

Labb2

Elias Berglin

29th November 2021

1 Week 3

1.1 Assignment

1.1.1 Hough transform

The code I used to transform and draw the graphs can be found in appendix A.
Figure 1 shows task a/b and figure 2 shows task c.

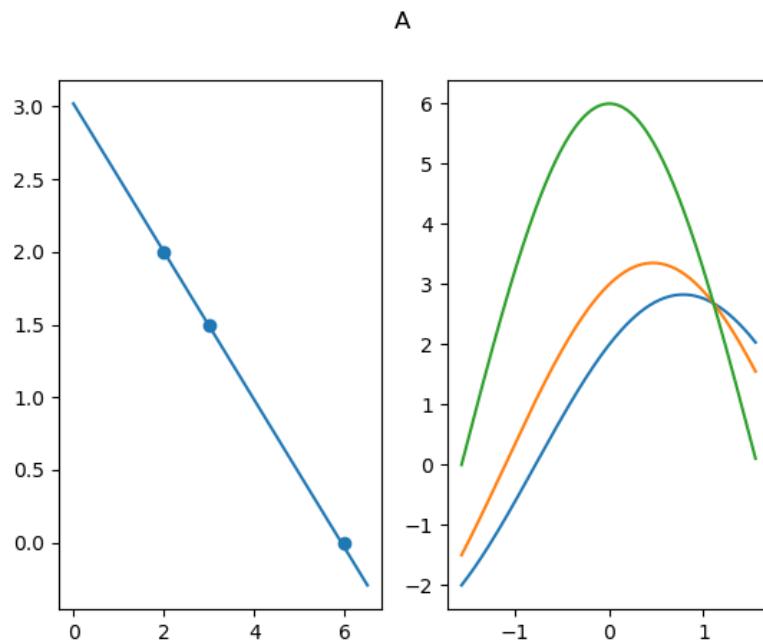


Figure 1: Task a/b. Image space on the left and hough space on right

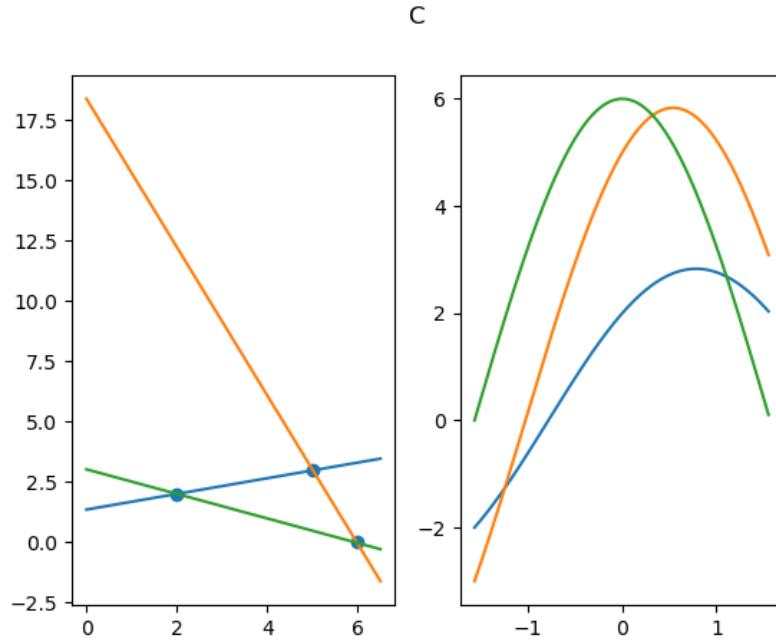


Figure 2: Task c. Image space on the left and hough space on right

1.1.2 Feature descriptors

I have read 2 comparison papers between different feature descriptors. SIFT seems to always be the most accurate. But it is also not as fast. Considering we have an AR application we need to have a somewhat faster algorithm to make it easier to track features. These papers say that ORB is almost as good as SIFT but much faster so I would go with ORB for my AR application. [1] [2]

1.1.3 Feature detection (and matching)

Python code used for the assignment can be found in appendix B. Figure 3 shows the features extracted from the first images using SIFT and my Harris implementation. Figure 4 shows the features extracted from the second image using SIFT and my Harris implementation.

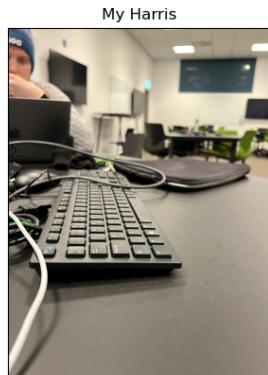


Figure 3: First image with Harris and SIFT



Figure 4: Second image with Harris and SIFT

1.2 Reflection

This week was interesting. I thought that feature detection/descriptors was a really interesting subject. The most interesting part was reading the research papers where they benchmarked different algorithms for finding features. I had a hard time grasping the concept of Hough transform but after watching the YouTube course I quickly understood how it worked. This has been the fact for all of the weeks. If I don't fully understand something during the lecture the YouTube course have always cleared it up for me.

My Harris detector did not perform well compared to SIFT. But when I compared it to the OpenCV implementation it was not far off. From reading the papers [1] [2] I now understand that SIFT is really good at finding features and thus it is understandable that Harris does not perform as well

2 Week 4

2.1 Assingment

2.1.1 2D Transformations

The question was for a similarity transform but we are not doing any scaling so we are really doing an euclidian transformation so the folling matrix was created based on the lecture slides:

$$\begin{pmatrix} \cos(15 * \frac{\pi}{180}) & -\sin(15 * \frac{\pi}{180}) & 3 \\ \sin(15 * \frac{\pi}{180}) & \cos(15 * \frac{\pi}{180}) & -2 \\ 0 & 0 & 1 \end{pmatrix} \quad (1)$$

To calculate the transfromation matrix (homography matrix) the code in appendix C was used and the following matrix was calculated:

$$\begin{pmatrix} 1.8 & 0.67 & 1 \\ 0.6 & 0.67 & 1 \\ 0.2 & 0 & 1 \end{pmatrix} \quad (2)$$

2.1.2 Stereo estimation

I had ha hard time understanding the website interface so I just took the first algorithm with a paper connected to it. So I chose RAFT-Stereo [3]. This algorithm is based on RAFT [4] and extends the concepts of GRU

2.1.3 Plane Sweep

Full code for Plane Sweep algorithm can be found in D. The images used as input can be found in figure 5, the depth image can be found in figure 6.

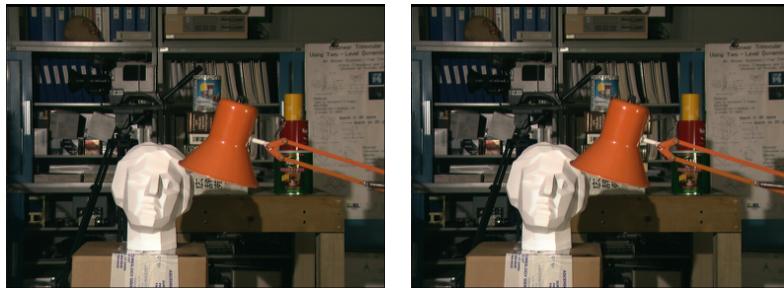


Figure 5: Imaged used to for Plane Sweep algorithm

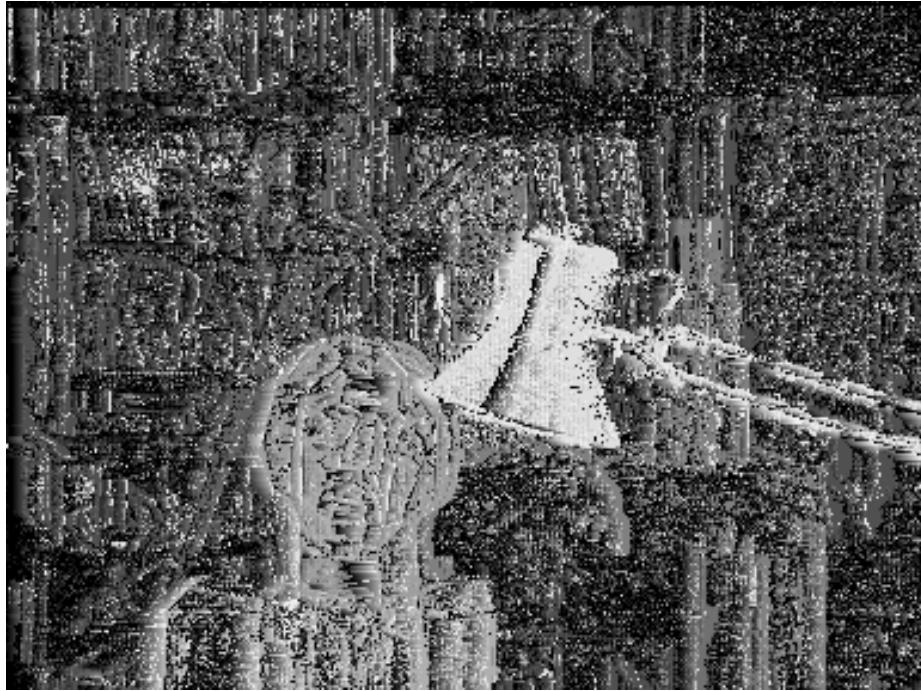


Figure 6: Images produced from Plane Sweep algorithm

2.1.4 Stitching

The code used to stitch images can be found in appendix E My own implementation of RANSAC and calculation of the homography matrix was used. To find feature points ORB was used, implemented by OpenCV. To match feature points a Brute Force Matcher from OpenCV was used. This was then used in KNN to get K best matches for each point. I then use a perspective warp to warp one image and then apply the other image on top of that image. The images used for stitching can be found in figure 7 and the produced output can be found in figure 8.

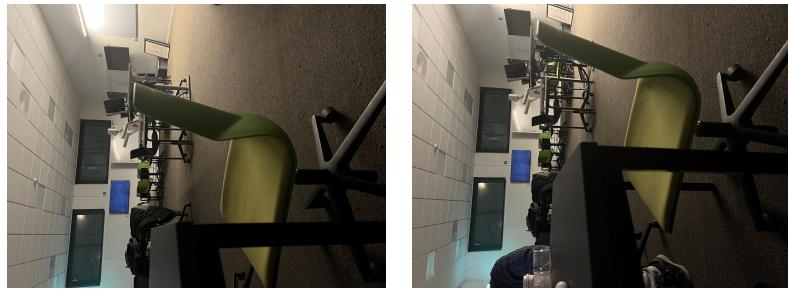


Figure 7: Images used for Stitching

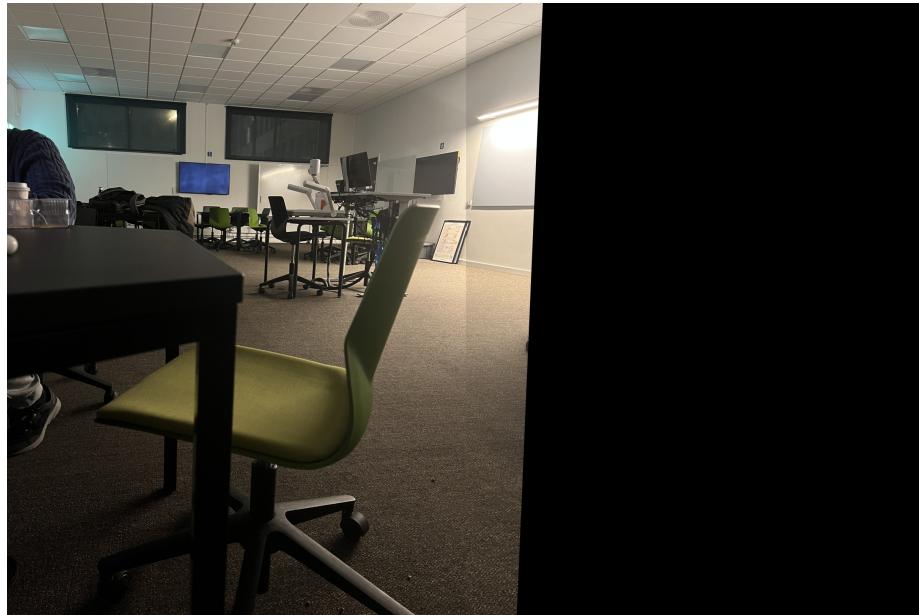


Figure 8: Images produced image Stitching

2.2 Reflection

This week was definitely the toughest so far in regards to assignments. It took longer than I want to admit to grasp the concepts of homography matrices and how transformations work with images. It was hard to find good information on how Plane Sweep worked and it only became clear on how to implement it after the seminar. I started in the wrong end this week. The first thing I started with was image stitching. The implementation of this became much easier when I fully understood how homography matrices works. It was really interesting to look at different methods for stereo matching.

References

- [1] BR Kavitha, G Ramya, and G Priya. “Performance comparison of various feature descriptors in object category detection application using SVM classifier”. In: (2019).
- [2] Shaharyar Ahmed Khan Tareen and Zahra Saleem. “A comparative analysis of sift, surf, kaze, akaze, orb, and brisk”. In: *2018 International conference on computing, mathematics and engineering technologies (iCoMET)*. IEEE. 2018, pp. 1–10.
- [3] Lahav Lipson, Zachary Teed, and Jia Deng. “RAFT-Stereo: Multilevel Recurrent Field Transforms for Stereo Matching”. In: *arXiv preprint arXiv:2109.07547* (2021).
- [4] Zachary Teed and Jia Deng. “Raft: Recurrent all-pairs field transforms for optical flow”. In: *European conference on computer vision*. Springer. 2020, pp. 402–419.

Appendix A Python code for drawing Hou Transform

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 def getCrossLine(y1, y2, x):
5     idx = np.argwhere(np.diff(np.sign(y1 - y2)) != 0)
6     theta = x[idx]
7     r = y1[idx]
8     ixa = np.arange(start=0, stop=7, step=0.5)
9     iya = np.zeros(len(ixa))
10    for i in range(len(ixa)):
11        iya[i] = ((-np.cos(theta)/np.sin(theta))*ixa[i]) + (r/np.
12            sin(theta))
13    return ixa, iya
14
15 thetas = np.deg2rad(np.arange(-90, 90))
16
17 xa = [2, 3, 6]
18 ya = [2, 1.5, 0]
19
20 ra = []
21 for i in range(len(xa)):
22     temp = []
23     for k in range(len(thetas)):
24         temp.append(xa[i]*np.cos(thetas[k]) + ya[i] * np.sin(thetas
25                         [k]))
26     ra.append(temp)
27
28 xc = [2, 5, 6]
29 yc = [2, 3, 0]
30 rc = []
31 for i in range(len(xc)):
32     temp = []
33     for k in range(len(thetas)):
34         temp.append(xc[i]*np.cos(thetas[k]) + yc[i] * np.sin(thetas
35                         [k]))
36     rc.append(temp)
37
38 ra = np.asarray(ra)
39 rc = np.asarray(rc)
40
41 ixa, iya = getCrossLine(ra[0], ra[1], thetas)
42 ixc1, iyc1 = getCrossLine(rc[0], rc[1], thetas)
43 ixc2, iyc2 = getCrossLine(rc[1], rc[2], thetas)
44 ixc3, iyc3 = getCrossLine(rc[0], rc[2], thetas)
45
46 fig, (ax1, ax2) = plt.subplots(1,2)
47 fig.suptitle('A')
48 ax1.scatter(xa, ya)
49 ax1.plot(ixa, iya)

```

```

51 for r in ra:
52     ax2.plot(thetas, r)
53
54
55 fig2, (cx1, cx2) = plt.subplots(1, 2)
56 fig2.suptitle('C')
57 cx1.scatter(xc, yc)
58 cx1.plot(ixc1, iyc1)
59 cx1.plot(ixc2, iyc2)
60 cx1.plot(ixc3, iyc3)
61 for r in rc:
62     cx2.plot(thetas, r)
63 plt.show()

```

Appendix B Harris implementation

```

1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5
6 def harris(img_name, window_size, k, threshold):
7     img = cv2.imread(img_name)
8     gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
9     img_gauss = cv2.GaussianBlur(gray, (3, 3), 0)
10    h = img.shape[0]
11    w = img.shape[1]
12
13    matrix_r = np.zeros((h, w))
14
15    dx = cv2.Sobel(img_gauss, cv2.CV_64F, 1, 0, ksize=3)
16    dy = cv2.Sobel(img_gauss, cv2.CV_64F, 0, 1, ksize=3)
17
18    dx2 = np.square(dx)
19    dy2 = np.square(dy)
20    dxy = dx * dy
21
22    offset = int(window_size/2)
23    print("Finding corners...")
24    for y in range(offset, h-offset):
25        for x in range(offset, w-offset):
26            sx2 = np.sum(dx2[y-offset:y+1+offset, x-offset:x+1+offset])
27            sy2 = np.sum(dy2[y-offset:y+1+offset, x-offset:x+1+offset])
28            sxy = np.sum(dxy[y-offset:y+1+offset, x-offset:x+1+offset])
29
30            H = np.array([[sx2, sxy], [sxy, sy2]])
31            det = np.linalg.det(H)
32            tr = np.trace(H)
33            R = det - k * (tr ** 2)
34            matrix_r[y - offset, x - offset] = R
35
36    cv2.normalize(matrix_r, matrix_r, 0, 1, cv2.NORM_MINMAX)

```

```

37     for y in range(offset, h - offset):
38         for x in range(offset, w - offset):
39             value = matrix_r[y, x]
40             if value > threshold:
41                 cv2.circle(img, (x, y), 3, (0, 255, 0))
42
43     plt.figure("Harris detector")
44     plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB)), plt.title("My
45     Harris")
46     plt.xticks([]), plt.yticks([])
47     plt.show()
48
49 def cv2Harris(img_name):
50     img = cv2.imread(img_name)
51     gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
52
53     harris = cv2.cornerHarris(gray, 2, 3, 0.04)
54     harris = cv2.dilate(harris, None)
55     img[harris > 0.01 * harris.max()] = [0, 0, 255]
56
57     plt.figure("Harris detector")
58     plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB)), plt.title(""
59     CV2 Harris")
60     plt.xticks([]), plt.yticks([])
61     plt.show()
62
63 def cv2Sift(img_name):
64     img = cv2.imread(img_name)
65     gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
66     sift = cv2.SIFT_create()
67     kp = sift.detect(gray, None)
68
69     img = cv2.drawKeypoints(gray, kp, img)
70     plt.figure("SIFT Detector")
71     plt.imshow(img), plt.title("SIFT")
72     plt.xticks([]), plt.yticks([])
73     plt.show()
74
75 img_name = "start2.jpeg"
76
77 harris(img_name, 5, 0.04, 0.30)
78 cv2Harris(img_name)
79 cv2Sift(img_name)

```

Appendix C Python code for calculating the homography matrix

```

1 import numpy as np
2
3 np.set_printoptions(suppress=True)
4
5 def calcHomo(p1, p2):
6     A = []
7     for i in range(0, len(p1)):

```

```

8     x, y = p1[i][0], p1[i][1]
9     u, v = p2[i][0], p2[i][1]
10    A.append([x, y, 1, 0, 0, 0, -u*x, -u*y, -u])
11    A.append([0, 0, 0, x, y, 1, -v*x, -v*y, -v])
12    A = np.asarray(A)
13    U, S, Vh = np.linalg.svd(A)
14    L = Vh[-1,:] / Vh[-1,-1]
15    H = L.reshape(3, 3)
16    return H
17
18 def calcNewPoints(src, H):
19     newPoints = []
20     for x,y in src:
21         vec = np.asarray([x,y,1])
22         mult = H.dot(vec)
23         newPoint = mult/mult[-1]
24         newPoints.append([newPoint[0], newPoint[1]])
25
26
27     return np.asarray(newPoints)
28
29 src = np.asarray([[0,0],[0,3],[5,3],[5,0]])
30 dst = np.asarray([[1,1],[3,3],[6,3],[5,2]])
31
32 H = calcHomo(src, dst)
33
34 print(H)
35
36 print(calcNewPoints(src, H))

```

Appendix D Python code for Plane Sweep

```

1 import cv2
2 import numpy as np
3
4 def depth(img1, img2, distance):
5     height, width = img1.shape
6     disparity = 0
7     depthArray = np.zeros((height, width))
8     maxDepth = 255/distance
9     for y in range(0,height):
10         for x in range(0, width):
11             prev_min_val = np.inf
12             for d in range(distance):
13                 temp_min_value = float(img1[y][x]) - float(img2[y][x-d])
14                 min_value = temp_min_value * temp_min_value
15                 if min_value < prev_min_val:
16                     prev_min_val = min_value
17                     disparity = d
18             depthArray[y][x] = disparity * maxDepth
19             cv2.imwrite("sd.png", depthArray)
20
21
22

```

```

23
24
25
26 img1 = cv2.imread("ps1.ppm", cv2.IMREAD_GRAYSCALE)
27 img2 = cv2.imread("ps2.ppm", cv2.IMREAD_GRAYSCALE)
28
29 stereo = cv2.StereoSGBM_create(numDisparities=16, blockSize=5)
30 disp = stereo.compute(img1, img2)
31 cv2.imwrite("cv2Sereo.png", disp)
32
33 depth(img1, img2, 16)

```

Appendix E Python code for Image Stitching

```

1 import cv2
2 import numpy as np
3 import random
4 np.set_printoptions(suppress=True)
5
6 def calcHomo(p1, p2):
7     A = []
8     for i in range(0, len(p1)):
9         x, y = p1[i][0][0], p1[i][0][1]
10        u, v = p2[i][0][0], p2[i][0][1]
11        A.append([x, y, 1, 0, 0, 0, -u*x, -u*y, -u])
12        A.append([0, 0, 0, x, y, 1, -v*x, -v*y, -v])
13    A = np.asarray(A)
14    U, S, Vh = np.linalg.svd(A)
15    L = Vh[-1, :] / Vh[-1, -1]
16    H = L.reshape(3, 3)
17    return H
18
19 def geoDistance(p1, p2, h):
20     p1 = p1[0]
21     p2 = p2[0]
22     p1 = np.append(p1, 1)
23     p2 = np.append(p2, 1)
24     estimatep2 = np.dot(h, p1)
25     estimatep2 = estimatep2/estimatep2[-1]
26     error = p2 - estimatep2
27
28     return np.linalg.norm(error)
29
30
31 def ransac(src, dst, threshold):
32     maxInliers = []
33     finalH = None
34     meme = 0
35     for i in range(1000):
36         meme = i
37         p1 = []
38         p2 = []
39         #First cord:
40         index = random.randrange(0, len(src))
41         p1.append(src[index])

```

```

42     p2.append(dst[index])
43     #Second cord:
44     index = random.randrange(0, len(src))
45     p1.append(src[index])
46     p2.append(dst[index])
47     #Third cord:
48     index = random.randrange(0, len(src))
49     p1.append(src[index])
50     p2.append(dst[index])
51     #Fourth cord:
52     index = random.randrange(0, len(src))
53     p1.append(src[index])
54     p2.append(dst[index])
55
56     h = calcHomo(p1, p2)
57     inlliers = []
58
59     for i in range(len(src)):
60         d = geoDistance(src[i], dst[i], h)
61         if d<5:
62             inlliers.append([src[i], dst[i]])
63
64         if len(inlliers) > len(maxInliers):
65             maxInliers = inlliers
66             finalH = h
67         if len(maxInliers) > (len(src) * threshold):
68             break
69     print("Num iter:",meme)
70     return finalH
71
72
73
74 img1 = cv2.imread("right.jpg")
75 img2 = cv2.imread("left.jpg")
76
77 img1_gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
78 img2_gray = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
79
80 detector = cv2.ORB_create(nfeatures=2000)
81
82 keypoints1, descriptors1 = detector.detectAndCompute(img1, None)
83 keypoints2, descriptors2 = detector.detectAndCompute(img2, None)
84
85 bf = cv2.BFMatcher()
86 matches = bf.knnMatch(descriptors1,descriptors2, k=2)
87
88 # Ratio test
89 good_matches = []
90 for m, n in matches:
91     if m.distance < 0.6 * n.distance:
92         good_matches.append(m)
93
94
95 src_pts = np.float32([keypoints1[m.queryIdx].pt for m in
96                     good_matches]).reshape(-1, 1, 2)
97 dst_pts = np.float32([keypoints2[m.trainIdx].pt for m in
98                     good_matches]).reshape(-1, 1, 2)

```

```
97
98
99 H2, _ = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
100 H = ransac(src_pts, dst_pts, 0.74)
101
102 print(H)
103 print(H2)
104
105 result = cv2.warpPerspective(img1, H,(img1.shape[1] + img2.shape
106 [1], img1.shape[0]))
107 cv2.imshow("Result", result)
108 result2 = cv2.warpPerspective(img1, H2,(img1.shape[1] + img2.shape
109 [1], img1.shape[0]))
110 result2[0:img2.shape[0], 0:img2.shape[1]] = img2
111 cv2.imwrite("output.jpg", result)
112 cv2.waitKey()
```