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.

$$\vec{x_d} = \mathrm{H}\vec{x_w} \tag{1}$$

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

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

The matrix H is

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

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

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

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

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

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

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.

$$\vec{x_d} = \mathrm{H}\vec{x_w} \tag{4}$$

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.

$$\vec{x_w} = \mathrm{H}^{-1}\vec{x_d} \tag{5}$$

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

$$H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & 1 \end{bmatrix} \tag{6}$$

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.
$$\vec{x_p} = \mathrm{H2}\vec{x_d} \tag{7}$$
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

$$L = \mathrm{H}_c^T L'$$

$$M = \mathrm{H}_c^T M'$$

The conic $C_\infty'^*$ 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 simplyfying 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.

$$s_{11}m_1'l_1' + s_{12}(m_1'l_2'm_2'l_1') + s_{22}m_2'l_2' = 0$$

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 condtion. 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}

$$\vec{x_w} = \mathrm{H}_c^{-1}\vec{x_p}$$

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 simplyfying 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 simplyfying 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 5x6 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 cosidered equal to width and height)

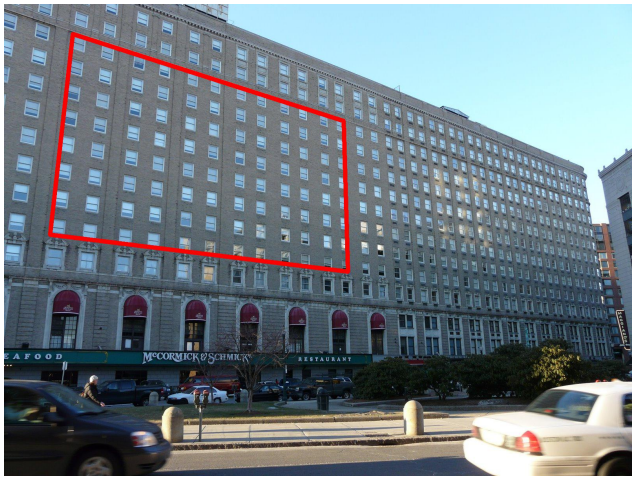

Figure 2: Bounding Box for 4 points

Figure 3: Distortion Removed using Point to Point Correspondence

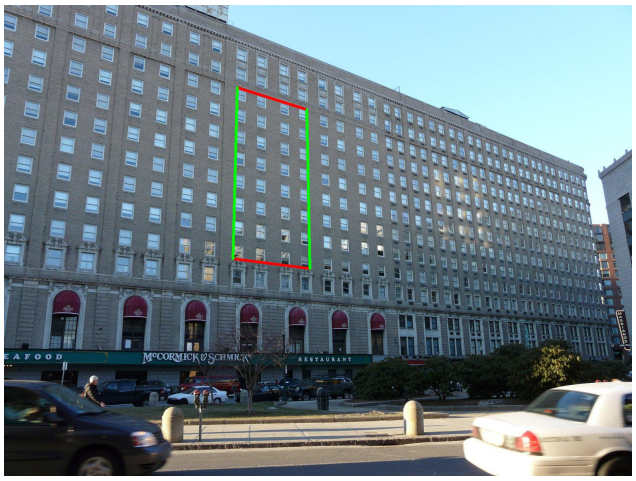

Figure 4: Parallel lines for Projective Distortion Removal

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.

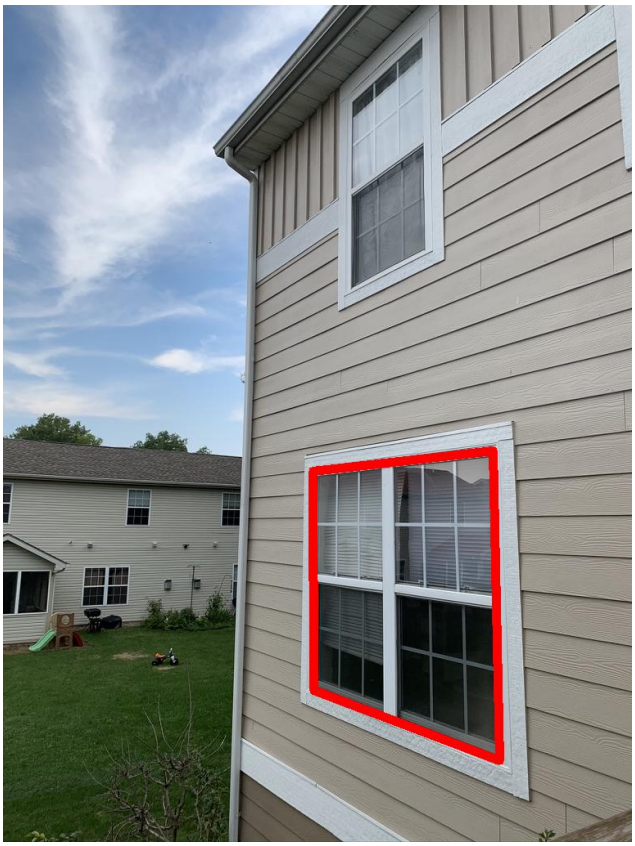

Figure 9: Bounding box for 4 points

Figure 10: Distortion Removal using Point to point method

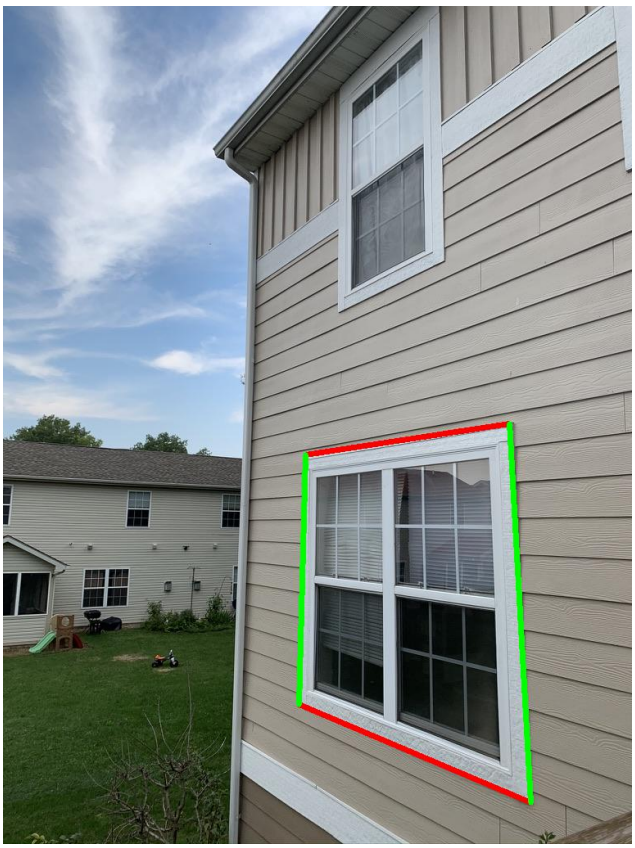

Figure 11: Parallel lines for Projective Distortion removal

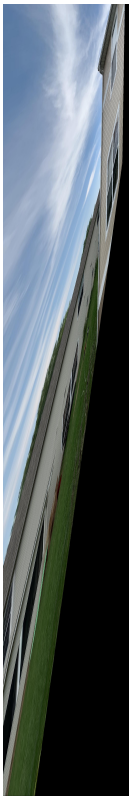

Figure 12: Output image after Projective Correction(Scaled in latex to display full)

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

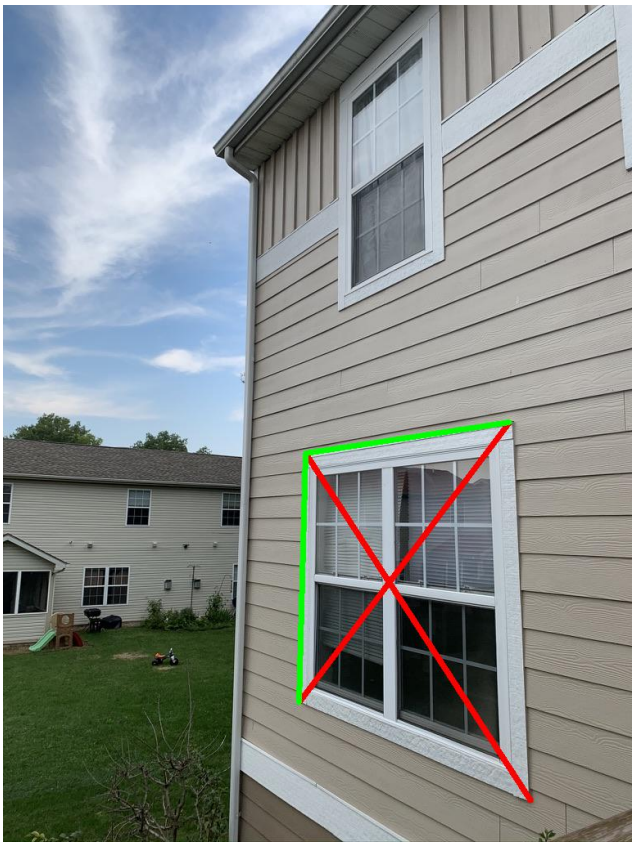

Figure 13: Orthogonal lines for Affine Distortion removal

Figure 14: Output after removing Affine Distortion

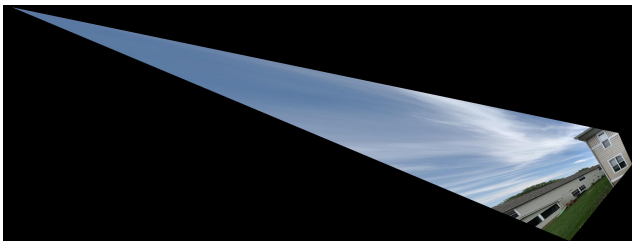

Figure 15: Distortion Removal using One Step Method

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

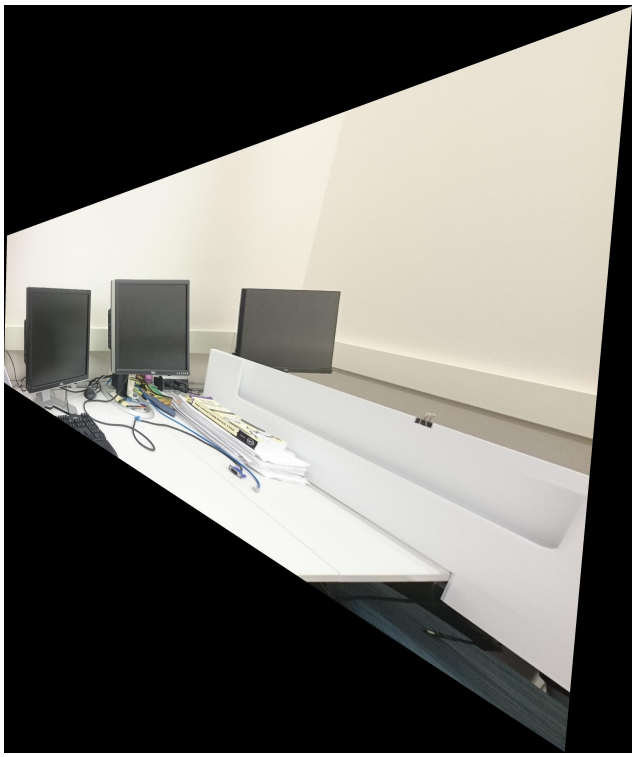

Figure 18: Distortion Correction using Point to Point Correspondence

Figure 19: Parallel Lines in the image

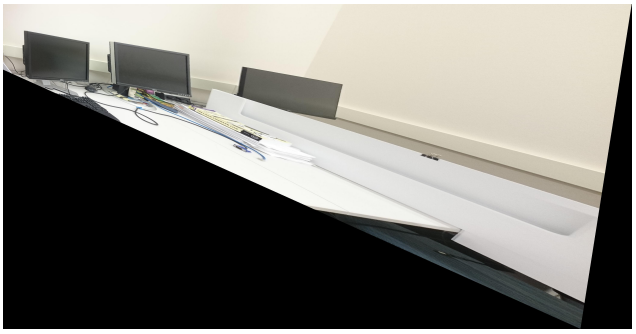

Figure 20: Image after removing Projective Distortion

Figure 21: Orthognal lines for Affine Removal

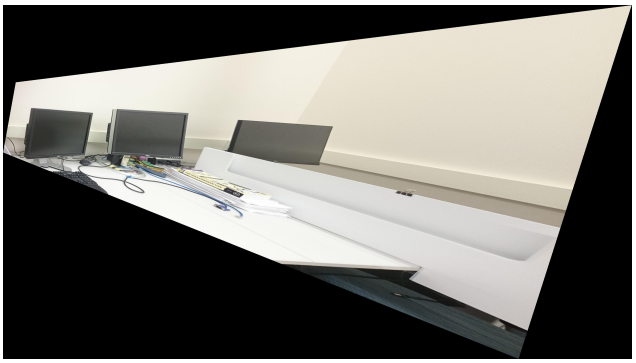

Figure 22: Affine Distortion Removed Image

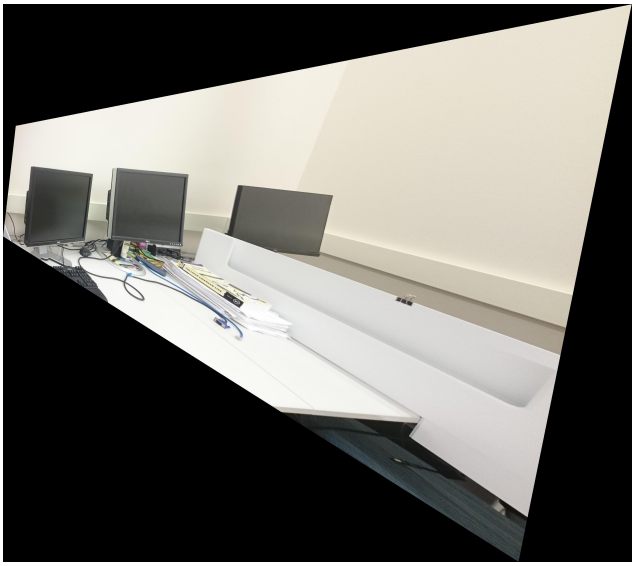

Figure 23: Distortion Removal using One step method

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

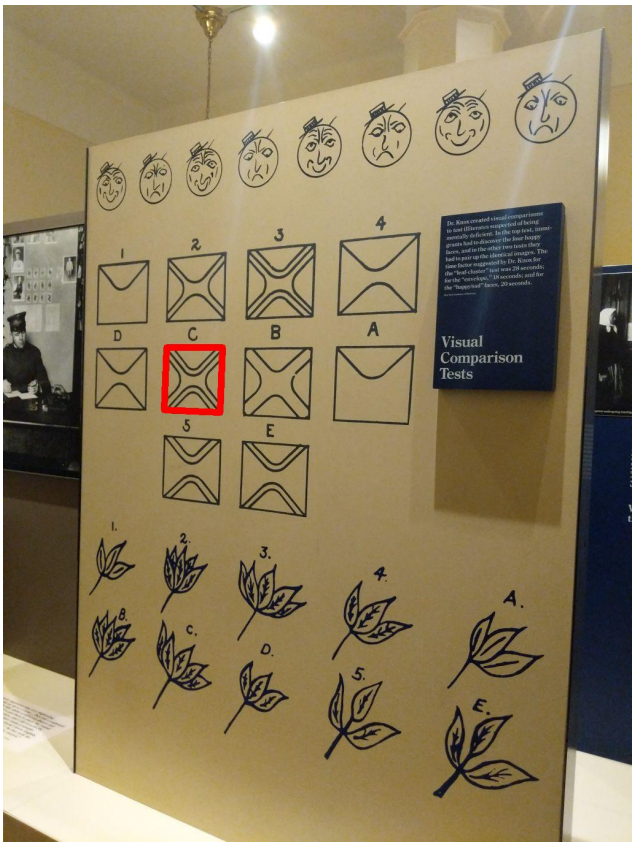

Figure 25: Bounding Box for 4 points

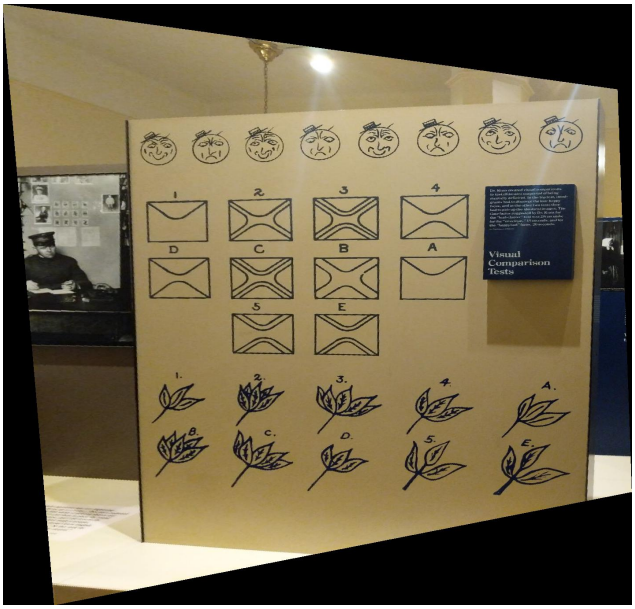

Figure 26: Distortion Removal using Point to Point Method

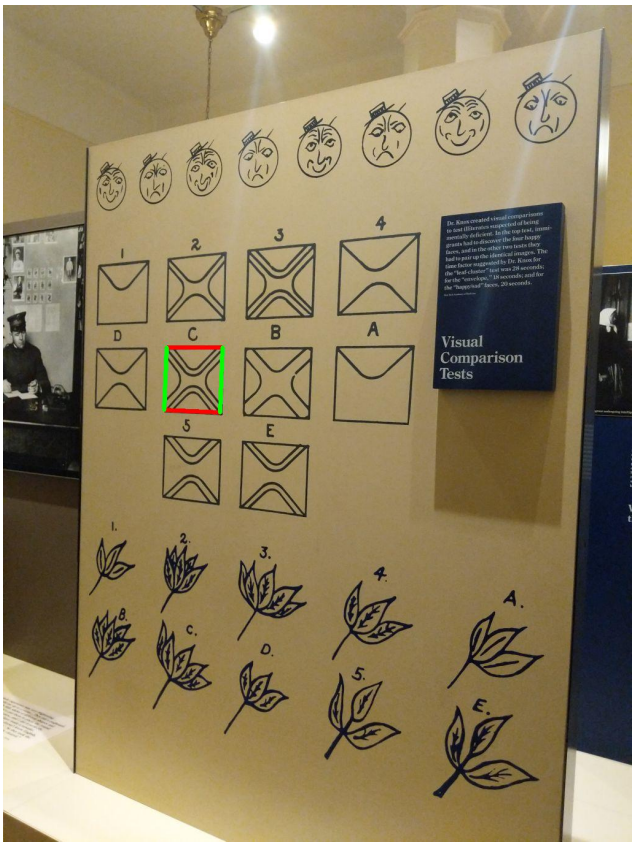

Figure 27: Parallel lines for Projective Removal

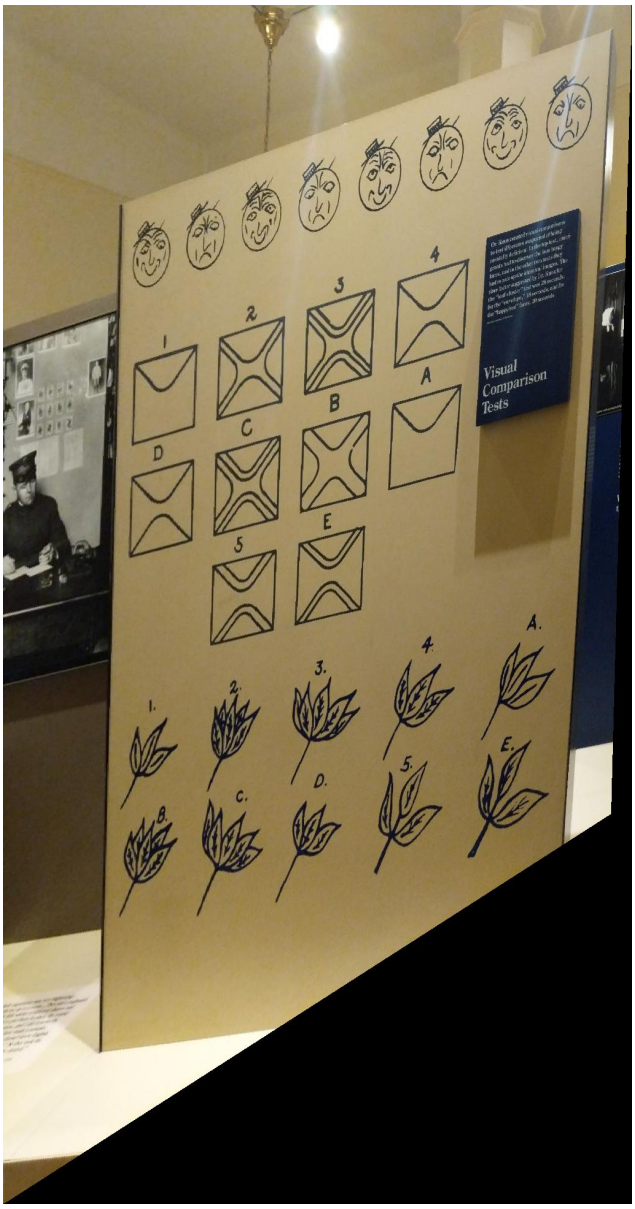

Figure 28: Projective Corrected Output

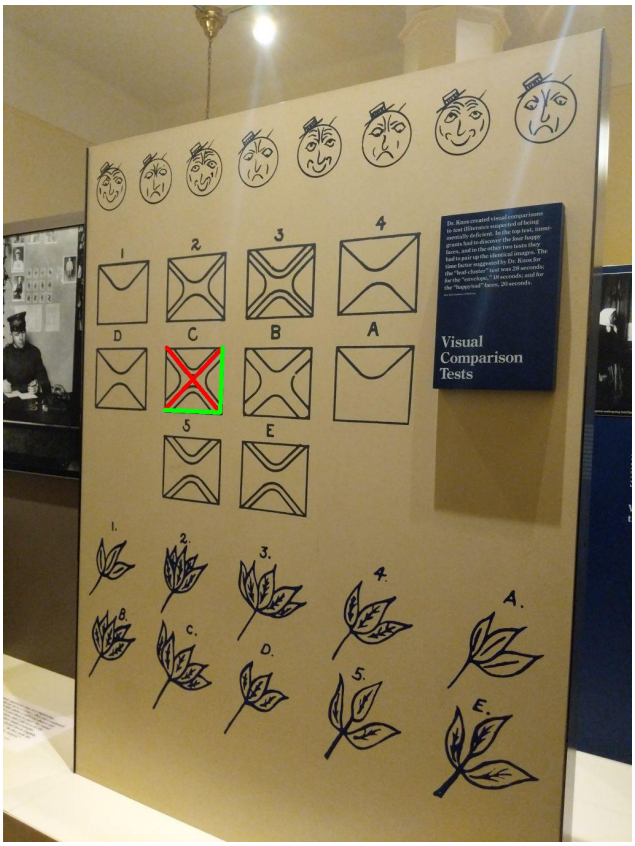

Figure 29: Orthogonal Lines for Affine Removal

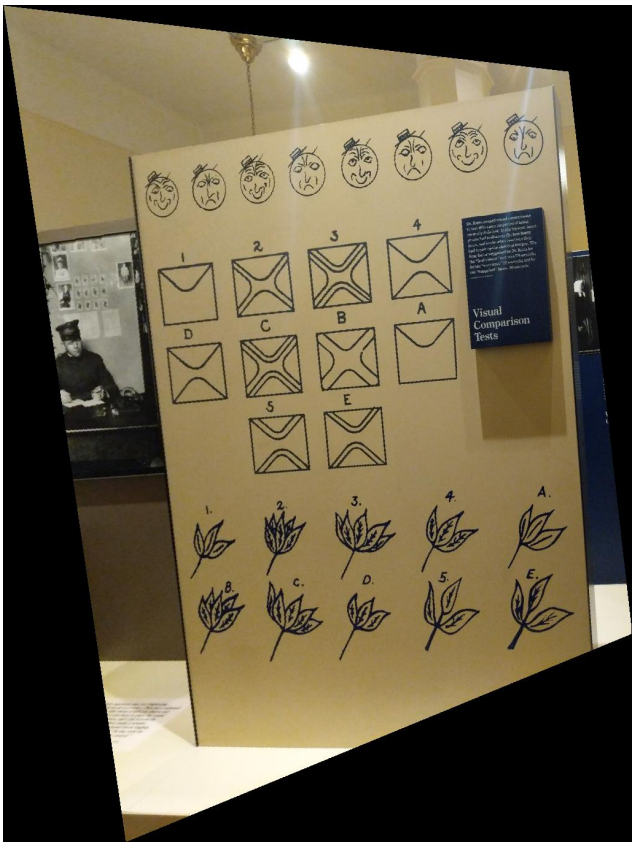

Figure 30: Affine Corrected Output

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

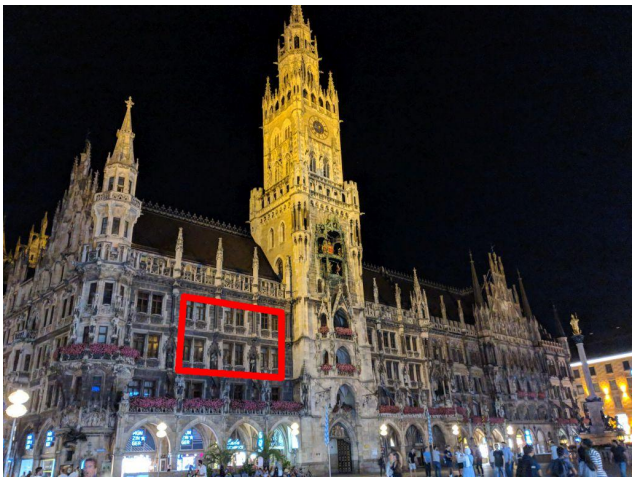

Figure 33: Bounding Box for 4 points

Figure 34: Distortion Removal using Point to Point Method

Figure 35: Parallel lines for Projective Distortion Removal

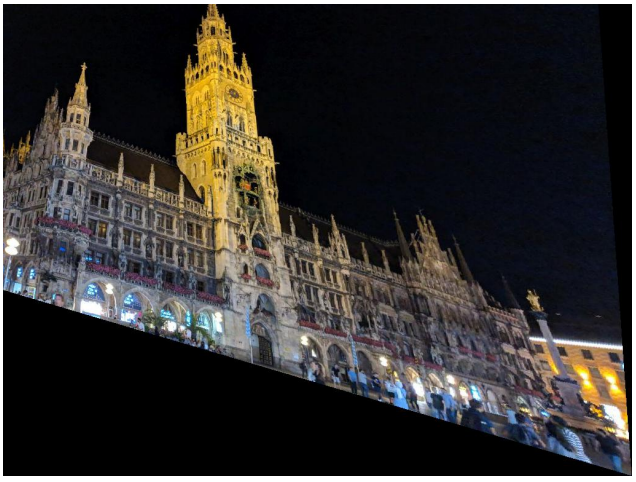

Figure 36: Projective Corrected Output

Figure 37: Perpendicular lines for Affine Dsitortion Removal

Figure 38: Affine Corrected Image

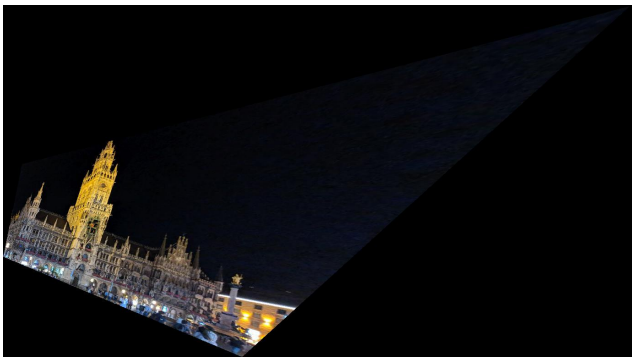

Figure 39: Distortion Removal using One Step Method

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.

```
img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
# using for loop for mapping image corners to world coordinates
for i in range(4):
  new_coords=np.matmul(H_inv,img_coords[i].T)
  new_coords=new_coords/new_coords[2]
  new_coords=np.rint(new_coords).astype(int)
  new_img_coords[i,:]=new_coords[0:2]

#using vectorized operation
new_coords=np.matmul(H_inv,img_coords.T)
new_coords=new_coords/new_coords[2]
new_coords=np.rint(new_coords).astype(int)
new_img_coords=new_coords.T[:,:2]
new_img_coords
```

6 Code Listings

```
# -*- coding: utf-8 -*-
"""hw3_img1.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1iKa8BG50PUYylC-9OuSHe3axj2L5VV8S

Often used functions:
"""

#to check to validitiy of the 4 points, we draw a bounding box aroud the image.
def draw_bbox(image,points):
```

```
  copy_image=image.copy()
  #draw bounding box to check correctnes of the points
  points_reshape = points.reshape((-1, 1, 2))
  color = (255, 0, 0)
  im1_bbox = cv2.polylines(copy_image, [points_reshape], isClosed=True,color=
    color,thickness = 10)
  cv2.imwrite("bbox.jpg",im1_bbox)
  plt.axis('off')
  plt.imshow(im1_bbox)
  plt.imsave("bounding box1.jpeg",im1_bbox)

def Homography_matrix(X,X_prime):
  '''
  This function takes input X and X_prime and computes the homography between
    them. Returns a 3x3 non-singular matrix
  H.
  '''
  H = np.ones((3,3))  #Initialise the Homography matrix
  A = np.zeros((8,8)) #initialise a matrix


  C=X_prime

  #build the A matrix column by column
  for i in range(0,4):
    A[2*i,0]=X[2*i]
    A[2*i,1]=X[2*i+1]
    A[2*i,2]=1
    A[2*i+1,3]=X[2*i]
    A[2*i+1,4]=X[2*i+1]
    A[2*i+1,5]=1
    A[2*i,6]=X[2*i]*X_prime[2*i]
    A[2*i+1,6]=X[2*i]*X_prime[2*i+1]
    A[2*i,7]=(X[2*i+1]*X_prime[2*i])
    A[2*i+1,7]=(X[2*i+1]*X_prime[2*i+1])
  A[:,6:]=-A[:,6:]

  #B=(A^-1)C
  B=np.dot(np.linalg.inv(A),C)

  #reshape the B vector to get the homography matrix
  H[0,0]=B[0]
  H[0,1]=B[1]
  H[0,2]=B[2]
  H[1,0]=B[3]
  H[1,1]=B[4]
  H[1,2]=B[5]
  H[2,0]=B[6]
  H[2,1]=B[7]
  H[2,2]=1
  return H

"""## **Image 1**

**Point to Point Method**
"""

import numpy as np
```

```
import matplotlib.pyplot as plt
import cv2
from PIL import Image
import math

#in order to get a better image made use of multiple windows and assumed the image coordinates in world plane accordingly.
def get_4_points(n):
  if n==1: #image 1a
    #for the image
    P=[117,586]   #x1,y1
    Q=[876,680]  #x2,y2
    R=[865,297] #x3,y3
    S=[175,76]  #x4,y4
    X_prime=P+Q+R+S
    return np.array([P,Q,R,S]),X_prime

#assume distance between the edges of two windows to be equal to the window size
def get_cat_points():
  #for the kittens
  P_dash=[0,85*15]
  Q_dash=[75*23,85*15]
  R_dash=[75*23,0]
  S_dash=[0,0]
  X=P_dash+Q_dash+R_dash+S_dash
  return np.array([P_dash,Q_dash,R_dash,S_dash]),X

#load and display the image
image=cv2.imread("/content/drive/My Drive/hw3_Task1_Images/Images/Img1.JPG")
image_orig = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
im1 = np.copy(image_orig)
plt.imshow(im1)
print(im1.shape)

#get the points in the image and world frames
P1,X=get_4_points(1)
P2,X_prime=get_cat_points()

draw_bbox(im1,P1)
#compute the Homography between the points
H=Homography_matrix(X,X_prime)
print(H)
H_inv=np.linalg.inv(H)

#get the image coordinates in world plane. We map the 4 corner points of the image to the world coordinates
#to get the size of the image in world coordinates.
width=im1.shape[1]
height=im1.shape[0]
print("width(x)=",width)
print("height(y)=",height)

img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
print(img_coords.T)

new_coords=np.matmul(H,img_coords.T)
new_coords=new_coords/new_coords[2]
```

```
new_coords=np.rint(new_coords).astype(int)
new_img_coords=new_coords.T[:,:2]
new_img_coords
print(new_img_coords)

'''
We can see from the projected corner points that the origin in this new coordinate is not at (0,0). Hence we need
to find the appropriate translation to bring the coordinates along origin.
'''

#The output image coordinates are very huge and take long time to compute. Hence we will scale the coordinates to
#get a reasonable image size with which we can work with.
scale1=width/(max(new_img_coords[:,0])-min(new_img_coords[:,0]))
scale2=height/(max(new_img_coords[:,1])-min(new_img_coords[:,1]))
scale=max(scale1, scale2)
print(scale)

#find the translation to bring the coordinates to positive values
tx=np.round(np.min(new_img_coords[:,0]))
ty=np.round(np.min(new_img_coords[:,1]))
print(tx,ty)

'''
Find out the shape of the new image in world coordinates.
'''

new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale2).astype(int)
new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale1).astype(int)
new_image=np.zeros((new_height,new_width,3))

max_X=im1.shape[1]
max_Y=im1.shape[0]



'''
To find the mapping of every point in the new image to the distorted image, we multiply every point with the inverse
of the Homography - i.e. map every point in real world coordinates to the image plane and find the correspondence.
'''
for i in range(0,new_width):  #x
  for j in range(0,new_height): #y

    k1=i/scale1+tx
    k2=j/scale2+ty
    init=np.array((k1,k2,1))
    mapped=np.matmul(H_inv,init)
    mapped=np.rint(mapped/mapped[2])
    mapped=mapped.astype(int)
    if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
      new_image[j,i]=im1[mapped[1],mapped[0]]

plt.figure(figsize=(20,20))
```

```
plt.imshow(new_image.astype(int))
ii=new_image.astype(np.uint8)
plt.imsave("points1.jpeg",ii)




"""**Vanishing Line Method**

*Projection Removal*
"""

#pick the 4 points for two pairs of parallel lines
P1=[583,646]   #x1,y1
P2=[779,671]  #x2,y2
P3=[771,266] #x3,y3
P4=[594,210]  #x4,y4

'''
Q1=[410,120] #l3
Q2=[533,8] #l3
Q3=[409,374] #l4
Q4=[539,312] #l4
'''
Q1=P1
Q2=P4
Q3=P2
Q4=P3
im_copy=im1.copy()
#display the lines
##https://www.geeksforgeeks.org/python-opencv-cv2-line-method/ for drawing lines on image
line_img=cv2.line(im_copy, tuple(P1), tuple(P2), (255,0,0), 5)
line_img=cv2.line(line_img, tuple(P3), tuple(P4), (255,0,0), 5)
line_img=cv2.line(line_img, tuple(Q1), tuple(Q2), (0,255,0), 5)
line_img=cv2.line(line_img, tuple(Q3), tuple(Q4), (0,255,0), 5)
plt.imshow(line_img)
plt.imsave("parallel_lines1.jpeg",line_img)

#get the first vanishing point
P_1=np.array(P1)
P_1=np.append(P_1,1)
P_2=np.array(P2)
P_2=np.append(P_2,1)
l1=np.cross(P_1,P_2)

P_3=np.array(P3)
P_3=np.append(P_3,1)
P_4=np.array(P4)
P_4=np.append(P_4,1)
l2=np.cross(P_3,P_4)

PV1=np.cross(l1,l2)
PV1=PV1/PV1[2]
print(PV1)

#get the second vanishing point
```

```
Q_1=np.array(Q1)
Q_1=np.append(Q_1,1)
Q_2=np.array(Q2)
Q_2=np.append(Q_2,1)
l3=np.cross(Q_1,Q_2)

Q_3=np.array(Q3)
Q_3=np.append(Q_3,1)
Q_4=np.array(Q4)
Q_4=np.append(Q_4,1)
l4=np.cross(Q_3,Q_4)
'''
Q_1=np.array(P1)
Q_1=np.append(Q_1,1)
Q_2=np.array(P3)
Q_2=np.append(Q_2,1)
l3=np.cross(Q_1,Q_2)

Q_3=np.array(P2)
Q_3=np.append(Q_3,1)
Q_4=np.array(P4)
Q_4=np.append(Q_4,1)
l4=np.cross(Q_3,Q_4)
'''
PV2=np.cross(l3,l4)
PV2=PV2/PV2[2]
print(PV2)

#find the vanishing line and H matrix
lV=np.cross(PV1,PV2)
lV=lV/lV[2]
lV

H2=np.array([[1,0,0],[0,1,0],lV])
H2
print(H2)
H2_inv=np.linalg.inv(H2)

#get the image dimension in world coordinates and apply appropriate scaling
img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
print(img_coords.T)

new_img_coords=np.zeros([4,2])
for i in range(4):

  new_coords=np.matmul(H2,img_coords[i].T)
  new_coords=new_coords/new_coords[2]
  new_coords=np.rint(new_coords).astype(int)
  new_img_coords[i,:]=new_coords[0:2]

scale1=width/(max(new_img_coords[:,0])-min(new_img_coords[:,0]))
scale2=height/(max(new_img_coords[:,1])-min(new_img_coords[:,1]))
scale=max(scale1, scale2)
print(scale)

tx=np.round(np.min(new_img_coords[:,0]))
ty=np.round(np.min(new_img_coords[:,1]))
print(tx,ty)
```

```
292
293 new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale)
        .astype(int)
294 new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale).
        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)
```

```
R_dash=R_dash_homo/R_dash_homo[-1]
S_dash_homo=np.matmul(H2,S_homo)
S_dash=S_dash_homo/S_dash_homo[-1]

#initialize the matrices
A = np.zeros((2,2))
S_mat = np.ones((2,2))
H_c = np.identity(3)

#get the first set of perpendicular lines
l1 = np.cross(P_dash,R_dash) #l'
m1 = np.cross(S_dash,Q_dash) #m'

#get the second set of perpendicular lines
l2 = np.cross(P_dash,Q_dash)
m2 = np.cross(Q_dash,R_dash)

#get the A and  matrix
A[0,0] = l1[0]*m1[0]
A[0,1] = l1[0]*m1[1] + l1[1]*m1[0]
A[1,:] = [l2[0]*m2[0], l2[0]*m2[1] + l2[1]*m2[0]]

B = np.array([-l1[1]*m1[1], -l2[1]*m2[1]])

#Find out S
S_elem = np.matmul(np.linalg.pinv(A),B)
S_mat[0,:] = [S_elem[0], S_elem[1]]
S_mat[1,0] = S_elem[1]
U, D, Vt = np.linalg.svd(S_mat, full_matrices=True)
D_sq = np.zeros((2,2))
D_sq[0,0] = np.sqrt(D[0])
D_sq[1,1] = np.sqrt(D[1])

#Get the value of A (AA'=S)
A_forS = np.matmul(np.matmul(Vt,D_sq),np.transpose(Vt))

H_c[0:2,0:2] = A_forS
H_c_inv=np.linalg.inv(H_c)

H_total=np.matmul(H_c_inv,H2)
H_total_inv=np.linalg.inv(H_total)

#mask the original image to projective + affine corrected image. Here we
    concat the two homographies that we found
# and apply directly on the original image. This method gave a better result
    as compared to applying homography on
# projection corrected image.
img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
print(img_coords.T)


new_img_coords=np.zeros([4,2])

for i in range(4):

  new_coords=np.matmul(H_total,img_coords[i].T)
  new_coords=new_coords/new_coords[2]
  new_coords=np.rint(new_coords).astype(int)
```

```
  new_img_coords[i,:]=new_coords[0:2]


scale1=width/(max(new_img_coords[:,0])-min(new_img_coords[:,0]))
scale2=height/(max(new_img_coords[:,1])-min(new_img_coords[:,1]))
scale=max(scale1, scale2)
print(scale)

tx=np.round(np.min(new_img_coords[:,0]))
ty=np.round(np.min(new_img_coords[:,1]))
print(tx,ty)

new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale2
    ).astype(int)
new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale1)
    .astype(int)
new_image=np.zeros((new_height,new_width,3))
print(new_image.shape)
max_X=im1.shape[1]
max_Y=im1.shape[0]

blank_image=np.zeros((new_height,new_width,3))
for i in range(0,new_image.shape[1]-1):  #x
  for j in range(0,new_image.shape[0]-1): #y

    k1=i/scale1+tx
    k2=j/scale2+ty
    init=np.array((k1,k2,1))
    mapped=np.matmul(H_total_inv,init)
    mapped=np.rint(mapped/mapped[2])
    mapped=mapped.astype(int)
    #if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
    if(mapped[0] > 0 and mapped[1] > 0 and mapped[0] < im1.shape[1] and mapped
    [1] < image.shape[0]):
      blank_image[j,i]=im1[mapped[1],mapped[0]]

plt.figure(figsize=(20,20))
plt.imshow(blank_image.astype(int))
ii=blank_image.astype(np.uint8)
plt.imsave("affine1.jpeg",ii)

"""**Single Step method**"""

#get the 4 points to find 5 orthogonal lines. Assumption is made that the bbox
    in the original plane corresponds to a square.
P=[117,586]   #x1,y1
Q=[394,621]  #x2,y2
R=[408,388] #x3,y3
S=[146,334]  #x4,y4

#get the homogeneous coordinates
P_homo=np.array(P)
P_homo=np.append(P_homo,1)
Q_homo=np.array(Q)
Q_homo=np.append(Q_homo,1)
R_homo=np.array(R)
R_homo=np.append(R_homo,1)
S_homo=np.array(S)
```

```
S_homo=np.append(S_homo,1)

#find the 5 perpendicular line pairs
l1=np.cross(P_homo,Q_homo)  #pl1
l1=l1/l1[2]
m1=np.cross(Q_homo,R_homo)  #pl1
m1=m1/m1[2]

l2=np.cross(Q_homo,R_homo)  #pl2
l2=l2/l2[2]
m2=np.cross(R_homo,S_homo)  #pl2
m2=m2/m2[2]

l3=np.cross(R_homo,S_homo)  #pl3
l3=l3/l3[2]
m3=np.cross(S_homo,P_homo)  #pl3
m3=m3/m3[2]

l4=np.cross(S_homo,P_homo)  #pl4
l4=l4/l4[2]
m4=np.cross(P_homo,Q_homo)  #pl4
m4=m4/m4[2]

l5=np.cross(P_homo,R_homo)  #pl5
l5=l5/l5[2]
m5=np.cross(Q_homo,S_homo)  #pl5
m5=m5/m5[2]

#get a 5x6 matrix corresponding to the equation
A=np.zeros((5,6))
A[0]=[l1[0]*m1[0],(l1[1]*m1[0]+l1[0]*m1[1])/2,l1[1]*m1[1],(l1[0]*m1[2]+l1[2]*
    m1[0])/2,(l1[1]*m1[2]+l1[2]*m1[1])/2,l1[2]*m1[2]]
A[1]=[l2[0]*m2[0],(l2[1]*m2[0]+l2[0]*m2[1])/2,l2[1]*m2[1],(l2[0]*m2[2]+l2[2]*
    m2[0])/2,(l2[1]*m2[2]+l2[2]*m2[1])/2,l2[2]*m2[2]]
A[2]=[l3[0]*m3[0],(l3[1]*m3[0]+l3[0]*m3[1])/2,l3[1]*m3[1],(l3[0]*m3[2]+l3[2]*
    m3[0])/2,(l3[1]*m3[2]+l3[2]*m3[1])/2,l3[2]*m3[2]]
A[3]=[l4[0]*m4[0],(l4[1]*m4[0]+l4[0]*m4[1])/2,l4[1]*m4[1],(l4[0]*m4[2]+l4[2]*
    m4[0])/2,(l4[1]*m4[2]+l4[2]*m4[1])/2,l4[2]*m4[2]]
A[4]=[l5[0]*m5[0],(l5[1]*m5[0]+l5[0]*m5[1])/2,l5[1]*m5[1],(l5[0]*m5[2]+l5[2]*
    m5[0])/2,(l5[1]*m5[2]+l5[2]*m5[1])/2,l5[2]*m5[2]]
A

#need to find the null vector for the A matrix. Will make use of sympy library
from sympy import Matrix
a=Matrix(A)
c=a.nullspace()

c=np.array(c).astype(np.int)
c=c/max(c[0])
c

#get the H matrix from C
S_mat = np.zeros((2,2))
S_mat[0,0] = c[0][0]
S_mat[1,0] = 0.5 * c[0,1]
S_mat[0,1] = 0.5 * c[0,1]
S_mat[1,1] = c[0,2]
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)
      .astype(int)
561 new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale).
      astype(int)
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
```

```
blank_image=np.zeros((new_height,new_width,3))
for i in range(0,new_image.shape[1]-1):  #x
  for j in range(0,new_image.shape[0]-1): #y

    k1=i/scale+tx
    k2=j/scale+ty
    init=np.array((k1,k2,1))
    mapped=np.matmul(H,init)
    mapped=np.rint(mapped/mapped[2])
    mapped=mapped.astype(int)
    #if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
    if(mapped[0] > 0 and mapped[1] > 0 and mapped[0] < im1.shape[1] and mapped
    [1] < image.shape[0]):
       blank_image[j,i]=im1[mapped[1],mapped[0]]

plt.figure(figsize=(20,20))
plt.imshow(blank_image.astype(np.int))
ii=blank_image.astype(np.uint8)
plt.imsave("onestep1.jpeg",ii)




"""## **Image 2**

**Point to Point Method**
"""

#These points are considering only the given dimensions in real world. These
    did not yield good results.
'''
def get_4_points(n):
  if n==1:  #image 1a
    #for the image
    P=[480,874]   #x1,y1
    Q=[606,924]  #x2,y2
    R=[477,719] #x3,y3
    S=[601,737]  #x4,y4
    X_prime=P+Q+R+S
    return np.array([P,Q,R,S]),X_prime

def get_cat_points():
  #for the kittens
  P_dash=[0,74]
  Q_dash=[84,74]
  R_dash=[0,0]
  S_dash=[84,0]
  X=P_dash+Q_dash+R_dash+S_dash
  return np.array([P_dash,Q_dash,R_dash,S_dash]),X


def get_4_points(n):
  if n==1:  #image 1a
    #for the image
    P=[480,874]   #x1,y1
    Q=[606,924]  #x2,y2
    R= [601,737]#x3,y3
```

```
    S= [477,719] #x4,y4
    X_prime=P+Q+R+S
    return np.array([P,Q,R,S]),X_prime

def get_cat_points():
  #for the kittens
  P_dash=[0,74]
  Q_dash=[84,74]
  R_dash=[84,0]
  S_dash=[0,0]
  X=P_dash+Q_dash+R_dash+S_dash
  return np.array([P_dash,Q_dash,R_dash,S_dash]),X



'''

#these are points related to the entire frame of the window. This gives a relatively better result.
def get_4_points(n):
  if n==1: #image 1a
    #for the image
    P=[379,838]   #x1,y1
    Q=[606,924]  #x2,y2
    R=[598,545] #x3,y3
    S=[378,570]  #x4,y4
    X_prime=P+Q+R+S
    return np.array([P,Q,R,S]),X_prime

def get_cat_points():
  #for the kittens
  P_dash=[0,148]
  Q_dash=[168,148]
  R_dash=[168,0]
  S_dash=[0,0]
  X=P_dash+Q_dash+R_dash+S_dash
  return np.array([P_dash,Q_dash,R_dash,S_dash]),X

#load and display the image
image=cv2.imread("/content/drive/My Drive/hw3_Task1_Images/Images/Img2.jpeg")
image_orig = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
im1 = np.copy(image_orig)
plt.imshow(im1)
print(im1.shape)

#get the points in the image and world frames
P1,X=get_4_points(1)
P2,X_prime=get_cat_points()
draw_bbox(im1,P1)
#compute the Homography between the points
H=Homography_matrix(X,X_prime)
print(H)
H_inv=np.linalg.inv(H)

#get the image coordinates in world plane. We map the 4 corner points of the image to the world coordinates
#to get the size of the image in world coordinates.
width=im1.shape[1]
```

```
height=im1.shape[0]
print("width(x)=",width)
print("height(y)=",height)

img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
print(img_coords.T)

new_coords=np.matmul(H,img_coords.T)
new_coords=new_coords/new_coords[2]
new_coords=np.rint(new_coords).astype(int)
new_img_coords=new_coords.T[:,:2]
new_img_coords

'''
We can see from the projected corner points that the origin in this new coordinate is not at (0,0). Hence we need
to find the appropriate translation to bring the coordinates along origin.
'''
scale1=width/(max(new_img_coords[:,0])-min(new_img_coords[:,0]))
scale2=height/(max(new_img_coords[:,1])-min(new_img_coords[:,1]))
scale=max(scale1, scale2)
print(scale)

tx=np.round(np.min(new_img_coords[:,0]))
ty=np.round(np.min(new_img_coords[:,1]))
print(tx,ty)

'''
Find out the shape of the new image in world coordinates.
'''

new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale2).astype(int)
new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale1).astype(int)
new_image=np.zeros((new_height,new_width,3))

max_X=im1.shape[1]
max_Y=im1.shape[0]

'''
To find the mapping of every point in the new image to the distorted image, we multiply every point with the inverse
of the Homography - i.e. map every point in real world coordinates to the image plane and find the correspondence.
'''
for i in range(0,new_width-1):  #x
  for j in range(0,new_height-1): #y

    k1=i/scale1+tx
    k2=j/scale2+ty
    init=np.array((k1,k2,1))
    mapped=np.matmul(H_inv,init)
    mapped=np.rint(mapped/mapped[2])
    mapped=mapped.astype(int)
    if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
      new_image[j,i]=im1[mapped[1],mapped[0]]

```

```
plt.figure(figsize=(20,20))
plt.imshow(new_image.astype(int))
ii=new_image.astype(np.uint8)
plt.imsave("points2.jpeg", ii)

"""**Two Step Method**

*Projection Removal*
"""

P1=[618,510]
P2=[369,548]
P3=[361,853]
P4=[644,973]

Q1=P1 #l3
Q2=P3 #l3
Q3=P4 #l4
Q4=P2 #l4
im_copy=im1.copy()
##https://www.geeksforgeeks.org/python-opencv-cv2-line-method/ for drawing lines on image
line_img=cv2.line(im_copy, tuple(P1), tuple(P2), (255,0,0), 5)
line_img=cv2.line(line_img, tuple(P3), tuple(P4), (255,0,0), 5)
line_img=cv2.line(line_img, tuple(P1), tuple(P4), (0,255,0), 5)
line_img=cv2.line(line_img, tuple(P3), tuple(P2), (0,255,0), 5)
plt.imshow(line_img)
plt.imsave("parallel2.jpeg",line_img)

P_1=np.array(P1)
P_1=np.append(P_1,1)
P_2=np.array(P2)
P_2=np.append(P_2,1)
l1=np.cross(P_1,P_2)

P_3=np.array(P3)
P_3=np.append(P_3,1)
P_4=np.array(P4)
P_4=np.append(P_4,1)
l2=np.cross(P_3,P_4)

PV1=np.cross(l1,l2)
PV1=PV1/PV1[2]
print(PV1)

Q_1=np.array(P1)
Q_1=np.append(Q_1,1)
Q_2=np.array(P4)
Q_2=np.append(Q_2,1)
l3=np.cross(Q_1,Q_2)

Q_3=np.array(P3)
Q_3=np.append(Q_3,1)
Q_4=np.array(P2)
Q_4=np.append(Q_4,1)
l4=np.cross(Q_3,Q_4)
'''
Q_1=np.array(P1)
```

```
Q_1=np.append(Q_1,1)
Q_2=np.array(P3)
Q_2=np.append(Q_2,1)
l3=np.cross(Q_1,Q_2)

Q_3=np.array(P2)
Q_3=np.append(Q_3,1)
Q_4=np.array(P4)
Q_4=np.append(Q_4,1)
l4=np.cross(Q_3,Q_4)
'''
PV2=np.cross(l3,l4)
PV2=PV2/PV2[2]
print(PV2)

lV=np.cross(PV1,PV2)
lV=lV/lV[2]

H2=np.array([[1,0,0],[0,1,0],lV])
H2
print(H2)
H2_inv=np.linalg.inv(H2)

img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
print(img_coords.T)

new_img_coords=np.zeros([4,2])
for i in range(4):

  new_coords=np.matmul(H2,img_coords[i].T)
  new_coords=new_coords/new_coords[2]
  new_coords=np.rint(new_coords).astype(int)
  new_img_coords[i,:]=new_coords[0:2]

scale1=width/(max(new_img_coords[:,0])-min(new_img_coords[:,0]))
scale2=height/(max(new_img_coords[:,1])-min(new_img_coords[:,1]))
scale=max(scale1, scale2)
print(scale)

tx=np.round(np.min(new_img_coords[:,0]))
ty=np.round(np.min(new_img_coords[:,1]))
print(tx,ty)

new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale)
    .astype(int)
new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale).
    astype(int)
new_image=np.zeros((new_height,new_width,3))
print(new_image.shape)

max_X=im1.shape[1]
max_Y=im1.shape[0]

for i in range(0,new_width-1):  #x
  for j in range(0,new_height-1): #y

    k1=i/scale+tx
    k2=j/scale+ty
```

```
    init=np.array((k1,k2,1))
    mapped=np.matmul(H2_inv,init)
    mapped=np.rint(mapped/mapped[2])
    mapped=mapped.astype(int)
    if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
      new_image[j,i]=im1[mapped[1],mapped[0]]

plt.imshow(new_image.astype(int))
ii=new_image.astype(np.uint8)
plt.imsave("proj2.jpeg",ii)

"""*Affine Removal*"""

#4 points for determining orthogonal lines (assuming the window is square)

P1=[481,873]
P2=[606,922]
P3=[598,735]
P4=[479,718]
im_copy=im1.copy()
line_img=cv2.line(im_copy, tuple(P1), tuple(P3), (255,0,0), 5)
line_img=cv2.line(line_img, tuple(P2), tuple(P4), (255,0,0), 5)
line_img=cv2.line(line_img, tuple(P1), tuple(P2), (0,255,0), 5)
line_img=cv2.line(line_img, tuple(P3), tuple(P2), (0,255,0), 5)
plt.imshow(line_img)
plt.imsave("perpend2.jpeg",line_img)

#map the 4 points onto the projective corrected image plane
P_homo=np.array(P1)
P_homo=np.append(P_homo,1)
Q_homo=np.array(P2)
Q_homo=np.append(Q_homo,1)
R_homo=np.array(P3)
R_homo=np.append(R_homo,1)
S_homo=np.array(P4)
S_homo=np.append(S_homo,1)

P_dash_homo=np.matmul(H2,P_homo)
P_dash=P_dash_homo/P_dash_homo[-1]
Q_dash_homo=np.matmul(H2,Q_homo)
Q_dash=Q_dash_homo/Q_dash_homo[-1]
R_dash_homo=np.matmul(H2,R_homo)
R_dash=R_dash_homo/R_dash_homo[-1]
S_dash_homo=np.matmul(H2,S_homo)
S_dash=S_dash_homo/S_dash_homo[-1]

##checking
A = np.zeros((2,2))
S_mat = np.ones((2,2))
H_c = np.identity(3)

l1 = np.cross(P_dash,Q_dash) #l'
m1 = np.cross(Q_dash,R_dash) #m'

l2 = np.cross(R_dash,P_dash)
m2 = np.cross(S_dash,Q_dash)

A[0,0] = l1[0]*m1[0]
```

```
A[0,1] = l1[0]*m1[1] + l1[1]*m1[0]
A[1,:] = [l2[0]*m2[0], l2[0]*m2[1] + l2[1]*m2[0]]

B = np.array([-l1[1]*m1[1], -l2[1]*m2[1]])
S_elem = np.matmul(np.linalg.pinv(A),B)
S_mat[0,:] = [S_elem[0], S_elem[1]]
S_mat[1,0] = S_elem[1]
U, D, Vt = np.linalg.svd(S_mat, full_matrices=True)
D_sq = np.zeros((2,2))
D_sq[0,0] = np.sqrt(D[0])
D_sq[1,1] = np.sqrt(D[1])

A_forS = np.matmul(np.matmul(Vt,D_sq),np.transpose(Vt))

H_c[0:2,0:2] = A_forS
H_c_inv= np.linalg.inv(H_c)

H_total=np.matmul(H_c_inv,H2)
H_total_inv=np.linalg.inv(H_total)

#mask the original image to projective + affine corrected image
img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
print(img_coords.T)


new_img_coords=np.zeros([4,2])

for i in range(4):

  new_coords=np.matmul(H_total,img_coords[i].T)
  new_coords=new_coords/new_coords[2]
  new_coords=np.rint(new_coords).astype(int)
  new_img_coords[i,:]=new_coords[0:2]


scale1=width/(max(new_img_coords[:,0])-min(new_img_coords[:,0]))
scale2=height/(max(new_img_coords[:,1])-min(new_img_coords[:,1]))
scale=max(scale1, scale2)
print(scale)

tx=np.round(np.min(new_img_coords[:,0]))
ty=np.round(np.min(new_img_coords[:,1]))
print(tx,ty)

new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale2
    ).astype(int)
new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale1)
    .astype(int)
new_image=np.zeros((new_height,new_width,3))
print(new_image.shape)
max_X=im1.shape[1]
max_Y=im1.shape[0]

blank_image=np.zeros((new_height,new_width,3))
for i in range(0,new_image.shape[1]-1):  #x
  for j in range(0,new_image.shape[0]-1): #y

    k1=i/scale1+tx
```

```
    k2=j/scale2+ty
    init=np.array((k1,k2,1))
    mapped=np.matmul(H_total_inv,init)
    mapped=np.rint(mapped/mapped[2])
    mapped=mapped.astype(int)
    #if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
    if(mapped[0] > 0 and mapped[1] > 0 and mapped[0] < im1.shape[1] and mapped
    [1] < image.shape[0]):
        blank_image[j,i]=im1[mapped[1],mapped[0]]

plt.figure(figsize=(20,20))
plt.imshow(blank_image.astype(int))
ii=blank_image.astype(np.uint8)
plt.imsave("affine2.jpeg",ii)




"""**One Step Method**"""

#4 points for determining orthogonal lines

P=[369,548]
Q=[361,853]
R=[644,973]
S=[618,510]

P_homo=np.array(P)
P_homo=np.append(P_homo,1)
Q_homo=np.array(Q)
Q_homo=np.append(Q_homo,1)
R_homo=np.array(R)
R_homo=np.append(R_homo,1)
S_homo=np.array(S)
S_homo=np.append(S_homo,1)

l1=np.cross(P_homo,Q_homo)  #pl1
l1=l1/l1[2]
m1=np.cross(Q_homo,R_homo)  #pl1
m1=m1/m1[2]

l2=np.cross(Q_homo,R_homo)  #pl2
l2=l2/l2[2]
m2=np.cross(R_homo,S_homo)  #pl2
m2=m2/m2[2]

l3=np.cross(R_homo,S_homo)  #pl3
l3=l3/l3[2]
m3=np.cross(S_homo,P_homo)  #pl3
m3=m3/m3[2]

l4=np.cross(S_homo,P_homo)  #pl4
l4=l4/l4[2]
m4=np.cross(P_homo,Q_homo)  #pl4
m4=m4/m4[2]

l5=np.cross(P_homo,R_homo)  #pl5
```

```
l5=l5/l5[2]
m5=np.cross(Q_homo,S_homo)   #pl5
m5=m5/m5[2]

A=np.zeros((5,6))
A[0]=[l1[0]*m1[0],(l1[1]*m1[0]+l1[0]*m1[1])/2,l1[1]*m1[1],(l1[0]*m1[2]+l1[2]*
    m1[0])/2,(l1[1]*m1[2]+l1[2]*m1[1])/2,l1[2]*m1[2]]
A[1]=[l2[0]*m2[0],(l2[1]*m2[0]+l2[0]*m2[1])/2,l2[1]*m2[1],(l2[0]*m2[2]+l2[2]*
    m2[0])/2,(l2[1]*m2[2]+l2[2]*m2[1])/2,l2[2]*m2[2]]
A[2]=[l3[0]*m3[0],(l3[1]*m3[0]+l3[0]*m3[1])/2,l3[1]*m3[1],(l3[0]*m3[2]+l3[2]*
    m3[0])/2,(l3[1]*m3[2]+l3[2]*m3[1])/2,l3[2]*m3[2]]
A[3]=[l4[0]*m4[0],(l4[1]*m4[0]+l4[0]*m4[1])/2,l4[1]*m4[1],(l4[0]*m4[2]+l4[2]*
    m4[0])/2,(l4[1]*m4[2]+l4[2]*m4[1])/2,l4[2]*m4[2]]
A[4]=[l5[0]*m5[0],(l5[1]*m5[0]+l5[0]*m5[1])/2,l5[1]*m5[1],(l5[0]*m5[2]+l5[2]*
    m5[0])/2,(l5[1]*m5[2]+l5[2]*m5[1])/2,l5[2]*m5[2]]
A

from sympy import Matrix
a=Matrix(A)
c=a.nullspace()

c=np.array(c).astype(np.int)
c=c/max(c[0])
c

S_mat = np.zeros((2,2))
S_mat[0,0] = c[0][0]
S_mat[1,0] = 0.5 * c[0,1]
S_mat[0,1] = 0.5 * c[0,1]
S_mat[1,1] = c[0,2]
S_mat

U, D, Vt = np.linalg.svd(S_mat)

K = np.zeros((2,2))
D_sq = np.diag(np.sqrt(D))
K = np.dot(Vt, np.dot(D_sq, Vt))

A=np.matmul(Vt,np.matmul(D_sq,Vt.T))
V = np.matmul(np.linalg.inv(A),[0.5*c[0,3], 0.5*c[0,4]]);
H=[[A[0,0], A[0,1], 0], [A[1,0], A[1,1], 0], [V[0], V[1],1]];
print(H)
H_inv=np.linalg.inv(H)
H_inv

img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
print(img_coords.T)

new_img_coords=np.zeros([4,2])
for i in range(4):

   new_coords=np.matmul(H_inv,img_coords[i].T)
   new_coords=new_coords/new_coords[2]
   new_coords=np.rint(new_coords).astype(int)
   new_img_coords[i,:]=new_coords[0:2]

print(new_img_coords)

```

```
scale1=width/(max(new_img_coords[:,0])-min(new_img_coords[:,0]))
scale2=height/(max(new_img_coords[:,1])-min(new_img_coords[:,1]))
scale=max(scale1, scale2)
print(scale)

tx=np.round(np.min(new_img_coords[:,0]))
ty=np.round(np.min(new_img_coords[:,1]))
print(tx,ty)

new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale)
    .astype(int)
new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale).
    astype(int)
new_image=np.zeros((new_height,new_width,3))
print(new_image.shape)
max_X=im1.shape[1]
max_Y=im1.shape[0]

blank_image=np.zeros((new_height,new_width,3))
for i in range(0,new_image.shape[1]-1):  #x
  for j in range(0,new_image.shape[0]-1): #y

    k1=i/scale+tx
    k2=j/scale+ty
    init=np.array((k1,k2,1))
    mapped=np.matmul(H,init)
    mapped=np.rint(mapped/mapped[2])
    mapped=mapped.astype(int)
    #if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
    if(mapped[0] > 0 and mapped[1] > 0 and mapped[0] < im1.shape[1] and mapped
    [1] < image.shape[0]):
      blank_image[j,i]=im1[mapped[1],mapped[0]]

plt.figure(figsize=(20,20))
plt.imshow(blank_image.astype(np.int))
ii=blank_image.astype(np.uint8)
plt.imsave("onestep2.jpeg",ii)




"""## **Image 3**

**Point to Point Method**
"""

#get the four points from the distorted and world images
def get_4_points(n):
  if n==1:  #image 1a
    #for the image
    P=[2092,1481]    #x1,y1
    Q=[2693,1322] #x2,y2
    R=[2667,717] #x3,y3
    S=[2057,700]  #x4,y4
    X_prime=P+Q+R+S
    return np.array([P,Q,R,S]),X_prime

```

```
def get_cat_points():
  #for the kittens
  P_dash=[0,36]
  Q_dash=[55,36]
  R_dash=[55,0]
  S_dash=[0,0]
  X=P_dash+Q_dash+R_dash+S_dash
  return np.array([P_dash,Q_dash,R_dash,S_dash]),X

image=cv2.imread("/content/drive/My Drive/hw3_Task1_Images/Images/Img3.JPG")
image_orig = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
im1 = np.copy(image_orig)
plt.imshow(im1)
print(im1.shape)

#get the 4 points and find the Homography between them
P1,X=get_4_points(1)
P2,X_prime=get_cat_points()
draw_bbox(im1,P1)
H=Homography_matrix(X,X_prime)
print(H)
H_inv=np.linalg.inv(H)

#get the image coordinates in world plane
width=im1.shape[1]
height=im1.shape[0]
print("width(x)=",width)
print("height(y)=",height)

img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
print(img_coords.T)
new_img_coords=np.zeros([4,2])
for i in range(4):

  new_coords=np.matmul(H,img_coords[i].T)
  new_coords=new_coords/new_coords[2]
  new_coords=np.rint(new_coords).astype(int)
  new_img_coords[i,:]=new_coords[0:2]

#the new image will have translation(coordinates negative) and scale errors.
   Need to correct them
scale1=width/(max(new_img_coords[:,0])-min(new_img_coords[:,0]))
scale2=height/(max(new_img_coords[:,1])-min(new_img_coords[:,1]))
scale=max(scale1, scale2)
print(scale)
tx=np.round(np.min(new_img_coords[:,0]))
ty=np.round(np.min(new_img_coords[:,1]))
print(tx,ty)

#get the dimensions of the new image
new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale2
    ).astype(int)
new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale1)
    .astype(int)
new_image=np.zeros((new_height,new_width,3))
new_width,new_height

max_X=im1.shape[1]
```

```
max_Y=im1.shape[0]

#get the indices of each pixel in the new image and find the correspondence between the distorted and world pixels.
for i in range(0,new_width):  #x
  for j in range(0,new_height): #y

    k1=i/scale1+tx
    k2=j/scale2+ty
    init=np.array((k1,k2,1))
    mapped=np.matmul(H_inv,init)
    mapped=np.rint(mapped/mapped[2])
    mapped=mapped.astype(int)
    if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
      new_image[j,i]=im1[mapped[1],mapped[0]]

plt.figure(figsize=(20,20))
plt.imshow(new_image.astype(int))
ii=new_image.astype(np.uint8)
plt.imsave("points3.jpeg",ii)

"""**Two Step Method**

*Projection Removal*
"""

#get the 4 points that correspond to two parallel lines
P1=[2092,1481]   #x1,y1
P2=[2693,1322] #x2,y2
P3=[2667,717] #x3,y3
P4=[2057,700]  #x4,y4

Q1=P1
Q2=P4
Q3=P2
Q4=P3
im_copy=im1.copy()
##https://www.geeksforgeeks.org/python-opencv-cv2-line-method/ for drawing lines on image
line_img=cv2.line(im_copy, tuple(P1), tuple(P2), (255,0,0), 5)
line_img=cv2.line(line_img, tuple(P3), tuple(P4), (255,0,0), 5)
line_img=cv2.line(line_img, tuple(Q1), tuple(Q2), (0,255,0), 5)
line_img=cv2.line(line_img, tuple(Q3), tuple(Q4), (0,255,0), 5)
plt.imshow(line_img)
plt.imsave("parallellines3.jpeg",line_img)

#find the first vanishing point
P_1=np.array(P1)
P_1=np.append(P_1,1)
P_2=np.array(P2)
P_2=np.append(P_2,1)
l1=np.cross(P_1,P_2)

P_3=np.array(P3)
P_3=np.append(P_3,1)
P_4=np.array(P4)
P_4=np.append(P_4,1)
l2=np.cross(P_3,P_4)
```

```
1235
1236 PV1=np.cross(l1,l2)
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)
      .astype(int)
1290 new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale).
      astype(int)
```

```
new_image=np.zeros((new_height,new_width,3))
print(new_image.shape)

max_X=im1.shape[1]
max_Y=im1.shape[0]

for i in range(0,new_width-1):  #x
  for j in range(0,new_height-1): #y

    k1=i/scale+tx
    k2=j/scale+ty
    init=np.array((k1,k2,1))
    mapped=np.matmul(H2_inv,init)
    mapped=np.rint(mapped/mapped[2])
    mapped=mapped.astype(int)
    if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
      new_image[j,i]=im1[mapped[1],mapped[0]]

plt.imshow(new_image.astype(int))

ii=new_image.astype(np.uint8)
plt.imsave("proj3.jpeg",ii)

"""*Affine Removal*"""

#display the two pairs of perpendicular lines
line_img=cv2.line(im_copy, tuple(P1), tuple(P2), (255,0,0), 5)
line_img=cv2.line(line_img, tuple(P2), tuple(P3), (255,0,0), 5)
line_img=cv2.line(line_img, tuple(P3), tuple(P4), (0,255,0), 5)
line_img=cv2.line(line_img, tuple(P4), tuple(P1), (0,255,0), 5)
plt.imshow(line_img)
plt.imsave("perpendicular3.jpeg",line_img)

#project these points onto the projection corrected image plane
P_homo=np.array(P1)
P_homo=np.append(P_homo,1)
Q_homo=np.array(P2)
Q_homo=np.append(Q_homo,1)
R_homo=np.array(P3)
R_homo=np.append(R_homo,1)
S_homo=np.array(P4)
S_homo=np.append(S_homo,1)

P_dash_homo=np.matmul(H2,P_homo)
P_dash=P_dash_homo/P_dash_homo[-1]
Q_dash_homo=np.matmul(H2,Q_homo)
Q_dash=Q_dash_homo/Q_dash_homo[-1]
R_dash_homo=np.matmul(H2,R_homo)
R_dash=R_dash_homo/R_dash_homo[-1]
S_dash_homo=np.matmul(H2,S_homo)
S_dash=S_dash_homo/S_dash_homo[-1]

#initialize the matrices
A = np.zeros((2,2))
S_mat = np.ones((2,2))
H_c = np.identity(3)

#get the first set of perpendicular lines
```

```
l1 = np.cross(P_dash,Q_dash) #l'
m1 = np.cross(Q_dash,R_dash) #m'

#get the second set of perpendicular lines
l2 = np.cross(R_dash,P_dash)
m2 = np.cross(S_dash,Q_dash)

#get the A and  matrix
A[0,0] = l1[0]*m1[0]
A[0,1] = l1[0]*m1[1] + l1[1]*m1[0]
A[1,:] = [l2[0]*m2[0], l2[0]*m2[1] + l2[1]*m2[0]]

B = np.array([-l1[1]*m1[1], -l2[1]*m2[1]])

#Find out S
S_elem = np.matmul(np.linalg.pinv(A),B)
S_mat[0,:] = [S_elem[0], S_elem[1]]
S_mat[1,0] = S_elem[1]
U, D, Vt = np.linalg.svd(S_mat, full_matrices=True)
D_sq = np.zeros((2,2))
D_sq[0,0] = np.sqrt(D[0])
D_sq[1,1] = np.sqrt(D[1])

#Get the value of A (AA'=S)
A_forS = np.matmul(np.matmul(Vt,D_sq),np.transpose(Vt))

H_c[0:2,0:2] = A_forS
H_c_inv=np.linalg.inv(H_c)

H_total=np.matmul(H_c_inv,H2)
H_total_inv=np.linalg.inv(H_total)

#mask the original image to projective + affine corrected image
img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
print(img_coords.T)


new_img_coords=np.zeros([4,2])

for i in range(4):

  new_coords=np.matmul(H_total,img_coords[i].T)
  new_coords=new_coords/new_coords[2]
  new_coords=np.rint(new_coords).astype(int)
  new_img_coords[i,:]=new_coords[0:2]


scale1=width/(max(new_img_coords[:,0])-min(new_img_coords[:,0]))
scale2=height/(max(new_img_coords[:,1])-min(new_img_coords[:,1]))
scale=max(scale1, scale2)
print(scale)

tx=np.round(np.min(new_img_coords[:,0]))
ty=np.round(np.min(new_img_coords[:,1]))
print(tx,ty)

new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale2
    ).astype(int)
```

```
new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale1)
    .astype(int)
new_image=np.zeros((new_height,new_width,3))
print(new_image.shape)
max_X=im1.shape[1]
max_Y=im1.shape[0]

blank_image=np.zeros((new_height,new_width,3))
for i in range(0,new_image.shape[1]-1):  #x
  for j in range(0,new_image.shape[0]-1): #y

    k1=i/scale1+tx
    k2=j/scale2+ty
    init=np.array((k1,k2,1))
    mapped=np.matmul(H_total_inv,init)
    mapped=np.rint(mapped/mapped[2])
    mapped=mapped.astype(int)
    #if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
    if(mapped[0] > 0 and mapped[1] > 0 and mapped[0] < im1.shape[1] and mapped
    [1] < image.shape[0]):
       blank_image[j,i]=im1[mapped[1],mapped[0]]

plt.figure(figsize=(10,10))
plt.imshow(blank_image.astype(int))
ii=blank_image.astype(np.uint8)
plt.imsave("affine3.jpeg",ii)

"""**One Step Method**"""

#4 points for determining orthogonal lines

P=[2092,1481]    #x1,y1
Q=[2693,1322] #x2,y2
R=[2667,717] #x3,y3
S=[2057,700]   #x4,y4

P_homo=np.array(P)
P_homo=np.append(P_homo,1)
Q_homo=np.array(Q)
Q_homo=np.append(Q_homo,1)
R_homo=np.array(R)
R_homo=np.append(R_homo,1)
S_homo=np.array(S)
S_homo=np.append(S_homo,1)

l1=np.cross(P_homo,Q_homo)   #pl1
l1=l1/l1[2]
m1=np.cross(Q_homo,R_homo)   #pl1
m1=m1/m1[2]

l2=np.cross(Q_homo,R_homo)   #pl2
l2=l2/l2[2]
m2=np.cross(R_homo,S_homo)   #pl2
m2=m2/m2[2]

l3=np.cross(R_homo,S_homo)   #pl3
l3=l3/l3[2]
m3=np.cross(S_homo,P_homo)   #pl3
```

```
m3=m3/m3[2]

l4=np.cross(S_homo,P_homo)   #pl4
l4=l4/l4[2]
m4=np.cross(P_homo,Q_homo)   #pl4
m4=m4/m4[2]

l5=np.cross(P_homo,R_homo)   #pl5
l5=l5/l5[2]
m5=np.cross(Q_homo,S_homo)   #pl5
m5=m5/m5[2]

A=np.zeros((5,6))
A[0]=[l1[0]*m1[0],(l1[1]*m1[0]+l1[0]*m1[1])/2,l1[1]*m1[1],(l1[0]*m1[2]+l1[2]*
    m1[0])/2,(l1[1]*m1[2]+l1[2]*m1[1])/2,l1[2]*m1[2]]
A[1]=[l2[0]*m2[0],(l2[1]*m2[0]+l2[0]*m2[1])/2,l2[1]*m2[1],(l2[0]*m2[2]+l2[2]*
    m2[0])/2,(l2[1]*m2[2]+l2[2]*m2[1])/2,l2[2]*m2[2]]
A[2]=[l3[0]*m3[0],(l3[1]*m3[0]+l3[0]*m3[1])/2,l3[1]*m3[1],(l3[0]*m3[2]+l3[2]*
    m3[0])/2,(l3[1]*m3[2]+l3[2]*m3[1])/2,l3[2]*m3[2]]
A[3]=[l4[0]*m4[0],(l4[1]*m4[0]+l4[0]*m4[1])/2,l4[1]*m4[1],(l4[0]*m4[2]+l4[2]*
    m4[0])/2,(l4[1]*m4[2]+l4[2]*m4[1])/2,l4[2]*m4[2]]
A[4]=[l5[0]*m5[0],(l5[1]*m5[0]+l5[0]*m5[1])/2,l5[1]*m5[1],(l5[0]*m5[2]+l5[2]*
    m5[0])/2,(l5[1]*m5[2]+l5[2]*m5[1])/2,l5[2]*m5[2]]
A

from sympy import Matrix
a=Matrix(A)
c=a.nullspace()
c=np.array(c).astype(np.int)
c=c/max(c[0])
c

S_mat = np.zeros((2,2))
S_mat[0,0] = c[0][0]
S_mat[1,0] = 0.5 * c[0,1]
S_mat[0,1] = 0.5 * c[0,1]
S_mat[1,1] = c[0,2]
S_mat

U, D, Vt = np.linalg.svd(S_mat)
K = np.zeros((2,2))
D_sq = np.diag(np.sqrt(D))
K = np.dot(Vt, np.dot(D_sq, Vt))
v = np.zeros((2,1))
rhs_vA = np.zeros((2,1))
rhs_vA [0] = 0.5 * c[0,3]
rhs_vA [1] = 0.5 * c[0,4]

v = np.dot(np.linalg.inv(K), rhs_vA)

H = np.zeros((3,3))

#Filling up the homography
H[0]=np.array([K[0,0], K[0,1], 0])
H[1]=np.array([K[1,0], K[1,1], 0])
H[2]=np.array([v[0], v[1], 1])

H_inv=np.linalg.inv(H)
```

```
H_inv

img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
print(img_coords.T)


new_img_coords=np.zeros([4,2])
'''
for i in range(4):

  new_coords=np.matmul(H_inv,img_coords[i].T)
  new_coords=new_coords/new_coords[2]
  new_coords=np.rint(new_coords).astype(int)
  new_img_coords[i,:]=new_coords[0:2]
'''
new_coords=np.matmul(H_inv,img_coords.T)
new_coords=new_coords/new_coords[2]
new_coords=np.rint(new_coords).astype(int)
new_img_coords=new_coords.T[:,:2]
new_img_coords
print(new_img_coords)

scale1=width/(max(new_img_coords[:,0])-min(new_img_coords[:,0]))
scale2=height/(max(new_img_coords[:,1])-min(new_img_coords[:,1]))
scale=max(scale1, scale2)
print(scale)

tx=np.round(np.min(new_img_coords[:,0]))
ty=np.round(np.min(new_img_coords[:,1]))
print(tx,ty)

new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale)
    .astype(int)
new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale).
    astype(int)
new_image=np.zeros((new_height,new_width,3))
print(new_image.shape)
max_X=im1.shape[1]
max_Y=im1.shape[0]

blank_image=np.zeros((new_height,new_width,3))
for i in range(0,new_image.shape[1]-1):  #x
  for j in range(0,new_image.shape[0]-1): #y

    k1=i/scale+tx
    k2=j/scale+ty
    init=np.array((k1,k2,1))
    mapped=np.matmul(H,init)
    mapped=np.rint(mapped/mapped[2])
    mapped=mapped.astype(int)
    #if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
    if(mapped[0] > 0 and mapped[1] > 0 and mapped[0] < im1.shape[1] and mapped
    [1] < image.shape[0]):
       blank_image[j,i]=im1[mapped[1],mapped[0]]

plt.figure(figsize=(20,20))
plt.imshow(blank_image.astype(np.int))
ii=blank_image.astype(np.uint8)
```

```
plt.imsave("onestep3.jpeg",ii)




"""## **My Image 1**

**Point to Point Method**
"""

def get_4_points(n):
  if n==1:  #image 1a
    #for the image
    P=[262,660]   #x1,y1
    Q=[353,664] #x2,y2
    R= [357,558]#x3,y3
    S= [266,558] #x4,y4
    X_prime=P+Q+R+S
    return np.array([P,Q,R,S]),X_prime

def get_cat_points():
  #for the kittens
  P_dash=[0,50]
  Q_dash=[75,50]
  R_dash=[75,0]
  S_dash=[0,0]
  X=P_dash+Q_dash+R_dash+S_dash
  return np.array([P_dash,Q_dash,R_dash,S_dash]),X

image=cv2.imread("/content/drive/My Drive/hw3_Task1_Images/Images/Ellis_island
    .jpg")
image_orig = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
im1 = np.copy(image_orig)
plt.imshow(im1)
print(im1.shape)
plt.imsave("my_img1.jpeg",im1)

P1,X=get_4_points(1)
P2,X_prime=get_cat_points()

draw_bbox(im1,P1)

H=Homography_matrix(X,X_prime)
print(H)
H_inv=np.linalg.inv(H)

#get the image coordinates in world plane
width=im1.shape[1]
height=im1.shape[0]
print("width(x)=",width)
print("height(y)=",height)

img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
print(img_coords.T)

'''
new_coords=np.matmul(H,img_coords.T)
```

```
new_coords=new_coords/new_coords[2]
new_coords=np.rint(new_coords).astype(int)
new_img_coords=new_coords.T[:,:2]
new_img_coords
'''
new_img_coords=np.zeros([4,2])
for i in range(4):

  new_coords=np.matmul(H,img_coords[i].T)
  new_coords=new_coords/new_coords[2]
  new_coords=np.rint(new_coords).astype(int)
  new_img_coords[i,:]=new_coords[0:2]

scale1=width/(max(new_img_coords[:,0])-min(new_img_coords[:,0]))
scale2=height/(max(new_img_coords[:,1])-min(new_img_coords[:,1]))
scale=max(scale1, scale2)
print(scale)

tx=np.round(np.min(new_img_coords[:,0]))
ty=np.round(np.min(new_img_coords[:,1]))
print(tx,ty)

new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale)
    .astype(int)
new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale).
    astype(int)
new_image=np.zeros((new_height,new_width,3))
new_width,new_height

max_X=im1.shape[1]
max_Y=im1.shape[0]

for i in range(0,new_width):  #x
  for j in range(0,new_height): #y

    k1=i/scale+tx
    k2=j/scale+ty
    init=np.array((k1,k2,1))
    mapped=np.matmul(H_inv,init)
    mapped=np.rint(mapped/mapped[2])
    mapped=mapped.astype(int)
    if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
      new_image[j,i]=im1[mapped[1],mapped[0]]

plt.imshow(new_image.astype(int))
ii=new_image.astype(np.uint8)
plt.imsave("pointsm1.jpeg",ii)

"""**Two Step Method**

*Projection Removal*
"""

P1=[262,660]    #x1,y1
P2=[353,664] #x2,y2
P3=[357,558]#x3,y3
P4=[266,558]

```

```
'''
Q1=[410,120] #l3
Q2=[533,8] #l3
Q3=[409,374] #l4
Q4=[539,312] #l4
'''
Q1=P1
Q2=P4
Q3=P2
Q4=P3
im_copy=im1.copy()
##https://www.geeksforgeeks.org/python-opencv-cv2-line-method/ for drawing
    lines on image
line_img=cv2.line(im_copy, tuple(P1), tuple(P2), (255,0,0), 5)
line_img=cv2.line(line_img, tuple(P3), tuple(P4), (255,0,0), 5)
line_img=cv2.line(line_img, tuple(Q1), tuple(Q2), (0,255,0), 5)
line_img=cv2.line(line_img, tuple(Q3), tuple(Q4), (0,255,0), 5)
plt.imshow(line_img)
plt.imsave("parallel_linesm1.jpeg",line_img)

P_1=np.array(P1)
P_1=np.append(P_1,1)
P_2=np.array(P2)
P_2=np.append(P_2,1)
l1=np.cross(P_1,P_2)

P_3=np.array(P3)
P_3=np.append(P_3,1)
P_4=np.array(P4)
P_4=np.append(P_4,1)
l2=np.cross(P_3,P_4)

PV1=np.cross(l1,l2)
PV1=PV1/PV1[2]
print(PV1)

Q_1=np.array(Q1)
Q_1=np.append(Q_1,1)
Q_2=np.array(Q2)
Q_2=np.append(Q_2,1)
l3=np.cross(Q_1,Q_2)

Q_3=np.array(Q3)
Q_3=np.append(Q_3,1)
Q_4=np.array(Q4)
Q_4=np.append(Q_4,1)
l4=np.cross(Q_3,Q_4)
'''
Q_1=np.array(P1)
Q_1=np.append(Q_1,1)
Q_2=np.array(P3)
Q_2=np.append(Q_2,1)
l3=np.cross(Q_1,Q_2)

Q_3=np.array(P2)
Q_3=np.append(Q_3,1)
Q_4=np.array(P4)
Q_4=np.append(Q_4,1)
```

```
l4=np.cross(Q_3,Q_4)
'''
PV2=np.cross(l3,l4)
PV2=PV2/PV2[2]
print(PV2)

lV=np.cross(PV1,PV2)
lV=lV/lV[2]
lV

H2=np.array([[1,0,0],[0,1,0],lV])
H2
print(H2)
H2_inv=np.linalg.inv(H2)

img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
print(img_coords.T)

new_img_coords=np.zeros([4,2])
for i in range(4):

  new_coords=np.matmul(H2,img_coords[i].T)
  new_coords=new_coords/new_coords[2]
  new_coords=np.rint(new_coords).astype(int)
  new_img_coords[i,:]=new_coords[0:2]

scale1=width/(max(new_img_coords[:,0])-min(new_img_coords[:,0]))
scale2=height/(max(new_img_coords[:,1])-min(new_img_coords[:,1]))
scale=max(scale1, scale2)
print(scale)

tx=np.round(np.min(new_img_coords[:,0]))
ty=np.round(np.min(new_img_coords[:,1]))
print(tx,ty)

new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale)
    .astype(int)
new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale).
    astype(int)
new_image=np.zeros((new_height,new_width,3))
print(new_image.shape)

max_X=im1.shape[1]
max_Y=im1.shape[0]

for i in range(0,new_width-1):  #x
  for j in range(0,new_height-1): #y

    k1=i/scale+tx
    k2=j/scale+ty
    init=np.array((k1,k2,1))
    mapped=np.matmul(H2_inv,init)
    mapped=np.rint(mapped/mapped[2])
    mapped=mapped.astype(int)
    if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
      new_image[j,i]=im1[mapped[1],mapped[0]]

plt.imshow(new_image.astype(int))
```

```
ii=new_image.astype(np.uint8)
plt.imsave("projm1.jpeg",ii)

"""*Affine Removal*"""

im_copy=im1.copy()
P1=[262,660]    #x1,y1
P2=[353,664] #x2,y2
P3=[357,558]#x3,y3
P4=[266,558]
line_img=cv2.line(im_copy, tuple(P1), tuple(P3), (255,0,0), 5)
line_img=cv2.line(line_img, tuple(P2), tuple(P4), (255,0,0), 5)
line_img=cv2.line(line_img, tuple(P1), tuple(P2), (0,255,0), 5)
line_img=cv2.line(line_img, tuple(P3), tuple(P2), (0,255,0), 5)
plt.imshow(line_img)
plt.imsave("perpendm1.jpeg",line_img)

#map the 4 points onto the projective corrected image plane
P_homo=np.array(P1)
P_homo=np.append(P_homo,1)
Q_homo=np.array(P2)
Q_homo=np.append(Q_homo,1)
R_homo=np.array(P3)
R_homo=np.append(R_homo,1)
S_homo=np.array(P4)
S_homo=np.append(S_homo,1)

P_dash_homo=np.matmul(H2,P_homo)
P_dash=P_dash_homo/P_dash_homo[-1]
Q_dash_homo=np.matmul(H2,Q_homo)
Q_dash=Q_dash_homo/Q_dash_homo[-1]
R_dash_homo=np.matmul(H2,R_homo)
R_dash=R_dash_homo/R_dash_homo[-1]
S_dash_homo=np.matmul(H2,S_homo)
S_dash=S_dash_homo/S_dash_homo[-1]

#initialize the matrices
A = np.zeros((2,2))
S_mat = np.ones((2,2))
H_c = np.identity(3)

#get the first set of perpendicular lines
l1 = np.cross(P_dash,R_dash) #l'
m1 = np.cross(S_dash,Q_dash) #m'

#get the second set of perpendicular lines
l2 = np.cross(P_dash,Q_dash)
m2 = np.cross(Q_dash,R_dash)

#get the A and  matrix
A[0,0] = l1[0]*m1[0]
A[0,1] = l1[0]*m1[1] + l1[1]*m1[0]
A[1,:] = [l2[0]*m2[0], l2[0]*m2[1] + l2[1]*m2[0]]

B = np.array([-l1[1]*m1[1], -l2[1]*m2[1]])

#Find out S
S_elem = np.matmul(np.linalg.pinv(A),B)
```

```
S_mat[0,:] = [S_elem[0], S_elem[1]]
S_mat[1,0] = S_elem[1]
U, D, Vt = np.linalg.svd(S_mat, full_matrices=True)
D_sq = np.zeros((2,2))
D_sq[0,0] = np.sqrt(D[0])
D_sq[1,1] = np.sqrt(D[1])

#Get the value of A (AA'=S)
A_forS = np.matmul(np.matmul(Vt,D_sq),np.transpose(Vt))

H_c[0:2,0:2] = A_forS
H_c_inv=np.linalg.inv(H_c)

H_total=np.matmul(H_c_inv,H2)
H_total_inv=np.linalg.inv(H_total)

#mask the original image to projective + affine corrected image
img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
print(img_coords.T)


new_img_coords=np.zeros([4,2])

for i in range(4):

  new_coords=np.matmul(H_total,img_coords[i].T)
  new_coords=new_coords/new_coords[2]
  new_coords=np.rint(new_coords).astype(int)
  new_img_coords[i,:]=new_coords[0:2]


scale1=width/(max(new_img_coords[:,0])-min(new_img_coords[:,0]))
scale2=height/(max(new_img_coords[:,1])-min(new_img_coords[:,1]))
scale=max(scale1, scale2)
print(scale)

tx=np.round(np.min(new_img_coords[:,0]))
ty=np.round(np.min(new_img_coords[:,1]))
print(tx,ty)

new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale2
    ).astype(int)
new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale1)
    .astype(int)
new_image=np.zeros((new_height,new_width,3))
print(new_image.shape)
max_X=im1.shape[1]
max_Y=im1.shape[0]

blank_image=np.zeros((new_height,new_width,3))
for i in range(0,new_image.shape[1]-1):  #x
  for j in range(0,new_image.shape[0]-1): #y

    k1=i/scale1+tx
    k2=j/scale2+ty
    init=np.array((k1,k2,1))
    mapped=np.matmul(H_total_inv,init)
    mapped=np.rint(mapped/mapped[2])
```

```
    mapped=mapped.astype(int)
    #if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
    if(mapped[0] > 0 and mapped[1] > 0 and mapped[0] < im1.shape[1] and mapped
    [1] < image.shape[0]):
        blank_image[j,i]=im1[mapped[1],mapped[0]]

plt.figure(figsize=(20,20))
plt.imshow(blank_image.astype(int))
ii=blank_image.astype(np.uint8)
plt.imsave("affinem1.jpeg",ii)

"""**One Step Method**"""

P=[262,660]    #x1,y1
Q=[353,664] #x2,y2
R=[357,558]#x3,y3
S=[266,558]

P_homo=np.array(P)
P_homo=np.append(P_homo,1)
Q_homo=np.array(Q)
Q_homo=np.append(Q_homo,1)
R_homo=np.array(R)
R_homo=np.append(R_homo,1)
S_homo=np.array(S)
S_homo=np.append(S_homo,1)

l1=np.cross(P_homo,Q_homo)   #pl1
l1=l1/l1[2]
m1=np.cross(Q_homo,R_homo)   #pl1
m1=m1/m1[2]

l2=np.cross(Q_homo,R_homo)   #pl2
l2=l2/l2[2]
m2=np.cross(R_homo,S_homo)   #pl2
m2=m2/m2[2]

l3=np.cross(R_homo,S_homo)   #pl3
l3=l3/l3[2]
m3=np.cross(S_homo,P_homo)   #pl3
m3=m3/m3[2]

l4=np.cross(S_homo,P_homo)   #pl4
l4=l4/l4[2]
m4=np.cross(P_homo,Q_homo)   #pl4
m4=m4/m4[2]

l5=np.cross(P_homo,R_homo)   #pl5
l5=l5/l5[2]
m5=np.cross(Q_homo,S_homo)   #pl5
m5=m5/m5[2]

A=np.zeros((5,6))
A[0]=[l1[0]*m1[0],(l1[1]*m1[0]+l1[0]*m1[1])/2,l1[1]*m1[1],(l1[0]*m1[2]+l1[2]*
    m1[0])/2,(l1[1]*m1[2]+l1[2]*m1[1])/2,l1[2]*m1[2]]
A[1]=[l2[0]*m2[0],(l2[1]*m2[0]+l2[0]*m2[1])/2,l2[1]*m2[1],(l2[0]*m2[2]+l2[2]*
    m2[0])/2,(l2[1]*m2[2]+l2[2]*m2[1])/2,l2[2]*m2[2]]
A[2]=[l3[0]*m3[0],(l3[1]*m3[0]+l3[0]*m3[1])/2,l3[1]*m3[1],(l3[0]*m3[2]+l3[2]*
```

```
    m3[0])/2,(l3[1]*m3[2]+l3[2]*m3[1])/2,l3[2]*m3[2]]
A[3]=[l4[0]*m4[0],(l4[1]*m4[0]+l4[0]*m4[1])/2,l4[1]*m4[1],(l4[0]*m4[2]+l4[2]*
    m4[0])/2,(l4[1]*m4[2]+l4[2]*m4[1])/2,l4[2]*m4[2]]
A[4]=[l5[0]*m5[0],(l5[1]*m5[0]+l5[0]*m5[1])/2,l5[1]*m5[1],(l5[0]*m5[2]+l5[2]*
    m5[0])/2,(l5[1]*m5[2]+l5[2]*m5[1])/2,l5[2]*m5[2]]
A

from sympy import Matrix
a=Matrix(A)
c=a.nullspace()
c=np.array(c).astype(np.int)
c=c/max(c[0])
c

S_mat = np.zeros((2,2))
S_mat[0,0] = c[0][0]
S_mat[1,0] = 0.5 * c[0,1]
S_mat[0,1] = 0.5 * c[0,1]
S_mat[1,1] = c[0,2]
S_mat

U, D, Vt = np.linalg.svd(S_mat)
K = np.zeros((2,2))
D_sq = np.diag(np.sqrt(D))
K = np.dot(Vt, np.dot(D_sq, Vt))
v = np.zeros((2,1))
rhs_vA = np.zeros((2,1))
rhs_vA [0] = 0.5 * c[0,3]
rhs_vA [1] = 0.5 * c[0,4]

v = np.dot(np.linalg.inv(K), rhs_vA)

H = np.zeros((3,3))

#Filling up the homography
H[0]=np.array([K[0,0], K[0,1], 0])
H[1]=np.array([K[1,0], K[1,1], 0])
H[2]=np.array([v[0], v[1], 1])

H_inv=np.linalg.inv(H)
H_inv

img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
print(img_coords.T)

new_img_coords=np.zeros([4,2])

for i in range(4):

  new_coords=np.matmul(H_inv,img_coords[i].T)
  new_coords=new_coords/new_coords[2]
  new_coords=np.rint(new_coords).astype(int)
  new_img_coords[i,:]=new_coords[0:2]


scale1=width/(max(new_img_coords[:,0])-min(new_img_coords[:,0]))
scale2=height/(max(new_img_coords[:,1])-min(new_img_coords[:,1]))
scale=max(scale1, scale2)
```

```
print(scale)

tx=np.round(np.min(new_img_coords[:,0]))
ty=np.round(np.min(new_img_coords[:,1]))
print(tx,ty)

new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale)
    .astype(int)
new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale).
    astype(int)
new_image=np.zeros((new_height,new_width,3))
print(new_image.shape)
max_X=im1.shape[1]
max_Y=im1.shape[0]

blank_image=np.zeros((new_height,new_width,3))
for i in range(0,new_image.shape[1]-1):  #x
  for j in range(0,new_image.shape[0]-1): #y

    k1=i/scale+tx
    k2=j/scale+ty
    init=np.array((k1,k2,1))
    mapped=np.matmul(H,init)
    mapped=np.rint(mapped/mapped[2])
    mapped=mapped.astype(int)
    #if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
    if(mapped[0] > 0 and mapped[1] > 0 and mapped[0] < im1.shape[1] and mapped
    [1] < image.shape[0]):
       blank_image[j,i]=im1[mapped[1],mapped[0]]

plt.figure(figsize=(20,20))
plt.imshow(blank_image.astype(np.int))
ii=blank_image.astype(np.uint8)
plt.imsave("onestepm1.jpeg",ii)




"""## **My Image 2**

**Point to Point Method**
"""

def get_4_points(n):
  if n==1:  #image 1a
    #for the image
    P=[285,594]    #x1,y1
    Q=[453,606] #x2,y2
    R= [453,500]#x3,y3
    S= [294,474] #x4,y4
    X_prime=P+Q+R+S
    return np.array([P,Q,R,S]),X_prime

def get_cat_points():
  #for the kittens
  P_dash=[0,120]
  Q_dash=[300,120]
```

```
  R_dash=[300,0]
  S_dash=[0,0]
  X=P_dash+Q_dash+R_dash+S_dash
  return np.array([P_dash,Q_dash,R_dash,S_dash]),X
'''
def get_4_points(n):
  if n==1: #image 1a
    #for the image
    P=[297,501]   #x1,y1
    Q=[391,496] #x2,y2
    R= [295,164]#x3,y3
    S=[387,174]  #x4,y4
    X_prime=P+Q+R+S
    return np.array([P,Q,R,S]),X_prime

def get_cat_points():
  #for the kittens
  P_dash=[0,250]
  Q_dash=[100,250]
  R_dash=[0,0]
  S_dash=[100,0]
  X=P_dash+Q_dash+R_dash+S_dash
  return np.array([P_dash,Q_dash,R_dash,S_dash]),X
'''

image=cv2.imread("/content/drive/My Drive/hw3_Task1_Images/Images/castle.jpg")
image_orig = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
im1 = np.copy(image_orig)
plt.imshow(im1)
print(im1.shape)
plt.imsave("my_img2.jpeg",im1)

P1,X=get_4_points(1)
P2,X_prime=get_cat_points()

draw_bbox(im1,P1)

H=Homography_matrix(X,X_prime)
print(H)
H_inv=np.linalg.inv(H)

#get the image coordinates in world plane
width=im1.shape[1]
height=im1.shape[0]
print("width(x)=",width)
print("height(y)=",height)

img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
print(img_coords.T)

'''
new_coords=np.matmul(H,img_coords.T)
new_coords=new_coords/new_coords[2]
new_coords=np.rint(new_coords).astype(int)
new_img_coords=new_coords.T[:,:2]
new_img_coords
'''
new_img_coords=np.zeros([4,2])
```

```
for i in range(4):

  new_coords=np.matmul(H,img_coords[i].T)
  new_coords=new_coords/new_coords[2]
  new_coords=np.rint(new_coords).astype(int)
  new_img_coords[i,:]=new_coords[0:2]

scale1=width/(max(new_img_coords[:,0])-min(new_img_coords[:,0]))
scale2=height/(max(new_img_coords[:,1])-min(new_img_coords[:,1]))
scale=max(scale1, scale2)
print(scale)

tx=np.round(np.min(new_img_coords[:,0]))
ty=np.round(np.min(new_img_coords[:,1]))
print(tx,ty)

new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale2
    ).astype(int)
new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale1)
    .astype(int)
new_image=np.zeros((new_height,new_width,3))
new_width,new_height

max_X=im1.shape[1]
max_Y=im1.shape[0]

import math
'''
for i in range(0,new_width):  #x
  for j in range(0,new_height): #y

    k1=i/scale1+tx
    k2=j/scale2+ty
    init=np.array((k1,k2,1))
    mapped=np.matmul(H_inv,init)
    mapped=np.rint(mapped/mapped[2])
    mapped=mapped.astype(int)
    if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
      new_image[j,i]=im1[mapped[1],mapped[0]]

'''
def apply_homography_vectorized(image, H):
    #To find size of the real world image
    H_inv = np.linalg.inv(H)

    image_corners = [[0,0],[0,image.shape[0]-1],[image.shape[1]-1,image.shape
    [0]-1],[image.shape[1]-1,0]]
    image_corners = np.asarray(image_corners)
    world_corners = np.zeros((4,2))
    image_shape = image.shape

    for i in range(len(image_corners)):
        point_img = np.append(image_corners[i],1)
        point_real = np.matmul(H,point_img)
        point_real = np.asarray((math.floor(point_real[0]/point_real[2]),math.
    floor(point_real[1]/point_real[2])))
        world_corners[i] = point_real.astype(int)

```

```
    print(world_corners)
    x_min = np.min(world_corners[:,0]).astype(int)+1
    x_max = np.max(world_corners[:,0]).astype(int)
    y_min = np.min(world_corners[:,1]).astype(int)+1
    y_max = np.max(world_corners[:,1]).astype(int)
    x_lim = x_max-x_min
    y_lim = y_max-y_min

    #IDK stack overflow did this
    H_inverse = np.linalg.inv(H)
    HC = np.ones((3, x_lim*y_lim)).astype(int)
    for indexH in range(x_lim):
        HC[0, indexH*y_lim:indexH*y_lim+y_lim] = np.repeat([indexH], y_lim) + x_min
        HC[1, indexH*y_lim:indexH*y_lim+y_lim] = np.arange(y_lim) + y_min
    HC_ = np.matmul(H_inverse, HC)
    HC_ = (np.round(HC_/HC_[2,:])).astype(int)

    #Ensure points lie inside image
    boundary = np.logical_and(HC_ [0,:]>=0, HC_[0,:]<=np.int64(image.shape[1]-1))
    boundary = np.logical_and(HC_ [1,:]>=0, boundary)
    boundary = np.logical_and(HC_[1,:]<=np.int64(image.shape[0]-1), boundary)


    world_image = np.zeros((y_lim,x_lim,3)).astype(int)
    for index in np.arange(x_lim*y_lim)[boundary]:
        world_image [HC[1,index]-y_min, HC[0,index]-x_min, :] = \
            image[HC_[1,index], HC_[0,index], :]
    return world_image.astype(np.uint8)

new_image=apply_homography_vectorized(im1,H)

plt.figure(figsize=(20,20))
plt.imshow(new_image.astype(int))
ii=new_image.astype(np.uint8)
plt.imsave("pointsm2.jpeg",ii)

"""**Two Step Method**

*Projection Removal*
"""

P1=[289,525]   #x1,y1
P2=[332,531] #x2,y2
P3=[335,479] #x3,y3
P4= [292,472]

'''
Q1=[410,120] #l3
Q2=[533,8] #l3
Q3=[409,374] #l4
Q4=[539,312] #l4
'''
Q1=P1
Q2=P4
Q3=P2
Q4=P3
```

```
im_copy=im1.copy()
##https://www.geeksforgeeks.org/python-opencv-cv2-line-method/ for drawing
    lines on image
line_img=cv2.line(im_copy, tuple(P1), tuple(P2), (255,0,0), 5)
line_img=cv2.line(line_img, tuple(P3), tuple(P4), (255,0,0), 5)
line_img=cv2.line(line_img, tuple(Q1), tuple(Q2), (0,255,0), 5)
line_img=cv2.line(line_img, tuple(Q3), tuple(Q4), (0,255,0), 5)
plt.imshow(line_img)
plt.imsave("parallel_lines1.jpeg",line_img)

P_1=np.array(P1)
P_1=np.append(P_1,1)
P_2=np.array(P2)
P_2=np.append(P_2,1)
l1=np.cross(P_1,P_2)

P_3=np.array(P3)
P_3=np.append(P_3,1)
P_4=np.array(P4)
P_4=np.append(P_4,1)
l2=np.cross(P_3,P_4)

PV1=np.cross(l1,l2)
PV1=PV1/PV1[2]
print(PV1)

Q_1=np.array(Q1)
Q_1=np.append(Q_1,1)
Q_2=np.array(Q2)
Q_2=np.append(Q_2,1)
l3=np.cross(Q_1,Q_2)

Q_3=np.array(Q3)
Q_3=np.append(Q_3,1)
Q_4=np.array(Q4)
Q_4=np.append(Q_4,1)
l4=np.cross(Q_3,Q_4)
'''
Q_1=np.array(P1)
Q_1=np.append(Q_1,1)
Q_2=np.array(P3)
Q_2=np.append(Q_2,1)
l3=np.cross(Q_1,Q_2)

Q_3=np.array(P2)
Q_3=np.append(Q_3,1)
Q_4=np.array(P4)
Q_4=np.append(Q_4,1)
l4=np.cross(Q_3,Q_4)
'''
PV2=np.cross(l3,l4)
PV2=PV2/PV2[2]
print(PV2)

lV=np.cross(PV1,PV2)
lV=lV/lV[2]
lV

```

```
H2=np.array([[1,0,0],[0,1,0],lV])
H2
print(H2)
H2_inv=np.linalg.inv(H2)

img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
print(img_coords.T)

new_img_coords=np.zeros([4,2])
for i in range(4):

  new_coords=np.matmul(H2,img_coords[i].T)
  new_coords=new_coords/new_coords[2]
  new_coords=np.rint(new_coords).astype(int)
  new_img_coords[i,:]=new_coords[0:2]

scale1=width/(max(new_img_coords[:,0])-min(new_img_coords[:,0]))
scale2=height/(max(new_img_coords[:,1])-min(new_img_coords[:,1]))
scale=max(scale1, scale2)
print(scale)

tx=np.round(np.min(new_img_coords[:,0]))
ty=np.round(np.min(new_img_coords[:,1]))
print(tx,ty)

new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale)
    .astype(int)
new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale).
    astype(int)
new_image=np.zeros((new_height,new_width,3))
print(new_image.shape)

max_X=im1.shape[1]
max_Y=im1.shape[0]

for i in range(0,new_width-1):  #x
  for j in range(0,new_height-1): #y

    k1=i/scale+tx
    k2=j/scale+ty
    init=np.array((k1,k2,1))
    mapped=np.matmul(H2_inv,init)
    mapped=np.rint(mapped/mapped[2])
    mapped=mapped.astype(int)
    if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
      new_image[j,i]=im1[mapped[1],mapped[0]]

plt.imshow(new_image.astype(int))
ii=new_image.astype(np.uint8)
plt.imsave("projm2.jpeg",ii)

"""*Affine Removal*"""

im_copy=im1.copy()
P1=[289,525]    #x1,y1
P2=[332,531] #x2,y2
P3=[335,479] #x3,y3
P4= [292,472]
```

```
line_img=cv2.line(im_copy, tuple(P1), tuple(P3), (255,0,0), 5)
line_img=cv2.line(line_img, tuple(P2), tuple(P4), (255,0,0), 5)
line_img=cv2.line(line_img, tuple(P1), tuple(P2), (0,255,0), 5)
line_img=cv2.line(line_img, tuple(P3), tuple(P2), (0,255,0), 5)
plt.imshow(line_img)
plt.imsave("perpendm2.jpeg",line_img)

#map the 4 points onto the projective corrected image plane
P_homo=np.array(P1)
P_homo=np.append(P_homo,1)
Q_homo=np.array(P2)
Q_homo=np.append(Q_homo,1)
R_homo=np.array(P3)
R_homo=np.append(R_homo,1)
S_homo=np.array(P4)
S_homo=np.append(S_homo,1)

P_dash_homo=np.matmul(H2,P_homo)
P_dash=P_dash_homo/P_dash_homo[-1]
Q_dash_homo=np.matmul(H2,Q_homo)
Q_dash=Q_dash_homo/Q_dash_homo[-1]
R_dash_homo=np.matmul(H2,R_homo)
R_dash=R_dash_homo/R_dash_homo[-1]
S_dash_homo=np.matmul(H2,S_homo)
S_dash=S_dash_homo/S_dash_homo[-1]

#initialize the matrices
A = np.zeros((2,2))
S_mat = np.ones((2,2))
H_c = np.identity(3)

#get the first set of perpendicular lines
l1 = np.cross(P_dash,R_dash) #l'
m1 = np.cross(S_dash,Q_dash) #m'

#get the second set of perpendicular lines
l2 = np.cross(P_dash,Q_dash)
m2 = np.cross(Q_dash,R_dash)

#get the A and  matrix
A[0,0] = l1[0]*m1[0]
A[0,1] = l1[0]*m1[1] + l1[1]*m1[0]
A[1,:] = [l2[0]*m2[0], l2[0]*m2[1] + l2[1]*m2[0]]

B = np.array([-l1[1]*m1[1], -l2[1]*m2[1]])

#Find out S
S_elem = np.matmul(np.linalg.pinv(A),B)
S_mat[0,:] = [S_elem[0], S_elem[1]]
S_mat[1,0] = S_elem[1]
U, D, Vt = np.linalg.svd(S_mat, full_matrices=True)
D_sq = np.zeros((2,2))
D_sq[0,0] = np.sqrt(D[0])
D_sq[1,1] = np.sqrt(D[1])

#Get the value of A (AA'=S)
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
      ).astype(int)
2446 new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale1)
      .astype(int)
2447 new_image=np.zeros((new_height,new_width,3))
2448 print(new_image.shape)
2449 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)
```

```
plt.imsave("affinem2.jpeg",ii)

"""**One Step Method**"""

P=[286,594]    #x1,y1
Q=[397,601] #x2,y2
R= [398,490]#x3,y3
S=[292,473]  #x4,y4

P_homo=np.array(P)
P_homo=np.append(P_homo,1)
Q_homo=np.array(Q)
Q_homo=np.append(Q_homo,1)
R_homo=np.array(R)
R_homo=np.append(R_homo,1)
S_homo=np.array(S)
S_homo=np.append(S_homo,1)

l1=np.cross(P_homo,Q_homo)  #pl1
l1=l1/l1[2]
m1=np.cross(Q_homo,R_homo)  #pl1
m1=m1/m1[2]

l2=np.cross(Q_homo,R_homo)  #pl2
l2=l2/l2[2]
m2=np.cross(R_homo,S_homo)  #pl2
m2=m2/m2[2]

l3=np.cross(R_homo,S_homo)  #pl3
l3=l3/l3[2]
m3=np.cross(S_homo,P_homo)  #pl3
m3=m3/m3[2]

l4=np.cross(S_homo,P_homo)  #pl4
l4=l4/l4[2]
m4=np.cross(P_homo,Q_homo)  #pl4
m4=m4/m4[2]

l5=np.cross(P_homo,R_homo)  #pl5
l5=l5/l5[2]
m5=np.cross(Q_homo,S_homo)  #pl5
m5=m5/m5[2]

A=np.zeros((5,6))
A
A[0]=[l1[0]*m1[0],(l1[1]*m1[0]+l1[0]*m1[1])/2,l1[1]*m1[1],(l1[0]*m1[2]+l1[2]*
    m1[0])/2,(l1[1]*m1[2]+l1[2]*m1[1])/2,l1[2]*m1[2]]
A[1]=[l2[0]*m2[0],(l2[1]*m2[0]+l2[0]*m2[1])/2,l2[1]*m2[1],(l2[0]*m2[2]+l2[2]*
    m2[0])/2,(l2[1]*m2[2]+l2[2]*m2[1])/2,l2[2]*m2[2]]
A[2]=[l3[0]*m3[0],(l3[1]*m3[0]+l3[0]*m3[1])/2,l3[1]*m3[1],(l3[0]*m3[2]+l3[2]*
    m3[0])/2,(l3[1]*m3[2]+l3[2]*m3[1])/2,l3[2]*m3[2]]
A[3]=[l4[0]*m4[0],(l4[1]*m4[0]+l4[0]*m4[1])/2,l4[1]*m4[1],(l4[0]*m4[2]+l4[2]*
    m4[0])/2,(l4[1]*m4[2]+l4[2]*m4[1])/2,l4[2]*m4[2]]
A[4]=[l5[0]*m5[0],(l5[1]*m5[0]+l5[0]*m5[1])/2,l5[1]*m5[1],(l5[0]*m5[2]+l5[2]*
    m5[0])/2,(l5[1]*m5[2]+l5[2]*m5[1])/2,l5[2]*m5[2]]
A

from sympy import Matrix
```

```
a=Matrix(A)
c=a.nullspace()
c=np.array(c).astype(np.int)
c=c/max(c[0])
c

S_mat = np.zeros((2,2))
S_mat[0,0] = c[0][0]
S_mat[1,0] = 0.5 * c[0,1]
S_mat[0,1] = 0.5 * c[0,1]
S_mat[1,1] = c[0,2]
S_mat

U, D, Vt = np.linalg.svd(S_mat)
K = np.zeros((2,2))
D_sq = np.diag(np.sqrt(D))
K = np.dot(Vt, np.dot(D_sq, Vt))

v = np.zeros((2,1))
rhs_vA = np.zeros((2,1))
rhs_vA [0] = 0.5 * c[0,3]
rhs_vA [1] = 0.5 * c[0,4]

v = np.dot(np.linalg.inv(K), rhs_vA)

H = np.zeros((3,3))

#Filling up the homography
H[0]=np.array([K[0,0], K[0,1], 0])
H[1]=np.array([K[1,0], K[1,1], 0])
H[2]=np.array([v[0], v[1], 1])

H_inv=np.linalg.inv(H)
H_inv

img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
print(img_coords.T)

new_img_coords=np.zeros([4,2])

for i in range(4):

  new_coords=np.matmul(H_inv,img_coords[i].T)
  new_coords=new_coords/new_coords[2]
  new_coords=np.rint(new_coords).astype(int)
  new_img_coords[i,:]=new_coords[0:2]


scale1=width/(max(new_img_coords[:,0])-min(new_img_coords[:,0]))
scale2=height/(max(new_img_coords[:,1])-min(new_img_coords[:,1]))
scale=max(scale1, scale2)
print(scale)

tx=np.round(np.min(new_img_coords[:,0]))
ty=np.round(np.min(new_img_coords[:,1]))
print(tx,ty)

new_height=np.round((max(new_img_coords[:,1])-min(new_img_coords[:,1]))*scale)
```

```
    .astype(int)
new_width=np.round((max(new_img_coords[:,0])-min(new_img_coords[:,0]))*scale).
    astype(int)
new_image=np.zeros((new_height,new_width,3))
print(new_image.shape)
max_X=im1.shape[1]
max_Y=im1.shape[0]

blank_image=np.zeros((new_height,new_width,3))
for i in range(0,new_image.shape[1]-1):  #x
  for j in range(0,new_image.shape[0]-1): #y

    k1=i/scale+tx
    k2=j/scale+ty
    init=np.array((k1,k2,1))
    mapped=np.matmul(H,init)
    mapped=np.rint(mapped/mapped[2])
    mapped=mapped.astype(int)
    #if mapped[0]>=0 and mapped[0]<max_X and mapped[1]>=0 and mapped[1]<max_Y:
    if(mapped[0] > 0 and mapped[1] > 0 and mapped[0] < im1.shape[1] and mapped
    [1] < image.shape[0]):
       blank_image[j,i]=im1[mapped[1],mapped[0]]

plt.figure(figsize=(20,20))
plt.imshow(blank_image.astype(np.int))
ii=blank_image.astype(np.uint8)
plt.imsave("onestepM2.jpeg",ii)
```