

ECE 661 Homework 3

Suyash Ail
sail@purdue.edu

September 22, 2020

The task of this homework is to correct the projective and affine distortion in the images using point correspondence, two step and one step methods.

1 Logic

1.1 Point Correspondence Method

We use point to point correspondence between the distorted image plane and the world plane to estimate the homography between the world image and distorted image. Let \vec{x}_w be the homogeneous representation of a point in the world coordinates. Let \vec{x}_d be the homogeneous representation of the point in the distorted space (range). The relationship between the two points is represented by a non-singular 3x3 matrix H which is called the homography matrix.

$$\vec{x}_d = H\vec{x}_w \quad (1)$$

H is a linear mapping from one homogeneous coordinate to another.

$$H : \mathbb{R}^3 \longrightarrow \mathbb{R}^3$$

The matrix H is

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix}$$

Since the Homography matrix is homogeneous, the information is in the ratios, hence $h_{33}=1$.

Let us assume a point in the domain $\vec{x}_w = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$ and a corresponding point in the range \vec{x}_d

$$= \begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \end{bmatrix}.$$

Hence

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

Since the information is in the ratios, we get the x and y coordinates of the transformed points as

$$x' = \frac{x'_1}{x'_3} = \frac{h_{11}x_1 + h_{12}x_2 + h_{13}x_3}{h_{31}x_1 + h_{32}x_2 + x_3}$$

$$y' = \frac{x'_2}{x'_3} = \frac{h_{21}x_1 + h_{22}x_2 + h_{23}x_3}{h_{31}x_1 + h_{32}x_2 + x_3}$$

Dividing the right hand side by x_3 gives

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

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

$$\therefore h_{11}x + h_{12}y + h_{13} - h_{31}xx' - h_{32}yx' - x' = 0 \quad (2)$$

$$h_{21}x + h_{22}y + h_{23} - h_{31}xy' - h_{32}yy' + y' = 0 \quad (3)$$

Here we have two equations with 8 unknowns. We see that one pair of corresponding points gives two equations. Hence we need 4 such points.

Rewriting the equation in matrix form we get

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

This can be written as $AB=C$. Hence we get $B = A^{-1}C$. Reshaping the B vector gives the Homography matrix. Once the Homography is estimated, we use the inverse Homography to map the image from the distorted plane to the world plane to get rid of both projective and affine distortion at the same time.

1.1.1 Steps

1. Find the homography matrix H using the 4 given corner points of the two images.

$$\vec{x}_d = H \vec{x}_w \quad (4)$$

2. Since this Homography maps world coordinates to distorted plane, we will use the inverse Homography to map the image from distorted to world plane.

$$\vec{x}_w = H^{-1} \vec{x}_d \quad (5)$$

3. The projected points are not integer values, hence we round off the points and convert to integer. Replace the pixel values of the range image with the corresponding pixel values of the domain image.

2 Two Step Method

The second method of removing projective and affine distortion involves two steps. In the first step we remove the pure projective distortion from the distorted image. In the second step we remove the pure affine distortion from the projective corrected image. The pure projective distortion is corrected by using the Vanishing Line(VL) method. We pick 2 pairs of parallel lines from the distorted image. The cross product of one pair of parallel lines will give the Vanishing Point. Cross product of the two obtained Vanishing Points gives the Vanishing Line.

Step 1- Removing Projective Distortion using Vanishing Line Method

The task is to find the Homography matrix H that will map the VL to l_∞ .

Let $L1, M1$ and $L2, M2$ be the two pairs of parallel lines in the image. The parallel lines are obtained by performing cross product of points in homogeneous coordinates. The VP is obtained by taking cross product of the two lines.

$$P1 = L1 \times M1$$

$$P2 = L2 \times M2$$

Then the VL is obtained by cross product between $P1$ and $P2$.

$$VL = P1 \times P2$$

The Homography matrix which project VL to l_∞ is

$$H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & 1 \end{bmatrix} \quad (6)$$

Once the Homography is found we can apply the homography to the distorted image to get the projective corrected image.

Steps:

1. Pick 4 points from the distorted image using Software.
2. Find the vanishing line using the above equation. The VL is normalized($l_3 = 0$) before inserting into the Homography matrix.
3. Once we compute $H2$ we can use it to map the distorted image to projective corrected image.

$$\vec{x}_p = H2\vec{x}_d \quad (7)$$

4. Find the 4 coordinates of the image in world plane by using the $H2$ matrix. The coordinates will have to be scaled and translated in order to display the image correctly
5. Using the $H2^{-1}$ matrix project all the points from the world image to the distorted plane to find correspondence.
6. The projected points are not integer values, hence we round off the points and convert to integer. Replace the pixel values of the range image with the corresponding pixel values of the domain image.

Step 2-Removing Affine Distortion

Once we remove projective distortion we will see that parallel lines are preserved. But due to presence of affine distortion, the angles between lines are not preserved causing unequal scaling in the image. We will use two perpendicular lines in the projective corrected image plane.

If we have two orthognonal lines $L=[l_1, l_2, l_3]^T$ and $M=[m_1, m_2, m_3]^T$, then the angle between them can be determined by the cosine formula

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

C_{∞}^* is the dual degenerate conic at infinity.

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

Now let us assume a Homography H_c that will transform the world points into the affine plane. Then the lines will be mapped from the affine plane to world plane by

$$L = H_c^T L'$$

$$M = H_c^T M'$$

The conic C_{∞}^{**} is mapped by $H^T C_{\infty}^* H$. The lines selected are orthogonal in real world, so cosine value is 0. Hence on substituting the corresponding terms we get

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

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

On simplifying the equation we get

$$\begin{bmatrix} l_1 & l_2 & l_3 \end{bmatrix} \begin{bmatrix} AA^T & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \end{bmatrix}$$

Assume $S = AA^T$. Therefore we can write the matrix vector product as an equation with three unknowns.

$$s_{11}m'_1l'_1 + s_{12}(m'_1l'_2m'_2l'_1) + s_{22}m'_2l'_2 = 0$$

s_{22} can be set to 1 as only the ratios matter. So we only have two unknowns s_{12} and s_{11} . Hence two pairs of orthogonal lines should suffice the condition. Compute SVD of S to get A.

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

$$A = VDV^T$$

Steps:

1. H_c is the mapping of points from the world coordinates to the affine distorted plane. Hence to compute the transformation from affine to real we have to take H_c^{-1}

$$\vec{x}_w = H_c^{-1} \vec{x}_p$$

2. Map the 4 corners of the image to world coordinates using H_c^{-1} . This new coordinates will have to be translated and scaled in order to get the image dimension.
3. Using the H_c matrix project all the points from the world image to the affine distorted plane to find correspondence.
4. The projected points are not integer values, hence we round off the points and convert to integer. Replace the pixel values of the range image with the corresponding pixel values of the domain image.

2.1 One-Step Method

In one step method the Homography for both Projective and Affine are calculated in a single step. We use a similar approach as that was done in the Affine removal step. But the Homography matrix H is now

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

On simplifying matrix vector product as in two step method, we get

$$C'_\infty^* = \begin{bmatrix} AA^T & A\nu \\ A^T\nu & 1 \end{bmatrix} = H^T C_\infty^* H = \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix}$$

Therefore we get

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

$$L'^T \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix} M' = 0$$

On simplifying the equation and writing as a vector product we get,

$$\begin{bmatrix} l'_1 m'_1 & \frac{l'_1 m'_2 + l'_2 m'_1}{2} & l'_2 m'_2 & l'_1 m'_3 + l'_3 m'_1 & l'_2 m'_3 + l'_3 m'_2 & l'_3 m'_3 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = 0$$

f can be set to 1 as only the ratios matters. Hence we have 5 unknowns in 1 equation. So we need 5 such equations to get all the unknowns. The 5 equations can be stacked to form a matrix vector product whose solution is the null vector. Once the 5 unknowns are calculated,

we follow the same approach as in the affine removal task to get the A matrix and v . After that the homography can be calculated. Steps:

1. Pick 5 orthogonal lines from the original image.
2. Using the above equation get the 5×6 matrix. The null vector is computed using `sympy.nullspace()`. Will need to normalize it to get $f=1$.
3. Get the C_∞^* matrix from the 5 unknowns and use the same procedure as Affine Removal to compute A and v .

3 Results

3.1 Image 1

3.1.1 Input Image



Figure 1: Image 1

A bigger bounding box is selected as opposed to the given points as the results weren't satisfactory. The world coordinates are accordingly scaled (gap between windows considered equal to width and height)

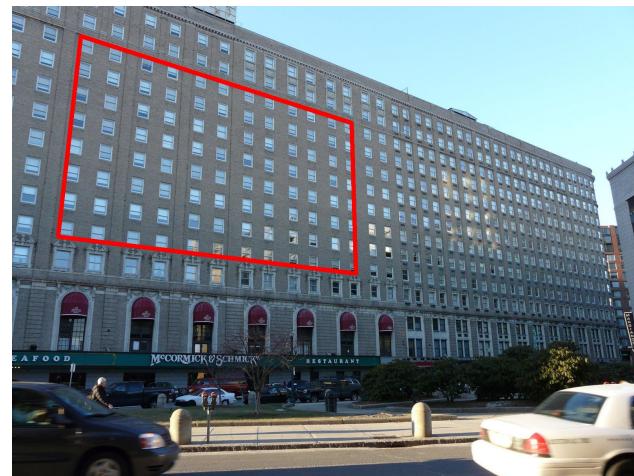


Figure 2: Bounding Box for 4 points

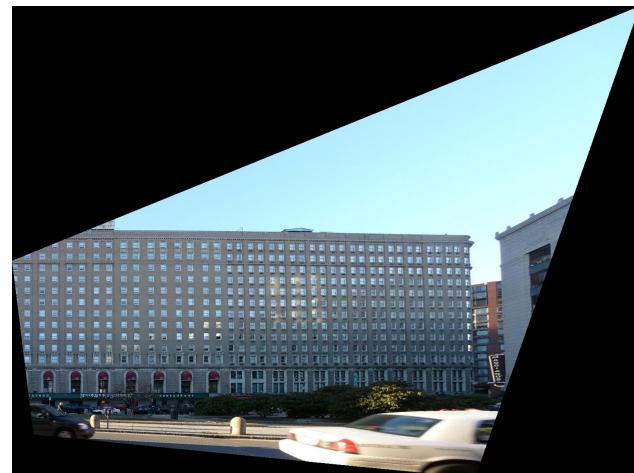


Figure 3: Distortion Removed using Point to Point Correspondence



Figure 4: Parallel lines for Projective Distortion Removal



Figure 5: Output after step1- Projective Corrected



Figure 6: Output after step2- Affine Corrected



Figure 7: Output using One Step Method

3.2 Image2



Figure 8: Image2

The 4 points are selected to be the outer edges of the window. This gave a better result as compared to the smaller one. The world coordinates have been scaled by 2.



Figure 9: Bounding box for 4 points



Figure 10: Distortion Removal using Point to point method

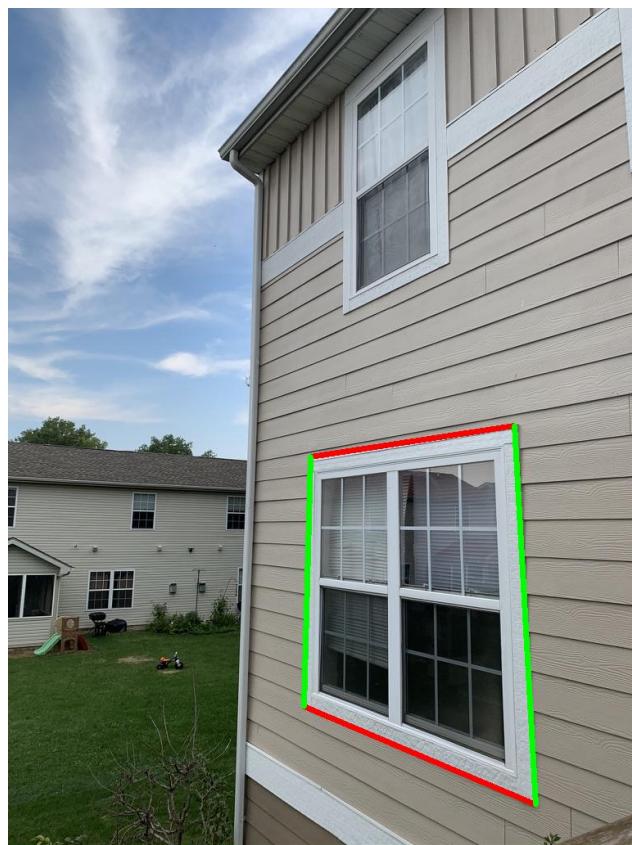


Figure 11: Parallel lines for Projective Distortion removal



Figure 12: Output image after Projective Correction(Scaled in latex to display full)

For the orthogonal lines, the window is assumed to be a square.

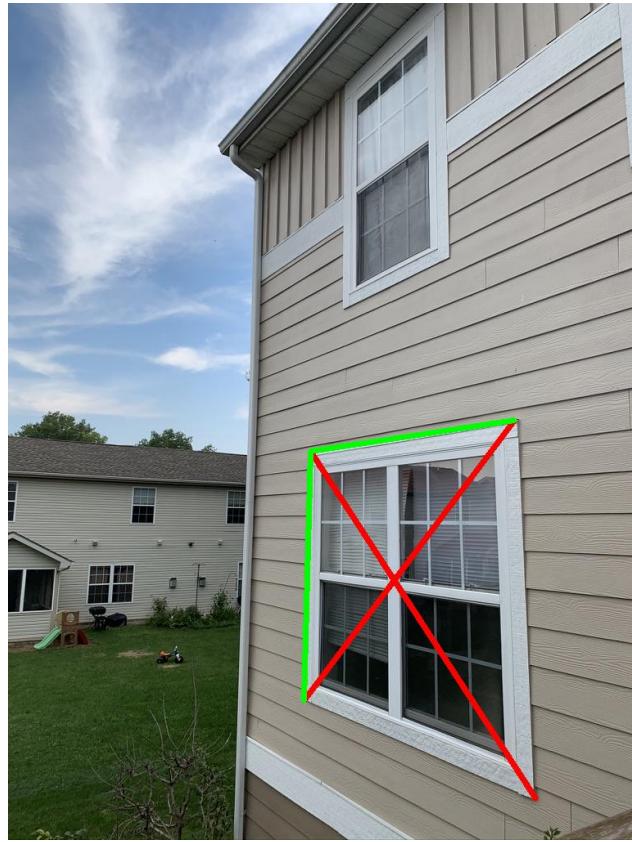


Figure 13: Orthogonal lines for Affine Distortion removal



Figure 14: Output after removing Affine Distortion



Figure 15: Distortion Removal using One Step Method

3.3 Image 3

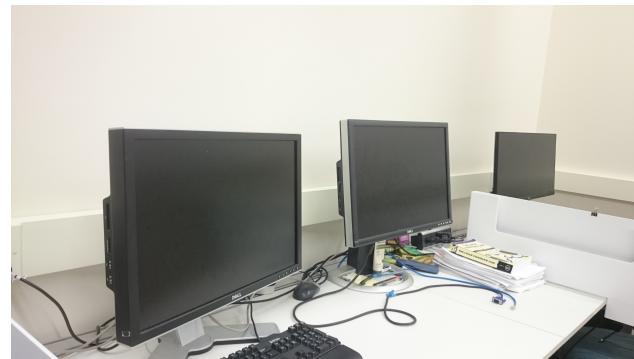


Figure 16: Image 3

The center monitor was chosen as the 4 point as it gave the best results as compared to other two.

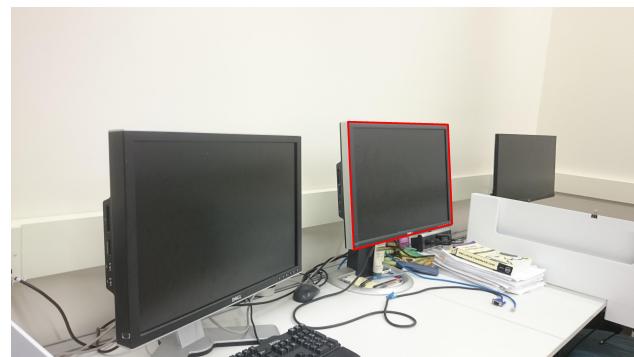


Figure 17: Bounding box for 4 points

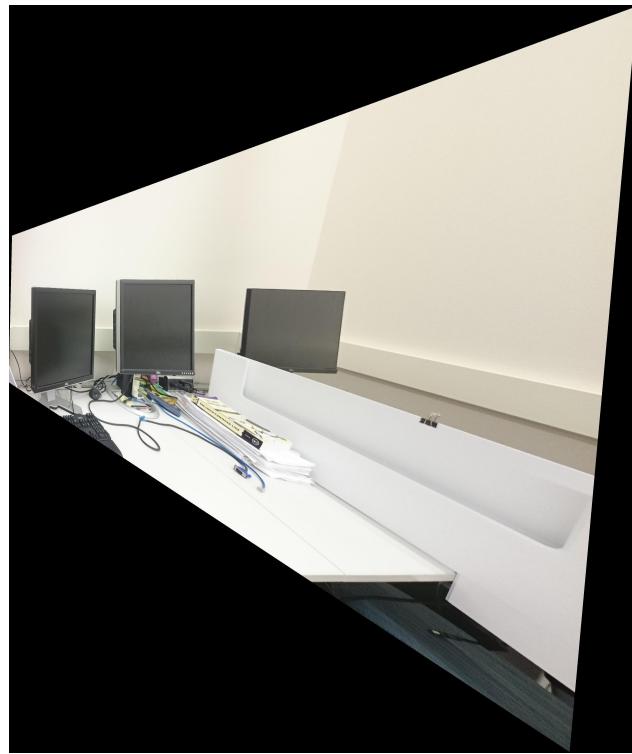


Figure 18: Distortion Correction using Point to Point Correspondence



Figure 19: Parallel Lines in the image



Figure 20: Image after removing Projective Distortion

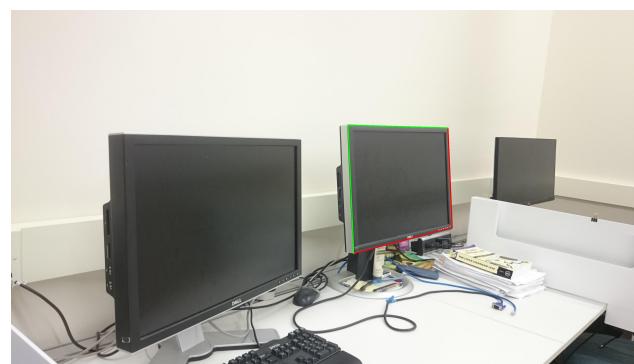


Figure 21: Orthogonal lines for Affine Removal

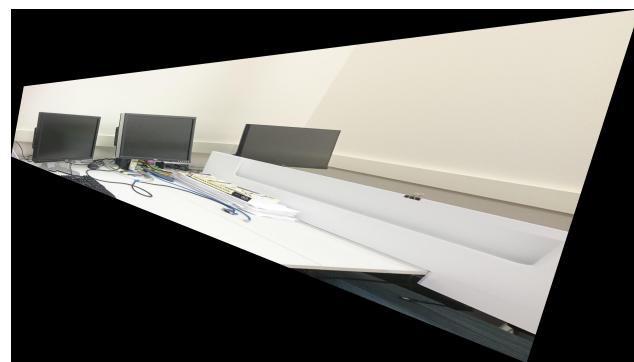


Figure 22: Affine Distortion Removed Image

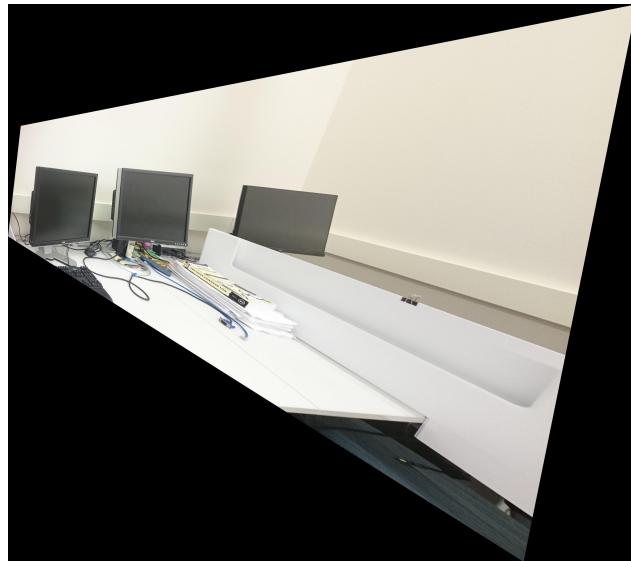


Figure 23: Distortion Removal using One step method

3.4 My Image 1

For this Image I have used the visual perception test board displayed at Ellis Island.



Figure 24: My Image 1

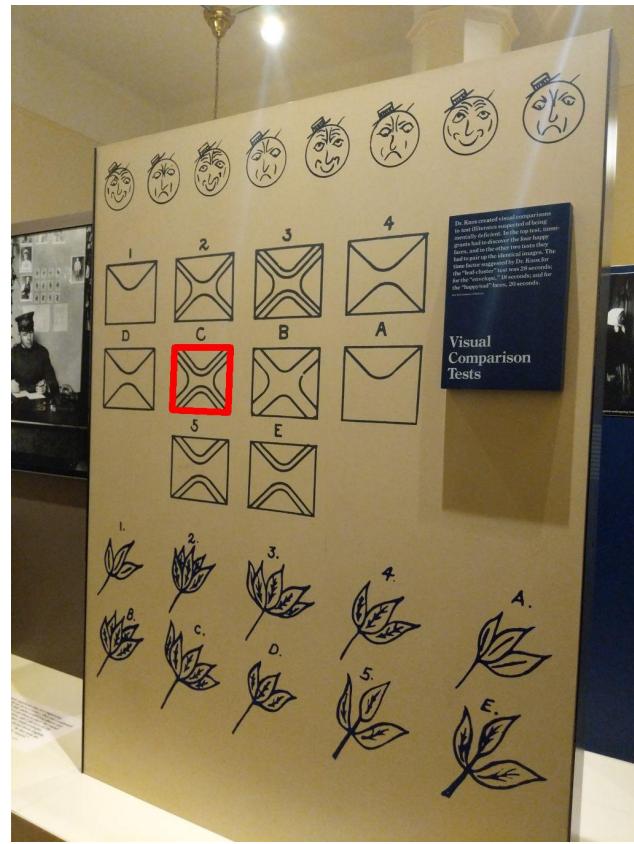


Figure 25: Bounding Box for 4 points



Figure 26: Distortion Removal using Point to Point Method

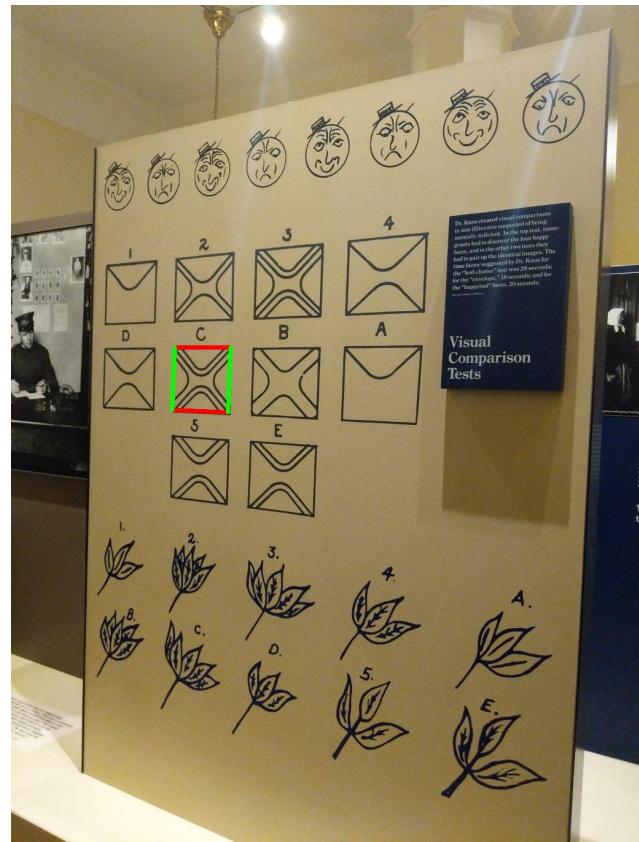


Figure 27: Parallel lines for Projective Removal

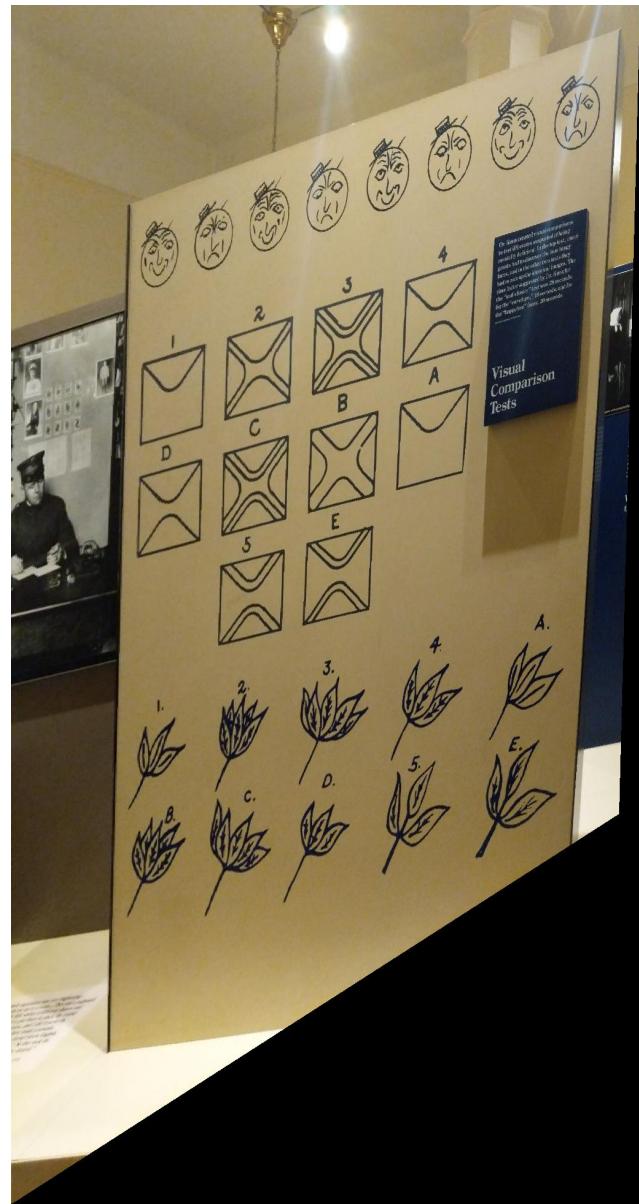


Figure 28: Projective Corrected Output

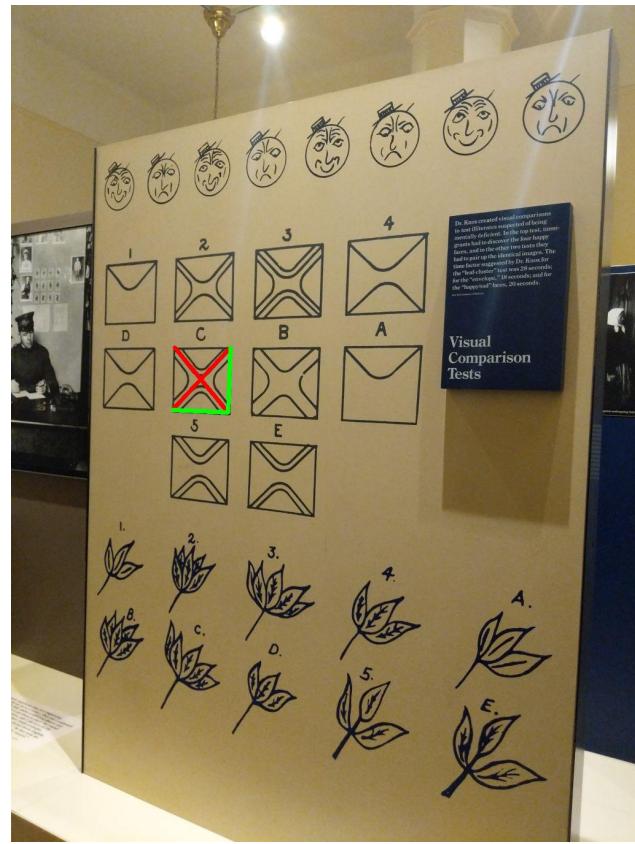


Figure 29: Orthogonal Lines for Affine Removal



Figure 30: Affine Corrected Output



Figure 31: Distortion Removal using One Step Method

3.5 My Image 2

This image was taken during my visit to Munich. It is a very famous cathedral that housed the first kings of Munich.



Figure 32: My Image 2

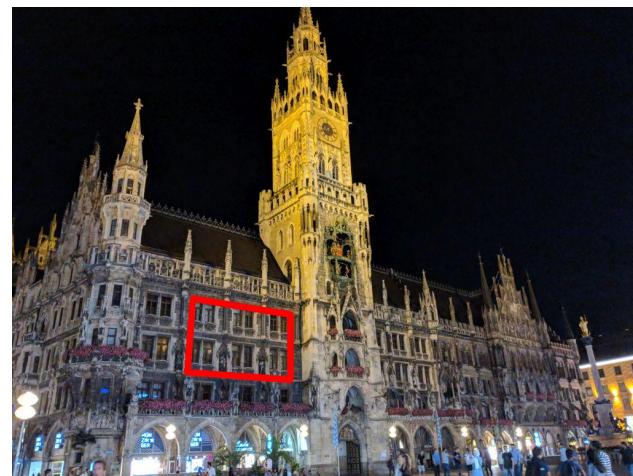


Figure 33: Bounding Box for 4 points



Figure 34: Distortion Removal using Point to Point Method



Figure 35: Parallel lines for Projective Distortion Removal

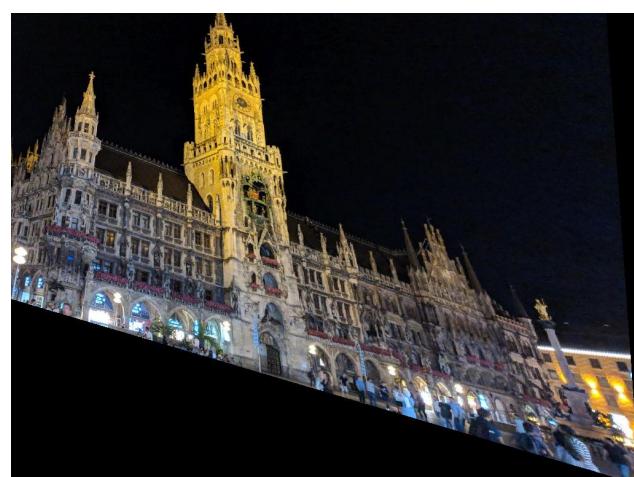


Figure 36: Projective Corrected Output



Figure 37: Perpendicular lines for Affine Distortion Removal



Figure 38: Affine Corrected Image

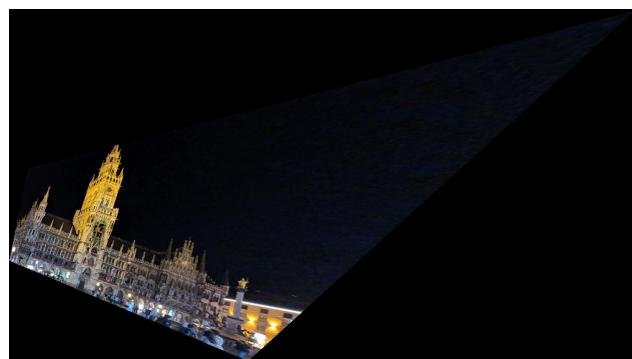


Figure 39: Distortion Removal using One Step Method

4 Observations

The point to point correspondence worked well on points near the bounding box but as we went far the distortions still remained. Also this approach is highly sensitive to the point

selection. After a lot of iterations on different points, the best points could be decided. The method however gives exceptional results if the points are correctly selected and cover a wide range in the image. The two step method was the most difficult to get working.

Point selection were equally important in this method. Getting the affine distortion removed image from the projective distortion removed image didn't result in good results. Hence the homographies were concatenated and applied to the original image which gave good results. One Step method was found to be the most robust among the three methods. The point

selection criterion though important but not as decisive as the other two methods provided some freedom in point selection. The only requirement was to ensure that the lines selected have to be perpendicular in real world coordinates. This method gave the best results on my own images.

5 Vectorization

I have tried vectorizing two instances of for loops to give a optimized code. Since this is my first time, the overall code isn't optimized and some images generated wrong outputs for the vectorized method as compared to for loops. Hence the code contains both vectorized and for loops.

```

1 img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
2 # using for loop for mapping image corners to world coordinates
3 for i in range(4):
4     new_coords=np.matmul(H_inv, img_coords[i].T)
5     new_coords=new_coords/new_coords[2]
6     new_coords=np.rint(new_coords).astype(int)
7     new_img_coords[i,:]=new_coords[0:2]
8
9 #using vectorized operation
10 new_coords=np.matmul(H_inv, img_coords.T)
11 new_coords=new_coords/new_coords[2]
12 new_coords=np.rint(new_coords).astype(int)
13 new_img_coords=new_coords.T[:,2]
14 new_img_coords
```

6 Code Listings

```

1 # -*- coding: utf-8 -*-
2 """hw3_img1.ipynb
3
4 Automatically generated by Colaboratory.
5
6 Original file is located at
7 https://colab.research.google.com/drive/1iKa8BG50PUYylC-9OuSHe3axj2L5VV8S
8
9 Often used functions:
10 """
11
12 #to check to validitiy of the 4 points, we draw a bounding box aroud the image
13 def draw_bbox(image, points):
```

```

14 copy_image=image.copy()
15 #draw bounding box to check correctness of the points
16 points_reshape = points.reshape((-1, 1, 2))
17 color = (255, 0, 0)
18 im1_bbox = cv2.polylines(copy_image, [points_reshape], isClosed=True, color=
    color, thickness = 10)
19 cv2.imwrite("bbox.jpg",im1_bbox)
20 plt.axis('off')
21 plt.imshow(im1_bbox)
22 plt.imsave("bounding_box1.jpeg",im1_bbox)
23
24 def Homography_matrix(X,X_prime):
25     """
26     This function takes input X and X_prime and computes the homography between
27     them. Returns a 3x3 non-singular matrix
28     H.
29     """
30     H = np.ones((3,3)) #Initialise the Homography matrix
31     A = np.zeros((8,8)) #initialise a matrix
32
33     C=X_prime
34
35     #build the A matrix column by column
36     for i in range(0,4):
37         A[2*i,0]=X[2*i]
38         A[2*i,1]=X[2*i+1]
39         A[2*i,2]=1
40         A[2*i+1,3]=X[2*i]
41         A[2*i+1,4]=X[2*i+1]
42         A[2*i+1,5]=1
43         A[2*i,6]=X[2*i]*X_prime[2*i]
44         A[2*i+1,6]=X[2*i]*X_prime[2*i+1]
45         A[2*i,7]=(X[2*i+1]*X_prime[2*i])
46         A[2*i+1,7]=(X[2*i+1]*X_prime[2*i+1])
47     A[:,6:]=-A[:,6:]
48
49     #B=(A^-1)C
50     B=np.dot(np.linalg.inv(A),C)
51
52     #reshape the B vector to get the homography matrix
53     H[0,0]=B[0]
54     H[0,1]=B[1]
55     H[0,2]=B[2]
56     H[1,0]=B[3]
57     H[1,1]=B[4]
58     H[1,2]=B[5]
59     H[2,0]=B[6]
60     H[2,1]=B[7]
61     H[2,2]=1
62     return H
63
64 """## **Image 1**
65
66 **Point to Point Method**
67 """
68
69 import numpy as np

```

```

70 import matplotlib.pyplot as plt
71 import cv2
72 from PIL import Image
73 import math
74
75 #in order to get a better image made use of multiple windows and assumed the
76 #image coordinates in world plane accordingly.
76 def get_4_points(n):
77     if n==1: #image 1a
78         #for the image
79         P=[117,586] #x1,y1
80         Q=[876,680] #x2,y2
81         R=[865,297] #x3,y3
82         S=[175,76] #x4,y4
83         X_prime=P+Q+R+S
84         return np.array([P,Q,R,S]),X_prime
85
86 #assume distance between the edges of two windows to be equal to the window
86 #size
87 def get_cat_points():
88     #for the kittens
89     P_dash=[0,85*15]
90     Q_dash=[75*23,85*15]
91     R_dash=[75*23,0]
92     S_dash=[0,0]
93     X=P_dash+Q_dash+R_dash+S_dash
94     return np.array([P_dash,Q_dash,R_dash,S_dash]),X
95
96 #load and display the image
97 image=cv2.imread("/content/drive/My Drive/hw3_Task1_Images/Images/Img1.JPG")
98 image_orig = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
99 im1 = np.copy(image_orig)
100 plt.imshow(im1)
101 print(im1.shape)
102
103 #get the points in the image and world frames
104 P1,X=get_4_points(1)
105 P2,X_prime=get_cat_points()
106
107 draw_bbox(im1,P1)
108 #compute the Homography between the points
109 H=Homography_matrix(X,X_prime)
110 print(H)
111 H_inv=np.linalg.inv(H)
112
113 #get the image coordinates in world plane. We map the 4 corner points of the
113 #image to the world coordinates
114 #to get the size of the image in world coordinates.
115 width=im1.shape[1]
116 height=im1.shape[0]
117 print("width(x)=",width)
118 print("height(y)=",height)
119
120 img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
121 print(img_coords.T)
122
123 new_coords=np.matmul(H,img_coords.T)
124 new_coords=new_coords/new_coords[2]

```

```

125 new_coords=np.rint(new_coords).astype(int)
126 new_img_coords=new_coords.T[:,2]
127 new_img_coords
128 print(new_img_coords)
129 ,
130 ,
131 We can see from the projected corner points that the origin in this new
132 coordinate is not at (0,0). Hence we need
133 to find the appropriate translation to bring the coordinates along origin.
134 ,
135 #The output image coordinates are very huge and take long time to compute.
136 Hence we will scale the coordinates to
137 #get a reasonable image size with which we can work with.
138 scale1=width/((max(new_img_coords[:,0])-min(new_img_coords[:,0])))
139 scale2=height/((max(new_img_coords[:,1])-min(new_img_coords[:,1])))
140 scale=max(scale1, scale2)
141 print(scale)
142 #find the translation to bring the coordinates to positive values
143 tx=np.round(np.min(new_img_coords[:,0]))
144 ty=np.round(np.min(new_img_coords[:,1]))
145 print(tx,ty)
146 ,
147 ,
148 Find out the shape of the new image in world coordinates.
149 ,
150
151 new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale2
152 ).astype(int)
153 new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale1)
154 new_image=np.zeros((new_height, new_width, 3))
155 max_X=im1.shape[1]
156 max_Y=im1.shape[0]
157
158
159
160 ,
161 To find the mapping of every point in the new image to the distorted image, we
162 multiply every point with the inverse
163 of the Homography - i.e. map every point in real world coordinates to the
164 image plane and find the correspondence.
165 ,
166 for i in range(0,new_width): #x
167     for j in range(0,new_height): #y
168
169         k1=i/scale1+tx
170         k2=j/scale2+ty
171         init=np.array((k1,k2,1))
172         mapped=np.matmul(H_inv, init)
173         mapped=np.rint(mapped/mapped[2])
174         mapped=mapped.astype(int)
175         if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
176             new_image[j,i]=im1[mapped[1], mapped[0]]
177
178 plt.figure(figsize=(20,20))

```

```

177 plt.imshow(new_image.astype(int))
178 ii=new_image.astype(np.uint8)
179 plt.imsave("points1.jpeg",ii)
180
181
182
183
184
185 """**Vanishing Line Method**
186
187 *Projection Removal*
188 """
189
190 #pick the 4 points for two pairs of parallel lines
191 P1=[583,646] #x1,y1
192 P2=[779,671] #x2,y2
193 P3=[771,266] #x3,y3
194 P4=[594,210] #x4,y4
195
196 ,,
197 Q1=[410,120] #l3
198 Q2=[533,8] #l3
199 Q3=[409,374] #l4
200 Q4=[539,312] #l4
201 ,,
202 Q1=P1
203 Q2=P4
204 Q3=P2
205 Q4=P3
206 im_copy=im1.copy()
207 #display the lines
208 ##https://www.geeksforgeeks.org/python-opencv-cv2-line-method/ for drawing
209 lines on image
210 line_img=cv2.line(im_copy, tuple(P1), tuple(P2), (255,0,0), 5)
211 line_img=cv2.line(line_img, tuple(P3), tuple(P4), (255,0,0), 5)
212 line_img=cv2.line(line_img, tuple(Q1), tuple(Q2), (0,255,0), 5)
213 line_img=cv2.line(line_img, tuple(Q3), tuple(Q4), (0,255,0), 5)
214 plt.imshow(line_img)
215 plt.imsave("parallel_lines1.jpeg",line_img)
216
217 #get the first vanishing point
218 P_1=np.array(P1)
219 P_1=np.append(P_1,1)
220 P_2=np.array(P2)
221 P_2=np.append(P_2,1)
222 l1=np.cross(P_1,P_2)
223
224 P_3=np.array(P3)
225 P_3=np.append(P_3,1)
226 P_4=np.array(P4)
227 P_4=np.append(P_4,1)
228 l2=np.cross(P_3,P_4)
229
230 PV1=np.cross(l1,l2)
231 PV1=PV1/PV1[2]
232 print(PV1)
233
234 #get the second vanishing point

```

```

234 Q_1=np.array(Q1)
235 Q_1=np.append(Q_1,1)
236 Q_2=np.array(Q2)
237 Q_2=np.append(Q_2,1)
238 l3=np.cross(Q_1,Q_2)
239
240 Q_3=np.array(Q3)
241 Q_3=np.append(Q_3,1)
242 Q_4=np.array(Q4)
243 Q_4=np.append(Q_4,1)
244 l4=np.cross(Q_3,Q_4)
245
246 Q_1=np.array(P1)
247 Q_1=np.append(Q_1,1)
248 Q_2=np.array(P3)
249 Q_2=np.append(Q_2,1)
250 l3=np.cross(Q_1,Q_2)
251
252 Q_3=np.array(P2)
253 Q_3=np.append(Q_3,1)
254 Q_4=np.array(P4)
255 Q_4=np.append(Q_4,1)
256 l4=np.cross(Q_3,Q_4)
257
258 PV2=np.cross(l3,l4)
259 PV2=PV2/PV2[2]
260 print(PV2)
261
262 #find the vanishing line and H matrix
263 lV=np.cross(PV1,PV2)
264 lV=lV/lV[2]
265 lV
266
267 H2=np.array([[1,0,0],[0,1,0],lV])
268 H2
269 print(H2)
270 H2_inv=np.linalg.inv(H2)
271
272 #get the image dimension in world coordinates and apply appropriate scaling
273 img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
274 print(img_coords.T)
275
276 new_img_coords=np.zeros([4,2])
277 for i in range(4):
278
279     new_coords=np.matmul(H2,img_coords[i].T)
280     new_coords=new_coords/new_coords[2]
281     new_coords=np.rint(new_coords).astype(int)
282     new_img_coords[i,:]=new_coords[0:2]
283
284 scale1=width/((max(new_img_coords[:,0])-min(new_img_coords[:,0])))
285 scale2=height/((max(new_img_coords[:,1])-min(new_img_coords[:,1])))
286 scale=max(scale1, scale2)
287 print(scale)
288
289 tx=np.round(np.min(new_img_coords[:,0]))
290 ty=np.round(np.min(new_img_coords[:,1]))
291 print(tx,ty)

```

```

292
293 new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale)
294 .astype(int)
294 new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale).
295 .astype(int)
295 new_image=np.zeros((new_height,new_width,3))
296 print(new_image.shape)
297
298 max_X=im1.shape[1]
299 max_Y=im1.shape[0]
300
301 #map the image from world to distortion plane to get output
302 for i in range(0,new_width-1): #x
303     for j in range(0,new_height-1): #y
304
305     k1=i/scale+tx
306     k2=j/scale+ty
307     init=np.array((k1,k2,1))
308     mapped=np.matmul(H2_inv,init)
309     mapped=np.rint(mapped/mapped[2])
310     mapped=mapped.astype(int)
311     if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
312         new_image[j,i]=im1[mapped[1],mapped[0]]
313
314 plt.imshow(new_image.astype(int))
315 ii=new_image.astype(np.uint8)
316 plt.imsave("proj1.jpeg",ii)
317
318 """*Affine Removal*"""
319
320 #get the 4 points for two pairs of orthogonal lines
321 im_copy=im1.copy()
322 P1=[158,238]
323 P2=[415,304]
324 P3=[425,156]
325 P4=[177,77]
326 line_img=cv2.line(im_copy, tuple(P1), tuple(P3), (255,0,0), 5)
327 line_img=cv2.line(line_img, tuple(P2), tuple(P4), (255,0,0), 5)
328 line_img=cv2.line(line_img, tuple(P1), tuple(P2), (0,255,0), 5)
329 line_img=cv2.line(line_img, tuple(P3), tuple(P2), (0,255,0), 5)
330 plt.imshow(line_img)
331 plt.imsave("perpend1.jpeg",line_img)
332
333 #map the 4 points onto the projective corrected image plane
334 P_homo=np.array(P1)
335 P_homo=np.append(P_homo,1)
336 Q_homo=np.array(P2)
337 Q_homo=np.append(Q_homo,1)
338 R_homo=np.array(P3)
339 R_homo=np.append(R_homo,1)
340 S_homo=np.array(P4)
341 S_homo=np.append(S_homo,1)
342
343 P_dash_homo=np.matmul(H2,P_homo)
344 P_dash=P_dash_homo/P_dash_homo[-1]
345 Q_dash_homo=np.matmul(H2,Q_homo)
346 Q_dash=Q_dash_homo/Q_dash_homo[-1]
347 R_dash_homo=np.matmul(H2,R_homo)

```

```

348 R_dash=R_dash_homo/R_dash_homo[ -1 ]
349 S_dash_homo=np . matmul (H2,S_homo)
350 S_dash=S_dash_homo/S_dash_homo[ -1 ]
351
352 #initialize the matrices
353 A = np . zeros ((2 ,2))
354 S_mat = np . ones ((2 ,2))
355 H_c = np . identity (3)
356
357 #get the first set of perpendicular lines
358 l1 = np . cross (P_dash ,R_dash) #l'
359 m1 = np . cross (S_dash ,Q_dash) #m'
360
361 #get the second set of perpendicular lines
362 l2 = np . cross (P_dash ,Q_dash)
363 m2 = np . cross (Q_dash ,R_dash)
364
365 #get the A and matrix
366 A[0 ,0] = l1[0]*m1[0]
367 A[0 ,1] = l1[0]*m1[1] + l1[1]*m1[0]
368 A[1 ,:] = [l2[0]*m2[0] , l2[0]*m2[1] + l2[1]*m2[0]]
369
370 B = np . array ([-l1[1]*m1[1] , -l2[1]*m2[1]])
371
372 #Find out S
373 S_elem = np . matmul (np . linalg . pinv (A) ,B)
374 S_mat[0 ,:] = [S_elem[0] , S_elem[1]]
375 S_mat[1 ,0] = S_elem[1]
376 U, D, Vt = np . linalg . svd (S_mat, full_matrices=True)
377 D_sq = np . zeros ((2 ,2))
378 D_sq[0 ,0] = np . sqrt (D[0])
379 D_sq[1 ,1] = np . sqrt (D[1])
380
381 #Get the value of A (AA'=S)
382 A_forS = np . matmul (np . matmul (Vt ,D_sq) ,np . transpose (Vt))
383
384 H_c[0:2 ,0:2] = A_forS
385 H_c_inv=np . linalg . inv (H_c)
386
387 H_total=np . matmul (H_c_inv ,H2)
388 H_total_inv=np . linalg . inv (H_total)
389
390 #mask the original image to projective + affine corrected image. Here we
391 # concat the two homographies that we found
392 # and apply directly on the original image. This method gave a better result
393 # as compared to applying homography on
394 # projection corrected image.
395 img_coords=np . array ([[0 ,0 ,1] ,[0 ,height ,1] ,[width ,0 ,1] ,[width ,height ,1]])
396 print (img_coords .T)
397
398 new_img_coords=np . zeros ([4 ,2])
399 for i in range (4):
400     new_coords=np . matmul (H_total ,img_coords [i] .T)
401     new_coords=new_coords/new_coords[2]
402     new_coords=np . rint (new_coords) . astype (int)
403

```

```

404     new_img_coords[ i ,:]= new_coords [ 0:2 ]
405
406
407 scale1=width/(max( new_img_coords [ : ,0] ) -min( new_img_coords [ : ,0] ) )
408 scale2=height/(max( new_img_coords [ : ,1] ) -min( new_img_coords [ : ,1] ) )
409 scale=max( scale1 , scale2 )
410 print( scale )
411
412 tx=np. round( np. min( new_img_coords [ : ,0] ) )
413 ty=np. round( np. min( new_img_coords [ : ,1] ) )
414 print( tx , ty )
415
416 new_height=np. round( ( max( new_img_coords [ : ,1] ) -min( new_img_coords [ : ,1] ) ) *scale2
417 ) . astype( int )
417 new_width=np. round( ( max( new_img_coords [ : ,0] ) -min( new_img_coords [ : ,0] ) ) *scale1 )
418 . astype( int )
418 new_image=np. zeros( ( new_height ,new_width ,3 ) )
419 print( new_image. shape )
420 max_X=im1. shape [ 1 ]
421 max_Y=im1. shape [ 0 ]
422
423 blank_image=np. zeros( ( new_height ,new_width ,3 ) )
424 for i in range(0 ,new_image. shape [ 1 ] -1) : #x
425     for j in range(0 ,new_image. shape [ 0 ] -1) : #y
426
427     k1=i /scale1+tx
428     k2=j /scale2+ty
429     init=np. array( ( k1 ,k2 ,1 ) )
430     mapped=np. matmul( H_total_inv , init )
431     mapped=np. rint( mapped/mapped [ 2 ] )
432     mapped=mapped. astype( int )
433     #if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
434     if (mapped [ 0 ] > 0 and mapped [ 1 ] > 0 and mapped [ 0 ] < im1. shape [ 1 ] and mapped
435 [ 1 ] < image. shape [ 0 ] ) :
436         blank_image [ j , i ]=im1 [ mapped [ 1 ] ,mapped [ 0 ] ]
437
438 plt. figure( figsize=(20 ,20) )
439 plt. imshow( blank_image. astype( int ) )
440 ii=blank_image. astype( np. uint8 )
441 plt. imsave( "affine1. jpeg" ,ii )
442 """
443     ***Single Step method***"""
444 #get the 4 points to find 5 orthogonal lines. Assumption is made that the bbox
445 #in the original plane corresponds to a square.
446 P=[117 ,586] #x1 ,y1
447 Q=[394 ,621] #x2 ,y2
448 R=[408 ,388] #x3 ,y3
449 S=[146 ,334] #x4 ,y4
450
451 #get the homogeneous coordinates
452 P_homo=np. array( P )
453 P_homo=np. append( P_homo ,1 )
454 Q_homo=np. array( Q )
455 Q_homo=np. append( Q_homo ,1 )
456 R_homo=np. array( R )
457 R_homo=np. append( R_homo ,1 )
458 S_homo=np. array( S )

```

```

458 S_homo=np.append(S_homo,1)
459
460 #find the 5 perpendicular line pairs
461 l1=np.cross(P_homo,Q_homo) #p11
462 l1=l1/l1[2]
463 m1=np.cross(Q_homo,R_homo) #p11
464 m1=m1/m1[2]
465
466 l2=np.cross(Q_homo,R_homo) #p12
467 l2=l2/l2[2]
468 m2=np.cross(R_homo,S_homo) #p12
469 m2=m2/m2[2]
470
471 l3=np.cross(R_homo,S_homo) #p13
472 l3=l3/l3[2]
473 m3=np.cross(S_homo,P_homo) #p13
474 m3=m3/m3[2]
475
476 l4=np.cross(S_homo,P_homo) #p14
477 l4=l4/l4[2]
478 m4=np.cross(P_homo,Q_homo) #p14
479 m4=m4/m4[2]
480
481 l5=np.cross(P_homo,R_homo) #p15
482 l5=l5/l5[2]
483 m5=np.cross(Q_homo,S_homo) #p15
484 m5=m5/m5[2]
485
486 #get a 5x6 matrix corresponding to the equation
487 A=np.zeros((5,6))
488 A[0]=[11[0]*m1[0],(11[1]*m1[0]+11[0]*m1[1])/2,11[1]*m1[1],(11[0]*m1[2]+11[2]*m1[0])/2,(11[1]*m1[2]+11[2]*m1[1])/2,11[2]*m1[2]]
489 A[1]=[12[0]*m2[0],(12[1]*m2[0]+12[0]*m2[1])/2,12[1]*m2[1],(12[0]*m2[2]+12[2]*m2[0])/2,(12[1]*m2[2]+12[2]*m2[1])/2,12[2]*m2[2]]
490 A[2]=[13[0]*m3[0],(13[1]*m3[0]+13[0]*m3[1])/2,13[1]*m3[1],(13[0]*m3[2]+13[2]*m3[0])/2,(13[1]*m3[2]+13[2]*m3[1])/2,13[2]*m3[2]]
491 A[3]=[14[0]*m4[0],(14[1]*m4[0]+14[0]*m4[1])/2,14[1]*m4[1],(14[0]*m4[2]+14[2]*m4[0])/2,(14[1]*m4[2]+14[2]*m4[1])/2,14[2]*m4[2]]
492 A[4]=[15[0]*m5[0],(15[1]*m5[0]+15[0]*m5[1])/2,15[1]*m5[1],(15[0]*m5[2]+15[2]*m5[0])/2,(15[1]*m5[2]+15[2]*m5[1])/2,15[2]*m5[2]]
493 A
494
495 #need to find the null vector for the A matrix. Will make use of sympy library
496 from sympy import Matrix
497 a=Matrix(A)
498 c=a.nullspace()
499
500 c=np.array(c).astype(np.int)
501 c=c/max(c[0])
502 c
503
504 #get the H matrix from C
505 S_mat = np.zeros((2,2))
506 S_mat[0,0] = c[0][0]
507 S_mat[1,0] = 0.5 * c[0,1]
508 S_mat[0,1] = 0.5 * c[0,1]
509 S_mat[1,1] = c[0,2]
510 S_mat

```

```

511
512 #take the svd of S to define A
513 U, D, Vt = np.linalg.svd(S_mat)
514
515 A = np.zeros((2,2))
516 D_sq = np.diag(np.sqrt(D))
517 A = np.dot(Vt, np.dot(D_sq, Vt))
518
519 v = np.zeros((2,1))
520 rhs_vA = np.zeros((2,1))
521 rhs_vA[0] = 0.5 * c[0,3]
522 rhs_vA[1] = 0.5 * c[0,4]
523
524 v = np.dot(np.linalg.inv(A), rhs_vA)
525
526 H = np.zeros((3,3))
527
528 #Filling up the homography
529 H[0]=np.array([A[0,0], A[0,1], 0])
530 H[1]=np.array([A[1,0], A[1,1], 0])
531 H[2]=np.array([v[0], v[1], 1])
532
533 H
534
535 H_inv=np.linalg.inv(H)
536 H_inv
537
538 img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
539 print(img_coords.T)
540
541 new_img_coords=np.zeros([4,2])
542
543 for i in range(4):
544
545     new_coords=np.matmul(H_inv, img_coords[i].T)
546     new_coords=new_coords/new_coords[2]
547     new_coords=np.rint(new_coords).astype(int)
548     new_img_coords[i,:]=new_coords[0:2]
549
550
551 scale1=width/((max(new_img_coords[:,0])-min(new_img_coords[:,0])))
552 scale2=height/((max(new_img_coords[:,1])-min(new_img_coords[:,1])))
553 scale=max(scale1, scale2)
554 print(scale)
555
556 tx=np.round(np.min(new_img_coords[:,0]))
557 ty=np.round(np.min(new_img_coords[:,1]))
558 print(tx, ty)
559
560 new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale)
561 .astype(int)
561 new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale).
562 new_image=np.zeros((new_height, new_width,3))
563 print(new_image.shape)
564 max_X=im1.shape[1]
565 max_Y=im1.shape[0]
566

```

```

567 blank_image=np.zeros((new_height,new_width,3))
568 for i in range(0,new_image.shape[1]-1): #x
569     for j in range(0,new_image.shape[0]-1): #y
570
571     k1=i/scale+tx
572     k2=j/scale+ty
573     init=np.array((k1,k2,1))
574     mapped=np.matmul(H,init)
575     mapped=np.rint(mapped/mapped[2])
576     mapped=mapped.astype(int)
577     #if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
578     if (mapped[0] > 0 and mapped[1] > 0 and mapped[0] < im1.shape[1] and mapped[1] < image.shape[0]):
579         blank_image[j,i]=im1[mapped[1],mapped[0]]
580
581 plt.figure(figsize=(20,20))
582 plt.imshow(blank_image.astype(np.int))
583 ii=blank_image.astype(np.uint8)
584 plt.imsave("onestep1.jpeg",ii)
585
586
587
588
589 """
590 ## **Image 2**
591
592 **Point to Point Method**
593 """
594
595 #These points are considering only the given dimensions in real world. These
596 #did not yield good results.
597
598 def get_4_points(n):
599     if n==1: #image 1a
600         #for the image
601         P=[480,874] #x1,y1
602         Q=[606,924] #x2,y2
603         R=[477,719] #x3,y3
604         S=[601,737] #x4,y4
605         X_prime=P+Q+R+S
606         return np.array([P,Q,R,S]),X_prime
607
608 def get_cat_points():
609     #for the kittens
610     P_dash=[0,74]
611     Q_dash=[84,74]
612     R_dash=[0,0]
613     S_dash=[84,0]
614     X=P_dash+Q_dash+R_dash+S_dash
615     return np.array([P_dash,Q_dash,R_dash,S_dash]),X
616
617 def get_4_points(n):
618     if n==1: #image 1a
619         #for the image
620         P=[480,874] #x1,y1
621         Q=[606,924] #x2,y2
622         R=[601,737] #x3,y3

```

```

623     S= [477,719] #x4,y4
624     X_prime=P+Q+R+S
625     return np.array ([P,Q,R,S]) ,X_prime
626
627 def get_cat_points():
628     #for the kittens
629     P_dash=[0,74]
630     Q_dash=[84,74]
631     R_dash=[84,0]
632     S_dash=[0,0]
633     X=P_dash+Q_dash+R_dash+S_dash
634     return np.array ([P_dash,Q_dash,R_dash,S_dash]) ,X
635
636
637
638 ,,
639
640 #these are points related to the entire frame of the window. This gives a
641 #relatively better result.
642 def get_4_points(n):
643     if n==1: #image 1a
644         #for the image
645         P=[379,838] #x1,y1
646         Q=[606,924] #x2,y2
647         R=[598,545] #x3,y3
648         S=[378,570] #x4,y4
649         X_prime=P+Q+R+S
650         return np.array ([P,Q,R,S]) ,X_prime
651
652 def get_cat_points():
653     #for the kittens
654     P_dash=[0,148]
655     Q_dash=[168,148]
656     R_dash=[168,0]
657     S_dash=[0,0]
658     X=P_dash+Q_dash+R_dash+S_dash
659     return np.array ([P_dash,Q_dash,R_dash,S_dash]) ,X
660
661 #load and display the image
662 image=cv2.imread("/content/drive/My Drive/hw3_Task1_Images/Images/Img2.jpeg")
663 image_orig = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
664 im1 = np.copy(image_orig)
665 plt.imshow(im1)
666 print(im1.shape)
667
668 #get the points in the image and world frames
669 P1,X=get_4_points(1)
670 P2,X_prime=get_cat_points()
671 draw_bbox(im1,P1)
672 #compute the Homography between the points
673 H=Homography_matrix(X,X_prime)
674 print(H)
675 H_inv=np.linalg.inv(H)
676
677 #get the image coordinates in world plane. We map the 4 corner points of the
678 #image to the world coordinates
679 #to get the size of the image in world coordinates.
680 width=im1.shape [1]

```

```

679 height=im1.shape[0]
680 print("width(x)=",width)
681 print("height(y)=",height)
682
683 img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
684 print(img_coords.T)
685
686 new_coords=np.matmul(H,img_coords.T)
687 new_coords=new_coords/new_coords[2]
688 new_coords=np.rint(new_coords).astype(int)
689 new_img_coords=new_coords.T[:,2]
690 new_img_coords
691
692 """
693 We can see from the projected corner points that the origin in this new
       coordinate is not at (0,0). Hence we need
694 to find the appropriate translation to bring the coordinates along origin.
695 """
696 scale1=width/((max(new_img_coords[:,0])-min(new_img_coords[:,0])))
697 scale2=height/((max(new_img_coords[:,1])-min(new_img_coords[:,1])))
698 scale=max(scale1, scale2)
699 print(scale)
700
701 tx=np.round(np.min(new_img_coords[:,0]))
702 ty=np.round(np.min(new_img_coords[:,1]))
703 print(tx,ty)
704
705 """
706 Find out the shape of the new image in world coordinates.
707 """
708
709 new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale2
       ).astype(int)
710 new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale1
       ).astype(int)
711 new_image=np.zeros((new_height,new_width,3))
712
713 max_X=im1.shape[1]
714 max_Y=im1.shape[0]
715
716 """
717 To find the mapping of every point in the new image to the distorted image, we
       multiply every point with the inverse
718 of the Homography - i.e. map every point in real world coordinates to the
       image plane and find the correspondence.
719 """
720 for i in range(0,new_width-1): #x
721     for j in range(0,new_height-1): #y
722
723         k1=i/scale1+tx
724         k2=j/scale2+ty
725         init=np.array((k1,k2,1))
726         mapped=np.matmul(H_inv,init)
727         mapped=np.rint(mapped/mapped[2])
728         mapped=mapped.astype(int)
729         if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
730             new_image[j,i]=im1[mapped[1],mapped[0]]
731

```

```

732 plt.figure(figsize=(20,20))
733 plt.imshow(new_image.astype(int))
734 ii=new_image.astype(np.uint8)
735 plt.imsave("points2.jpeg", ii)
736
737 """**Two Step Method**
738
739 *Projection Removal*
740 """
741
742 P1=[618,510]
743 P2=[369,548]
744 P3=[361,853]
745 P4=[644,973]
746
747 Q1=P1 #13
748 Q2=P3 #13
749 Q3=P4 #14
750 Q4=P2 #14
751 im_copy=im1.copy()
752 ##https://www.geeksforgeeks.org/python-opencv-cv2-line-method/ for drawing
    lines on image
753 line_img=cv2.line(im_copy, tuple(P1), tuple(P2), (255,0,0), 5)
754 line_img=cv2.line(line_img, tuple(P3), tuple(P4), (255,0,0), 5)
755 line_img=cv2.line(line_img, tuple(P1), tuple(P4), (0,255,0), 5)
756 line_img=cv2.line(line_img, tuple(P3), tuple(P2), (0,255,0), 5)
757 plt.imshow(line_img)
758 plt.imsave("parallel2.jpeg",line_img)
759
760 P_1=np.array(P1)
761 P_1=np.append(P_1,1)
762 P_2=np.array(P2)
763 P_2=np.append(P_2,1)
764 l1=np.cross(P_1,P_2)
765
766 P_3=np.array(P3)
767 P_3=np.append(P_3,1)
768 P_4=np.array(P4)
769 P_4=np.append(P_4,1)
770 l2=np.cross(P_3,P_4)
771
772 PV1=np.cross(l1,l2)
773 PV1=PV1/PV1[2]
774 print(PV1)
775
776 Q_1=np.array(P1)
777 Q_1=np.append(Q_1,1)
778 Q_2=np.array(P4)
779 Q_2=np.append(Q_2,1)
780 l3=np.cross(Q_1,Q_2)
781
782 Q_3=np.array(P3)
783 Q_3=np.append(Q_3,1)
784 Q_4=np.array(P2)
785 Q_4=np.append(Q_4,1)
786 l4=np.cross(Q_3,Q_4)
787 ,,
788 Q_1=np.array(P1)

```

```

789 Q_1=np.append(Q_1,1)
790 Q_2=np.array(P3)
791 Q_2=np.append(Q_2,1)
792 l3=np.cross(Q_1,Q_2)
793
794 Q_3=np.array(P2)
795 Q_3=np.append(Q_3,1)
796 Q_4=np.array(P4)
797 Q_4=np.append(Q_4,1)
798 l4=np.cross(Q_3,Q_4)
799 ,,
800 PV2=np.cross(l3,l4)
801 PV2=PV2/PV2[2]
802 print(PV2)
803
804 lV=np.cross(PV1,PV2)
805 lV=lV[2]
806
807 H2=np.array([[1,0,0],[0,1,0],lV])
808 H2
809 print(H2)
810 H2_inv=np.linalg.inv(H2)
811
812 img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
813 print(img_coords.T)
814
815 new_img_coords=np.zeros([4,2])
816 for i in range(4):
817
818     new_coords=np.matmul(H2,img_coords[i].T)
819     new_coords=new_coords/new_coords[2]
820     new_coords=np.rint(new_coords).astype(int)
821     new_img_coords[i,:]=new_coords[0:2]
822
823 scale1=width/(max(new_img_coords[:,0])-min(new_img_coords[:,0]))
824 scale2=height/(max(new_img_coords[:,1])-min(new_img_coords[:,1]))
825 scale=max(scale1, scale2)
826 print(scale)
827
828 tx=np.round(np.min(new_img_coords[:,0]))
829 ty=np.round(np.min(new_img_coords[:,1]))
830 print(tx,ty)
831
832 new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale)
833 .astype(int)
834 new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale).
835 .astype(int)
836 new_image=np.zeros((new_height,new_width,3))
837 print(new_image.shape)
838
839 max_X=im1.shape[1]
840 max_Y=im1.shape[0]
841
842 for i in range(0,new_width-1): #x
843     for j in range(0,new_height-1): #y
844
845         k1=i/scale+tx
846         k2=j/scale+ty

```

```

845     init=np.array((k1,k2,1))
846     mapped=np.matmul(H2_inv, init)
847     mapped=np.rint(mapped/mapped[2])
848     mapped=mapped.astype(int)
849     if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
850         new_image[j,i]=im1[mapped[1], mapped[0]]
851
852 plt.imshow(new_image.astype(int))
853 ii=new_image.astype(np.uint8)
854 plt.imsave("proj2.jpeg", ii)
855
856 """*Affine Removal*"""
857
858 #4 points for determining orthogonal lines (assuming the window is square)
859
860 P1=[481,873]
861 P2=[606,922]
862 P3=[598,735]
863 P4=[479,718]
864 im_copy=im1.copy()
865 line_img=cv2.line(im_copy, tuple(P1), tuple(P3), (255,0,0), 5)
866 line_img=cv2.line(line_img, tuple(P2), tuple(P4), (255,0,0), 5)
867 line_img=cv2.line(line_img, tuple(P1), tuple(P2), (0,255,0), 5)
868 line_img=cv2.line(line_img, tuple(P3), tuple(P2), (0,255,0), 5)
869 plt.imshow(line_img)
870 plt.imsave("perpend2.jpeg",line_img)
871
872 #map the 4 points onto the projective corrected image plane
873 P_homo=np.array(P1)
874 P_homo=np.append(P_homo,1)
875 Q_homo=np.array(P2)
876 Q_homo=np.append(Q_homo,1)
877 R_homo=np.array(P3)
878 R_homo=np.append(R_homo,1)
879 S_homo=np.array(P4)
880 S_homo=np.append(S_homo,1)
881
882 P_dash_homo=np.matmul(H2,P_homo)
883 P_dash=P_dash_homo/P_dash_homo[-1]
884 Q_dash_homo=np.matmul(H2,Q_homo)
885 Q_dash=Q_dash_homo/Q_dash_homo[-1]
886 R_dash_homo=np.matmul(H2,R_homo)
887 R_dash=R_dash_homo/R_dash_homo[-1]
888 S_dash_homo=np.matmul(H2,S_homo)
889 S_dash=S_dash_homo/S_dash_homo[-1]
890
891 ##checking
892 A = np.zeros((2,2))
893 S_mat = np.ones((2,2))
894 H_c = np.identity(3)
895
896 l1 = np.cross(P_dash,Q_dash) #l
897 m1 = np.cross(Q_dash,R_dash) #m
898
899 l2 = np.cross(R_dash,P_dash)
900 m2 = np.cross(S_dash,Q_dash)
901
902 A[0,0] = l1[0]*m1[0]

```

```

903 A[0,1] = 11[0]*m1[1] + 11[1]*m1[0]
904 A[1,:] = [12[0]*m2[0], 12[0]*m2[1] + 12[1]*m2[0]]
905
906 B = np.array([-11[1]*m1[1], -12[1]*m2[1]])
907 S_elem = np.matmul(np.linalg.pinv(A), B)
908 S_mat[0,:] = [S_elem[0], S_elem[1]]
909 S_mat[1,0] = S_elem[1]
910 U, D, Vt = np.linalg.svd(S_mat, full_matrices=True)
911 D_sq = np.zeros((2,2))
912 D_sq[0,0] = np.sqrt(D[0])
913 D_sq[1,1] = np.sqrt(D[1])
914
915 A_forS = np.matmul(np.matmul(Vt, D_sq), np.transpose(Vt))
916
917 H_c[0:2,0:2] = A_forS
918 H_c_inv = np.linalg.inv(H_c)
919
920 H_total=np.matmul(H_c_inv,H2)
921 H_total_inv=np.linalg.inv(H_total)
922
923 #mask the original image to projective + affine corrected image
924 img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
925 print(img_coords.T)
926
927
928 new_img_coords=np.zeros([4,2])
929
930 for i in range(4):
931
932     new_coords=np.matmul(H_total, img_coords[i].T)
933     new_coords=new_coords/new_coords[2]
934     new_coords=np.rint(new_coords).astype(int)
935     new_img_coords[i,:]=new_coords[0:2]
936
937
938 scale1=width/((max(new_img_coords[:,0])-min(new_img_coords[:,0])))
939 scale2=height/((max(new_img_coords[:,1])-min(new_img_coords[:,1])))
940 scale=max(scale1, scale2)
941 print(scale)
942
943 tx=np.round(np.min(new_img_coords[:,0]))
944 ty=np.round(np.min(new_img_coords[:,1]))
945 print(tx,ty)
946
947 new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale2)
948     .astype(int)
949 new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale1)
950     .astype(int)
951 new_image=np.zeros((new_height, new_width, 3))
952 print(new_image.shape)
953 max_X=im1.shape[1]
954 max_Y=im1.shape[0]
955
956 blank_image=np.zeros((new_height, new_width, 3))
957 for i in range(0,new_image.shape[1]-1): #x
958     for j in range(0,new_image.shape[0]-1): #y
959
960         k1=i/scale1+tx

```

```

959     k2=j / scale2+ty
960     init=np. array (( k1 , k2 , 1 ))
961     mapped=np. matmul ( H_total_inv , init )
962     mapped=np. rint ( mapped / mapped [ 2 ])
963     mapped=mapped. astype ( int )
964     #if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
965     if ( mapped [ 0 ] > 0 and mapped [ 1 ] > 0 and mapped [ 0 ] < im1. shape [ 1 ] and mapped
966     [ 1 ] < image. shape [ 0 ] ) :
967         blank_image [ j , i ]=im1 [ mapped [ 1 ] , mapped [ 0 ] ]
968
969 plt. figure ( figsize=(20,20) )
970 plt. imshow ( blank_image . astype ( int ) )
971 ii=blank_image . astype ( np. uint8 )
972 plt. imsave ( "affine2.jpeg" , ii )
973
974
975
976
977 """**One Step Method**"""
978
979 #4 points for determining orthogonal lines
980
981 P=[369,548]
982 Q=[361,853]
983 R=[644,973]
984 S=[618,510]
985
986 P_homo=np. array ( P )
987 P_homo=np. append ( P_homo , 1 )
988 Q_homo=np. array ( Q )
989 Q_homo=np. append ( Q_homo , 1 )
990 R_homo=np. array ( R )
991 R_homo=np. append ( R_homo , 1 )
992 S_homo=np. array ( S )
993 S_homo=np. append ( S_homo , 1 )
994
995 l1=np. cross ( P_homo , Q_homo ) #p11
996 l1=l1/l1 [ 2 ]
997 m1=np. cross ( Q_homo , R_homo ) #p11
998 m1=m1/m1 [ 2 ]
999
1000 l2=np. cross ( Q_homo , R_homo ) #p12
1001 l2=l2/l2 [ 2 ]
1002 m2=np. cross ( R_homo , S_homo ) #p12
1003 m2=m2/m2 [ 2 ]
1004
1005 l3=np. cross ( R_homo , S_homo ) #p13
1006 l3=l3/l3 [ 2 ]
1007 m3=np. cross ( S_homo , P_homo ) #p13
1008 m3=m3/m3 [ 2 ]
1009
1010 l4=np. cross ( S_homo , P_homo ) #p14
1011 l4=l4/l4 [ 2 ]
1012 m4=np. cross ( P_homo , Q_homo ) #p14
1013 m4=m4/m4 [ 2 ]
1014
1015 l5=np. cross ( P_homo , R_homo ) #p15

```

```

1016 15=15/15 [2]
1017 m5=np.cross(Q_homo,S_homo) #p15
1018 m5=m5/m5[2]
1019
1020 A=np.zeros((5,6))
1021 A[0]=[11[0]*m1[0],(11[1]*m1[0]+11[0]*m1[1])/2,11[1]*m1[1],(11[0]*m1[2]+11[2]*m1[0])/2,(11[1]*m1[2]+11[2]*m1[1])/2,11[2]*m1[2]]
1022 A[1]=[12[0]*m2[0],(12[1]*m2[0]+12[0]*m2[1])/2,12[1]*m2[1],(12[0]*m2[2]+12[2]*m2[0])/2,(12[1]*m2[2]+12[2]*m2[1])/2,12[2]*m2[2]]
1023 A[2]=[13[0]*m3[0],(13[1]*m3[0]+13[0]*m3[1])/2,13[1]*m3[1],(13[0]*m3[2]+13[2]*m3[0])/2,(13[1]*m3[2]+13[2]*m3[1])/2,13[2]*m3[2]]
1024 A[3]=[14[0]*m4[0],(14[1]*m4[0]+14[0]*m4[1])/2,14[1]*m4[1],(14[0]*m4[2]+14[2]*m4[0])/2,(14[1]*m4[2]+14[2]*m4[1])/2,14[2]*m4[2]]
1025 A[4]=[15[0]*m5[0],(15[1]*m5[0]+15[0]*m5[1])/2,15[1]*m5[1],(15[0]*m5[2]+15[2]*m5[0])/2,(15[1]*m5[2]+15[2]*m5[1])/2,15[2]*m5[2]]
1026 A
1027
1028 from sympy import Matrix
1029 a=Matrix(A)
1030 c=a.nullspace()
1031
1032 c=np.array(c).astype(np.int)
1033 c=c/max(c[0])
1034 c
1035
1036 S_mat = np.zeros((2,2))
1037 S_mat[0,0] = c[0][0]
1038 S_mat[1,0] = 0.5 * c[0,1]
1039 S_mat[0,1] = 0.5 * c[0,1]
1040 S_mat[1,1] = c[0,2]
1041 S_mat
1042
1043 U, D, Vt = np.linalg.svd(S_mat)
1044
1045 K = np.zeros((2,2))
1046 D_sq = np.diag(np.sqrt(D))
1047 K = np.dot(Vt, np.dot(D_sq, Vt))
1048
1049 A=np.matmul(Vt,np.matmul(D_sq,Vt.T))
1050 V = np.matmul(np.linalg.inv(A),[0.5*c[0,3], 0.5*c[0,4]]);
1051 H=[[A[0,0], A[0,1], 0], [A[1,0], A[1,1], 0], [V[0], V[1], 1]];
1052 print(H)
1053 H_inv=np.linalg.inv(H)
1054 H_inv
1055
1056 img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
1057 print(img_coords.T)
1058
1059 new_img_coords=np.zeros([4,2])
1060 for i in range(4):
1061
1062 new_coords=np.matmul(H_inv,img_coords[i].T)
1063 new_coords=new_coords/new_coords[2]
1064 new_coords=np.rint(new_coords).astype(int)
1065 new_img_coords[i,:]=new_coords[0:2]
1066
1067 print(new_img_coords)
1068

```

```

1069 scale1=width/((max(new_img_coords[:,0])-min(new_img_coords[:,0])))
1070 scale2=height/((max(new_img_coords[:,1])-min(new_img_coords[:,1])))
1071 scale=max(scale1, scale2)
1072 print(scale)
1073
1074 tx=np.round(np.min(new_img_coords[:,0]))
1075 ty=np.round(np.min(new_img_coords[:,1]))
1076 print(tx, ty)
1077
1078 new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale)
1079 .astype(int)
1080 new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale).
1081 .astype(int)
1082 new_image=np.zeros((new_height, new_width, 3))
1083 print(new_image.shape)
1084 max_X=im1.shape[1]
1085 max_Y=im1.shape[0]
1086
1087 blank_image=np.zeros((new_height, new_width, 3))
1088 for i in range(0, new_image.shape[1]-1): #x
1089     for j in range(0, new_image.shape[0]-1): #y
1090
1091     k1=i/scale+tx
1092     k2=j/scale+ty
1093     init=np.array((k1, k2, 1))
1094     mapped=np.matmul(H, init)
1095     mapped=np.rint(mapped/mapped[2])
1096     mapped=mapped.astype(int)
1097     #if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
1098     if (mapped[0] > 0 and mapped[1] > 0 and mapped[0] < im1.shape[1] and mapped
1099     [1] < im1.shape[0]):
1100         blank_image[j, i]=im1[mapped[1], mapped[0]]
1101
1102 plt.figure(figsize=(20,20))
1103 plt.imshow(blank_image.astype(np.int))
1104 ii=blank_image.astype(np.uint8)
1105 plt.imsave("onestep2.jpeg", ii)
1106
1107 """
1108 """## **Image 3**
1109
1110 **Point to Point Method**
1111 """
1112
1113 #get the four points from the distorted and world images
1114 def get_4_points(n):
1115     if n==1: #image 1a
1116         #for the image
1117         P=[2092,1481] #x1,y1
1118         Q=[2693,1322] #x2,y2
1119         R=[2667,717] #x3,y3
1120         S=[2057,700] #x4,y4
1121         X_prime=P+Q+R+S
1122         return np.array([P,Q,R,S]), X_prime
1123

```

```

1124 def get_cat_points():
1125     #for the kittens
1126     P_dash=[0,36]
1127     Q_dash=[55,36]
1128     R_dash=[55,0]
1129     S_dash=[0,0]
1130     X=P_dash+Q_dash+R_dash+S_dash
1131     return np.array([P_dash,Q_dash,R_dash,S_dash]),X
1132
1133 image=cv2.imread('/content/drive/My Drive/hw3_Task1_Images/Images/Img3.JPG')
1134 image_orig = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
1135 im1 = np.copy(image_orig)
1136 plt.imshow(im1)
1137 print(im1.shape)
1138
1139 #get the 4 points and find the Homography between them
1140 P1,X=get_4_points(1)
1141 P2,X_prime=get_cat_points()
1142 draw_bbox(im1,P1)
1143 H=Homography_matrix(X,X_prime)
1144 print(H)
1145 H_inv=np.linalg.inv(H)
1146
1147 #get the image coordinates in world plane
1148 width=im1.shape[1]
1149 height=im1.shape[0]
1150 print("width(x)=",width)
1151 print("height(y)=",height)
1152
1153 img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
1154 print(img_coords.T)
1155 new_img_coords=np.zeros([4,2])
1156 for i in range(4):
1157
1158     new_coords=np.matmul(H,img_coords[i].T)
1159     new_coords=new_coords/new_coords[2]
1160     new_coords=np.rint(new_coords).astype(int)
1161     new_img_coords[i,:]=new_coords[0:2]
1162
1163 #the new image will have translation(coordinates negative) and scale errors.
1164     Need to correct them
1164 scale1=width/((max(new_img_coords[:,0])-min(new_img_coords[:,0])))
1165 scale2=height/((max(new_img_coords[:,1])-min(new_img_coords[:,1])))
1166 scale=max(scale1, scale2)
1167 print(scale)
1168 tx=np.round(np.min(new_img_coords[:,0]))
1169 ty=np.round(np.min(new_img_coords[:,1]))
1170 print(tx,ty)
1171
1172 #get the dimensions of the new image
1173 new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale2)
1174     .astype(int)
1174 new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale1)
1175     .astype(int)
1175 new_image=np.zeros((new_height,new_width,3))
1176 new_width, new_height
1177
1178 max_X=im1.shape[1]

```

```

1179 max_Y=im1.shape[0]
1180
1181 #get the indices of each pixel in the new image and find the correspondence
1182     between the distorted and world pixels.
1183 for i in range(0,new_width): #x
1184     for j in range(0,new_height): #y
1185
1186         k1=i/scale1+tx
1187         k2=j/scale2+ty
1188         init=np.array((k1,k2,1))
1189         mapped=np.matmul(H_inv,init)
1190         mapped=np.rint(mapped/mapped[2])
1191         mapped=mapped.astype(int)
1192         if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
1193             new_image[j,i]=im1[mapped[1],mapped[0]]
1194
1195 plt.figure(figsize=(20,20))
1196 plt.imshow(new_image.astype(int))
1197 ii=new_image.astype(np.uint8)
1198 plt.imsave("points3.jpeg",ii)
1199 """
1200 """**Two Step Method**
1201 *Projection Removal*
1202 """
1203
1204 #get the 4 points that correspond to two parallel lines
1205 P1=[2092,1481] #x1,y1
1206 P2=[2693,1322] #x2,y2
1207 P3=[2667,717] #x3,y3
1208 P4=[2057,700] #x4,y4
1209
1210 Q1=P1
1211 Q2=P4
1212 Q3=P2
1213 Q4=P3
1214 im_copy=im1.copy()
1215 ##https://www.geeksforgeeks.org/python-opencv-cv2-line-method/ for drawing
1216     lines on image
1217 line_img=cv2.line(im_copy, tuple(P1), tuple(P2), (255,0,0), 5)
1218 line_img=cv2.line(line_img, tuple(P3), tuple(P4), (255,0,0), 5)
1219 line_img=cv2.line(line_img, tuple(Q1), tuple(Q2), (0,255,0), 5)
1220 line_img=cv2.line(line_img, tuple(Q3), tuple(Q4), (0,255,0), 5)
1221 plt.imshow(line_img)
1222 plt.imsave("parallellines3.jpeg",line_img)
1223
1224 #find the first vanishing point
1225 P_1=np.array(P1)
1226 P_1=np.append(P_1,1)
1227 P_2=np.array(P2)
1228 P_2=np.append(P_2,1)
1229 l1=np.cross(P_1,P_2)
1230
1231 P_3=np.array(P3)
1232 P_3=np.append(P_3,1)
1233 P_4=np.array(P4)
1234 P_4=np.append(P_4,1)
1235 l2=np.cross(P_3,P_4)

```

```

1235
1236 PV1=np.cross(11,12)
1237 PV1=PV1/PV1[2]
1238 print(PV1)
1239
1240 #find the second vanishing point
1241 Q_1=np.array(Q1)
1242 Q_1=np.append(Q_1,1)
1243 Q_2=np.array(Q2)
1244 Q_2=np.append(Q_2,1)
1245 l3=np.cross(Q_1,Q_2)
1246
1247 Q_3=np.array(Q3)
1248 Q_3=np.append(Q_3,1)
1249 Q_4=np.array(Q4)
1250 Q_4=np.append(Q_4,1)
1251 l4=np.cross(Q_3,Q_4)
1252
1253 PV2=np.cross(l3,l4)
1254 PV2=PV2/PV2[2]
1255 print(PV2)
1256
1257 #find the vanishing line
1258 lV=np.cross(PV1,PV2)
1259 lV=lV/lV[2]
1260 lV
1261
1262 #get the H matrix
1263 H2=np.array([[1,0,0],[0,1,0],lV])
1264 H2
1265 print(H2)
1266 H2_inv=np.linalg.inv(H2)
1267
1268 #get the mapping from distorted plane to projection corrected plane
1269 img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
1270 print(img_coords.T)
1271
1272 new_img_coords=np.zeros([4,2])
1273 for i in range(4):
1274
1275     new_coords=np.matmul(H2,img_coords[i].T)
1276     new_coords=new_coords/new_coords[2]
1277     new_coords=np.rint(new_coords).astype(int)
1278     new_img_coords[i,:]=new_coords[0:2]
1279
1280 scale1=width/(max(new_img_coords[:,0])-min(new_img_coords[:,0]))
1281 scale2=height/(max(new_img_coords[:,1])-min(new_img_coords[:,1]))
1282 scale=max(scale1, scale2)
1283 print(scale)
1284
1285 tx=np.round(np.min(new_img_coords[:,0]))
1286 ty=np.round(np.min(new_img_coords[:,1]))
1287 print(tx,ty)
1288
1289 new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale)
1290     .astype(int)
1291 new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale)
1292     .astype(int)

```

```

1291 new_image=np.zeros((new_height,new_width,3))
1292 print(new_image.shape)
1293
1294 max_X=im1.shape[1]
1295 max_Y=im1.shape[0]
1296
1297 for i in range(0,new_width-1): #x
1298     for j in range(0,new_height-1): #y
1299
1300         k1=i/scale+tx
1301         k2=j/scale+ty
1302         init=np.array((k1,k2,1))
1303         mapped=np.matmul(H2_inv,init)
1304         mapped=np.rint(mapped/mapped[2])
1305         mapped=mapped.astype(int)
1306         if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
1307             new_image[j,i]=im1[mapped[1],mapped[0]]
1308
1309 plt.imshow(new_image.astype(int))
1310
1311 ii=new_image.astype(np.uint8)
1312 plt.imsave("proj3.jpeg",ii)
1313
1314 """*Affine Removal*"""
1315
1316 #display the two pairs of perpendicular lines
1317 line_img=cv2.line(im_copy, tuple(P1), tuple(P2), (255,0,0), 5)
1318 line_img=cv2.line(line_img, tuple(P2), tuple(P3), (255,0,0), 5)
1319 line_img=cv2.line(line_img, tuple(P3), tuple(P4), (0,255,0), 5)
1320 line_img=cv2.line(line_img, tuple(P4), tuple(P1), (0,255,0), 5)
1321 plt.imshow(line_img)
1322 plt.imsave("perpendicular3.jpeg",line_img)
1323
1324 #project these points onto the projection corrected image plane
1325 P_homo=np.array(P1)
1326 P_homo=np.append(P_homo,1)
1327 Q_homo=np.array(P2)
1328 Q_homo=np.append(Q_homo,1)
1329 R_homo=np.array(P3)
1330 R_homo=np.append(R_homo,1)
1331 S_homo=np.array(P4)
1332 S_homo=np.append(S_homo,1)
1333
1334 P_dash_homo=np.matmul(H2,P_homo)
1335 P_dash=P_dash_homo/P_dash_homo[-1]
1336 Q_dash_homo=np.matmul(H2,Q_homo)
1337 Q_dash=Q_dash_homo/Q_dash_homo[-1]
1338 R_dash_homo=np.matmul(H2,R_homo)
1339 R_dash=R_dash_homo/R_dash_homo[-1]
1340 S_dash_homo=np.matmul(H2,S_homo)
1341 S_dash=S_dash_homo/S_dash_homo[-1]
1342
1343 #initialize the matrices
1344 A = np.zeros((2,2))
1345 S_mat = np.ones((2,2))
1346 H_c = np.identity(3)
1347
1348 #get the first set of perpendicular lines

```

```

1349 l1 = np.cross(P_dash,Q_dash) #1'
1350 m1 = np.cross(Q_dash,R_dash) #m'
1351
1352 #get the second set of perpendicular lines
1353 l2 = np.cross(R_dash,P_dash)
1354 m2 = np.cross(S_dash,Q_dash)
1355
1356 #get the A and matrix
1357 A[0,0] = l1[0]*m1[0]
1358 A[0,1] = l1[0]*m1[1] + l1[1]*m1[0]
1359 A[1,:] = [l2[0]*m2[0], l2[0]*m2[1] + l2[1]*m2[0]]
1360
1361 B = np.array([-l1[1]*m1[1], -l2[1]*m2[1]])
1362
1363 #Find out S
1364 S_elem = np.matmul(np.linalg.pinv(A),B)
1365 S_mat[0,:] = [S_elem[0], S_elem[1]]
1366 S_mat[1,0] = S_elem[1]
1367 U, D, Vt = np.linalg.svd(S_mat, full_matrices=True)
1368 D_sq = np.zeros((2,2))
1369 D_sq[0,0] = np.sqrt(D[0])
1370 D_sq[1,1] = np.sqrt(D[1])
1371
1372 #Get the value of A (AA'=S)
1373 A_forS = np.matmul(np.matmul(Vt,D_sq),np.transpose(Vt))
1374
1375 H_c[0:2,0:2] = A_forS
1376 H_c_inv=np.linalg.inv(H_c)
1377
1378 H_total=np.matmul(H_c_inv,H2)
1379 H_total_inv=np.linalg.inv(H_total)
1380
1381 #mask the original image to projective + affine corrected image
1382 img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
1383 print(img_coords.T)
1384
1385
1386 new_img_coords=np.zeros([4,2])
1387
1388 for i in range(4):
1389
1390     new_coords=np.matmul(H_total,img_coords[i].T)
1391     new_coords=new_coords/new_coords[2]
1392     new_coords=np.rint(new_coords).astype(int)
1393     new_img_coords[i,:]=new_coords[0:2]
1394
1395
1396 scale1=width/((max(new_img_coords[:,0])-min(new_img_coords[:,0])))
1397 scale2=height/((max(new_img_coords[:,1])-min(new_img_coords[:,1])))
1398 scale=max(scale1, scale2)
1399 print(scale)
1400
1401 tx=np.round(np.min(new_img_coords[:,0]))
1402 ty=np.round(np.min(new_img_coords[:,1]))
1403 print(tx,ty)
1404
1405 new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale2
1406 ) .astype(int)

```

```

1406 new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale1)
1407 .astype(int)
1408 new_image=np.zeros((new_height,new_width,3))
1409 print(new_image.shape)
1410 max_X=im1.shape[1]
1411 max_Y=im1.shape[0]
1412
1413 blank_image=np.zeros((new_height,new_width,3))
1414 for i in range(0,new_image.shape[1]-1): #x
1415     for j in range(0,new_image.shape[0]-1): #y
1416
1417     k1=i/scale1+tx
1418     k2=j/scale2+ty
1419     init=np.array((k1,k2,1))
1420     mapped=np.matmul(H_total_inv,init)
1421     mapped=np.rint(mapped/mapped[2])
1422     mapped=mapped.astype(int)
1423     #if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
1424     if (mapped[0] > 0 and mapped[1] > 0 and mapped[0] < im1.shape[1] and mapped[1] < image.shape[0]):
1425         blank_image[j,i]=im1[mapped[1],mapped[0]]
1426
1427 plt.figure(figsize=(10,10))
1428 plt.imshow(blank_image.astype(int))
1429 ii=blank_image.astype(np.uint8)
1430 plt.imsave("affine3.jpeg",ii)
1431 """
1432 ***One Step Method"""
1433 #4 points for determining orthogonal lines
1434
1435 P=[2092,1481] #x1,y1
1436 Q=[2693,1322] #x2,y2
1437 R=[2667,717] #x3,y3
1438 S=[2057,700] #x4,y4
1439
1440 P_homo=np.array(P)
1441 P_homo=np.append(P_homo,1)
1442 Q_homo=np.array(Q)
1443 Q_homo=np.append(Q_homo,1)
1444 R_homo=np.array(R)
1445 R_homo=np.append(R_homo,1)
1446 S_homo=np.array(S)
1447 S_homo=np.append(S_homo,1)
1448
1449 l1=np.cross(P_homo,Q_homo) #p11
1450 l1=l1/l1[2]
1451 m1=np.cross(Q_homo,R_homo) #p11
1452 m1=m1/m1[2]
1453
1454 l2=np.cross(Q_homo,R_homo) #p12
1455 l2=l2/l2[2]
1456 m2=np.cross(R_homo,S_homo) #p12
1457 m2=m2/m2[2]
1458
1459 l3=np.cross(R_homo,S_homo) #p13
1460 l3=l3/l3[2]
1461 m3=np.cross(S_homo,P_homo) #p13

```

```

1462 m3=m3/m3[ 2 ]
1463
1464 14=np . cross (S_homo,P_homo)  #p14
1465 14=14/14 [ 2 ]
1466 m4=np . cross (P_homo,Q_homo)  #p14
1467 m4=m4/m4[ 2 ]
1468
1469 15=np . cross (P_homo,R_homo)  #p15
1470 15=15/15 [ 2 ]
1471 m5=np . cross (Q_homo,S_homo)  #p15
1472 m5=m5/m5[ 2 ]
1473
1474 A=np . zeros (( 5 ,6 ))
1475 A[0]=[11 [ 0 ]*m1 [ 0 ] ,( 11 [ 1 ]*m1 [ 0 ]+11 [ 0 ]*m1 [ 1 ] )/2,11 [ 1 ]*m1 [ 1 ] ,( 11 [ 0 ]*m1 [ 2 ]+11 [ 2 ]*
1476 m1 [ 0 ] )/2 ,( 11 [ 1 ]*m1 [ 2 ]+11 [ 2 ]*m1 [ 1 ] )/2,11 [ 2 ]*m1 [ 2 ]]
1477 A[1]=[12 [ 0 ]*m2 [ 0 ] ,( 12 [ 1 ]*m2 [ 0 ]+12 [ 0 ]*m2 [ 1 ] )/2,12 [ 1 ]*m2 [ 1 ] ,( 12 [ 0 ]*m2 [ 2 ]+12 [ 2 ]*
1478 m2 [ 0 ] )/2 ,( 12 [ 1 ]*m2 [ 2 ]+12 [ 2 ]*m2 [ 1 ] )/2,12 [ 2 ]*m2 [ 2 ]]
1479 A[2]=[13 [ 0 ]*m3 [ 0 ] ,( 13 [ 1 ]*m3 [ 0 ]+13 [ 0 ]*m3 [ 1 ] )/2,13 [ 1 ]*m3 [ 1 ] ,( 13 [ 0 ]*m3 [ 2 ]+13 [ 2 ]*
1480 m3 [ 0 ] )/2 ,( 13 [ 1 ]*m3 [ 2 ]+13 [ 2 ]*m3 [ 1 ] )/2,13 [ 2 ]*m3 [ 2 ]]
1481 A[3]=[14 [ 0 ]*m4 [ 0 ] ,( 14 [ 1 ]*m4 [ 0 ]+14 [ 0 ]*m4 [ 1 ] )/2,14 [ 1 ]*m4 [ 1 ] ,( 14 [ 0 ]*m4 [ 2 ]+14 [ 2 ]*
1482 m4 [ 0 ] )/2 ,( 14 [ 1 ]*m4 [ 2 ]+14 [ 2 ]*m4 [ 1 ] )/2,14 [ 2 ]*m4 [ 2 ]]
1483 A[4]=[15 [ 0 ]*m5 [ 0 ] ,( 15 [ 1 ]*m5 [ 0 ]+15 [ 0 ]*m5 [ 1 ] )/2,15 [ 1 ]*m5 [ 1 ] ,( 15 [ 0 ]*m5 [ 2 ]+15 [ 2 ]*
1484 m5 [ 0 ] )/2 ,( 15 [ 1 ]*m5 [ 2 ]+15 [ 2 ]*m5 [ 1 ] )/2,15 [ 2 ]*m5 [ 2 ]]
1485 A
1486
1487 from sympy import Matrix
1488 a=Matrix(A)
1489 c=a . nullspace ()
1490 c=np . array (c) . astype (np . int )
1491 c=c/max (c [ 0 ])
1492 c
1493
1494 S_mat = np . zeros (( 2 ,2 ))
1495 S_mat [ 0 ,0 ] = c [ 0 ][ 0 ]
1496 S_mat [ 1 ,0 ] = 0.5 * c [ 0 ,1 ]
1497 S_mat [ 0 ,1 ] = 0.5 * c [ 0 ,1 ]
1498 S_mat [ 1 ,1 ] = c [ 0 ,2 ]
1499 S_mat
1500
1501 U, D, Vt = np . linalg . svd (S_mat)
1502 K = np . zeros (( 2 ,2 ))
1503 D_sq = np . diag (np . sqrt (D))
1504 K = np . dot (Vt, np . dot (D_sq, Vt))
1505 v = np . zeros (( 2 ,1 ))
1506 rhs_vA = np . zeros (( 2 ,1 ))
1507 rhs_vA [ 0 ] = 0.5 * c [ 0 ,3 ]
1508 rhs_vA [ 1 ] = 0.5 * c [ 0 ,4 ]
1509
1510 v = np . dot (np . linalg . inv (K) , rhs_vA)
1511
1512 H = np . zeros (( 3 ,3 ))
1513
1514 #Filling up the homography
1515 H[0]=np . array ([K[0 ,0] , K[0 ,1] , 0])
1516 H[1]=np . array ([K[1 ,0] , K[1 ,1] , 0])
1517 H[2]=np . array ([v[0] , v[1] , 1])
1518
1519 H_inv=np . linalg . inv (H)

```

```

1515 H_inv
1516
1517 img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
1518 print(img_coords.T)
1519
1520
1521 new_img_coords=np.zeros([4,2])
1522 ,,
1523 for i in range(4):
1524
1525     new_coords=np.matmul(H_inv,img_coords[i].T)
1526     new_coords=new_coords/new_coords[2]
1527     new_coords=np.rint(new_coords).astype(int)
1528     new_img_coords[i,:]=new_coords[0:2]
1529 ,,
1530 new_coords=np.matmul(H_inv,img_coords.T)
1531 new_coords=new_coords/new_coords[2]
1532 new_coords=np.rint(new_coords).astype(int)
1533 new_img_coords=new_coords.T[:,2]
1534 new_img_coords
1535 print(new_img_coords)
1536
1537 scale1=width/((max(new_img_coords[:,0])-min(new_img_coords[:,0])))
1538 scale2=height/((max(new_img_coords[:,1])-min(new_img_coords[:,1])))
1539 scale=max(scale1, scale2)
1540 print(scale)
1541
1542 tx=np.round(np.min(new_img_coords[:,0]))
1543 ty=np.round(np.min(new_img_coords[:,1]))
1544 print(tx,ty)
1545
1546 new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale)
1547 .astype(int)
1548 new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale).
1549 .astype(int)
1550 new_image=np.zeros((new_height,new_width,3))
1551 print(new_image.shape)
1552 max_X=im1.shape[1]
1553 max_Y=im1.shape[0]
1554
1555 blank_image=np.zeros((new_height,new_width,3))
1556 for i in range(0,new_image.shape[1]-1): #x
1557     for j in range(0,new_image.shape[0]-1): #y
1558
1559         k1=i/scale+tx
1560         k2=j/scale+ty
1561         init=np.array((k1,k2,1))
1562         mapped=np.matmul(H,init)
1563         mapped=np.rint(mapped/mapped[2])
1564         mapped=mapped.astype(int)
1565         #if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
1566         if (mapped[0] > 0 and mapped[1] > 0 and mapped[0] < im1.shape[1] and mapped
1567         [1] < image.shape[0]):
1568             blank_image[j,i]=im1[mapped[1],mapped[0]]
1569
1570 plt.figure(figsize=(20,20))
1571 plt.imshow(blank_image.astype(np.int))
1572 ii=blank_image.astype(np.uint8)

```

```

1570 plt.imsave("onestep3.jpeg",ii)
1571
1572
1573
1574
1575
1576 """## **My Image 1**
1577
1578 **Point to Point Method**
1579 """
1580
1581 def get_4_points(n):
1582     if n==1: #image 1a
1583         #for the image
1584         P=[262,660] #x1,y1
1585         Q=[353,664] #x2,y2
1586         R= [357,558]#x3,y3
1587         S= [266,558] #x4,y4
1588         X_prime=P+Q+R+S
1589     return np.array([P,Q,R,S]),X_prime
1590
1591 def get_cat_points():
1592     #for the kittens
1593     P_dash=[0,50]
1594     Q_dash=[75,50]
1595     R_dash=[75,0]
1596     S_dash=[0,0]
1597     X=P_dash+Q_dash+R_dash+S_dash
1598     return np.array([P_dash,Q_dash,R_dash,S_dash]),X
1599
1600 image=cv2.imread("/content/drive/My Drive/hw3_Task1_Images/Images/Ellis_island.jpg")
1601 image_orig = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
1602 im1 = np.copy(image_orig)
1603 plt.imshow(im1)
1604 print(im1.shape)
1605 plt.imsave("my_img1.jpeg",im1)
1606
1607 P1,X=get_4_points(1)
1608 P2,X_prime=get_cat_points()
1609
1610 draw_bbox(im1,P1)
1611
1612 H=Homography_matrix(X,X_prime)
1613 print(H)
1614 H_inv=np.linalg.inv(H)
1615
1616 #get the image coordinates in world plane
1617 width=im1.shape[1]
1618 height=im1.shape[0]
1619 print("width(x)=",width)
1620 print("height(y)=",height)
1621
1622 img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
1623 print(img_coords.T)
1624
1625 ,
1626 new_coords=np.matmul(H,img_coords.T)

```

```

1627 new_coords=new_coords/new_coords[2]
1628 new_coords=np.rint(new_coords).astype(int)
1629 new_img_coords=new_coords.T[:,2]
1630 new_img_coords
1631 ,,
1632 new_img_coords=np.zeros([4,2])
1633 for i in range(4):
1634
1635     new_coords=np.matmul(H,img_coords[i].T)
1636     new_coords=new_coords/new_coords[2]
1637     new_coords=np.rint(new_coords).astype(int)
1638     new_img_coords[i,:]=new_coords[0:2]
1639
1640 scale1=width/((max(new_img_coords[:,0])-min(new_img_coords[:,0])))
1641 scale2=height/((max(new_img_coords[:,1])-min(new_img_coords[:,1])))
1642 scale=max(scale1, scale2)
1643 print(scale)
1644
1645 tx=np.round(np.min(new_img_coords[:,0]))
1646 ty=np.round(np.min(new_img_coords[:,1]))
1647 print(tx,ty)
1648
1649 new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale)
1650 .astype(int)
1650 new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale).
1650 .astype(int)
1651 new_image=np.zeros((new_height,new_width,3))
1652 new_width,new_height
1653
1654 max_X=im1.shape[1]
1655 max_Y=im1.shape[0]
1656
1657 for i in range(0,new_width): #x
1658     for j in range(0,new_height): #y
1659
1660         k1=i/scale+tx
1661         k2=j/scale+ty
1662         init=np.array((k1,k2,1))
1663         mapped=np.matmul(H_inv,init)
1664         mapped=np.rint(mapped/mapped[2])
1665         mapped=mapped.astype(int)
1666         if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
1667             new_image[j,i]=im1[mapped[1],mapped[0]]
1668
1669 plt.imshow(new_image.astype(int))
1670 ii=new_image.astype(np.uint8)
1671 plt.imsave("pointsml.jpeg",ii)
1672
1673 """**Two Step Method**
1674
1675 *Projection Removal*
1676 """
1677
1678 P1=[262,660] #x1,y1
1679 P2=[353,664] #x2,y2
1680 P3=[357,558]#x3,y3
1681 P4=[266,558]
1682

```

```

1683 , ,
1684 Q1=[410,120] #13
1685 Q2=[533,8] #13
1686 Q3=[409,374] #14
1687 Q4=[539,312] #14
1688 , ,
1689 Q1=P1
1690 Q2=P4
1691 Q3=P2
1692 Q4=P3
1693 im_copy=im1.copy()
1694 ##https://www.geeksforgeeks.org/python-opencv-cv2-line-method/ for drawing
1695 lines on image
1696 line_img=cv2.line(im_copy, tuple(P1), tuple(P2), (255,0,0), 5)
1697 line_img=cv2.line(line_img, tuple(P3), tuple(P4), (255,0,0), 5)
1698 line_img=cv2.line(line_img, tuple(Q1), tuple(Q2), (0,255,0), 5)
1699 line_img=cv2.line(line_img, tuple(Q3), tuple(Q4), (0,255,0), 5)
1700 plt.imshow(line_img)
1701 plt.imsave("parallel_linesm1.jpeg",line_img)
1702
1703 P_1=np.array(P1)
1704 P_1=np.append(P_1,1)
1705 P_2=np.array(P2)
1706 P_2=np.append(P_2,1)
1707 P_1=P_1
1708 P_2=P_2
1709 P_3=np.array(P3)
1710 P_3=np.append(P_3,1)
1711 P_4=np.array(P4)
1712 P_4=np.append(P_4,1)
1713 P_1=P_1
1714 P_2=P_2
1715 P_3=P_3
1716 P_4=P_4
1717 P_1=P_1
1718 P_2=P_2
1719 P_3=P_3
1720 P_4=P_4
1721 P_1=P_1
1722 P_2=P_2
1723 P_3=P_3
1724 P_4=P_4
1725 P_1=P_1
1726 P_2=P_2
1727 P_3=P_3
1728 P_4=P_4
1729 P_1=P_1
1730 P_2=P_2
1731 P_3=P_3
1732 P_4=P_4
1733 P_1=P_1
1734 P_2=P_2
1735 P_3=P_3
1736 P_4=P_4
1737 P_1=P_1
1738 P_2=P_2
1739 P_3=P_3

```

```

1740 14=np.cross(Q_3,Q_4)
1741 14,
1742 PV2=np.cross(13,14)
1743 PV2=PV2/PV2[2]
1744 print(PV2)
1745
1746 lV=np.cross(PV1,PV2)
1747 lV=lV/lV[2]
1748 lV
1749
1750 H2=np.array([[1,0,0],[0,1,0],lV])
1751 H2
1752 print(H2)
1753 H2_inv=np.linalg.inv(H2)
1754
1755 img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
1756 print(img_coords.T)
1757
1758 new_img_coords=np.zeros([4,2])
1759 for i in range(4):
1760
1761     new_coords=np.matmul(H2,img_coords[i].T)
1762     new_coords=new_coords/new_coords[2]
1763     new_coords=np.rint(new_coords).astype(int)
1764     new_img_coords[i,:]=new_coords[0:2]
1765
1766 scale1=width/(max(new_img_coords[:,0])-min(new_img_coords[:,0]))
1767 scale2=height/(max(new_img_coords[:,1])-min(new_img_coords[:,1]))
1768 scale=max(scale1, scale2)
1769 print(scale)
1770
1771 tx=np.round(np.min(new_img_coords[:,0]))
1772 ty=np.round(np.min(new_img_coords[:,1]))
1773 print(tx,ty)
1774
1775 new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale)
1776 .astype(int)
1777 new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale).
1778 .astype(int)
1779 new_image=np.zeros((new_height,new_width,3))
1780 print(new_image.shape)
1781
1782 max_X=im1.shape[1]
1783 max_Y=im1.shape[0]
1784
1785 for i in range(0,new_width-1): #x
1786     for j in range(0,new_height-1): #y
1787
1788         k1=i/scale+tx
1789         k2=j/scale+ty
1790         init=np.array((k1,k2,1))
1791         mapped=np.matmul(H2_inv,init)
1792         mapped=np.rint(mapped/mapped[2])
1793         mapped=mapped.astype(int)
1794         if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
1795             new_image[j,i]=im1[mapped[1],mapped[0]]
1796
1797 plt.imshow(new_image.astype(int))

```

```

1796 ii=new_image.astype(np.uint8)
1797 plt.imsave("projm1.jpeg",ii)
1798
1799 """*Affine Removal*"""
1800
1801 im_copy=im1.copy()
1802 P1=[262,660] #x1,y1
1803 P2=[353,664] #x2,y2
1804 P3=[357,558]#x3,y3
1805 P4=[266,558]
1806 line_img=cv2.line(im_copy, tuple(P1), tuple(P3), (255,0,0), 5)
1807 line_img=cv2.line(line_img, tuple(P2), tuple(P4), (255,0,0), 5)
1808 line_img=cv2.line(line_img, tuple(P1), tuple(P2), (0,255,0), 5)
1809 line_img=cv2.line(line_img, tuple(P3), tuple(P2), (0,255,0), 5)
1810 plt.imshow(line_img)
1811 plt.imsave("perpendm1.jpeg",line_img)
1812
1813 #map the 4 points onto the projective corrected image plane
1814 P_homo=np.array(P1)
1815 P_homo=np.append(P_homo,1)
1816 Q_homo=np.array(P2)
1817 Q_homo=np.append(Q_homo,1)
1818 R_homo=np.array(P3)
1819 R_homo=np.append(R_homo,1)
1820 S_homo=np.array(P4)
1821 S_homo=np.append(S_homo,1)
1822
1823 P_dash_homo=np.matmul(H2,P_homo)
1824 P_dash=P_dash_homo/P_dash_homo[-1]
1825 Q_dash_homo=np.matmul(H2,Q_homo)
1826 Q_dash=Q_dash_homo/Q_dash_homo[-1]
1827 R_dash_homo=np.matmul(H2,R_homo)
1828 R_dash=R_dash_homo/R_dash_homo[-1]
1829 S_dash_homo=np.matmul(H2,S_homo)
1830 S_dash=S_dash_homo/S_dash_homo[-1]
1831
1832 #initialize the matrices
1833 A = np.zeros((2,2))
1834 S_mat = np.ones((2,2))
1835 H_c = np.identity(3)
1836
1837 #get the first set of perpendicular lines
1838 l1 = np.cross(P_dash,R_dash) #l'
1839 m1 = np.cross(S_dash,Q_dash) #m'
1840
1841 #get the second set of perpendicular lines
1842 l2 = np.cross(P_dash,Q_dash)
1843 m2 = np.cross(Q_dash,R_dash)
1844
1845 #get the A and matrix
1846 A[0,0] = l1[0]*m1[0]
1847 A[0,1] = l1[0]*m1[1] + l1[1]*m1[0]
1848 A[1,:]=[l2[0]*m2[0], l2[0]*m2[1] + l2[1]*m2[0]]
1849
1850 B = np.array([-l1[1]*m1[1], -l2[1]*m2[1]])
1851
1852 #Find out S
1853 S_elem = np.matmul(np.linalg.pinv(A),B)

```

```

1854 S_mat[0,:] = [S_elem[0], S_elem[1]]
1855 S_mat[1,0] = S_elem[1]
1856 U, D, Vt = np.linalg.svd(S_mat, full_matrices=True)
1857 D_sq = np.zeros((2,2))
1858 D_sq[0,0] = np.sqrt(D[0])
1859 D_sq[1,1] = np.sqrt(D[1])
1860
1861 #Get the value of A (AA'=S)
1862 A_forS = np.matmul(np.matmul(Vt, D_sq), np.transpose(Vt))
1863
1864 H_c[0:2,0:2] = A_forS
1865 H_c_inv=np.linalg.inv(H_c)
1866
1867 H_total=np.matmul(H_c_inv,H2)
1868 H_total_inv=np.linalg.inv(H_total)
1869
1870 #mask the original image to projective + affine corrected image
1871 img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
1872 print(img_coords.T)
1873
1874
1875 new_img_coords=np.zeros([4,2])
1876
1877 for i in range(4):
1878
1879     new_coords=np.matmul(H_total,img_coords[i].T)
1880     new_coords=new_coords/new_coords[2]
1881     new_coords=np.rint(new_coords).astype(int)
1882     new_img_coords[i,:]=new_coords[0:2]
1883
1884
1885 scale1=width/(max(new_img_coords[:,0])-min(new_img_coords[:,0]))
1886 scale2=height/(max(new_img_coords[:,1])-min(new_img_coords[:,1]))
1887 scale=max(scale1, scale2)
1888 print(scale)
1889
1890 tx=np.round(np.min(new_img_coords[:,0]))
1891 ty=np.round(np.min(new_img_coords[:,1]))
1892 print(tx,ty)
1893
1894 new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale2)
1895     .astype(int)
1895 new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale1)
1896     .astype(int)
1896 new_image=np.zeros((new_height,new_width,3))
1897 print(new_image.shape)
1898 max_X=im1.shape[1]
1899 max_Y=im1.shape[0]
1900
1901 blank_image=np.zeros((new_height,new_width,3))
1902 for i in range(0,new_image.shape[1]-1): #x
1903     for j in range(0,new_image.shape[0]-1): #y
1904
1905         k1=i/scale1+tx
1906         k2=j/scale2+ty
1907         init=np.array((k1,k2,1))
1908         mapped=np.matmul(H_total_inv,init)
1909         mapped=np.rint(mapped/mapped[2])

```

```

1910 mapped=mapped.astype(int)
1911 #if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
1912 if (mapped[0] > 0 and mapped[1] > 0 and mapped[0] < im1.shape[1] and mapped
1913 [1] < image.shape[0]):
1914     blank_image[j, i]=im1[mapped[1], mapped[0]]
1915
1916 plt.figure(figsize=(20,20))
1917 plt.imshow(blank_image.astype(int))
1918 ii=blank_image.astype(np.uint8)
1919 plt.imsave("affinem1.jpeg", ii)
1920
1921 """**One Step Method**"""
1922 P=[262,660] #x1,y1
1923 Q=[353,664] #x2,y2
1924 R=[357,558]#x3,y3
1925 S=[266,558]
1926
1927 P_homo=np.array(P)
1928 P_homo=np.append(P_homo,1)
1929 Q_homo=np.array(Q)
1930 Q_homo=np.append(Q_homo,1)
1931 R_homo=np.array(R)
1932 R_homo=np.append(R_homo,1)
1933 S_homo=np.array(S)
1934 S_homo=np.append(S_homo,1)
1935
1936 l1=np.cross(P_homo,Q_homo) #p11
1937 l1=l1/l1[2]
1938 m1=np.cross(Q_homo,R_homo) #p11
1939 m1=m1/m1[2]
1940
1941 l2=np.cross(Q_homo,R_homo) #p12
1942 l2=l2/l2[2]
1943 m2=np.cross(R_homo,S_homo) #p12
1944 m2=m2/m2[2]
1945
1946 l3=np.cross(R_homo,S_homo) #p13
1947 l3=l3/l3[2]
1948 m3=np.cross(S_homo,P_homo) #p13
1949 m3=m3/m3[2]
1950
1951 l4=np.cross(S_homo,P_homo) #p14
1952 l4=l4/l4[2]
1953 m4=np.cross(P_homo,Q_homo) #p14
1954 m4=m4/m4[2]
1955
1956 l5=np.cross(P_homo,R_homo) #p15
1957 l5=l5/l5[2]
1958 m5=np.cross(Q_homo,S_homo) #p15
1959 m5=m5/m5[2]
1960
1961 A=np.zeros((5,6))
1962 A[0]=[11[0]*m1[0],(11[1]*m1[0]+11[0]*m1[1])/2,11[1]*m1[1],(11[0]*m1[2]+11[2]*
1963 m1[0])/2,(11[1]*m1[2]+11[2]*m1[1])/2,11[2]*m1[2]]
1963 A[1]=[12[0]*m2[0],(12[1]*m2[0]+12[0]*m2[1])/2,12[1]*m2[1],(12[0]*m2[2]+12[2]*
1964 m2[0])/2,(12[1]*m2[2]+12[2]*m2[1])/2,12[2]*m2[2]]
1964 A[2]=[13[0]*m3[0],(13[1]*m3[0]+13[0]*m3[1])/2,13[1]*m3[1],(13[0]*m3[2]+13[2]*

```

```

1965 m3[0]) /2,(13[1]*m3[2]+13[2]*m3[1]) /2,13[2]*m3[2]]
1966 A[3]=[14[0]*m4[0],(14[1]*m4[0]+14[0]*m4[1]) /2,14[1]*m4[1],(14[0]*m4[2]+14[2]*m4[0]) /2,(14[1]*m4[2]+14[2]*m4[1]) /2,14[2]*m4[2]]
1967 A[4]=[15[0]*m5[0],(15[1]*m5[0]+15[0]*m5[1]) /2,15[1]*m5[1],(15[0]*m5[2]+15[2]*m5[0]) /2,(15[1]*m5[2]+15[2]*m5[1]) /2,15[2]*m5[2]]
1968 A
1969
1970 from sympy import Matrix
1971 a=Matrix(A)
1972 c=a.nullspace()
1973 c=np.array(c).astype(np.int)
1974 c=c/max(c[0])
1975 c
1976 S_mat = np.zeros((2,2))
1977 S_mat[0,0] = c[0][0]
1978 S_mat[1,0] = 0.5 * c[0,1]
1979 S_mat[0,1] = 0.5 * c[0,1]
1980 S_mat[1,1] = c[0,2]
1981 S_mat
1982
1983 U, D, Vt = np.linalg.svd(S_mat)
1984 K = np.zeros((2,2))
1985 D_sq = np.diag(np.sqrt(D))
1986 K = np.dot(Vt, np.dot(D_sq, Vt))
1987 v = np.zeros((2,1))
1988 rhs_vA = np.zeros((2,1))
1989 rhs_vA[0] = 0.5 * c[0,3]
1990 rhs_vA[1] = 0.5 * c[0,4]
1991
1992 v = np.dot(np.linalg.inv(K), rhs_vA)
1993
1994 H = np.zeros((3,3))
1995
1996 #Filling up the homography
1997 H[0]=np.array([K[0,0], K[0,1], 0])
1998 H[1]=np.array([K[1,0], K[1,1], 0])
1999 H[2]=np.array([v[0], v[1], 1])
2000
2001 H_inv=np.linalg.inv(H)
2002 H_inv
2003
2004 img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
2005 print(img_coords.T)
2006
2007 new_img_coords=np.zeros([4,2])
2008
2009 for i in range(4):
2010
2011     new_coords=np.matmul(H_inv,img_coords[i].T)
2012     new_coords=new_coords/new_coords[2]
2013     new_coords=np.rint(new_coords).astype(int)
2014     new_img_coords[i,:]=new_coords[0:2]
2015
2016
2017 scale1=width/(max(new_img_coords[:,0])-min(new_img_coords[:,0]))
2018 scale2=height/(max(new_img_coords[:,1])-min(new_img_coords[:,1]))
2019 scale=max(scale1, scale2)

```

```

2020 print ( scale )
2021
2022 tx=np . round ( np . min ( new_img_coords [ : , 0 ] ) )
2023 ty=np . round ( np . min ( new_img_coords [ : , 1 ] ) )
2024 print ( tx , ty )
2025
2026 new_height=np . round ( ( max ( new_img_coords [ : , 1 ] ) - min ( new_img_coords [ : , 1 ] ) ) * scale )
2027 . astype ( int )
2028 new_width=np . round ( ( max ( new_img_coords [ : , 0 ] ) - min ( new_img_coords [ : , 0 ] ) ) * scale ) .
2029 astype ( int )
2030 new_image=np . zeros ( ( new_height , new_width , 3 ) )
2031 print ( new_image . shape )
2032 max_X=im1 . shape [ 1 ]
2033 max_Y=im1 . shape [ 0 ]
2034
2035 blank_image=np . zeros ( ( new_height , new_width , 3 ) )
2036 for i in range ( 0 , new_image . shape [ 1 ] - 1 ) : #x
2037     for j in range ( 0 , new_image . shape [ 0 ] - 1 ) : #y
2038
2039     k1=i / scale + tx
2040     k2=j / scale + ty
2041     init=np . array ( ( k1 , k2 , 1 ) )
2042     mapped=np . matmul ( H , init )
2043     mapped=np . rint ( mapped / mapped [ 2 ] )
2044     mapped=mapped . astype ( int )
2045     #if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
2046     if ( mapped [ 0 ] > 0 and mapped [ 1 ] > 0 and mapped [ 0 ] < im1 . shape [ 1 ] and mapped
2047         [ 1 ] < image . shape [ 0 ] ) :
2048         blank_image [ j , i ] = im1 [ mapped [ 1 ] , mapped [ 0 ] ]
2049
2050 plt . figure ( figsize=(20,20) )
2051 plt . imshow ( blank_image . astype ( np . int ) )
2052 ii=blank_image . astype ( np . uint8 )
2053 plt . imsave ( "onestepm1 . jpeg" , ii )
2054
2055 """
2056 """## **My Image 2**
2057
2058 **Point to Point Method**
2059 """
2060
2061 def get_4_points ( n ) :
2062     if n==1: #image 1a
2063         #for the image
2064         P=[285,594] #x1,y1
2065         Q=[453,606] #x2,y2
2066         R=[453,500] #x3,y3
2067         S=[294,474] #x4,y4
2068         X_prime=P+Q+R+S
2069         return np . array ( [ P , Q , R , S ] ) , X_prime
2070
2071 def get_cat_points ( ) :
2072     #for the kittens
2073     P_dash=[0,120]
2074     Q_dash=[300,120]

```

```

2075 R_dash=[300,0]
2076 S_dash=[0,0]
2077 X=P_dash+Q_dash+R_dash+S_dash
2078 return np.array([P_dash,Q_dash,R_dash,S_dash]),X
2079 ,,
2080 def get_4_points(n):
2081     if n==1: #image 1a
2082         #for the image
2083         P=[297,501] #x1,y1
2084         Q=[391,496] #x2,y2
2085         R=[295,164] #x3,y3
2086         S=[387,174] #x4,y4
2087         X_prime=P+Q+R+S
2088         return np.array([P,Q,R,S]),X_prime
2089
2090 def get_cat_points():
2091     #for the kittens
2092     P_dash=[0,250]
2093     Q_dash=[100,250]
2094     R_dash=[0,0]
2095     S_dash=[100,0]
2096     X=P_dash+Q_dash+R_dash+S_dash
2097     return np.array([P_dash,Q_dash,R_dash,S_dash]),X
2098 ,,
2099
2100 image=cv2.imread("/content/drive/My Drive/hw3_Task1_Images/Images/castle.jpg")
2101 image_orig = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
2102 im1 = np.copy(image_orig)
2103 plt.imshow(im1)
2104 print(im1.shape)
2105 plt.imsave("my_img2.jpeg",im1)
2106
2107 P1,X=get_4_points(1)
2108 P2,X_prime=get_cat_points()
2109
2110 draw_bbox(im1,P1)
2111
2112 H=Homography_matrix(X,X_prime)
2113 print(H)
2114 H_inv=np.linalg.inv(H)
2115
2116 #get the image coordinates in world plane
2117 width=im1.shape[1]
2118 height=im1.shape[0]
2119 print("width(x)=",width)
2120 print("height(y)=",height)
2121
2122 img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
2123 print(img_coords.T)
2124 ,,
2125
2126 new_coords=np.matmul(H,img_coords.T)
2127 new_coords=new_coords/new_coords[2]
2128 new_coords=np.rint(new_coords).astype(int)
2129 new_img_coords=new_coords.T[:,2]
2130 new_img_coords
2131 ,,
2132 new_img_coords=np.zeros([4,2])

```

```

2133 for i in range(4):
2134
2135     new_coords=np.matmul(H,img_coords[ i ].T)
2136     new_coords=new_coords/new_coords[2]
2137     new_coords=np.rint(new_coords).astype(int)
2138     new_img_coords[ i ,:]=new_coords[0:2]
2139
2140 scale1=width/((max(new_img_coords[:,0])-min(new_img_coords[:,0])))
2141 scale2=height/((max(new_img_coords[:,1])-min(new_img_coords[:,1])))
2142 scale=max(scale1, scale2)
2143 print(scale)
2144
2145 tx=np.round(np.min(new_img_coords[:,0]))
2146 ty=np.round(np.min(new_img_coords[:,1]))
2147 print(tx,ty)
2148
2149 new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale2)
2150     .astype(int)
2150 new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale1)
2151     .astype(int)
2151 new_image=np.zeros((new_height,new_width,3))
2152 new_width,new_height
2153
2154 max_X=im1.shape[1]
2155 max_Y=im1.shape[0]
2156
2157 import math
2158 '''
2159 for i in range(0,new_width): #x
2160     for j in range(0,new_height): #y
2161
2162         k1=i/scale1+tx
2163         k2=j/scale2+ty
2164         init=np.array((k1,k2,1))
2165         mapped=np.matmul(H_inv,init)
2166         mapped=np.rint(mapped/mapped[2])
2167         mapped=mapped.astype(int)
2168         if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
2169             new_image[j,i]=im1[mapped[1],mapped[0]]
2170
2171 '''
2172 def apply_homography_vectorized(image, H):
2173     #To find size of the real world image
2174     H_inv = np.linalg.inv(H)
2175
2176     image_corners = [[0,0],[0,image.shape[0]-1],[image.shape[1]-1,image.shape[0]-1],[image.shape[1]-1,0]]
2177     image_corners = np.asarray(image_corners)
2178     world_corners = np.zeros((4,2))
2179     image_shape = image.shape
2180
2181     for i in range(len(image_corners)):
2182         point_img = np.append(image_corners[ i ],1)
2183         point_real = np.matmul(H,point_img)
2184         point_real = np.asarray((math.floor(point_real[0]/point_real[2]),math.
2185         floor(point_real[1]/point_real[2])))
2186         world_corners[ i ] = point_real.astype(int)

```

```

2187 print(world_corners)
2188 x_min = np.min(world_corners[:, 0]).astype(int)+1
2189 x_max = np.max(world_corners[:, 0]).astype(int)
2190 y_min = np.min(world_corners[:, 1]).astype(int)+1
2191 y_max = np.max(world_corners[:, 1]).astype(int)
2192 x_lim = x_max-x_min
2193 y_lim = y_max-y_min
2194
2195 #IDK stack overflow did this
2196 H_inverse = np.linalg.inv(H)
2197 HC = np.ones((3, x_lim*y_lim)).astype(int)
2198 for indexH in range(x_lim):
2199     HC[0, indexH*y_lim:indexH*y_lim+y_lim] = np.repeat([indexH], y_lim) +
2200     x_min
2201     HC[1, indexH*y_lim:indexH*y_lim+y_lim] = np.arange(y_lim) + y_min
2202 HC_ = np.matmul(H_inverse, HC)
2203 HC_ = (np.round(HC_/HC_[2, :])).astype(int)
2204
2205 #Ensure points lie inside image
2206 boundary = np.logical_and(HC_[0, :] >= 0, HC_[0, :] <= np.int64(image.shape[1]-1))
2207 boundary = np.logical_and(HC_[1, :] >= 0, boundary)
2208 boundary = np.logical_and(HC_[1, :] <= np.int64(image.shape[0]-1), boundary)
2209
2210 world_image = np.zeros((y_lim, x_lim, 3)).astype(int)
2211 for index in np.arange(x_lim*y_lim)[boundary]:
2212     world_image[HC[1, index]-y_min, HC[0, index]-x_min, :] = \
2213         image[HC[1, index], HC[0, index], :]
2214 return world_image.astype(np.uint8)
2215
2216 new_image=apply_homography_vectorized(im1,H)
2217
2218 plt.figure(figsize=(20,20))
2219 plt.imshow(new_image.astype(int))
2220 ii=new_image.astype(np.uint8)
2221 plt.imsave("pointsm2.jpeg",ii)
2222
2223 """**Two Step Method**
2224
2225 *Projection Removal*
2226 """
2227
2228 P1=[289,525] #x1,y1
2229 P2=[332,531] #x2,y2
2230 P3=[335,479] #x3,y3
2231 P4=[292,472]
2232
2233 ,,
2234 Q1=[410,120] #13
2235 Q2=[533,8] #13
2236 Q3=[409,374] #14
2237 Q4=[539,312] #14
2238 ,,
2239 Q1=P1
2240 Q2=P4
2241 Q3=P2
2242 Q4=P3

```

```

2243 im_copy=im1.copy()
2244 ##https://www.geeksforgeeks.org/python-opencv-cv2-line-method/ for drawing
     lines on image
2245 line_img=cv2.line(im_copy, tuple(P1), tuple(P2), (255,0,0), 5)
2246 line_img=cv2.line(line_img, tuple(P3), tuple(P4), (255,0,0), 5)
2247 line_img=cv2.line(line_img, tuple(Q1), tuple(Q2), (0,255,0), 5)
2248 line_img=cv2.line(line_img, tuple(Q3), tuple(Q4), (0,255,0), 5)
2249 plt.imshow(line_img)
2250 plt.imsave("parallel_lines1.jpeg",line_img)
2251
2252 P_1=np.array(P1)
2253 P_1=np.append(P_1,1)
2254 P_2=np.array(P2)
2255 P_2=np.append(P_2,1)
2256 l1=np.cross(P_1,P_2)
2257
2258 P_3=np.array(P3)
2259 P_3=np.append(P_3,1)
2260 P_4=np.array(P4)
2261 P_4=np.append(P_4,1)
2262 l2=np.cross(P_3,P_4)
2263
2264 PV1=np.cross(l1,l2)
2265 PV1=PV1/PV1[2]
2266 print(PV1)
2267
2268 Q_1=np.array(Q1)
2269 Q_1=np.append(Q_1,1)
2270 Q_2=np.array(Q2)
2271 Q_2=np.append(Q_2,1)
2272 l3=np.cross(Q_1,Q_2)
2273
2274 Q_3=np.array(Q3)
2275 Q_3=np.append(Q_3,1)
2276 Q_4=np.array(Q4)
2277 Q_4=np.append(Q_4,1)
2278 l4=np.cross(Q_3,Q_4)
2279 ,,
2280 Q_1=np.array(P1)
2281 Q_1=np.append(Q_1,1)
2282 Q_2=np.array(P3)
2283 Q_2=np.append(Q_2,1)
2284 l5=np.cross(Q_1,Q_2)
2285
2286 Q_3=np.array(P2)
2287 Q_3=np.append(Q_3,1)
2288 Q_4=np.array(P4)
2289 Q_4=np.append(Q_4,1)
2290 l6=np.cross(Q_3,Q_4)
2291 ,,
2292 PV2=np.cross(l3,l4)
2293 PV2=PV2/PV2[2]
2294 print(PV2)
2295
2296 lV=np.cross(PV1,PV2)
2297 lV=lV/lV[2]
2298 lV
2299

```

```

2300 H2=np.array([[1,0,0],[0,1,0],1V])
2301 H2
2302 print(H2)
2303 H2_inv=np.linalg.inv(H2)
2304
2305 img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
2306 print(img_coords.T)
2307
2308 new_img_coords=np.zeros([4,2])
2309 for i in range(4):
2310
2311     new_coords=np.matmul(H2,img_coords[i].T)
2312     new_coords=new_coords/new_coords[2]
2313     new_coords=np.rint(new_coords).astype(int)
2314     new_img_coords[i,:]=new_coords[0:2]
2315
2316 scale1=width/(max(new_img_coords[:,0])-min(new_img_coords[:,0]))
2317 scale2=height/(max(new_img_coords[:,1])-min(new_img_coords[:,1]))
2318 scale=max(scale1, scale2)
2319 print(scale)
2320
2321 tx=np.round(np.min(new_img_coords[:,0]))
2322 ty=np.round(np.min(new_img_coords[:,1]))
2323 print(tx,ty)
2324
2325 new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale)
2326     .astype(int)
2326 new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale).
2327     .astype(int)
2327 new_image=np.zeros((new_height,new_width,3))
2328 print(new_image.shape)
2329
2330 max_X=im1.shape[1]
2331 max_Y=im1.shape[0]
2332
2333 for i in range(0,new_width-1): #x
2334     for j in range(0,new_height-1): #y
2335
2336         k1=i/scale+tx
2337         k2=j/scale+ty
2338         init=np.array((k1,k2,1))
2339         mapped=np.matmul(H2_inv,init)
2340         mapped=np.rint(mapped/mapped[2])
2341         mapped=mapped.astype(int)
2342         if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
2343             new_image[j,i]=im1[mapped[1],mapped[0]]
2344
2345 plt.imshow(new_image.astype(int))
2346 ii=new_image.astype(np.uint8)
2347 plt.imsave("projm2.jpeg",ii)
2348
2349 """*Affine Removal"""
2350
2351 im_copy=im1.copy()
2352 P1=[289,525] #x1,y1
2353 P2=[332,531] #x2,y2
2354 P3=[335,479] #x3,y3
2355 P4=[292,472]

```

```

2356
2357 line_img=cv2.line(im_copy, tuple(P1), tuple(P3), (255,0,0), 5)
2358 line_img=cv2.line(line_img, tuple(P2), tuple(P4), (255,0,0), 5)
2359 line_img=cv2.line(line_img, tuple(P1), tuple(P2), (0,255,0), 5)
2360 line_img=cv2.line(line_img, tuple(P3), tuple(P2), (0,255,0), 5)
2361 plt.imshow(line_img)
2362 plt.imsave("perpendm2.jpeg",line_img)
2363
2364 #map the 4 points onto the projective corrected image plane
2365 P_homo=np.array(P1)
2366 P_homo=np.append(P_homo,1)
2367 Q_homo=np.array(P2)
2368 Q_homo=np.append(Q_homo,1)
2369 R_homo=np.array(P3)
2370 R_homo=np.append(R_homo,1)
2371 S_homo=np.array(P4)
2372 S_homo=np.append(S_homo,1)
2373
2374 P_dash_homo=np.matmul(H2,P_homo)
2375 P_dash=P_dash_homo/P_dash_homo[-1]
2376 Q_dash_homo=np.matmul(H2,Q_homo)
2377 Q_dash=Q_dash_homo/Q_dash_homo[-1]
2378 R_dash_homo=np.matmul(H2,R_homo)
2379 R_dash=R_dash_homo/R_dash_homo[-1]
2380 S_dash_homo=np.matmul(H2,S_homo)
2381 S_dash=S_dash_homo/S_dash_homo[-1]
2382
2383 #initialize the matrices
2384 A = np.zeros((2,2))
2385 S_mat = np.ones((2,2))
2386 H_c = np.identity(3)
2387
2388 #get the first set of perpendicular lines
2389 l1 = np.cross(P_dash,R_dash) #l1
2390 m1 = np.cross(S_dash,Q_dash) #m1
2391
2392 #get the second set of perpendicular lines
2393 l2 = np.cross(P_dash,Q_dash)
2394 m2 = np.cross(Q_dash,R_dash)
2395
2396 #get the A and matrix
2397 A[0,0] = l1[0]*m1[0]
2398 A[0,1] = l1[0]*m1[1] + l1[1]*m1[0]
2399 A[1,:] = [l2[0]*m2[0], l2[0]*m2[1] + l2[1]*m2[0]]
2400
2401 B = np.array([-l1[1]*m1[1], -l2[1]*m2[1]])
2402
2403 #Find out S
2404 S_elem = np.matmul(np.linalg.pinv(A),B)
2405 S_mat[0,:] = [S_elem[0], S_elem[1]]
2406 S_mat[1,0] = S_elem[1]
2407 U, D, Vt = np.linalg.svd(S_mat, full_matrices=True)
2408 D_sq = np.zeros((2,2))
2409 D_sq[0,0] = np.sqrt(D[0])
2410 D_sq[1,1] = np.sqrt(D[1])
2411
2412 #Get the value of A (AA'=S)
2413 A_forS = np.matmul(np.matmul(Vt,D_sq),np.transpose(Vt))

```

```

2414
2415 H_c[0:2,0:2] = A_forS
2416 H_c_inv=np.linalg.inv(H_c)
2417
2418 H_total=np.matmul(H_c_inv,H2)
2419 H_total_inv=np.linalg.inv(H_total)
2420
2421 #mask the original image to projective + affine corrected image
2422 img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
2423 print(img_coords.T)
2424
2425
2426 new_img_coords=np.zeros([4,2])
2427
2428 for i in range(4):
2429
2430     new_coords=np.matmul(H_total,img_coords[i].T)
2431     new_coords=new_coords/new_coords[2]
2432     new_coords=np.rint(new_coords).astype(int)
2433     new_img_coords[i,:]=new_coords[0:2]
2434
2435
2436 scale1=width/((max(new_img_coords[:,0])-min(new_img_coords[:,0])))
2437 scale2=height/((max(new_img_coords[:,1])-min(new_img_coords[:,1])))
2438 scale=max(scale1, scale2)
2439 print(scale)
2440
2441 tx=np.round(np.min(new_img_coords[:,0]))
2442 ty=np.round(np.min(new_img_coords[:,1]))
2443 print(tx,ty)
2444
2445 new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale2)
2446     .astype(int)
2446 new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale1)
2447     .astype(int)
2448 new_image=np.zeros((new_height,new_width,3))
2449 print(new_image.shape)
2450 max_X=im1.shape[1]
2450 max_Y=im1.shape[0]
2451
2452 blank_image=np.zeros((new_height,new_width,3))
2453 for i in range(0,new_image.shape[1]-1): #x
2454     for j in range(0,new_image.shape[0]-1): #y
2455
2456         k1=i/scale1+tx
2457         k2=j/scale2+ty
2458         init=np.array((k1,k2,1))
2459         mapped=np.matmul(H_total_inv,init)
2460         mapped=np.rint(mapped/mapped[2])
2461         mapped=mapped.astype(int)
2462         #if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
2463             if (mapped[0] > 0 and mapped[1] > 0 and mapped[0] < im1.shape[1] and mapped[1] < image.shape[0]):
2464                 blank_image[j,i]=im1[mapped[1],mapped[0]]
2465
2466 plt.figure(figsize=(20,20))
2467 plt.imshow(blank_image.astype(int))
2468 ii=blank_image.astype(np.uint8)

```

```

2469 plt.imsave( "affinem2.jpeg" , ii )
2470
2471 """**One Step Method**"""
2472
2473 P=[286,594] #x1,y1
2474 Q=[397,601] #x2,y2
2475 R=[398,490]#x3,y3
2476 S=[292,473] #x4,y4
2477
2478 P_homo=np.array(P)
2479 P_homo=np.append(P_homo,1)
2480 Q_homo=np.array(Q)
2481 Q_homo=np.append(Q_homo,1)
2482 R_homo=np.array(R)
2483 R_homo=np.append(R_homo,1)
2484 S_homo=np.array(S)
2485 S_homo=np.append(S_homo,1)
2486
2487 l1=np.cross(P_homo,Q_homo) #p11
2488 l1=l1/l1[2]
2489 m1=np.cross(Q_homo,R_homo) #p11
2490 m1=m1/m1[2]
2491
2492 l2=np.cross(Q_homo,R_homo) #p12
2493 l2=l2/l2[2]
2494 m2=np.cross(R_homo,S_homo) #p12
2495 m2=m2/m2[2]
2496
2497 l3=np.cross(R_homo,S_homo) #p13
2498 l3=l3/l3[2]
2499 m3=np.cross(S_homo,P_homo) #p13
2500 m3=m3/m3[2]
2501
2502 l4=np.cross(S_homo,P_homo) #p14
2503 l4=l4/l4[2]
2504 m4=np.cross(P_homo,Q_homo) #p14
2505 m4=m4/m4[2]
2506
2507 l5=np.cross(P_homo,R_homo) #p15
2508 l5=l5/l5[2]
2509 m5=np.cross(Q_homo,S_homo) #p15
2510 m5=m5/m5[2]
2511
2512 A=np.zeros((5,6))
2513 A
2514 A[0]=[11[0]*m1[0],(11[1]*m1[0]+11[0]*m1[1])/2,11[1]*m1[1],(11[0]*m1[2]+11[2]*m1[0])/2,(11[1]*m1[2]+11[2]*m1[1])/2,11[2]*m1[2]]
2515 A[1]=[12[0]*m2[0],(12[1]*m2[0]+12[0]*m2[1])/2,12[1]*m2[1],(12[0]*m2[2]+12[2]*m2[0])/2,(12[1]*m2[2]+12[2]*m2[1])/2,12[2]*m2[2]]
2516 A[2]=[13[0]*m3[0],(13[1]*m3[0]+13[0]*m3[1])/2,13[1]*m3[1],(13[0]*m3[2]+13[2]*m3[0])/2,(13[1]*m3[2]+13[2]*m3[1])/2,13[2]*m3[2]]
2517 A[3]=[14[0]*m4[0],(14[1]*m4[0]+14[0]*m4[1])/2,14[1]*m4[1],(14[0]*m4[2]+14[2]*m4[0])/2,(14[1]*m4[2]+14[2]*m4[1])/2,14[2]*m4[2]]
2518 A[4]=[15[0]*m5[0],(15[1]*m5[0]+15[0]*m5[1])/2,15[1]*m5[1],(15[0]*m5[2]+15[2]*m5[0])/2,(15[1]*m5[2]+15[2]*m5[1])/2,15[2]*m5[2]]
2519 A
2520
2521 from sympy import Matrix

```

```

2522 a=Matrix(A)
2523 c=a.nullspace()
2524 c=np.array(c).astype(np.int)
2525 c=c/max(c[0])
2526 c
2527
2528 S_mat = np.zeros((2,2))
2529 S_mat[0,0] = c[0][0]
2530 S_mat[1,0] = 0.5 * c[0,1]
2531 S_mat[0,1] = 0.5 * c[0,1]
2532 S_mat[1,1] = c[0,2]
2533 S_mat
2534
2535 U, D, Vt = np.linalg.svd(S_mat)
2536 K = np.zeros((2,2))
2537 D_sq = np.diag(np.sqrt(D))
2538 K = np.dot(Vt, np.dot(D_sq, Vt))
2539
2540 v = np.zeros((2,1))
2541 rhs_vA = np.zeros((2,1))
2542 rhs_vA[0] = 0.5 * c[0,3]
2543 rhs_vA[1] = 0.5 * c[0,4]
2544
2545 v = np.dot(np.linalg.inv(K), rhs_vA)
2546
2547 H = np.zeros((3,3))
2548
2549 #Filling up the homography
2550 H[0]=np.array([K[0,0], K[0,1], 0])
2551 H[1]=np.array([K[1,0], K[1,1], 0])
2552 H[2]=np.array([v[0], v[1], 1])
2553
2554 H_inv=np.linalg.inv(H)
2555 H_inv
2556
2557 img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
2558 print(img_coords.T)
2559
2560 new_img_coords=np.zeros([4,2])
2561
2562 for i in range(4):
2563
2564     new_coords=np.matmul(H_inv, img_coords[i].T)
2565     new_coords=new_coords/new_coords[2]
2566     new_coords=np.rint(new_coords).astype(int)
2567     new_img_coords[i,:]=new_coords[0:2]
2568
2569
2570 scale1=width/(max(new_img_coords[:,0])-min(new_img_coords[:,0]))
2571 scale2=height/(max(new_img_coords[:,1])-min(new_img_coords[:,1]))
2572 scale=max(scale1, scale2)
2573 print(scale)
2574
2575 tx=np.round(np.min(new_img_coords[:,0]))
2576 ty=np.round(np.min(new_img_coords[:,1]))
2577 print(tx, ty)
2578
2579 new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale)

```

```

. astype( int )
2580 new_width=np. round(( max( new_img_coords [:,0] ) -min( new_img_coords [:,0] ) )* scale ) .
    astype( int )
2581 new_image=np. zeros( ( new_height , new_width ,3 ) )
2582 print( new_image . shape )
2583 max_X=im1 . shape [1]
2584 max_Y=im1 . shape [0]
2585
2586 blank_image=np. zeros( ( new_height , new_width ,3 ) )
2587 for i in range(0,new_image . shape [1] -1): #x
2588     for j in range(0,new_image . shape [0] -1): #y
2589
2590         k1=i / scale+tx
2591         k2=j / scale+ty
2592         init=np. array( ( k1 , k2 ,1 ) )
2593         mapped=np. matmul( H, init )
2594         mapped=np. rint( mapped / mapped [2] )
2595         mapped=mapped. astype( int )
2596         #if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
2597             if ( mapped [0] > 0 and mapped [1] > 0 and mapped [0] < im1 . shape [1] and mapped
2598 [1] < image . shape [0] ):
2599                 blank_image [j , i]=im1 [ mapped [1] , mapped [0] ]
2600 plt . figure( figsize=(20,20) )
2601 plt . imshow( blank_image . astype( np. int ) )
2602 ii=blank_image . astype( np. uint8 )
2603 plt . imsave( "onestepM2. jpeg" , ii )

```