

# ECE 661 Computer Vision: HW2

Qiuchen Zhai  
qzhai@purdue.edu

September 9, 2020

## 1 Logic and Methodology

### 1.1 Computation of HC matrix

If  $\mathbf{x} \in \mathbb{R}^3$  is the domain point, and  $\mathbf{x}' \in \mathbb{R}^3$  is the corresponding range point. Then the linear mapping on homogeneous 3-vectors is

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

where  $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$ ,  $\mathbf{x}' = \begin{pmatrix} x'_1 \\ x'_2 \\ x'_3 \end{pmatrix}$  and  $\mathbf{H} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{pmatrix}$ .

Then

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

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

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

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

The above two equations could be rewritten in the form of following equations:

$$x' = h_{11}x + h_{12}y + h_{13} - h_{31}xx' - h_{32}yy' \quad (5)$$

$$y' = h_{21}x + h_{22}y + h_{23} - h_{31}xy' - h_{32}yy' \quad (6)$$

While there're 8 unknown parameters in HC matrix, at least 4 pairs of points are required to solve the unknowns. Assuming the points  $(x_1, y_2)$ ,  $(x_2, y_2)$ ,  $(x_3, y_3)$ ,  $(x_4, y_4)$  are mapped to the points  $(x'_1, y'_2)$ ,

$(x'_2, y'_2), (x'_3, y'_3), (x'_4, y'_4)$ , the problem is formulated as following

$$\begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x'_1 & -y_1x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y'_1 & -y_1y'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x'_2 & -y_2x'_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y'_2 & -y_2y'_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x'_3 & -y_3x'_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y'_3 & -y_3y'_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x'_4 & -y_4x'_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y'_4 & -y_4y'_4 \end{pmatrix} \begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{pmatrix} = \begin{pmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \\ x'_4 \\ y'_4 \end{pmatrix} \quad (7)$$

Or equivalently,

$$\mathbf{A}\mathbf{h} = \mathbf{b} \quad (8)$$

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

Then the unknowns could be found by solving the following equations

$$\mathbf{h} = \mathbf{A}^{-1}\mathbf{b} \quad (9)$$

## 1.2 Weighted average of neighboring pixels

While using the HC matrix to estimate coordinates, it may turn out to be non-integer values. To figure out this problem, the strategy is to take the weighted average of neighboring pixels. Let  $A, B, C, D$  denote the four neighboring pixels around the estimated coordinate, with RGB pixel values  $a, b, c, d$  respectively. Assuming the Euclidean distances between the estimated coordinate and the neighboring pixels are  $d_A, d_B, d_C, d_D$  respectively, the pixel value of the estimated point is given by

$$\frac{a * w_A + b * w_B + c * w_C + d * w_D}{w_A + w_B + w_C + w_D} \quad (10)$$

where  $w_A, w_B, w_C, w_D$  are inverse of Euclidean distance  $w_A = d_A^{-1}, w_B = d_B^{-1}, w_C = d_C^{-1}, w_D = d_D^{-1}$ .

## 2 Implementation Steps

### 2.1 Part 1: Project image to ROI in different views

- Find the region of interest (ROI) and localize the coordinates of related points in both images.
- Compute the HC matrix using the equations described in previous section.
- Map the points from the image to ROI using the HC matrix and compute the pixel values using the strategy of weighted average of neighboring points.

### 2.2 Part 2: Determine the projections between two images through the intermediate agent

- Determine the homography  $\mathbf{h}_{ab}$  between Fig. 1a and Fig. 1b.
- Determine the homography  $\mathbf{h}_{bc}$  between Fig. 1b and Fig. 1c.
- Multiply the two HC matrices.
- Project the image to ROI using the HC matrix  $\mathbf{h}_{ab}\mathbf{h}_{bc}$ .

### 3 Task 1

#### 3.1 Part 1: Project kitten image to ROI in different views



Figure 1: Labelled points of four original images



Figure 2: The projected images (Fig a-c from left to right)

**3.2 Part 2: Determine the projections between two images through the intermediate agent**



Figure 3: The original image (left )and projected image(right)

## 4 Task 2

### 4.1 Part 1: Project image to ROI in different views



Figure 4: Labeled points of four original images

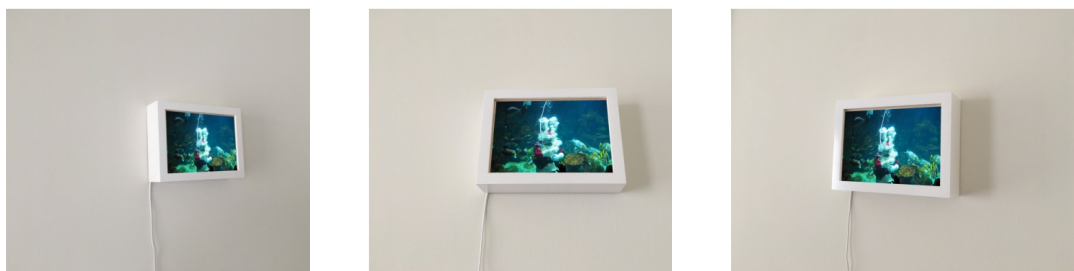


Figure 5: The projected images (Fig photoframe1-3 from left to right)

**4.2 Part 2: Determine the projections between two images through the intermediate agent**

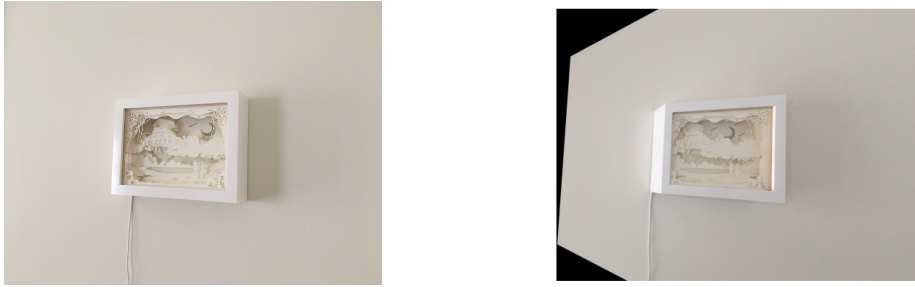


Figure 6: The original image (left )and projected image(right)

## 5 Source Code

### 5.1 Task1Part1.py

```
1000 import numpy as np
1001 import cv2
1002 import matplotlib.pyplot as plt
1003 from skimage import io
1004
1005 # read image
1006 img1 = io.imread('painting1.jpeg')
1007 img2 = io.imread('painting2.jpeg')
1008 img3 = io.imread('painting3.jpeg')
1009 img = io.imread('kittens.jpeg')
1010
1011 # record the coordinates of points
1012 PQSR1 = np.array([[298, 510], [238, 1610], [1780, 352], [1686, 1830]])
1013 PQSR2 = np.array([[344, 700], [334, 2334], [1890, 756], [1886, 2006]])
1014 PQSR3 = np.array([[106, 441], [121, 1370], [1221, 302], [1102, 1866]])
1015 PQSR = np.array([[0, 0], [0, 1125], [1920, 0], [1920, 1125]])
1016
1017 # plot the image with points of ROI
1018 # #
1019 # =====
1020
1021 # # Plot
1022 # point = ["P", "Q", "S", "R"]
1023 #
1024 # plt.imshow(img1)
1025 # for i in range(4):
1026 #     plt.scatter(PQSR1[i][0], PQSR1[i][1])
1027 #     plt.annotate(point[i] + '({},{})'.format(PQSR1[i][0], PQSR1[i][1]), (PQSR1[i]
1028 #         ][0], PQSR1[i][1]), c='r')
1029 # plt.title('painting1.jpeg')
1030 # plt.axis('off')
1031 # plt.savefig('labelled1.jpeg')
1032 # plt.clf()
1033 #
1034 # plt.imshow(img2)
1035 # for i in range(4):
1036 #     plt.scatter(PQSR2[i][0], PQSR2[i][1])
1037 #     plt.annotate(point[i] + '({},{})'.format(PQSR2[i][0], PQSR2[i][1]), (PQSR2[i]
1038 #         ][0], PQSR2[i][1]), c='r')
1039 # plt.title('painting2.jpeg')
1040 # plt.axis('off')
1041 # plt.savefig('labelled2.jpeg')
1042 # plt.clf()
1043 #
1044 # plt.imshow(img3)
1045 # for i in range(4):
1046 #     plt.scatter(PQSR3[i][0], PQSR3[i][1])
1047 #     plt.annotate(point[i] + '({},{})'.format(PQSR3[i][0], PQSR3[i][1]), (PQSR3[i]
1048 #         ][0], PQSR3[i][1]), c='r')
1049 # plt.title('painting3.jpeg')
1050 # plt.axis('off')
1051 # plt.savefig('labelled3.jpeg')
1052 # plt.clf()
1053 #
1054 # plt.imshow(img)
1055 # for i in range(4):
1056 #     plt.scatter(PQSR[i][0], PQSR[i][1])
1057 #     plt.annotate(point[i] + '({},{})'.format(PQSR[i][0], PQSR[i][1]), (PQSR[i][0],
1058 #         PQSR[i][1]), c='r')
1059 # plt.title('kittens.jpeg')
```



```

1054 # plt.axis('off')
1055 # plt.savefig('labelled4.jpeg')
1056 # plt.clf()

1058 # Compute the homography coordinates
1059 def compute_hc(src, dst):
1060     """
1061     The module computes the homography matrix by  $AH = b$ .
1062     :param src: source points.
1063     :param dst: mapped points.
1064     :return: homography matrix  $H$ .
1065     """
1066
1067     A = np.zeros((8, 8))
1068     b = np.zeros((8, 1))
1069
1070     for i in range(len(src)):
1071         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]]
1072         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]]
1073         b[2 * i] = dst[i][0]
1074         b[2 * i + 1] = dst[i][1]
1075
1076     h = np.dot(np.linalg.pinv(A), b)
1077     homo_mat = np.append(h, 1)
1078
1079     return homo_mat.reshape((3, 3))
1080
1081 def compute_pixel_val(img, loc):
1082     """
1083     This module returns the pixel value given by weighting average of neighbor pixels.
1084     :param img: the mapping img.
1085     :param loc: the coordinates of mapped points.
1086     :return: the pixel value.
1087     """
1088
1089     loc0_f = np.int(np.floor(loc[0]))
1090     loc1_f = np.int(np.floor(loc[1]))
1091     loc0_c = np.int(np.ceil(loc[0]))
1092     loc1_c = np.int(np.ceil(loc[1]))
1093     a = img[loc0_f][loc1_f]
1094     b = img[loc0_f][loc1_c]
1095     c = img[loc0_c][loc1_f]
1096     d = img[loc0_c][loc1_c]
1097     dx = float(loc[0] - loc0_f)
1098     dy = float(loc[1] - loc1_f)
1099     Wa = 1 / np.linalg.norm([dx, dy])
1100     Wb = 1 / np.linalg.norm([1 - dx, dy])
1101     Wc = 1 / np.linalg.norm([dx, 1 - dy])
1102     Wd = 1 / np.linalg.norm([1 - dx, 1 - dy])
1103     output = (a * Wa + b * Wb + c * Wc + d * Wd) / (Wa + Wb + Wc + Wd)
1104
1105     return output
1106
1107 def mapping(src_img, dst_img, points_src, h_mat):
1108     """
1109     The mapping function maps the source image to the destination.
1110     :param src_img: source image.
1111     :param dst_img: the projection image.
1112     :param points_src: the  $P'$ ,  $Q'$ ,  $S'$ ,  $R'$  points.
1113     :param h_mat: the HC matrix  $H$ .

```

```

1116         :return: the projected image.
1117         """
1118         temp = np.zeros(dst_img.shape[0:2])
1119         pts = np.array([[points_src[0][0], points_src[0][1]], [points_src[1][0],
1120         points_src[1][1]], [points_src[3][0],
1121         points_src[3][1]], [points_src[2][0], points_src[2][1]]])
1122         cv2.fillPoly(temp, [pts], 255)
1123         # plt.imshow(temp)
1124         # plt.show()
1125         # plt.clf()
1126         loc_hc = np.array([[np.dot(h_mat, np.array([j, i, 1])) for j in range(temp.shape
1127         [1])] for i in range(temp.shape[0])])
1128         loc = np.array([loc_hc[:, :, 0] / loc_hc[:, :, 2], loc_hc[:, :, 1] / loc_hc[:, :,
1129         2]])
1130         for i in range(temp.shape[0]):
1131             for j in range(temp.shape[1]):
1132                 loc0 = loc[1, i, j]
1133                 loc1 = loc[0, i, j]
1134                 if (loc0 > 0) and (loc1 > 0) and (loc0 < src_img.shape[0]-1) and (loc1 <
1135         src_img.shape[1]-1):
1136                     dst_img[i, j][temp[i, j] == 255] = compute_pixel_val(src_img, [loc0,
1137         loc1])
1138         return dst_img
1139
1140     # Image projection
1141     h_ad = compute_hc(PQSR1, PQSR)
1142     dst_img = mapping(img, img1, PQSR1, h_ad)
1143     plt.imshow(dst_img)
1144     plt.axis('off')
1145     plt.savefig('dtoa.jpeg')
1146     plt.clf()
1147
1148     h_bd = compute_hc(PQSR2, PQSR)
1149     dst_img = mapping(img, img2, PQSR2, h_bd)
1150     plt.imshow(dst_img)
1151     plt.axis('off')
1152     plt.savefig('dtob.jpeg')
1153     plt.clf()
1154
1155     h_cd = compute_hc(PQSR3, PQSR)
1156     dst_img = mapping(img, img3, PQSR3, h_cd)
1157     plt.imshow(dst_img)
1158     plt.axis('off')
1159     plt.savefig('dtoc.jpeg')
1160     plt.clf()

```

Task1Part1.py

## 5.2 Task1Part2.py

```

1000 import numpy as np
1001 import cv2
1002 import matplotlib.pyplot as plt
1003 from skimage import io
1004
1005 # read image
1006 img1 = io.imread('painting1.jpeg')
1007 img2 = io.imread('painting2.jpeg')
1008 img3 = io.imread('painting3.jpeg')
1009 img = io.imread('kittens.jpeg')
1010

```

```

1012 # record the coordinates of points
1013 PQSR1 = np.array([[298, 510], [238, 1610], [1780, 352], [1686, 1830]])
1014 PQSR2 = np.array([[344, 700], [334, 2334], [1890, 756], [1886, 2006]])
1015 PQSR3 = np.array([[106, 441], [121, 1370], [1221, 302], [1102, 1866]])
1016 PQSR = np.array([[0, 0], [0, 1125], [1920, 0], [1920, 1125]])

1018 def compute_hc(src, dst):
1019     """
1020     The module computes the homography matrix by  $AH = b$ .
1021     :param src: source points.
1022     :param dst: mapped points.
1023     :return: homography matrix H.
1024     """
1025     A = np.zeros((8, 8))
1026     b = np.zeros((8, 1))

1028     for i in range(len(src)):
1029         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]]
1030         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]]
1031         b[2 * i] = dst[i][0]
1032         b[2 * i + 1] = dst[i][1]

1034     h = np.dot(np.linalg.pinv(A), b)
1035     homo_mat = np.append(h, 1)

1037     return homo_mat.reshape((3, 3))

1040 def compute_pixel_val(img, loc):
1041     """
1042     This module returns the pixel value given by weighting average of neighbor pixels.
1043     :param img: the mapping img.
1044     :param loc: the coordinates of mapped points.
1045     :return: the pixel value.
1046     """

1048     loc0_f = np.int(np.floor(loc[0]))
1049     loc1_f = np.int(np.floor(loc[1]))
1050     loc0_c = np.int(np.ceil(loc[0]))
1051     loc1_c = np.int(np.ceil(loc[1]))
1052     a = img[loc0_f][loc1_f]
1053     b = img[loc0_f][loc1_c]
1054     c = img[loc0_c][loc1_f]
1055     d = img[loc0_c][loc1_c]
1056     dx = float(loc[0] - loc0_f)
1057     dy = float(loc[1] - loc1_f)
1058     Wa = 1 / np.linalg.norm([dx, dy])
1059     Wb = 1 / np.linalg.norm([1 - dx, dy])
1060     Wc = 1 / np.linalg.norm([dx, 1 - dy])
1061     Wd = 1 / np.linalg.norm([1 - dx, 1 - dy])
1062     output = np.divide((a * Wa + b * Wb + c * Wc + d * Wd), (Wa + Wb + Wc + Wd), where
1063                        =(Wa + Wb + Wc + Wd)!=0)
1064     return output

1066 def mapping(src_img, dst_img, points_src, h_mat):
1067     """
1068     The mapping function maps the source image to the destination.
1069     :param src_img: source image.
1070     :param dst_img: the projection image.
1071     :param points_src: the P', Q', S', R' points.

```

```

1072 :param h_mat: the HC matrix H.
1073 :return: the projected image.
1074 """
1075 temp = np.zeros(dst_img.shape[0:2])
1076 pts = np.array([[points_src[0][0], points_src[0][1]], [points_src[1][0],
1077 points_src[1][1]], [points_src[3][0],
1078 points_src[3][1]], [points_src[2][0], points_src[2][1]]])
1079 cv2.fillPoly(temp, [pts], 255)
1080 # plt.imshow(temp)
1081 # plt.show()
1082 # plt.clf()
1083 loc_hc = np.array([[np.dot(h_mat, np.array([j, i, 1])) for j in range(temp.shape
1084 [1])] for i in range(temp.shape[0])])
1085 loc = np.array([loc_hc[:, :, 0] / loc_hc[:, :, 2], loc_hc[:, :, 1] / loc_hc[:, :,
1086 2]])
1087 for i in range(temp.shape[0]):
1088     for j in range(temp.shape[1]):
1089         loc0 = loc[1, i, j]
1090         loc1 = loc[0, i, j]
1091         if (loc0 > 0) and (loc1 > 0) and (loc0 < src_img.shape[0]-1) and (loc1 <
1092 src_img.shape[1]-1):
1093             dst_img[i, j][temp[i, j] == 255] = compute_pixel_val(src_img, [loc0,
1094 loc1])
1095         else:
1096             dst_img[i, j] = 0
1097
1098 return dst_img
1099
1100 # Image projection
1101 h_ac = compute_hc(PQSR2, PQSR1)
1102 h_bc = compute_hc(PQSR3, PQSR2)
1103 pts = np.array([[0, 0], [0, img3.shape[0]], [img3.shape[1], 0], [img3.shape[1], img3.
1104 shape[0]]])
1105 dst_img = mapping(img1, img3, pts, np.dot(h_ac, h_bc))
1106 plt.imshow(dst_img)
1107 plt.axis('off')
1108 plt.savefig('atoc.jpeg')
1109 plt.clf()

```

Task1Part2.py

### 5.3 Task2Part1.py

```

1000 import numpy as np
1001 import cv2
1002 import matplotlib.pyplot as plt
1003 from skimage import io
1004
1005 # read image
1006 img1 = io.imread('photoframe1.jpeg')
1007 img2 = io.imread('photoframe2.jpeg')
1008 img3 = io.imread('photoframe3.jpeg')
1009 img = io.imread('santaclaus.jpeg')
1010
1011 # record the coordinates of points
1012 PQSR1 = np.array([[1816, 1165], [1836, 1875], [2586, 1232], [2622, 1839]])
1013 PQSR2 = np.array([[1450, 1060], [1380, 1864], [2702, 1042], [2776, 1866]])
1014 PQSR3 = np.array([[1297, 1173], [1255, 1959], [2385, 1114], [2373, 2020]])
1015 PQSR = np.array([[0, 0], [0, 1536], [2048, 0], [2048, 1536]])
1016
1017 # plot images with ROI
1018 # #

```

```

=====
# point = ["P", "Q", "S", "R"]
1020 #
# plt.imshow(img1)
1022 # for i in range(4):
#     plt.scatter(PQSR1[i][0], PQSR1[i][1])
1024 #     plt.annotate(point[i] + '({},{})'.format(PQSR1[i][0], PQSR1[i][1]), (PQSR1[i]
#         ][0], PQSR1[i][1]), c='r')
# plt.title('photoframe1.jpeg')
1026 # plt.axis('off')
# plt.savefig('task2_labelled1.jpeg')
1028 # plt.clf()
#
1030 # plt.imshow(img2)
# for i in range(4):
1032 #     plt.scatter(PQSR2[i][0], PQSR2[i][1])
#     plt.annotate(point[i] + '({},{})'.format(PQSR2[i][0], PQSR2[i][1]), (PQSR2[i]
#         ][0], PQSR2[i][1]), c='r')
1034 # plt.title('photoframe2.jpeg')
# plt.axis('off')
1036 # plt.savefig('task2_labelled2.jpeg')
# plt.clf()
1038 # #
# plt.imshow(img3)
1040 # for i in range(4):
#     plt.scatter(PQSR3[i][0], PQSR3[i][1])
1042 #     plt.annotate(point[i] + '({},{})'.format(PQSR3[i][0], PQSR3[i][1]), (PQSR3[i]
#         ][0], PQSR3[i][1]), c='r')
# plt.title('photoframe3.jpeg')
1044 # plt.axis('off')
# plt.savefig('task2_labelled3.jpeg')
1046 # plt.clf()
#
1048 # plt.imshow(img)
# for i in range(4):
1050 #     plt.scatter(PQSR[i][0], PQSR[i][1])
#     plt.annotate(point[i] + '({},{})'.format(PQSR[i][0], PQSR[i][1]), (PQSR[i][0],
#         PQSR[i][1]), c='r')
1052 # plt.title('santaclaus.jpeg')
# plt.axis('off')
1054 # plt.savefig('task2_labelled4.jpeg')
# plt.clf()
1056
1058 def compute_hc(src, dst):
    """
    1060     The module computes the homography matrix by  $AH = b$ .
    :param src: source points.
    1062     :param dst: mapped points.
    :return: homography matrix H.
    1064     """
    1066     A = np.zeros((8, 8))
    b = np.zeros((8, 1))
    1068
    for i in range(len(src)):
    1070         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]]
        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]]
    1072         b[2 * i] = dst[i][0]
        b[2 * i + 1] = dst[i][1]
    1074

```

```

1076     h = np.dot(np.linalg.pinv(A), b)
1077     homo_mat = np.append(h, 1)
1078
1079     return homo_mat.reshape((3, 3))
1080
1081 def compute_pixel_val(img, loc):
1082     """
1083     This module returns the pixel value given by weighting average of neighbor pixels.
1084     :param img: the mapping img.
1085     :param loc: the coordinates of mapped points.
1086     :return: the pixel value.
1087     """
1088
1089     loc0_f = np.int(np.floor(loc[0]))
1090     loc1_f = np.int(np.floor(loc[1]))
1091     loc0_c = np.int(np.ceil(loc[0]))
1092     loc1_c = np.int(np.ceil(loc[1]))
1093     a = img[loc0_f][loc1_f]
1094     b = img[loc0_f][loc1_c]
1095     c = img[loc0_c][loc1_f]
1096     d = img[loc0_c][loc1_c]
1097     dx = float(loc[0] - loc0_f)
1098     dy = float(loc[1] - loc1_f)
1099     Wa = 1 / np.linalg.norm([dx, dy])
1100     Wb = 1 / np.linalg.norm([1 - dx, dy])
1101     Wc = 1 / np.linalg.norm([dx, 1 - dy])
1102     Wd = 1 / np.linalg.norm([1 - dx, 1 - dy])
1103     output = (a * Wa + b * Wb + c * Wc + d * Wd) / (Wa + Wb + Wc + Wd)
1104
1105     return output
1106
1107 def mapping(src_img, dst_img, points_src, h_mat):
1108     """
1109     The mapping function maps the source image to the destination.
1110     :param src_img: source image.
1111     :param dst_img: the projection image.
1112     :param points_src: the P', Q', S', R' points.
1113     :param h_mat: the HC matrix H.
1114     :return: the projected image.
1115     """
1116
1117     temp = np.zeros(dst_img.shape[0:2])
1118     pts = np.array([[points_src[0][0], points_src[0][1]], [points_src[1][0],
1119     points_src[1][1]], [points_src[3][0],
1120     points_src[3][1]], [points_src[2][0], points_src[2][1]]])
1121     cv2.fillPoly(temp, [pts], 255)
1122     # plt.imshow(temp)
1123     # plt.show()
1124     # plt.clf()
1125     loc_hc = np.array([[np.dot(h_mat, np.array([j, i, 1])) for j in range(temp.shape
1126     [1])] for i in range(temp.shape[0])])
1127     loc = np.array([loc_hc[:, :, 0] / loc_hc[:, :, 2], loc_hc[:, :, 1] / loc_hc[:, :,
1128     2]])
1129     for i in range(temp.shape[0]):
1130         for j in range(temp.shape[1]):
1131             loc0 = loc[1, i, j]
1132             loc1 = loc[0, i, j]
1133             if (loc0 > 0) and (loc1 > 0) and (loc0 < src_img.shape[0]-1) and (loc1 <
1134             src_img.shape[1]-1):
1135                 dst_img[i, j][temp[i, j] == 255] = compute_pixel_val(src_img, [loc0,
1136                 loc1])
1137
1138     return dst_img

```

```

1134
1136 # image projection
1137 h_ad = compute_hc(PQSR1, PQSR)
1138 dst_img = mapping(img, img1, PQSR1, h_ad)
1139 plt.imshow(dst_img)
1140 plt.axis('off')
1141 plt.savefig('task2_dtoa.jpeg')
1142 plt.clf()
1143
1144 h_bd = compute_hc(PQSR2, PQSR)
1145 dst_img = mapping(img, img2, PQSR2, h_bd)
1146 plt.imshow(dst_img)
1147 plt.axis('off')
1148 plt.savefig('task2_dtob.jpeg')
1149 plt.clf()
1150
1151 h_cd = compute_hc(PQSR3, PQSR)
1152 dst_img = mapping(img, img3, PQSR3, h_cd)
1153 plt.imshow(dst_img)
1154 plt.axis('off')
1155 plt.savefig('task2_dtoc.jpeg')
1156 plt.clf()

```

Task2Part1.py

## 5.4 Task2Part2.py

```

1000 import numpy as np
1001 import cv2
1002 import matplotlib.pyplot as plt
1003 from skimage import io
1004
1005 # read image
1006 img1 = io.imread('photoframe1.jpeg')
1007 img2 = io.imread('photoframe2.jpeg')
1008 img3 = io.imread('photoframe3.jpeg')
1009 img = io.imread('santaclaus.jpeg')
1010
1011 # record the coordinates of points
1012 PQSR1 = np.array([[1816, 1165], [1836, 1875], [2586, 1232], [2622, 1839]])
1013 PQSR2 = np.array([[1450, 1060], [1380, 1864], [2702, 1042], [2776, 1866]])
1014 PQSR3 = np.array([[1297, 1173], [1255, 1959], [2385, 1114], [2373, 2020]])
1015 PQSR = np.array([[0, 0], [0, 1536], [2048, 0], [2048, 1536]])
1016
1017 def compute_hc(src, dst):
1018     """
1019     The module computes the homography matrix by  $AH = b$ .
1020     :param src: source points.
1021     :param dst: mapped points.
1022     :return: homography matrix  $H$ .
1023     """
1024     A = np.zeros((8, 8))
1025     b = np.zeros((8, 1))
1026
1027     for i in range(len(src)):
1028         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]]
1029         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]]
1030         b[2 * i] = dst[i][0]
1031         b[2 * i + 1] = dst[i][1]

```

```

1034     h = np.dot(np.linalg.pinv(A), b)
1035     homo_mat = np.append(h, 1)
1036
1037     return homo_mat.reshape((3, 3))
1038
1039 def compute_pixel_val(img, loc):
1040     """
1041     This module returns the pixel value given by weighting average of neighbor pixels.
1042     :param img: the mapping img.
1043     :param loc: the coordinates of mapped points.
1044     :return: the pixel value.
1045     """
1046
1047     loc0_f = np.int(np.floor(loc[0]))
1048     loc1_f = np.int(np.floor(loc[1]))
1049     loc0_c = np.int(np.ceil(loc[0]))
1050     loc1_c = np.int(np.ceil(loc[1]))
1051     a = img[loc0_f][loc1_f]
1052     b = img[loc0_f][loc1_c]
1053     c = img[loc0_c][loc1_f]
1054     d = img[loc0_c][loc1_c]
1055     dx = float(loc[0] - loc0_f)
1056     dy = float(loc[1] - loc1_f)
1057     Wa = 1 / np.linalg.norm([dx, dy])
1058     Wb = 1 / np.linalg.norm([1 - dx, dy])
1059     Wc = 1 / np.linalg.norm([dx, 1 - dy])
1060     Wd = 1 / np.linalg.norm([1 - dx, 1 - dy])
1061     output = np.divide((a * Wa + b * Wb + c * Wc + d * Wd), (Wa + Wb + Wc + Wd), where
1062     =(Wa + Wb + Wc + Wd)!=0)
1063     return output
1064
1065 def mapping(src_img, dst_img, points_src, h_mat):
1066     """
1067     The mapping function maps the source image to the destination.
1068     :param src_img: source image.
1069     :param dst_img: the projection image.
1070     :param points_src: the P', Q', S', R' points.
1071     :param h_mat: the HC matrix H.
1072     :return: the projected image.
1073     """
1074
1075     temp = np.zeros(dst_img.shape[0:2])
1076     pts = np.array([[points_src[0][0], points_src[0][1]], [points_src[1][0],
1077     points_src[1][1]], [points_src[3][0],
1078     points_src[3][1]], [points_src[2][0], points_src[2][1]]])
1079     cv2.fillPoly(temp, [pts], 255)
1080     # plt.imshow(temp)
1081     # plt.show()
1082     # plt.clf()
1083     loc_hc = np.array([[np.dot(h_mat, np.array([j, i, 1])) for j in range(temp.shape
1084     [1])] for i in range(temp.shape[0])])
1085     loc = np.array([loc_hc[:, :, 0] / loc_hc[:, :, 2], loc_hc[:, :, 1] / loc_hc[:, :,
1086     2]])
1087     print('yes')
1088     for i in range(temp.shape[0]):
1089         for j in range(temp.shape[1]):
1090             loc0 = loc[1, i, j]
1091             loc1 = loc[0, i, j]
1092             if (loc0 > 0) and (loc1 > 0) and (loc0 < src_img.shape[0]-1) and (loc1 <
1093     src_img.shape[1]-1):
1094                 dst_img[i, j][temp[i, j] == 255] = compute_pixel_val(src_img, [loc0,
1095     loc1])

```



```

1092         else:
1093             dst_img[i, j] = 0
1094
1095     return dst_img
1096
1097 # image projection
1098 h_ac = compute_hc(PQSR2, PQSR1)
1099 h_bc = compute_hc(PQSR3, PQSR2)
1100 pts = np.array([[0, 0], [0, img3.shape[0]], [img3.shape[1], 0], [img3.shape[1], img3.
1101                 shape[0]]])
1102 dst_img = mapping(img1, img3, pts, np.dot(h_ac, h_bc))
1103 plt.imshow(dst_img)
1104 plt.axis('off')
1105 plt.savefig('task2_atoc.jpeg')
1106 plt.clf()

```

Task2Part2.py