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

$$\mathbf{x}' = \mathbf{H}\mathbf{x} \tag{1}$$

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

$$\begin{pmatrix} x_1' \\ x_2' \\ x_3' \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} h_{11}x_1 + h_{12}x_2 + h_{13}x_3 \\ h_{21}x_1 + h_{22}x_2 + h_{23}x_3 \\ h_{31}x_1 + h_{32}x_2 + x_3 \end{pmatrix} \tag{2}$$

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

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

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

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

$$x' = h_{11}x + h_{12}y + h_{13} - h_{31}xx' - h_{32}yx' \tag{5}$$

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

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

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

Or equivalently,

$$\mathbf{Ah} = \mathbf{b} \tag{8}$$

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 $\mathbf{h} = \begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{pmatrix}$.

Then the unknowns could be found by solving the following equations

$$\mathbf{h} = \mathbf{A^{-1}b} \tag{9}$$

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

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

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 $\mathbf{h_{ab}h_{bc}}$.

3 Task 1

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

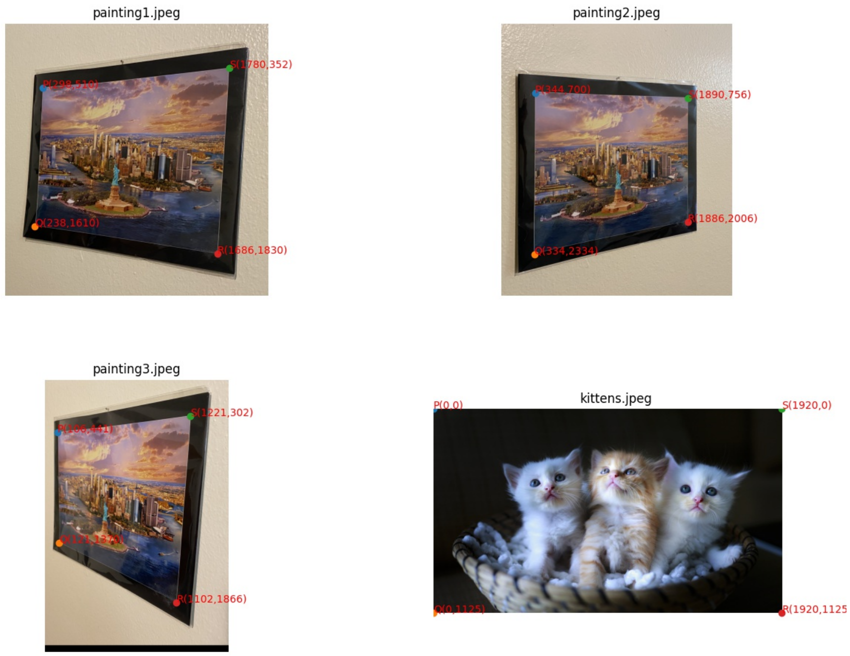

Figure 1: Labelled points of four original images

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

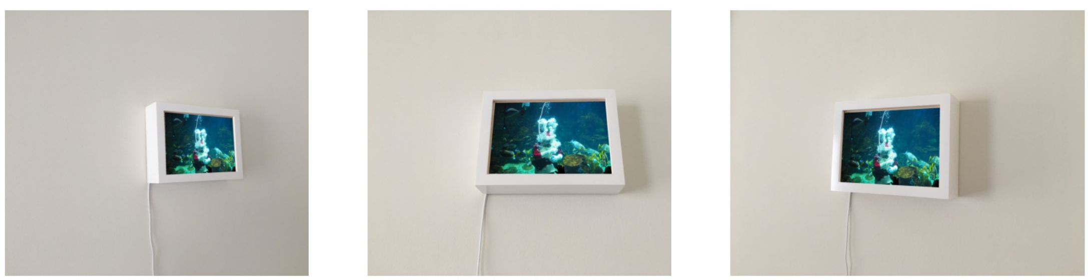

Figure 5: The projected images (Fig photoframe1-3 from left to right)

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

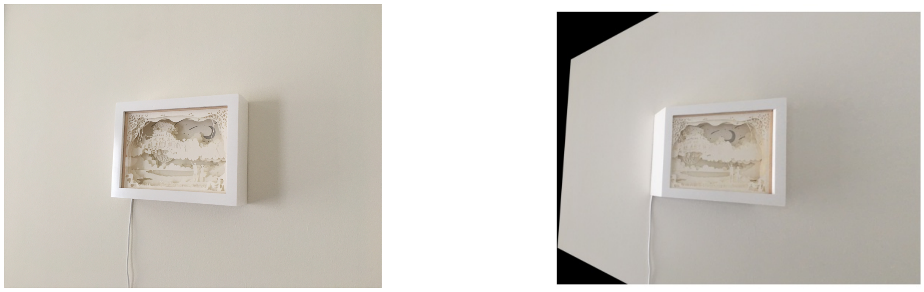

Figure 6: The original image (left)and projected image(right)

5 Source Code

5.1 Task1Part1.py

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from skimage import io

# read image
img1 = io.imread('painting1.jpeg')
img2 = io.imread('painting2.jpeg')
img3 = io.imread('painting3.jpeg')
img = io.imread('kittens.jpeg')

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

# plot the image with points of ROI
# # ====================================================================================================

# # Plot
# point = ["P", "Q", "S", "R"]
#
# plt.imshow(img1)
# for i in range(4):
#     plt.scatter(PQSR1[i][0], PQSR1[i][1])
#     plt.annotate(point[i] + '({},{})'.format(PQSR1[i][0], PQSR1[i][1]), (PQSR1[i][0], PQSR1[i][1]), c='r')
# plt.title('painting1.jpeg')
# plt.axis('off')
# plt.savefig('labelled1.jpeg')
# plt.clf()
#
# plt.imshow(img2)
# for i in range(4):
#     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')
# plt.title('painting2.jpeg')
# plt.axis('off')
# plt.savefig('labelled2.jpeg')
# plt.clf()
# #
# plt.imshow(img3)
# for i in range(4):
#     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.title('painting3.jpeg')
# plt.axis('off')
# plt.savefig('labelled3.jpeg')
# plt.clf()
#
# plt.imshow(img)
# for i in range(4):
#     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')
# plt.title('kittens.jpeg')
```

```
# plt.axis('off')
# plt.savefig('labelled4.jpeg')
# plt.clf()

# Compute the homography coordinates
def compute_hc(src, dst):
    """
    The module computes the homography matrix by AH = b.
    :param src: source points.
    :param dst: mapped points.
    :return: homography matrix H.
    """

    A = np.zeros((8, 8))
    b = np.zeros((8, 1))

    for i in range(len(src)):
        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]]
        b[2 * i] = dst[i][0]
        b[2 * i + 1] = dst[i][1]

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

    return homo_mat.reshape((3, 3))


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

    loc0_f = np.int(np.floor(loc[0]))
    loc1_f = np.int(np.floor(loc[1]))
    loc0_c = np.int(np.ceil(loc[0]))
    loc1_c = np.int(np.ceil(loc[1]))
    a = img[loc0_f][loc1_f]
    b = img[loc0_f][loc1_c]
    c = img[loc0_c][loc1_f]
    d = img[loc0_c][loc1_c]
    dx = float(loc[0] - loc0_f)
    dy = float(loc[1] - loc1_f)
    Wa = 1 / np.linalg.norm([dx, dy])
    Wb = 1 / np.linalg.norm([1 - dx, dy])
    Wc = 1 / np.linalg.norm([dx, 1 - dy])
    Wd = 1 / np.linalg.norm([1 - dx, 1 - dy])
    output = (a * Wa + b * Wb + c * Wc + d * Wd) / (Wa + Wb + Wc + Wd)

    return output


def mapping(src_img, dst_img, points_src, h_mat):
    """
    The mapping function maps the source image to the destination.
    :param src_img: source image.
    :param dst_img: the projection image.
    :param points_src: the P', Q', S', R' points.
    :param h_mat: the HC matrix H.
```

```
    :return: the projected image.
    """
    temp = np.zeros(dst_img.shape[0:2])
    pts = np.array([[points_src[0][0], points_src[0][1]], [points_src[1][0],
    points_src[1][1]], [points_src[3][0],
                        points_src[3][1]], [points_src[2][0], points_src[2][1]]])
    cv2.fillPoly(temp, [pts], 255)
    # plt.imshow(temp)
    # plt.show()
    # plt.clf()
    loc_hc = np.array([[np.dot(h_mat, np.array([j, i, 1])) for j in range(temp.shape
    [1])] for i in range(temp.shape[0])])
    loc = np.array([loc_hc[:, :, 0] / loc_hc[:, :, 2], loc_hc[:, :, 1] / loc_hc[:, :,
    2]])
    for i in range(temp.shape[0]):
        for j in range(temp.shape[1]):
            loc0 = loc[1, i, j]
            loc1 = loc[0, i, j]
            if (loc0 > 0) and (loc1 > 0) and (loc0 < src_img.shape[0]-1) and (loc1 <
    src_img.shape[1]-1):
                dst_img[i, j][temp[i, j] == 255] = compute_pixel_val(src_img, [loc0,
    loc1])

    return dst_img


# Image projection
h_ad = compute_hc(PQSR1, PQSR)
dst_img = mapping(img, img1, PQSR1, h_ad)
plt.imshow(dst_img)
plt.axis('off')
plt.savefig('dtoa.jpeg')
plt.clf()

h_bd = compute_hc(PQSR2, PQSR)
dst_img = mapping(img, img2, PQSR2, h_bd)
plt.imshow(dst_img)
plt.axis('off')
plt.savefig('dtob.jpeg')
plt.clf()

h_cd = compute_hc(PQSR3, PQSR)
dst_img = mapping(img, img3, PQSR3, h_cd)
plt.imshow(dst_img)
plt.axis('off')
plt.savefig('dtoc.jpeg')
plt.clf()
```

Task1Part1.py

5.2 Task1Part2.py

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from skimage import io

# read image
img1 = io.imread('painting1.jpeg')
img2 = io.imread('painting2.jpeg')
img3 = io.imread('painting3.jpeg')
img = io.imread('kittens.jpeg')

```

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


def compute_hc(src, dst):
    """
    The module computes the homography matrix by AH = b.
    :param src: source points.
    :param dst: mapped points.
    :return: homography matrix H.
    """
    A = np.zeros((8, 8))
    b = np.zeros((8, 1))

    for i in range(len(src)):
        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]]
        b[2 * i] = dst[i][0]
        b[2 * i + 1] = dst[i][1]

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

    return homo_mat.reshape((3, 3))


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

    loc0_f = np.int(np.floor(loc[0]))
    loc1_f = np.int(np.floor(loc[1]))
    loc0_c = np.int(np.ceil(loc[0]))
    loc1_c = np.int(np.ceil(loc[1]))
    a = img[loc0_f][loc1_f]
    b = img[loc0_f][loc1_c]
    c = img[loc0_c][loc1_f]
    d = img[loc0_c][loc1_c]
    dx = float(loc[0] - loc0_f)
    dy = float(loc[1] - loc1_f)
    Wa = 1 / np.linalg.norm([dx, dy])
    Wb = 1 / np.linalg.norm([1 - dx, dy])
    Wc = 1 / np.linalg.norm([dx, 1 - dy])
    Wd = 1 / np.linalg.norm([1 - dx, 1 - dy])
    output = np.divide((a * Wa + b * Wb + c * Wc + d * Wd), (Wa + Wb + Wc + Wd), where
    =(Wa + Wb + Wc + Wd)!=0)
    return output


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

```
    :param h_mat: the HC matrix H.
    :return: the projected image.
    """
    temp = np.zeros(dst_img.shape[0:2])
    pts = np.array([[points_src[0][0], points_src[0][1]], [points_src[1][0], points_src[1][1]], [points_src[3][0],
                    points_src[3][1]], [points_src[2][0], points_src[2][1]]])
    cv2.fillPoly(temp, [pts], 255)
    # plt.imshow(temp)
    # plt.show()
    # plt.clf()
    loc_hc = np.array([[np.dot(h_mat, np.array([j, i, 1])) for j in range(temp.shape[1])] for i in range(temp.shape[0])])
    loc = np.array([loc_hc[:, :, 0] / loc_hc[:, :, 2], loc_hc[:, :, 1] / loc_hc[:, :, 2]])
    for i in range(temp.shape[0]):
        for j in range(temp.shape[1]):
            loc0 = loc[1, i, j]
            loc1 = loc[0, i, j]
            if (loc0 > 0) and (loc1 > 0) and (loc0 < src_img.shape[0]-1) and (loc1 < src_img.shape[1]-1):
                dst_img[i, j][temp[i, j] == 255] = compute_pixel_val(src_img, [loc0, loc1])
            else:
                dst_img[i, j] = 0

    return dst_img


# Image projection
h_ac = compute_hc(PQSR2, PQSR1)
h_bc = compute_hc(PQSR3, PQSR2)
pts = np.array([[0, 0], [0, img3.shape[0]], [img3.shape[1], 0], [img3.shape[1], img3.shape[0]]])
dst_img = mapping(img1, img3, pts, np.dot(h_ac, h_bc))
plt.imshow(dst_img)
plt.axis('off')
plt.savefig('atoc.jpeg')
plt.clf()
```

Task1Part2.py

5.3 Task2Part1.py

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from skimage import io

# read image
img1 = io.imread('photoframe1.jpeg')
img2 = io.imread('photoframe2.jpeg')
img3 = io.imread('photoframe3.jpeg')
img = io.imread('santaclaus.jpeg')

# record the coordinates of points
PQSR1 = np.array([[1816, 1165], [1836, 1875], [2586, 1232], [2622, 1839]])
PQSR2 = np.array([[1450, 1060], [1380, 1864], [2702, 1042], [2776, 1866]])
PQSR3 = np.array([[1297, 1173], [1255, 1959], [2385, 1114], [2373, 2020]])
PQSR = np.array([[0, 0], [0, 1536], [2048, 0], [2048, 1536]])

# plot images with ROI
# #
```

```
    ==========================================================================================

# point = ["P", "Q", "S", "R"]
#
# plt.imshow(img1)
# for i in range(4):
#     plt.scatter(PQSR1[i][0], PQSR1[i][1])
#     plt.annotate(point[i] + '({},{})'.format(PQSR1[i][0], PQSR1[i][1]), (PQSR1[i
    ][0], PQSR1[i][1]), c='r')
# plt.title('photoframe1.jpeg')
# plt.axis('off')
# plt.savefig('task2_labelled1.jpeg')
# plt.clf()
#
# plt.imshow(img2)
# for i in range(4):
#     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')
# plt.title('photoframe2.jpeg')
# plt.axis('off')
# plt.savefig('task2_labelled2.jpeg')
# plt.clf()
# #
# plt.imshow(img3)
# for i in range(4):
#     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.title('photoframe3.jpeg')
# plt.axis('off')
# plt.savefig('task2_labelled3.jpeg')
# plt.clf()
#
# plt.imshow(img)
# for i in range(4):
#     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')
# plt.title('santaclaus.jpeg')
# plt.axis('off')
# plt.savefig('task2_labelled4.jpeg')
# plt.clf()


def compute_hc(src, dst):
    """
    The module computes the homography matrix by AH = b.
    :param src: source points.
    :param dst: mapped points.
    :return: homography matrix H.
    """

    A = np.zeros((8, 8))
    b = np.zeros((8, 1))

    for i in range(len(src)):
        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]]
        b[2 * i] = dst[i][0]
        b[2 * i + 1] = dst[i][1]

```

```python
    h = np.dot(np.linalg.pinv(A), b)
    homo_mat = np.append(h, 1)

    return homo_mat.reshape((3, 3))


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

    loc0_f = np.int(np.floor(loc[0]))
    loc1_f = np.int(np.floor(loc[1]))
    loc0_c = np.int(np.ceil(loc[0]))
    loc1_c = np.int(np.ceil(loc[1]))
    a = img[loc0_f][loc1_f]
    b = img[loc0_f][loc1_c]
    c = img[loc0_c][loc1_f]
    d = img[loc0_c][loc1_c]
    dx = float(loc[0] - loc0_f)
    dy = float(loc[1] - loc1_f)
    Wa = 1 / np.linalg.norm([dx, dy])
    Wb = 1 / np.linalg.norm([1 - dx, dy])
    Wc = 1 / np.linalg.norm([dx, 1 - dy])
    Wd = 1 / np.linalg.norm([1 - dx, 1 - dy])
    output = (a * Wa + b * Wb + c * Wc + d * Wd) / (Wa + Wb + Wc + Wd)

    return output


def mapping(src_img, dst_img, points_src, h_mat):
    """
    The mapping function maps the source image to the destination.
    :param src_img: source image.
    :param dst_img: the projection image.
    :param points_src: the P', Q', S', R' points.
    :param h_mat: the HC matrix H.
    :return: the projected image.
    """
    temp = np.zeros(dst_img.shape[0:2])
    pts = np.array([[points_src[0][0], points_src[0][1]], [points_src[1][0], points_src[1][1]], [points_src[3][0],
                    points_src[3][1]], [points_src[2][0], points_src[2][1]]])
    cv2.fillPoly(temp, [pts], 255)
    # plt.imshow(temp)
    # plt.show()
    # plt.clf()
    loc_hc = np.array([[np.dot(h_mat, np.array([j, i, 1])) for j in range(temp.shape[1])] for i in range(temp.shape[0])])
    loc = np.array([loc_hc[:, :, 0] / loc_hc[:, :, 2], loc_hc[:, :, 1] / loc_hc[:, :, 2]])
    for i in range(temp.shape[0]):
        for j in range(temp.shape[1]):
            loc0 = loc[1, i, j]
            loc1 = loc[0, i, j]
            if (loc0 > 0) and (loc1 > 0) and (loc0 < src_img.shape[0]-1) and (loc1 < src_img.shape[1]-1):
                dst_img[i, j][temp[i, j] == 255] = compute_pixel_val(src_img, [loc0, loc1])

    return dst_img
```

```


# image projection
h_ad = compute_hc(PQSR1, PQSR)
dst_img = mapping(img, img1, PQSR1, h_ad)
plt.imshow(dst_img)
plt.axis('off')
plt.savefig('task2_dtoa.jpeg')
plt.clf()

h_bd = compute_hc(PQSR2, PQSR)
dst_img = mapping(img, img2, PQSR2, h_bd)
plt.imshow(dst_img)
plt.axis('off')
plt.savefig('task2_dtob.jpeg')
plt.clf()

h_cd = compute_hc(PQSR3, PQSR)
dst_img = mapping(img, img3, PQSR3, h_cd)
plt.imshow(dst_img)
plt.axis('off')
plt.savefig('task2_dtoc.jpeg')
plt.clf()
```

Task2Part1.py

5.4 Task2Part2.py

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from skimage import io

# read image
img1 = io.imread('photoframe1.jpeg')
img2 = io.imread('photoframe2.jpeg')
img3 = io.imread('photoframe3.jpeg')
img = io.imread('santaclaus.jpeg')

# record the coordinates of points
PQSR1 = np.array([[1816, 1165], [1836, 1875], [2586, 1232], [2622, 1839]])
PQSR2 = np.array([[1450, 1060], [1380, 1864], [2702, 1042], [2776, 1866]])
PQSR3 = np.array([[1297, 1173], [1255, 1959], [2385, 1114], [2373, 2020]])
PQSR = np.array([[0, 0], [0, 1536], [2048, 0], [2048, 1536]])


def compute_hc(src, dst):
    """
    The module computes the homography matrix by AH = b.
    :param src: source points.
    :param dst: mapped points.
    :return: homography matrix H.
    """
    A = np.zeros((8, 8))
    b = np.zeros((8, 1))

    for i in range(len(src)):
        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]]
        b[2 * i] = dst[i][0]
        b[2 * i + 1] = dst[i][1]
```

```python
    h = np.dot(np.linalg.pinv(A), b)
    homo_mat = np.append(h, 1)

    return homo_mat.reshape((3, 3))


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

    loc0_f = np.int(np.floor(loc[0]))
    loc1_f = np.int(np.floor(loc[1]))
    loc0_c = np.int(np.ceil(loc[0]))
    loc1_c = np.int(np.ceil(loc[1]))
    a = img[loc0_f][loc1_f]
    b = img[loc0_f][loc1_c]
    c = img[loc0_c][loc1_f]
    d = img[loc0_c][loc1_c]
    dx = float(loc[0] - loc0_f)
    dy = float(loc[1] - loc1_f)
    Wa = 1 / np.linalg.norm([dx, dy])
    Wb = 1 / np.linalg.norm([1 - dx, dy])
    Wc = 1 / np.linalg.norm([dx, 1 - dy])
    Wd = 1 / np.linalg.norm([1 - dx, 1 - dy])
    output = np.divide((a * Wa + b * Wb + c * Wc + d * Wd), (Wa + Wb + Wc + Wd), where
    =(Wa + Wb + Wc + Wd)!=0)
    return output


def mapping(src_img, dst_img, points_src, h_mat):
    """
    The mapping function maps the source image to the destination.
    :param src_img: source image.
    :param dst_img: the projection image.
    :param points_src: the P', Q', S', R' points.
    :param h_mat: the HC matrix H.
    :return: the projected image.
    """
    temp = np.zeros(dst_img.shape[0:2])
    pts = np.array([[points_src[0][0], points_src[0][1]], [points_src[1][0],
    points_src[1][1]], [points_src[3][0],
                        points_src[3][1]], [points_src[2][0], points_src[2][1]]])
    cv2.fillPoly(temp, [pts], 255)
    # plt.imshow(temp)
    # plt.show()
    # plt.clf()
    loc_hc = np.array([[np.dot(h_mat, np.array([j, i, 1])) for j in range(temp.shape
    [1])] for i in range(temp.shape[0])])
    loc = np.array([loc_hc[:, :, 0] / loc_hc[:, :, 2], loc_hc[:, :, 1] / loc_hc[:, :,
    2]])
    print('yes')
    for i in range(temp.shape[0]):
        for j in range(temp.shape[1]):
            loc0 = loc[1, i, j]
            loc1 = loc[0, i, j]
            if (loc0 > 0) and (loc1 > 0) and (loc0 < src_img.shape[0]-1) and (loc1 <
    src_img.shape[1]-1):
                dst_img[i, j][temp[i, j] == 255] = compute_pixel_val(src_img, [loc0,
    loc1])
```

```
            else:
                dst_img[i, j] = 0

    return dst_img


# image projection
h_ac = compute_hc(PQSR2, PQSR1)
h_bc = compute_hc(PQSR3, PQSR2)
pts = np.array([[0, 0], [0, img3.shape[0]], [img3.shape[1], 0], [img3.shape[1], img3.
    shape[0]]])
dst_img = mapping(img1, img3, pts, np.dot(h_ac, h_bc))
plt.imshow(dst_img)
plt.axis('off')
plt.savefig('task2_atoc.jpeg')
plt.clf()
```

Task2Part2.py