#Nayak\_Varun Niranjan\_Assignment 3

#importing necessary libraries

import cv2

import numpy as np

import glob

#initiates live feed from webcam

cap = cv2.VideoCapture(0)

#the defined image is read using 'imread' function

target = cv2.imread('santa.jpg')

#setting a condition that atleat 30 matches are necessary to detect the object

MIN\_MATCH\_COUNT = 30

# feature description of points is done using SIFT

# initialize the AKAZE descriptor, then detect keypoints and extract local invariant descriptors from the image

feature\_detector = cv2.KAZE\_create()

#feature\_detector = cv2.xfeatures2d.SIFT\_create()

#extracts the keypoints and computes descriptors using SIFT

(tkp,tdes) = feature\_detector.detectAndCompute(target,None)

#initializes parameters for Flann-based matcher

FLANN\_INDEX\_KDTREE = 0

index\_params = dict(algorithm = FLANN\_INDEX\_KDTREE, trees = 5)

search\_params = dict(checks = 50)

#initializes the Flann-based matcher object

flann = cv2.FlannBasedMatcher(index\_params, search\_params)

#naming the output window

cv2.namedWindow("Matches",cv2.WINDOW\_AUTOSIZE)

#setting the threshold for the mask function

maskThreshold=10

while (True):

#capturing a frame (frame by frame)

ret, query = cap.read()

#obtaining the dimensions of the image

frame\_width, frame\_height, frame\_depth = query.shape

#extracts the keypoints and computes descriptors using SIFT

(qkp,qdes) = feature\_detector.detectAndCompute(query,None)

#create BFMatcher object

#match descriptors

matches = flann.knnMatch(tdes,qdes,k=2)

# store all the good matches as per Lowe's ratio test.

good = []

for m,n in matches:

if m.distance < 0.7\*n.distance:

good.append(m)

if len(good)>MIN\_MATCH\_COUNT:#length increases

src\_pts = np.float32([ tkp[m.queryIdx].pt for m in good ]).reshape(-1,1,2)

dst\_pts = np.float32([ qkp[m.trainIdx].pt for m in good ]).reshape(-1,1,2)

#finds the tranformation between two sets of points and output is masked

M, mask = cv2.findHomography(src\_pts, dst\_pts, cv2.RANSAC,5.0)

#convert a series to list and thereby return contiguous flattened self mask array

matchesMask = mask.ravel().tolist()

h,w,d = target.shape

#these dimensions decide the image points

pts1 = np.float32([ [0,0],[0,h-1],[w-1,h-1],[w-1,0] ]).reshape(-1,1,2)

dst = cv2.perspectiveTransform(pts1,M)

query = cv2.polylines(query,[np.int32(dst)],True,255,3, cv2.LINE\_AA)

#now providing image so that it will be overlayed on the checkerboard

#reading the image from the defined subject

images = glob.glob('grinch.jpg')

#the provided image is selected

currentImage = 0

#now the defined image is read using 'imread' function

replaceImg = cv2.imread(images[currentImage])

#obtaining the dimensions of the image (rows, columns & channels)

rows, cols, ch = replaceImg.shape

#these dimensions decide the image points

pts2 = np.float32([[0, 0], [cols, 0], [cols, rows], [0, rows]])

#perspective transform matrix is calculated from four pairs of points

M = cv2.getPerspectiveTransform(pts2, pts1)

#obtaining the dimensions of the image (rows, columns & channels)

rows, cols, ch = query.shape

#applies a perspective transformation to the image

dst = cv2.warpPerspective(replaceImg, M, (cols, rows))

#mask function is used for adding the two images

#maskThreshold is used to substract the black background from different image

ret, mask = cv2.threshold(cv2.cvtColor(dst, cv2.COLOR\_BGR2GRAY), maskThreshold, 1, cv2.THRESH\_BINARY\_INV)

#erode and dilate commands are used to denoise which are present in the image

mask = cv2.erode(mask, (3, 3))

mask = cv2.dilate(mask, (3, 3))

#both the images are added using the mask function so that the image will be overlayed on the checkerboard

for c in range(0, 3):

query[:, :, c] = dst[:, :, c] \* (1 - mask[:, :]) + query[:, :, c] \* mask[:, :]

#displaying the output image

cv2.imshow('img', query)

else:

#displays if the image is not recognised by the webcam

print ("Not enough matches: %d/%d" % (len(good),MIN\_MATCH\_COUNT))

matchesMask = None

#draw matches in green color

draw\_params = dict(matchColor = (0,255,0),

singlePointColor = None,

#draw only inliers

matchesMask = matchesMask,

flags = 2)

#it extracts the keypoints from the image

corr\_img = cv2.drawMatches(target,tkp,query,qkp,good,None,\*\*draw\_params)

#the input image is quite large, so we can resize it to fit within the screen

corr\_img = cv2.resize(corr\_img, (0,0), fx=0.5, fy=0.5)

cv2.imshow("Matches",corr\_img)

#applying condition for infinte loop to display

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

break

#release everything

cap.release()

cv2.destroyAllWindows()