ECE661: Computer Vision

Risi Jaiswal
rjaiswa@purdue.edu

September 22, 2020

In this task we have to remove projective and affine distortion using following three methods:

Point to Point Correspondence

Let pixels in camera images are given by $X_2 = (x, y)$ and $X_2' = (x', y')$ in real world image. I have assumed real world length unit equal to one pixel. Let's assume homography corresponding to distortion is given by following matrix denoted by $\mathbf{H}$

$$\mathbf{H} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

Since We are using homogeneous coordinate system and information lies in ratio so we can get away with one element of matrix. Without loss of generality $i = 1$. Since H is homography which will map points from X_3 to X_3' (where X_3 and X_3' are homogeneous representations of X_2 and X_2'), we can write

$$\implies \mathbf{H}X_3 = X_3' \tag{1}$$

$$\implies \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x'' \\ y'' \\ z'' \end{bmatrix} \tag{2}$$

$$\implies x'' = ax + by + c, y'' = dx + ey + f \text{ and } gx + hy + 1 = z'' \tag{3}$$

$$\implies x' = \frac{x''}{z''} \text{ and } y' = \frac{y''}{z''} \tag{4}$$

$$\implies x' = \frac{ax + by + c}{gx + hy + 1} \text{ and } y' = \frac{dx + ey + f}{gx + hy + 1} \tag{5}$$

$$\implies x' = ax + by + c - gxx' - hx'y \text{ and } y' = dx + ey + f - gxy' - hyy' \tag{6}$$

As we can see to determine $\mathbf{H}$ completely we need at least 8 equations so we need to take 4 points. All these 8 equations can be written as a system of linear

equations in matrix as shown below

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_1' & -y_1x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_1' & -y_1y_1' \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x_2' & -y_2x_2' \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y_2' & -y_2y_2' \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x_3' & -y_3x_3' \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y_3' & -y_3y_3' \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x_4' & -y_4x_4' \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y_4' & -y_4y_4' \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix} = \begin{bmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ x_3' \\ y_3' \\ x_4 \\ y_4' \end{bmatrix}$$

By Solving above system of linear equations we get homography matrix.
By applying inverse of homography matrix we can remove projective and affine distortion in image.

Steps

1- Find H matrix as described in above method
2- Apply inverse of H to camera corners to get real world corners (which origin are x_{min} and y_{min}).
3- Calculate width and height using corners.
4- Calculate scaling of real world and camera image (s).
5- Create empty matrix of real world size and iterate from (x_{min}, y_{min}) point with scale s and copy corresponding points from camera image using H matrix.

Two step Method

In this method first we remove projective distortion by through out vanishing lines at l_∞ then we remove affine distortion by using the property of angle invariant formula derived from dual property of degenerate conics. To push vanishing line first we find the vanishing line by finding the line joining two intersections points from two set of parallel lines (corresponding to real world). Let l_1, l_2 & l_3 are the parameters of vanishing line then homography matrix to push vanishing line at infinity is given by

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & l_3 \end{bmatrix}$$

Applying this homography matrix as similar to previous task , projective distortion got removed.
To remove affine distortion we use following equation which gives angle between two lines :

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

If we take two lines l' and m' in camera image which are orthogonal in corresponding real world then we can write (substituting generic affine homography matrix)

$$l'^T C^*_\infty T m' = l'^T H C^* H^T m' = 0$$

$$\implies \begin{bmatrix} l'_1 & l'_2 & l'_3 \end{bmatrix} \begin{bmatrix} A & t \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} A^T & 0 \\ t & 1 \end{bmatrix} \begin{bmatrix} m'_1 \\ m'_2 \\ m'_3 \end{bmatrix} = 0$$

$$\begin{bmatrix} l'_1 & l'_2 & l'_3 \end{bmatrix} \begin{bmatrix} AA^T & 0 \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} m'_1 \\ m'_2 \\ m'_3 \end{bmatrix} = 0$$

By taking $AA^T = S = \begin{bmatrix} s_{11} & s_{21} \\ s_{12} & s_{22} \end{bmatrix}$

$$\begin{bmatrix} l'_1 & l'_2 \end{bmatrix} \begin{bmatrix} s_{11} & s_{21} \\ s_{12} & s_{22} \end{bmatrix} \begin{bmatrix} m'_1 \\ m'_2 \end{bmatrix} = 0$$

Since we have 2 parameters (since s_{22} can be taken as 1 and $s_{12} = s_{21}$) to know S, we need to find two pairs of orthogonal lines and solve above system of linear equations.
After getting S we can get A by SVD of S.
Omce we know A we can put in generic form of affine homography to get the homography matrix which can be used as in previous method to remove affine distortion.

One step Method

In this method we take projection of C^*_∞ as below

$$\begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix}$$

By substituting this in angle formula and taking orthogonal pair of lines we get following equation

$$\begin{bmatrix} l'_1 & l'_2 & l'_3 \end{bmatrix} \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & 1 \end{bmatrix} \begin{bmatrix} m'_1 \\ m'_2 \\ m'_3 \end{bmatrix} = 0$$

To solve above system of linear equations we require 5 such equations which can be obtained by taking 5 pairs of orthogonal lines. Once we we get a, b, c, d, e parameters we can get A matrix by SVD of S matrix given by

$$S = AA^T = \begin{bmatrix} a & b/2 \\ b/2 & c \end{bmatrix}$$

To get vector v following equation need to be solved

$$Av = \begin{bmatrix} d/2 & e/2 \end{bmatrix}$$

Once we got A and v we can get generic homographic transformation matrix which can be used to remove distortion as similar to previous tasks.

Figure 1: Image 1 with points

Figure 2: Image 1 point to point

Figure 3: Image 2 with points

Figure 4: Image 2 point to point

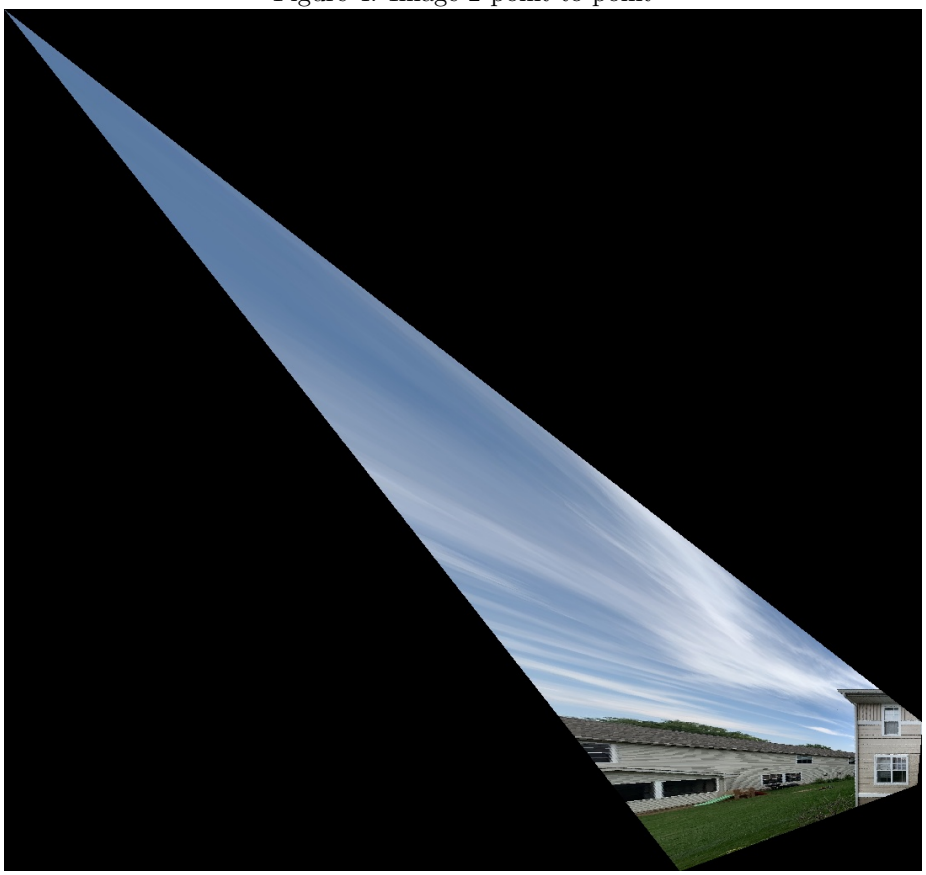

Figure 5: Image 3 with points

Figure 6: Image 3 point to point

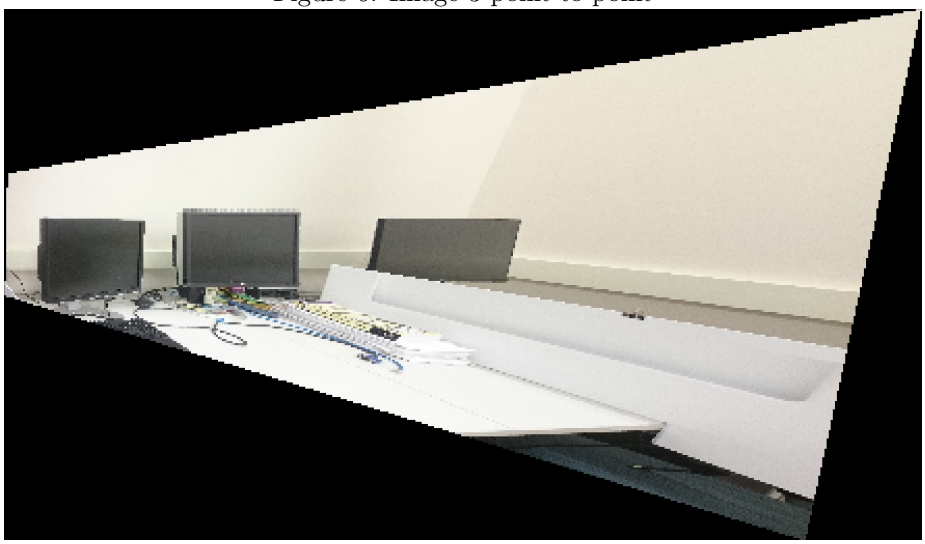

Figure 7: Image 1 with lines for affine

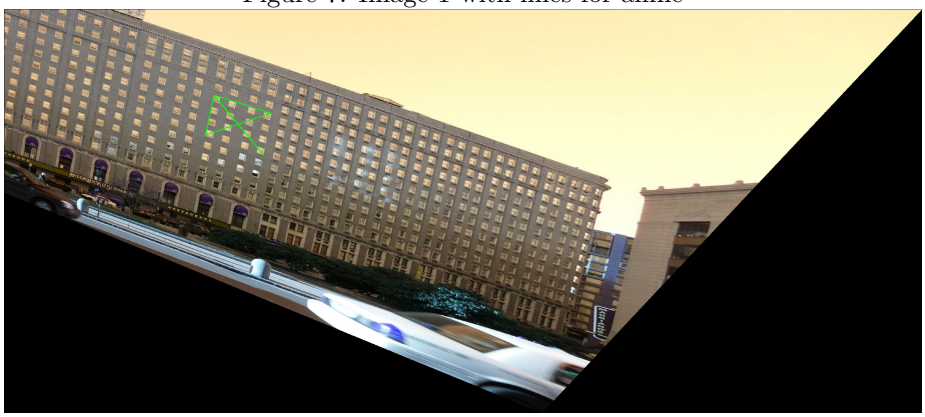

Figure 8: Image 1 with lines

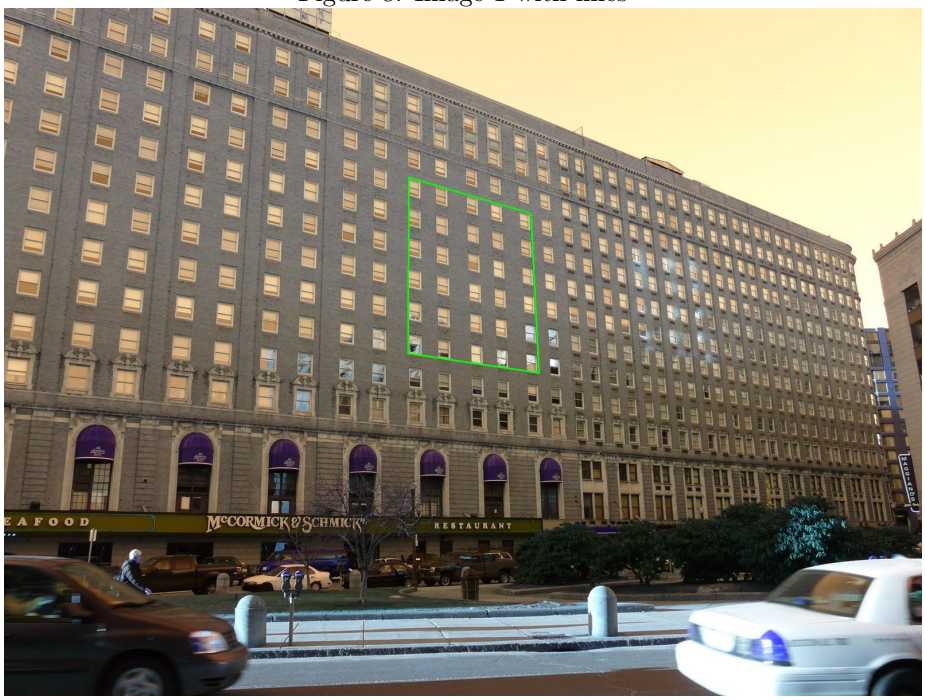

Figure 9: Image 1 with line projective only

Figure 10: Image 2 with lines for affine

Figure 11: Image 2 with lines

Figure 12: Image 2 with line projective only

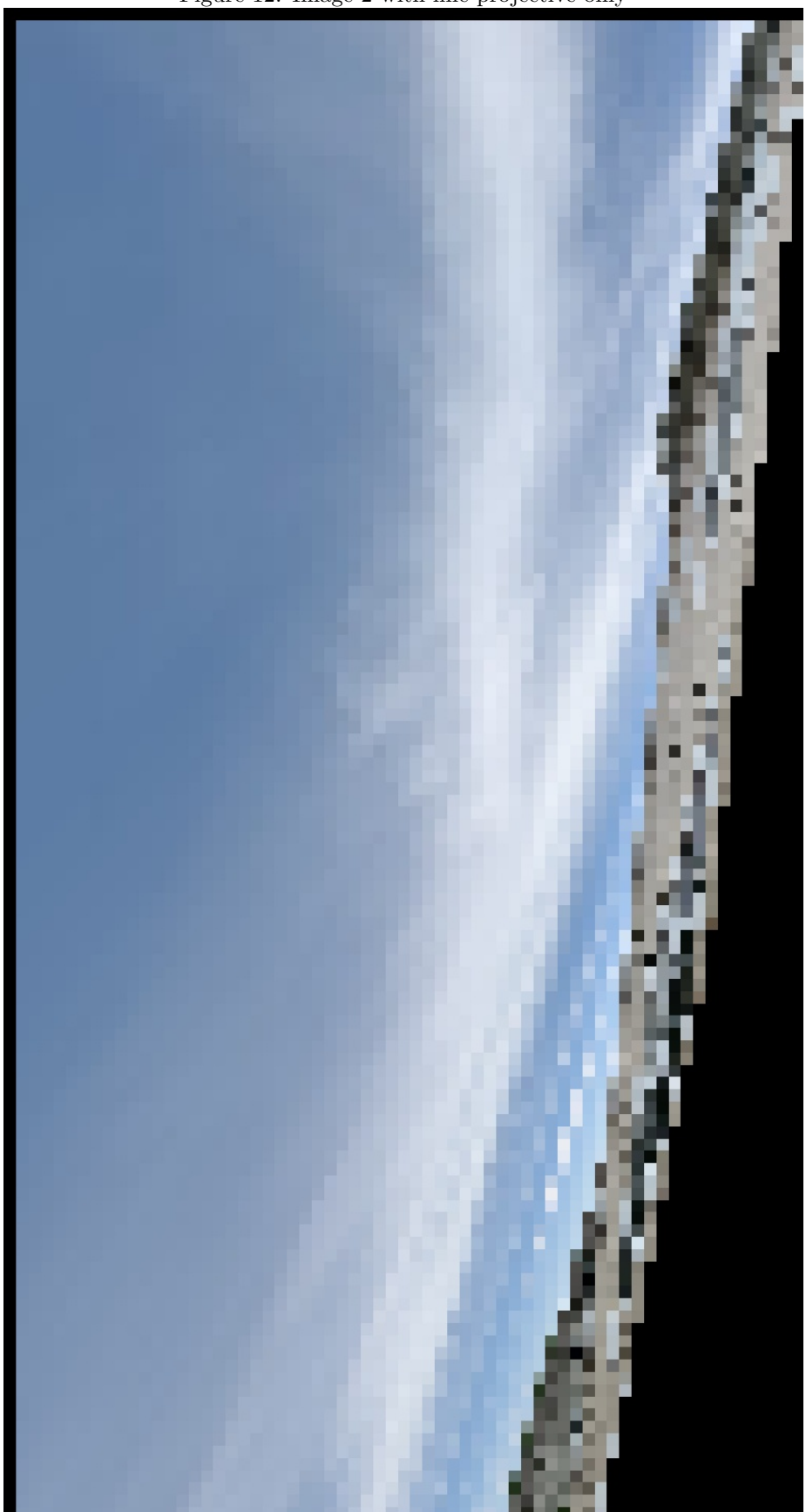

Figure 13: Image 3 with lines for affine

Figure 14: Image 3 with lines

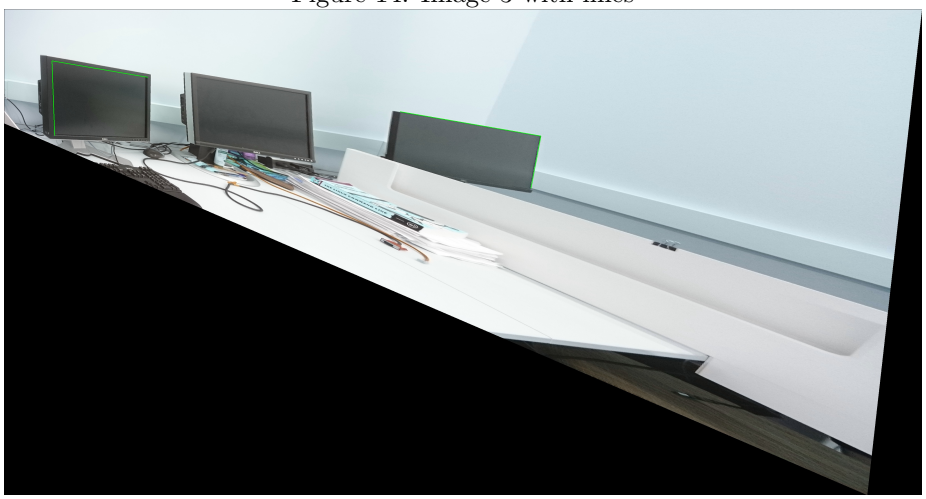

Figure 15: Image 3 with line projective only

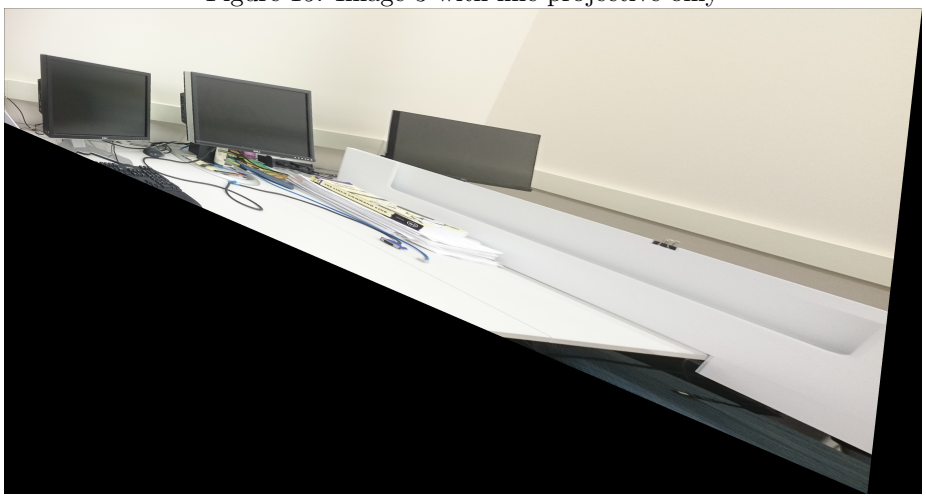

Figure 16: Image 1 one step

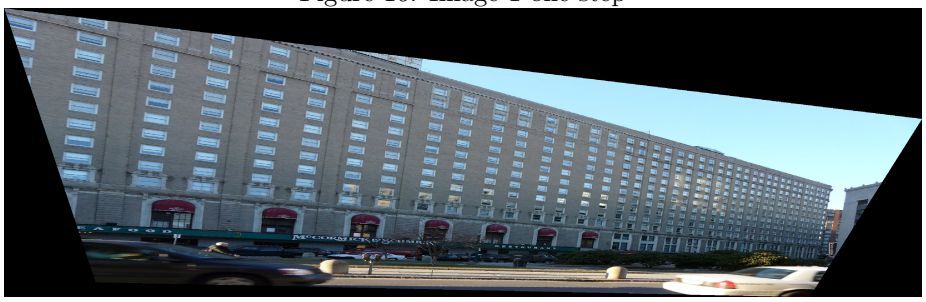

Figure 17: Image 2 one step

Figure 18: Image 3 one step

Source Code

Task1

```
# -*- coding: utf-8 -*-
"""
Created on Mon Sep 15 14:40:27 2020

@author: risij
"""

import cv2
import numpy as np
import os
from math import ceil,floor
import pdb

import time
start_time = time.time()
np.set_printoptions(suppress=True)


class Homography:
    def __init__(self,undistorted_image,distorted_image,Undist_points=None,Dist_points=None):
        '''
         Loading Images
        '''
        self.img_undist = cv2.imread(os.path.join(path,undistorted_image))
        self.img_dist = cv2.imread(os.path.join(path,distorted_image))
        self.Width=self.img_undist.shape[1]
        self.Height=self.img_undist.shape[0]

        '''
        If ROI is not provided in constructor then using full image as ROI
        '''
```

```
    if(type(Dist_points)==np.ndarray):
        self.Dist_points=Dist_points
    else:
        self.Dist_points = np.array([[0,0],[0,self.Height],[self.Width,self.Height],[self.Width,0]])

    if(type(Undist_points)==np.ndarray):
        self.Undist_points=Undist_points
    else:
        self.Undist_points = np.array([[0,0],[0,self.img_undist.shape[0]],[self.img_undist.shape[1],self.img_undist.shape[0]],[self.img_undist.shape[1],0]])


    self.undistorted_image=undistorted_image
    self.distorted_image=distorted_image


def generate_homography(self):
    '''
    Creating Homography matrices for all 4 points P,Q,R,S and X_prime vector
    '''
    self.H = np.zeros((8,8))
    self.X_prime = np.zeros((8,1))

    for i in range(len(self.Dist_points)):
        self.H[2*i]   = np.array([self.Undist_points[i][0],self.Undist_points[i][1],1,0,0,0,-self.Undist_points[i][0]*self.Dist_points[i][0],-self.Undist_points[i][1]*se
        self.H[2*i+1] =np.array([0,0,0,self.Undist_points[i][0],self.Undist_points[i][1],1,-self.Undist_points[i][0]*self.Dist_points[i][1],-self.Undist_points[i][1]*se
        self.X_prime[2*i]=self.Dist_points[i][0]
        self.X_prime[2*i+1]=self.Dist_points[i][1]

    '''
    Solving system of linear equations
    '''
    self.h = np.linalg.solve(self.H,self.X_prime)
    self.h = np.reshape(np.vstack((self.h,1)),(3,3))

def projective_homography(self,P):
    '''
```

```
    calculating projective homography
    '''
    L1 = np.cross(P[0],P[3])
    L1 = L1/L1[2]
    L2 = np.cross(P[1],P[2])
    L2 = L2/L2[2]

    VP1 = np.cross(L1,L2)
    VP1= VP1/VP1[2]

    L3 = np.cross(P[0],P[1])
    L3 = L3/L3[2]
    L4 = np.cross(P[2],P[3])
    L4 = L4/L4[2]

    VP2 = np.cross(L3,L4)
    VP2 = VP2/VP2[2]

    VL = np.cross(VP1,VP2)
    VL = VL/VL[2]
    self.h = np.linalg.inv(np.array([[1,0,0],[0,1,0],[VL[0],VL[1],VL[2]]]))

def affine_homography(self,Q):
    '''
    calculating affine homography
    '''
    l1_o = np.cross(Q[0],Q[1])
    m1_o = np.cross(Q[0],Q[2])
    l1_o = l1_o/l1_o[2]
    m1_o = m1_o/m1_o[2]


    l2_o = np.cross(Q[3],Q[4])
    m2_o = np.cross(Q[3],Q[5])
    l2_o = l2_o/l2_o[2]
    m2_o = m2_o/m2_o[2]
```

```
    A = np.vstack(([l1_o[0]*m1_o[0],l1_o[0]*m1_o[1]+l1_o[1]*m1_o[0]],[l2_o[0]*m2_o[0],l2_o[0]*m2_o[1]+l2_o[1]*m2_o[0]]))

    b = np.array([-l1_o[1]*m1_o[1],-l2_o[1]*m2_o[1]])

    temp = np.linalg.solve(A,b)
    S = np.identity(2)

    S[0,0] = temp[0]
    S[1,0] = temp[1]
    S[0,1] = temp[1]

    svd_decomp = np.linalg.svd(S)
    D = np.diag(np.sqrt(svd_decomp[1]))

    A = np.matmul(np.matmul(svd_decomp[0],D),svd_decomp[0].T)

    H_A = np.identity(3)

    H_A[0:2,0:2] = A
    self.h =np.linalg.inv(H_A)


def one_step_homography(self,point_dict):
    l1 = np.cross(point_dict["P"],point_dict["Q"])
    m1 = np.cross(point_dict["Q"],point_dict["R"])
    l1 = l1/l1[2]
    m1 = m1/m1[2]


    l2 = np.cross(point_dict["P"],point_dict["Q"])
    m2 = np.cross(point_dict["P"],point_dict["S"])
    l2 = l2/l2[2]
    m2 = m2/m2[2]
```

```
l3 = np.cross(point_dict["S"],point_dict["R"])
m3 = np.cross(point_dict["Q"],point_dict["R"])
l3 = l3/l3[2]
m3 = m3/m3[2]

l4 = np.cross(point_dict["A"],point_dict["B"])
m4 = np.cross(point_dict["A"],point_dict["D"])
l4 = l4/l4[2]
m4 = m4/m4[2]

l5 = np.cross(point_dict["C"],point_dict["D"])
m5 = np.cross(point_dict["B"],point_dict["C"])
l4 = l4/l4[2]
m4 = m4/m4[2]


A = np.identity(5)
b = np.zeros((5,1))
L = [[l1,m1],[l2,m2],[l3,m3],[l4,m4],[l5,m5]]
for i in range(5):
    A[i,:] = [L[i][0][0]*L[i][1][0],(L[i][0][1]*L[i][1][0]+L[i][0][0]*L[i][1][1])/2,L[i][0][1]*L[i][1][1], (L[i][0][0]*L[i][1][2]+L[i][0][2]*L[i][1][0])/2, (L[i][0]
    b[i] = -L[i][0][2]*L[i][1][2]


temp = np.linalg.solve(A,b)
temp = temp/np.max(temp)

S = np.zeros((2,2))
S[0,0] = temp[0]
S[1,0] = temp[1]/2
S[0,1] = temp[1]/2
S[1,1] = temp[2]

svd_decomp = np.linalg.svd(S)
D = np.diag(np.sqrt(svd_decomp[1]))
A = np.matmul(np.matmul(svd_decomp[0],D),svd_decomp[0].T)
```

```
        H = np.identity(3)

        H[0:2,0:2] = A
        H[2,0:2] = np.linalg.solve(A,temp[3:5]/2).T
        self.h = H


def project_homography(homography,method_name):
    coordinate_r3 = np.matmul(np.linalg.inv(homography.h),np.array([[0,homography.img_dist.shape[1],homography.img_dist.shape[1],0],[0,0,homography.img_dist.shape[0],homogra
    coordinate_r2 = coordinate_r3[0:2]/coordinate_r3[2]

    scaling_width = homography.img_dist.shape[1]/(ceil(np.max(coordinate_r2[0][:]))-floor(np.min(coordinate_r2[0][:])))
    scaling_height = homography.img_dist.shape[0]/(ceil(np.max(coordinate_r2[1][:]))-floor(np.min(coordinate_r2[1][:])))

    scale = max(scaling_width,scaling_height)

    image_width = np.round((ceil(np.max(coordinate_r2[0][:]))-floor(np.min(coordinate_r2[0][:])))*scale).astype(int)
    image_height = np.round((ceil(np.max(coordinate_r2[1][:]))-floor(np.min(coordinate_r2[1][:])))*scale).astype(int)

    xmin = int(floor(np.min(coordinate_r2[0][:])))
    ymin = int(floor(np.min(coordinate_r2[1][:])))

    N = np.zeros((image_height,image_width,3))
    y=np.vstack([np.vstack(np.meshgrid(range(N.shape[0]),range(N.shape[1]))[::-1]).reshape(2,-1),[1]*(N.shape[0]*N.shape[1])])
    mesh_y = np.copy(y)
    y[0] = y[0]/scale+xmin
    y[1] = y[1]/scale+ymin

    z = np.matmul(homography.h,y)
    z = np.round(z[0:2,:]/z[2],4)
    y=mesh_y
    z=np.vstack((z,y))
    z=z.T

    z=z[(z[:,0]>0) & (z[:,0]<homography.img_dist.shape[1])]
```

```
    z=z[(z[:,1]>0) & (z[:,1]<homography.img_dist.shape[0])]
    z = z.astype(int)


    for item in z:
            N[item[3],item[2]] = homography.img_dist[item[1],item[0]]


    file_name = os.path.basename(homography.distorted_image).split(".")[0]+method_name+"."+os.path.basename(homography.distorted_image).split(".")[1]
    cv2.imwrite(os.path.join(path,file_name),N)
    return N




path = "hw3_Task1_Images"
######################### Task 1 part 1 ########################################
print("Task 1 Part 1")
Undist_points_A = np.array([[0,0],[75,0],[75,807.5],[0,807.5]])
Dist_points_A = np.array([[643,497],[667,502],[667,538],[643,535]])


Undist_points_A = np.array([[0,0],[0,1135],[1934,1135],[1934,0]])
Dist_points_A = np.array([[177,77],[117,586],[879,681],[865,297]])

homography = Homography("Measurements/Img1.jpg","images/Img1.jpg",Undist_points=Undist_points_A,Dist_points=Dist_points_A)
homography.generate_homography()

project_homography(homography,"point_to_point")

painting = cv2.imread(homography.distorted_image)
painting = cv2.cvtColor(painting,cv2.COLOR_BGR2RGB)
i=0
for point in Dist_points_A:
```

```
    painting = cv2.putText(painting,chr(80+i),(point[0]+20,point[1]-20), cv2.FONT_HERSHEY_SIMPLEX,1,(255,0,0),2)
    i+=1


cv2.imwrite(os.path.join(path,"points_on_img1.jpg"),painting)

########################################Task 1 part 2 ################################################################
print("Task 1 Part 2")

Undist_points_A = np.array([[0,0],[84*2,0],[84*2,74*2],[0,74*2]])
Dist_points_A = np.array([[382,576],[594,551],[608,927],[378,837]])

homography = Homography("Measurements/Img2.jpg","images/Img2.jpeg",Undist_points=Undist_points_A,Dist_points=Dist_points_A)
homography.generate_homography()
project_homography(homography,"point_to_point")


painting = cv2.imread(homography.distorted_image)
painting = cv2.cvtColor(painting,cv2.COLOR_BGR2RGB)
i=0
for point in Dist_points_A:
    painting = cv2.putText(painting,chr(80+i),(point[0]+20,point[1]-20), cv2.FONT_HERSHEY_SIMPLEX,1,(255,0,0),2)
    i+=1

cv2.imwrite(os.path.join(path,"points_on_img2.jpg"),painting)

#
#############################################################################################################
#
########################################Task 1 part 3 ################################################################
#
print("Task 1 Part 3")
Undist_points_A = np.array([[0,0],[55,0],[55,36],[0,36]])
Dist_points_A = np.array([[2057,697],[2667,717],[2695,1331],[2089,1477]])
```

```
homography = Homography("Measurements/Img3.jpg","images/Img3.jpg",Undist_points=Undist_points_A,Dist_points=Dist_points_A)
homography.generate_homography()

project_homography(homography,"point_to_point")

painting = cv2.imread(homography.distorted_image)
painting = cv2.cvtColor(painting,cv2.COLOR_BGR2RGB)
i=0
for point in Dist_points_A:
    painting = cv2.putText(painting,chr(80+i),(point[0]+20,point[1]-20), cv2.FONT_HERSHEY_SIMPLEX,1,(255,0,0),2)
    i+=1

cv2.imwrite(os.path.join(path,"points_on_img3.jpg"),painting)

#######################################Task 1 part 4 ###################################################################

print("Task 1 Part 4")
Undist_points_A = np.array([[0,0],[35,0],[35,23],[0,23]])
Dist_points_A = np.array([[223,288],[491,230],[478,695],[214,566]])


homography = Homography("images/board.jpg","images/board.jpg",Undist_points=Undist_points_A,Dist_points=Dist_points_A)
homography.generate_homography()

project_homography(homography,"point_to_point")

painting = cv2.imread(os.path.join(path,homography.distorted_image))
painting = cv2.cvtColor(painting,cv2.COLOR_BGR2RGB)
i=0
for point in Dist_points_A:
    painting = cv2.putText(painting,chr(80+i),(point[0]+20,point[1]-20), cv2.FONT_HERSHEY_SIMPLEX,1,(255,0,0),2)
    i+=1

cv2.imwrite(os.path.join(path,"points_on_board.jpg"),painting)

#######################################Task 1 part 5 ###################################################################
```

```
print("Task 1 Part 5")
Undist_points_A = np.array([[0,0],[11.4,0],[11.4,17.9],[0,17.9]])
Dist_points_A = np.array([[643,167],[958,85],[953,689],[633,600]])


homography = Homography("images/board.jpg","images/board.jpg",Undist_points=Undist_points_A,Dist_points=Dist_points_A)
homography.generate_homography()

project_homography(homography,"point_to_point_2")

painting = cv2.imread(os.path.join(path,homography.distorted_image))
painting = cv2.cvtColor(painting,cv2.COLOR_BGR2RGB)
i=0
for point in Dist_points_A:
    painting = cv2.putText(painting,chr(80+i),(point[0]+20,point[1]-20), cv2.FONT_HERSHEY_SIMPLEX,1,(255,0,0),2)
    i+=1

cv2.imwrite(os.path.join(path,"points_on_board.jpg"),painting)

####################################Task 2 part 1#################################################################

print("Task 2 Part 1")

P = np.array([[705,295,1],[918,358,1],[932,636,1],[704,602,1]])


homography = Homography("Measurements/Img1.jpg","images/Img1.jpg")
homography.projective_homography(P)
project_homography(homography,"_two_step_projective_only")


painting = cv2.imread(homography.distorted_image)
painting = cv2.cvtColor(painting,cv2.COLOR_BGR2RGB)

for line in [(P[0],P[3]),[P[1],P[2]],[P[0],P[1]],[P[2],P[3]]]:
```

```
    cv2.line(painting,tuple(line[0][0:2]),tuple(line[1][0:2]),(0, 255, 0),2)

cv2.imwrite(os.path.join(path,"lines_on_image_1_for_projective.jpg"),painting)


Q = np.array([[624,255,1],[794,309,1],[771,433,1],[599,373,1]])

homography = Homography("Measurements/Img3.jpg","Img1two_step_projective_only.jpg")
homography.affine_homography(Q)
project_homography(homography,"_two_step_after_Affine")


painting = cv2.imread(homography.distorted_image)
painting = cv2.cvtColor(painting,cv2.COLOR_BGR2RGB)

for line in [(Q[0],Q[1]),[Q[0],Q[3]],[Q[0],Q[2]],[Q[1],Q[3]]]:
    cv2.line(painting,tuple(line[0][0:2]),tuple(line[1][0:2]),(0, 255, 0),2)

cv2.imwrite(os.path.join(path,"lines_on_image_1_after_projective_for_affine.jpg"),painting)




#####################################Task 2 part 2 ##################################################################
print("Task 2 Part 2")

P = np.array([[381,576,1],[595,551,1],[609,923,1],[379,836,1]])


homography = Homography("Measurements/Img2.jpg","images/Img2.jpeg")
homography.projective_homography(P)
project_homography(homography,"_two_step_projective_only")


painting = cv2.imread(homography.distorted_image)
painting = cv2.cvtColor(painting,cv2.COLOR_BGR2RGB)
```

```
for line in [(P[0],P[3]),[P[1],P[2]],[P[0],P[1]],[P[2],P[3]]]:
    cv2.line(painting,tuple(line[0][0:2]),tuple(line[1][0:2]),(0, 255, 0),2)

cv2.imwrite(os.path.join(path,"lines_on_image_2_for_projective.jpg"),painting)



Q = np.array([[446,225,1],[551,250,1],[548,339,1],[461,319,1]])

homography = Homography("Measurements/Img3.jpg","Img2_two_step_projective_only.jpeg")
homography.affine_homography(Q)
project_homography(homography,"_two_step_after_Affine")


painting = cv2.imread(homography.distorted_image)
painting = cv2.cvtColor(painting,cv2.COLOR_BGR2RGB)

for line in [(Q[0],Q[1]),[Q[0],Q[3]],[Q[0],Q[2]],[Q[1],Q[3]]]:
    cv2.line(painting,tuple(line[0][0:2]),tuple(line[1][0:2]),(0, 255, 0),2)

cv2.imwrite(os.path.join(path,"lines_on_image_1_after_projective_for_affine.jpg"),painting)



#######################################Task 2 part 3 ###################################################################
print("Task 2 Part 3")

P = np.array([[2092,742,1],[2653,748,1],[2676,1304,1],[2122,1432,1]])



homography = Homography("Measurements/Img3.jpg","images/Img3.jpg")
homography.projective_homography(P)
N = project_homography(homography,"two_step_projective_only")
```

```
painting = cv2.imread(homography.distorted_image)
painting = cv2.cvtColor(painting,cv2.COLOR_BGR2RGB)

for line in [(P[0],P[3]),[P[1],P[2]],[P[0],P[1]],[P[2],P[3]]]:
    cv2.line(painting,tuple(line[0][0:2]),tuple(line[1][0:2]),(0, 255, 0),2)

cv2.imwrite(os.path.join(path,"lines_on_image_3_for_projective.jpg"),painting)



Q = np.array([[213,230,1],[639,303,1],[225,562,1],[2388,564,1],[1762,456,1],[2344,796,1]])

homography = Homography("Measurements/Img3.jpg","Img3two_step_projective_only.jpg")
homography.affine_homography(Q)
project_homography(homography,"two_step_after_Affine")

painting = cv2.imread(os.path.join(path,homography.distorted_image))
painting = cv2.cvtColor(painting,cv2.COLOR_BGR2RGB)

for line in [(Q[0],Q[1]),[Q[0],Q[2]],[Q[3],Q[4]],[Q[3],Q[5]]]:
    cv2.line(painting,tuple(line[0][0:2]),tuple(line[1][0:2]),(0, 255, 0),2)

cv2.imwrite(os.path.join(path,"lines_on_image_3_after_projective_for_affine.jpg"),painting)
#pdb.set_trace()


#####################################Task 2 part 4 ##################################################################
print("Task 2 Part 4")

P = np.array([[223,288,1],[491,230,1],[478,695,1],[214,566,1]])


homography = Homography("images/board.jpg","images/board.jpg")
homography.projective_homography(P)
```

```
N = project_homography(homography,"two_step_projective_only")


painting = cv2.imread(os.path.join(path,homography.distorted_image))
painting = cv2.cvtColor(painting,cv2.COLOR_BGR2RGB)

for line in [(P[0],P[3]),[P[1],P[2]],[P[0],P[1]],[P[2],P[3]]]:
    cv2.line(painting,tuple(line[0][0:2]),tuple(line[1][0:2]),(0, 255, 0),2)

cv2.imwrite(os.path.join(path,"lines_on_board_for_projective.jpg"),painting)


Q = np.array([[213,230,1],[639,303,1],[225,562,1],[2388,564,1],[1762,456,1],[2344,796,1]])

homography = Homography("images/board.jpg","boardtwo_step_projective_only.jpg")
homography.affine_homography(Q)
project_homography(homography,"two_step_after_Affine")

painting = cv2.imread(os.path.join(path,homography.distorted_image))
painting = cv2.cvtColor(painting,cv2.COLOR_BGR2RGB)

for line in [(Q[0],Q[1]),[Q[0],Q[2]],[Q[3],Q[4]],[Q[3],Q[5]]]:
    cv2.line(painting,tuple(line[0][0:2]),tuple(line[1][0:2]),(0, 255, 0),2)

cv2.imwrite(os.path.join(path,"lines_on_board_after_projective_for_affine.jpg"),painting)


#####################################Task 2 part 5 ##################################################################
print("Task 2 Part 5")

P = np.array([[285,386,1],[938,365,1],[957,835,1],[366,1041,1]])


homography = Homography("images/monitor.jpg","images/monitor.jpg")
homography.projective_homography(P)
N = project_homography(homography,"two_step_projective_only")
```

```
painting = cv2.imread(os.path.join(path,homography.distorted_image))
painting = cv2.cvtColor(painting,cv2.COLOR_BGR2RGB)

for line in [(P[0],P[3]),[P[1],P[2]],[P[0],P[1]],[P[2],P[3]]]:
    cv2.line(painting,tuple(line[0][0:2]),tuple(line[1][0:2]),(0, 255, 0),2)

cv2.imwrite(os.path.join(path,"lines_on_monitor_for_projective.jpg"),painting)


Q = np.array([[95,129,1],[443,170,1],[484,421,1],[522,275,1],[545,434,1],[900,322]])

homography = Homography("images/monitor.jpg","monitortwo_step_projective_only.jpg")
homography.affine_homography(Q)
project_homography(homography,"two_step_after_Affine")

painting = cv2.imread(os.path.join(path,homography.distorted_image))
painting = cv2.cvtColor(painting,cv2.COLOR_BGR2RGB)

for line in [(Q[0],Q[1]),[Q[0],Q[2]],[Q[3],Q[4]],[Q[3],Q[5]]]:
    cv2.line(painting,tuple(line[0][0:2]),tuple(line[1][0:2]),(0, 255, 0),2)

cv2.imwrite(os.path.join(path,"lines_on_monitor_after_projective_for_affine.jpg"),painting)


#####################################Task 3 part 1 ################################################################
print("Task 3 Part 1")
point_dict = {"P":[592,264,1],"Q":[827,331,1],"R":[834,568,1],"S":[586,522,1],"A":[562,196,1],"B":[975,330,1],"C":[984,497,1],"D":[556,399,1]}

homography = Homography("Measurements/Img1.jpg","images/Img1.jpg")
homography.one_step_homography(point_dict)
project_homography(homography,"one_step")


painting = cv2.imread(os.path.join(path,homography.distorted_image))
```

```
painting = cv2.cvtColor(painting,cv2.COLOR_BGR2RGB)


for line in [(point_dict["P"],point_dict["Q"]),(point_dict["Q"],point_dict["R"]),(point_dict["P"],point_dict["S"]),(point_dict["S"],point_dict["R"]),(point_dict["A"],point_
    cv2.line(painting,tuple(line[0][0:2]),tuple(line[1][0:2]),(0, 255, 0),2)

cv2.imwrite(os.path.join(path,"lines_on_image_1_onestep.jpg"),painting)

####################################Task 3 part 2 ###################################################################

print("Task 3 Part 2")

point_dict = {"P":[382,577,1],"Q":[595,554,1],"R":[607,925,1],"S":[380,836,1],"A":[426,119,1],"B":[512,39,1],"C":[516,296,1],"D":[425,345,1]}



homography = Homography("Measurements/Img2.jpg","images/Img2.jpeg")
homography.one_step_homography(point_dict)
project_homography(homography,"one_step")


painting = cv2.imread(os.path.join(path,homography.distorted_image))
painting = cv2.cvtColor(painting,cv2.COLOR_BGR2RGB)


for line in [(point_dict["P"],point_dict["Q"]),(point_dict["Q"],point_dict["R"]),(point_dict["P"],point_dict["S"]),(point_dict["S"],point_dict["R"]),(point_dict["A"],point_
    cv2.line(painting,tuple(line[0][0:2]),tuple(line[1][0:2]),(0, 255, 0),2)

cv2.imwrite(os.path.join(path,"lines_on_image_2_onestep.jpg"),painting)

####################################Task 3 part 3 ###################################################################
print("Task 3 Part 3")

point_dict = {"P":[759,826,1],"Q":[1743,829,1],"R":[1752,1579,1],"S":[798,1998,1],"A":[2097,741,1],"B":[2656,751,1],"C":[2676,1300,1],"D":[2125,1428,1]}
```

```
homography = Homography("Measurements/Img3.jpg","images/Img3.jpg")
homography.one_step_homography(point_dict)
project_homography(homography,"one_step")


painting = cv2.imread(os.path.join(path,homography.distorted_image))
painting = cv2.cvtColor(painting,cv2.COLOR_BGR2RGB)


for line in [(point_dict["P"],point_dict["Q"]),(point_dict["Q"],point_dict["R"]),(point_dict["P"],point_dict["S"]),(point_dict["S"],point_dict["R"]),(point_dict["A"],point_
    cv2.line(painting,tuple(line[0][0:2]),tuple(line[1][0:2]),(0, 255, 0),2)

cv2.imwrite(os.path.join(path,"lines_on_image_3_onestep.jpg"),painting)


#####################################Task 3 part 4 ##################################################################
print("Task 3 Part 4")

point_dict = {"P":[1866,760,1],"Q":[2929,479,1],"R":[2908,2757,1],"S":[1828,2458,1],"A":[362,1228,1],"B":[1314,997,1],"C":[1265,2854,1],"D":[320,2336,1]}



homography = Homography("images/board.jpg","images/board.jpg")
homography.one_step_homography(point_dict)
project_homography(homography,"one_step")


painting = cv2.imread(os.path.join(path,homography.distorted_image))
painting = cv2.cvtColor(painting,cv2.COLOR_BGR2RGB)


for line in [(point_dict["P"],point_dict["Q"]),(point_dict["Q"],point_dict["R"]),(point_dict["P"],point_dict["S"]),(point_dict["S"],point_dict["R"]),(point_dict["A"],point_
    cv2.line(painting,tuple(line[0][0:2]),tuple(line[1][0:2]),(0, 255, 0),2)

cv2.imwrite(os.path.join(path,"lines_on_board_onestep.jpg"),painting)
```

```
#####################################Task 3 part 5 ###############################################################


print("Task 3 Part 5")

point_dict = {"P":[225,288,1],"Q":[489,231,1],"R":[477,698,1],"S":[215,565,1],"A":[643,167],"B":[958,85,1],"C":[953,689,1],"D":[633,600,1]}



homography = Homography("images/board.jpg","images/board.jpg")
homography.one_step_homography(point_dict)
project_homography(homography,"one_step")


painting = cv2.imread(os.path.join(path,homography.distorted_image))
painting = cv2.cvtColor(painting,cv2.COLOR_BGR2RGB)


for line in [(point_dict["P"],point_dict["Q"]),(point_dict["Q"],point_dict["R"]),(point_dict["P"],point_dict["S"]),(point_dict["S"],point_dict["R"]),(point_dict["A"],point_
    cv2.line(painting,tuple(line[0][0:2]),tuple(line[1][0:2]),(0, 255, 0),2)

cv2.imwrite(os.path.join(path,"lines_on_board_onestep.jpg"),painting)

#####################################Task Extra  ###############################################################
```