Computer Vision HW2 Report

1. Fundamental Matrix Estimation from Point Correspondences

a. Eight-Point Algorithm :

p is the Projected point on the image plane of image1, where $p^T = (u, v, 1)$

p' is the Projected point on the image plane of image2, where $p'^T = (u', v', 1)$

F is the fundamental matrix between image plane 1 and 2

F = $\begin{pmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{pmatrix}$

According to the relationship between image plane 1 and 2, we have $p^T F p' = 0$

$$(u, v, 1) \begin{pmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{pmatrix} \begin{pmatrix} u' \\ v' \\ 1 \end{pmatrix} = 0$$

And the matrix $p^T F p' = 0$ can be substitute into

$$(uu', uv', u, vu', vv', v, u', v', 1) \begin{pmatrix} F_{11} \\ F_{12} \\ F_{13} \\ F_{21} \\ F_{22} \\ F_{23} \\ F_{31} \\ F_{32} \\ F_{33} \end{pmatrix} = 0$$

Now we have 46 points (P) in image 1, and 46 points (P') in image 2

where $P_i^T = (u_i, v_i, 1)$, i = 1~46 and $P_i'^T = (u_i', v_i', 1)$, i = 1~46

the point P_i in image plane 1 corresponds to the point P_i' in image plane 2

Then we use the points P and P' to solve the fundamental matrix F by solving the linear system $Af = 0$

$$A = \begin{pmatrix} u_1u_1' & u_1v_1' & u & vu_1' & v_1 & u_1' & v_1' & 1 \\ u_2u_2' & u_2v_2' & u & vu_2' & v_2 & u_2' & v_2' & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_{46}u_{46}' & u_{46}v_{46}' & u & vu_{46}' & v_{46} & u_{46}' & v_{46}' & 1 \end{pmatrix} \quad f = \begin{pmatrix} F_{11} \\ F_{12} \\ F_{13} \\ F_{21} \\ F_{22} \\ F_{23} \\ F_{31} \\ F_{32} \\ F_{33} \end{pmatrix}$$

We use the least square method to find the solution of f.

a. First, we use the singular decomposition of Matrix A to find the approximation solution of f.

The approximation solution of f is the last column of V, where $A = SVD(A) = U \sum V^T$

Thus we have $f = lastColumn(V)$

Now we get the approximated fundamental matrix F, which is denoted as $\hat{F}$

```
# construct Matrix A
for i in range(n):
  u, v = points1[i].x, points1[i].y
  u_, v_ = points2[i].x, points2[i].y
  A[i] = np.array([u*u_, u*v_, u, v*u_, v*v_, v, u_, v_, 1])

# calculate Approximation F (F_) By least square method
# the least square solution is equal to
U0, s0, v_T = np.linalg.svd(W, full_matrices=True)
F_ = np.array([np.array([v_T[-1][0], v_T[-1][1], v_T[-1][2]]),
               np.array([v_T[-1][3], v_T[-1][4], v_T[-1][5]]),
               np.array([v_T[-1][6], v_T[-1][7], v_T[-1][8]])])
```

b. Find the Estimated F (F) by enforcing the rank 2 constraint

$$\text{Find F that minimizes} \quad \left\| \mathrm{F} - \hat{\mathrm{F}} \right\|_{\text{Frobenius norm (*)}} = 0$$

$$\text{Subject to det(F)=0}$$

$$F = \begin{pmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{pmatrix} = U \begin{pmatrix} s0 & 0 & 0 \\ 0 & s1 & 0 \\ 0 & 0 & 0 \end{pmatrix} V^T$$

, Where $SVD(\hat{F}) = U \begin{pmatrix} s0 & 0 & 0 \\ 0 & s1 & 0 \\ 0 & 0 & s2 \end{pmatrix} V^T$

Now we get the Fundamental Matrix F

```
'''
claculate Fundemantal Matrix F
To calculte ||F - F_||=0 Subject to det(F)=0 (rank(F)=2)
Solved By SVD, SVD(F_)=U*S*V_T
F = U*S_*V_T
S_ : 1.S_[i][j]==S[i][j] when i!=2 and j!=2 2.S[2][2]=0(Since Rank(F)=2)
'''

S1 = np.zeros((3,3))
U1, s1, V1_T = np.linalg.svd(F_, full_matrices=True)
S1[0][0] = s1[0]
S1[1][1] = s1[1]
S1[2][2] = 0 # Enforce Rank 2 Constraint
F = np.matmul(np.matmul(U1, S1), V1_T)
```

- The Result of the Fundemantal Matrix

```
Fundemantal Matrix (unnormalized)
[[-5.63087200e-06 -2.77622828e-05  1.07623595e-02]
 [ 2.74976583e-05 -6.74748522e-06 -1.22519240e-02]
 [-6.42650411e-03  1.52182033e-02 -9.99730547e-01]]
```

b. Normalized Eight-Point Algorithm

- The flow of Normalized Eight-Point Algorithm

  Step 1 : Normalize the points in both image plane 1 and 2.

  where points1'[i] = A1*points1[i] and points2'[i] = A2*points2[i]

A1 is the transformation matrix of points in image 1

A2 is the transformation matrix of points in image 2

Step 2 : Run the Eight Point Algorithm mentioned in the previous question to get Normalized Fundamental Matrix F_q

Step 3 : Denormalized the Fundamental Matrix F

where F = A1 * F_q * A2

```
# Step 1
normalized_pts1, A1 = normalizedPoint(points1)
normalized_pts2, A2 = normalizedPoint(points2)
# Step 2
F_q = EightPointAlgo(normalized_pts1, normalized_pts2)
# Step 3
F = np.matmul(np.matmul(A1.T, F_q), A2)
```

- Normalize Points

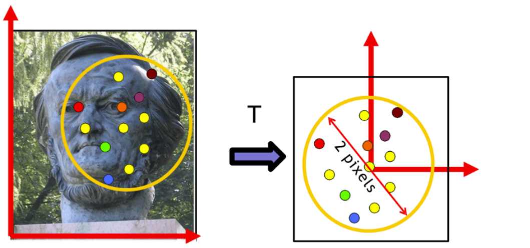
Step 1 : Find the Centroid (質心) of all points

Step 2 : Find the mean distance between the points to centroid

Step 3 : Transform the point ⇒ The transformation is composed of Translation and Scaling

- Translation : 將Origin Point (0, 0)平移至Centroid Point， 其它point隨之平移

- Scaling : 將mean distance scale到 $\sqrt{2}$

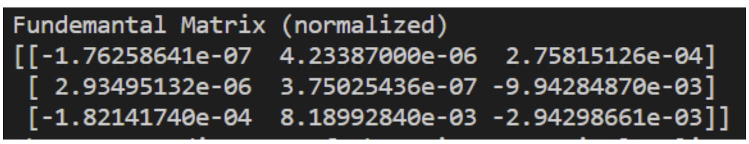


Coordinate system of the image before applying T

Coordinate system of the image after applying T

```
# Step 1 - Find Centrid
centroid_x = np.sum([pt.x for pt in point])/len(point)
centroid_y = np.sum([pt.y for pt in point])/len(point)

# Step 2 - Find mean distance
distance_sum = 0
for pt in point:
  distance_sum += math.sqrt((pt.x - centroid_x)**2 + (pt.y - centroid_y)**2)
mean_distance = distance_sum/len(point)

# Step 3 - Calculate the transformation matrix
scale = math.sqrt(2)/mean_distance
T = np.array([np.array([scale, 0, -scale*centroid_x]),
              np.array([0, scale, -scale*centroid_y]),
              np.array([0, 0, 1])])

# Step 3 - Normalize the points by applying transformation of all points
for i in range(n):
  pt = np.array([point[i].x, point[i].y, 1])
  new_pt = np.matmul(T, pt)
  normalized_point.append(Point(new_pt[0]/new_pt[2], new_pt[1]/new_pt[2]))

# The normalized_point is a set of points after normalization
# T is the transformation matrix of normalization
return normalized_point, T
```

- The result of the Fundemantal Matrix



```
Fundemantal Matrix (normalized)
[[-1.76258641e-07  4.23387000e-06  2.75815126e-04]
 [ 2.93495132e-06  3.75025436e-07 -9.94284870e-03]
 [-1.82141740e-04  8.18992840e-03 -2.94298661e-03]]
```

c. Plot the epipolar line and Calculate the average distance between the points to corresponding epipolar line

- Plot the epipolar line

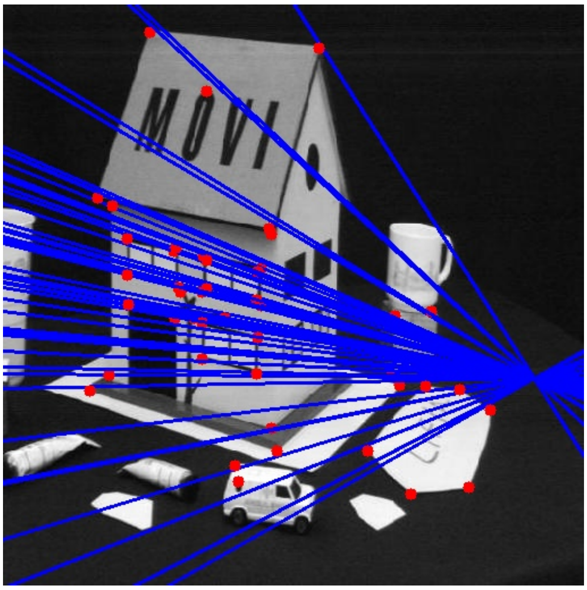


Image1 (before normalization)

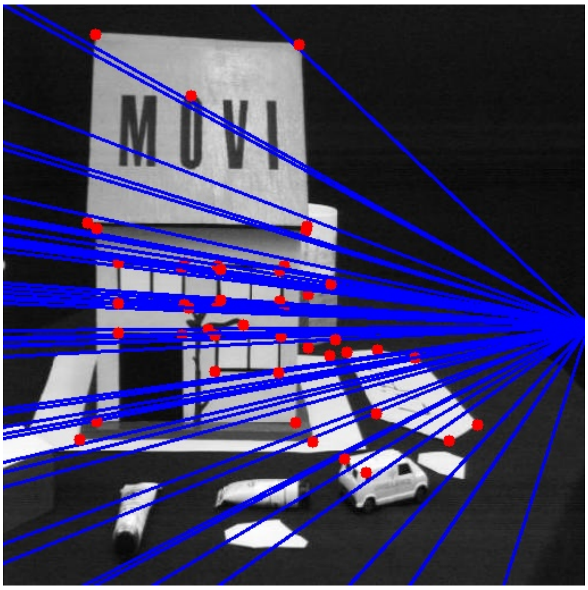


Image2 (before normalization)

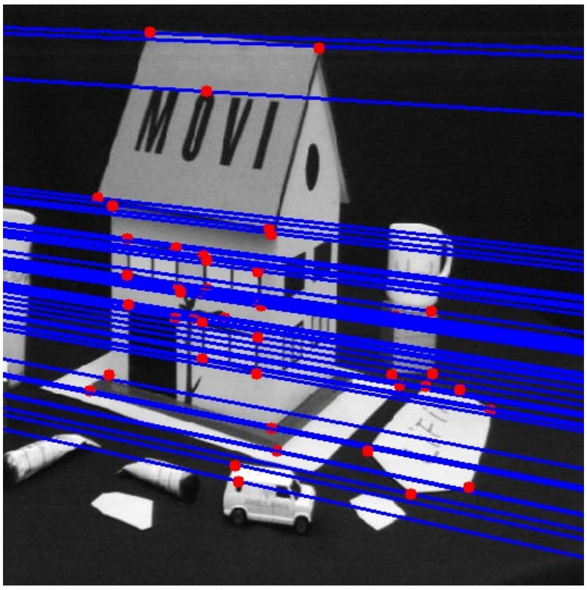


Image1 (after normalization)

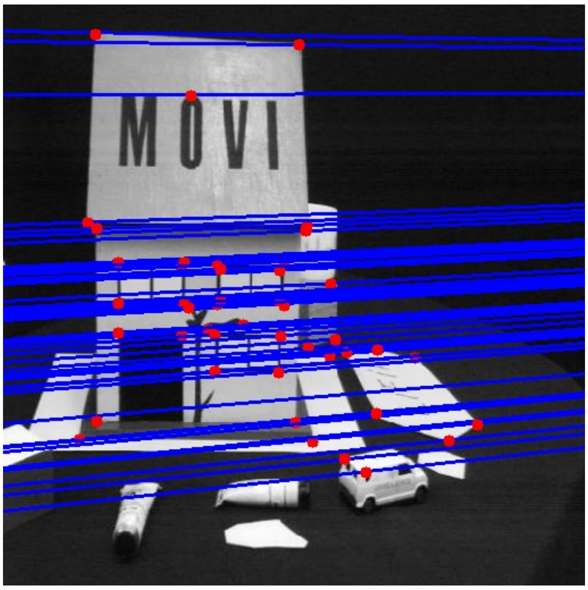


Image2 (after normalization)

- Calculate the average distance between the points to corresponding epipolar line

```
the Average distance of the points to epipolar line in image 1 (unnormalized) is  9.701438829424138
the Average distance of the points to epipolar line in image 1 (normalized) is  0.8894960616888739
the Average distance of the points to epipolar line in image 2 (unnormalized) is  14.568227190477133
the Average distance of the points to epipolar line in image 2 (normalized) is  0.8917172367782387
```

2. Homography Transform

a. Implemenatation of computing Homography Matrix

The formula Homography transfomation is p' = Hp

$$\begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} \cong \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

Where p is the point before transformation,

p' is the point after transformation,

H is the Homography Transformation Matrix

Then we can derive the following equation by p' = Hp

$$\begin{aligned} x_i' &= \frac{h_{00}x_i + h_{01}y_i + h_{02}}{h_{20}x_i + h_{21}y_i + h_{22}} \\ y_i' &= \frac{h_{10}x_i + h_{11}y_i + h_{12}}{h_{20}x_i + h_{21}y_i + h_{22}} \end{aligned}$$

$$\begin{aligned} x_i'(h_{20}x_i + h_{21}y_i + h_{22}) &= h_{00}x_i + h_{01}y_i + h_{02} \\ y_i'(h_{20}x_i + h_{21}y_i + h_{22}) &= h_{10}x_i + h_{11}y_i + h_{12} \end{aligned}$$

$$\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x_i'x_i & -x_i'y_i & -x_i' \\ 0 & 0 & 0 & x_i & y_i & 1 & -y_i'x_i & -y_i'y_i & -y_i' \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Now, we will solve the Homography Equation by finding n points in original image, and n corrspondence points in new image

$$\underbrace{\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1'x_1 & -x_1'y_1 & -x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y_1'x_1 & -y_1'y_1 & -y_1' \\ & & & & \vdots & & & & \\ x_n & y_n & 1 & 0 & 0 & 0 & -x_n'x_n & -x_n'y_n & -x_n' \\ 0 & 0 & 0 & x_n & y_n & 1 & -y_n'x_n & -y_n'y_n & -y_n' \end{bmatrix}}_{\substack{\mathbf{A} \\ 2n \times 9}} \underbrace{\begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix}}_{\substack{\mathbf{h} \\ 9}} = \underbrace{\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}}_{\substack{\mathbf{0} \\ 2n}}$$

Solving Homography Equation :

Step 1 : finding n points in original image, and n corrspondence points in new image

Note : n points in original image consist of the area we want to rectify in origin image.

Step 2 : Solve h by least square method. ⇒ The approximation solution of h is the one of the EigenVector of A, which correspond to the minimum EigenValue

Step 3 : After solving Ah = 0, we get the vector h. By reshaping matrix h, we will get the Homography Transformation Matrix H.

```
# Step 1
src_points = [Point(418, 800), Point(896, 1016), Point(886, 65), Point(435, 342)]
taget_points = [Point(500, 1200), Point(1200, 1200), Point(1200, 500), Point(500, 500)]
```

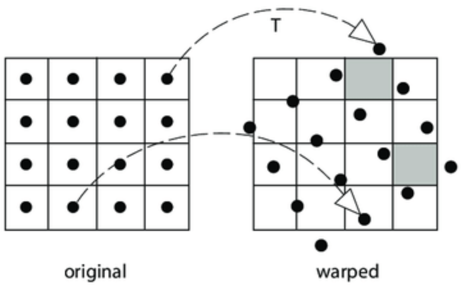
```
def ComputeHomography(points1, points2):
    '''
    Solving the Homographies
    find the solution of Ah = 0 via least square method
    to solve h w.r.t Ah = 0 is to minimize |Ah - 0|^2
    Homography h is equal to the eigenvector of A.T * A with the smallest eigenvalue

    where A.shape=(2n,9) h.shape=(9,1) n=len(point1)
    '''

    n = len(points1)

    A = np.zeros((2*n,9))
    h = np.zeros((9,1))
    H = np.zeros((3,3))

    # Construct the matrix A according to the Homoraphy Formula
    for i in range(n):
        pt1 = points1[i]
        pt2 = points2[i]
        A[2*i,:] = np.array([pt1.x, pt1.y, 1, 0, 0, 0, -(pt2.x)*(pt1.x), -(pt2.x)*(pt1.y), -(pt2.x)])
        A[2*i+1,:] = np.array([0, 0, 0, pt1.x, pt1.y, 1, -(pt2.y)*(pt1.x), -(pt2.y)*(pt1.y), -(pt2.y)])

    # Step 2 find eginvector w.r.t min EigenValue
    ATA = np.array(np.matmul(A.T, A))
    egi_val, egi_vector = np.linalg.eig(ATA)
    min_egival_idx = np.where(egi_val == np.amin(egi_val)) # find minium EgienValue of A.T * A
    h = egi_vector[:, min_egival_idx].flatten()

    # Step 3 find H (Homography Transform Matrix)
    H[0] = np.array([h[0], h[1], h[2]])
    H[1] = np.array([h[3], h[4], h[5]])
    H[2] = np.array([h[6], h[7], h[8]])

    print('Homography Transform Matrix:')
    print(H)

    return H
```

b. Implementation of Backward Warpping and Bilinear Interpolation

- Warping
  - Forward Warpping :

    Iterate through all the points in the original image

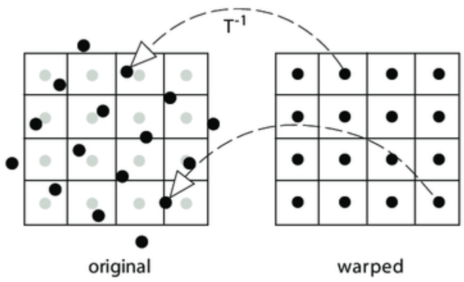
    For each point in original image, find the Correponding point in warped image by transformation matrix T



  - Backward Warpping :

    Iterate through all the points in the warped image

    For each point in warped image, find the Correponding point in original image by transformation matrix T^{-1}



- Bilinear Interpolation

- Using the pixel value of point (x1, y1) (x2, y1) (x1, y2) (x2, y2) to find the pixel value of Point P

  Note : pixel value of (x1, y1) is Q11, pixel value of (x2, y1) is Q21, pixel value of (x1, y2) is Q12, and pixel value of (x2, y2) is Q22

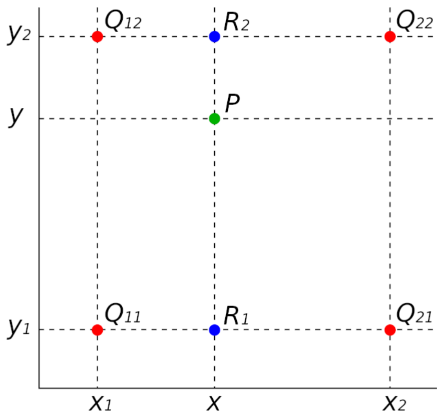
- First Calculate the Pixel value of (x, y1) and (x, y2)

  pixel value of (x, y1) = R1 = Q11 · (x2 – x) / (x2 – x1) + Q21 · (x – x1) / (x2 – x1)

  pixel value of (x, y2) = R2 = Q12 · (x2 – x) / (x2 – x1) + Q22 · (x – x1) / (x2 – x1)

- Then Calculate the Pixel value of Point P

  pixel value of P = R1 · (y2 – y) / (y2 – y1) + R2 · (y – y1) / (y2 – y1)



- Implementation of Backward warpping and Bilinear Interpolation (Some of the discriptions are in the comment of the code)

  Itreate through all the pixels in warped image

  - Step 1 : Find the Correponding point for the pixel in warped image(p') in original image (p) by transformation matrix H^{-1} $\Rightarrow p = H^{-1}p'$
  - Step 2 : Find the pixel value of p' (point in the warped image) by applying Bilinear Interpolation method to point p (point in original image found by Homography Transformation). Bilinear Interpolation method will calculate the pixel value of point p. And the pixel value of point p is equal to the pixel value of point p'.

```
def Backward_and_Bilinear_Wrapping(img, H):
    '''
    Implement Image wrapping through Backward Wrapping and Bilinear Interpolation

    input:
        img : image to be wrapping
        H(3*3) : Wrapping Matrix

    output:
        new_img : image after wrapping
    '''

    new_img = np.zeros((1800, 1800, 3))

    H_Inverse = np.linalg.inv(H)

    for i in range(new_img.shape[0]):
        for j in range(new_img.shape[1]):
            for k in range(new_img.shape[2]):
                '''
                Forward wrapping : use the point in origin image to find the correspond point in ouput image
                [new_x, new_y, new_z] = H * [x, y, 1]

                Backward wrapping : use the point in ouput image to find the correspond point in origin image
                [x, y, 1] = H_inverse * [new_x, new_y, new_z]

                note : [new_x, new_y, new_z] is in Homogeneous Coordinate
                '''

                # Implementation of BackWard Wrapping
                # Find the corresponding Point of warped image to the original image
                new_y, new_x = i, j
                x, y, z = np.matmul(H_Inverse, np.array([new_x, new_y, 1]))

                # convert Homogeneous Cordinate to Euclidean Coordinate
                x = x / z
                y = y / z
```

```
                round_x = int(round(x))
                round_y = int(round(y))

                ceil_x = int(ceil(x))
                ceil_y = int(ceil(y))

                # Bilinear Interpolation
                if round_x-1 >= 0 and round_x+1 < img.shape[1] and round_y-1 >= 0 and round_y+1 < img.shape[0]:

                    Q11 = img[round_y, round_x, k] # pixel value of Point(round_x, round_y)
                    Q21 = img[round_y, ceil_x, k]  # pixel value of Point(ceil_x, round_y)
                    Q12 = img[ceil_y, round_x, k]  # pixel value of Point(round_x, ceil_y)
                    Q22 = img[ceil_y, ceil_x, k]   # pixel value of Point(ceil_x, ceil_y)

                    if ceil_x-round_x > 0:
                        R1 = Q11*(ceil_x-x)/(ceil_x-round_x) + Q21*(x-round_x)/(ceil_x-round_x)  # pixel value of Point(X, round_y)
                        R2 = Q12*(ceil_x-x)/(ceil_x-round_x) + Q22*(x-round_x)/(ceil_x-round_x)  # pixel value of Point(X, ceil_y)
                    else:
                        R1 = Q11 # When ceil_x == round_x, R1 = Q11 = Q21
                        R2 = Q12 # When ceil_x == round_x, R2 = Q12 = Q22

                    if ceil_y-round_y > 0:
                        P = R1*(ceil_y-y)/(ceil_y-round_y) + R2*(y-round_y)/(ceil_y-round_y)     #pixel value of Point(X, y)
                    else:
                        P = R1 # When ceil_y == round_y, P = R1 = R2 (Since Q11 == Q12 and Q21 == Q22)
                    new_img[new_y, new_x, k] = P

    cv2.imwrite('Image.jpg', new_img[350:1350, 350:1350])

    return new_img
```

c. Specify a set of point correspondences for the source image of the Delta building and the target one .

Point in Source Image	Point in Warped Image
(418, 800)	(500, 1200)
(896, 1016)	(1200, 1200)
(886, 65)	(1200, 500)
(435, 342)	(500, 500)

d. Output the Homography Transform matrix

```
Homography Transform Matrix:
[[-2.41399098e-03 -1.18176986e-04  8.18197259e-01]
 [-1.20419102e-03 -9.45651319e-04  5.74930480e-01]
 [-1.31176813e-06 -1.05840287e-07  6.22059489e-05]]
```

e. Selected Image and Rectified Image

Rectified image

Selected image