that :

- With increasing in threshold the number of corners detected decreases.
- With increasing value of ‘k’ the number of corners detected decreases.

- With the increase in window-size number of corner response also increases, because all it computes product of derivation(smooth) so increases the random parts which leads to detection of many corners in vicinity.
- The images using harris corner and myharris(self implemented) almost detect similar corners positions.

Strengths	Weakness
• User friendly(parameters can be changed using track bars)	• More execution time(due to use of python and low system specs)
	• Detector is not able to detect all the possible corners but still can detect most of them

5. References

a) https://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=cornerharris#cornerharris
b) http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_table_of_contents_feature2d/py_table_of_contents_feature2d.html
c) https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_matcher/py_matcher.html
d) https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_features_harris/py_features_harris.html

6. Code:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Oct 21 19:23:50 2017

@author: charusaxena231
"""

import cv2
import numpy as np
import math
import time

while(1):
```

```
    cap = cv2.VideoCapture(0)
    ret,img = cap.read()
    cap.release()

    time.sleep(3)

    cap = cv2.VideoCapture(0)
    ret,second = cap.read()
    cap.release()

    gray= cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    def HarrisHandler(i):
        n=cv2.getTrackbarPos('threshold','frame')
        k = cv2.getTrackbarPos('kvalue','frame')
        w = cv2.getTrackbarPos('windowsize','frame')
        if n==0:
            n= 0.5
        else:
            n = (n/10)*1

        if k==0:
            k==10
        else:
            k = (k/100)*2
        print("k value",k)

        if w==0:
            w=3

        eigen = cv2.cornerEigenValsAndVecs(gray, w, 3)
        mc= np.zeros(gray.shape)

        rows,cols = gray.shape
#=======================================================================
=======
        for i in range(rows):
            for j in range(cols):
                lam1 = eigen[i,j][0]
                lam2 = eigen[i,j][1]
                mc[i,j]=(lam1*lam2)-(k*(math.pow((lam1+lam2),2)))
#=======================================================================
=======
```

```
        corner = []
        minvalue,maxvalue,minloc,maxloc=cv2.minMaxLoc(mc)
        threshold = n * maxvalue

        for index , x in np.ndenumerate(mc):
                if x > threshold:
                    corner.append(index)

    #normalise

        criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 300, 0.1)
        total_corners = np.float32(corner)

        cv2.cornerSubPix( gray, total_corners, (5,5), (-1,-1), criteria )

#=====================================================================
=======
        uniqueCorners = np.unique(np.int0(total_corners),axis=0)


        flag=1
        img1 = cv2.cvtColor(gray,cv2.COLOR_GRAY2BGR)
#=====================================================================
=======
        #print(type(uniqueCorners))
        for i in uniqueCorners:
            x,y = i.ravel()
            cv2.rectangle(img1, (y-10,x-10), (y+10,x+10), (0,255,0))
            cv2.putText(img1,str(flag),(y,x),cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255))
            flag+=1
        kp1 = [cv2.KeyPoint(x[1], x[0], 3) for x in uniqueCorners]

        second_image_processing(threshold,k,img1,kp1,w)

    #def orb(img1,second1):
        #++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    def second_image_processing(thresh,k_value,img1,kp1,w):
        gray2= cv2.cvtColor(second, cv2.COLOR_BGR2GRAY)

        eigen = cv2.cornerEigenValsAndVecs(gray2, w, 3)
```

```
        print(type(eigen))
        mc1= np.zeros(gray2.shape)

        rows,cols = gray2.shape
        for i in range(rows):
            for j in range(cols):
                lam1 = eigen[i,j][0]
                lam2 = eigen[i,j][1]
                mc1[i,j] =(lam1*lam2)-(k_value*(math.pow((lam1+lam2),2)))



        corner2 = []
        minvalue,maxvalue,minloc,maxloc=cv2.minMaxLoc(mc1)


#=========================================================================
=======
#=========================================================================
======
        for index , x in np.ndenumerate(mc1):
                if x > thresh:
                    corner2.append(index)
    #normalise

        criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 300, 1)
        total_corners1 = np.float32(corner2)

        cv2.cornerSubPix( gray2, total_corners1, (5,5), (-1,-1), criteria )
        uniqueCorners1 = np.unique(np.int0(total_corners1),axis=0)


        print("unique",len(uniqueCorners1))
        flag=1
        second1 = cv2.cvtColor(gray2,cv2.COLOR_GRAY2BGR)
        print(flag)
        for i in uniqueCorners1:
            x,y = i.ravel()
            cv2.rectangle(second1, (y-10,x-10), (y+10,x+10), (0,255,0))
            cv2.putText(second1,str(flag),(y,x),cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255))
            flag+=1
        kp2 = [cv2.KeyPoint(x[1], x[0], 3) for x in uniqueCorners1]
```

```
    feature_matching(img1,second1,kp1,kp2)
    #++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


def feature_matching(img1,second1,kp1,kp2):

    orb = cv2.ORB_create()
    kp3,des1 = orb.compute(img1,kp1)
    kp4,des2 = orb.compute(second1,kp2)

    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

    matches = bf.match(des1,des2)

    matches = sorted(matches, key = lambda x:x.distance)
    print(kp3)
    img3 = cv2.drawMatches(img1,kp3,second1,kp4,matches,None,flags=2)
    cv2.destroyWindow("img")
    cv2.imshow("img",img3)


#harrisHandler(img,img2,gray)
cv2.imshow("frame",img)
cv2.createTrackbar('threshold','frame',0,5,HarrisHandler)
cv2.createTrackbar('kvalue','frame',0,8,HarrisHandler)
cv2.createTrackbar('windowsize','frame',0,8,HarrisHandler)


k = cv2.waitKey(0)
if k ==27:
    cv2.destroyAllWindows()
    break
elif k == ord('q'):
    cv2.destroyAllWindows()
```