Aditya Chauhan
ECE661: Computer Vision
September 20, 2022

Homework 3

1. INTRODUCTION

In this homework two form of transformations are removed for removing distortions in the images, *projective* and *affine*.

To achieve this task three steps are implemented.

For the implementation of the three tasks I in the start marked 5 set of points that can result in 20 pair of orthogonal lines. Figure 1 are images provided with this homework and Figure 2 are the images that are personally found and used.

(A) Building

(B) Nighthawk

FIGURE 1. Input images originally provided with 20 points marked which are selected for solving Task 1 and 2. Points of same color are used for obtaining pairs of lines that work in correspondence with each other.

1.1. **TASK 1 : Point 2 Point Correspondence.** In this task I used a four points from above 20 points to be used in point to point homography technique. i selected the points which due to distortions are not in complete 90 degrees angle and I mapped them to the points that would be corresponding to them if the lines formed by these points would be 90 degrees. Example of this is: If my points are X as shown below than there corresponding points to make all the angles between two adjacent edges 90 degree will be possible if points are as shown in X'.

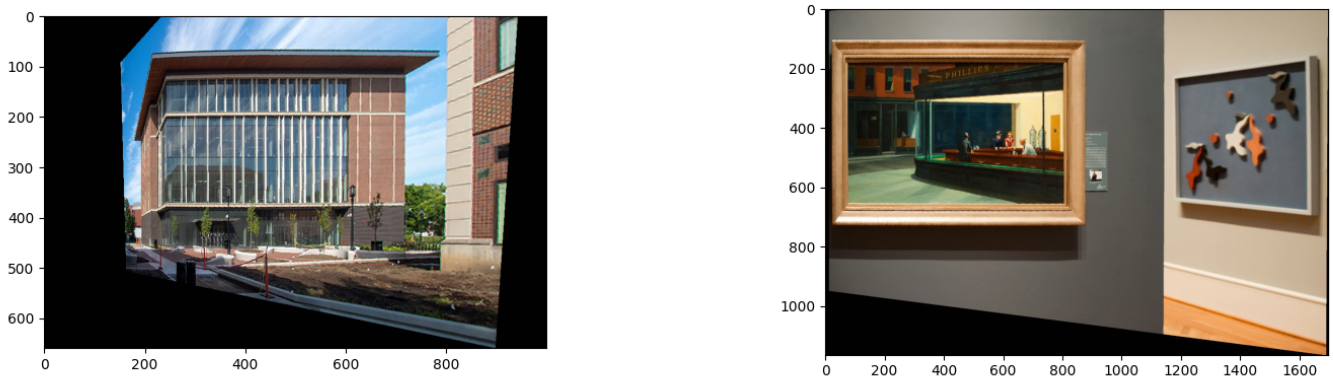
$$X = \begin{bmatrix} 325 & 124 \\ 436 & 157 \\ 287 & 611 \\ 403 & 627 \end{bmatrix} \qquad X' = \begin{bmatrix} 325 & 124 \\ 436 & 124 \\ 325 & 611 \\ 436 & 611 \end{bmatrix}$$

To map we just follow the $X' = HX$ as a relation and results are shown for this homography in Figure 3 for originally provided images and Figure 4 for personally collected images.

1.2. **TASK 2 : 2-Step Method.** For this method I first remove the projective distortions in the image and than affine distortions.

(A) Rack (B) Building

FIGURE 2. Input images taken personally with 20 points marked which are selected for solving Task 1 and 2. Points of same color are used for obtaining pairs of lines that work in correspondence with each other.

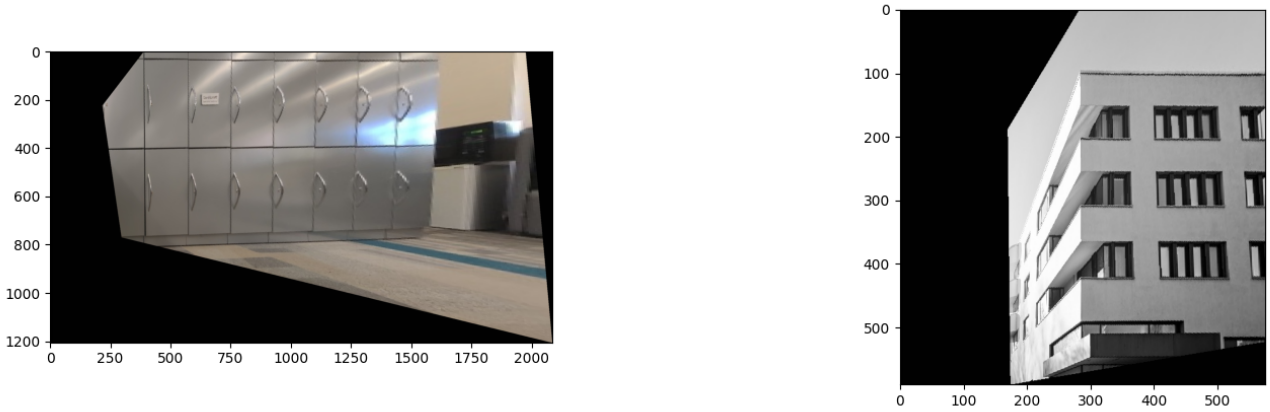


(A) Building (B) Nighthawk

FIGURE 3. Output Images after Point to Point homography for originally provided images.

1.2.1. ***Removing Projective distortions***. For this I use one set of points from my pre defined 5 set of points to get the pair of parallel lines in both horizontal and vertical directions respectively. That means 4 lines total which are possible by just using four points.This is shown in Figure 5 and 6 Than I calculate the vanishing points from both line pairs. Each parallel line pair gives me one point. Using these two points I calculate the vanishing line which I than map to l_∞. To do this I make a homography with three homogeneous coordinate of vanishing line(l_{VL}). If:

$$l_{VL} = \begin{bmatrix} l_1 \\ l_2 \\ l_3 \end{bmatrix} \qquad than \qquad H_{VL} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & l_3 \end{bmatrix}$$



(A) Rack (B) Building

FIGURE 4. Output Images after Point to Point homography for personally collected images.

Using this H_{VL} we can remove the projective distortions. What remains now is to purely affine. Results after removing the projective distortions are shown in Figure 7 for originally provided images and Figure 8 for personally acquired images.

(A) Building (B) Nighthawk

FIGURE 5. Input images showing the parallel lines chosen for removing the projective distortion in originally provided images. The lines are chosen in pair of horizontal and vertical. Single point set is used to get these lines hence the same color.

1.2.2. ***Removing Affine distortions***. For this task we need to use the equation:

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

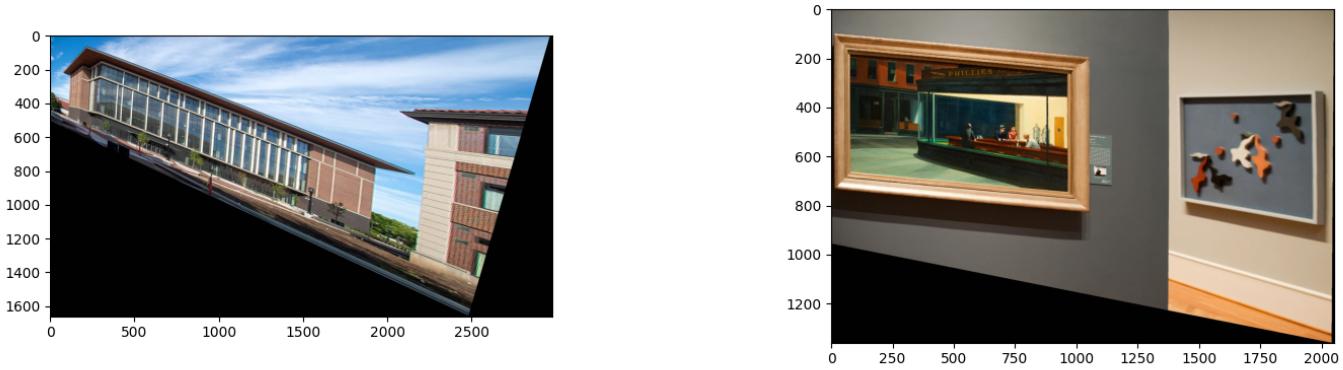
Since using this equation we are trying to get rid of affine distortion which means we are trying to get back the 90 degree angle between the lines. SO that measn $\theta = 90$ so $cos\theta = 0$. Thus only relevant equation left is the numeration that is:

$$l^T C_\infty^* m = 0$$

Now we need a pair of lines that make the 90 degree angle and in out case since we have distorted image we first use that to the pair of lines that can potentially become a 90 degree after removing the distortion, Figure 9 and 10 shows the concept of lines that are considered when taking 90 degree angles in account

(A) Rack (B) Building

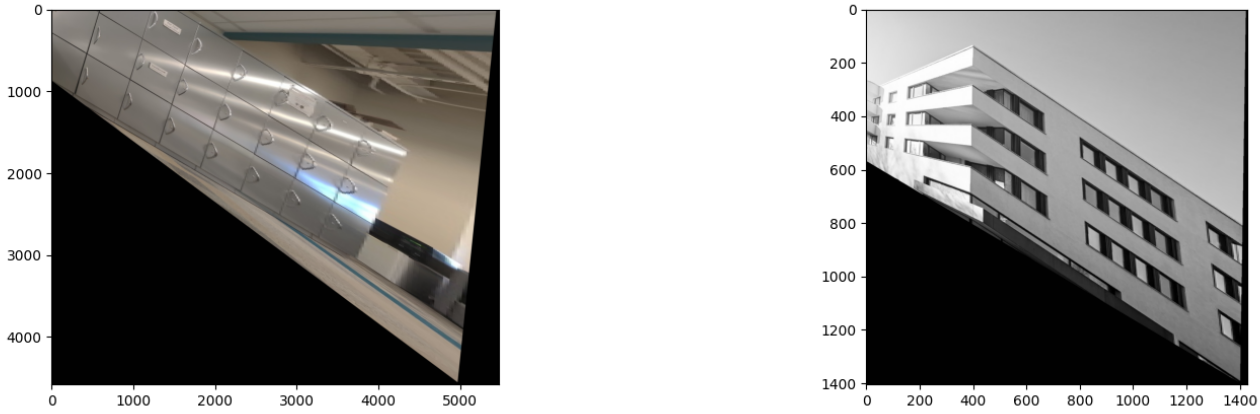
FIGURE 6. Input images showing the parallel lines chosen for removing the projective distortion in personally collected images. The lines are chosen in pair of horizontal and vertical. Single point set is used to get these lines hence the same color.



(A) Building (B) Nighthawk

FIGURE 7. Output Images after removing projective distortion for originally provided images.

and for this we have selected only two pair of lines from two set of points in this case. To remove the affine distortion we modify the equation by replacing l and m with there homography equation. For line it is



(A) Rack (B) Building

FIGURE 8. Output Images after removing projective distortion from personally collected images.

give by $l' = H^{-T}l$ similar for m. Replacing this in equation and rearranging we get

$$cos(\theta) = l'^T H C^*_\infty H^T m'$$

$$where \quad C^*_\infty = \begin{bmatrix} I_{2\times2} & 0 \\ 0^T & 1 \end{bmatrix}_{3\times3}$$

$$and \quad H = \begin{bmatrix} A_{2\times2} & t \\ 0^T & 1 \end{bmatrix}_{3\times3}$$

$$l' = \begin{bmatrix} l'_1 \\ l'_2 \\ l'_3 \end{bmatrix} \quad and \quad h' = \begin{bmatrix} h'_1 \\ h'_2 \\ h'_3 \end{bmatrix}$$

Placing all these values in the equation for $cos\theta$ and expanding the equation we get:

$$\begin{bmatrix} l'_{11}m'_{11} & (l'_{11}m'_{12} + l'_{12}m'_{11}) \\ l'_{21}m'_{21} & (l'_{21}m'_{22} + l'_{22}m'_{21}) \end{bmatrix} \begin{bmatrix} s_{11} \\ s_{12} \end{bmatrix} = \begin{bmatrix} -l'_{22}m'_{22} \\ -l'_{22}m'_{22} \end{bmatrix}$$

$$where \quad S = \begin{bmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{bmatrix} \quad here \;\; s_{12} == s_{21} \;\; and \quad s_{22} = 1$$

Once we solve for s we get a 2×2 matrix which we person SVD on and get the value of A one we have an A we have a homography than can be applied to the image to remove the affine distortions.

Results for this are shown in Figure 11 and 12

1.3. **TASK 3 : 1-Step Method.** In this method we are trying to remove both projective and affine in one step. To do this we modify the $cos\theta$ equation to consider $C^{*\prime}_\infty$ by this matrix:

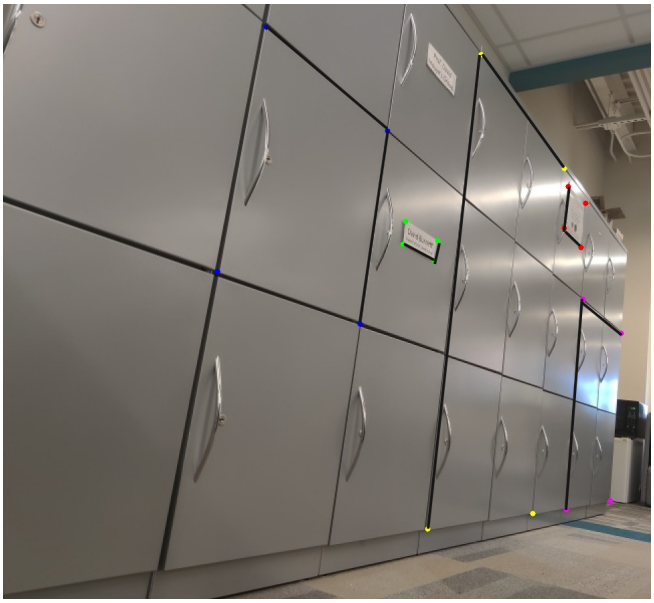
$$C^{*\prime}_\infty = \begin{bmatrix} AA^T & Av \\ v^T A^T & v^T v \end{bmatrix} = \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix}$$

Again to solve this we will use above equation but this time we need minimum of five pair of lines that can potentially form 90 degree angle. Since we can make f=1 leaving us with 5 unknowns. The equation

(A) Building

(B) Nighthawk

FIGURE 9. Input images showing the lines that are considered at a time as a pair of each other forming 90 degree angle are shown between the points in the image. In each set of point it is to be noted that only one pair is marked by all the four pairs are considered for each set of point. And the marking of line in the image is to serve as a POC for understanding what all pair of lines are possible.



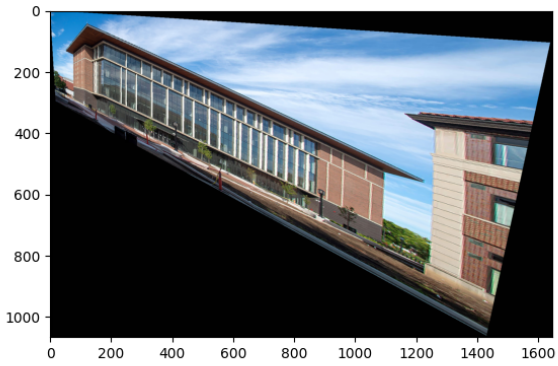
(A) Rack

(B) Building

FIGURE 10. Input images showing the lines that are considered at a time as a pair of each other forming 90 degree angle are shown between the points in the image. In each set of point it is to be noted that only one pair is marked by all the four pairs are considered for each set of point. And the marking of line in the image is to serve as a POC for understanding what all pair of lines are possible.

to solve is

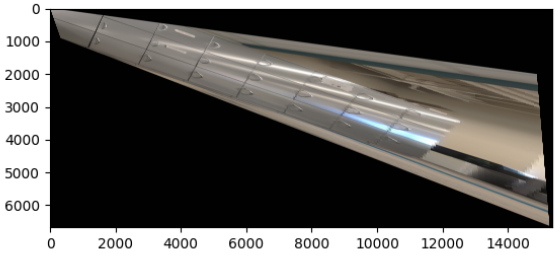
$$\begin{bmatrix} m'_1 l'_1 & l'_1 m'_2 + l'_2 m'_1 & l'_2 m'_2 & l'_1 m'_3 + l'_3 m'_1 & l'_2 m'_3 + l'_3 m'_2 \end{bmatrix} \begin{bmatrix} a \\ b/2 \\ c \\ d/2 \\ e/2 \end{bmatrix} = \begin{bmatrix} -l'_3 m'_3 \end{bmatrix}$$



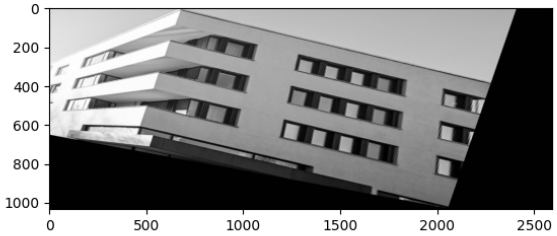
(A) Building

(B) Nighthawk

FIGURE 11. Output Images after removing affine distortion for originally provided images.



(A) Rack



(B) Building

FIGURE 12. Output Images after removing affine distortion from personally collected images.

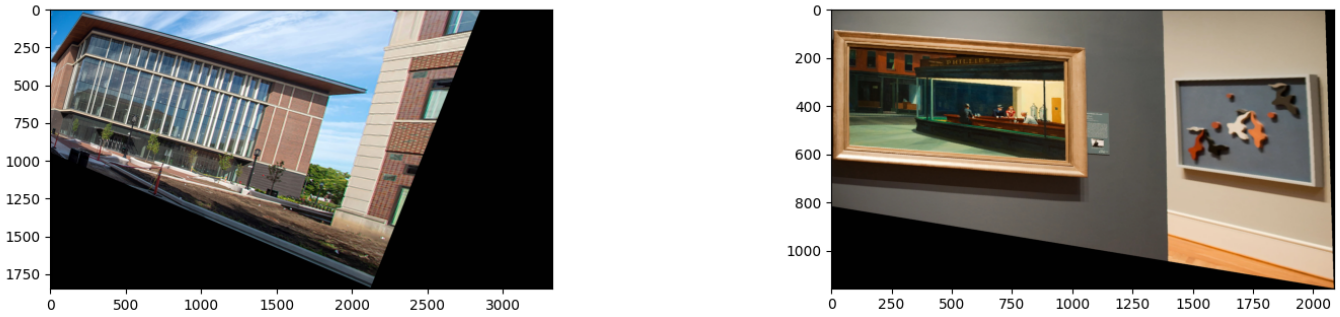
This is single line representation to solve we need minimum of 5. But I did it with more than 5. For my case I considered the 20 pair of lines instead of 5 and solved the problem by using pseudo inverse.

After solving this equation we get the values of a,b,c,d,e and we use them to make homography. Out homography matrix is:

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

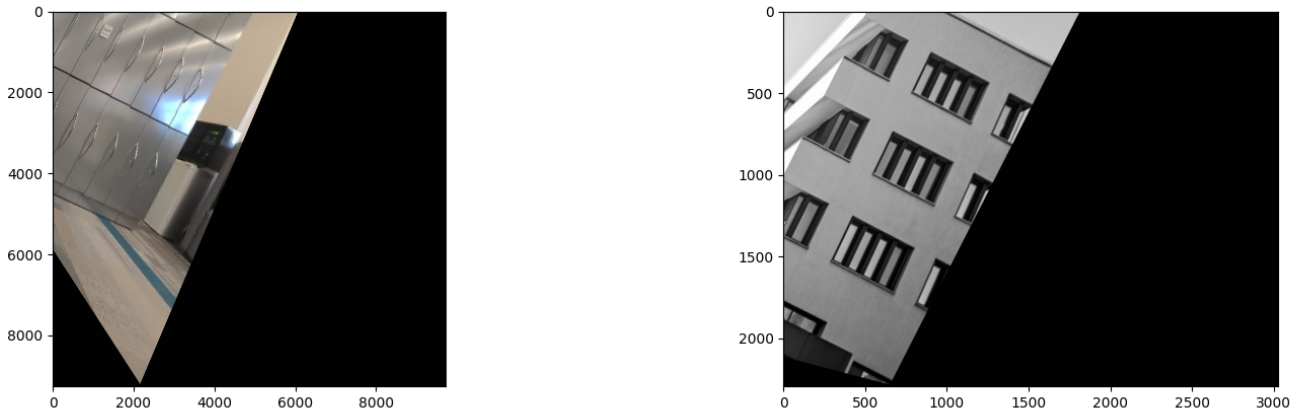
$$where \;\; A.A^T = \begin{bmatrix} a & b/2 \\ b/2 & c \end{bmatrix} \;\; and \;\; v = A^{-1} \begin{bmatrix} d/2 \\ e/2 \end{bmatrix}$$

Y=Using this we get the homography which we can use to remove both projective and affine distortion in one go. The lines selected for this case are all possible non diagonal lines for each point set of same color for each image in Figure 9 and 10. This basically gives us 20 pair of l and m lines. Results for this task are shown in Figure 13 and 14.



(A) Building (B) Nighthawk

FIGURE 13. Output Images after removing both projective and affine distortion for originally provided images in one step.



(A) Rack (B) Building

FIGURE 14. Output Images after removing both projective and affine distortion for personally collected images in one step.

2. CODE

2.1. TASK 1.

```
import configparser
import os
import matplotlib.pyplot as plt
from tqdm import tqdm
from  utility import *
from vision import *

config = configparser.ConfigParser()
config.read('hw3config.txt')

def main():
    x_img_path = os.path.join(config['PARAMETERS']['top_dir'], config['PARAMETERS']['x_path'])
    x_pts = config['PARAMETERS']['ol_pts1']
```

```
    homo_mode = config['PARAMETERS']['homo_mode']
    x_img = readImgCV(x_img_path)

    ho, wo, co = x_img.shape
    print(ho,wo,co)

    ##################################################################################################
    ##################     Homography calculations for P2P correspondenc
    ##################
    ##################################################################################################


    x_pts = str2np(x_pts)
    x_prime_pts=primePtsP2P(x_pts)
    vision = Vision(x_pts, x_prime_pts)
    h = vision.calc_homograpy(homo_mode)
    hinv=np.linalg.inv(h)


    hPrime = xpeqhx(0, 0, h)
    wpa1, hpa1 = hPrime[0], hPrime[1]
    hPrime = xpeqhx(0, ho, h)
    wpa2, hpa2 = hPrime[0], hPrime[1]
    hPrime = xpeqhx(wo, 0, h)
    wpa3, hpa3 = hPrime[0], hPrime[1]
    hPrime = xpeqhx(wo, ho, h)
    wpa4, hpa4 = hPrime[0], hPrime[1]
    wpa = max(wpa1, max(wpa2, max(wpa3, wpa4)))
    hpa = max(hpa1, max(hpa2, max(hpa3, hpa4)))

    ##################################################################################################
    ##################     Loop over empty image to fetch the coord from the
    images       #######
    ##################################################################################################

    empty_img = np.ones((int(hpa), int(wpa), 3), np.int32)
    # print(empty_img.shape)
    for c in tqdm(range(int(wpa))):
        for r in range(int(hpa)):
            X_prime=xpeqhx(c,r,hinv)
            X_prime=X_prime.astype(int)
            if X_prime[1] < ho and X_prime[0] < wo and X_prime[0]>=0 and X_prime
    [1]>=0:
                empty_img[r][c] = x_img[X_prime[1]][X_prime[0]]
    plt.imshow(empty_img)
    plt.show()

if __name__=='__main__':
    main()
```

Listing 1. Code block for Point to Point Correspondence

2.2. **TASK 2.**

```
import configparser
import os
import matplotlib.pyplot as plt
import numpy as np
from utility import *
from tqdm import tqdm

from vision import *

config = configparser.ConfigParser()
config.read('hw3config.txt')

def main():
    x_img_path = os.path.join(config['PARAMETERS']['top_dir'], config['PARAMETERS']['x_path'])
    x_pts = config['PARAMETERS']['ol_pts1']
    x_img = readImgCV(x_img_path)

    ho, wo, co = x_img.shape
    print(ho,wo,co)
    x_pts = str2np(x_pts)

    #
    ##############################################################################################

    ###################     Code block to remove the projective homography using VL     #########
    #
    ##############################################################################################


    hvl, vl = vanishing_line_homography(x_pts)
    hvlinv=np.linalg.inv(hvl)
    hPrime = xpeqhx(0, 0, hvl)
    wpa1, hpa1 = hPrime[0], hPrime[1]
    hPrime = xpeqhx(0, ho, hvl)
    wpa2, hpa2 = hPrime[0], hPrime[1]
    hPrime = xpeqhx(wo, 0, hvl)
    wpa3, hpa3 = hPrime[0], hPrime[1]
    hPrime = xpeqhx(wo, ho, hvl)
    wpa4, hpa4 = hPrime[0], hPrime[1]
    wpa = max(wpa1, max(wpa2, max(wpa3, wpa4)))
    hpa = max(hpa1, max(hpa2, max(hpa3, hpa4)))
    empty_img = np.ones((int(hpa), int(wpa), 3), np.int32)
    print(empty_img.shape)

    for c in tqdm(range(int(wpa))):
        for r in range(int(hpa)):
            X_prime=xpeqhx(c,r,hvlinv)
            # print(X_prime)
            if np.ceil(X_prime[1]) < ho and np.ceil(X_prime[0]) < wo and X_prime[0]>=0 and X_prime[1]>=0:
                empty_img[r][c] = x_img[int(X_prime[1])][int(X_prime[0])]

    plt.imshow(empty_img)
    plt.show()

```

```
51      #
        ##########################################################################################

52      ###################    Code block to remove the affine homography using
        degenate conic #######
53      #
        ##########################################################################################


54
55      ol_pts1 = config['PARAMETERS']['ol_pts1']
56      ol_pts2 = config['PARAMETERS']['ol_pts2']
57      ol_pts1 = str2np(ol_pts1)
58      ol_pts2 = str2np(ol_pts2)
59      pts_array = np.array((ol_pts1,ol_pts2))
60      lset = []
61      mset = []
62      for pts in range(len(pts_array)):
63          lset.append(make_line_hc(pts_array[pts][0], pts_array[pts][1]))
64          mset.append(make_line_hc(pts_array[pts][1], pts_array[pts][3]))
65      L = np.ones((len(lset),2))
66      for lines in range(len(lset)):
67          L[lines][0] = lset[lines][0] * mset[lines][0]
68          L[lines][1] = (lset[lines][0] * mset[lines][1] + lset[lines][1] * mset[
        lines][0])
69      M = np.ones(len(mset))
70      for lines in range(len(lset)):
71          M[lines] = -(lset[lines][1]*mset[lines][1])
72      Linv = np.linalg.inv(L)
73      S = np.dot(Linv, M.T)
74      print(S)
75      Smat = np.ones((2,2))
76      Smat[0][0] = S[0]
77      Smat[0][1] = Smat[1][0] = S[1]
78      Smat[1][1] = 1
79      print("Smat",Smat)
80      U,D2,UT = np.linalg.svd(Smat)
81      print(U,D2,UT)
82      print("D2",D2)
83      D = np.sqrt(D2)
84      D = np.diag(D)
85      print("D", D)
86      A = np.dot(np.dot(U,D), np.transpose(U))
87      # A = np.dot(U,np.dot(D, np.transpose(U)))
88      print("A",A)
89      ha = np.zeros((3,3))
90      ha[0][0] = A[0][0]
91      ha[0][1] = A[0][1]
92      ha[1][0] = A[1][0]
93      ha[1][1] = A[1][1]
94      ha[2][2]=1
95      ha = np.dot(np.linalg.inv(ha), hvl)
96      hainv=np.linalg.inv(ha)
97      print(ha)
98      hPrime = xpeqhx(0, 0, ha)
99      wpa1,hpa1 = hPrime[0],hPrime[1]
100     hPrime = xpeqhx(0, ho, ha)
101     wpa2,hpa2 = hPrime[0],hPrime[1]
102     hPrime = xpeqhx(wo, 0, ha)
103     wpa3,hpa3 = hPrime[0],hPrime[1]
104     hPrime = xpeqhx(wo, ho, ha)
```

```
    wpa4,hpa4 = hPrime[0],hPrime[1]
    wpa = max(wpa1, max(wpa2, max(wpa3,wpa4)))
    hpa = max(hpa1, max(hpa2, max(hpa3,hpa4)))
    empty_img2 = np.ones((int(hpa), int(wpa), 3), np.int32)
    print(empty_img2.shape)
    for c in tqdm(range(int(wpa))):
        for r in range(int(hpa)):
            X_prime = xpeqhx(c, r, hainv)
            if np.ceil(X_prime[1]) < ho and np.ceil(X_prime[0]) < wo and X_prime
    [0] >= 0 and X_prime[1] >= 0:
                empty_img2[r][c] = x_img[int(X_prime[1])][int(X_prime[0])]
    plt.imshow(empty_img2)
    plt.show()

if __name__=='__main__':
    main()
```

LISTING 2. Code block for 2-Step Method

2.3. TASK 3.

```
import configparser
import os
import matplotlib.pyplot as plt
import numpy as np
from tqdm import tqdm
from utility import *
from vision import *

config = configparser.ConfigParser()
config.read('hw3config.txt')

def main():
    x_img_path = os.path.join(config['PARAMETERS']['top_dir'], config['PARAMETERS']
    ['x_path'])

    x_img = readImgCV(x_img_path)

    ho, wo, co = x_img.shape
    print(ho,wo,co)
    ol_pts1 = config['PARAMETERS']['ol_pts1']
    ol_pts2 = config['PARAMETERS']['ol_pts2']
    ol_pts3 = config['PARAMETERS']['ol_pts3']
    ol_pts4 = config['PARAMETERS']['ol_pts4']
    ol_pts5 = config['PARAMETERS']['ol_pts5']

    ol_pts1 = str2np(ol_pts1)
    ol_pts2 = str2np(ol_pts2)
    ol_pts3 = str2np(ol_pts3)
    ol_pts4 = str2np(ol_pts4)
    ol_pts5 = str2np(ol_pts5)

    pts_array = np.array((ol_pts1,ol_pts2,ol_pts3,ol_pts4,ol_pts5))
    lset = []
    mset = []
    for pts in range(len(pts_array)):
        lset.append(make_line_hc(pts_array[pts][0], pts_array[pts][1]))
        mset.append(make_line_hc(pts_array[pts][1], pts_array[pts][3]))
        lset.append(make_line_hc(pts_array[pts][1], pts_array[pts][3]))
```

```
        mset.append(make_line_hc(pts_array[pts][3], pts_array[pts][2]))
        lset.append(make_line_hc(pts_array[pts][3], pts_array[pts][2]))
        mset.append(make_line_hc(pts_array[pts][2], pts_array[pts][0]))
        lset.append(make_line_hc(pts_array[pts][2], pts_array[pts][0]))
        mset.append(make_line_hc(pts_array[pts][0], pts_array[pts][1]))

    lset=np.array(lset)
    mset=np.array(mset)
    print("l here")
    print(lset.shape,lset)
    print("m here")
    print(mset.shape,mset)
    l1 = make_line_hc(ol_pts1[0], ol_pts1[1])
    m1 = make_line_hc(ol_pts1[1], ol_pts1[3])
    print(l1,m1)
    print("end")

    L = np.ones((len(lset),5))
    for lines in range(len(lset)):
        L[lines][0] = lset[lines][0] * mset[lines][0]
        L[lines][1] = (lset[lines][0] * mset[lines][1] + lset[lines][1] * mset[
lines][0]) / 2
        L[lines][2] = lset[lines][1] * mset[lines][1]
        L[lines][3] = (lset[lines][0] * mset[lines][2] + lset[lines][2] * mset[
lines][0]) / 2
        L[lines][4] = (lset[lines][1] * mset[lines][2] + lset[lines][2] * mset[
lines][1]) / 2

    print(L.shape,L)

    M = np.ones(len(mset))*-1
    for lines in range(len(lset)):
        M[lines] = -(lset[lines][2]*mset[lines][2])

    #
##########################################################################################

    ##########    Code block to remove both projective and  affine homography in
one step #######
    #
##########################################################################################


    S=np.linalg.inv(L.T@L)@L.T@M
    # S = np.dot(Linv, M)
    S=S/max(S)
    print("S",S)
    AAT = np.ones((2,2))
    AAT[0][0]=S[0]
    AAT[0][1]=AAT[1][0]=S[1]/2
    AAT[1][1]=S[2]
    d = np.ones(2)
    d[0]=S[3]/2
    d[1]=S[4]/2
    U,D2,UT = np.linalg.svd(AAT)
    D = np.sqrt(D2)
    D = np.diag(D)
    A = np.dot(np.dot(UT,D), UT.T)
    v = np.dot(np.linalg.inv(A),d)
    ha = np.zeros((3,3))
```

```
    ha[0][0] = A[0][0]
    ha[0][1] = A[0][1]
    ha[1][0] = A[1][0]
    ha[1][1] = A[1][1]
    ha[2][0] = v[0]
    ha[2][1] = v[1]
    ha[2][2]=1
    print(ha)
    # hnorm=ha
    hainv=np.linalg.inv(ha)

    hPrime = xpeqhx(0, 0, hainv)
    wpa1,hpa1 = hPrime[0],hPrime[1]
    hPrime = xpeqhx(0, ho, hainv)
    wpa2,hpa2 = hPrime[0],hPrime[1]
    hPrime = xpeqhx(wo, 0, hainv)
    wpa3,hpa3 = hPrime[0],hPrime[1]
    hPrime = xpeqhx(wo, ho, hainv)
    wpa4,hpa4 = hPrime[0],hPrime[1]
    wpa = max(wpa1, max(wpa2, max(wpa3,wpa4)))
    hpa = max(hpa1, max(hpa2, max(hpa3,hpa4)))
    print(wpa,hpa)
    empty_img = np.ones((int(hpa), int(wpa), 3))
    # empty_img = np.ones((2000,2000, 3))
    empty_img=empty_img.astype(int)
    print(empty_img.shape)
    for c in tqdm(range(int(wpa))):
        for r in range(int(hpa)):
            X_prime = xpeqhx(c, r, ha)
            # print(X_prime)
            if np.ceil(X_prime[1]) < ho and np.ceil(X_prime[0]) <wo and X_prime[0]
    >= 0 and X_prime[1] >= 0:
                empty_img[r][c] = x_img[int(X_prime[1])][int(X_prime[0])]
            # else: print(X_prime)
    plt.imshow(empty_img)
    plt.show()

if __name__=='__main__':
    main()
```

LISTING 3. Code block for 1-Step Method

2.4. UTILITY FUNCTIONS AND CONFIG FILE.

```
import os

import cv2
import matplotlib.pyplot as plt
import numpy as np
from einops import rearrange

def readImgCV(path):
    img = cv2.imread(path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    return img

def cvrt2homo(pt):
    return np.append(pt, 1)

```

```
16
17
18 def str2np(s):
19     x_prime_pts = s.split(',')
20     x_prime_pts = [int(val) for val in x_prime_pts]
21     x_prime_pts = np.array(x_prime_pts)
22     x_prime_pts = rearrange(x_prime_pts, '(c h)-> c h ', c=4, h=2)
23     return x_prime_pts
24
25 def save_img(name, path, img):
26     img_path = os.path.join(path, name)
27     img=cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
28     cv2.imwrite(img_path, img)
29
30 def xpeqhx(x,y,h):
31     X = np.array((x, y, 1))
32     X_prime = np.matmul(h, X)
33     X_prime = X_prime / X_prime[2]
34     # X_prime = X_prime.astype(int)
35     return X_prime
36
37 def point2point(x,h):
38     X = np.array((x[0], x[1], 1))
39     X_prime = np.matmul(h, X)
40     X_prime = X_prime/X_prime[2]
41     X_prime = X_prime.astype(int)
42     X_prime = np.array([X_prime[0], X_prime[1]])
43     return X_prime
44 def vanishing_line_homography(x_pts):
45     l12 = np.cross(cvrt2homo(x_pts[0]), cvrt2homo(x_pts[1]))
46     l34 = np.cross(cvrt2homo(x_pts[2]), cvrt2homo(x_pts[3]))
47     l31 = np.cross(cvrt2homo(x_pts[2]), cvrt2homo(x_pts[0]))
48     l42 = np.cross(cvrt2homo(x_pts[3]), cvrt2homo(x_pts[1]))
49     vpt1 = np.cross(l12, l34)
50     vpt1 = vpt1 / vpt1[2]
51     vpt2 = np.cross(l31, l42)
52     vpt2 = vpt2 / vpt2[2]
53     vl = np.cross(vpt1, vpt2)
54     h = np.zeros((3,3))
55     h[0][0] = 1
56     h[1][1] = 1
57     h[2][0] = vl[0]/vl[2]
58     h[2][1] = vl[1]/vl[2]
59     h[2][2] = vl[2]/vl[2]
60     return h, vl
61
62 def primePtsP2P(pts):
63     X=np.ones((len(pts),2))
64     # print(X)
65     X[0,0]=pts[0,0]
66     X[0,1]=pts[0,1]
67     X[1,0]=pts[1,0]
68     X[1,1]=pts[0,1]
69     X[2,0]=pts[0,0]
70     X[2,1]=pts[2,1]
71     X[3,0]=pts[1,0]
72     X[3,1]=pts[2,1]
73     return X
74 def make_line_hc(pts1, pts2):
75     l = np.cross(cvrt2homo(pts1), cvrt2homo(pts2))
```

```
    return l

class Vision:
    def __init__(self, x, x_prime, homo_mat_size=3):
        self.x = x
        self.x_prime = x_prime
        self.h = np.ones((homo_mat_size,homo_mat_size))
        self.h_mat_size = homo_mat_size

    def calc_homograpy(self, homo_mode):
        if homo_mode=='projective':
            A = np.ones((len(self.x)*2,len(self.x)*2))
            C = np.ones(len(self.x)*2)
            for i in range(len(self.x)):
                A[2*i] = np.array([self.x[i][0], self.x[i][1], 1, 0, 0, 0, -self.x[i][0]*self.x_prime[i][0], -self.x[i][1]*self.x_prime[i][0]])
                A[(2*i)+1] = np.array([0, 0, 0, self.x[i][0], self.x[i][1], 1, -self.x[i][0]*self.x_prime[i][1], -self.x[i][1]*self.x_prime[i][1]])
                C[2*i] = self.x_prime[i][0]
                C[(2*i)+1] = self.x_prime[i][1]
            Ainv = np.linalg.inv(A)
            B = np.dot(Ainv, C)
            B = np.append(B, 1)
            self.h = rearrange(B, '(c h)-> c h',c=self.h_mat_size, h=self.h_mat_size)
        if homo_mode=='affine':
            A = np.ones(((len(self.x)-1) * 2, (len(self.x) -1)* 2))
            C = np.ones((len(self.x) -1) * 2)
            for i in range(len(self.x)-1):
                A[2 * i] = np.array([self.x[i][0], self.x[i][1], 1, 0, 0, 0])
                A[(2 * i) + 1] = np.array([0, 0, 0, self.x[i][0], self.x[i][1], 1])
                C[2 * i] = self.x_prime[i][0]
                C[(2 * i) + 1] = self.x_prime[i][1]
            Ainv = np.linalg.inv(A)
            B = np.dot(Ainv, C)
            B = np.append(B, (0,0,1))
            self.h = rearrange(B, '(c h)-> c h', c=self.h_mat_size, h=self.h_mat_size)
        return self.h
```

Listing 4. Code block for homography related utility functions

```
import configparser
import os

from vision import *
import argparse
config = configparser.ConfigParser()
config.read('hw3config.txt')

x_img_path = os.path.join(config['PARAMETERS']['top_dir'], config['PARAMETERS']['x_path'])
top_dir= config['PARAMETERS']['top_dir']
img_name= config['PARAMETERS']['x_path']
img_name = img_name.split('.')[0]
img_name_dotted=img_name+"_dotted.jpg"
x_img = readImgCV(x_img_path)

```

```
ho, wo, co = x_img.shape
print(ho,wo,co)
ol_pts1 = config['PARAMETERS']['ol_pts1']
ol_pts2 = config['PARAMETERS']['ol_pts2']
ol_pts3 = config['PARAMETERS']['ol_pts3']
ol_pts4 = config['PARAMETERS']['ol_pts4']
ol_pts5 = config['PARAMETERS']['ol_pts5']

ol_pts1 = str2np(ol_pts1)
ol_pts2 = str2np(ol_pts2)
ol_pts3 = str2np(ol_pts3)
ol_pts4 = str2np(ol_pts4)
ol_pts5 = str2np(ol_pts5)

color = [(0,0,255),(0,255,0),(255,0,0),(255,255,0),(255,0,255)]
pts_array = np.array((ol_pts1, ol_pts2, ol_pts3, ol_pts4, ol_pts5))
for pts in range(len(pts_array)):
    for i in range(len(pts_array[pts])):
        cv2.circle(x_img, (pts_array[pts][i][0], pts_array[pts][i][1]), radius=4,
   color=color[pts], thickness=-1)

img_path = os.path.join(top_dir, img_name_dotted)
x_img=cv2.cvtColor(x_img, cv2.COLOR_BGR2RGB)
cv2.imwrite(img_path, x_img)
```

Listing 5. Code block for plotting the points

```
[PARAMETERS]
homo_mat_size=3
top_dir=hw3images
output=hw3outputs
#x_path=building.jpg
#x_path=nighthawks.jpg
x_path=rack.jpg
#x_path=photo-building.jpg

#building task 1,2,3
#ol_pts1=243,123,718,292,234,374,723,411
#ol_pts2=360,233,377,239,360,246,374,248
#ol_pts3=483,263,512,271,483,390,512,393
#ol_pts4=266,209,283,211,266,225,283,228
#ol_pts5=684,319,709,325,687,409,713,409

#nighthawk task 1,2,3
#ol_pts1=78,182,804,221,78,653,805,620
#ol_pts2=867,419,925,418,868,583,926,580
#ol_pts3=208,322,257,322,204,489,252,485
#ol_pts4=769,374,795,370,769,468,793,471
#ol_pts5=222,206,248,208,221,277,247,276

#rack task 1,2,3
ol_pts1=393,34,574,189,320,399,533,477
ol_pts2=601,324,649,353,597,358,644,383
ol_pts3=844,272,870,297,838,334,863,363
ol_pts4=714,75,838,245,634,783,791,760
ol_pts5=866,440,923,491,841,754,907,741

#pb task 1,2,3
#ol_pts1=284,104,571,330,278,489,570,553
```

```
#ol_pts2=355,206,392,231,358,250,394,272
#ol_pts3=440,347,526,393,441,386,530,430
#ol_pts4=466,442,505,456,469,484,509,494
#ol_pts5=336,289,393,317,336,335,395,360

mapping=r2d
homo_mode=projective
```

LISTING 6. Config text file where all the points and images are read from in the code