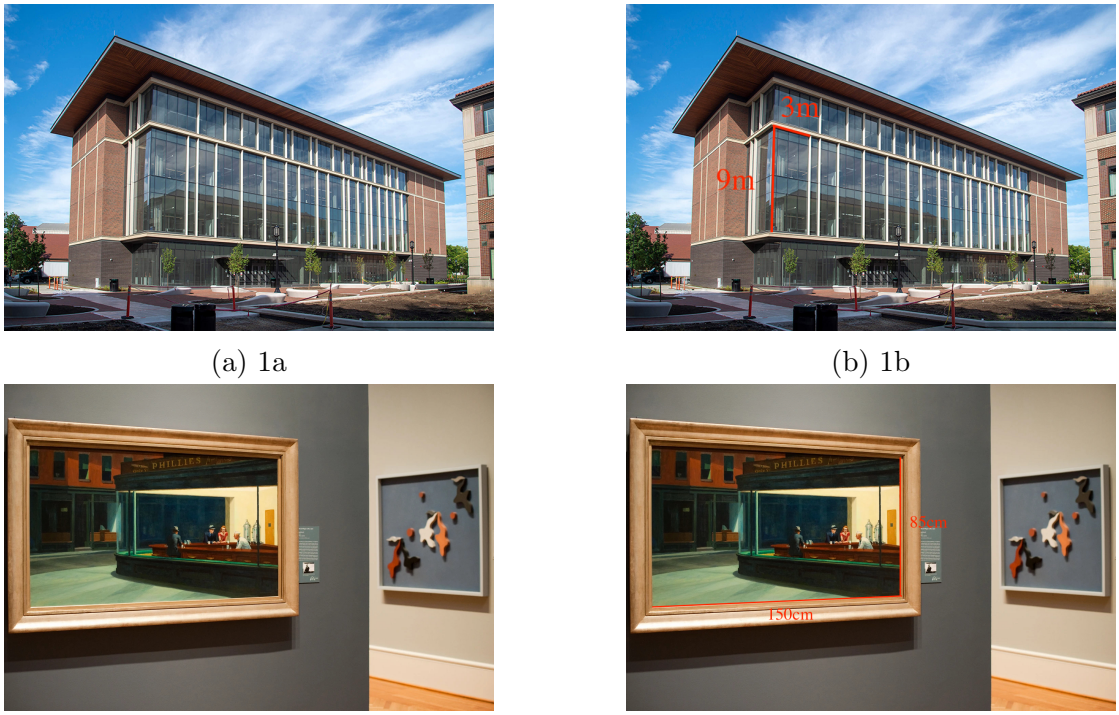
Task 1

Task 1.1: Point-to-Point Correspondences

Task : Show results after distortion corrections using point-to-point correspondences, this is a trivial extension of Homework 2. With the given height and width, your points in the undistorted image should be (0, 0), (0, width), (height, 0), and (height, width). After you have found the correspondences you simply need to apply the homography to the input image to remove distortion.

The images are given below:



(a) 1a (b) 1b

(c) 1c (d) 1d

Figure 1: Input Images

Background

Brief Summary of the process:

1. Get pixel coordinates of the 4 points on the Figs. 1a and 1c corresponding to the 4 corners whose dimensions are shown on Figs. 1b and 1d

2. Plot the coordinates on the images and create a box using the coordinates on each image.

3. Calculate the Homographies using the coordinates between the original undistorted scene as the domain space and its photograph as the range space which has projective and affine distortions.

4. Create the undistored blank image using the H and the distored image size.

5. Calculate H^{-1} to multiply it with pixel on the domain space (undistored image coordinate) to find the corresponding pixel in the range space (distored image coordinate).

6. Loop through each pixel on the range space image and copy the color of the corresponding pixel of the domain space pixel.

As explained in HW 2:

Getting the coordinates: I used an online tool (https://pixspy.com/). It allowed me to upload a picture, and get the coordinates of the pixels when I hover the mouse above them. The coordinates of the pixels for each Fig. 1a and 1c are given below:

	Fig. 1a	Fig. 1c
P(Upper left corner)	[240, 200]	[76, 180]
Q(Bottom left corner)	[236, 368]	[78, 653]
R(Bottom right corner)	[294, 374]	[805, 620]
S(Upper right corner)	[298, 216]	[802, 219]

Table 1: Coordinates for domain space

Given the measurements height and width in the real world, the undistorted image points are (0, 0), (0, width), (height, 0), and (height, width). The measurements are converted into cm for the purpose of consistency

	Fig. 1a	Fig. 1c
P′(Upper left corner)	[0, 0]	[0, 0]
Q′(Bottom left corner)	[0, 900]	[0, 85]
R′(Bottom right corner)	[300, 900]	[150, 85]
S′(Upper right corner)	[300, 0]	[150, 0]

Table 2: Coordinates for range space

Plot the coordinates: For the domain space image, the coordinates of PQRS were first used to draw circles at the PQRS points. Then a box was drawn by connecting the points with a line using the OpenCV tool 'polylines'. This allowed me to see if the point coordinates that I got from the online tool were correct or not. The resulting images with the box around the PQRS region is shown in the Output section below as Fig. 2a and 2b.

Calculating Homographies: To calculate the Homography between a point x in the domain space and a point x' in the range space, the following matrix equation was simplified and solved simultaneously:

$$x' = \mathrm{H}x$$

$$\begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$x'_1 = h_{11}x_1 + h_{12}x_2 + h_{13}x_3$$

$$x'_2 = h_{21}x_1 + h_{22}x_2 + h_{23}x_3$$

$$x'_3 = h_{31}x_1 + h_{32}x_2 + x_3$$

Dividing the first two equations by the third equation

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

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

Since we have 8 unknowns and each given pair of coordinates give 2 equations, 4 pairs of coordinates are used to form 8 equations and this is simplified into this matrix form:

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

Showing this as $Ax = b$, x can be calculated as $x = A^{-1}b$.

Creating the undistored blank image: After the H is calculated between the 4 pairs of pixels in the distored image and the undistored world space, this H is used to create the size of the undistored image space from the distored image size.

Output

The 4 points that are used to calculate the homography are drawn on the images shown below:

(a) 2a

(b) 2b

Figure 2: Box Images

The unidstored images after the distortion was removed by point to point correspondences are shown below:

(a) 3a

(b) 3b

Figure 3: Output Images for point to point correspondence method

Task 1.2: Two-Step and One-Step Methods

Task: Show results after distortion correction using the two-step and one-step approaches.

Two-Step Method

Background

Brief summary of the process:

1. Find the Vanishing Points by using coordinates of points on two sets of parallel lines in the real world scene. Find the Vanishing Line using the Vanishing Points.
2. Estimate the Homography that takes the Vanishing Line back to Line at Infinity.
3. Apply this homography on the Image to get the real world scene which now does not have projective distortion.
4. For removing affine distortion, first two sets of orthogonal lines in real world scene are calculated.
5. Using these lines, a homography is estimated that takes the angles back to 90°
6. Apply this homography to the whole image to get the corrected real world scene which now does not have affine distortion.

Finding Vanishing points and Vanishing line: The point coordinates that are used to find the Vanishing Points are shown below:

	Building	Fig. Nighthawks
P′(Upper left corner)	[242, 130]	[76, 180]
Q′(Bottom left corner)	[657, 273]	[78, 653]
R′(Bottom right corner)	[660, 403]	[805, 620]
S′(Upper right corner)	[236, 369]	[802, 219]

Table 3: Coordinates for finding Vanishing Points

Suppose the points are p1, p2, p3, p4.
line 1 = $p1 \times p2$
line 2 = $p3 \times p4$
VP_1 = line 1 × line 2

line 3 = $p1 \times p4$
line 4 = $p2 \times p3$
VP_2 = line 3 × line 4
VL = $VP_1 \times VP_2$

Estimating the homography that takes the Vanishing Line back to L_∞: Using the VL calculated in the previous step, H can be estimated. If VL $= \begin{bmatrix} l_1 & l_2 & l_3 \end{bmatrix}^T$, then H that takes the Vanishing Line back to Line at Infinity has the form:

$$H_p = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & l_3 \end{bmatrix}$$

Once the projective distortion is removed, the new coordinates of the chosen points are calculated by multiplying H_p with the coordinates. Using these new coordinates, affine matrix is estimated.

Estimating the homography that removes Affine distortion: The angle between two lines $L = \begin{bmatrix} l_1 & l_2 & l_3 \end{bmatrix}^T$ and $M = \begin{bmatrix} m_1 & m_2 & m_3 \end{bmatrix}^T$ can be calculated using the following equation:

$$Cos\theta = \frac{L^T C_\infty^* M}{\sqrt{(L^T C_\infty^* L)(M^T C_\infty^* M)}}$$

Using the relation $L = H^T L'$ and $M = H^T M'$, and the fact that lines will be orthogonal in the world image, we get:

$$0 = L' H C_\infty^* H^T M'$$

where $C_\infty^* = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ and $H = \begin{bmatrix} A & 0 \\ 0 & 1 \end{bmatrix}$

Expanding the above equation gives:

$$\begin{bmatrix} l_1' & l_2' & l_3' \end{bmatrix} \begin{bmatrix} AA^T & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} m_1' \\ m_2' \\ m_3' \end{bmatrix}$$

Let $S = AA^T = \begin{bmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{bmatrix}$

Expanding it further:

$$s_{11} m_1' l_1' + s_1 2(l_1' m_2' + l_2' m_1') + s_{22} l_2' m2' = 0$$

By setting $s_{22} = 1$ and computing the SVD for S, we can compute A by following:

$$S = AA^T = VD^2V^T, A = VDV^T$$

Using this A, H_a can be formed and used to remove affine distortion. To remove the projective and affine from the image, H_a^{-1} is multiplied with H_p and the new H is applied to the image. The results of this process are shown in the Output section below.

Output

(a) 4a: Points and lines used for 2-step method

(b) 4b: After projective distortion was removed

(c) 4c: After projective and affine distortion was removed

Figure 4: Output Images for two-step method

(a) 5a: Points and lines used for 2-step method

(b) 5b: After projective distortion was removed

(c) 5c: After projective and affine distortion was removed

Figure 5: Output Images for 2-step method

One-Step Method

Background

Brief summary of the process:

1. Get coordinates for points to calculate 6 sets of orthogonal lines.
2. Using the 6 set of lines, estimate a homography H that can remove projective and affine distortion in one go.
3. Apply the H^{-1} to the image to get the real world scene.

Calculating 6 sets of lines: The point coordinates that are used to calculate the lines are shown below:

	Building	Nighthawks
P1	[346, 227]	[76, 180]
Q1	[345,378]	[78, 653]
R1	[460,387]	[805, 620]
S1	[460,256]	[802, 219]
P2	[240,200]	[12,100]
Q2	[236,367]	[14,729]
R2	[295,373]	[865,678]
S2	[298,216]	[863,162]

Table 4: Coordinates for lines for one-step method

Set 1: line 1 = P1 × Q1, line 2 = P1 × S1
Set 2: line 3 = P1 × S1, line 4 = S1 × R1
Set 3: line 5 = S1 × R1, line 6 = R1 × Q1
Set 4: line 7 = R1 × Q1, line 8 = Q1 × P1
Set 5: line 9 = P2 × S2, line 10 = S2 × R2
Set 6: line 11 = S2 × R2, line 12 = R2 × Q2

Estimating Homography to remove Projective and Affine distortion:
A general Homography has the form:

$$\mathrm{H} = \begin{bmatrix} A & 0 \\ v^T & 1 \end{bmatrix}$$

Using the dual conic function:

$$C_\infty^{*\prime} = \mathrm{H} C_\infty^* \mathrm{H}^T$$

where $C_\infty^* = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

$$C_\infty^{*'} = \begin{bmatrix} AA^T & Av \\ v^T A^T & v^T v \end{bmatrix}$$

$$C_\infty^{*'} = \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix}$$

Let L and M be the orthogonal lines in real world space, we have the following:

$$L^{T'} C_\infty^{*'} M' = 0$$

$$\begin{bmatrix} l_1' & l_2' & l_3' \end{bmatrix} \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix} \begin{bmatrix} m_1' \\ m_2' \\ m_3' \end{bmatrix} = 0$$

Setting $f = 1$, and using the 5 sets of orthogonal lines, 5 equations can be solved simultaneously to get the 5 variables. After normalizing it, apply SVD on S matrix to calculate matrix A.

$$S = AA^T = \begin{bmatrix} a & b/2 \\ b/2 & c \end{bmatrix}$$

$$S = AA^T = VD^2V^T, A = VDV^T$$

And then get v using the relation $v = A^{-1} \begin{bmatrix} d/2 & e/2 \end{bmatrix}^T$.

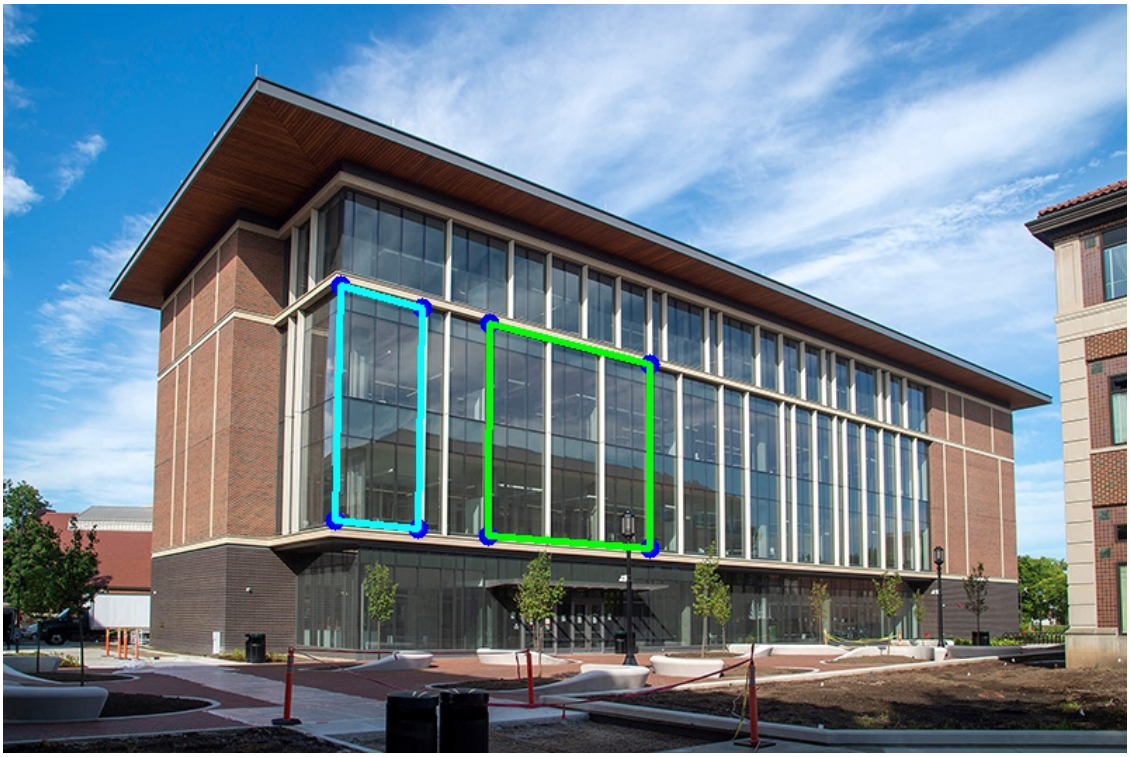
Observations

1. Point to point correspondence gave the best result that shows the least amount of distortion in the final image.

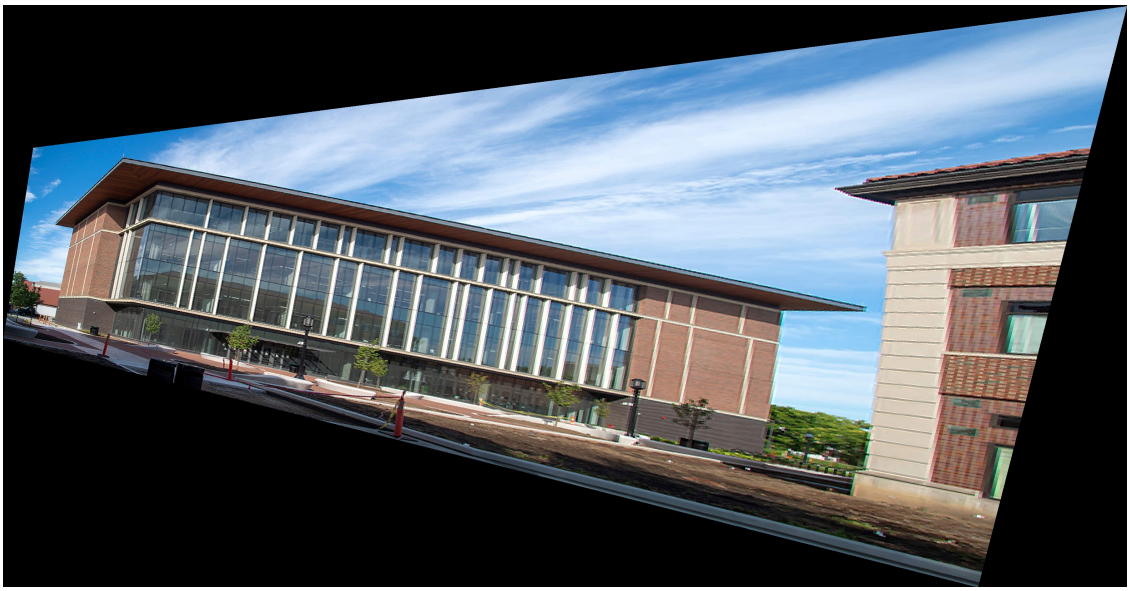
2. The two-step method final output was clear of distortion but was squeezed vertically.

3. Having one more set of orthogonal lines in one-step method improved the results drastically.

Output



(a) 6a: Points and lines used for 1-step method



(b) 6b: After distortion was removed

Figure 6: Output Images for one-step method

(a) 7a: Points and lines used for 1-step method

(b) 7b: After distortion was removed

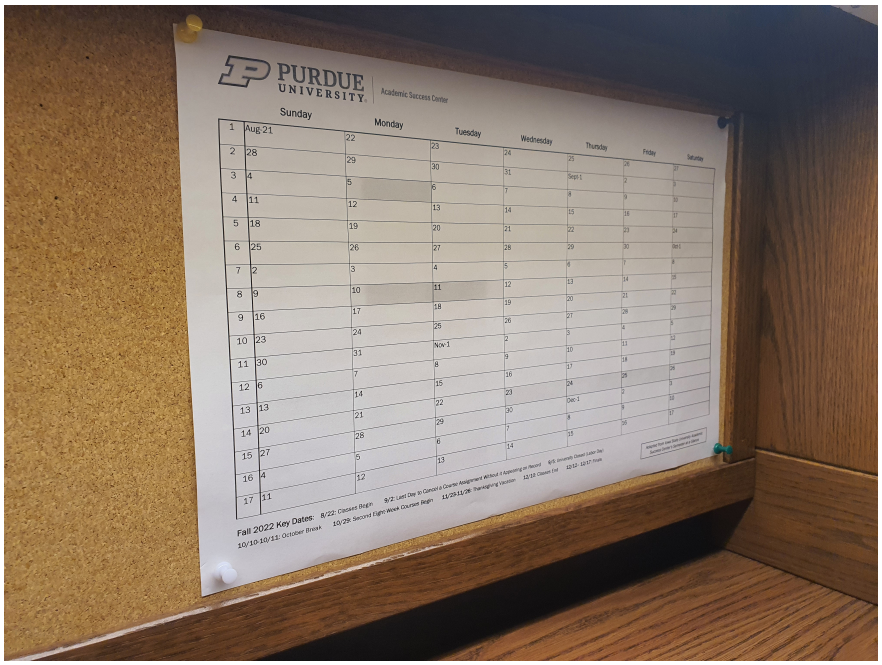
Figure 7: Output Images for one-step method

Task 2

Repeat the steps, outlined in Task 1, on at least two of your own images.

The input images are shown below:

(a) 8a: Input image 1



(b) 8b: Input image 2

Figure 8: Input images for task 2

Background

Coordinates:

Paintings		
	image	real world
P1	[537,465]	[0, 0]
Q1	[538,947]	[0, 350]
R1	[940,842]	[400, 350]
S1	[933,485]	[400, 0]

Calendar		
	image	real world
P1	[459,222]	[0, 0]
Q1	[491,973]	[0, 340]
R1	[672,933]	[100, 340]
S1	[652,241]	[100, 0]

Table 5: Coordinates for task 2.1 and 2.2

	Paintings	Calendar
P1	[537,465]	[459,222]
Q1	[538,947]	[491,973]
R1	[940,842]	[672,933]
S1	[933,485]	[652,241]
P2	[240,200]	[412,266]
Q2	[236,367]	[420,451]
R2	[295,373]	[819,454]
S2	[298,216]	[816,298]

Table 6: Coordinates for task 2.3

Observations

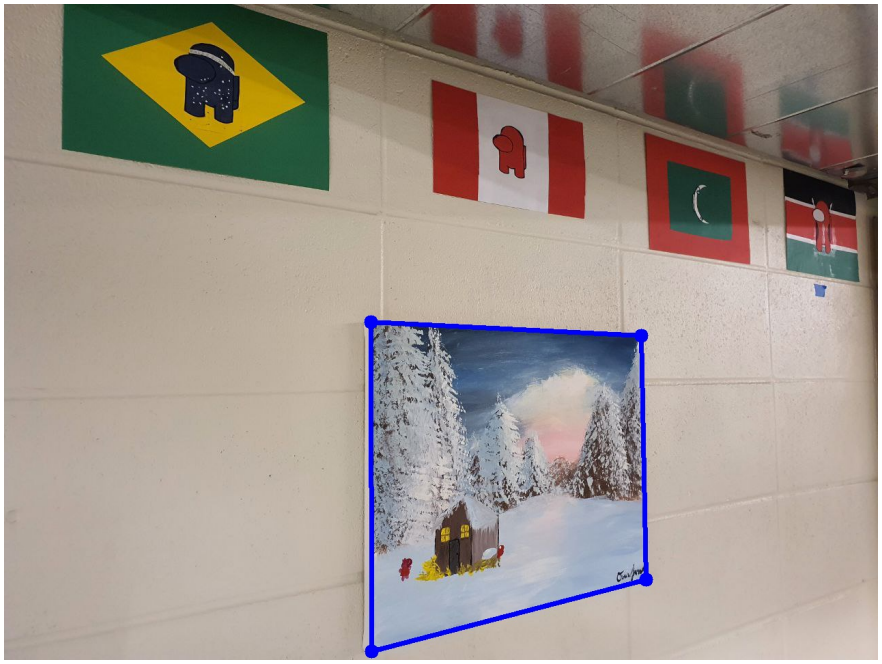
1. Point to point correspondence gave the best result that shows the least amount of distortion in the final image.

2. The two-step method final output was clear of distortion but was squeezed vertically.

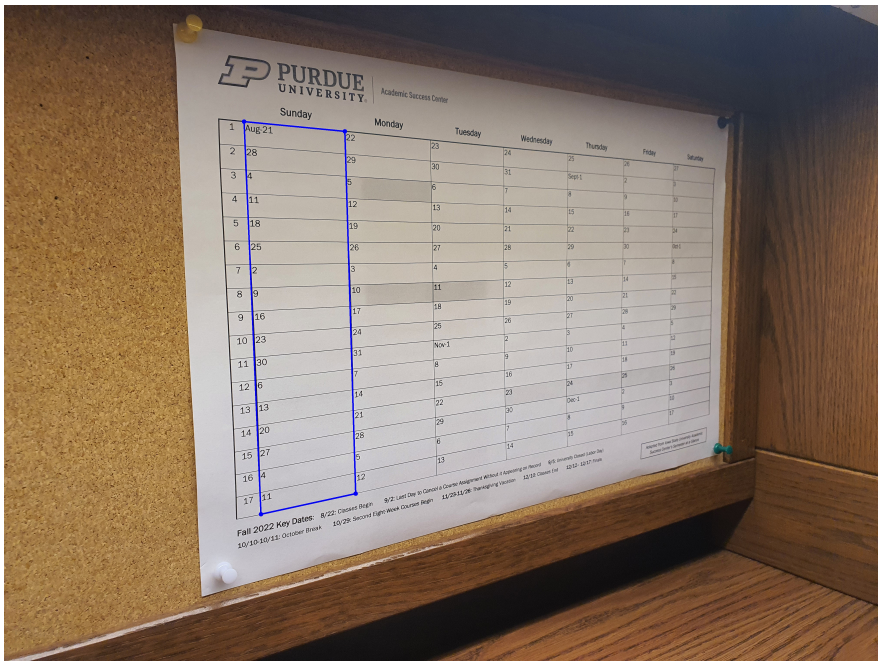
3. Having one more set of orthogonal lines in one-step method improved the results drastically, but some distortion can still be seen with the new images.

4. For one-step method, the code took the longest time to run compared to the other two methods. If the resolution of the original image was big, it would take up to to 15 minutes to generate an output.

Output



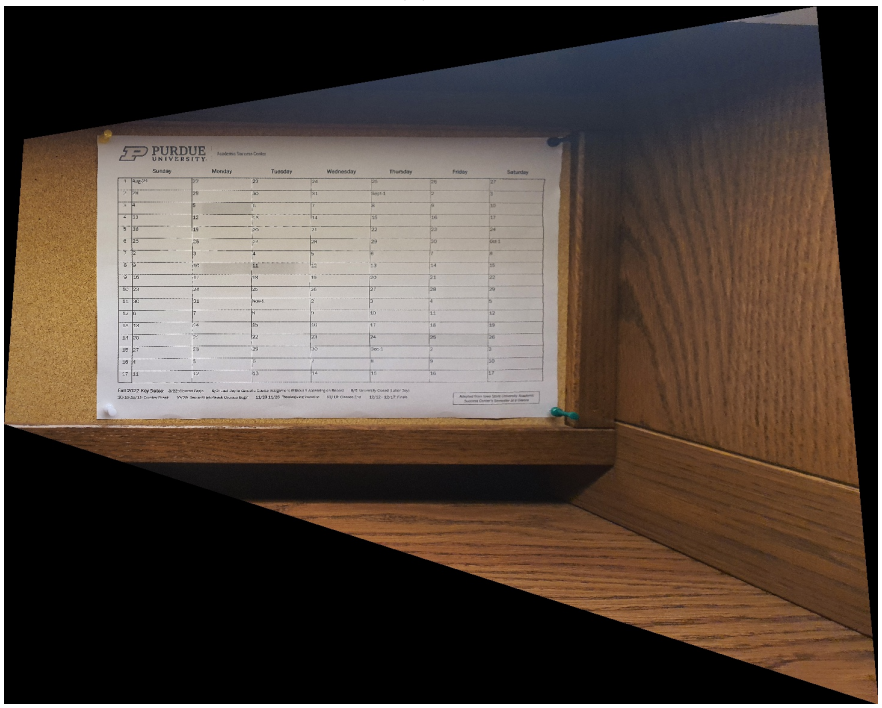
(a) 9a



(b) 9b

Figure 9: Box images of points used for point to point correspondence method

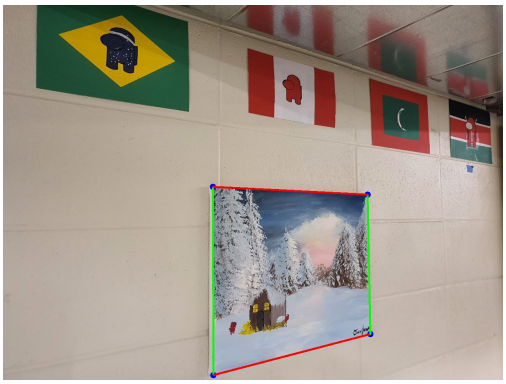
(a) 10a



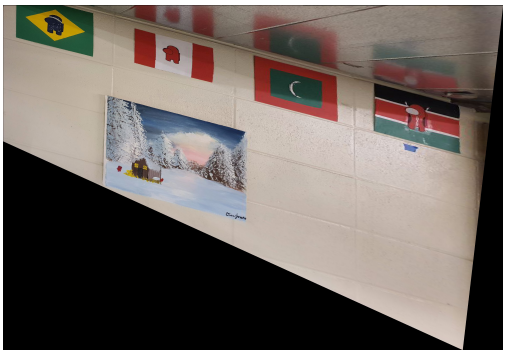
(b) 10b

Figure 10: Output images for point to point correspondence method

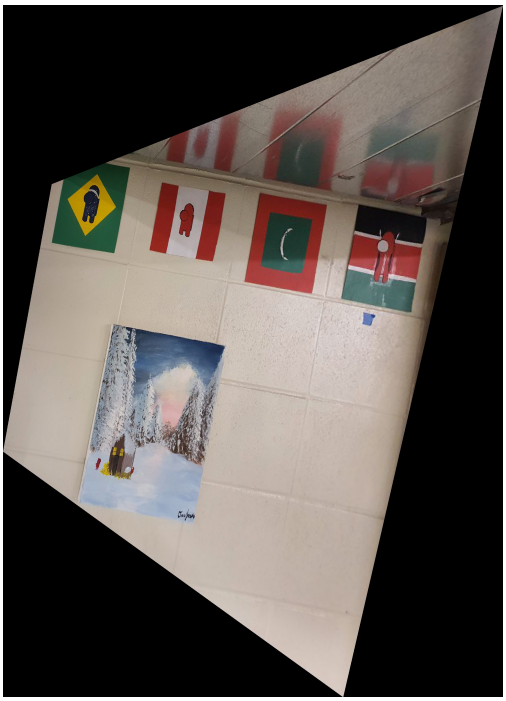
The outputs for task 2.2 two-step method are below:



(a) 11a: Points and lines used for 2-step method

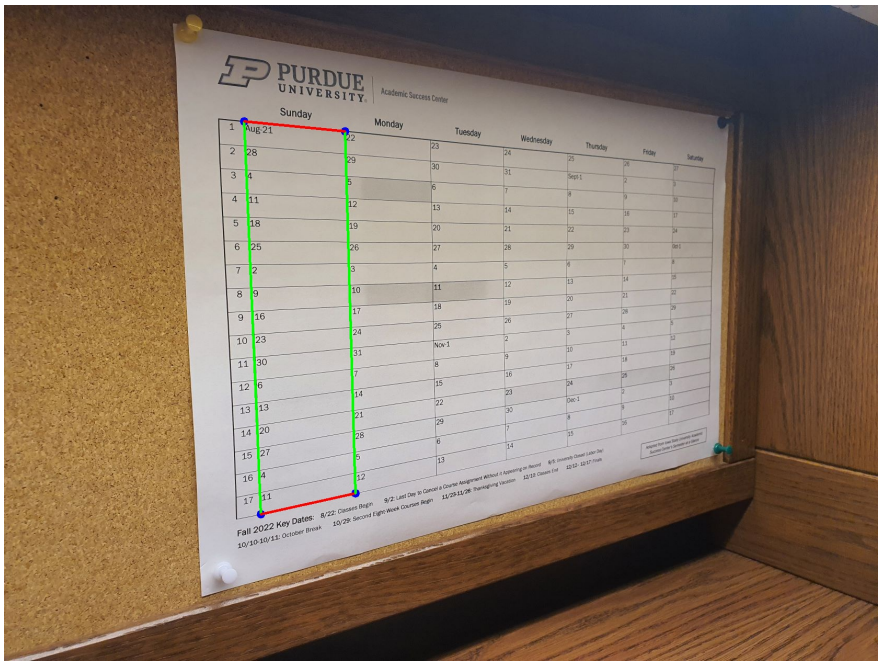


(b) 11b: After projective distortion was removed

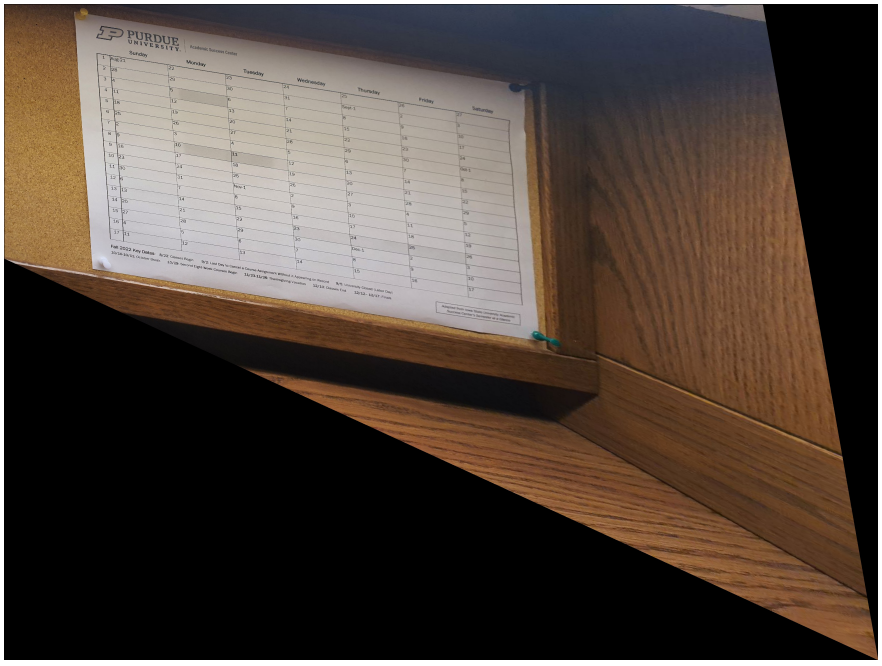


(c) 11c: After projective and affine distortion was removed

Figure 11: Output Images for two-step method



(a) 12a: Points and lines used for 2-step method

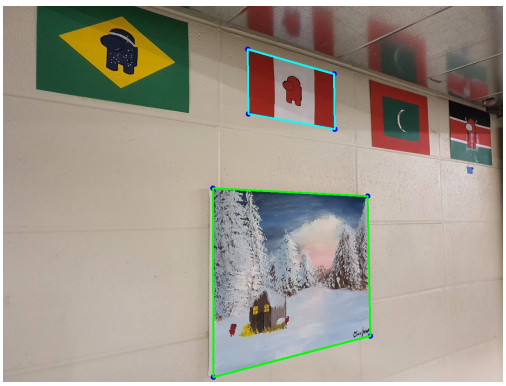


(b) 12b: After projective distortion was removed

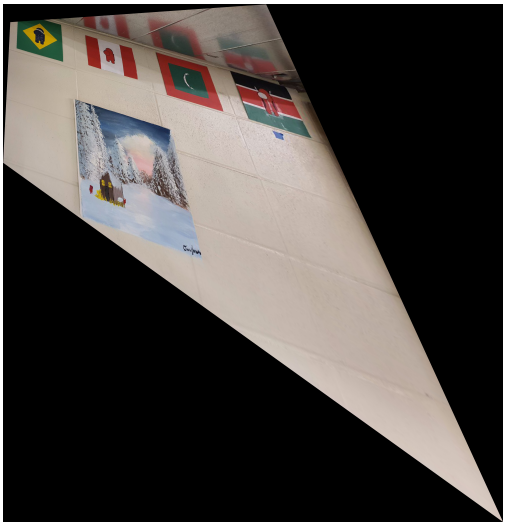
(c) 12c: After projective and affine distortion was removed

Figure 12: Output Images for 2-step method

The outputs for task 2.3 one-step method are below:

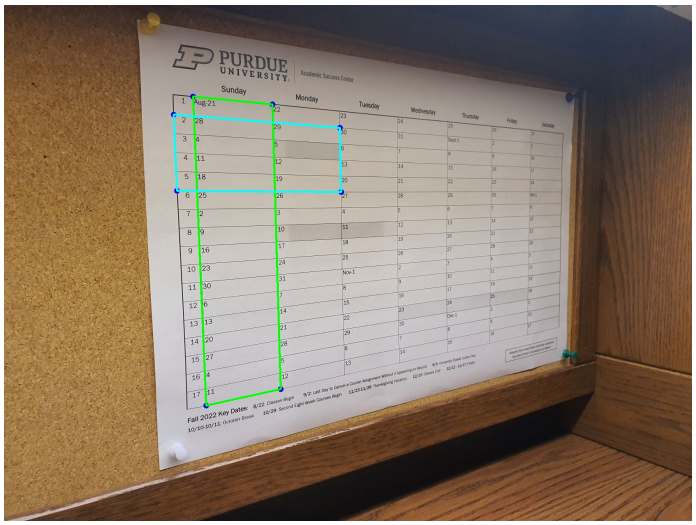


(a) 13a: Points and lines used for 1-step method



(b) 13b: After projective and affine distortion was removed

Figure 13: Output Images for one-step method



(a) 14a: Points and lines used for 2-step method

(b) 14b: After projective and affine distortion was removed

Figure 14: Output Images for 1-step method

Code

Task 1.1 and 2.1

```
import numpy as np
import cv2

#-------------------------Functions-------------------------------#
def drawCircleNBox(img, coords):
    img_box = img.copy()
    #draw circles at the coords
    for x,y in coords:
        cv2.circle(img_box, (x,y), 10, (255, 0, 0), -1)

    #draw lines connecting the coords to form a box
    coords.append(coords[0])
    cv2.polylines(img_box, [np.array(coords)], True, (255, 0, 0), 5)

    #return Img
    return img_box

def calcHomography(dom_coords, range_coords):
    ## AH = b

    # A and B Matrix
    A = np.zeros((8,8))
    b = np.zeros((8,1))
    for i in range(4):
        xp = range_coords[i][0]
        yp = range_coords[i][1]
        x = dom_coords[i][0]
        y = dom_coords[i][1]
        # A matrix
        A[2*i][0:8] = [x, y, 1 , 0 , 0 , 0 ,-x*xp, -y*xp]
        A[2*i+1][0:8] = [0, 0, 0, x, y, 1, -x*yp, -y*yp]
        # B matrix
        b[2*i] = xp
        b[2*i+1] = yp

    # H matrix
    H = np.zeros((3,3))
    H = np.matmul(np.linalg.pinv(A),b)
    H = np.append(H,[1])
    H = np.reshape(H, (3,3))
    return H

def getRectImg(dom_img_coords, range_img_coords, range_img):
    # Draw circles and bounding box on card
    boxImg = drawCircleNBox(range_img, dom_img_coords)

    # Generate a Homography Matrix
    H = calcHomography(dom_img_coords, range_img_coords)
```

```
    # Map range space (image) points to domain space (real world) points
                                            using H_inv
    img_rect = rectImage(range_img, H)

    return boxImg, img_rect

def rectImage(range_img, H):
    # Inverse of H
    H_inv = np.linalg.pinv(H)

    # Get world coordinates
    world_coords = []
    img_coords = [(0, 0), (0, range_img.shape[0]-1), (range_img.shape[1]-1
                                            , range_img.shape[0]-1), (
                                            range_img.shape[1]-1, 0)]
    for i in img_coords:
        img_coord = np.array((i[0], i[1], 1))
        world_coord = np.matmul(H, img_coord)
        world_coord = world_coord/world_coord[2]
        world_coords.append([world_coord[0],world_coord[1]])
    # Get size of world image
    x_max = int(max([c[0] for c in world_coords]))
    x_min = int(min([c[0] for c in world_coords]))
    y_max = int(max([c[1] for c in world_coords]))
    y_min = int(min([c[1] for c in world_coords]))
    x_length = x_max - x_min
    y_length = y_max - y_min

    world_img = np.zeros((y_length, x_length, 3))
    for row in range(0, y_length):
        for col in range(0, x_length):
            domain_coord = np.array((col+x_min, row+y_min, 1))
            range_coord = np.matmul(H_inv,domain_coord)
            range_coord = range_coord / range_coord[2]
            range_coord = range_coord.astype(int)
            #print(domain_coord,range_coord)
            if (range_coord[0]<(range_img.shape[1]-1) and range_coord[1]<(
                                                range_img.shape[0]-1) and
                                                 range_coord[0]>0 and
                                                range_coord[1]>0 ):
                world_img[row, col] = range_img[range_coord[1],
                                                    range_coord[0]]
    return world_img
# ----------------------------Task 1-----------------------------#

# Image Coordinates
building_img_coords = [(240,200), (236, 368), (294, 374), (298,214)]
building_irl_coords = [(0, 0), (0, 450),(150, 450), (150, 0)]
nighthawks_img_coords = [(76, 180), (78, 653), (805, 620), (802, 219)]
nighthawks_irl_coords = [(0, 0), (0, 85), (150, 85), (150, 0)]

paintings_img_coords = [(537,465),(538,947),(940,842),(933,485)]
paintings_irl_coords = [(0,0),(0,350),(400,350),(400,0)]
calendar_img_coords = [(459,222),(491,973),(672,933),(652,241)]
```

```
calendar_irl_coords = [(0,0),(0,340),(100,340),(100,0)]

# Input Images
building_img = cv2.imread('HW3/building.jpg')
nighthawks_img = cv2.imread('HW3/nighthawks.jpg')

paintings_img = cv2.imread('HW3/paintings.jpg')
calendar_img = cv2.imread('HW3/calendar.jpg')

#----------------------------Task1.1: p-t-p correspondence
                                    ------------------------------#
building_boxImg, building_imgRect = getRectImg(building_img_coords,
                                    building_irl_coords, building_img)
nighthawks_boxImg, nighthawks_imgRect = getRectImg(nighthawks_img_coords,
                                    nighthawks_irl_coords, nighthawks_img
                                    )

paintings_boxImg, paintings_imgRect = getRectImg(paintings_img_coords,
                                    paintings_irl_coords, paintings_img)
calendar_boxImg, calendar_imgRect = getRectImg(calendar_img_coords,
                                    calendar_irl_coords, calendar_img)


# Show and save Images
cv2.imwrite('HW3/building_boxImg.jpeg',building_boxImg)
cv2.imwrite('HW3/building_imgRect.jpeg',building_imgRect)
cv2.imwrite('HW3/nighthawks_boxImg.jpeg',nighthawks_boxImg)
cv2.imwrite('HW3/nighthawks_imgRect.jpeg',nighthawks_imgRect)
cv2.imwrite('HW3/paintings_boxImg.jpeg',paintings_boxImg)
cv2.imwrite('HW3/paintings_imgRect.jpeg',paintings_imgRect)
cv2.imwrite('HW3/calendar_boxImg.jpeg',calendar_boxImg)
cv2.imwrite('HW3/calendar_imgRect.jpeg',calendar_imgRect)
```

Task 1.2 and 2.2

```
import numpy as np
import cv2

#-------------------------Functions-------------------------------#
def drawCircleNBox(img, coords):
    img_box = img.copy()
    #draw circles at the coords
    for x,y in coords:
        cv2.circle(img_box, (x,y), 8, (255, 0, 0), -1)

    #draw lines connecting the coords to form a box
    coords.append(coords[0])
    for i in range(4):
        if i == 0 or i==2 or i==4:
            cv2.line(img_box, coords[i], coords[i+1], (0, 255, 0), 4)
        else:
            cv2.line(img_box, coords[i], coords[i+1], (0, 0, 255), 4)
```

```
    #return Img
    return img_box

def getProjectiveH(coords):
    p1 = (coords[0][0],coords[0][1],1)
    p2 = (coords[1][0],coords[1][1],1)
    p3 = (coords[2][0],coords[2][1],1)
    p4 = (coords[3][0],coords[3][1],1)

    l1 = np.cross(p1,p2)
    l2 = np.cross(p3,p4)
    vp_1 = np.cross(l1,l2)
    vp_1 = vp_1/vp_1[2]

    l3 = np.cross(p1,p4)
    l4 = np.cross(p2,p3)
    vp_2 = np.cross(l3,l4)
    vp_2 = vp_2/vp_2[2]

    vl = np.cross(vp_2, vp_1)
    vl = vl/vl[2]

    H = np.zeros((3,3))
    H[0,0] = 1
    H[1,1] = 1
    H[2] = vl

    return H

def getAffineH(coords):
    p1 = (coords[0][0],coords[0][1],1)
    p2 = (coords[1][0],coords[1][1],1)
    p3 = (coords[2][0],coords[2][1],1)
    p4 = (coords[3][0],coords[3][1],1)

    # Get parallel lines
    l1 = np.cross(p1,p2)
    m1 = np.cross(p1,p4)
    l1 = l1/l1[2]
    m1 = m1/m1[2]

    l2 = np.cross(p2,p3)
    m2 = np.cross(p3,p4)
    l2 = l2/l2[2]
    m2 = m2/m2[2]

    A = np.zeros((2,2))
    b = np.zeros((2,1))
    A = [[l1[0]*m1[0], l1[0]*m1[1]+l1[1]*m1[0]],[l2[0]*m2[0], l2[0]*m2[1]+
                                                 l2[1]*m2[0]]]
    b = [[-l1[1]*m1[1]],[-l2[1]*m2[1]]]

    s = np.matmul(np.linalg.pinv(A),b)
```

```
    S = np.zeros((2,2))
    S[0,0] = s[0]
    S[0,1] = s[1]
    S[1,0] = s[1]
    S[1,1] = 1
    V,D,V_h = np.linalg.svd(S)
    D = np.sqrt(np.diag(D))
    A_mat = np.matmul(np.matmul(V,D),V.transpose())
    H = np.zeros((3,3))
    H[0,0] = A_mat[0,0]
    H[0,1] = A_mat[0,1]
    H[1,0] = A_mat[1,0]
    H[1,1] = A_mat[1,1]
    H[2,2] = 1
    return H

def rectImage(range_img, H):

    # Inverse of H
    H_inv = np.linalg.pinv(H)
    H_inv = H_inv/H_inv[2,2]
    # Get world coordinates
    world_coords = []
    img_coords = [(0, 0), (0, range_img.shape[0]-1), (range_img.shape[1]-1
                                        , range_img.shape[0]-1), (
                                        range_img.shape[1]-1, 0)]
    for i in img_coords:
        img_coord = np.array((i[0], i[1], 1))
        world_coord = np.matmul(H, img_coord)
        world_coord = world_coord/world_coord[2]
        world_coords.append([world_coord[0],world_coord[1]])
    # Get size of world image
    x_max = int(max([c[0] for c in world_coords]))
    x_min = int(min([c[0] for c in world_coords]))
    y_max = int(max([c[1] for c in world_coords]))
    y_min = int(min([c[1] for c in world_coords]))
    x_length = x_max - x_min
    y_length = y_max - y_min
    print(y_length, x_length)
    world_img = np.zeros((y_length, x_length, 3))
    for row in range(0, y_length):
        for col in range(0, x_length):
            domain_coord = np.array((col+x_min, row+y_min, 1))
            range_coord = np.matmul(H_inv,domain_coord)
            range_coord = range_coord / range_coord[2]
            range_coord = range_coord.astype(int)
            if (range_coord[0]<(range_img.shape[1]-1) and range_coord[1]<(
                                            range_img.shape[0]-1) and
                                             range_coord[0]>0 and
                                            range_coord[1]>0 ):
                world_img[row, col] = range_img[range_coord[1],
                                                range_coord[0]]
    return world_img
```

```
def getRectImg(dom_img_coords, range_img):
    # Draw circles and bounding box on card
    boxImg = drawCircleNBox(range_img, dom_img_coords)

    # Removing Projective Distortion
    H_proj = getProjectiveH(dom_img_coords)
    project_img_rect = rectImage(range_img, H_proj)

    dom_coords_affine = []
    for c in dom_img_coords:
        coord = np.array((c[0], c[1], 1))
        coords = np.matmul(H_proj, coord)
        coords = coords/coords[2]
        coords = coords.astype(int)
        dom_coords_affine.append([coords[0], coords[1]])
    print("newcoords:", dom_coords_affine)

    # Removing Affine Distortion
    H_affine = getAffineH(dom_coords_affine)

    H_projAff = np.matmul(np.linalg.pinv(H_affine),H_proj)
    # Removing both distortion
    img_rect = rectImage(range_img, H_projAff)

    return boxImg, project_img_rect, img_rect

#----------------------------------Task---------------------------------#
# Input Images
building_img = cv2.imread('HW3/building.jpg')
nighthawks_img = cv2.imread('HW3/nighthawks.jpg')

building_pqrs = [(242, 130), (657, 273), (660, 403), (236, 369)]
nighthawks_pqrs = [(76, 180), (78, 653), (805, 620), (802, 219)]

paintings_img = cv2.imread('HW3/paintings.jpg')
calendar_img = cv2.imread('HW3/calendar.jpg')

paintings_pqrs = [(537,465),(538,947),(940,842),(933,485)]
calendar_pqrs = [(459,222),(491,973),(672,933),(652,241)]

building_boxImg, building_img_ProjRect, building_img_Rect  = getRectImg(
                                    building_pqrs, building_img)
nighthawks_boxImg, nighthawks_img_ProjRect, nighthawks_img_Rect  =
                                    getRectImg(nighthawks_pqrs,
                                    nighthawks_img)
paintings_boxImg, paintings_img_ProjRect, paintings_img_Rect  = getRectImg
                                    (paintings_pqrs, paintings_img)
calendar_boxImg, calendar_img_ProjRect, calendar_img_Rect  = getRectImg(
                                    calendar_pqrs, calendar_img)


cv2.imwrite('HW3/building_box_img1.jpeg',building_boxImg)
cv2.imwrite('HW3/building_ProjectRect.jpeg',building_img_ProjRect)
cv2.imwrite('HW3/building_Rect1.2.jpeg',building_img_Rect)
```

```
cv2.imwrite('HW3/nighthawks_boxImg1.jpeg',nighthawks_boxImg)
cv2.imwrite('HW3/nighthawks_ProjRect.jpeg',nighthawks_img_ProjRect)
cv2.imwrite('HW3/nighthawks_Rect1.2.jpeg',nighthawks_img_Rect)

cv2.imwrite('HW3/paintings_boxImg1.jpeg',paintings_boxImg)
cv2.imwrite('HW3/paintings_img_ProjRect.jpeg',paintings_img_ProjRect)
cv2.imwrite('HW3/paintings_img_Rect1.2.jpeg',paintings_img_Rect)
cv2.imwrite('HW3/calendar_boxImg1.jpeg',calendar_boxImg)
cv2.imwrite('HW3/calendar_img_ProjRect.jpeg',calendar_img_ProjRect)
cv2.imwrite('HW3/calendar_img_Rect1.2.jpeg',calendar_img_Rect)
```

Task 1.3 and 2.3

```
import numpy as np
import cv2

#-------------------------Functions-------------------------------#
def drawCircleNBox(img, coords, coords2):
    img_box = img.copy()
    #draw circles at the coords
    for x,y in coords:
        cv2.circle(img_box, (x,y), 7, (255, 0, 0), -1)
    for x,y in coords2:
        cv2.circle(img_box, (x,y), 7, (255, 0, 0), -1)

    #draw lines connecting the coords to form a box
    coords.append(coords[0])
    for i in range(4):
        cv2.line(img_box, coords[i], coords[i+1], (0, 255, 0), 4)

    coords2.append(coords2[0])
    for i in range(4):
        cv2.line(img_box, coords2[i], coords2[i+1], (255, 255, 0), 4)

    #return Img
    return img_box

def getH(coords, coords2):
    p1 = (coords[0][0],coords[0][1],1)
    q1 = (coords[1][0],coords[1][1],1)
    r1 = (coords[2][0],coords[2][1],1)
    s1 = (coords[3][0],coords[3][1],1)

    p2 = (coords2[0][0],coords2[0][1],1)
    q2 = (coords2[1][0],coords2[1][1],1)
    r2 = (coords2[2][0],coords2[2][1],1)
    s2 = (coords2[3][0],coords2[3][1],1)

    l1 = np.cross(p1,q1)
    l2 = np.cross(p1,s1)
    l1 = l1/l1[2]
    l2 = l2/l2[2]
```

```
l3 = np.cross(p1,s1)
l4 = np.cross(s1,r1)
l3 = l3/l3[2]
l4 = l4/l4[2]

l5 = np.cross(s1,r1)
l6 = np.cross(r1,q1)
l5 = l5/l5[2]
l6 = l6/l6[2]

l7 = np.cross(r1,q1)
l8 = np.cross(q1,p1)
l7 = l7/l7[2]
l8 = l8/l8[2]

l9 = np.cross(p2,q2)
l10 = np.cross(p2,s2)
l9 = l1/l1[2]
l10 = l2/l2[2]

l11 = np.cross(p2,s2)
l12 = np.cross(s2,r2)
l11 = l11/l11[2]
l12 = l12/l12[2]

l13 = np.cross(s2,r2)
l14 = np.cross(r2,q2)
l13 = l13/l13[2]
l14 = l14/l14[2]



lines = [l1, l2, l3, l4, l5, l6, l7, l8, l9, l10, l11, l12, l13, l14]


A = np.zeros((7,5))
b = np.zeros((7,1))
for i in range(7):
    l = lines[2*i]
    m = lines[2*i+1]
    A[i][0:5] = [l[0]*m[0], (l[0]*m[1]+l[1]*m[0])/2, l[1]*m[1], (l[0]*
                                            m[2]+l[2]*m[0])/2, (l[1]*m[2]
                                            +l[2]*m[1])/2]
    b[i] = -l[2]*m[2]
s = np.matmul(np.linalg.pinv(A),b)
s = s/np.max(s)
S = np.zeros((2,2))
S[0,0] = s[0]
S[0,1] = s[1]/2
S[1,0] = s[1]/2
S[1,1] = s[2]
V,D,V_h = np.linalg.svd(S)
D = np.sqrt(np.diag(D))
```

```
    A_mat = np.matmul(np.matmul(V,D),V.transpose())
    bb = np.array([s[3]/2, s[4]/2])
    v = np.matmul(np.linalg.pinv(A_mat),bb)

    H = np.zeros((3,3))
    H[0,0] = A_mat[0,0]
    H[0,1] = A_mat[0,1]
    H[1,0] = A_mat[1,0]
    H[1,1] = A_mat[1,1]
    H[2,0] = v[0]
    H[2,1] = v[1]
    H[2,2] = 1

    return H

def rectImage(range_img, H):
    # Inverse of H
    H_inv = np.linalg.pinv(H)
    H_inv = H_inv/H_inv[2,2]
    # Get world coordinates
    world_coords = []
    img_coords = [(0, 0), (0, range_img.shape[0]-1), (range_img.shape[1]-1
                                        , range_img.shape[0]-1), (
                                        range_img.shape[1]-1, 0)]
    for i in img_coords:
        img_coord = np.array((i[0], i[1], 1))
        world_coord = np.matmul(H, img_coord)
        world_coord = world_coord/world_coord[2]
        world_coords.append([world_coord[0],world_coord[1]])

    # Get size of world image
    x_max = int(max([c[0] for c in world_coords]))
    x_min = int(min([c[0] for c in world_coords]))
    y_max = int(max([c[1] for c in world_coords]))
    y_min = int(min([c[1] for c in world_coords]))
    x_length = x_max - x_min
    y_length = y_max - y_min
    world_img = np.zeros((y_length, x_length, 3))
    for row in range(0, y_length):
        for col in range(0, x_length):
            domain_coord = np.array((col+x_min, row+y_min, 1))
            range_coord = np.matmul(H_inv,domain_coord)
            range_coord = range_coord / range_coord[2]
            range_coord = range_coord.astype(int)
            if (range_coord[0]<(range_img.shape[1]-1) and range_coord[1]<(
                                        range_img.shape[0]-1) and
                                         range_coord[0]>0 and
                                        range_coord[1]>0 ):
                world_img[row, col] = range_img[range_coord[1],
                                            range_coord[0]]
    return world_img

def getRectImage(img, coords1, coords2):
    boxImg = drawCircleNBox(img, coords1, coords2)
```

```
    # Get the Homography
    H = getH(coords1,coords2)
    H_inv = np.linalg.pinv(H)
    H_inv = H_inv/H_inv[2,2]

    # Get the rectified image
    img_Rect  = rectImage(img, H_inv)

    return boxImg, img_Rect

#----------------------------------Task---------------------------------#
# Input Images
building_img = cv2.imread('HW3/building.jpg')

# Coordinates
building_pqrs = [(346, 227),(345,378),(460,387),(460,256)]
building_pqrs2 = [(240,200),(236,367),(295,373),(298,216)]

building_boxImg, building_img_Rect = getRectImage(building_img,
                                     building_pqrs,building_pqrs2)

cv2.imwrite('HW3/building_box_img2.jpeg',building_boxImg)
cv2.imwrite('HW3/building_Rect1.3.jpeg',building_img_Rect)

# Input Image
nighthawks_img = cv2.imread('HW3/nighthawks.jpg')

# Coordinates
nighthawks_pqrs = [(76, 180), (78, 653), (805, 620), (802, 219)]
nighthawks_pqrs2 = [(12,100),(14,729),(865,678),(863,162)]

nighthawks_boxImg,nighthawks_img_Rect = getRectImage(nighthawks_img,
                                     nighthawks_pqrs,nighthawks_pqrs2)

cv2.imwrite('HW3/nighthawks_box_img2.jpeg', nighthawks_boxImg)
cv2.imwrite('HW3/nighthawks_Rect1.3.jpeg', nighthawks_img_Rect)

# Input Image
paintings_img = cv2.imread('HW3/paintings.jpg')

# Coordinates
paintings_pqrs = [(537,465),(538,947),(940,842),(933,485)]
paintings_pqrs2 = [(626,110),(628,275),(849,314),(847,173)]

paintings_boxImg, paintings_img_Rect = getRectImage(paintings_img,
                                     paintings_pqrs,paintings_pqrs2)

cv2.imwrite('HW3/paintings_boxImg2.jpeg', paintings_boxImg)
cv2.imwrite('HW3/paintings_img_Rect1.3.jpeg', paintings_img_Rect)

# Input Image
calendar_img = cv2.imread('HW3/calendar.jpg')
```

```
# Coordinates
calendar_pqrs = [(459,222),(491,973),(672,933),(652,241)]
calendar_pqrs2 = [(412,266),(420,451),(819,454),(816,298)]

calendar_boxImg, calendar_img_Rect = getRectImage(calendar_img,
                                     calendar_pqrs,calendar_pqrs2)

cv2.imwrite('HW3/calendar_boxImg2.jpeg', calendar_boxImg)
cv2.imwrite('HW3/calendar_img_Rect1.3.jpeg', calendar_img_Rect)
```