

ECE 661 Computer Vision: HW2

Qiuchen Zhai
qzhai@purdue.edu

September 22, 2020

1 Logic and Methodology

1.1 Point-to-point correspondences

The linear mapping on homogeneous 3-vectors is given by

$$\mathbf{x}' = \mathbf{H}\mathbf{x} \quad (1)$$

where $\mathbf{x} \in \mathbb{R}^3$ is the domain point, $\mathbf{x}' \in \mathbb{R}^3$ is the corresponding range point, and \mathbf{H} is the HC matrix. Thus we have

$$\begin{pmatrix} x'_1 \\ x'_2 \\ x'_3 \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} h_{11}x_1 + h_{12}x_2 + h_{13}x_3 \\ h_{21}x_1 + h_{22}x_2 + h_{23}x_3 \\ h_{31}x_1 + h_{32}x_2 + x_3 \end{pmatrix} \quad (2)$$

Let $x = \frac{x_1}{x_3}$, $y = \frac{x_2}{x_3}$, $x' = \frac{x'_1}{x'_3}$, and $y' = \frac{x'_2}{x'_3}$. Then we get the following equations

$$x' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + 1} \quad (3)$$

$$y' = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + 1} \quad (4)$$

Therefore, the HC matrix could be solved by

$$\mathbf{h} = \mathbf{A}^{-1}\mathbf{b} \quad (5)$$

where $\mathbf{A} = \begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x'_1 & -y_1x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y'_1 & -y_1y'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x'_2 & -y_2x'_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y'_2 & -y_2y'_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x'_3 & -y_3x'_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y'_3 & -y_3y'_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x'_4 & -y_4x'_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y'_4 & -y_4y'_4 \end{pmatrix}$, $\mathbf{b} = \begin{pmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \\ x'_4 \\ y'_4 \end{pmatrix}$, and $\mathbf{h} = \begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{pmatrix}$.

One important fundamental invariant of the Projective Linear group $PL(3)$ is that straight lines going to straight lines. Once the HC matrix is obtained, the image distortion could be removed by point-to-point correspondences.

1.2 Two-step method

1.2.1 Remove the projective distortion

It's been proved that, a homography \mathbf{H} is affine if and only if the *line at infinity* \mathbf{l}_∞ is mapped to \mathbf{l}_∞ . While the *vanishing line* the *line at infinity* mapped to a physical line through a general projective transform mapping, we could utilize the *vanishing line* to get rid of projective distortion. First, we need to choose two pairs of parallel lines in the physical world and compute their intersection points in the image with distortion. Assume the pixel coordinates of the corners of a rectangular frame formed by the two pairs of lines are p, q, s, r , the two pairs of lines and their intersection points are given by

$$l_{pq} = p \times q, l_{rs} = r \times s \Rightarrow x_1 = l_{pq} \times l_{rs} \quad (6)$$

$$l_{qs} = q \times s, l_{pr} = p \times r \Rightarrow x_2 = l_{qs} \times l_{pr} \quad (7)$$

Then, the vanish line could be obtained by computing the cross product of the two intersection points.

$$\mathbf{l}_{VL} = x_1 \times x_2 \quad (8)$$

After we get the vanishing line, the homography that sends $\mathbf{l}_{VL} = \begin{pmatrix} l_1 \\ l_2 \\ l_3 \end{pmatrix}$ back to $\mathbf{l}_\infty = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$ is

$$H = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & l_3 \end{pmatrix} \quad (9)$$

1.2.2 Remove the affine distortion

Since the affine distortion turns a 90° angle into some value θ , our correction should take θ back to the 90° angle. Assume θ is the angle between the lines $\mathbf{l} = \begin{pmatrix} l_1 \\ l_2 \\ l_3 \end{pmatrix}$ and $\mathbf{m} = \begin{pmatrix} m_1 \\ m_2 \\ m_3 \end{pmatrix}$ in an image with affine distortion. Then we have

$$\cos\theta = \frac{l_1 m_1 + l_2 m_2}{\sqrt{(l_1^2 + l_2^2)(m_1^2 + m_2^2)}} \quad (10)$$

Since $C_\infty^* = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$, we have

$$\cos\theta = \frac{\mathbf{l}^T C_\infty^* \mathbf{m}}{\sqrt{(\mathbf{l}^T C_\infty^* \mathbf{l})(\mathbf{m}^T C_\infty^* \mathbf{m})}} \quad (11)$$

Now assume the angle θ is 90° in the original scene, we have the following constraint for estimating \mathbf{H}

$$\mathbf{l}'^T \mathbf{H} C_\infty^* \mathbf{H}^T \mathbf{l}' = \mathbf{0} \quad (12)$$

where

$$\mathbf{l}'^T \mathbf{H} C_\infty^* \mathbf{H}^T \mathbf{l}' = (l'_1 \ l'_2 \ l'_3) \begin{pmatrix} \mathbf{A} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{A}^T & \mathbf{0} \\ \mathbf{t}^T & 1 \end{pmatrix} \begin{pmatrix} m'_1 \\ m'_2 \\ m'_3 \end{pmatrix} = (l'_1 \ l'_2 \ l'_3) \begin{pmatrix} \mathbf{A} \mathbf{A}^T & \mathbf{0} \\ \mathbf{t}^T & 0 \end{pmatrix} \begin{pmatrix} m'_1 \\ m'_2 \\ m'_3 \end{pmatrix} \quad (13)$$

The above equation yields the following constraint

$$\begin{pmatrix} l'_1 & l'_2 \end{pmatrix} \mathbf{A} \mathbf{A}^T \begin{pmatrix} m'_1 \\ m'_2 \end{pmatrix} = \mathbf{0} \quad (14)$$

Let $\mathbf{S} = \mathbf{A} \mathbf{A}^T = \begin{pmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{pmatrix}$, then

$$l'_1 m'_1 s_{11} + (l'_1 m'_2 + l'_2 m'_1) s_{12} + l'_2 m'_2 s_{22} = 0 \quad (15)$$

There're three unknowns in the equation. Therefore, at least two pairs of orthogonal lines are needed to solve the \mathbf{S} . Assume \mathbf{A} is positive definite , then we could use the singular value decomposition (SVD)

$$\mathbf{S} = \mathbf{A} \mathbf{A}^T = \mathbf{V} \mathbf{D}^2 \mathbf{V}^T \quad (16)$$

Then

$$\mathbf{A} = \mathbf{V} \mathbf{D} \mathbf{V}^T \quad (17)$$

And the corresponding HC matrix is

$$\mathbf{H} = \begin{pmatrix} \mathbf{A} & 0 \\ 0 & 1 \end{pmatrix} \quad (18)$$

1.3 One-step method

In one-step method to remove both distortions, the projection of the \mathbf{C}_∞^* is

$$\mathbf{C}_\infty^{*''} = \begin{pmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{pmatrix} \quad (19)$$

In HC matrix, we could set $f = 1$ since only the ratios matter. Therefore, we need to identify five pairs of orthogonal lines in world coordinates to estimate the five unknowns $a, b, c, d, \text{ and } e$ by solving

$$\mathbf{l}'^T \mathbf{C}_\infty^{*''} \mathbf{m}' = 0 \quad (20)$$

For each pair of orthogonal lines $\mathbf{l} = \begin{pmatrix} l'_1 \\ l'_2 \\ l'_3 \end{pmatrix}$ and $\mathbf{m} = \begin{pmatrix} m'_1 \\ m'_2 \\ m'_3 \end{pmatrix}$, we have

$$l'_1 m'_1 a + \frac{1}{2}(l'_1 m'_2 + l'_2 m'_1) b + l'_2 m'_2 c + \frac{1}{2}(l'_1 m'_3 + l'_3 m'_1) d + \frac{1}{2}(l'_2 m'_3 + l'_3 m'_2) e = -l'_3 m'_3 \quad (21)$$

After estimating $\mathbf{C}_\infty^{*'}$, the homography \mathbf{H} that corrects both projective and affine distortion could be obtained by singular value decomposition of $\mathbf{C}_\infty^{*'}$

$$\mathbf{C}_\infty^{*''} = \mathbf{H} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \mathbf{H}^T \quad (22)$$

2 Results

2.1 Task1

2.1.1 Method 1: Point-to-point Correspondences

1. Point-to-point correspondences applied to the first image:



Figure 1: original Img1



Figure 2: measurements of Img1

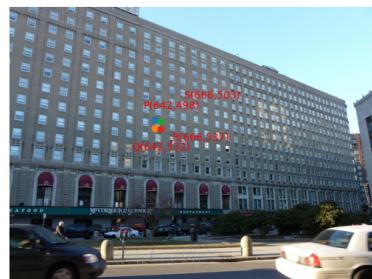


Figure 3: labelled points in Img1



Figure 4: Img1 after ptp mapping

2. Point-to-point correspondences applied to the second image:



Figure 5: original Img2



Figure 6: measurements of Img2



Figure 7: labelled points in Img2

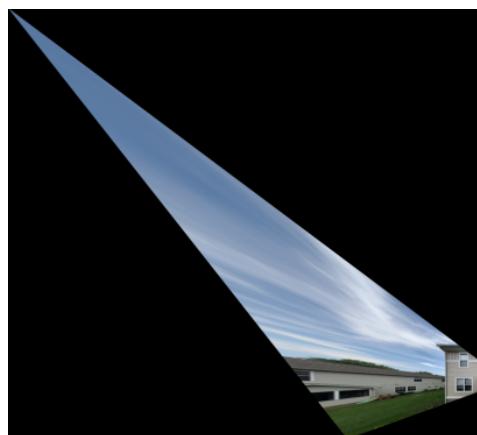


Figure 8: Img2 after ptp mapping

3. Point-to-point correspondences applied to the third image:

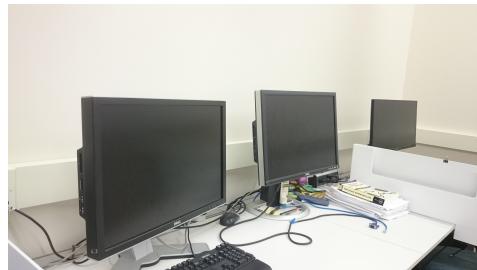


Figure 9: original Img3



Figure 10: measurements of Img3



Figure 11: labelled points in Img3

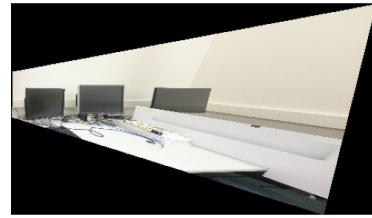


Figure 12: Img3 after ptp mapping

2.1.2 Method 2: Two-step Method

1. Two-step method applied to the first image:



Figure 13: original Img1

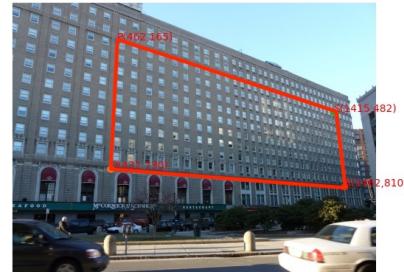


Figure 14: lines used for Img1



Figure 15: after removing projective distortion



Figure 16: after removing affine distortion

2. Two-step method Correspondences applied to the second image:



Figure 17: original Img2



Figure 18: lines used for Img2

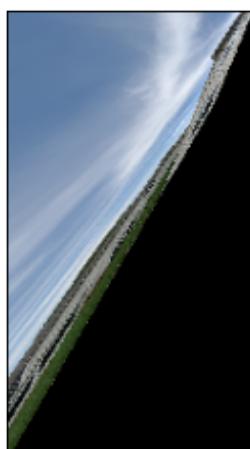


Figure 19: after removing projective distortion



Figure 20: after removing affine distortion

3. Two-step method applied to the third image:

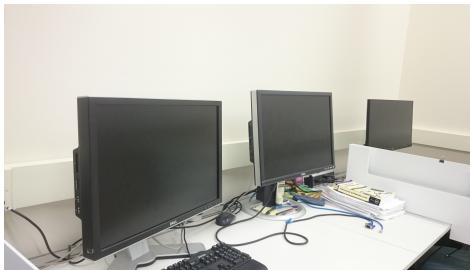


Figure 21: original Img3

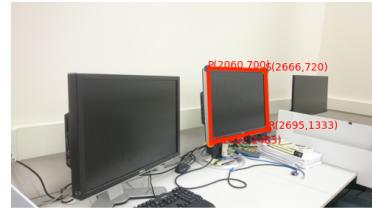


Figure 22: lines used for Img3



Figure 23: after removing projective distortion



Figure 24: after removing affine distortion



Figure 25: after removing affine distortion (Enlarged version of Figure 24)

2.1.3 Method 3: One-step Method

1. One-step method applied to the first image:



Figure 26: lines used for Img1

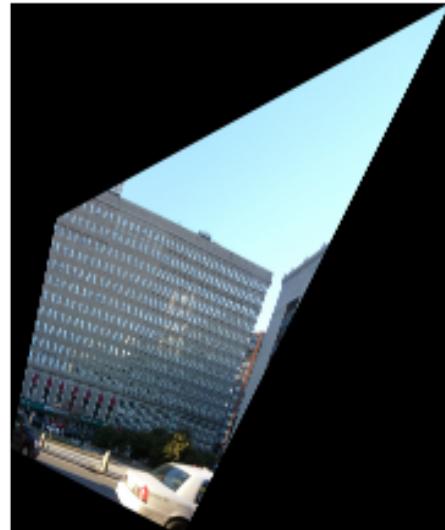


Figure 27: result of Img1

2. One-step method Correspondences applied to the second image:



Figure 28: lines used for Img2

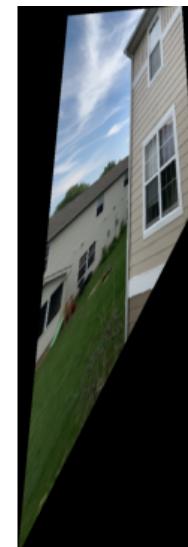


Figure 29: result of Img2

3. One-step method applied to the third image:



Figure 30: lines used for Img3



Figure 31: result of Img3

2.2 Task2

2.2.1 Method 1: Point-to-point Correspondences

1. Point-to-point correspondences applied to the fourth image:



Figure 32: original Img4



Figure 33: measurements of Img4



Figure 34: labelled points in Img4



Figure 35: Img4 after ptp mapping

2. Point-to-point correspondences applied to the fifth image:



Figure 36: original Img5



Figure 37: measurements of Img5



Figure 38: labelled points in Img5



Figure 39: Img5 after ptp mapping

2.2.2 Method2: Two-step Method

1. Two-step method applied to the fourth image:



Figure 40: original Img4



Figure 41: lines used for Img4



Figure 42: after removing projective distortion



Figure 43: after removing affine distortion

2. Two-step method applied to the fifth image:



Figure 44: original Img5



Figure 45: lines used for Img5

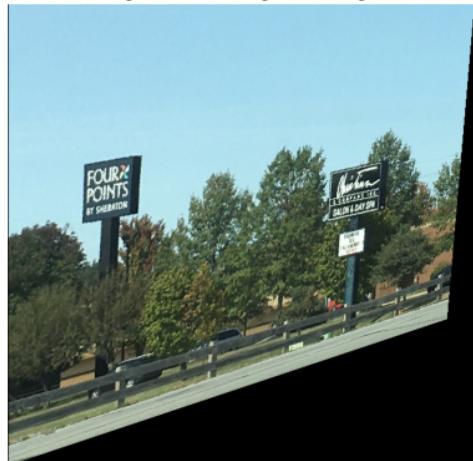


Figure 46: after removing projective dis-tortion



Figure 47: after removing affine distor-tion

2.2.3 Method 3: One-step Method

1. One-step method applied to the fourth image:



Figure 48: lines used for Img4



Figure 49: result of Img4

2. One-step method Correspondences applied to the fifth image:



Figure 50: lines used for Img5

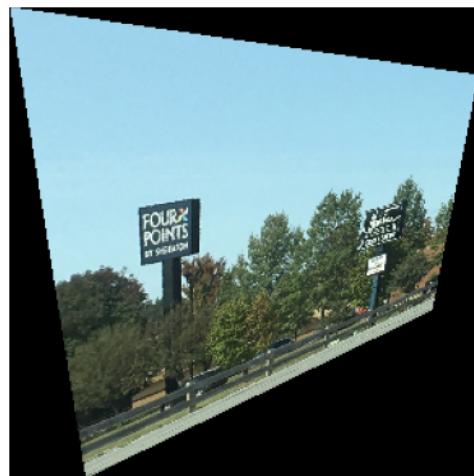


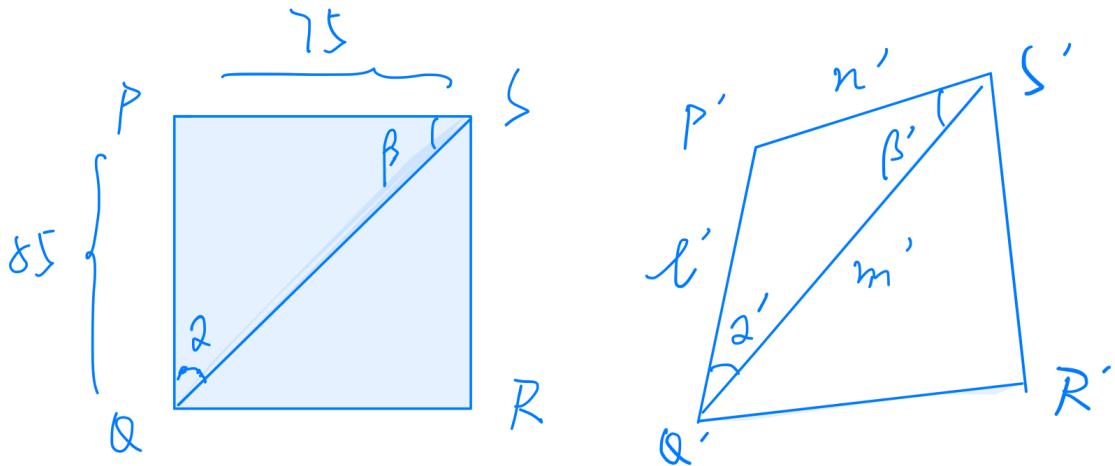
Figure 51: result of Img5

2.3 Observations

- Both the point-to-point correspondences and one-step method are sensitive to the initial points. Thus we need to be careful while choosing the points and lines.
- The point-to-point correspondences method is accurate though sometimes we might not be able to retrieve the physical measurements in the world plane.
- The two-step method is robust and we can see the process of getting rid of projective distortion and affine distortion.
- The results from two-step method could still include distortions.
- The two-step method is fast.
- The one-step method is sensitive to the orthogonal lines. However, we could pick up more than five pairs of orthogonal lines in the image to facilitate the result.

3 Extra credits

The first image is utilized to demonstrate the length ratios. First, consider the window in the real-world shown in left with physical measurements. We could easily compute the angle values $\alpha = 41.423665625002656^\circ$ and $\beta = 48.57633437499736^\circ$.



Real-world window

Distorted window

Figure 52: geometry between two coordinates

To compute the length ratios in the original scene using $\mathbf{C}_\infty^{* \prime}$, we need to find the corresponding points in the distorted image.



Figure 53: Point indication for image used

Then we estimate the angles by

$$\cos \alpha = \frac{\mathbf{l}'^T C_{\infty}^* \mathbf{m}'}{\sqrt{(\mathbf{l}'^T C_{\infty}^* \mathbf{l}')(\mathbf{m}'^T C_{\infty}^* \mathbf{m}')}} \quad (23)$$

$$\cos \beta = \frac{\mathbf{n}'^T C_{\infty}^* \mathbf{m}'}{\sqrt{(\mathbf{n}'^T C_{\infty}^* \mathbf{n}')(\mathbf{m}'^T C_{\infty}^* \mathbf{m}')}} \quad (24)$$

where $C_{\infty}^* = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$, $\mathbf{l}' = \mathbf{P}' \times \mathbf{Q}'$, $\mathbf{m}' = \mathbf{S}' \times \mathbf{Q}'$, $\mathbf{n}' = \mathbf{P}' \times \mathbf{S}'$.

Thus we get $\alpha_{est} = 41.51984067302296^\circ$ and $\beta_{est} = 52.679921802611936^\circ$. And the length ratio is then given by $\sin \alpha_{est} : \sin \beta_{est} \approx 0.8335367861149746$, which is close to the real length ratio 0.8823529411764706.

4 Source Code

4.1 method1.py

```
1000 import numpy as np
1001 import cv2
1002 import matplotlib.pyplot as plt
1003 from skimage import io
1004
1005
1006 def compute_hc(src, dst):
1007     """
1008     The module computes the homography matrix by  $AH = b$ .
1009     :param src: source points.
1010     :param dst: mapped points.
1011     :return: homography matrix H.
1012     """
1013
1014     A = np.zeros((8, 8))
1015     b = np.zeros((8, 1))
1016
1017     for i in range(len(src)):
1018         A[2 * i] = [src[i][0], src[i][1], 1, 0, 0, 0, -src[i][0]*dst[i][0], -src[i][1]*dst[i][0]]
1019         A[2 * i + 1] = [0, 0, 0, src[i][0], src[i][1], 1, -src[i][0]*dst[i][1], -src[i][1]*dst[i][1]]
1020         b[2 * i] = dst[i][0]
1021         b[2 * i + 1] = dst[i][1]
1022
1023     h = np.dot(np.linalg.pinv(A), b)
1024     homo_mat = np.append(h, 1)
1025
1026     return homo_mat.reshape((3, 3))
1027
1028
1029 def compute_pixel_val(img, loc):
1030     """
1031     This module returns the pixel value given by weighting average of neighbor pixels.
1032     :param img: the mapping img.
1033     :param loc: the coordinates of mapped points.
1034     :return: the pixel value.
1035     """
1036
1037     loc0_f = np.int(np.floor(loc[0]))
1038     loc1_f = np.int(np.floor(loc[1]))
1039     loc0_c = np.int(np.ceil(loc[0]))
1040     loc1_c = np.int(np.ceil(loc[1]))
1041     a = img[loc0_f][loc1_f]
1042     b = img[loc0_f][loc1_c]
1043     c = img[loc0_c][loc1_f]
1044     d = img[loc0_c][loc1_c]
1045     dx = float(loc[0] - loc0_f)
1046     dy = float(loc[1] - loc1_f)
1047     Wa = 1 / np.linalg.norm([dx, dy])
1048     Wb = 1 / np.linalg.norm([1 - dx, dy])
1049     Wc = 1 / np.linalg.norm([dx, 1 - dy])
1050     Wd = 1 / np.linalg.norm([1 - dx, 1 - dy])
1051     output = (a * Wa + b * Wb + c * Wc + d * Wd) / (Wa + Wb + Wc + Wd)
1052
1053     return output
1054
1055
1056 def ptp_mapping(img, h_mat):
1057     # Determine the coordinates of domain plane
```

```

1058     m, n, d = np.shape(img)
1059     P = np.array([0, 0, 1])
1060     Q = np.array([0, m-1, 1])
1061     S = np.array([n-1, 0, 1])
1062     R = np.array([n-1, m-1, 1])
1063     # Compute the projection of corners
1064     locP = np.dot(h_mat, P) / np.dot(h_mat, P)[2]
1065     locQ = np.dot(h_mat, Q) / np.dot(h_mat, Q)[2]
1066     locS = np.dot(h_mat, S) / np.dot(h_mat, S)[2]
1067     locR = np.dot(h_mat, R) / np.dot(h_mat, R)[2]
1068     # Determine the size of mapped image
1069     x_min = np.int(np.floor(np.min([locP[1], locQ[1], locR[1], locS[1]])))
1070     x_max = np.int(np.ceil(np.max([locP[1], locQ[1], locR[1], locS[1]])))
1071     x_scale = x_max - x_min
1072     y_min = np.int(np.floor(np.min([locP[0], locQ[0], locR[0], locS[0]])))
1073     y_max = np.int(np.ceil(np.max([locP[0], locQ[0], locR[0], locS[0]])))
1074     y_scale = y_max - y_min
1075
1076     # Compute scaling factor
1077     # if we fix width, then the scaling factor s_w = w_o / w_i
1078     scaling_w = y_scale / n
1079     # if we fix length, then the scaling factor s_h = h_o / h_i
1080     scaling_h = x_scale / m
1081     # s = max{s_w, s_h}
1082     s = np.maximum(scaling_w, scaling_h)
1083     if s < 1:
1084         s = 1
1085     # Determine the output size
1086     output = np.zeros((int(np.ceil(x_scale/s)), int(np.ceil(y_scale/s)), 3))
1087     # Compute the projection
1088     h_inv = np.linalg.pinv(h_mat) / np.linalg.pinv(h_mat)[2][2]
1089     # Create array
1090     y_pt, x_pt = np.meshgrid(np.arange(0, y_scale*s, 1*s), np.arange(0, x_scale*s, 1*s))
1091     # flattened array along axis=0
1092     pts = np.vstack((y_pt.ravel(), x_pt.ravel())).T + np.array([[y_min, x_min]])
1093     # Add third coordinates
1094     pts = np.append(pts, np.ones([len(pts), 1]), 1)
1095     # Transformation
1096     translated_pts = (np.dot(h_inv, pts.T)).T
1097     # Normalization
1098     translated_pts = translated_pts[:, :2] / translated_pts[:, [-1]]
1099     for i in range(0, x_scale):
1100         for j in range(0, y_scale):
1101             loc0 = translated_pts[i*y_scale + j, 1]
1102             loc1 = translated_pts[i*y_scale + j, 0]
1103             if (loc0 > 0) and (loc1 > 0) and (loc0 < img.shape[0] - 1) and (loc1 < img
1104 .shape[1] - 1):
1105                 output[i][j] = compute_pixel_val(img, [loc0, loc1])
1106
1107     return output.astype(np.uint8)
1108
1109 point = ['P', 'Q', 'S', 'R']
1110 # Image projection
1111 img1 = io.imread('hw3_Task1_Images/Images/Img1.JPG')
1112 PQSR1 = np.array([[642, 498], [642, 532], [666, 503], [666, 537]])
1113 PQSR = np.array([[0, 0], [0, 85], [75, 0], [75, 85]])
1114 # Plot
1115 plt.imshow(img1)
1116 plt.scatter(PQSR1[0][0], PQSR1[0][1])
1117 plt.annotate(point[0] + '({},{})'.format(PQSR1[0][0], PQSR1[0][1]), (PQSR1[0][0]-50,
1118 PQSR1[0][1]-50), c='r')
1119 plt.scatter(PQSR1[1][0], PQSR1[1][1])

```

```

1120 plt.annotate(point[1] + '({},{})'.format(PQSR1[1][0], PQSR1[1][1]), (PQSR1[1][0]-100,
    PQSR1[1][1]+100), c='r')
1121 plt.scatter(PQSR1[2][0], PQSR1[2][1])
1122 plt.annotate(point[2] + '({},{})'.format(PQSR1[2][0], PQSR1[2][1]), (PQSR1[2][0]+100,
    PQSR1[2][1]-100), c='r')
1123 plt.scatter(PQSR1[3][0], PQSR1[3][1])
1124 plt.annotate(point[3] + '({},{})'.format(PQSR1[3][0], PQSR1[3][1]), (PQSR1[3][0]+50,
    PQSR1[3][1]+50), c='r')
1125 plt.axis('off')
1126 plt.savefig('img1_labelledpt.jpeg')
1127 plt.show()
1128 plt.clf()
1129 # Computation
1130 H = compute_hc(PQSR1, PQSR)
1131 output = ptp_mapping(img1, H)
1132 plt.imshow(output)
1133 plt.axis('off')
1134 plt.savefig("img1_results.jpeg")
1135 plt.show()
1136 plt.clf()

1137 img2 = io.imread('hw3_Task1_Images/Images/Img2.jpeg')
1138 PQSR2 = np.array([[480, 722], [481, 874], [600, 739], [606, 923]])
1139 PQSR = np.array([[0, 0], [0, 74], [84, 0], [84, 74]])
1140 # Plot
1141 plt.imshow(img2)
1142 for i in range(4):
1143     plt.scatter(PQSR2[i][0], PQSR2[i][1])
1144     plt.annotate(point[i] + '({},{})'.format(PQSR2[i][0], PQSR2[i][1]), (PQSR2[i][0],
        PQSR2[i][1]), c='r')
1145 plt.axis('off')
1146 plt.savefig('img2_labelledpt.jpeg')
1147 plt.show()
1148 plt.clf()
1149 # Computation
1150 H2 = compute_hc(PQSR2, PQSR)
1151 output = ptp_mapping(img2, H2)
1152 plt.imshow(output)
1153 plt.axis('off')
1154 plt.savefig("img2_results.jpeg")
1155 plt.show()
1156 plt.clf()

1157 img3 = io.imread('hw3_Task1_Images/Images/Img3.JPG')
1158 PQSR3 = np.array([[2060, 700], [2092, 1483], [2666, 720], [2695, 1333]])
1159 PQSR = np.array([[0, 0], [0, 36], [55, 0], [55, 36]])
1160 # Plot
1161 plt.imshow(img3)
1162 for i in range(4):
1163     plt.scatter(PQSR3[i][0], PQSR3[i][1])
1164     plt.annotate(point[i] + '({},{})'.format(PQSR3[i][0], PQSR3[i][1]), (PQSR3[i][0],
        PQSR3[i][1]), c='r')
1165 plt.title('painting3.jpeg')
1166 plt.axis('off')
1167 plt.savefig('img3_labelledpt.jpeg')
1168 plt.show()
1169 plt.clf()
1170 # Computation
1171 H3 = compute_hc(PQSR3, PQSR)
1172 output = ptp_mapping(img3, H3)
1173 plt.imshow(output)
1174 plt.axis('off')
1175 plt.savefig("img3_results.jpeg")

```

```

1178 plt.show()
1180 plt.clf()

1182 img4 = io.imread('hw3_Task1_Images/Images/Img4.jpeg')
PQSR4 = np.array([[625, 622], [609, 1044], [778, 521], [764, 970]])
1184 PQSR = np.array([[0, 0], [0, 120], [40, 0], [40, 120]])
# Plot
1186 plt.imshow(img4)
for i in range(4):
    plt.scatter(PQSR4[i][0], PQSR4[i][1])
    plt.annotate(point[i] + '({},{})'.format(PQSR4[i][0], PQSR4[i][1]), (PQSR4[i][0],
    PQSR4[i][1]), c='r')
1190 plt.axis('off')
plt.savefig('img4_method1_labels.jpeg')
1192 plt.show()
plt.clf()
1194 # Computation
H4 = compute_hc(PQSR4, PQSR)
1196 output = ptp_mapping(img4, H4)
plt.imshow(output)
1198 plt.axis('off')
plt.savefig("img4_results.jpeg")
1200 plt.show()
plt.clf()

1204 img5 = io.imread('hw3_Task1_Images/Images/img5.jpeg')
PQSR5 = np.array([[573, 302], [572, 386], [692, 296], [692, 384]])
1206 PQSR = np.array([[0, 0], [0, 100], [200, 0], [200, 100]])
# Plot
1208 plt.imshow(img5)
for i in range(4):
    plt.scatter(PQSR5[i][0], PQSR5[i][1])
    plt.annotate(point[i] + '({},{})'.format(PQSR5[i][0], PQSR5[i][1]), (PQSR5[i][0],
    PQSR5[i][1]), c='r')
1212 plt.axis('off')
plt.savefig('img5_labelledpt.jpeg')
1214 plt.show()
plt.clf()
1216 # Computation
H5 = compute_hc(PQSR5, PQSR)
1218 output = ptp_mapping(img5, H5)
plt.imshow(output)
1220 plt.axis('off')
plt.savefig("img5_results.jpeg")
1222 plt.show()
plt.clf()

1226 img6 = io.imread('hw3_Task1_Images/Images/Img6.jpeg')
PQSR6 = np.array([[232, 63], [232, 1466], [693, 133], [692, 1453]])
1228 PQSR = np.array([[0, 0], [0, 50], [20, 0], [20, 50]])
# Plot
1230 plt.imshow(img6)
for i in range(4):
    plt.scatter(PQSR6[i][0], PQSR6[i][1])
    plt.annotate(point[i] + '({},{})'.format(PQSR6[i][0], PQSR6[i][1]), (PQSR6[i][0],
    PQSR6[i][1]), c='r')
1234 plt.axis('off')
plt.savefig('img6_method1_labels.jpeg')
1236 plt.show()
plt.clf()
1238 # Computation

```

```

1240     H6 = compute_hc(PQSR6, PQSR)
1241     output = ptp_mapping(img6, H6)
1242     plt.imshow(output)
1243     plt.axis('off')
1244     plt.savefig("img6_results.jpeg")
1245     plt.show()
1246     plt.clf()

```

method1.py

4.2 method2.py

```

1000 import numpy as np
1001 import cv2
1002 import matplotlib.pyplot as plt
1003 from skimage import io
1004
1005
1006 def compute_pixel_val(img, loc):
1007     """
1008         This module returns the pixel value given by weighting average of neighbor pixels.
1009         :param img: the mapping img.
1010         :param loc: the coordinates of mapped points.
1011         :return: the pixel value.
1012     """
1013
1014     loc0_f = np.int(np.floor(loc[0]))
1015     loc1_f = np.int(np.floor(loc[1]))
1016     loc0_c = np.int(np.ceil(loc[0]))
1017     loc1_c = np.int(np.ceil(loc[1]))
1018     a = img[loc0_f][loc1_f]
1019     b = img[loc0_f][loc1_c]
1020     c = img[loc0_c][loc1_f]
1021     d = img[loc0_c][loc1_c]
1022     dx = float(loc[0] - loc0_f)
1023     dy = float(loc[1] - loc1_f)
1024     Wa = 1 / np.linalg.norm([dx, dy])
1025     Wb = 1 / np.linalg.norm([1 - dx, dy])
1026     Wc = 1 / np.linalg.norm([dx, 1 - dy])
1027     Wd = 1 / np.linalg.norm([1 - dx, 1 - dy])
1028     output = (a * Wa + b * Wb + c * Wc + d * Wd) / (Wa + Wb + Wc + Wd)
1029
1030     return output
1031
1032 def img_map(img, h_mat):
1033     # Determine the coordinates of domain plane
1034     m, n, d = np.shape(img)
1035     P = np.array([0, 0, 1])
1036     Q = np.array([0, m-1, 1])
1037     S = np.array([n-1, 0, 1])
1038     R = np.array([n-1, m-1, 1])
1039
1040     # Compute the projection of corners
1041     locP = np.dot(h_mat, P) / np.dot(h_mat, P)[2]
1042     locQ = np.dot(h_mat, Q) / np.dot(h_mat, Q)[2]
1043     locS = np.dot(h_mat, S) / np.dot(h_mat, S)[2]
1044     locR = np.dot(h_mat, R) / np.dot(h_mat, R)[2]
1045
1046     # Determin the size of mapped image
1047     x_min = np.int(np.floor(np.min([locP[1], locQ[1], locR[1], locS[1]])))
1048     x_max = np.int(np.ceil(np.max([locP[1], locQ[1], locR[1], locS[1]])))
1049     x_scale = x_max - x_min
1050     y_min = np.int(np.floor(np.min([locP[0], locQ[0], locR[0], locS[0]])))
1051     y_max = np.int(np.ceil(np.max([locP[0], locQ[0], locR[0], locS[0]])))

```

```

y_scale = y_max - y_min
1052
# Compute scaling factor
1054 # if we fix width, then the scaling factor  $s_w = w_o / w_i$ 
scaling_w = y_scale / n
1056 # if we fix length, then the scaling factor  $s_h = h_o / h_i$ 
scaling_h = x_scale / m
1058 #  $s = \max\{s_w, s_h\}$ 
s = np.maximum(scaling_w, scaling_h)
1060 if s < 1:
    s = 1
1062 # Determine the output size
output = np.zeros((int(np.ceil(x_scale/s)), int(np.ceil(y_scale/s)), 3))
1064 # Compute the projection
h_inv = np.linalg.pinv(h_mat) / np.linalg.pinv(h_mat)[2][2]
1066 # Create array
y_pt, x_pt = np.meshgrid(np.arange(0, y_scale*s, 1*s), np.arange(0, x_scale*s, 1*s))
1068 # flattened array along axis=0
pts = np.vstack((y_pt.ravel(), x_pt.ravel())).T + np.array([[y_min, x_min]])
1070 # Add third coordinates
pts = np.append(pts, np.ones([len(pts), 1]), 1)
1072 # Transformation
translated_pts = (np.dot(h_inv, pts.T)).T
1074 # Normalization
translated_pts = translated_pts[:, :2] / translated_pts[:, [-1]]
1076 for i in range(0, x_scale):
    for j in range(0, y_scale):
        loc0 = translated_pts[i*y_scale + j, 1]
        loc1 = translated_pts[i*y_scale + j, 0]
        if (loc0 > 0) and (loc1 > 0) and (loc0 < img.shape[0] - 1) and (loc1 < img
        .shape[1] - 1):
            output[i][j] = compute_pixel_val(img, [loc0, loc1])
1082
return output.astype(np.uint8)
1084
1086 def H_projective(pts):
1087     l1 = np.cross(pts[0], pts[1])
1088     l2 = np.cross(pts[2], pts[3])
1089     int_pt12 = np.cross(l1, l2) / np.cross(l1, l2)[2]
1090
1091     l3 = np.cross(pts[0], pts[2])
1092     l4 = np.cross(pts[1], pts[3])
1093     int_pt34 = np.cross(l3, l4) / np.cross(l3, l4)[2]
1094
1095     l_VL = np.cross(int_pt12, int_pt34) / np.cross(int_pt12, int_pt34)[2]
1096     H = np.array([[1, 0, 0], [0, 1, 0], [l_VL[0], l_VL[1], l_VL[2]]])
1098
return H
1100
1101 def H_affine(pts):
1102     l1 = np.cross(pts[0], pts[1]) / np.cross(pts[0], pts[1])[2]
1103     m1 = np.cross(pts[0], pts[2]) / np.cross(pts[0], pts[2])[2]
1104     l2 = np.cross(pts[1], pts[3]) / np.cross(pts[1], pts[3])[2]
1105     m2 = np.cross(pts[2], pts[3]) / np.cross(pts[2], pts[3])[2]
1106
1107     A = np.array([[l1[0] * m1[0], l1[0] * m1[1] + l1[1] * m1[0]], [l2[0] * m2[0], l2
1108 [0] * m2[1] + l2[1] * m2[0]]])
1109     b = np.array([-l1[1] * m1[1], -l2[1] * m2[1]])
1110     S = np.zeros((2, 2), dtype=float)
1111     S[0][0] = np.dot(np.linalg.pinv(A), b)[0]
1112     S[0][1] = np.dot(np.linalg.pinv(A), b)[1]

```

```

1112     S[1][0] = np.dot(np.linalg.pinv(A), b)[1]
1113     S[1][1] = 1
1114
1115     U, D, V = np.linalg.svd(S, full_matrices=True)
1116     sol = np.dot(np.dot(U, np.sqrt(np.diag(D))), U.transpose())
1117     H = np.array([[sol[0][0], sol[0][1], 0], [sol[1][0], sol[1][1], 0], [0, 0, 1]])
1118
1119     return H
1120
1121
1122 point = ["P", "Q", "S", "R"]
1123 # Remove distortion of img1
1124 img1 = io.imread('hw3_Task1_Images/Images/Img1.JPG')
1125 PQSR1 = np.array([[462, 165], [431, 730], [1415, 482], [1462, 810]])
1126 # Plot
1127 plt.imshow(img1)
1128 for i in range(4):
1129     plt.scatter(PQSR1[i][0], PQSR1[i][1])
1130     plt.annotate(point[i] + '({},{})'.format(PQSR1[i][0], PQSR1[i][1]), (PQSR1[i][0],
1131     PQSR1[i][1]), c='r')
1132     plt.axis('off')
1133 plt.savefig('img1_method2_labelled1.jpeg')
1134 plt.show()
1135 plt.clf()
1136 # Computation
1137 pts = np.array([[462, 165, 1], [431, 730, 1], [1415, 482, 1], [1462, 810, 1]])
1138 H1 = H_projective(pts)
1139 output1 = img_map(img1, H1)
1140 plt.imshow(output1, cmap='gray')
1141 plt.axis('off')
1142 plt.savefig("img1_method2_projective.jpeg")
1143 plt.show()
1144 plt.clf()
1145 H2 = H_affine(pts)
1146 output2 = img_map(img1, np.dot(np.linalg.pinv(H2), H1))
1147 plt.imshow(output2, cmap='gray')
1148 plt.axis('off')
1149 plt.savefig("img1_method2_affine.jpeg")
1150 plt.show()
1151 plt.clf()
1152
1153
1154 # Remove distortion of img2
1155 img2 = io.imread('hw3_Task1_Images/Images/Img2.jpeg')
1156 PQSR2 = np.array([[368, 552], [362, 853], [621, 510], [642, 974]])
1157 # Plot
1158 plt.imshow(img2)
1159 for i in range(4):
1160     plt.scatter(PQSR2[i][0], PQSR2[i][1])
1161     plt.annotate(point[i] + '({},{})'.format(PQSR2[i][0], PQSR2[i][1]), (PQSR2[i][0],
1162     PQSR2[i][1]), c='r')
1163     plt.axis('off')
1164 plt.savefig('img2_method2_labels.jpeg')
1165 plt.show()
1166 plt.clf()
1167 # Computation
1168 pts = np.array([[368, 552, 1], [362, 853, 1], [621, 510, 1], [642, 974, 1]])
1169 H1 = H_projective(pts)
1170 output1 = img_map(img2, H1)
1171 plt.imshow(output1, cmap='gray')
1172 plt.axis('off')
1173 plt.savefig("img2_method2_projective.jpeg")
1174 plt.show()

```

```

1174 plt.clf()
1175 H2 = H_affine(pts)
1176 output2 = img_map(img2, np.dot(np.linalg.pinv(H2), H1))
1177 plt.imshow(output2, cmap='gray')
1178 plt.axis('off')
1179 plt.savefig("img2_method2_affine.jpeg")
1180 plt.show()
1181 plt.clf()
1182
1184 # Remove distortion of img3
1185 img3 = io.imread('hw3_Task1_Images/Images/Img3.JPG')
1186 PQSR3 = np.array([[2060, 700], [2092, 1483], [2666, 720], [2695, 1333]])
1187 # Plot
1188 plt.imshow(img3, cmap='gray')
1189 for i in range(4):
1190     plt.scatter(PQSR3[i][0], PQSR3[i][1])
1191     plt.annotate(point[i] + '({},{})'.format(PQSR3[i][0], PQSR3[i][1]), (PQSR3[i][0],
1192     PQSR3[i][1]), c='r')
1193 plt.axis('off')
1194 plt.savefig('img3_method2_labels.jpeg')
1195 plt.show()
1196 plt.clf()
1197 # Computation
1198 pts = np.array([[2060, 700, 1], [2092, 1483, 1], [2666, 720, 1], [2695, 1333, 1]])
1199 H1 = H_projective(pts)
1200 output1 = img_map(img3, H1)
1201 plt.imshow(output1, cmap='gray')
1202 plt.axis('off')
1203 plt.savefig("img3_method2_projective.jpeg")
1204 plt.show()
1205 plt.clf()
1206 H2 = H_affine(pts)
1207 output2 = img_map(img3, np.dot(np.linalg.pinv(H2), H1))
1208 plt.imshow(output2, cmap='gray')
1209 plt.axis('off')
1210 plt.savefig("img3_method2_affine.jpeg")
1211 plt.show()
1212 plt.clf()
1213
1214 # Remove distortion of img4
1215 img4 = io.imread('hw3_Task1_Images/Images/Img4.jpeg')
1216 PQSR4 = np.array([[625, 622], [609, 1044], [778, 521], [764, 970]])
1217 # Plot
1218 plt.imshow(img4)
1219 for i in range(4):
1220     plt.scatter(PQSR4[i][0], PQSR4[i][1])
1221     plt.annotate(point[i] + '({},{})'.format(PQSR4[i][0], PQSR4[i][1]), (PQSR4[i][0],
1222     PQSR4[i][1]), c='r')
1223 plt.axis('off')
1224 plt.savefig('img4_method2_labels.jpeg')
1225 plt.show()
1226 plt.clf()
1227 # Computation
1228 pts = np.array([[625, 622, 1], [609, 1044, 1], [778, 521, 1], [764, 970, 1]])
1229 H1 = H_projective(pts)
1230 output1 = img_map(img4, H1)
1231 plt.imshow(output1, cmap='gray')
1232 plt.axis('off')
1233 plt.savefig("img4_method2_projective.jpeg")
1234 plt.show()
1235 plt.clf()
1236 H2 = H_affine(pts)

```

```

1236 output2 = img_map(img4, np.dot(np.linalg.pinv(H2), H1))
1237 plt.imshow(output2, cmap='gray')
1238 plt.axis('off')
1239 plt.savefig("img4_method2_affine.jpeg")
1240 plt.show()
1241 plt.clf()
1242
# Remove distortion of img 5
1244 img5 = io.imread('hw3_Task1_Images/Images/img5.jpeg')
1245 PQSR5 = np.array([[573, 302], [572, 386], [692, 296], [692, 384]])
1246 # Plot
1247 plt.imshow(img5)
1248 for i in range(4):
    plt.scatter(PQSR5[i][0], PQSR5[i][1])
    plt.annotate(point[i] + '({},{})'.format(PQSR5[i][0], PQSR5[i][1]), (PQSR5[i][0],
    PQSR5[i][1]), c='r')
    plt.axis('off')
1250 plt.savefig('img5_method2_labels.jpeg')
1251 plt.show()
1252 plt.clf()
1253 # Computation
1254 pts = np.array([[573, 302, 1], [572, 386, 1], [692, 296, 1], [692, 384, 1]])
1255 H1 = H_projective(pts)
1256 output1 = img_map(img5, H1)
1257 plt.imshow(output1, cmap='gray')
1258 plt.axis('off')
1259 plt.savefig("img5_method2_projective.jpeg")
1260 plt.show()
1261 plt.clf()
1262 H2 = H_affine(pts)
1263 output2 = img_map(img5, np.dot(np.linalg.pinv(H2), H1))
1264 plt.imshow(output2, cmap='gray')
1265 plt.axis('off')
1266 plt.savefig("img5_method2_affine.jpeg")
1267 plt.show()
1268 plt.clf()
1269
# Remove distortion of img6
1272 img6 = io.imread('hw3_Task1_Images/Images/Img6.jpeg')
1273 PQSR6 = np.array([[232, 63], [232, 1466], [693, 133], [692, 1453]])
1274 # Plot
1275 plt.imshow(img6)
1276 for i in range(4):
    plt.scatter(PQSR6[i][0], PQSR6[i][1])
    plt.annotate(point[i] + '({},{})'.format(PQSR6[i][0], PQSR6[i][1]), (PQSR6[i][0],
    PQSR6[i][1]), c='r')
    plt.axis('off')
1278 plt.savefig('img6_method2_labels.jpeg')
1279 plt.show()
1280 plt.clf()
1281 # Computation
1282 pts = np.array([[232, 63, 1], [232, 1466, 1], [693, 133, 1], [692, 1453, 1]])
1283 H1 = H_projective(pts)
1284 output1 = img_map(img6, H1)
1285 plt.imshow(output1, cmap='gray')
1286 plt.axis('off')
1287 plt.savefig("img6_method2_projective.jpeg")
1288 plt.show()
1289 plt.clf()
1290 H2 = H_affine(pts)
1291 output2 = img_map(img6, np.dot(np.linalg.pinv(H2), H1))
1292 plt.imshow(output2, cmap='gray')
1293 plt.axis('off')
1294 plt.savefig("img6_method2_affine.jpeg")

```

```

1298    plt.show()
1299    plt.clf()

```

method2.py

4.3 method3.py

```

1000 import numpy as np
1001 import cv2
1002 import matplotlib.pyplot as plt
1003 from skimage import io
1004
1005
1006 import numpy as np
1007 import cv2
1008 import matplotlib.pyplot as plt
1009 from skimage import io
1010
1011
1012 def compute_pixel_val(img, loc):
1013     """
1014         This module returns the pixel value given by weighting average of neighbor pixels.
1015         :param img: the mapping img.
1016         :param loc: the coordinates of mapped points.
1017         :return: the pixel value.
1018     """
1019
1020     loc0_f = np.int(np.floor(loc[0]))
1021     loc1_f = np.int(np.floor(loc[1]))
1022     loc0_c = np.int(np.ceil(loc[0]))
1023     loc1_c = np.int(np.ceil(loc[1]))
1024     a = img[loc0_f][loc1_f]
1025     b = img[loc0_f][loc1_c]
1026     c = img[loc0_c][loc1_f]
1027     d = img[loc0_c][loc1_c]
1028     dx = float(loc[0] - loc0_f)
1029     dy = float(loc[1] - loc1_f)
1030     Wa = 1 / np.linalg.norm([dx, dy])
1031     Wb = 1 / np.linalg.norm([1 - dx, dy])
1032     Wc = 1 / np.linalg.norm([dx, 1 - dy])
1033     Wd = 1 / np.linalg.norm([1 - dx, 1 - dy])
1034     output = (a * Wa + b * Wb + c * Wc + d * Wd) / (Wa + Wb + Wc + Wd)
1035
1036     return output
1037
1038 def img_map(img, h_mat):
1039     # Determine the coordinates of domain plane
1040     m, n, d = np.shape(img)
1041     P = np.array([0, 0, 1])
1042     Q = np.array([0, m-1, 1])
1043     S = np.array([n-1, 0, 1])
1044     R = np.array([n-1, m-1, 1])
1045
1046     # Compute the projection of corners
1047     locP = np.dot(h_mat, P) / np.dot(h_mat, P)[2]
1048     locQ = np.dot(h_mat, Q) / np.dot(h_mat, Q)[2]
1049     locS = np.dot(h_mat, S) / np.dot(h_mat, S)[2]
1050     locR = np.dot(h_mat, R) / np.dot(h_mat, R)[2]
1051
1052     # Determin the size of mapped image
1053     x_min = np.int(np.floor(np.min([locP[1], locQ[1], locR[1], locS[1]])))
1054     x_max = np.int(np.ceil(np.max([locP[1], locQ[1], locR[1], locS[1]])))
1055     x_scale = x_max - x_min
1056     y_min = np.int(np.floor(np.min([locP[0], locQ[0], locR[0], locS[0]])))

```

```

1056     y_max = np.int(np.ceil(np.max([locP[0], locQ[0], locR[0], locS[0]])))
1058     y_scale = y_max - y_min
1059
1060     # Compute scaling factor
1061     # if we fix width, then the scaling factor s_w = w_o / w_i
1062     scaling_w = y_scale / n
1063     # if we fix length, then the scaling factor s_h = h_o / h_i
1064     scaling_h = x_scale / m
1065     # s = max{s_w, s_h}
1066     s = np.maximum(scaling_w, scaling_h)
1067     if s < 1:
1068         s = 1
1069
1070     # Determine the output size
1071     output = np.zeros((int(np.ceil(x_scale/s)), int(np.ceil(y_scale/s)), 3))
1072     # Compute the projection
1073     h_inv = np.linalg.pinv(h_mat) / np.linalg.pinv(h_mat)[2][2]
1074     # Create array
1075     y_pt, x_pt = np.meshgrid(np.arange(0, y_scale*s, 1*s), np.arange(0, x_scale*s, 1*s))
1076     # flattened array along axis=0
1077     pts = np.vstack((y_pt.ravel(), x_pt.ravel())).T + np.array([[y_min, x_min]])
1078     # Add third coordinates
1079     pts = np.append(pts, np.ones([len(pts), 1]), 1)
1080     # Transformation
1081     translated_pts = (np.dot(h_inv, pts.T)).T
1082     # Normalization
1083     translated_pts = translated_pts[:, :2] / translated_pts[:, [-1]]
1084     for i in range(0, x_scale):
1085         for j in range(0, y_scale):
1086             loc0 = translated_pts[i*y_scale + j, 1]
1087             loc1 = translated_pts[i*y_scale + j, 0]
1088             if (loc0 > 0) and (loc1 > 0) and (loc0 < img.shape[0] - 1) and (loc1 < img
1089             .shape[1] - 1):
1090                 output[i][j] = compute_pixel_val(img, [loc0, loc1])
1091
1092     return output.astype(np.uint8)
1093
1094 def compute_coe(array1, array2):
1095     output = np.array([array1[0] * array2[0], (array1[0]*array2[1]+array1[1]*array2
1096     [0])/2,
1097                         array1[1]*array2[2], (array1[0]*array2[2]+array1[2]*array2[0])
1098     /2,
1099                         (array1[1]*array2[2]+array1[2]*array2[1])/2])
1100
1101     return output
1102
1103 def H_matrix(pts):
1104     # PQ \ PS
1105     l1 = np.cross(pts[0], pts[1]) / np.max(np.cross(pts[0], pts[1]))
1106     m1 = np.cross(pts[0], pts[2]) / np.max(np.cross(pts[0], pts[2]))
1107     # PS \ SR
1108     l2 = np.cross(pts[0], pts[2]) / np.max(np.cross(pts[0], pts[2]))
1109     m2 = np.cross(pts[2], pts[3]) / np.max(np.cross(pts[2], pts[3]))
1110     # SR \ QR
1111     l3 = np.cross(pts[2], pts[3]) / np.max(np.cross(pts[2], pts[3]))
1112     m3 = np.cross(pts[1], pts[3]) / np.max(np.cross(pts[1], pts[3]))
1113     # PQ \ QR
1114     l4 = np.cross(pts[0], pts[1]) / np.max(np.cross(pts[0], pts[1]))
1115     m4 = np.cross(pts[1], pts[3]) / np.max(np.cross(pts[1], pts[3]))
1116     # AB \ BC
1117     l5 = np.cross(pts[4], pts[5]) / np.max(np.cross(pts[4], pts[5]))
1118     m5 = np.cross(pts[5], pts[6]) / np.max(np.cross(pts[5], pts[6]))

```

```

1116     # Compute S_mat
1117     A = []
1118     A.append(compute_coe(11, m1))
1119     A.append(compute_coe(12, m2))
1120     A.append(compute_coe(13, m3))
1121     A.append(compute_coe(14, m4))
1122     A.append(compute_coe(15, m5))
1123     A = np.asarray(A)
1124     b = np.array([[-11[2] * m1[2]], [-12[2] * m2[2]], [-13[2] * m3[2]], [-14[2] * m4
1125 [2]], [-15[2] * m5[2]]])
1126     sol = np.dot(np.linalg.pinv(A), b) / np.max(np.abs(np.dot(np.linalg.pinv(A), b)))
1127     S = np.zeros((2, 2), dtype=float)
1128     S[0][0] = sol[0]
1129     S[0][1] = sol[1] / 2
1130     S[1][0] = sol[1] / 2
1131     S[1][1] = sol[2]
1132     # Compute SVD of S
1133     S_mat = np.array(S, dtype=float)
1134     U, D, V = np.linalg.svd(S_mat, full_matrices=True)
1135     temp = np.dot(U, np.sqrt(np.diag(D))), U.transpose()
1136     v_vec = np.dot(np.linalg.pinv(temp), np.array([sol[3] / 2, sol[4] / 2]))
1137     H = np.array([[temp[0][0], temp[0][1], 0], [temp[1][0], temp[1][1], 0], [v_vec[0],
1138 v_vec[1], 1]], dtype=float)

1139     return H

1140
1141
1142 # # Method 3 for img1
1143 # img1 = io.imread('hw3_Task1_Images/Images/Img1.JPG')
1144 # PQSR1 = np.array([[462, 165], [431, 730], [1415, 482], [1462, 810], [165, 69], [98,
1145 876], [425, 883]])
1146 # point1 = ["P", "Q", "S", "R", "A", "B", "C"]
1147 # # Plot
1148 # plt.imshow(img1)
1149 # for i in range(7):
1150 #     plt.scatter(PQSR1[i][0], PQSR1[i][1])
1151 #     plt.annotate(point1[i] + '({},{})'.format(PQSR1[i][0], PQSR1[i][1]), (PQSR1[i]
1152 [0], PQSR1[i][1]), c='r')
1153 # plt.axis('off')
1154 # plt.savefig('img1_method3_labels.jpeg')
1155 # plt.show()
1156 # plt.clf()
1157 # pts = np.array([[462, 165, 1], [431, 730, 1], [1415, 482, 1], [1462, 810, 1], [165,
1158 69, 1], [98, 876, 1], [425, 883, 1]])
1159 # # Computation
1160 # H1 = H_matrix(pts)
1161 # output1 = img_map(img1, np.linalg.inv(H1))
1162 # plt.imshow(output1, cmap='gray')
1163 # plt.axis('off')
1164 # plt.savefig("img1_method3_result.jpeg")
1165 # plt.show()
1166 # plt.clf()

1167 # Method 3 for img2
1168 img2 = io.imread('hw3_Task1_Images/Images/Img2.jpeg')
1169 PQSR2 = np.array([[368, 552], [362, 853], [621, 510], [642, 974], [309, 342], [291,
1170 896], [496, 1023]])
1171 point2 = ["P", "Q", "S", "R", "A", "B", "C"]
1172 # Plot
1173 plt.imshow(img2)
1174 for i in range(7):
1175     plt.scatter(PQSR2[i][0], PQSR2[i][1])
1176     plt.annotate(point2[i] + '({},{})'.format(PQSR2[i][0], PQSR2[i][1]), (PQSR2[i][0],
1177 PQSR2[i][1]), c='r')

```

```

1174 plt.axis('off')
1175 plt.savefig('img2_method3_labels.jpeg')
1176 plt.show()
1177 plt.clf()
1178 pts = np.array([[368, 552, 1], [362, 853, 1], [621, 510, 1], [642, 974, 1], [309, 342,
1179 1], [291, 896, 1], [496, 1023, 1]])
# Computation
H2 = H_matrix(pts)
1180 output2 = img_map(img2, np.linalg.inv(H2))
plt.imshow(output2, cmap='gray')
1182 plt.axis('off')
plt.savefig("img2_method3_result.jpeg")
1184 plt.show()
plt.clf()
1186
# Method 3 for image3
1188 point = ['P', 'Q', 'S', 'R', 'A', 'B', 'C']
img3 = io.imread('hw3_Task1_Images/Images/Img3.JPG')
1189 PQSR3 = np.array([[2060, 700], [2092, 1483], [2666, 720], [2695, 1333], [689, 750],
1190 [741, 2088], [1774, 1621]])
plt.imshow(img3, cmap='gray')
1192 for i in range(7):
    plt.scatter(PQSR3[i][0], PQSR3[i][1])
    plt.annotate(point[i] + '({},{})'.format(PQSR3[i][0], PQSR3[i][1]), (PQSR3[i][0],
    PQSR3[i][1]), c='r')
plt.axis('off')
1196 plt.savefig('img3_method3_labels.jpeg')
plt.show()
1198 plt.clf()
1199 pts3 = np.array([[2060, 700, 1], [2092, 1483, 1], [2666, 720, 1], [2695, 1333, 1],
1200 [689, 750, 1], [741, 2088, 1], [1774, 1621, 1]])
H3 = H_matrix(pts3)
1201 output3 = img_map(img3, np.linalg.inv(H3))
plt.imshow(output3, cmap='gray')
1202 plt.axis('off')
1204 plt.savefig("img3_method3_result.jpeg")
plt.show()
1206 plt.clf()

1208
# Method 3 for image4
1209 img4 = io.imread('hw3_Task1_Images/Images/Img4.jpeg')
1210 PQSR4 = np.array([[625, 622], [609, 1044], [778, 521], [764, 970], [396, 776], [378,
1211 1153], [497, 1094]])
1212 point4 = ["P", "Q", "S", "R", "A", "B", "C"]
# Plot
1214 plt.imshow(img4)
for i in range(7):
    plt.scatter(PQSR4[i][0], PQSR4[i][1])
    plt.annotate(point4[i] + '({},{})'.format(PQSR4[i][0], PQSR4[i][1]), (PQSR4[i][0],
    PQSR4[i][1]), c='r')
1218 plt.axis('off')
plt.savefig('img4_method3_labels.jpeg')
1220 plt.show()
plt.clf()
1222 pts4 = np.array([[625, 622, 1], [609, 1044, 1], [778, 521, 1], [764, 970, 1], [396,
1223 776, 1], [378, 1153, 1], [497, 1094, 1]])
H4 = H_matrix(pts4)
1224 output4 = img_map(img4, np.linalg.pinv(H4))
plt.imshow(output4, cmap='gray')
1226 plt.axis('off')
plt.savefig("img4_method3_result.jpeg")
1228 plt.show()
plt.clf()

```

```

1230
1231
1232 # Method 3 for image5
1233 img5 = io.imread('hw3_Task1_Images/Images/img5.jpeg')
1234 PQSR5 = np.array([[573, 302], [572, 386], [692, 296], [692, 384], [110, 230], [106,
1235     316], [184, 227], [181, 313]])
1236 point5 = ["P", "Q", "S", "R", "A", "B", "C", "D"]
1237 plt.imshow(img5)
1238 for i in range(len(point5)):
1239     plt.scatter(PQSR5[i][0], PQSR5[i][1])
1240     plt.annotate(point5[i] + '({},{})'.format(PQSR5[i][0], PQSR5[i][1]), (PQSR5[i][0],
1241         PQSR5[i][1]), c='r')
1242 plt.axis('off')
1243 plt.savefig('img5_method3_labels.jpeg')
1244 plt.show()
1245 plt.clf()
1246 pts = np.array([[573, 302, 1], [572, 386, 1], [692, 296, 1], [692, 384, 1], [110, 230,
1247     1], [106, 316, 1], [184, 227, 1], [181, 313, 1]])
1248 H = H_matrix(pts)
1249 output5 = img_map(img5, np.linalg.inv(H))
1250 plt.imshow(output5, cmap='gray')
1251 plt.axis('off')
1252 plt.savefig("img5_method3_result.jpeg")
1253 plt.show()
1254 plt.clf()

```

method3.py