comp9517 project part1 report

For this part of the project, we need input at least 2 arguments to execute python file.

$e.g.$: python test10.py Video_sample_1.mp4 "537,50,40,172" "244,74,50,60" "353,310,215,40"

$test10.py$ is the python file, $video_sample_1.mp4$ is the video we read from, and followed 3 arguments is the location we want to detect in the first frame of the video.

In this python file, first we use opencv built-in library to create surf object
$surf = cv2.xfeatures2d.SURF\_create()$

and then take the inputs followed to construct an array of bound box (eg: in this example we contain three array in this array because we input three bound boxes).

Next code is to generate different colours according to the lenght of input boxes, we use HUE color to do this because it is easily to generate a color just needing one number rather than RBG color.

Then we read file using the opencv built-in library ($cv2.VideoCapture().read()$), and take the first frame converted to gray picture using $cv2.cvtColor(firstframe, cv2.COLOR\_BGR2GRAY)$.

Next according to the box coordinnates we can get the correspond area in the frame using the function $calculate\_box$. Because the pictures is stored as an $numpy$ array, just choose its corresponding slice and return.

Then use function get_kp_des and $draw\_kp$ using the built-in library $surf.detectAndcompute$ and $cv2.drawKeyPoint$ to get the keypoints, descriptors and draw the keypoints on the pictures. This is the first frame keypoint, descriptors of the interested boxes and then show it on screen.

Next we also get the keypoints and descriptors of the first frame using $surf.detectAndCompute$ then we can do the descriptors matching using the built-in library $cv2.BFMatcher$ and $bf.knnMatch$ used by my function $knnmatch$.

In order to get gond enough matches so we use some criterias to jugde the matches and get the good enough list containing satisfied matches descriptoers. Now draw the match line between pair-wise keypoints using $cv2.drawMatchesKnn$ and show them. This is the first frame match of this video.

From now do the same loop to get next sequence frame as before. In order to get smooth and reasonable trajectory, before this loop we generate $kalmanfilter$ object outside, and for each time we calculate next frame, generate $kalmanfilter'sprediction$ inside the loop.

In this loop, we combine these bound boxes of these frame with the origin whole image together using $drawMatchesKnn$ with arguments $originimage, boundbox1, boundbox2$...pair-wise sequence, Next, calculate the bound box we want to draw around the interested areas using the function $noisyleaveout$ to remove some noisy keypoint and return the good x and y coordinates. In this function use some another filter such as $boxfilter$ and built-in library $cv2.minmaxloc$. The concept of leaving out noisy keypoint is the highest density of the keypoint area has the more probability of the interested area we want. The box filter calculate the highest grayscale point in a curtain area and the function $minmaxloc$ reuturn the minimum and maximum location of this image.

Because the $matches$ and $keypoint$ object does not contain the x, y coordinates straightly so they need to be paraphrase using $DMatch.queryIdx$ and $DMatch.trainIdx$ to get the $queryId$ and $trainId$ pair-wise, and then from $keypoint[queryIdx].pt$ and $keypoint[trainIdx].pt$ can get the corresponding x and y coordinates of the bound boxes. And after using these some techniques to leave out noisy matches, the remaining matches can provide more accuray x, y coordinates and after some simple calculations returing the x, y point we want to draw rectangle from.

Then calculate the x and y coordinates after $kalmanfilter$ to draw a bound box as well as the center of bound box.

Next gathering all the center point together to draw the line pair-wise sequence. Then add the FPS calculating before using $cv2.getTickFrequency$ divided by the $cv2.getTickCount($ aftermerge the bound box image and origin image$)-cv2.getTickCount($merge them before$)$ though the $cv2.putText$ function to show FPS on the left top of the display image sequence.

Finally, when read the end of the video, stop and release the video.