CS682 : Computer Vision Submissions

Brinston Gonsalves

Homework 4

Download Python Script
Download All Output Images in High Resolution

```
How to use :
1. Download the python script.
2. Create a images folder in the same directory as the script.
3. Copy all the provided test images to the newly created folder on which the operations are required to be performed

** Please Use Open Image In new Tab to look at larger images.


import cv2
import numpy as np
import scipy as sp
import random
from skimage.transform import warp, AffineTransform
from skimage.measure import ransac
```

The get_SIFT_features function takes an image as input , finds the SIFT features in it and returns the keypoints and descriptors in the image. For calculating the SIFT features, we use the functions provided by OpenCV.

```
def get_SIFT_features(img) :

    # input : Image file
    # output : keypoints, descriptors

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

    sift = cv2.xfeatures2d.SIFT_create()

    keypoints, descriptors = sift.detectAndCompute(gray,None)

    return keypoints, descriptors
```

The get_best_matches function takes the SIFT desciptors of two images and attempts to match them. In the knnMatch function under BFMatcher(), we have set the value of k as 2, since we want to create 2 correspondences for each match. For discarding the bad correspondences, I am using the Lowe's method that states that a factor of 75% is a good balance to discard the false possitives. Thus, only good matches are considered for further processing.

```
def get_best_matches(descriptors1, descriptors2) :

    # input : descriptors of two image files
    # output : selected matches

    matches = cv2.BFMatcher().knnMatch(descriptors1, descriptors2, 2)
    selected_matches = [m for m, n in matches if m.distance < 0.75 * n.distance]

    return selected_matches
```

The draw_keypoints function takes the image and its SIFT desciptors as an input and returns the copy of the original image with the keypoints drawn on it. We utilized the drawKeypoints function provided by OpenCV which plotted the keypoints along with their orientations.

```
def draw_keypoints(keypoints, img) :
    # output : image with plotted keypoints

    img_out = img.copy()
```

```
    cv2.drawKeypoints(img,keypoints, img_out, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
    return img_out
```

The draw_matches function takes the 2 images, its SIFT keypoints with the calculated matches as input , appends the two images side by side and draws lines of random color from one image to other for every match using the keypoints. We finally return a image that has 2 images appended with corresponding keypoints displayed by lines.

```
def draw_matches(img1, img2, matches, keypoints1, keypoints2) :
    h1, w1 = img1.shape[:2]
    h2, w2 = img2.shape[:2]
    view = sp.zeros((max(h1, h2), w1 + w2, 3), sp.uint8)
    view1 = img1.copy()
    view2 = img2.copy()
    view[:h1, :w1] = view1
    view[:h2, w1:] = view2

    for m in matches:
        color = tuple([sp.random.randint(0, 255) for _ in range(3)])
        cv2.line(view, (int(keypoints1[m.queryIdx].pt[0]), int(keypoints1[m.queryIdx].pt[1])),
                               (int(keypoints2[m.trainIdx].pt[0] + w1), int(keypoints2[m.trainIdx].pt[1])),
                               (random.randint(0, 255), random.randint(0, 255), random.randint(0, 255)),
                               lineType=cv2.LINE_8)

    return view
```

The get_Affine_Transformation function takes the keypoints and descriptors of the two images with the generated matches. Here we consider the keypoints of image 1 as the source points and the keypoints of image 2 as the destination points, We then use the ransac function with Affine Transform to transform the image such that the source points and destination points correspond to the same position. Since ransac algorithm is based on sampling, we provide it with min_samples, residual_threshold and max_trials values.

```
def get_Affine_Transformation(keypoints1, keypoints2, descriptors1, descriptors2, matches) :


    src_pts = np.float32([keypoints1[match.queryIdx].pt for match in matches])
    dst_pts = np.float32([keypoints2[match.trainIdx].pt for match in matches])

    M, _ = ransac((src_pts, dst_pts), AffineTransform, min_samples=4, residual_threshold=2, max_trials=200)

    return M
```

The get_Homograph_transformation function takes the SIFT desciptors, keypoints and corresponding matches as an input and returns a transformation matrix that is a transformation between the source points and destination points. Here, for calculating the transformation matrix, we have used the findHomography function provided by OpenCV.

```
def get_Homograph_transformation (keypoints1, keypoints2, descriptors1, descriptors2, matches) :
    src_pts = np.float32([keypoints1[match.queryIdx].pt for match in matches])
    dst_pts = np.float32([keypoints2[match.trainIdx].pt for match in matches])

    M, status = cv2.findHomography(src_pts,dst_pts,cv2.RANSAC,5.0)

    return M, status
```

The get_points function takes the keypoints of an image as input and returns a list of coordinate points for each corresponding keypoint.

```
def get_points(keypoints) :

    point_list = [x.pt for x in keypoints]
    return point_list
```

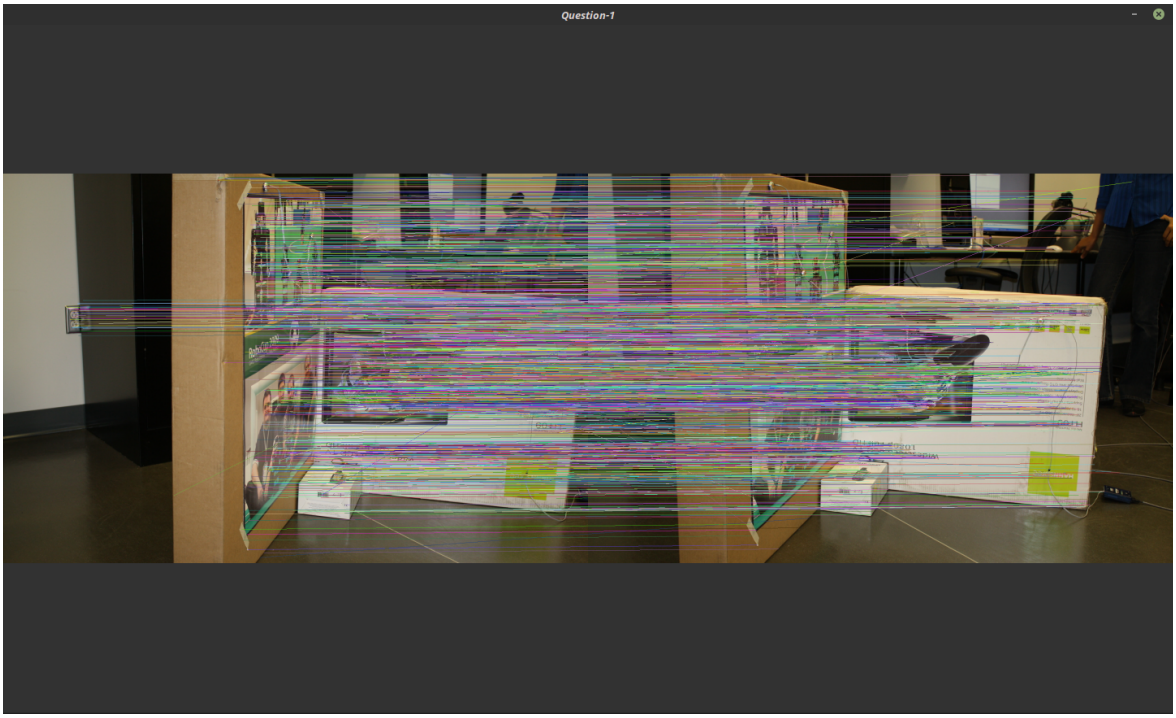
The warp_images function takes the 2 images as input along with the generated homography matrix, transforms the second image according to the given homography and we place the first image over the generated image as a plane. Depending upon the homography we calculate the size of the output image. We then return the entire processed image that contains both the images with the matched portions intersecting over one another.

```
def warp_images(img1, img2, homography):
    # Warps second image to plane of first image based on provided homography.
```

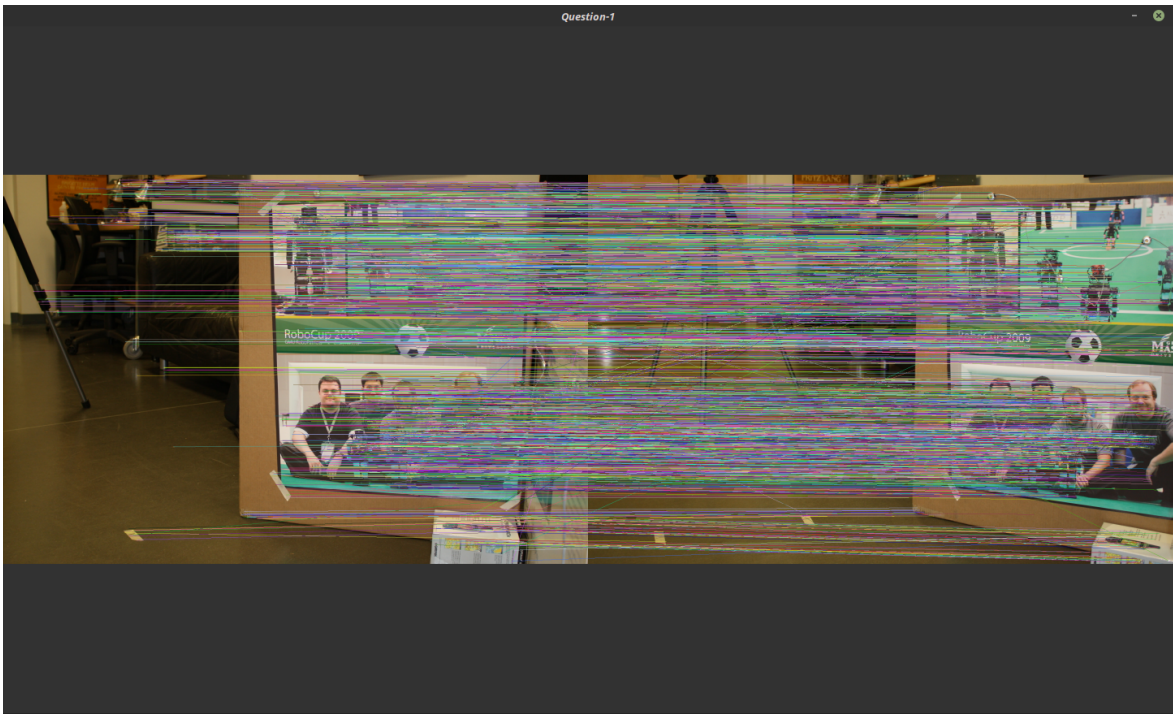
```
    # Warps img2 to img1.

    h1, w1 = img1.shape[:2]
    h2, w2 = img2.shape[:2]


    points1 = np.array([[0, 0], [0, h1], [w1, 0], [w1, h1]],np.float32).reshape(-1,1,2)
    points2 = np.array([[0, 0], [0, h2], [w2, 0], [w2, h2]],np.float32).reshape(-1,1,2)
    points2_pers = cv2.perspectiveTransform(points2, homography)

    all_points = np.concatenate((points1,points2_pers))
    [x_min, y_min] = np.int32(all_points.min(axis=0).ravel())
    [x_max, y_max] = np.int32(all_points.max(axis=0).ravel())

#    print(x_min,y_min,x_max,y_max)

    smallest_coords = [-x_min, -y_min]
    homography_translation = np.array([[1, 0, smallest_coords[0]], [0, 1, smallest_coords[1]], [0, 0, 1]])

    x_size = x_max - x_min
    y_size = y_max - y_min

    homography_final = homography_translation.dot(homography)

    output_img = cv2.warpPerspective(img2, homography_final, (x_size, y_size))
    output_img[smallest_coords[1]:h1+smallest_coords[1], smallest_coords[0]:w1+smallest_coords[0]] = img1

    return output_img
```

Question 1 :

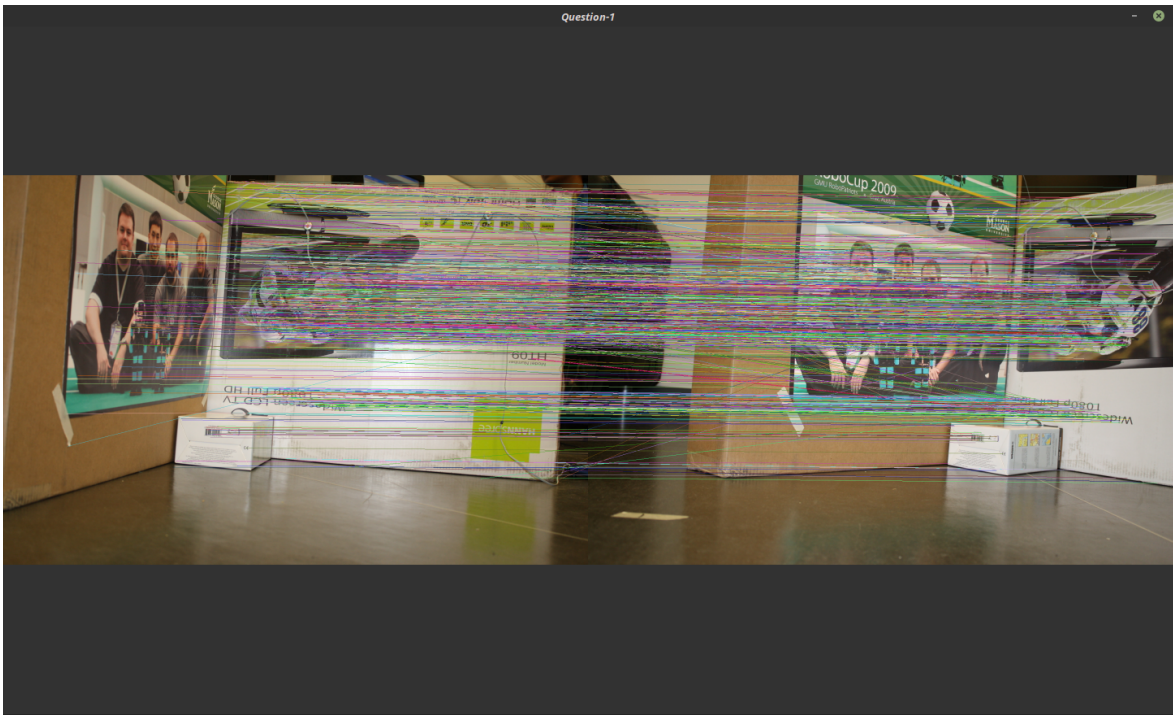
IMG_1188.JPG & IMG_1189.JPG



IMG_1200.JPG & IMG_1201.JPG



IMG_1210.JPG & IMG_1211.JPG




```
def question_1() :
    '''
    img1 = cv2.imread("images/IMG_1188.JPG")
    img2 = cv2.imread("images/IMG_1189.JPG")
    '''

    '''
```

```
    img1 = cv2.imread("images/IMG_1200.JPG")
    img2 = cv2.imread("images/IMG_1201.JPG")
    '''


    img1 = cv2.imread("images/IMG_1210.JPG")
    img2 = cv2.imread("images/IMG_1211.JPG")



    kp1, des1 = get_SIFT_features(img1)
    kp2,des2 = get_SIFT_features(img2)
    matches = get_best_matches(des1,des2)
    img_out = draw_matches(img1,img2,matches,kp1,kp2)
    cv2.namedWindow('Question-1',cv2.WINDOW_NORMAL)
    cv2.imshow("Question-1",img_out)
    cv2.waitKey(0)
```

Question 2 :

Here I have shown the Affine and Homography Transformation Matrix under the corresponding images.
The Affine transform as we recieve it is a 3x3 matrix, but since the 3rd row is just a Identity matrix, I have displayed just the first two rows of it.

```
def question_2() :


    img1 = cv2.imread("images/IMG_1188.JPG")
    img2 = cv2.imread("images/IMG_1189.JPG")

    '''
    ------ Transformation Matrix - Affine ------
    [[  1.02333808e+00  -1.60811860e-02   1.85181549e+02]
     [  4.38329502e-04   1.00558400e+00   3.42286682e+00]]

    ------ Transformation Matrix - Homography ------
    [[  9.77620304e-01   1.54075064e-02  -1.81539749e+02]
     [ -3.50916205e-04   9.94283795e-01  -3.31968951e+00]
     [  8.19764239e-08  -1.58300352e-07   1.00000000e+00]]
    '''

    '''
    img1 = cv2.imread("images/IMG_1200.JPG")
    img2 = cv2.imread("images/IMG_1201.JPG")
    '''

    '''
    ------ Transformation Matrix - Affine ------
    [[  1.01881707e+00   2.10916959e-02  -3.25989594e+02]
     [ -1.08655766e-02   9.99276400e-01   6.10512733e+00]]

    ------ Transformation Matrix - Homography ------
    [[  7.59037316e-01  -2.20614355e-02   4.27962555e+02]
     [ -2.04153638e-02   9.02750790e-01   3.03729134e+01]
     [ -8.79734143e-05  -3.05624349e-06   1.00000000e+00]]
    '''

    '''
    img1 = cv2.imread("images/IMG_1210.JPG")
    img2 = cv2.imread("images/IMG_1211.JPG")
    '''

    '''
    ------ Transformation Matrix - Affine ------
    [[  1.07259655e+00  -1.49559788e-02  -7.87505737e+02]
     [  5.73794059e-02   1.00863004e+00  -1.36618805e+02]]

    ------ Transformation Matrix - Homography ------
    [[  4.61558968e-01  -8.99012107e-03   8.97355774e+02]
     [ -1.37915149e-01   8.11669171e-01   1.66353638e+02]
     [ -1.85847326e-04  -9.10070230e-06   1.00000000e+00]]
    '''

    kp1, des1 = get_SIFT_features(img1)
```

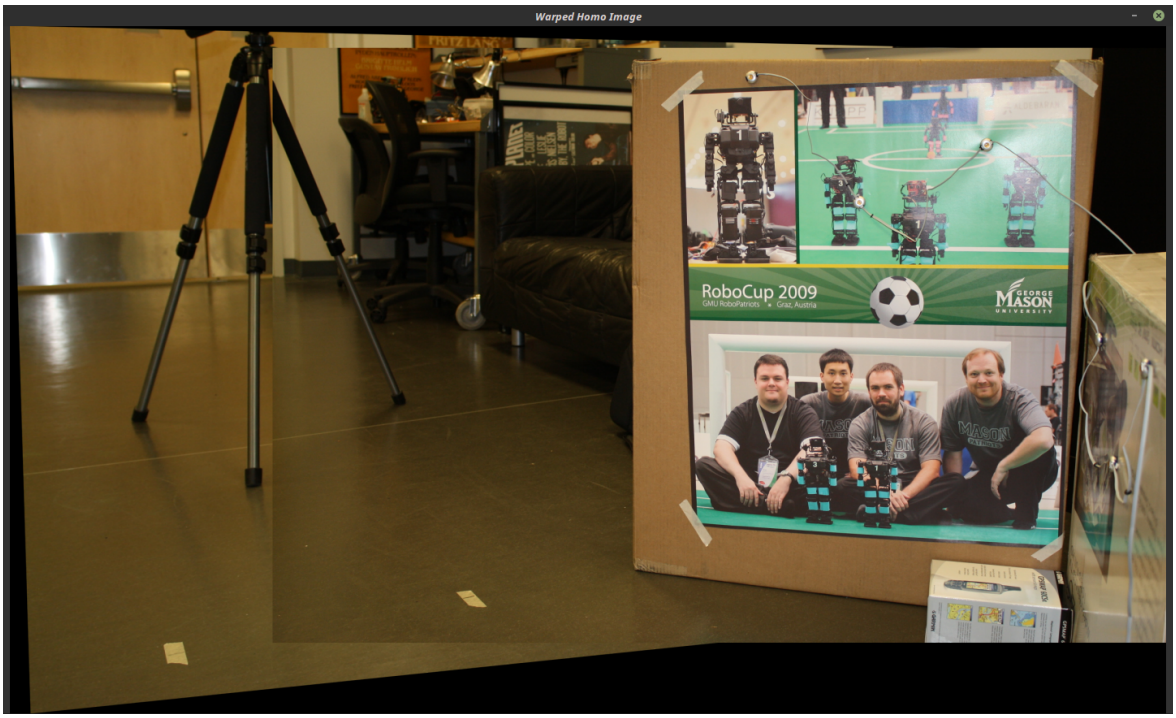
```
kp2, des2 = get_SIFT_features(img2)

matches = get_best_matches(des1, des2)

tranformation_matrix_affine = get_Affine_Transformation(kp1,kp2,des1,des2, matches)
print ("------ Transformation Matrix - Affine ------")
print(np.float32(tranformation_matrix_affine._inv_matrix)[0:2,:])

tranformation_matrix_homography, _ = get_Homograph_transformation(kp1,kp2,des1,des2, matches)
print("------ Transformation Matrix - Homography ------")
print(np.float32(tranformation_matrix_homography))
```

Question 3 :

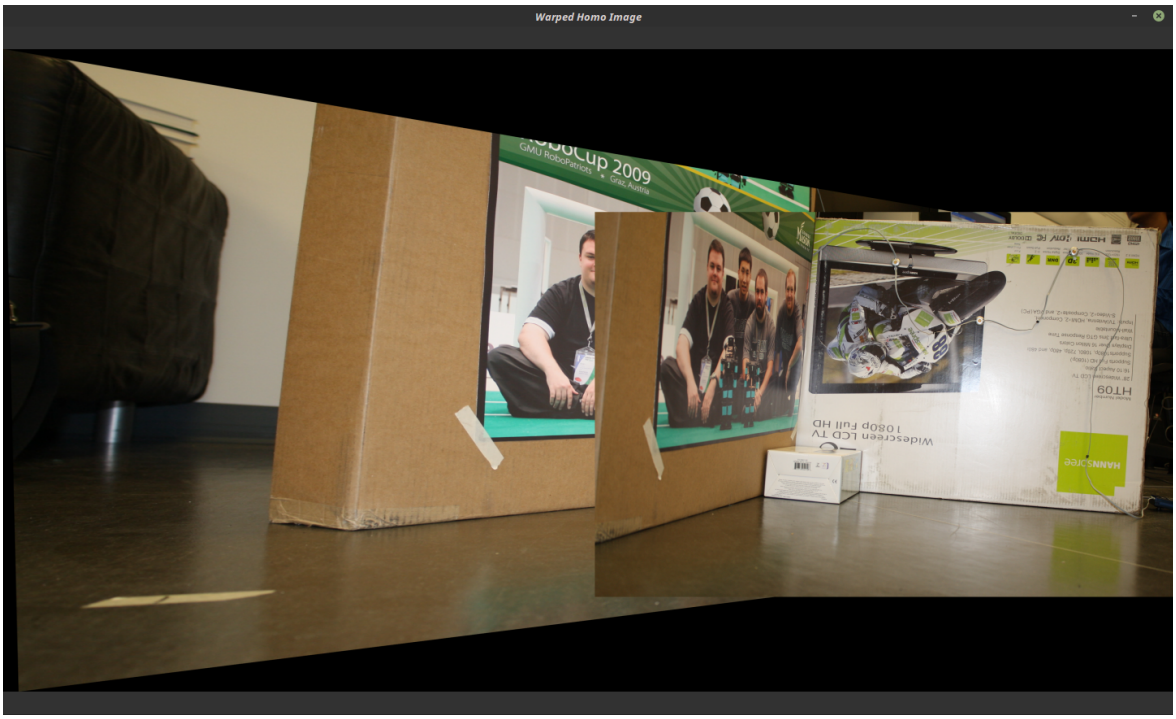
Here, we see that all the images are warped together appropriately except for the third image where, since the difference between the input images IMG_1210.JPG & IMG_1211.JPG is too large, the output looks relatively more distorted. The output is correct though, since we can see that the white box from both the images is warped perfectly together.

IMG_1188.JPG & IMG_1189.JPG

IMG_1200.JPG & IMG_1201.JPG



IMG_1210.JPG & IMG_1211.JPG

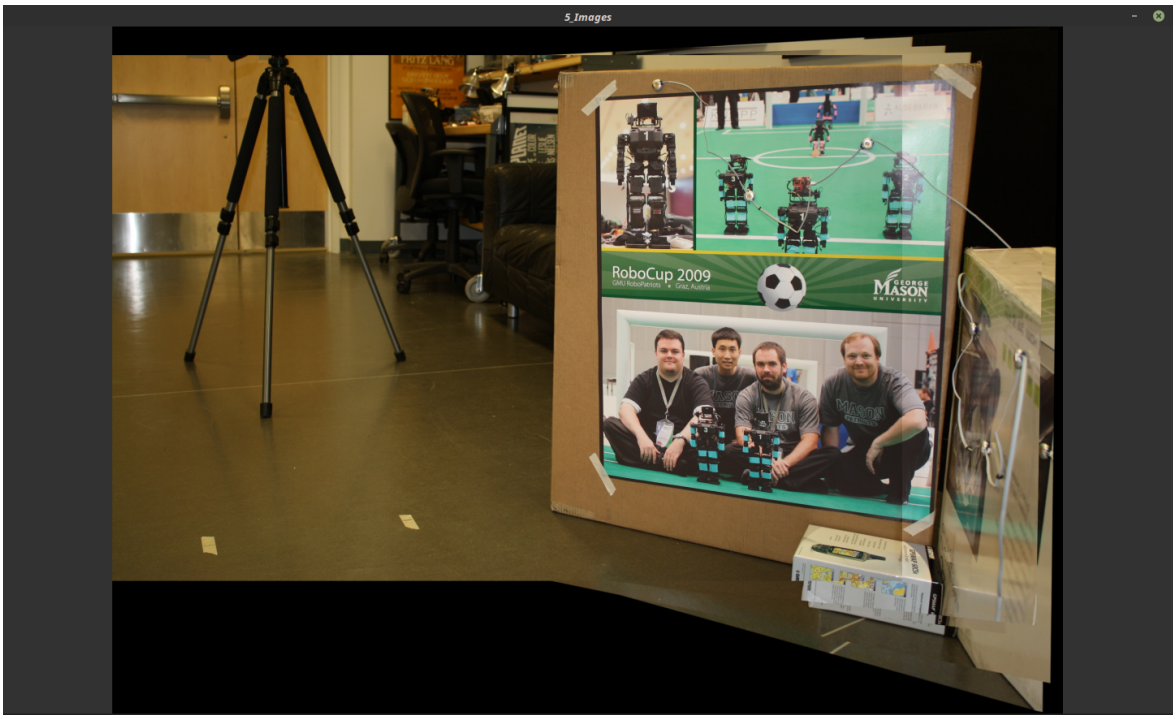


IMG_1211.JPG & IMG_1212.JPG

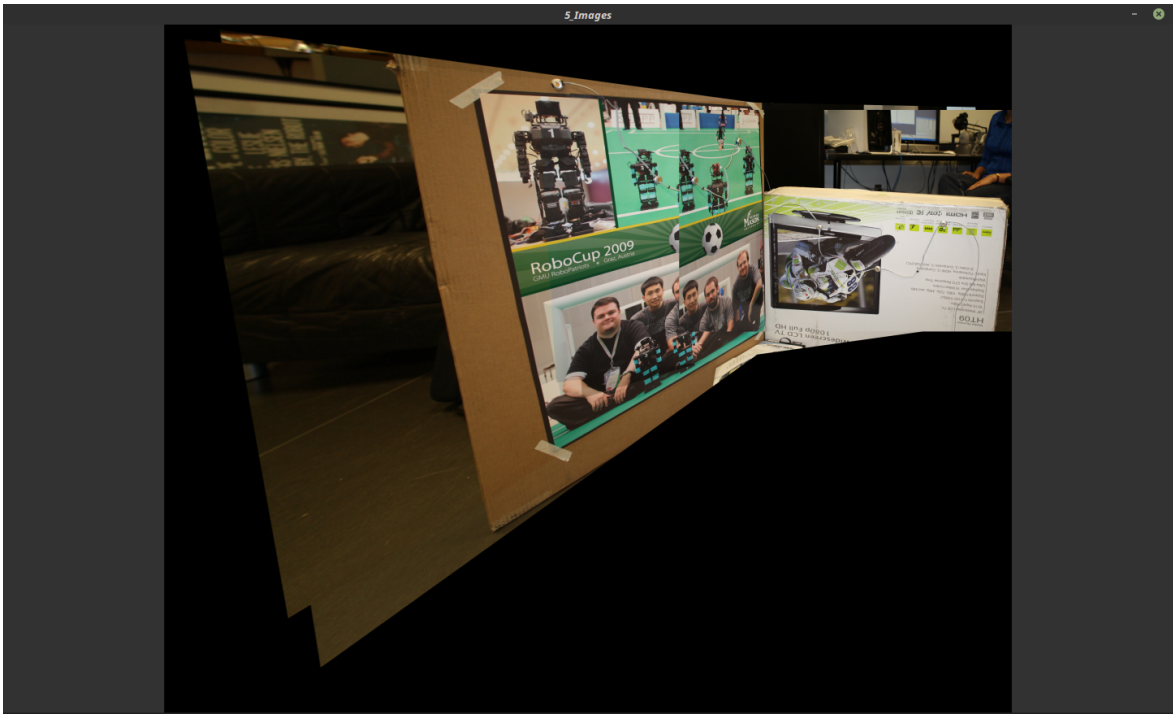

```
def question_3():
    '''
    img1 = cv2.imread("images/IMG_1188.JPG")
    img2 = cv2.imread("images/IMG_1189.JPG")
    '''

    '''
    img1 = cv2.imread("images/IMG_1200.JPG")
    img2 = cv2.imread("images/IMG_1201.JPG")
    '''

    '''
    img1 = cv2.imread("images/IMG_1210.JPG")
    img2 = cv2.imread("images/IMG_1211.JPG")
    '''

    img1 = cv2.imread("images/IMG_1211.JPG")
    img2 = cv2.imread("images/IMG_1212.JPG")

    kp1, des1 = get_SIFT_features(img1)
    kp2, des2 = get_SIFT_features(img2)

    matches = get_best_matches(des1, des2)

    tranformation_matrix_affine = get_Affine_Transformation(kp1, kp2, des1, des2, matches)
    tranformation_matrix_affine = np.float32(tranformation_matrix_affine._inv_matrix)[0:2, 0:3]
    print("------ Transformation Matrix - Affine ------")
    print(tranformation_matrix_affine)

    tranformation_matrix_homography, status = get_Homograph_transformation(kp1, kp2, des1, des2, matches)
    print("------ Transformation Matrix - Homography ------")
    print(np.float32(tranformation_matrix_homography))

    warped_homo_image = warp_images(img2, img1, tranformation_matrix_homography)
    cv2.namedWindow("Warped Homo Image", cv2.WINDOW_NORMAL)
    cv2.imshow("Warped Homo Image", warped_homo_image)
    cv2.waitKey(0)
```

Question 4 :

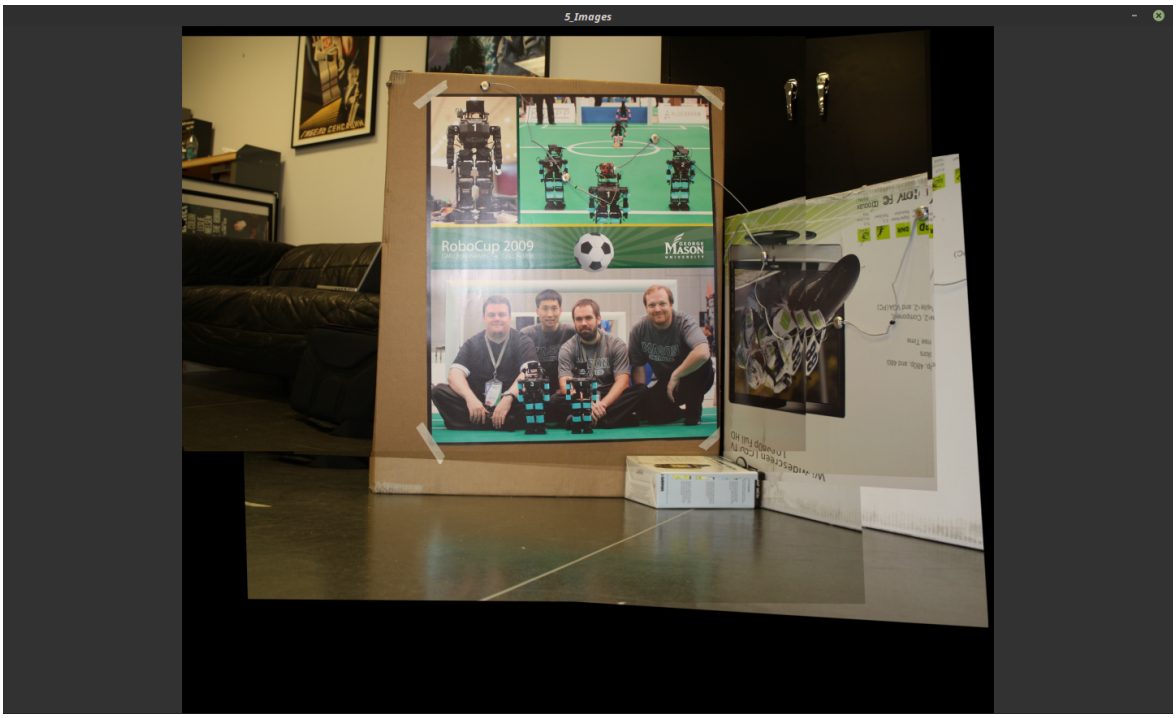
IMG_1197.JPG to IMG_1201.JPG



IMG_1202.JPG to IMG_1206.JPG



IMG_1213.JPG to IMG_1217.JPG


5_Images
RoboCup 2009