COMPUTER VISION

ECE 661

HOMEWORK 3

Tanzeel U. Rehman
Email: *trehman@purdue.edu*

1. Introduction:

This homework uses three different methods for removing the projective and affine distortions from the images. These methods return the homography matrices, which can then be inverted and applied on the real-world distorted images to eliminate the distortions.

a) The first method uses point-to-point correspondences (similar to the homework 2) between image pixel coordinates of the images and the points obtained from the given physical measurements for computing the homography matrix.

b) The second method is a two-step approach in which we first remove the projective distortion using the Vanishing line approach (Lecture 4). In the second step, we removed the affine distortion by using the angle-to-angle correspondence (*Cosθ* expression) with θ = 90 degrees i.e., orthogonal lines.

c) The third method is a one-step approach in which both projective and affine distortion are removed in one phase.

1.1. Description of Methods:

A. Point-to-point correspondence method:

Given a point X in a world scene and its corresponding pixel location X' in the distorted image plane can be given by Eq. 1.

$$X' = HX \qquad \text{(Eq.1)}$$

Where, X and X' are homogeneous coordinates of the form $\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$ and $\begin{bmatrix} x_1' \\ x_2' \\ x_3' \end{bmatrix}$ for physical points (x, y) in world plane and (x', y') in distorted image plane respectively. H is non-singular homography matrix in homogeneous coordinates which can be represented by Eq. 2. Using Eq. 2 and Eq. 1, Eq. 2 can be re-written as Eq. 3, which can further be expended as Eq. 4. The physical coordinate of each pixel ($x' = x_1'/x_3'$, $y' = x_2'/x_3'$) can be obtained as in Eq. 5. Dividing both numerator and denominator by x_3 for both (x', y'), we get just the physical coordinates on both sides as given in Eq. 6. The Eq. 6 can be rearranged as Eq. 7.

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \quad \text{(Eq.2)}$$

$$\begin{bmatrix} x_1' \\ x_2' \\ x_3' \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_1 \\ x_1 \end{bmatrix} \quad \text{(Eq.3)}$$

$$x_1' = h_{11}x_1 + h_{12}x_2 + h_{13}x_3 \quad \text{(Eq.4)}$$

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

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

$$x' = \frac{x_1'}{x_3'} = \frac{h_{11}x_1 + h_{12}x_2 + h_{13}x_3}{h_{31}x_1 + h_{32}x_2 + x_3} \quad \text{(Eq.5)}$$

$$y' = \frac{x_2'}{x_3'} = \frac{h_{21}x_1 + h_{22}x_2 + h_{23}x_3}{h_{31}x_1 + h_{32}x_2 + x_3}$$

$$x' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x_1 + h_{32}x_2 + 1} \quad \text{(Eq.6)}$$

$$y' = \frac{x_2'}{x_3'} = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + 1}$$

$$x' = h_{11}x + h_{12}y + h_{13} - h_{31}xx' - h_{32}x'y \quad \text{(Eq.7)}$$

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

Now using the Eq. 7, we can compute the unknown parameters of homography matrix *H*. Since there are eight unknown parameters, therefore, we need atleast 8 equations to obtain the unique solution which can be obtained using atleast 4 point pair [(*x*, *y*), (*x′*, *y′*)] correspondences. Using these four point pair correspondences, we can write the Eq. 7 in a matrix format given as Eq. 8, which can simply be written as Eq. 8. The Eq. 8 can finally be solved by taking the inverse of matrix *A* (or through least squares in case of over-determined equation system) to compute 8 unknowns of homography matrix ($\hat{X}$ in Eq. 9).

$$\begin{bmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ x_3' \\ y_3' \\ x_4' \\ y_4' \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_1' & -y_1x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_1' & -y_1y_1' \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x_2' & -y_2x_2' \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y_2' & -y_2y_2' \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x_3' & -y_3x_3' \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y_3' & -y_3y_3' \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x_4' & -y_4x_4' \\ 0 & 0 & 0 & x_4 & y_4 & 0 & -x_4y_4' & -y_4y_4' \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} \quad \text{(Eq.8)}$$

$$Y = A\hat{X} \quad \text{(Eq.9)}$$

After estimating the homography matrix between world and image plane, we can use the inverse homography matrix to map from image plane to the world plane to remove the distortions.

B. Projective distortion removal using vanishing lines:

In the two-step method, first projective distortion was removed using vanishing lines. In this method we computed the homography matrix that sends the vanishing lines back to l_∞. For computing this matrix, we first identified two pairs of parallel lines in the physical scene. To obtain the homogeneous representation of a specific line, we identified two points on these lines and take their cross product. Let two pairs of parallel lines be l_1, m_1 and l_2, m_2, respectively. Taking the cross product of a pair of parallel lines (in physical scene) yielded the homogeneous representation of the vanishing point (intersection point) for these parallel lines (Eq. 10).

$$P_1 = l_1 \times m_1 \quad \text{(Eq. 10)}$$

$$P_2 = l_2 \times m_2$$

$$VL = P_1 \times P_2 \quad \text{(Eq. 11)}$$

Where, P_1 *and* P_2 are two vanishing points obtained from two parallel line pairs. The vanishing line joining these two points will be obtained using Eq. 12. Let, the homogeneous coordinates of

vanishing line be $VL = [ll_1 \quad ll_2 \quad ll_3]^T$, then homography matrix for sending the vanishing line back to l_∞ can be written as Eq. 13.

$$H_p = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ ll_1 & ll_2 & ll_3 \end{bmatrix} \quad \text{(Eq. 12)}$$

Once applied to image, the inverse of homography H_p removes the projective distortion.

C. Affine distortion removal using angle-to-angle correspondence (*Cosθ* expression):

As the affine distortion causes the unequal scaling in orthogonal directions i.e., angles are not preserved, therefore, we need two orthogonal lines in physical space to remove affine distortion. Let $L = [l_1 \quad l_2 \quad l_3]^T$ and $M = [m_1 \quad m_2 \quad m_3]^T$ be the two orthogonal lines in the physical space, then from lecture 5, angle between them can be expressed by Eq. 14.

$$Cos\theta = \frac{L^T C_\infty{}^* M}{\sqrt{(L^T C_\infty{}^* L)(M^T C_\infty{}^* M)}} \quad \text{(Eq. 14)}$$

Where,

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

Now, let *Ha* be the affine homography, which maps the world plane to image plane, then lines mapped to image will be given by Eq. 15, whereas conics mapped to image will be given by Eq. 16. Since, we selected the orthogonal lines in the physical scene, therefore *Cosθ* = 0. Substituting the values of *L*, *M* and $C_\infty{}^*$ from Eq. 15 and Eq. 16 to Eq. 14 together with *Cosθ* = 0 will simplify to Eq. 17.

$$L = H_a{}^T L' \quad \text{(Eq. 15)}$$

$$M = H_a{}^T M'$$

$$C_\infty{}^* = {H_a}^{-1} C_\infty{}^{*\prime} {H_a}^{-T} \quad \text{(Eq. 16)}$$

$$L'^T H_a \, C_\infty{}^* {H_a}^T M' = 0 \quad \text{(Eq. 17)}$$

Where,

$$H_a = \begin{bmatrix} A & 0 \\ 0 & 1 \end{bmatrix}$$

Expending the Eq. 17 yield Eq. 18. Denoting $S = AA^T = \begin{bmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{bmatrix}$, and since the *S* is symmetric i.e., $s_{12} = s_{21}$, Eq. 18 can further be simplified to Eq. 19. As we are working in homogeneous coordinates and information is contained in the ratios, we can set $s_{12} = 1$. Now we have only two unknowns in Eq. 19, which can be obtained by using two pairs of orthogonal lines. After obtaining the *S* matrix, we can decompose it using singular value decomposition (SVD) to get the *A* matrix by relying on the relation in Eq. 20.

$$[{l_1}' \quad {l_2}' \quad {l_3}'] \begin{bmatrix} AA^T & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} {m_1}' \\ {m_2}' \\ {m_3}' \end{bmatrix} = 0 \quad \text{(Eq. 18)}$$

$$s_{11} m_1' {l_1}' + s_{12}({l_1}' m_2' + {l_2}' m_1') + s_{22} l_2' m_2' = 0 \quad \text{(Eq. 19)}$$

$$A = VDV^T \quad \text{(Eq. 20)}$$

$$S = AA^T = VD^2V^T$$

Once matrix A is computed, the homography *Ha* for eliminating the affine distortion can be computed as per Eq. 17.

D. One-step method:

The third method directly removes both projective and affine transformation in the one-move. This method uses the image $C_\infty{}^{*\prime}$ of dual degenerate conic $C_\infty{}^*$. Let *H* be the homography matrix responsible for removing both projective and affine distortion, then this homography in general form can be written as Eq. 21. The image of dual degenerate conic can be written as Eq. 22. A pair

of orthogonal lines ($L = [l_1 \quad l_2 \quad l_3]^T$ and $M = [m_1 \quad m_2 \quad m_3]^T$) in the physical space can be represented in image as Eq. 23. There are five unknowns needed to compute the homography matrix as we can set *f =1* (since only ratios matter), which can be calculated by using 5 pairs of orthogonal lines in physical space. Given 5 pairs of orthogonal lines, we denoted $S = AA^T = \begin{bmatrix} a & b/2 \\ b/2 & c \end{bmatrix}$ (Eq. 24) and decomposed it using singular value decomposition (SVD) to get the *A* matrix similar to the section C. Similar to the angle-to-angle correspondence method, we computed the vector *v* using Eq. 25. The homography calculated in this method in inverted and applied to images to remove both projective and affine distortions together.

$$Hap = \begin{bmatrix} A & 0 \\ v^t & 1 \end{bmatrix} \quad \text{(Eq. 21)}$$

$$C_\infty{}^{*\prime} = HC_\infty{}^{*}H^T = \begin{bmatrix} AA^T & Av \\ v^tA^t & v^tv \end{bmatrix} = \begin{bmatrix} a & b/2 & c/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix} \quad \text{(Eq. 22)}$$

$$L'^T C_\infty{}^{*\prime} M' = 0 \quad \text{(Eq. 23)}$$

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

$$S = AA^T = \begin{bmatrix} a & b/2 \\ b/2 & c \end{bmatrix} \quad \text{(Eq. 24)}$$

$$A \times v = [\frac{d}{2}, \frac{e}{2}]^T \quad \text{(Eq. 25)}$$

1.2. Computing RGB Values for each pixel location:

After computing the homography matrix, it can be inverted and applied to the homogeneous coordinates of point *X′* to obtain the homogeneous coordinates of *X*. These coordinates can be used

to obtain the point coordinates in the physical space (x, y), however, these values can be non-integral. For this homework, we rounded the non-integral values of (x, y) to the nearest integer. These rounded integers were then used for extracting the RGB values from the range image and were projected on the domain image.

1.3. Assumptions and Notes:

a) In this homework, we used minimal (only four) corresponding point pairs to estimate the homography matrix.

b) Two pairs of lines including PQ || RS, PR||QS are used for vanishing line method (projective distortion removal) and two pairs of orthogonal lines PQ $\perp$ QS and PS $\perp$ QR for angle-to-angle correspondence (affine distortion removal).

c) Five pairs including PQ $\perp$ PR, PR $\perp$ RS, PQ $\perp$ QS, QS $\perp$ SR and PS $\perp$ QR were used for one step method.

d) Selecting the appropriate points in the images are very important to achieve the desirable results.

e) We normalized the lines, points and homographies before applying to obtain the undistorted image. We ensured that 1 pair of orthogonal lines is completely different from others**.**

f) For two step method, we computed the total homography via combining vanishing line and the angle-to-angle homography by taking the dot product of $H_a{}^{-1}$ and H_p to directly remove both projective and affine distortions directly.

g) We computed the scale factor between world plane and image plane using the method mentioned in the homework handout. This scale factor was used while warping the images.

h) The real world height and width (cm) of the selected objects in my input images are given in the caption of Figure 16, and 21.

i) For applying warping, we just found the nearest integer pixel coordinates. However, this method does not always yield very nice results, as it can leave some pixels unfilled during warping and also cause edges to be relatively blurry. In future applications, an interpolation technique needs to be used to achieve aesthetically appealing results.

j) **Sanity check:** Without using the scale factor, we corrected the pixel coordinates of picked distorted points through the computed homography. The distance between the corrected points corresponded to the measured physical distance.

2. Task 1:

2.1. Input and resulting images:

Fig 1: Input Image 1. The points PQSR were used for removing the distorting using all three methods (see details which pairs were used for specific method in section 1.3). Yellow lines represent diagonal lines.

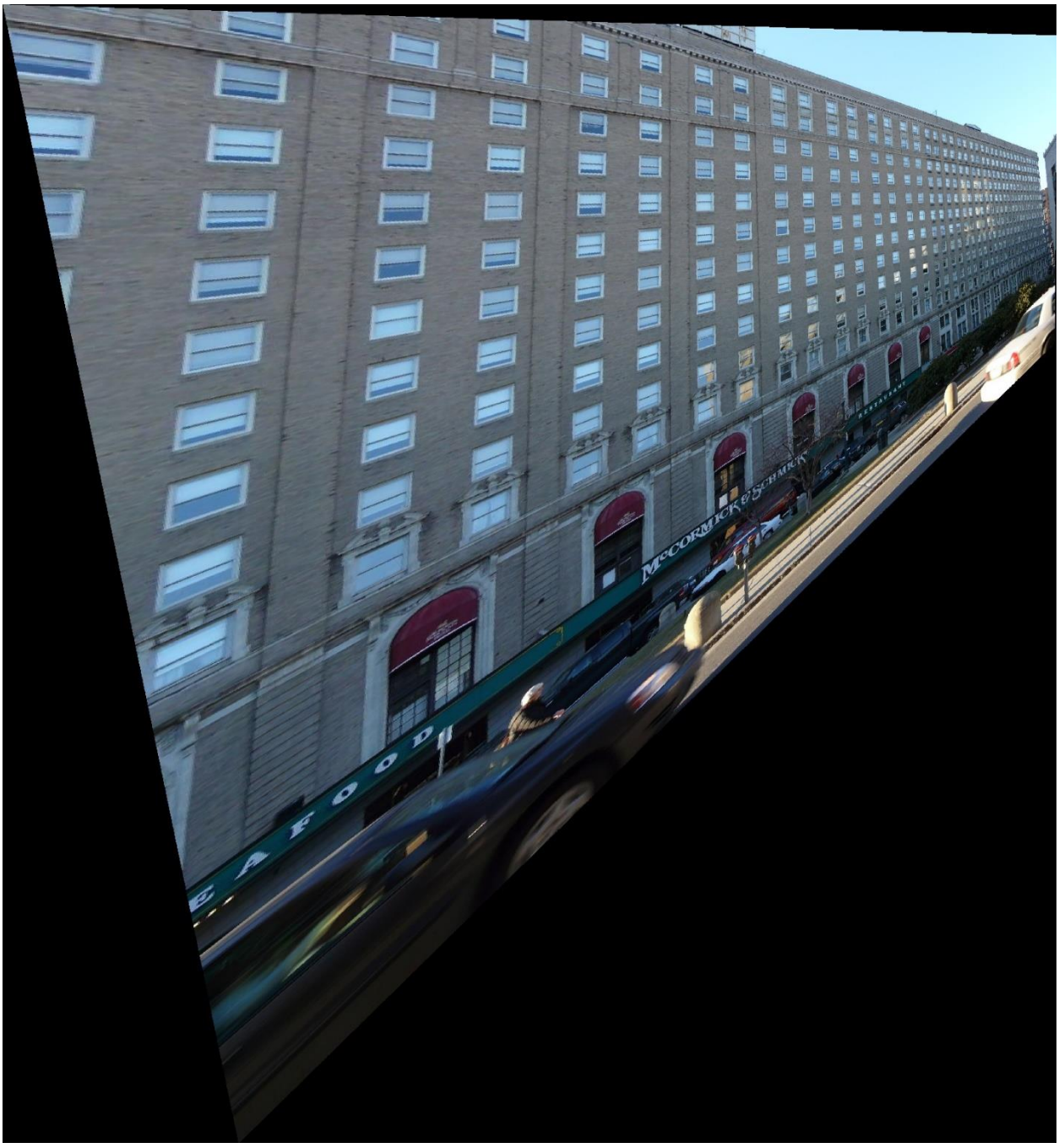

Fig 2: Distortion removal from input image 1 using the point-to-point correspondence method.

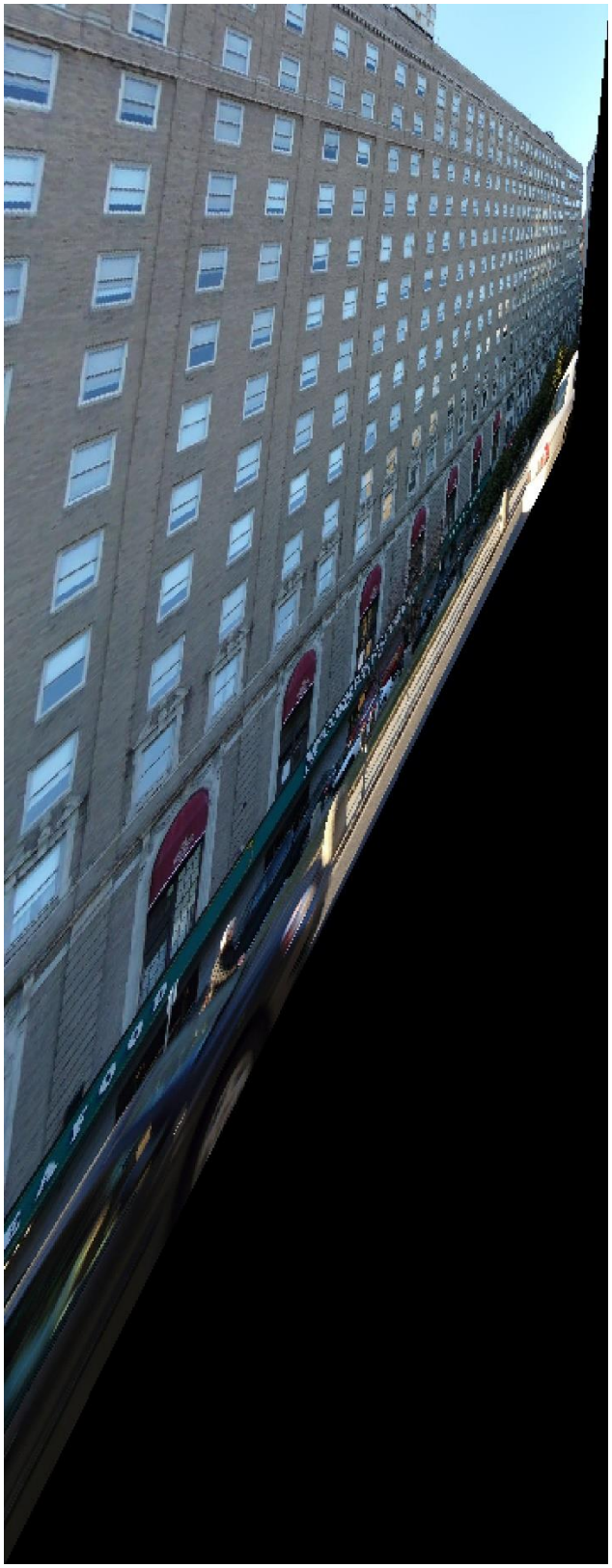

Fig 3: Projective distortion removal from input image 1 using vanishing lines method (1st of 2-steps).

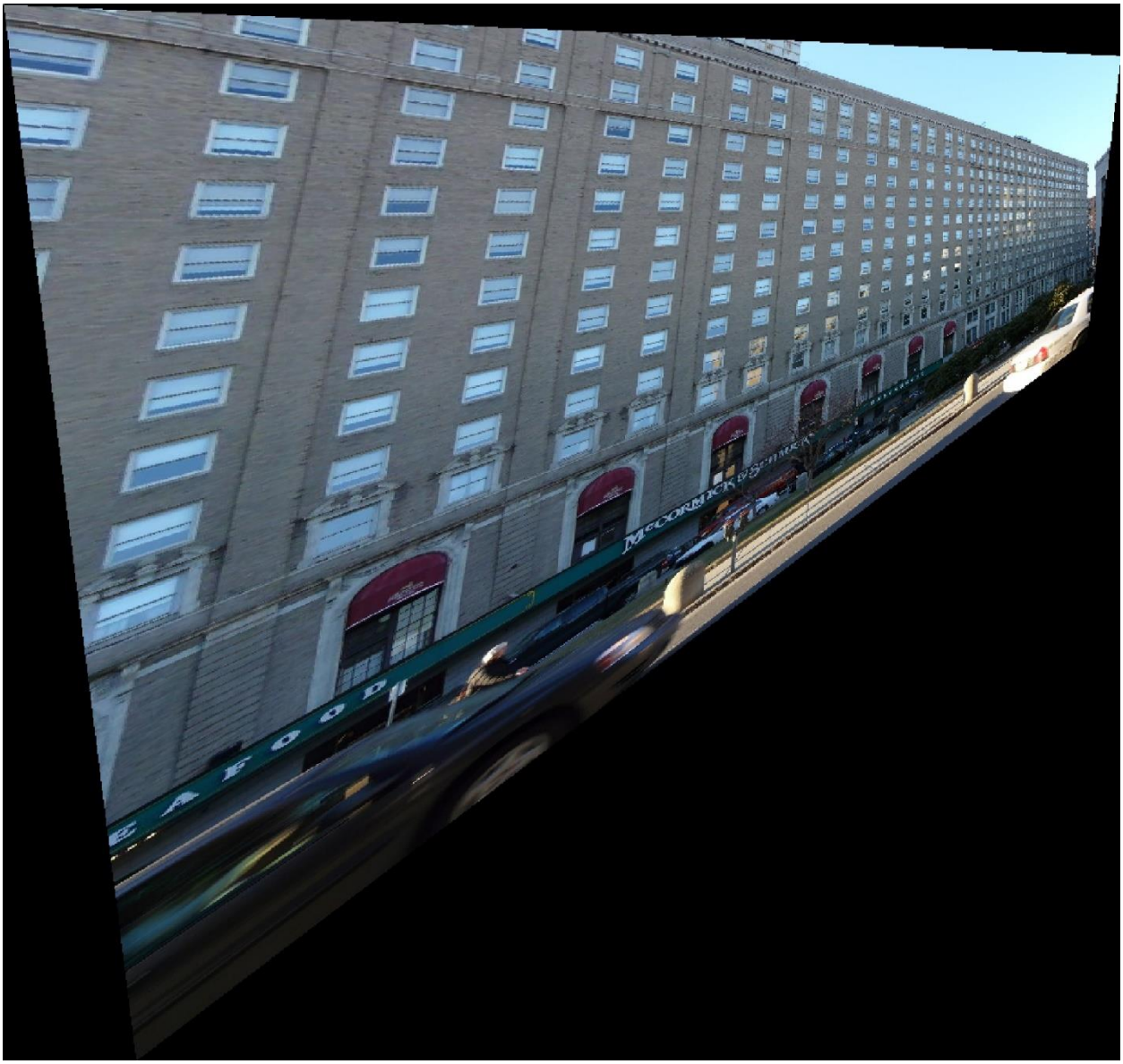

Fig 4: Affine distortion removal from input image 1 using angle-to-angle correspondence (2nd of 2-steps).

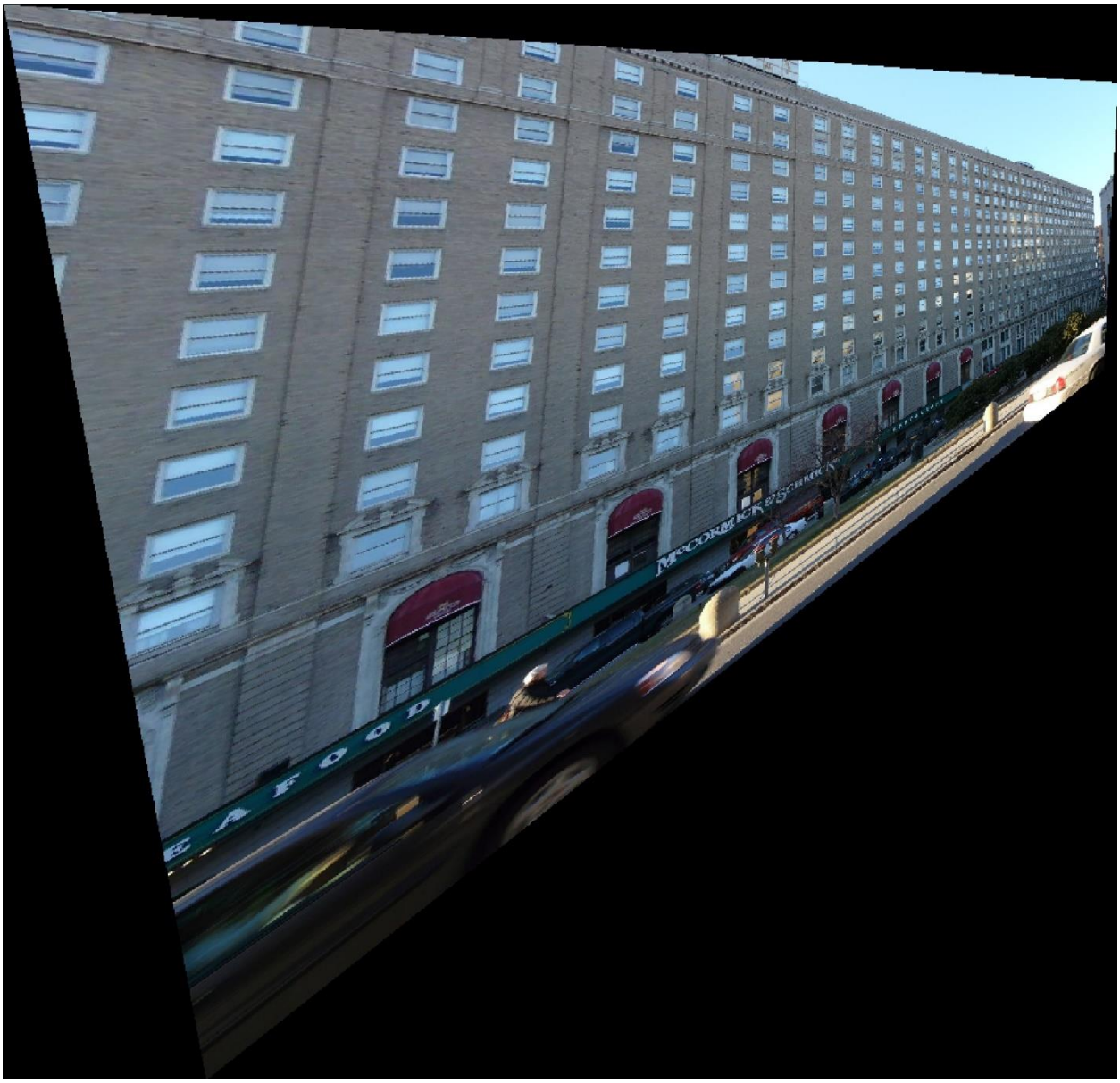

Fig 5: Removing both projective and affine distortion from input image 1 using one-step method.

Fig 6: Input Image 2. The points PQSR are used for removing projective and affine distortion using the one and two-step methods. Yellow lines represent diagonal lines. For point-to-point correspondence, points P′ Q′ S′ R′ were used. (See details which pairs were used for specific method in section 1.3).

Fig. 7: Distortion removal from input image 2 using the point-to-point correspondence method.

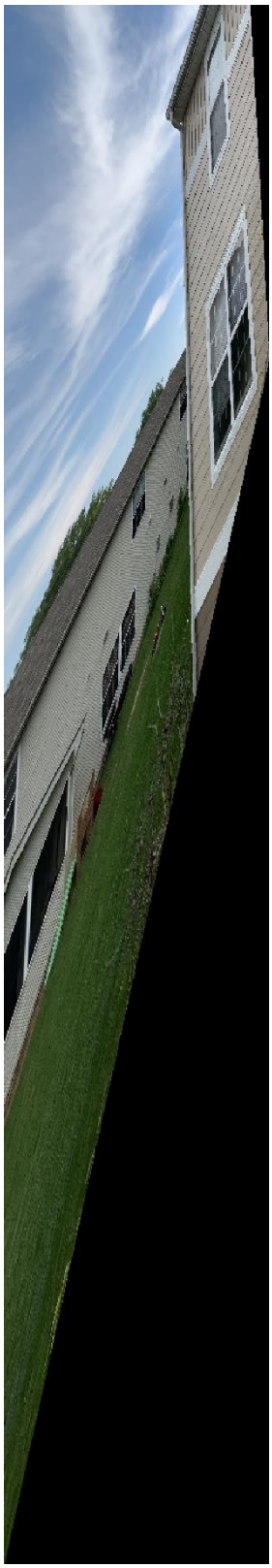

Fig 8: Projective distortion removal from input image 2 using vanishing lines method (1st of 2-steps).

Fig 9: Affine distortion removal from input image 2 using angle-to-angle correspondence (2nd of 2-steps).

Fig 10: Removing both projective and affine distortion from input image 2 using one-step method.

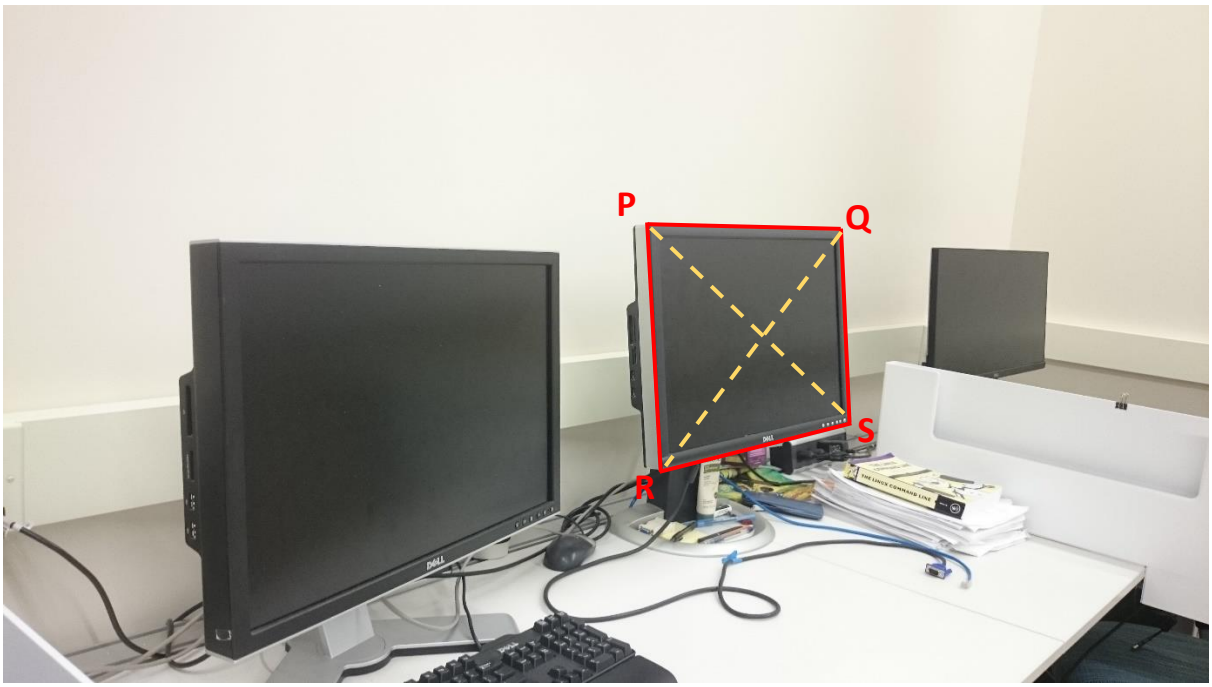

Fig 11: Input Image 3. The points PQSR were used for removing the distorting using all three methods (see details which pairs were used for specific method in section 1.3). Yellow lines represent diagonal lines.

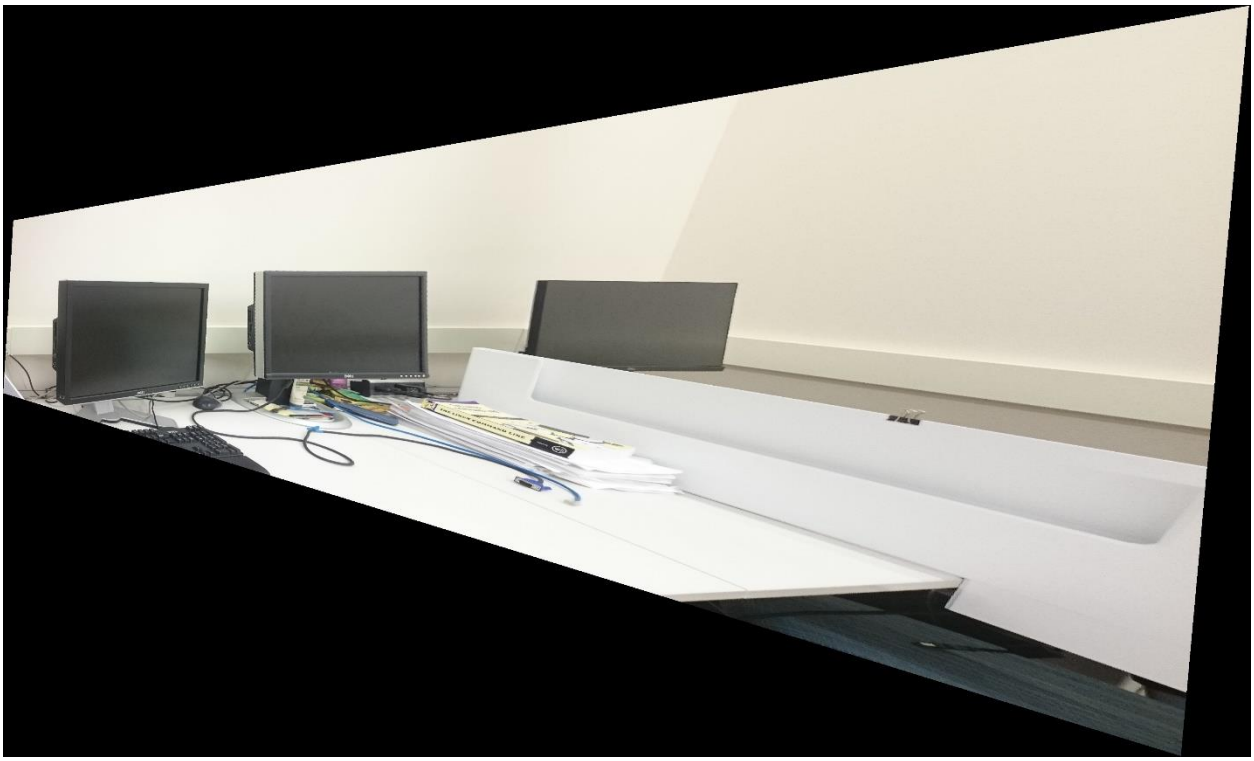

Fig. 12: Distortion removal from input image 3 using the point-to-point correspondence method.

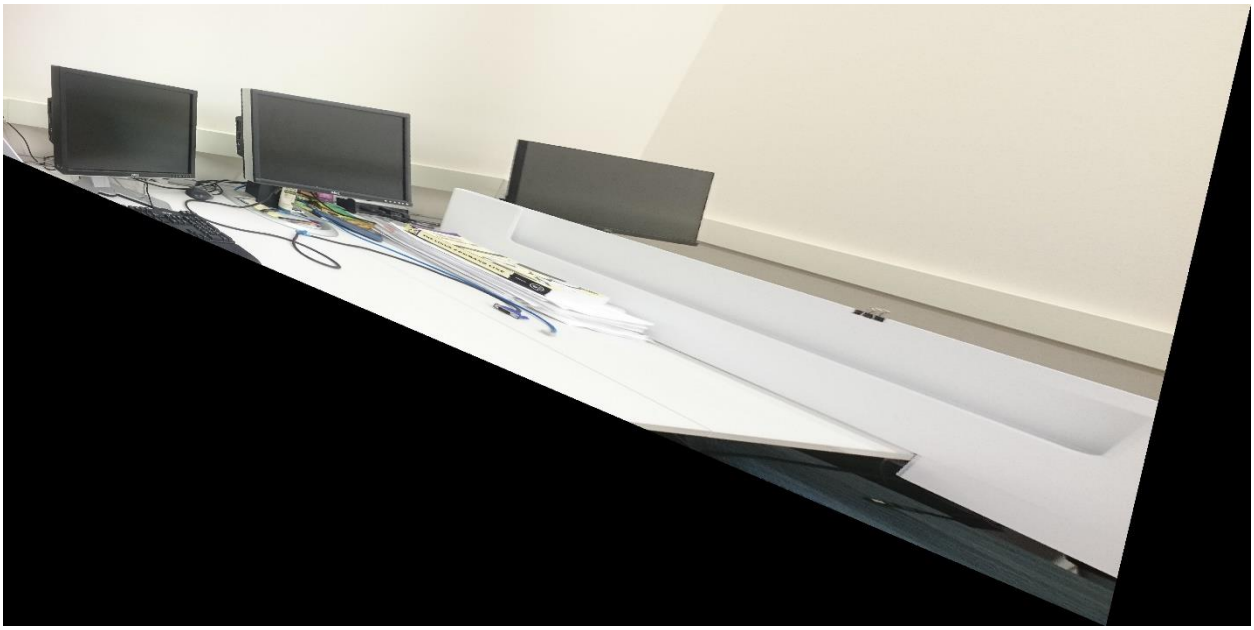

Fig 13: Projective distortion removal from input image 3 using vanishing lines method (1st of 2-steps).

Fig 14: Affine distortion removal from input image 3 using angle-to-angle correspondence (2nd of 2-steps).

Fig 15: Removing both projective and affine distortion from input image 3 using one-step method.

3.1.Task 2:

3.2.Inputs of Task 2.1

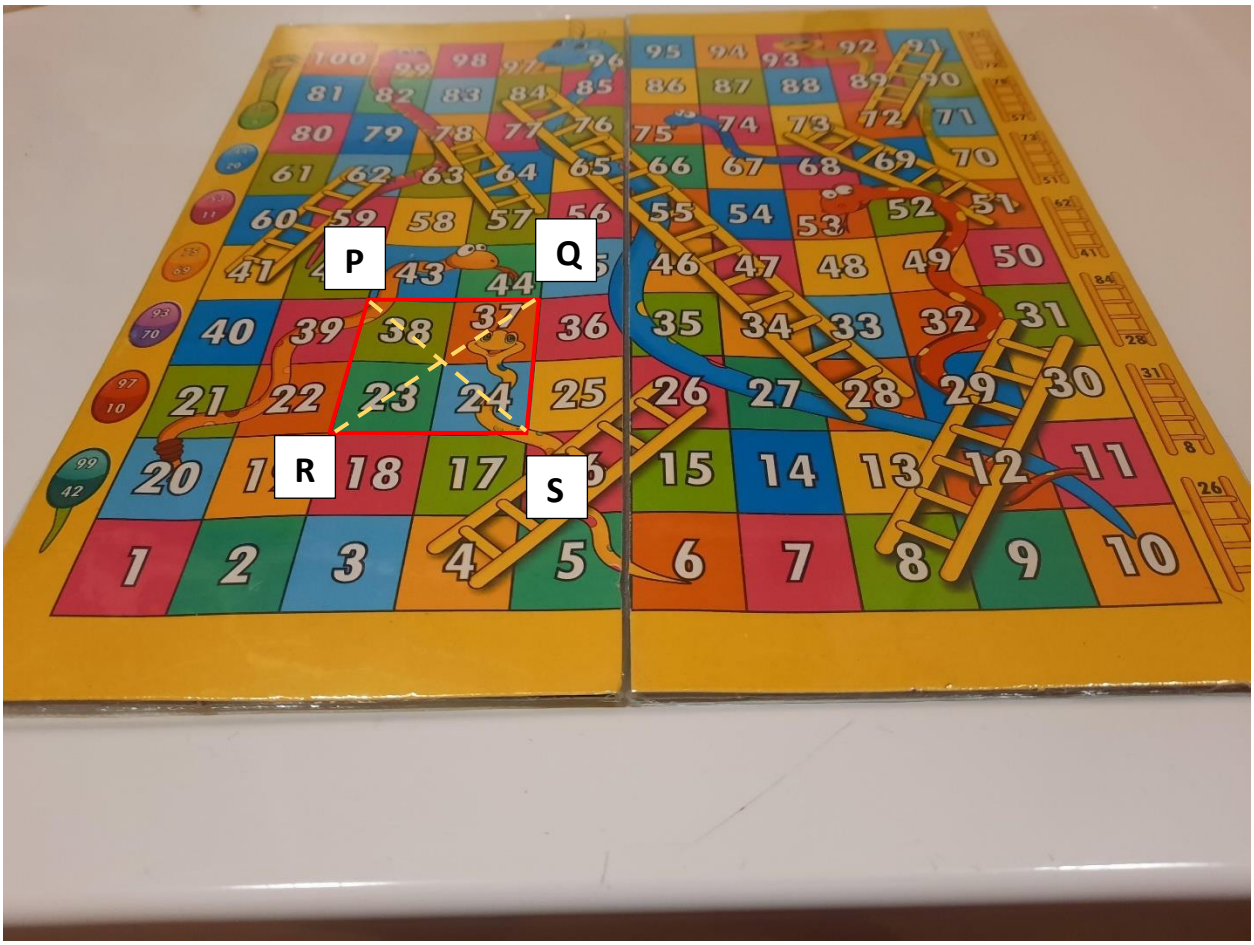

Fig 16: My input image 1. The points PQSR were used for removing the distorting using all three methods (see details which pairs were used for specific method in section 1.3). Yellow lines represent diagonal lines. I assumed the width of two boxes (RS) = 4.11 cm and Height of two boxes (QS) = 4.1 cm, respectively.

Fig. 17: Distortion removal from my input image 1 using the point-to-point correspondence method.

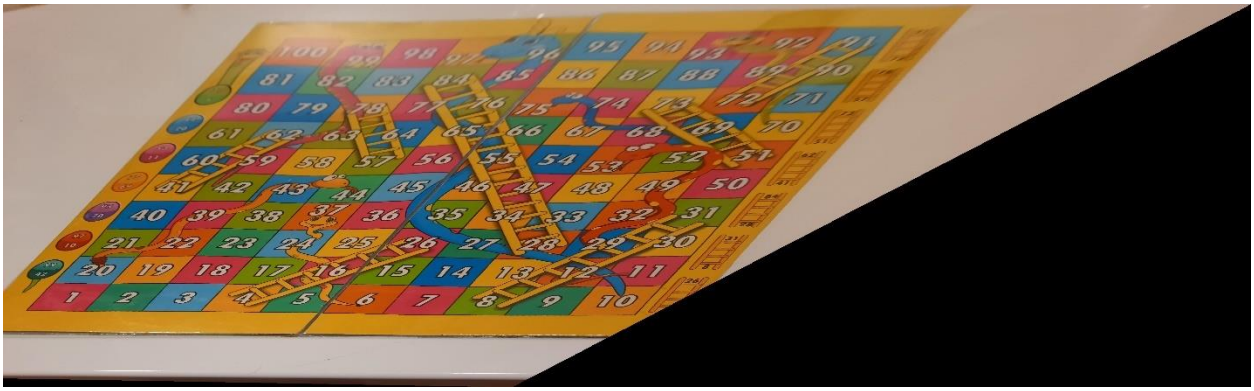

Fig 18: Projective distortion removal from my input image 1 using vanishing lines method (1st of 2-steps).

Fig 19: Affine distortion removal from my input image 1 using angle-to-angle correspondence (2nd of 2-steps).

Fig 20: Removing both projective and affine distortion from my input image 1 using one-step method.

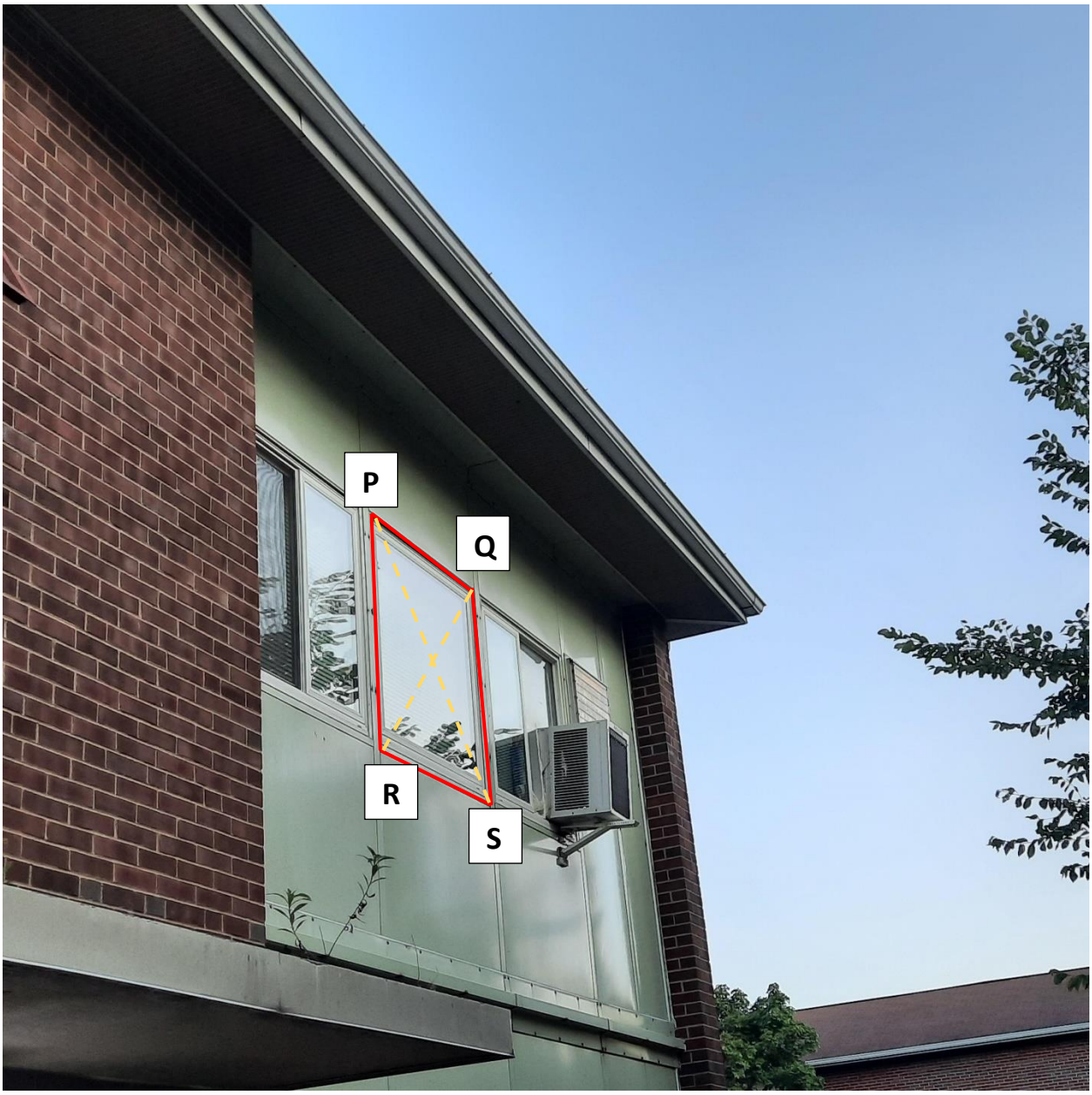

Fig 21: My input image 2. The points PQSR were used for removing the distorting using all three methods (see details which pairs were used for specific method in section 1.3). Yellow lines represent diagonal lines. I assumed the width of window (RS) = 115cm and Height of window (QS) = 140 cm, respectively.

Fig. 22: Distortion removal from my input image 2 using the point-to-point correspondence method.

Fig 23: Projective distortion removal from my input image 2 using vanishing lines method (1st of 2-steps).

Fig 24: Affine distortion removal from my input image 2 using angle-to-angle correspondence (2nd of 2-steps).

Fig 25: Removing both projective and affine distortion from my input image 2 using one-step method.

4. Observation of different methods:

1. After removing the distortion from given and my own images using three different methods, I observed that point-to-point correspondence method is more intuitive and can

lead to the acceptable results relatively quickly. One problem which I observed was that if the points are not properly picked, we can get relatively poor results most probably due to noisy homography matrix. For this method, also the sequence of the points is very important, otherwise orientation of new image will be different.

2. The two step approach requires atleast two pairs of parallel and two pairs of perpendicular lines in real-space for removing both projective and affine transformation. This method is relatively easy to understand and less sensitive to the points picked compared to one step method. However, it requires to go through two steps to completely remove the distortion.

3. For one-step method, the pairs of orthogonal lines need to be picked properly, otherwise the homography matrix will not be very accurate, so it is sensitive to the points chosen.

4. I also observed that if we are not using the interpolation as in the case of this homework for warping, we might have blurry edges in the images and sometime some missing pixel values.

5. In case of big building façade (e.g. given Image 1), if the points are incorrectly picked, or picked in the region impacted by a lot of distortion, we might end up with very poor results.

5. Source Code:

5.1. Function calls for Task 1.1:

```
'''-------Main Code for Task-1.1---------'''
# Sequentially go over through 3 images for removing distortion using point-to-point
#Image 1
image_1 = cv2.imread('Img1.JPG')
PQSR_img_1 = np.array([[463,282],[489, 289],[487, 325],[461, 317]])
PQSR_world_1 = np.array([[0,0],[75,0],[75,85],[0,85]])
```

```
H1 = find_homography (PQSR_img_1 , PQSR_world_1)
H1_inv = np.linalg.inv(H1)
xmin, ymin, scale, undistImage1 = Bounds_Undistorted(H1,image_1)
undistImage1= transform_image(xmin, ymin, scale, undistImage1, H1_inv,image_1)
cv2.imwrite('Img_1_Results.jpg',undistImage1)
cv2.imshow('Img_1_undistorted',undistImage1)

#Image 2
image_2 = cv2.imread('Img2.jpeg')
PQSR_img_2 = np.array([[473,564],[590, 549],[599, 722],[480, 710]])
PQSR_world_2 = np.array([[0,0],[84,0],[84,74],[0,74]])
H2 = find_homography (PQSR_img_2 , PQSR_world_2)
H2_inv = np.linalg.inv(H2)
xmin, ymin, scale, undistImage2 = Bounds_Undistorted(H2_inv,image_2)
undistImage= transform_image(xmin, ymin, scale, undistImage2,H2_inv,image_2)
cv2.imwrite('Img_2_Results.jpg',undistImage2)
cv2.imshow('Img_2_undistorted',undistImage2)

#Image 3
image_3 = cv2.imread('Img3.JPG')
PQSR_img_3 = np.array([[2055,695],[2670, 715],[2692, 1332],[2088, 1475]])
PQSR_world_3 = np.array([[0,0],[55,0],[55,36],[0,36]])
H3 = find_homography (PQSR_img_3 , PQSR_world_3)
H3_inv = np.linalg.inv(H3)
xmin, ymin, scale, undistImage3 = Bounds_Undistorted(H3,image_3)
undistImage3= transform_image(xmin, ymin, scale, undistImage3, H3_inv,image_3)
cv2.imwrite('Img_3_Results.jpg',undistImage3)
cv2.imshow('Img_3_undistorted',undistImage3)
```

5.2.Function calls for Task 1.2:

```
'''----------Task 1.2: Vanishing line + Afine----------'''
# Image 1
image_1 = cv2.imread('Img1.JPG')
PQSR_img_1 = np.array([[463,282],[489, 289],[487, 325],[461, 317]])
H_V = find_VL_homography (PQSR_img_1)
H_V_inv = np.linalg.inv(H_V)
xmin, ymin, scale, undistImage_VL = Bounds_Undistorted(H_V,image_1)
undistImage_VL= transform_image(xmin, ymin, scale, undistImage_VL, H_V_inv,image_1)
cv2.imwrite('Img_1_VL.jpg',undistImage_VL)
#Remove affine
H_Aff = ProjectiveCorr_Points(H_V,PQSR_img_1)
H_Aff_inv = np.linalg.inv(H_Aff)
H_VL_Aff = np.matmul(H_Aff_inv,H_V)
H_VL_Aff_inv = np.linalg.inv(H_VL_Aff)
xmin, ymin, scale, undistImage_VLAF = Bounds_Undistorted(H_VL_Aff,image_1)
```

```
undistImage_VLAF= transform_image(xmin, ymin, scale, undistImage_VLAF, H_VL_Aff_inv,image_1)
cv2.imwrite('Img_1_VL_Aff.jpg',undistImage_VLAF)

# Image 2
image_2 = cv2.imread('Img2.jpeg')
PQSR_img_2 = np.array([[424,231],[514, 167],[514, 298],[425, 346]])
H_V = find_VL_homography (PQSR_img_2)
H_V_inv = np.linalg.inv(H_V)
xmin, ymin, scale, undistImage_VL = Bounds_Undistorted(H_V,image_2)
undistImage_VL= transform_image(xmin, ymin, scale, undistImage_VL, H_V_inv,image_2)
cv2.imwrite('Img_2_VL.jpg',undistImage_VL)
#Remove affine
H_Aff = ProjectiveCorr_Points(H_V,PQSR_img_2)
H_Aff_inv = np.linalg.inv(H_Aff)
H_VL_Aff = np.matmul(H_Aff_inv,H_V)
H_VL_Aff_inv = np.linalg.inv(H_VL_Aff)
xmin, ymin, scale, undistImage_VLAF = Bounds_Undistorted(H_VL_Aff,image_2)
undistImage_VLAF= transform_image(xmin, ymin, scale, undistImage_VLAF, H_VL_Aff_inv,image_2)
cv2.imwrite('Img_2_VL_Aff.jpg',undistImage_VLAF)

# Image 3
image_3 = cv2.imread('Img3.jpg')
PQSR_img_3 = np.array([[2058,698],[2658, 715],[2689, 1329],[2094, 1472]])
H_V = find_VL_homography (PQSR_img_3)
H_V_inv = np.linalg.inv(H_V)
xmin, ymin, scale, undistImage_VL = Bounds_Undistorted(H_V,image_3)
undistImage_VL= transform_image(xmin, ymin, scale, undistImage_VL, H_V_inv,image_3)
cv2.imwrite('Img_3_VL.jpg',undistImage_VL)
# Remove affine
H_Aff = ProjectiveCorr_Points(H_V,PQSR_img_3)
H_Aff_inv = np.linalg.inv(H_Aff)
H_VL_Aff = np.matmul(H_Aff_inv,H_V)
H_VL_Aff_inv = np.linalg.inv(H_VL_Aff)
xmin, ymin, scale, undistImage_VLAF = Bounds_Undistorted(H_VL_Aff,image_3)
undistImage_VLAF= transform_image(xmin, ymin, scale, undistImage_VLAF, H_VL_Aff_inv,image_3)
cv2.imwrite('Img_3_VL_Aff.jpg',undistImage_VLAF)
```

5.3.Function calls for Task 1.3:

```
"""------One step method------"""
#Image 1
image_1 = cv2.imread('Img1.jpg')
PQSR_img_1 = np.array([[463,282],[489, 289],[487, 325],[461, 317]])
H_one = find_homography_one(PQSR_img_1)
```

```
H_one_inv = np.linalg.inv(H_one)
xmin, ymin, scale, undistImage_one = Bounds_Undistorted(H_one_inv,image_1)
undistImage_one = transform_image(xmin, ymin, scale, undistImage_one, H_one,image_1)
cv2.imwrite('Img_1_One_step.jpg',undistImage_one)

#Image 2
image_2 = cv2.imread('Img2.jpeg')
PQSR_img_2 = np.array([[424,231],[514, 167],[514, 298],[425, 346]])
H_one = find_homography_one(PQSR_img_2)
H_one_inv = np.linalg.inv(H_one)
xmin, ymin, scale, undistImage_one = Bounds_Undistorted(H_one_inv,image_2)
undistImage_one = transform_image(xmin, ymin, scale, undistImage_one, H_one,image_2)
cv2.imwrite('Img_2_One_step.jpg',undistImage_one)

#Image 3
image_3 = cv2.imread('Img3.jpg')
PQSR_img_3 = np.array([[2058,698],[2658, 715],[2689, 1329],[2094, 1472]])
H_one = find_homography_one(PQSR_img_3)
H_one_inv = np.linalg.inv(H_one)
xmin, ymin, scale, undistImage_one = Bounds_Undistorted(H_one_inv,image_3)
undistImage_one = transform_image(xmin, ymin, scale, undistImage_one, H_one,image_3)
cv2.imwrite('Img_3_One_step.jpg',undistImage_one)
```

5.4.Function calls for Task 2.1:

```
'''--------Code For Task 2.1-----------'''
#My image 1
My_image_1 = cv2.imread('My_img_1.jpg')
PQSR_Myimg_1 = np.array([[1168,939],[1721, 941],[1680, 1370],[1049, 1370]])
#PQSR_world_1 = np.array([[0,0],[2.54,0],[2.54,2.54],[0,2.54]])
PQSR_world_1 = np.array([[0,0],[4.1,0],[4.1,4.1],[0,4.1]])
H3 = find_homography (PQSR_Myimg_1 , PQSR_world_1)
H3_inv = np.linalg.inv(H3)
xmin, ymin, scale, undistImage3 = Bounds_Undistorted(H3,My_image_1)
undistImage3= transform_image(xmin, ymin, scale, undistImage3, H3_inv,My_image_1)
cv2.imwrite('My_Img_1_Results.jpg',undistImage3)
cv2.imshow('My_Img_1_undistorted',undistImage3)

#My image 2
My_image_2 = cv2.imread('My_img_2.jpg')
PQSR_Myimg_2 = np.array([[1009,1393],[1296, 1609],[1321, 2179],[1044, 2006]])
PQSR_world_2 = np.array([[0,0],[115,0],[115,140],[0,140]])
H3 = find_homography (PQSR_Myimg_2 , PQSR_world_2)
H3_inv = np.linalg.inv(H3)
xmin, ymin, scale, undistImage3 = Bounds_Undistorted(H3,My_image_2)
undistImage3= transform_image(xmin, ymin, scale, undistImage3, H3_inv,My_image_2)
cv2.imwrite('My_Img_2_Results.jpg',undistImage3)
```

```
cv2.imshow('My_Img_2_undistorted',undistImage3)
```

5.5.Function calls for Task 2.2:

```
'''-------------Task 1.2: Vanishing line + Afine--------------------'''
#My image 1
My_image_1 = cv2.imread('My_img_1.jpg')
PQSR_Myimg_1 = np.array([[1168,939],[1721, 941],[1680, 1370],[1049, 1370]])
H_V = find_VL_homography (PQSR_Myimg_1)
H_V_inv = np.linalg.inv(H_V)
xmin, ymin, scale, undistImage_VL = Bounds_Undistorted(H_V,My_image_1)
undistImage_VL= transform_image(xmin, ymin, scale, undistImage_VL, H_V_inv,My_image_1)
cv2.imwrite('My_Img_1_VL.jpg',undistImage_VL)
# Remove affine
H_Aff = ProjectiveCorr_Points(H_V,PQSR_Myimg_1)
H_Aff_inv = np.linalg.inv(H_Aff)
H_VL_Aff = np.dot(H_Aff_inv,H_V)
H_VL_Aff_inv = np.linalg.inv(H_VL_Aff)
xmin, ymin, scale, undistImage_VLAF = Bounds_Undistorted(H_VL_Aff,My_image_1)
undistImage_VLAF= transform_image(xmin, ymin, scale, undistImage_VLAF,
H_VL_Aff_inv,My_image_1)
cv2.imwrite('My_Img_1_VL_Aff.jpg',undistImage_VLAF)

#My image 2
My_image_2 = cv2.imread('My_img_2.jpg')
PQSR_Myimg_2 = np.array([[1009,1393],[1296, 1609],[1321, 2179],[1044, 2006]])
H_V = find_VL_homography (PQSR_Myimg_2)
H_V_inv = np.linalg.inv(H_V)
xmin, ymin, scale, undistImage_VL = Bounds_Undistorted(H_V,My_image_2)
undistImage_VL= transform_image(xmin, ymin, scale, undistImage_VL, H_V_inv,My_image_2)
cv2.imwrite('My_Img_2_VL.jpg',undistImage_VL)
# Remove affine
H_Aff = ProjectiveCorr_Points(H_V,PQSR_Myimg_2)
H_Aff_inv = np.linalg.inv(H_Aff)
H_VL_Aff = np.dot(H_Aff_inv,H_V)
H_VL_Aff_inv = np.linalg.inv(H_VL_Aff)
xmin, ymin, scale, undistImage_VLAF = Bounds_Undistorted(H_VL_Aff,My_image_2)
undistImage_VLAF= transform_image(xmin, ymin, scale, undistImage_VLAF,
H_VL_Aff_inv,My_image_2)
cv2.imwrite('My_Img_2_VL_Aff.jpg',undistImage_VLAF)
```

5.6.Function calls for Task 2.2:

```
'''---------One_step Method------------'''
#My image 1
My_image_1 = cv2.imread('My_img_1.jpg')
PQSR_Myimg_1 = np.array([[1168,939],[1721, 941],[1680, 1370],[1049, 1370]])
```

```
H_one = find_homography_one(PQSR_Myimg_1)
H_one_inv = np.linalg.inv(H_one)
xmin, ymin, scale, undistImage_one = Bounds_Undistorted(H_one_inv,My_image_1)
undistImage_one = transform_image(xmin, ymin, scale, undistImage_one, H_one,My_image_1)
cv2.imwrite('My_Img_1_One_step.jpg',undistImage_one)

#My image 2
My_image_2 = cv2.imread('My_img_2.jpg')
PQSR_Myimg_2 = np.array([[1009,1393],[1296, 1609],[1321, 2179],[1044, 2006]])
H_one = find_homography_one(PQSR_Myimg_2)
H_one_inv = np.linalg.inv(H_one)
xmin, ymin, scale, undistImage_one = Bounds_Undistorted(H_one_inv,My_image_2)
undistImage_one = transform_image(xmin, ymin, scale, undistImage_one, H_one,My_image_2)
cv2.imwrite('My_Img_2_One_step.jpg',undistImage_one)
```

A. Code for computing the homography matrix using point to point correspondence:

```
import numpy as np
import cv2

def find_homography (Domain_pts, Range_pts):
    '''
    function for estimating the 3 x 3 Homography matrix
     Inputs:
        Domain_pts: An n x 2 array containing coordinates of domian image points(Xi,Yi)
        range_point: An n x 2 array containing coordinates of range image points(Xi',Yi')
    Output: A 3 x 3 Homography matrix
    '''
    # Find num of points provided
    n = Domain_pts.shape[0]
    #Initialize A Design matrix having size of 2n x 8
    A = np.zeros((2*n,8))
    #Reshape the Range_pts to 1D vector of size 2*num_pts x 8
    y = Range_pts.reshape((2*n, 1))
    #Loop through all the points provided and stack them vertically, this will result in 2n x 8 Design
matrix
    for i in range (n):
        A[i*2:i*2+2]=Get_A_matrix(Domain_pts[i],Range_pts[i])
    '''
    Compute the h vector (2n x 1) by using least sqaures solution. In case we have unique
    solution from 4 points, we can directly inverse the A design matrix. But for the overdetermined
    system, we can use the least sqaures estimation.
    '''
    h=np.dot(np.linalg.inv(np.dot(A.T,A)),np.dot(A.T,y))
    # Add the last element as 1 in the h vector to obtain HC. Now h will be 2*n+1 x 1 vector.
    h = np.concatenate((h,1), axis=None)
    #Reshape the h vector to H homography matrix
```

```
    H = h.reshape((3,3))
    return H


def Get_A_matrix(domain_point,range_point):
    '''
    function for generating a 2 x 8 design matrix needed to compute Homography
    Inputs:
        domain_point: Coordinates of a point in the domain image (x,y)
        range_point: Coordinates of corresponding point in the range image (x',y')
    Output: A 2 x 8 design matrix
    '''
    #Extract the x and y coordinates from a point pair
    x,y=domain_point[0], domain_point[1]
    xr,yr=range_point[0], range_point[1]
    #  Make A matrix
    A=np.array([[x,y,1,0,0,0,-x*xr,-y*xr],[0,0,0,x,y,1,-x*yr,-y*yr]])
    return A
```

B. Code for computing the homography matrix using vanishing line method (projective distortion removal):

```
def find_VL_homography (Domain_pts):

    # PQ line is parallel to RS (First Pair)
    # PQ Line
    PQ = np.cross(np.append(Domain_pts[0],1),np.append(Domain_pts[1],1))
    # RS Line
    RS = np.cross(np.append(Domain_pts[3],1),np.append(Domain_pts[2],1))

    # First Vanishing point
    Vanish_1 = np.cross(PQ,RS)
    #From HC to physical space
    Vanish_1 = Vanish_1/Vanish_1[2]

    # PR line is parallel to QS
    PR = np.cross(np.append(Domain_pts[0],1),np.append(Domain_pts[3],1))
    QS = np.cross(np.append(Domain_pts[1],1),np.append(Domain_pts[2],1))

    # First Vanishing point
    Vanish_2 = np.cross(PR,QS)
    #From HC to physical space
    Vanish_2 = Vanish_2/Vanish_2[2]

    # Vanishing line from Vanishing points
    Van_line = np.cross(Vanish_1,Vanish_2)
    #HC to Physical
```

```
    Van_line = Van_line/Van_line[2]

    #Compute the Homography matrix
    H_VL = np.identity(3)
    H_VL[2,:] = Van_line

    return H_VL
```

C. Code for computing the homography matrix using angle-to-angle correspondence (affine distortion removal):

```
def find_Aff_homography(DomPts_ProjectCorr):
    '''
    Function for correcting the affine distortion in the images using
    real-world orthogonal lines. This function needs to be used for computing
    the homography after correctiing the projective transformation.
    '''
    # Initialize the A and S matrices having 2 x 2 size.
    A = np.zeros((2,2))
    S_mat = np.ones((2,2))
    # PQ line is perpendicular to QS (First Pair)
    # PQ Line
    PQ_l1 =
np.cross(np.append(DomPts_ProjectCorr[0],1),np.append(DomPts_ProjectCorr[1],1))
    PQ_l1 = PQ_l1/PQ_l1[2]  #HC to physical
    # QS Line
    QS_m1 =
np.cross(np.append(DomPts_ProjectCorr[1],1),np.append(DomPts_ProjectCorr[2],1))
    QS_m1 = QS_m1/QS_m1[2]

    # PS line is perpendicular to QR (diagonal pair)
    PS_l2 =
np.cross(np.append(DomPts_ProjectCorr[0],1),np.append(DomPts_ProjectCorr[2],1))
    PS_l2 = PS_l2/PS_l2[2]
    QR_m2 =
np.cross(np.append(DomPts_ProjectCorr[1],1),np.append(DomPts_ProjectCorr[3],1))
    QR_m2 = QR_m2/QR_m2[2]

    #Create A and B matrices
    A[0,:] = [PQ_l1[0]*QS_m1[0], PQ_l1[0]*QS_m1[1] + PQ_l1[1]*QS_m1[0]]
    A[1,:] = [PS_l2[0]*QR_m2[0], PS_l2[0]*QR_m2[1] + PS_l2[1]*QR_m2[0]]

    Y = np.array([-PQ_l1[1]*QS_m1[1], -PS_l2[1]*QR_m2[1]])
    # Computing the S matrix
    S = np.linalg.inv(A)@Y  #No need of least squares as A is 2 x 2
    S_mat[0,:] = [S[0], S[1]]
    S_mat[1,0] = S[1]
```

```
    # Get the matrix A
    h=Decompose_S_mat(S_mat)
    #Compute the Homography matrix
    H_Aff = np.identity(3)
    H_Aff[0:2,0:2] = h
    return H_Aff

def ProjectiveCorr_Points(H_VanishLine,DomPts):
    '''
    Function for applying the homography calculated using the vanishing line method on the
    image coordinates picked for removing the affine distortion. Then supply these projective
    corrected coordinates to the function "find_Aff_homography"

    '''
    #Apply the projective homograpy
    P_pr_corr = H_VanishLine@np.append(DomPts[0],1)
    Q_pr_corr = H_VanishLine@np.append(DomPts[1],1)
    S_pr_corr = H_VanishLine@np.append(DomPts[2],1)
    R_pr_corr = H_VanishLine@np.append(DomPts[3],1)

    #Convert from HC to Physical space
    P_pr_corr = np.floor(P_pr_corr/P_pr_corr[2])
    Q_pr_corr = np.floor(Q_pr_corr/Q_pr_corr[2])
    R_pr_corr = np.floor(R_pr_corr/R_pr_corr[2])
    S_pr_corr = np.floor(S_pr_corr/S_pr_corr[2])


    dom_pts_corr=np.array([P_pr_corr[0:2],Q_pr_corr[0:2],S_pr_corr[0:2],R_pr_corr[0:2]])
    # Get the affine
    H_Aff = find_Aff_homography(dom_pts_corr)
    return H_Aff
```

D. Code for computing the homography matrix using one-step method:

```
def find_homography_one(Domain_pts):
    '''
    Find the homography using 5 pairs of orthogonal lines (Theta=90) using one-step method
    '''
    #Initialize A Design matrix having size of 5 x 5 and S matrix of size 2x2
    A=np.zeros((5,5))
    Y=np.zeros((5,1))
    S_mat = np.zeros((2,2))

    # PQ line is perpendicular to PR (First Pair)
    PQ_l1 = np.cross(np.append(Domain_pts[0],1),np.append(Domain_pts[1],1))
    PR_m1 = np.cross(np.append(Domain_pts[0],1),np.append(Domain_pts[3],1))
    PQ_l1 = PQ_l1/PQ_l1[2]  #HC to physical
```

```
PR_m1 = PR_m1/PR_m1[2]

# PR line is perpendicular to RS (2nd Pair)
PR_l2 = np.cross(np.append(Domain_pts[0],1),np.append(Domain_pts[3],1))
RS_m2 = np.cross(np.append(Domain_pts[3],1),np.append(Domain_pts[2],1))
PR_l2 = PR_l2/PR_l2[2]  #HC to physical
RS_m2 = RS_m2/RS_m2[2]

# QS line is perpendicular to SR (3rd Pair)
QS_l3 = np.cross(np.append(Domain_pts[1],1),np.append(Domain_pts[2],1))
SR_m3 = np.cross(np.append(Domain_pts[2],1),np.append(Domain_pts[3],1))
QS_l3 = QS_l3/QS_l3[2]  #HC to physical
SR_m3 = SR_m3/SR_m3[2]

# PQ line is perpendicular to QS (4th Pair)
PQ_l4 = np.cross(np.append(Domain_pts[0],1),np.append(Domain_pts[1],1))
QS_m4 = np.cross(np.append(Domain_pts[1],1),np.append(Domain_pts[2],1))
PQ_l4 = PQ_l4/PQ_l4[2]  #HC to physical
QS_m4 = QS_m4/QS_m4[2]

# PS line is perpendicular to RQ (5th Pair)
PS_l5 = np.cross(np.append(Domain_pts[0],1),np.append(Domain_pts[2],1))
QR_m5 = np.cross(np.append(Domain_pts[1],1),np.append(Domain_pts[3],1))
PS_l5 = PS_l5/PS_l5[2]  #HC to physical
QR_m5 = QR_m5/QR_m5[2]
# Get the A and Y matrices for each pair of line
A[0,:],Y[0] = Get_A_Y_vectors_one(PQ_l1,PR_m1)
A[1,:],Y[1] = Get_A_Y_vectors_one(PR_l2,RS_m2)
A[2,:],Y[2] = Get_A_Y_vectors_one(QS_l3,SR_m3)
A[3,:],Y[3] = Get_A_Y_vectors_one(PQ_l4,QS_m4)
A[4,:],Y[4] = Get_A_Y_vectors_one(PS_l5,QR_m5)

#Calculate and normalze C @ inf matrix
c_inf=np.linalg.inv(A)@Y
c_inf = c_inf/np.max(c_inf)
# Computing the S matrix
S_mat[0,:] = [c_inf[0],c_inf[1]/2]
S_mat[1,:] = [c_inf[1]/2,c_inf[2]]
# Get the matrix A
h=Decompose_S_mat(S_mat)
# Computer the vector v
v = (np.linalg.inv(h))@(np.array([c_inf[3]/2,c_inf[4]/2]))
#Compute the Homography matrix
H_one = np.zeros((3,3))
H_one[0]=np.append(h[0,:], 0)
H_one[1]=np.append(h[1,:], 0)
```

```
    H_one[2]=np.append(v, 1)

    return H_one


def Get_A_Y_vectors_one(l,m):
    '''
    function for generating a 5 x 1 design matrix (A) and a Y vector
    needed to compute Homography
    Inputs:
       Two orthogonal lines
    Output: A 5 x 1 design matrix and 5x 1 Y vector
    '''
    # Extract the coordinates of two lines
    l1,l2,l3=l[0], l[1],l[2]
    m1,m2,m3=m[0], m[1],m[2]
    # Make A matrix
    A = np.array([l1*m1, 0.5*(l2*m1 + l1*m2), l2*m2, 0.5*(l1*m3 + l3*m1), 0.5*(l3*m2 +
l2*m3)])
    Y = -l3*m3
    return A,Y
```

E. Function for decomposing the S matrix (called by code in section C and D):

```
def Decompose_S_mat(S_mat):
    #SVD decomposition of S matrix
    U, D, V = np.linalg.svd(S_mat, full_matrices=True)
    #Diagonlize the D matrix
    Ss=np.diag(D)
    D=np.sqrt(Ss)   #Sqaure root
    # Matrix A
    h = (V@D)@V.T
    return h
```

F. Code for computing the scale factor and applying the warping to images:

```
def Bounds_Undistorted(Homography,image):
    #Shape of the distorted image
    image_shape = image.shape

    #Distorted Homogeneous Coordinates of image Bounds
    ImgP= np.array([0,0,1])     # Top left corner of image (X,Y,1)
    ImgQ= np.array([image_shape[1],0,1])    # Top right corner
    ImgS = np.array([image_shape[1],image_shape[0],1]) #Bottom right
    ImgR = np.array([0,image_shape[0],1])   #bottom left

    #Apply the homography on the distroted image bounds to obtain the Corrected image
bounds
    WorldP = np.dot(Homography,ImgP)
```

```
    WorldP = WorldP/WorldP[2]
    WorldQ = np.dot(Homography,ImgQ)
    WorldQ = WorldQ/WorldQ[2]
    WorldS = np.dot(Homography,ImgS)
    WorldS = WorldS/WorldS[2]
    WorldR = np.dot(Homography,ImgR)
    WorldR = WorldR/WorldR[2]

    #Find the extreme points of the corrected image bounds
    max_point = np.maximum(np.maximum(np.maximum(WorldP ,WorldQ ), WorldS), WorldR)
    min_point = np.minimum (np.minimum (np.minimum(WorldP,WorldQ), WorldS), WorldR)
    #Find the coordinates of cextreme points of corrected image bounds
    xmax,ymax = max_point[0],max_point[1]
    xmin,ymin = min_point[0],min_point[1]
    # New width and height of corrected image
    width_corr = xmax-xmin
    height_corr =ymax-ymin
    #Compute the GSD in x and y direction. This should be roughly equal to the (width_pixels/physical_width)
    scalex = image_shape[1]/width_corr
    scaley = image_shape[0] / height_corr

    #Scale the new image using max scale
    scale=max(scalex,scaley)

    #Create an empty corrected image that will be further filled
    corrected_image = np.zeros((int(np.round(height_corr*scale)), int(np.round(width_corr*scale)), 3),dtype='uint8')

    return xmin, ymin, scale, corrected_image

def get_grid(x, y):
    #Get the array of coordinates of image pixels
    pix_coords = np.indices((x, y))
    # Flatten coordinates into 2*width*height
    pix_coords = pix_coords.reshape(2, -1)
    # Make the pixel coordinates to HC
    pix_coords=np.append(pix_coords,np.full((1,pix_coords.shape[1]),1),axis=0)
    return pix_coords

def transform_image(xmin, ymin, scaleFactor, undis_image,H_inv,image_org):
        '''
        Function responsible for the transforming the image
        This is the vectorized implementation of HW2 code
```

```
'''
#Get the height and width of undistorted empty Image
height, width = undis_image.shape[:2]
# Get the coordinates
coords = get_grid(width, height)
x_domain, y_domain = coords[0], coords[1]
# Scale while warping + add the min limits to coordinates of undistorted empty image
coords = coords.astype(np.int)

coords[0]=(coords[0]/scaleFactor) + xmin
coords[1]=(coords[1]/scaleFactor) + ymin
#Apply the inverse Homography to the coordinates
Pix_range = H_inv@ coords
#Convert from the range image pixel HC to physical space
Pix_range =Pix_range/Pix_range[2]
Pix_range = np.floor(Pix_range).astype(np.int)
# Coordinates of range image
xcoord_range, ycoord_range = Pix_range[0, :], Pix_range[1, :]

#print(min(x_ori),max(x_ori))
#print(min(xcoord2),max(xcoord2))

# Find the pixels of distroted image that are withinbounds
indices = np.where((xcoord_range >= 0) & (xcoord_range < image_org.shape[1]) &
        (ycoord_range >= 0) & (ycoord_range < image_org.shape[0]))
# pixel coordinates on both domain and range image which are within bounds
xpix, ypix = x_domain[indices], y_domain[indices]
xpix2, ypix2 = xcoord_range[indices], ycoord_range[indices]
# Map the image
undis_image[ypix, xpix] = image_org[ypix2,xpix2]
return undis_image
#return (xpix,ypix,xpix2,ypix2)
```