

ECE 661 - Computer Vision

Arnav Singh (sing1272@purdue.edu)

September 4, 2024

Contents

1	Task 1	2
1.1	Part - A	3
1.2	Part - B	5
1.3	Part - C	6
2	Task 2	8
2.1	Part - A	8
2.2	Part - B	9
2.3	Part - C	10
3	Extra Credit	11
3.1	Part - A	11
3.2	Part - B	12
3.3	Part - C	14
4	Source code	15

1 Task 1

Our task is to project an image of climber Alex Honnold on three different orientations of wall mounted photo frame. The four images used in this task are -



(a) `img1.jpg` Orientation 1 of the photo frame



(b) `img2.jpg` Orientation 2 of the photo frame



(c) `img3.jpg` Orientation 3 of the photo frame



(d) `alex_honnold.jpg` Alex Honnold climbing El Capitan

Figure 1: Images used in this task.

1.1 Part - A

The solution of this task is carried out in 3 steps:

1. Read in Alex's image (**image1**) and an orientation of the photo frame (**image2**). Choose the interest points both images. For **image1**, we will consider the FOUR corners as the interest points. For **image2**, we will consider the FOUR corners of the photo frame as interest points. I picked the four corners of the photo frame using the Python based tool using OpenCV for least amount of manual work.
2. Once four points are selected from both images, we find a unique homography between the two images. The equations to find the unique homography is as follows -

$$\mathbf{x}' = \mathbf{H}\mathbf{x} \implies \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$x' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}}, \quad y' = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}}$$

Rearranging the terms gives,

$$\begin{array}{ll} h_{11}x + h_{12}y + h_{13} & -h_{31}xx' - h_{32}yy' = h_{33}x' \\ h_{21}x + h_{22}y + h_{23} & -h_{31}xy' - h_{32}yy' = h_{33}y' \end{array}$$

Even though we only have 8 equations for the 9 variables involved, we only care about the solution that has $h_{33} = 1$ because they form an equivalence class.

The unknowns $\mathbf{h}$ and the points $\mathbf{x}, \mathbf{x}'$ form the following set of equations -

$$\mathbf{A}\mathbf{h} = \mathbf{b}$$

where,

$$\mathbf{A} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x'_1 & -y_1y'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y'_1 & -y_1y'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x'_2 & -y_2y'_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y'_2 & -y_2y'_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x'_3 & -y_3y'_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y'_3 & -y_3y'_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x'_4 & -y_4y'_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y'_4 & -y_4y'_4 \end{bmatrix} \quad \mathbf{h} = \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \\ x'_4 \\ y'_4 \end{bmatrix}$$

We use NumPy module's inbuilt `linalg.solve` method solve the linear equation above. The Python code for this is as follows -

```
1 def find_homography(p1, p2):
2     # Find H such that p2 = H p1
3
4     assert len(p1) == len(p2) == 4
5
6     A = []
7     b = []
8     for (x, y), (xp, yp) in zip(p1, p2):
9         A.append([x, y, 1, 0, 0, 0, -xp*x, -xp*y])
10        A.append([0, 0, 0, x, y, 1, -yp*x, -yp*y])
11        b.append(xp)
12        b.append(yp)
13
14    A = np.array(A)
15    b = np.array(b)
16    H = np.linalg.solve(A, b)
17    H = np.append(H, 1)
18    H = H.reshape(3, 3)
19
20    return H
```

Listing 1: Python function to find homography between 4 sets of points.

3. To apply this transformation, we iterate over the pixels of `image2`. For each destination coordinate (`dest_coord`) on `image2`, we find the source coordinate (`source_coord`) on `image1`. If `source_coord` lies within bounds then we replace the pixel at `dest_coord` with the pixel at `source_coord`. To save computation time, we find a bounding box on the four corners of the picture frame.

```

1 def apply_homography(image1, image2):
2     # Function to project entire image1 onto selected points of image2
3     # with general homography
4     selected_points = select_points(image2)
5
6     h1, w1 = image1.shape[:2]
7
8     p = [(0, 0), (w1 - 1, 0), (w1 - 1, h1 - 1), (0, h1 - 1)]
9
10    H = find_homography(selected_points, p)
11    warped_image = np.copy(image2)
12
13    min_x = np.min(selected_points[:, 0])
14    min_y = np.min(selected_points[:, 1])
15    max_x = np.max(selected_points[:, 0])
16    max_y = np.max(selected_points[:, 1])
17
18    for y in range(min_y, max_y + 1):
19        for x in range(min_x, max_x + 1):
20            dest_coord = np.array([x, y, 1])
21            source_coord = np.dot(H, dest_coord)
22            source_coord /= source_coord[2]
23
24            # Use nearest neighbours
25            source_x, source_y = int(source_coord[0]), int(source_coord[1])
26            if 0 <= source_x < w1 and 0 <= source_y < h1:
27                warped_image[y, x, :] = image1[source_y, source_x, :]
28
29    return warped_image, selected_points

```

Listing 2: Python function to project `image1` on selected points of `image2`.



(a) Projection onto orientation 1 (b) Projection onto orientation 2 (c) Projection onto orientation 3

Figure 2: General homographic projections of `alex_honnold.jpg` on `img1.jpg`, `img2.jpg`, `img3.jpg`.

The pixel coordinates that were selected in clockwise orientation as the corners of the picture frames are -

- Orientation 1 : (422, 881), (2548, 896), (2405, 2313), (701, 3187)
- Orientation 2 : (535, 1462), (1877, 844), (1953, 2841), (467, 2698)
- Orientation 3 : (1221, 572), (2971, 2321), (1794, 3195), (263, 1808)

1.2 Part - B

We must verify that the product of the homographies from orientations from 1 to 2 and 2 to 3 is equal to the homography from 1 to 3. Following is a Python script to verify the claim -

```
1 if __name__ == '__main__':
2     p1 = np.array([(422, 881), (2548, 896), (2405, 2313), (701, 3187)])
3     p2 = np.array([(535, 1462), (1877, 844), (1953, 2841), (467, 2698)])
4     p3 = np.array([(1221, 572), (2971, 2321), (1794, 3195), (263, 1808)])
5
6     H12 = find_homography(p1, p2)
7     H23 = find_homography(p2, p3)
8     H13 = find_homography(p1, p3)
9
10    H = np.dot(H23, H12)
11    H /= H[2, 2]
12
13    print(H13)
14    print(H)
15    print(np.sum(np.abs(H - H13)))
16
17    img1 = cv.imread("./HW2_images/img1.jpg")
18    img3 = cv.imread("./HW2_images/img3.jpg")
19    warped_image = np.copy(img3)
20    h1, w1 = img1.shape[:2]
21    h3, w3 = img3.shape[:2]
22    H13_inverse = np.linalg.inv(H13)
23    for y in range(h3):
24        for x in range(w3):
25            dest_coord = np.array([x, y, 1])
26            source_coord = np.dot(H13_inverse, dest_coord)
27            source_coord /= source_coord[2]
28            source_x, source_y = int(source_coord[0]), int(source_coord[1])
29            if 0 <= source_x < w1 and 0 <= source_y < h1:
30                warped_image[y, x, :] = img1[source_y, source_x, :]
31            else:
32                warped_image[y, x, :] = np.array([0, 0, 0])
```

Listing 3: Python implementation to verify closure of the homography group.

The output from this code is shown below. As we can see the two homographies are almost equal.

```
[[ 2.38387867e-01 -4.18037010e-01  1.38618046e+03]
 [ 3.45716480e-01  3.89801393e-01  3.46696975e+01]
 [-1.74469883e-04 -1.17251577e-05  1.00000000e+00]]
```

```
[[ 2.38387867e-01 -4.18037010e-01  1.38618046e+03]
 [ 3.45716480e-01  3.89801393e-01  3.46696975e+01]
 [-1.74469883e-04 -1.17251577e-05  1.00000000e+00]]
```

```
1.1881608537485638e-12
```



Figure 3: Homography H13 applied on img1.jpg showing the image is similar to img3.jpg.

1.3 Part - C

The equations related to affine homography estimation are as follows -

$$\mathbf{x}' = \mathbf{H}\mathbf{x} \implies \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$x' = h_{11}x + h_{12}y + h_{13}, \quad y' = h_{21}x + h_{22}y + h_{23}$$

Here, we have 8 equations and 6 variables. So, we will use the least squares fit on a over-determined system to determine the values of the 6 unknowns. They form a system of equations as follows -

$$\mathbf{A}^\top \mathbf{A} \mathbf{h} = \mathbf{A}^\top \mathbf{b}$$

where,

$$\mathbf{A} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ x_3 & y_3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_3 & y_3 & 1 \\ x_4 & y_4 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_4 & y_4 & 1 \end{bmatrix} \quad \mathbf{h} = \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \\ x'_4 \\ y'_4 \end{bmatrix}$$

The Python code for this is as follows -

```
1 def find_affine_four_points(p1, p2):
2     # Function to find the affine homography using least squares fit
3     assert len(p1) == len(p2) == 4
4
5     A = []
6     b = []
7     for (x, y), (xp, yp) in zip(p1, p2):
8         A.append([x, y, 1, 0, 0, 0])
9         A.append([0, 0, 0, x, y, 1])
10        b.append(xp)
11        b.append(yp)
12
13    A = np.array(A)
14    b = np.dot(A.T, np.array(b))
15    S = np.dot(A.T, A)
16    H = np.linalg.solve(S, b)
17    H = H.reshape(2, 3)
18    H = np.vstack((H, [0, 0, 1]))
19
20    return H
```

Listing 4: Finding the affine transform using a Least Squares fit.

A similar algorithm is used to apply the transformation. The Python implementation of it is as follows -

```
1 def apply_affine(image1, image2):
2     # Function to project entire image1 onto selected points of image2
3     # with affine homography
4     selected_points = select_points(image2)
5
6     h1, w1 = image1.shape[:2]
7
8     p = np.array([(0, 0), (w1 - 1, 0), (w1 - 1, h1 - 1), (0, h1 - 1)])
9
10    H = find_affine_four_points(selected_points, p)
11    warped_image = np.copy(image2)
12
13    min_x = np.min(selected_points[:, 0])
14    min_y = np.min(selected_points[:, 1])
15    max_x = np.max(selected_points[:, 0])
16    max_y = np.max(selected_points[:, 1])
17
18    for y in range(min_y, max_y + 1):
19        for x in range(min_x, max_x + 1):
```

```

20     dest_coord = np.array([x, y, 1])
21     source_coord = np.dot(H, dest_coord)
22
23     source_x, source_y = int(source_coord[0]), int(source_coord[1])
24     if 0 <= source_x < w1 and 0 <= source_y < h1:
25         warped_image[y, x, :] = image1[source_y, source_x, :]
26
27     return warped_image, selected_points

```

Listing 5: Python function to project `image1` on selected points of `image2` using an affine homography.

The affine projections are -



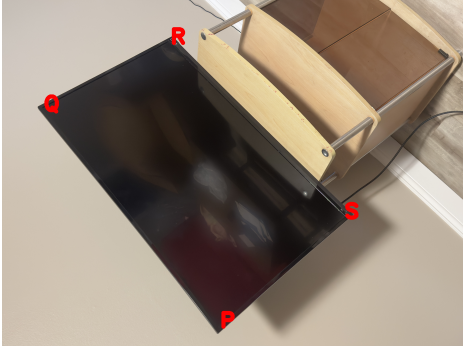
Figure 4: Affine homographic projections of `alex_honnold.jpg` on `img1.jpg`, `img2.jpg`, `img3.jpg`.

Of course these projections onto orientation 1, 2 are not very good because of the obtuse camera angles. The opposite edges of the photo frame which are parallel in the photo frame plane do not remain parallel in the image plane of the camera. This suggests that homographies 1, 2 are not affine and cannot be approximated as affine homographies. However, as we see, the camera is head on with the photo frame and the parallel edges remain almost parallel. This suggests that homography 3 can be approximated using an affine homography and the projection is considered good.

2 Task 2

Perform all three parts from the previous task.

I took pictures of my TV at various angles and chose Wayne Rooney's Bicycle kick at the Manchester Derby as the image I want to project on my TV. The images are -



(a) img4.jpg Orientation 1 of the TV



(b) img5.jpg Orientation 2 of the TV



(c) img6.jpg Orientation 3 of the TV



(d) img4.jpg Orientation 1 of the TV

Figure 5: Images used in this task.

2.1 Part - A

The projected images are -



Figure 6: General homographic projections of rooney.jpg on img4.jpg, img5.jpg, img6.jpg.

The pixel coordinates that were selected in clockwise orientation as the corners of the picture frames are -

- Orientation 1 : (2694, 4045), (523, 1380), (2084, 552), (4226, 2703)
- Orientation 2 : (733, 2399), (2817, 475), (3788, 1694), (1856, 3370)
- Orientation 3 : (361, 1351), (2865, 285), (4055, 2855), (1437, 2865)

2.2 Part - B

The code to check the condition is as follows -

```
1 if __name__ == '__main__':
2     p1 = np.array([(2694, 4045), (523, 1380), (2084, 552), (4226, 2703)])
3     p2 = np.array([(733, 2399), (2817, 475), (3788, 1694), (1856, 3370)])
4     p3 = np.array([(361, 1351), (2865, 285), (4055, 2855), (1437, 2865)])
5
6     H12 = find_homography(p1, p2)
7     H23 = find_homography(p2, p3)
8     H13 = find_homography(p1, p3)
9
10    H = np.dot(H23, H12)
11    H /= H[2, 2]
12
13    print(H13)
14    print(H)
15    print(np.sum(np.abs(H - H13)))
16
17    img1 = cv.imread("./HW2_images/img4.jpg")
18    img3 = cv.imread("./HW2_images/img6.jpg")
19
20    warped_image = np.copy(img3)
21    h1, w1 = img1.shape[:2]
22    h3, w3 = img3.shape[:2]
23
24    H13_inverse = np.linalg.inv(H13)
25    for y in range(h3):
26        for x in range(w3):
27            dest_coord = np.array([x, y, 1])
28            source_coord = np.dot(H13_inverse, dest_coord)
29            source_coord /= source_coord[2]
30
31            source_x, source_y = int(source_coord[0]), int(source_coord[1])
32            if 0 <= source_x < w1 and 0 <= source_y < h1:
33                warped_image[y, x, :] = img1[source_y, source_x, :]
34            else:
35                warped_image[y, x, :] = np.array([0, 0, 0])
```

The output from the code is -

```
[[ 7.13797017e-01 -1.97202602e+00  7.33528858e+03]
 [ 2.81150447e+00 -6.77054592e-01 -3.99717553e+01]
 [ 3.25928327e-04  4.13243076e-04  1.00000000e+00]]
```

```
[[ 7.13797017e-01 -1.97202602e+00  7.33528858e+03]
 [ 2.81150447e+00 -6.77054592e-01 -3.99717553e+01]
 [ 3.25928327e-04  4.13243076e-04  1.00000000e+00]]
```

2.1186388689427293e-12

As we can see the two homographies are almost equal.



Figure 7: Homography H13 applied on img4.jpg showing the image is similar to img6.jpg.

2.3 Part - C

The affine projections are -



Figure 8: Affine homographic projections of `rooney.jpg` on `img4.jpg`, `img5.jpg`, `img6.jpg`.

Again as noticed from the previous task, the orientations 1, 2 of TV are head on and the opposite edges of the TV in the image plane are almost parallel. Thus, the affine projection does well. On the other hand, orientations 3 is captured when the camera is at an obtuse angle. An affine homography cannot be used to project onto this orientation of the camera plane.

3 Extra Credit

3.1 Part - A

1. Parameterize the rotation homography:

We can see that the rotation homography is part of the Euclidean sub-group in the hierarchy of the planar projective transforms. The invariant of this sub-group is distance, as rotation doesn't change the distance. In this case, the rotation is done along the Z -axis, as the image is in the XY -plane. The homography can be written as a function of the angle of rotation α -

$$\mathbf{H}_r(\alpha) = \begin{bmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2. Parameterize the horizontal tilt homography:

If the image is in the XY -plane, the horizontal tilt can be visualized as a rotation along the Y -axis. This visualization helps with the parametrization of the homography with the angle of rotation α and can be written as -

$$\mathbf{H}_h(\alpha) = \begin{bmatrix} \cos \alpha & 0 & \sin \alpha \\ 0 & 1 & 0 \\ -\sin \alpha & 0 & \cos \alpha \end{bmatrix} = \begin{bmatrix} 1 & 0 & \tan \alpha \\ 0 & \sec \alpha & 0 \\ -\tan \alpha & 0 & 1 \end{bmatrix} \quad (\text{if } \cos \alpha \neq 0)$$

3. Parameterize the vertical tilt homography:

If the image is in the XY -plane, the vertical tilt can be visualized as a rotation along the X -axis. Similarly, the parametrization of the homography with the angle of rotation α can be written as -

$$\mathbf{H}_v(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & \sin \alpha \\ 0 & -\sin \alpha & \cos \alpha \end{bmatrix} = \begin{bmatrix} \sec \alpha & 0 & 0 \\ 0 & 1 & \tan \alpha \\ 0 & -\tan \alpha & 1 \end{bmatrix} \quad (\text{if } \cos \alpha \neq 0)$$

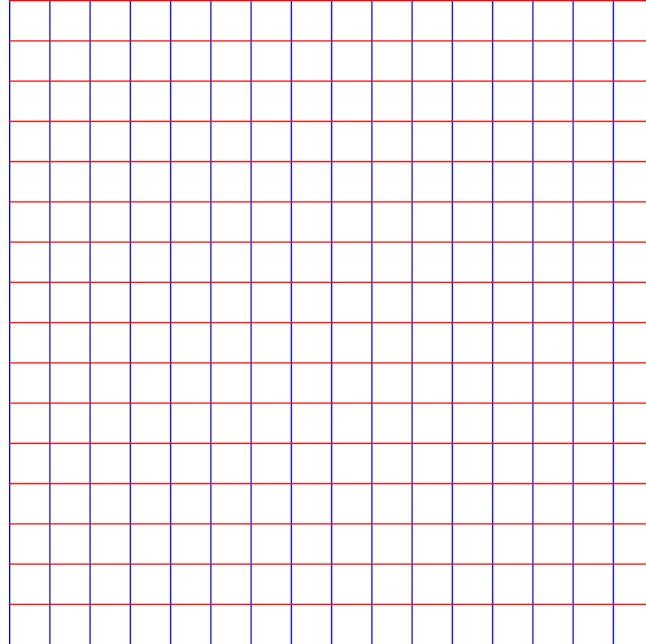


Figure 9: We shall apply these homographies on an image that has horizontal and vertical lines.

3.2 Part - B

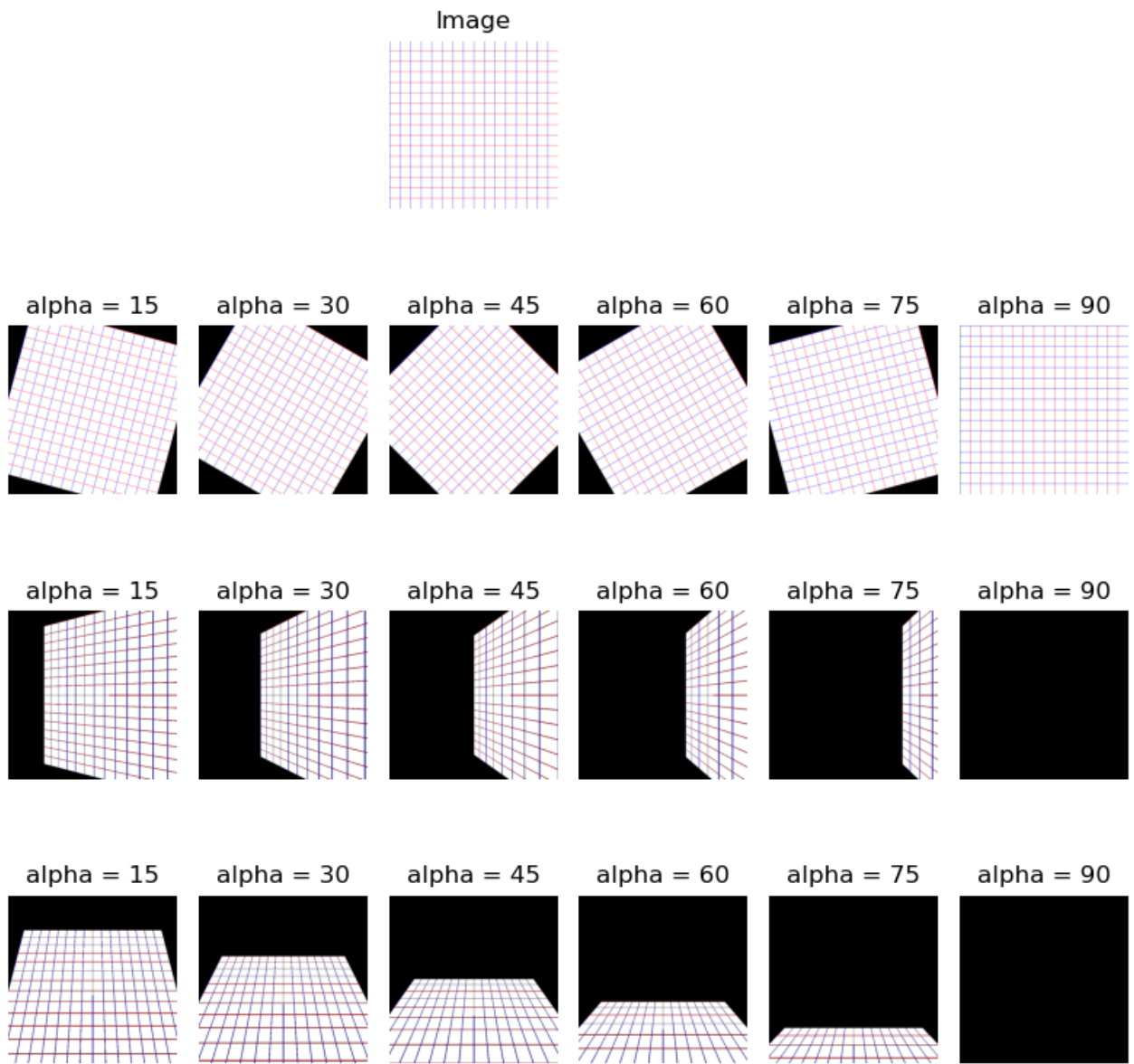


Figure 10: First row shows rotation. Second row shows horizontal tilting. Third shows vertical tilting.


```

1 if __name__ == '__main__':
2     h, w = 512, 512
3
4     H_norm = normalize_coords(h, w)
5     H_denorm = denormalize_coords(h, w)
6
7     alpha_rotate = [15, 30, 45, 60, 75, 90]
8     alpha_htilt = [15, 30, 45, 60, 75, 90]
9     alpha_vtilt = [15, 30, 45, 60, 75, 90]
10
11 # Create an image with horizontal and vertical lines
12 img = 255 * np.ones((h, w, 3))
13 for y in range(0, h):
14     for x in range(0, w):
15         if x % 32 == 0:
16             img[y, x, :] = [0, 0, 255]
17         if y % 32 == 0:
18             img[y, x, :] = [255, 0, 0]
19
20 fig = plt.figure(figsize=(8, 8))
21 ax = plt.subplot2grid((4, len(alpha_rotate)), (0, 2))
22 ax.imshow(img)
23 ax.set_title("Image")
24 ax.axis('off')
25 cv.imwrite(f"./tilt/img.png", cv.cvtColor(img.astype(np.uint8), cv.COLOR_RGB2BGR))
26
27
28 for i in range(len(alpha_rotate)):
29     H_rotate = rotate_homography(np.deg2rad(alpha_rotate[i]))
30     H_rotate = H_denorm @ H_rotate @ H_norm
31     rotate = cv.warpPerspective(img, H_rotate, (w, h))
32     ax = plt.subplot2grid((4, len(alpha_rotate)), (1, i))
33     ax.imshow(rotate)
34     ax.set_title(f"alpha = {alpha_rotate[i]}")
35     ax.axis('off')
36
37     H_htilt = horizontal_tilt_homography(np.deg2rad(alpha_htilt[i]))
38     H_htilt = H_denorm @ H_htilt @ H_norm
39     htilt = cv.warpPerspective(img, H_htilt, (w, h))
40     htilt = htilt.astype(np.uint8)*255
41     ax = plt.subplot2grid((4, len(alpha_rotate)), (2, i))
42     ax.imshow(htilt)
43     ax.set_title(f"alpha = {alpha_htilt[i]}")
44     ax.axis('off')
45     cv.imwrite(f"./tilt/htilt_{i}.png", cv.cvtColor(htilt, cv.COLOR_RGB2BGR))
46
47     H_vtilt = vertical_tilt_homography(np.deg2rad(alpha_vtilt[i]))
48     H_vtilt = H_denorm @ H_vtilt @ H_norm
49     vtilt = cv.warpPerspective(img, H_vtilt, (w, h))
50     vtilt = vtilt.astype(np.uint8)*255
51     ax = plt.subplot2grid((4, len(alpha_rotate)), (3, i))
52     ax.imshow(vtilt)
53     ax.set_title(f"alpha = {alpha_vtilt[i]}")
54     ax.axis('off')
55     cv.imwrite(f"./tilt/vtilt_{i}.png", cv.cvtColor(vtilt, cv.COLOR_RGB2BGR))
56
57 plt.tight_layout()
58 plt.savefig("./HW2_images/rot.png")
59 plt.show()

```

Listing 6: Python code to apply rotation, horizontal and vertical tilt homographies.

3.3 Part - C

1. Invariant under Horizontal tilt:

From the image it is clear that under horizontal tilting, the vertical lines stay vertical. This can be proven using a known relation of how lines transform under planar projective transformations -

$$\mathbf{l}' = \mathbf{H}^{-\top} \mathbf{l}$$

A vertical line $\mathbf{l} \equiv x - k = 0$ in homogeneous coordinates would be represented as $\mathbf{l} = (1, 0, -k)^\top$

For horizontal tilt by an angle α -

$$\begin{aligned} \mathbf{H}_h &= \begin{bmatrix} \cos \alpha & 0 & \sin \alpha \\ 0 & 1 & 0 \\ -\sin \alpha & 0 & \cos \alpha \end{bmatrix} \\ \Rightarrow \mathbf{H}_h^{-1} &= \begin{bmatrix} \cos \alpha & 0 & -\sin \alpha \\ 0 & 1 & 0 \\ \sin \alpha & 0 & \cos \alpha \end{bmatrix} \\ \Rightarrow \mathbf{H}_h^{-\top} &= \begin{bmatrix} \cos \alpha & 0 & \sin \alpha \\ 0 & 1 & 0 \\ -\sin \alpha & 0 & \cos \alpha \end{bmatrix} \end{aligned}$$

Thus, the line in the range plane would be

$$\mathbf{l}' = \mathbf{H}_h^{-\top} \mathbf{l} = \begin{bmatrix} \cos \alpha & 0 & \sin \alpha \\ 0 & 1 & 0 \\ -\sin \alpha & 0 & \cos \alpha \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ -k \end{bmatrix} = \begin{bmatrix} \cos \alpha - k \sin \alpha \\ 0 \\ -\sin \alpha - k \cos \alpha \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ -\frac{\sin \alpha + k \cos \alpha}{\cos \alpha - k \sin \alpha} \end{bmatrix}$$

This shows that the vertical line still remains vertical.

2. Invariant under Vertical tilt:

Again, from the figures we see that the horizontal lines stay horizontal under vertical tilting. The proof follows the same steps as above -

$$\mathbf{l}' = \mathbf{H}^{-\top} \mathbf{l}$$

A vertical line $\mathbf{l} \equiv y - k = 0$ in homogeneous coordinates would be represented as $\mathbf{l} = (0, 1, -k)^\top$

For vertical tilt by an angle α -

$$\begin{aligned} \mathbf{H}_v &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & \sin \alpha \\ 0 & -\sin \alpha & \cos \alpha \end{bmatrix} \\ \Rightarrow \mathbf{H}_v^{-1} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{bmatrix} \\ \Rightarrow \mathbf{H}_v^{-\top} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & \sin \alpha \\ 0 & -\sin \alpha & \cos \alpha \end{bmatrix} \end{aligned}$$

Thus, the line in the range plane would be

$$\mathbf{l}' = \mathbf{H}_v^{-\top} \mathbf{l} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & \sin \alpha \\ 0 & -\sin \alpha & \cos \alpha \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ -k \end{bmatrix} = \begin{bmatrix} 0 \\ \cos \alpha - k \sin \alpha \\ -\sin \alpha - k \cos \alpha \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ -\frac{\sin \alpha + k \cos \alpha}{\cos \alpha - k \sin \alpha} \end{bmatrix}$$

This shows that the horizontal line still remains horizontal.

4 Source code

```
1 import cv2 as cv
2 import numpy as np
3
4
5 class ClickRecorder:
6     # Object to record clicked points
7     def __init__(self):
8         self.clicked_points = []
9
10    def mouse_callback(self, event, x, y, flags, param):
11        if event == cv.EVENT_LBUTTONDOWN:
12            self.clicked_points.append((x, y))
13            print(f"Clicked point: {self.clicked_points[-1]}")
14
15    def get_clicked_points(self):
16        return self.clicked_points
17
18
19 def select_points(img):
20     # Function to select and return selected points on an image
21     click_recorder = ClickRecorder()
22     cv.namedWindow("select_points", cv.WINDOW_NORMAL)
23     cv.resizeWindow("select_points", 600, 600)
24     cv.imshow("select_points", img)
25     cv.setMouseCallback("select_points", click_recorder.mouse_callback)
26     cv.waitKey(0)
27     print(f"All selected points: {click_recorder.get_clicked_points()}")
28     cv.destroyAllWindows()
29
30     return np.array(click_recorder.get_clicked_points())
31
32
33 def find_homography(p1, p2):
34     # Find H such that  $p2 = H p1$ 
35
36     assert len(p1) == len(p2) == 4
37
38     A = []
39     b = []
40     for (x, y), (xp, yp) in zip(p1, p2):
41         A.append([x, y, 1, 0, 0, 0, -xp*x, -xp*y])
42         A.append([0, 0, 0, x, y, 1, -yp*x, -yp*y])
43         b.append(xp)
44         b.append(yp)
45
46     A = np.array(A)
47     b = np.array(b)
48     H = np.linalg.solve(A, b)
49     H = np.append(H, 1)
50     H = H.reshape(3, 3)
51
52     return H
53
54
55 def apply_homography(image1, image2):
56     # Function to project entire image1 onto selected points of image2
57     # with general homography
58     selected_points = select_points(image2)
59
60     h1, w1 = image1.shape[:2]
61
62     p = [(0, 0), (w1 - 1, 0), (w1 - 1, h1 - 1), (0, h1 - 1)]
63
64     H = find_homography(selected_points, p)
65     warped_image = np.copy(image2)
66
67     min_x = np.min(selected_points[:, 0])
68     min_y = np.min(selected_points[:, 1])
69     max_x = np.max(selected_points[:, 0])
70     max_y = np.max(selected_points[:, 1])
71
```

```

72     for y in range(min_y, max_y + 1):
73         for x in range(min_x, max_x + 1):
74             dest_coord = np.array([x, y, 1])
75             source_coord = np.dot(H, dest_coord)
76             source_coord /= source_coord[2]
77
78             # Use nearest neighbours
79             source_x, source_y = int(source_coord[0]), int(source_coord[1])
80             if 0 <= source_x < w1 and 0 <= source_y < h1:
81                 warped_image[y, x, :] = image1[source_y, source_x, :]
82
83     return warped_image, selected_points
84
85 def find_affine_four_points(p1, p2):
86     # Function to find the affine homography using least squares fit
87     assert len(p1) == len(p2) == 4
88
89     A = []
90     b = []
91     for (x, y), (xp, yp) in zip(p1, p2):
92         A.append([x, y, 1, 0, 0, 0])
93         A.append([0, 0, 0, x, y, 1])
94         b.append(xp)
95         b.append(yp)
96
97     A = np.array(A)
98     b = np.dot(A.T, np.array(b))
99     S = np.dot(A.T, A)
100    H = np.linalg.solve(S, b)
101    H = H.reshape(2, 3)
102    H = np.vstack((H, [0, 0, 1]))
103
104    return H
105
106
107 def apply_affine(image1, image2):
108     # Function to project entire image1 onto selected points of image2
109     # with affine homography
110     selected_points = select_points(image2)
111
112     h1, w1 = image1.shape[:2]
113
114     p = np.array([(0, 0), (w1 - 1, 0), (w1 - 1, h1 - 1), (0, h1 - 1)])
115
116     H = find_affine_four_points(selected_points, p)
117     warped_image = np.copy(image2)
118
119     min_x = np.min(selected_points[:, 0])
120     min_y = np.min(selected_points[:, 1])
121     max_x = np.max(selected_points[:, 0])
122     max_y = np.max(selected_points[:, 1])
123
124     for y in range(min_y, max_y + 1):
125         for x in range(min_x, max_x + 1):
126             dest_coord = np.array([x, y, 1])
127             source_coord = np.dot(H, dest_coord)
128
129             source_x, source_y = int(source_coord[0]), int(source_coord[1])
130             if 0 <= source_x < w1 and 0 <= source_y < h1:
131                 warped_image[y, x, :] = image1[source_y, source_x, :]
132
133     return warped_image, selected_points
134
135 # Extra credit
136
137 def normalize_coords(h, w):
138     # Homography to normalize the coordinates
139     H = np.array([[2/w, 0, -1],
140                  [0, 2/h, -1],
141                  [0, 0, 1]])
142
143     return H
144

```

```

145 def denormalize_coords(h, w):
146     # Homography to de-normalize the coordinates
147     H = np.array([[w/2, 0, w/2],
148                   [0, h/2, h/2],
149                   [0, 0, 1]])
150
151     return H
152
153 def rotate_homography(alpha):
154     # Homography to rotate along the z-axis
155     H = np.array([[np.cos(alpha), -np.sin(alpha), 0],
156                   [np.sin(alpha), np.cos(alpha), 0],
157                   [0, 0, 1]])
158
159     return H
160
161 def horizontal_tilt_homography(alpha):
162     # Homography to rotate along the y-axis
163     H = np.array([[np.cos(alpha), 0, np.sin(alpha)],
164                   [0, 1, 0],
165                   [-np.sin(alpha), 0, np.cos(alpha)]])
166
167     return H
168
169 def vertical_tilt_homography(alpha):
170     # Homography to rotate along the x-axis
171     H = np.array([[1, 0, 0],
172                   [0, np.cos(alpha), np.sin(alpha)],
173                   [0, -np.sin(alpha), np.cos(alpha)]])
174
175
176     return H

```

Listing 7: Python code which contains all the functions.

```

1 import cv2 as cv
2 import numpy as np
3 import os
4
5 from homography import apply_homography
6
7 if __name__ == "__main__":
8     img_dir = "./HW2_images/"
9     img_filenames = ["img1.jpg", "img2.jpg", "img3.jpg", "alex_honnold.jpg"]
10    file_path = "points_t1.txt"
11    j = 0
12    for i, img_filename in enumerate(img_filenames[:-1]):
13
14        path = os.path.join(img_dir, img_filename)
15        img2 = cv.imread(path)
16
17        path = os.path.join(img_dir, img_filenames[-1])
18        img1 = np.array(cv.imread(path))
19
20        w, p = apply_homography(img1, img2)
21
22        # Mark the selected points
23        text = ['P', 'Q', 'R', 'S']
24        cv.namedWindow("points", cv.WINDOW_NORMAL)
25        cv.resizeWindow("points", 600, 600)
26        font = cv.FONT_HERSHEY_SIMPLEX
27        fontScale = 8
28        color = (0, 0, 255)
29        thickness = 50
30        img2 = cv.putText(img2, text[0], p[0], font, fontScale, color, thickness, cv.LINE_AA)
31        img2 = cv.putText(img2, text[1], p[1], font, fontScale, color, thickness, cv.LINE_AA)
32        img2 = cv.putText(img2, text[2], p[2], font, fontScale, color, thickness, cv.LINE_AA)
33        img2 = cv.putText(img2, text[3], p[3], font, fontScale, color, thickness, cv.LINE_AA)
34        cv.imshow("points", img2)
35        cv.imwrite(os.path.join(img_dir, f"points_{i}.jpg"), img2)
36
37
38        cv.namedWindow("warped", cv.WINDOW_NORMAL)
39        cv.resizeWindow("warped", 600, 600)

```



```

40     cv.imshow("warped", w)
41     cv.imwrite(os.path.join(img_dir, f"warped_{i}.jpg"), w)
42
43     # Save the points in a .txt file
44     line_number = 3 * j + i
45     with open(file_path, 'r') as f:
46         lines = f.readlines()
47     if line_number >= len(lines):
48         lines.append(f"img{i+1}: {p.flatten()} \n")
49     else:
50         lines[line_number] = f"img{i+1}: {p.flatten()} \n"
51     with open(file_path, 'w') as f:
52         f.writelines(lines)
53
54     cv.waitKey(0)
55
56     cv.destroyAllWindows()

```

Listing 8: Python driver code to run Part - A of Task 1.

```

1  import cv2 as cv
2  import numpy as np
3
4  from homography import find_homography
5
6  if __name__ == '__main__':
7      p1 = np.array([(422, 881), (2548, 896), (2405, 2313), (701, 3187)])
8      p2 = np.array([(535, 1462), (1877, 844), (1953, 2841), (467, 2698)])
9      p3 = np.array([(1221, 572), (2971, 2321), (1794, 3195), (263, 1808)])
10
11     H12 = find_homography(p1, p2)
12     H23 = find_homography(p2, p3)
13     H13 = find_homography(p1, p3)
14
15     H = np.dot(H23, H12)
16     H /= H[2, 2]
17
18     print(H13)
19     print(H)
20     print(np.sum(np.abs(H - H13)))
21
22     img1 = cv.imread("./HW2_images/img1.jpg")
23     img3 = cv.imread("./HW2_images/img3.jpg")
24
25     cv.namedWindow("warped", cv.WINDOW_NORMAL)
26     cv.resizeWindow("warped", 600, 600)
27     cv.imshow("warped", img1)
28     cv.waitKey(0)
29
30     warped_image = np.copy(img3)
31     h1, w1 = img1.shape[:2]
32     h3, w3 = img3.shape[:2]
33
34     H13_inverse = np.linalg.inv(H13)
35     for y in range(h3):
36         for x in range(w3):
37             dest_coord = np.array([x, y, 1])
38             source_coord = np.dot(H13_inverse, dest_coord)
39             source_coord /= source_coord[2]
40
41             source_x, source_y = int(source_coord[0]), int(source_coord[1])
42             if 0 <= source_x < w1 and 0 <= source_y < h1:
43                 warped_image[y, x, :] = img1[source_y, source_x, :]
44             else:
45                 warped_image[y, x, :] = np.array([0, 0, 0])
46
47     cv.namedWindow("warped", cv.WINDOW_NORMAL)
48     cv.resizeWindow("warped", 600, 600)
49     cv.imshow("warped", warped_image)
50     cv.waitKey(0)
51     cv.destroyWindow("warped")

```

Listing 9: Python driver code to run Part - B of Task 1.

```

1 import cv2 as cv
2 import numpy as np
3 import os
4
5 from homography import apply_affine
6
7 if __name__ == '__main__':
8     img_dir = "./HW2_images/"
9     img_filenames = ["img1.jpg", "img2.jpg", "img3.jpg", "alex_honnold.jpg"]
10    file_path = "points_t1.txt"
11    j = 1
12    for i, img_filename in enumerate(img_filenames[:-1]):
13
14        path = os.path.join(img_dir, img_filename)
15        img2 = cv.imread(path)
16
17        path = os.path.join(img_dir, img_filenames[-1])
18        img1 = np.array(cv.imread(path))
19
20        w, p = apply_affine(img1, img2)
21        cv.namedWindow("affine", cv.WINDOW_NORMAL)
22        cv.resizeWindow("affine", 600, 600)
23        cv.imshow("affine", w)
24        cv.imwrite(os.path.join(img_dir, f"affine_{i}.jpg"), w)
25
26        line_number = 3 * j + i
27        with open(file_path, 'r') as f:
28            lines = f.readlines()
29            if line_number >= len(lines):
30                lines.append(f"img{i+1}: {p.flatten()} \n")
31            else:
32                lines[line_number] = f"img{i+1}: {p.flatten()} \n"
33        with open(file_path, 'w') as f:
34            f.writelines(lines)
35
36        cv.waitKey(0)
37
38    cv.destroyAllWindows()

```

Listing 10: Python driver code to run Part - C of Task 1.

```

1 import cv2 as cv
2 import numpy as np
3 import os
4
5 from homography import apply_homography
6
7 if __name__ == '__main__':
8     img_dir = "./HW2_images/"
9     img_filenames = ["img4.jpg", "img5.jpg", "img6.jpg", "rooney.jpg"]
10    file_path = "points_t2.txt"
11    j = 0
12    for i, img_filename in enumerate(img_filenames[:-1]):
13
14        path = os.path.join(img_dir, img_filename)
15        img2 = cv.imread(path)
16
17        path = os.path.join(img_dir, img_filenames[-1])
18        img1 = np.array(cv.imread(path))
19
20        w, p = apply_homography(img1, img2)
21
22        text = ['P', 'Q', 'R', 'S']
23        cv.namedWindow("points", cv.WINDOW_NORMAL)
24        cv.resizeWindow("points", 600, 600)
25        font = cv.FONT_HERSHEY_SIMPLEX
26        fontScale = 8
27        color = (0, 0, 255)
28        thickness = 50
29        img2 = cv.putText(img2, text[0], p[0], font, fontScale, color, thickness, cv.LINE_AA)
30        img2 = cv.putText(img2, text[1], p[1], font, fontScale, color, thickness, cv.LINE_AA)
31        img2 = cv.putText(img2, text[2], p[2], font, fontScale, color, thickness, cv.LINE_AA)
32        img2 = cv.putText(img2, text[3], p[3], font, fontScale, color, thickness, cv.LINE_AA)
33        cv.imshow("points", img2)

```

```

34     cv.imwrite(os.path.join(img_dir, f"points_{i+3}.jpg"), img2)
35     cv.namedWindow("warped", cv.WINDOW_NORMAL)
36     cv.resizeWindow("warped", 600, 600)
37     cv.imshow("warped", w)
38     cv.imwrite(os.path.join(img_dir, f"warped_{i+3}.jpg"), w)
39
40     line_number = 3 * j + i
41     with open(file_path, 'r') as f:
42         lines = f.readlines()
43     if line_number >= len(lines):
44         lines.append(f"img{i+1}: {p.flatten()} \n")
45     else:
46         lines[line_number] = f"img{i+1}: {p.flatten()} \n"
47     with open(file_path, 'w') as f:
48         f.writelines(lines)
49
50     cv.waitKey(0)
51
52     cv.destroyAllWindows()

```

Listing 11: Python driver code to run Part - A of Task 2.

```

1  import cv2 as cv
2  import numpy as np
3  import os
4
5  from homography import find_homography, select_points
6  if __name__ == '__main__':
7      p1 = np.array([(2694, 4045), (523, 1380), (2084, 552), (4226, 2703)])
8      p2 = np.array([(733, 2399), (2817, 475), (3788, 1694), (1856, 3370)])
9      p3 = np.array([(361, 1351), (2865, 285), (4055, 2855), (1437, 2865)])
10
11     H12 = find_homography(p1, p2)
12     H23 = find_homography(p2, p3)
13     H13 = find_homography(p1, p3)
14
15     H = np.dot(H23, H12)
16     H /= H[2, 2]
17
18     print(H13)
19     print(H)
20     print(np.sum(np.abs(H - H13)))
21
22     img1 = cv.imread("./HW2_images/img4.jpg")
23     img3 = cv.imread("./HW2_images/img6.jpg")
24
25     cv.namedWindow("warped", cv.WINDOW_NORMAL)
26     cv.resizeWindow("warped", 600, 600)
27     cv.imshow("warped", img1)
28     cv.waitKey(0)
29
30     warped_image = np.copy(img3)
31     h1, w1 = img1.shape[:2]
32     h3, w3 = img3.shape[:2]
33
34     H13_inverse = np.linalg.inv(H13)
35     for y in range(h3):
36         for x in range(w3):
37             dest_coord = np.array([x, y, 1])
38             source_coord = np.dot(H13_inverse, dest_coord)
39             source_coord /= source_coord[2]
40
41             source_x, source_y = int(source_coord[0]), int(source_coord[1])
42             if 0 <= source_x < w1 and 0 <= source_y < h1:
43                 warped_image[y, x, :] = img1[source_y, source_x, :]
44             else:
45                 warped_image[y, x, :] = np.array([0, 0, 0])
46
47     cv.namedWindow("warped", cv.WINDOW_NORMAL)
48     cv.resizeWindow("warped", 600, 600)
49     cv.imshow("warped", warped_image)
50     cv.waitKey(0)

```

```
51 cv.destroyAllWindows("warped")
```

Listing 12: Python driver code to run Part - B of Task 2.

```
1 import cv2 as cv
2 import numpy as np
3 import os
4
5 from homography import apply_affine
6
7 if __name__ == '__main__':
8     img_dir = "./HW2_images/"
9     img_filenames = ["img4.jpg", "img5.jpg", "img6.jpg", "rooney.jpg"]
10    file_path = "points_t2.txt"
11    j = 1
12    for i, img_filename in enumerate(img_filenames[:-1]):
13
14        path = os.path.join(img_dir, img_filename)
15        img2 = cv.imread(path)
16
17        path = os.path.join(img_dir, img_filenames[-1])
18        img1 = np.array(cv.imread(path))
19
20        w, p = apply_affine(img1, img2)
21        cv.namedWindow("affine", cv.WINDOW_NORMAL)
22        cv.resizeWindow("affine", 600, 600)
23        cv.imshow("affine", w)
24        cv.imwrite(os.path.join(img_dir, f"affine_{i+3}.jpg"), w)
25
26        line_number = 3 * j + i
27        with open(file_path, 'r') as f:
28            lines = f.readlines()
29            if line_number >= len(lines):
30                lines.append(f"img{i+1}: {p.flatten()} \n")
31            else:
32                lines[line_number] = f"img{i+1}: {p.flatten()} \n"
33        with open(file_path, 'w') as f:
34            f.writelines(lines)
35
36        cv.waitKey(0)
37
38    cv.destroyAllWindows()
```

Listing 13: Python driver code to run Part - C of Task 2.

```
1 import cv2 as cv
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 from homography import normalize_coords, denormalize_coords, rotate_homography, \
6     horizontal_tilt_homography, vertical_tilt_homography
7
8 if __name__ == '__main__':
9     h, w = 512, 512
10
11     H_norm = normalize_coords(h, w)
12     H_denorm = denormalize_coords(h, w)
13
14     alpha_rotate = [15, 30, 45, 60, 75, 90]
15     alpha_htilt = [15, 30, 45, 60, 75, 90]
16     alpha_vtilt = [15, 30, 45, 60, 75, 90]
17
18     # Create an image with horizontal and vertical lines
19     img = 255 * np.ones((h, w, 3))
20     for y in range(0, h):
21         for x in range(0, w):
22             if x % 32 == 0:
23                 img[y, x, :] = [0, 0, 255]
24             if y % 32 == 0:
25                 img[y, x, :] = [255, 0, 0]
26
27     fig = plt.figure(figsize=(8, 8))
28     ax = plt.subplot2grid((4, len(alpha_rotate)), (0, 2))
```

```

29 ax.imshow(img)
30 ax.set_title("Image")
31 ax.axis('off')
32 cv.imwrite(f"./tilt/img.png", cv.cvtColor(img.astype(np.uint8), cv.COLOR_RGB2BGR))
33
34
35 for i in range(len(alpha_rotate)):
36     H_rotate = rotate_homography(np.deg2rad(alpha_rotate[i]))
37     H_rotate = H_denorm @ H_rotate @ H_norm
38     rotate = cv.warpPerspective(img, H_rotate, (w, h))
39     ax = plt.subplot2grid((4, len(alpha_rotate)), (1, i))
40     ax.imshow(rotate)
41     ax.set_title(f"alpha = {alpha_rotate[i]}")
42     ax.axis('off')
43
44     H_htilt = horizontal_tilt_homography(np.deg2rad(alpha_htilt[i]))
45     H_htilt = H_denorm @ H_htilt @ H_norm
46     htilt = cv.warpPerspective(img, H_htilt, (w, h))
47     htilt = htilt.astype(np.uint8)*255
48     ax = plt.subplot2grid((4, len(alpha_rotate)), (2, i))
49     ax.imshow(htilt)
50     ax.set_title(f"alpha = {alpha_htilt[i]}")
51     ax.axis('off')
52     cv.imwrite(f"./tilt/htilt_{i}.png", cv.cvtColor(htilt, cv.COLOR_RGB2BGR))
53
54     H_vtilt = vertical_tilt_homography(np.deg2rad(alpha_vtilt[i]))
55     H_vtilt = H_denorm @ H_vtilt @ H_norm
56     vtilt = cv.warpPerspective(img, H_vtilt, (w, h))
57     vtilt = vtilt.astype(np.uint8)*255
58     ax = plt.subplot2grid((4, len(alpha_rotate)), (3, i))
59     ax.imshow(vtilt)
60     ax.set_title(f"alpha = {alpha_vtilt[i]}")
61     ax.axis('off')
62     cv.imwrite(f"./tilt/vtilt_{i}.png", cv.cvtColor(vtilt, cv.COLOR_RGB2BGR))
63
64 plt.tight_layout()
65 plt.savefig("./HW2_images/rot.png")
66 plt.show()

```

Listing 14: Python driver code to run extra credit assignment.