FALL 22 ECE 661 – COMPUTER VISION

Homework 3

Sahithi Kodali – 34789866
kodali1@purdue.edu

TASK – 1:

Two images shown below are provided to perform metric rectification i.e. removing projective and affine distortion from the images.

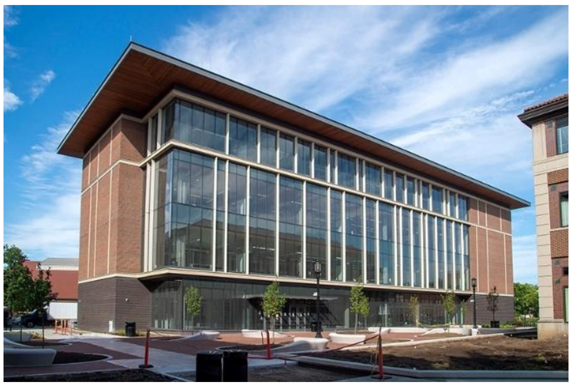
Image 1

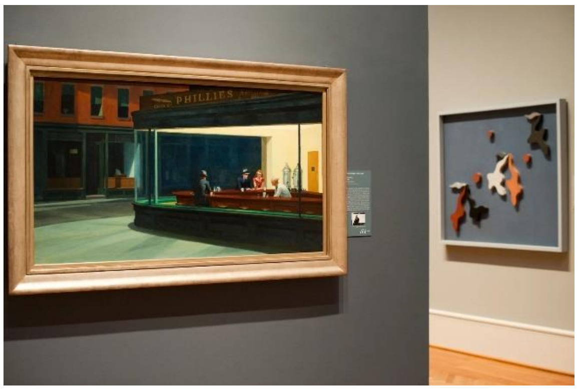

Image 2

Three methods are used to perform this task which are further detailed in this report.

1.1 Point-to-point Correspondence

The main method used in this task is similar to the concept in Homework 2. The main concept previously used in HW2 is discussed here again.

Main Concept:

To perform such point-to-point correspondence methods, we must understand the concept of Homography. To project an image onto a world image frame, we must find the relationship between the corresponding points on these image frames. If the homogeneous representation of a point in the domain space/image plane and the projective space/world plane are $\vec{x}\ (x, y, 1)$ and $\vec{x}'(x', y', 1)$ respectively, then the relationship between these two points is represented as,

$$\vec{x}' = H\vec{x}$$

where, H is a non-singular 3x3 matrix known as Homography matrix and can be written as,

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

Since, Homography matrix is a linear mapping of homogeneous coordinates we only consider the ratios of the coordinates i.e., if point (x_1, x_2, x_3) exists then it is same as point (x_1/x_3, x_2/x_3,1) which is as in the representation of the points $\vec{x}$ and $\vec{x}'$. Hence, the coordinate $H_{(3,3)} = i = 1$.

From above,

$$\vec{x}' = H\vec{x}$$

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

$$x' = \mathrm{ax + by + c}$$

$$y' = \mathrm{dx + ey + f}$$

$$1 = \mathrm{gx + hy + 1}$$

Thus, the x' and y' coordinates can be written as,

$$x' = \frac{ax+by+c}{gx+hy+1} => gxx' + hyx' + x' = ax + by + c => x' = ax + by + c - gxx' - hyx'$$

$$y' = \frac{dx+ey+f}{gx+hy+1} => gxy' + hyy' + y' = dx + ey + f => y' = dx + ey + f - gxy' - hyy'$$

The equations of x' and y' contains 8 unknowns, hence, at least 4 points giving 8 equations are required to determine these unknowns. The matrix multiplication of the equations using four points can be written as,

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

Hence, the Homography matrix can be computed as $H = A^{-1}B$

Weighted Pixel values concept:

Weighted pixel values is required to obtain an image that is free of blank pixels. After mapping the world plane and image plane coordinates, the coordinates obtained are usually in the float format. However, a float coordinate doesn't exist in the image plane and hence we have to find a weighted pixel value of the coordinates in the integral format by considering the pixel values around the coordinate. The four pixels surrounding a point (x`,y`) are taken as a combination of floor and ceil values i.e.,

P_1 = floor(x`), floor(y`)

P_2 = floor(x`), ceil(y`)

P_3 = ceil(x`), floor(y`)

P_4 = ceil(x`), ceil(y`)

The distance between these points and the mapped coordinate gives the weight (w). The weighted pixel value at the coordinate is determined from the pixel values (pv) corresponding to the above four near points and weights shown as below,

P_w (x`,y`) = (w_1*pv_1 + w_2*pv_2 + w_3*pv_3 + w_4*pv_4) / w_1 + w_2 +w_3 + w_4

To perform this task the following steps are followed,

i.) Find the pixel coordinates of the ROI i.e., chosen frame PQRS in the image (highlighted in Red in the images below). Let the world coordinates be the dimensions of the image i.e., the four corner points of the give image. The tool GIMP is used to determine the pixel coordinates of the images.

ii.) Determine the Homography matrix H = $A^{-1}B$ for the corresponding points of image and world planes.

iii.) From the Homography matrix obtained, the points are mapped from the image plane onto the world plane. So, we consider H^{-1} here i.e. A = $H^{-1}B$

iv.) From these mapped coordinates, we determine the minimum and maximum (x`, y`) values to determine a offset that can map the entire image onto the world coordinates frame which is also chosen based on the minimum and maximum (x`, y`).

v.) The mapped coordinates in float values are transformed to integral values using weighted pixels concept to pick the corresponding pixel near to the coordinate.

vi.) Replace each pixel coordinates in the world plane with its corresponding weighted pixel values to get the images free of distortion.

The resulting images obtained after performing these steps are shown below. We can observe that the projective and affine distortion are removed i.e the parallel lines and orthogonality between the lines are restored.

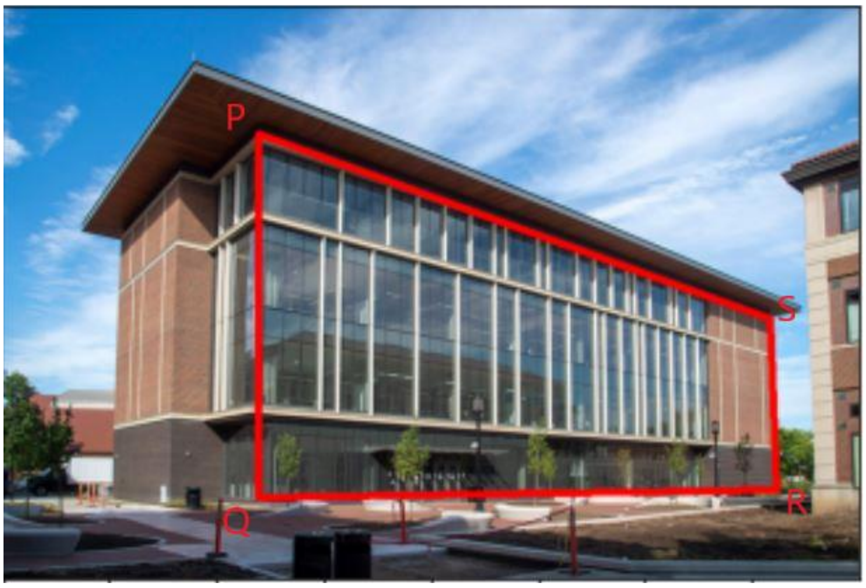

Figure 1 – Frame PQRS chosen in the Image 1

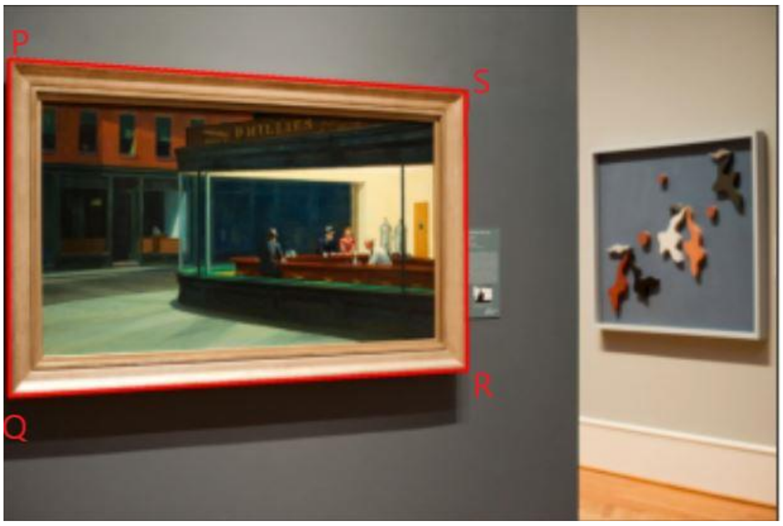

Figure 2 – Frame PQRS chosen in the Image 2

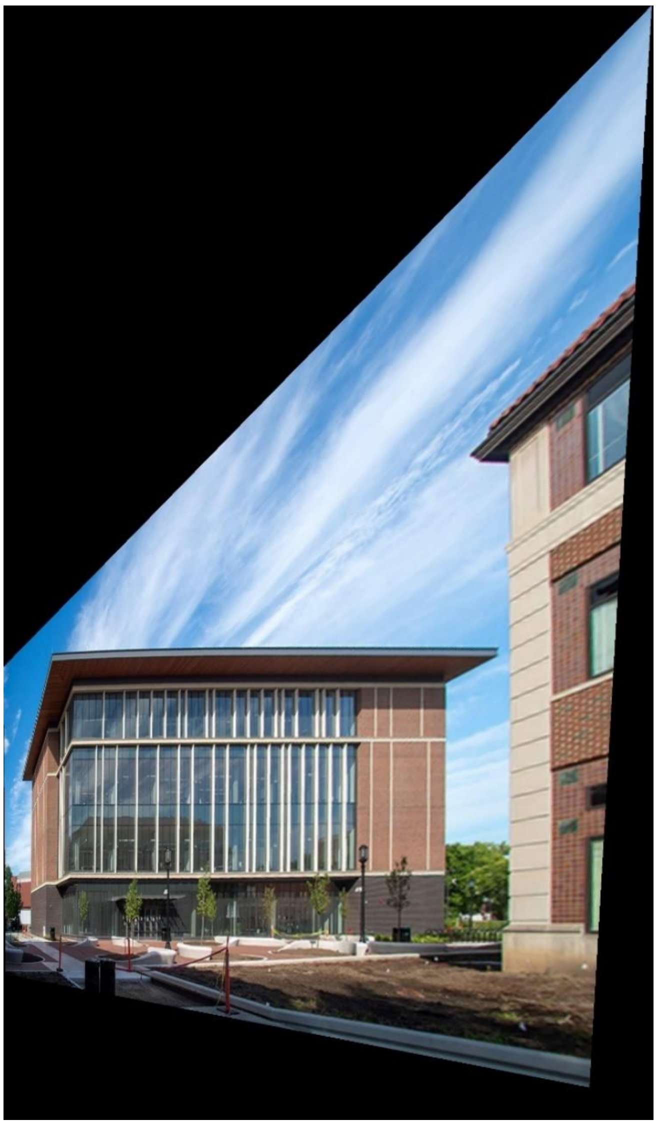

Figure 3 – Image obtained after performing point-to-point correspondence method to remove distortion in the Image 1

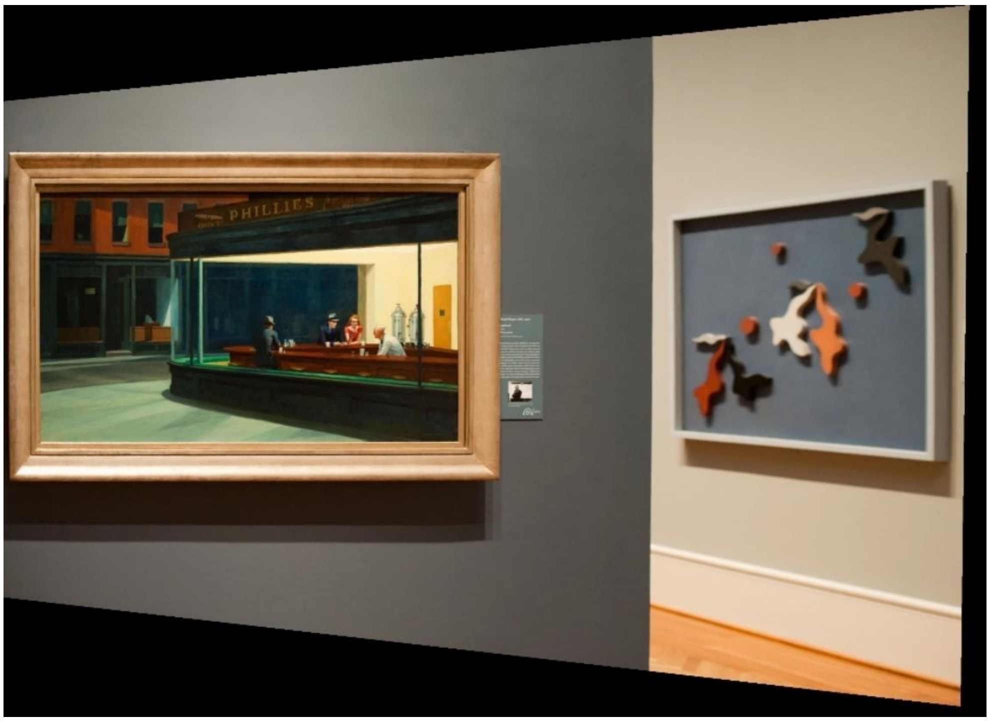

Figure 4 – Image obtained after performing point-to-point correspondence method to remove distortion in the Image 2

1.2 The Two-Step Method

This method involves two steps to remove distortion from a given image,

Step-1: Remove the projective distortion i.e., remove the converging lines to bring back the parallel lines that exist in the world plane.

Step-2: Remove the affine distortion i.e., remove the angles between the parallel lines to turn them into 90 degrees that exist in the world plane.

1.2.1 Removing Projective Distortion

Projective Distortion can be removed by mapping the vanishing line back the line at infinity i.e., l_{VL} -> l_{∞}.

To calculate the vanishing line, we need to find two pairs of parallel lines. In a projective distorted image, the lines that are parallel in world plane converge in the image plane giving a point of intersection known as vanishing points. We obtain two such points to calculate the vanishing line.

Say, line l1 is formed from points p1 and p2 and line l2 from points p3 and p4. The intersection of lines l1 and l2 gives a vanishing point vp1. Similarly, from two other lines l3 and l4 vp2 can be calculated. The intersection of points vp1 and vp2 gives the vanishing line as shown below,

$$l_1 = p_1 \text{ x } p_2$$

$$l_2 = p_3 \text{ x } p_4$$

$$l_3 = p_1` \times p_2`$$

$$l_4 = p_3` \times p_4`$$

$$vp_1 = l_1 \times l_2$$

$$vp_2 = l_3 \times l_4$$

$$l_{VL} = vp_1 \times vp_2$$

From the vanishing line, the Homography matrix H can be written as below,

$$H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ vl_1 & vl_2 & vl_3 \end{bmatrix} \text{; where the HC representation of line } l_{VL} = \begin{bmatrix} vl_1 \\ vl_2 \\ vl_3 \end{bmatrix}$$

From this H, we can further calculate the mapping coordinates and map the image from image plane to world plane obtaining an image free from projective distortion. Since the points are transformed by H and the lines are transformed by H^{-T}, we can prove this by solving $H^{-T}l_{VL}$ which will be equal to $l_\infty = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$.

The parallel lines chosen for this task are the opposite sides of the bounded rectangle PQRS of the input images shown in previous method, and the steps followed are same as in previous method.

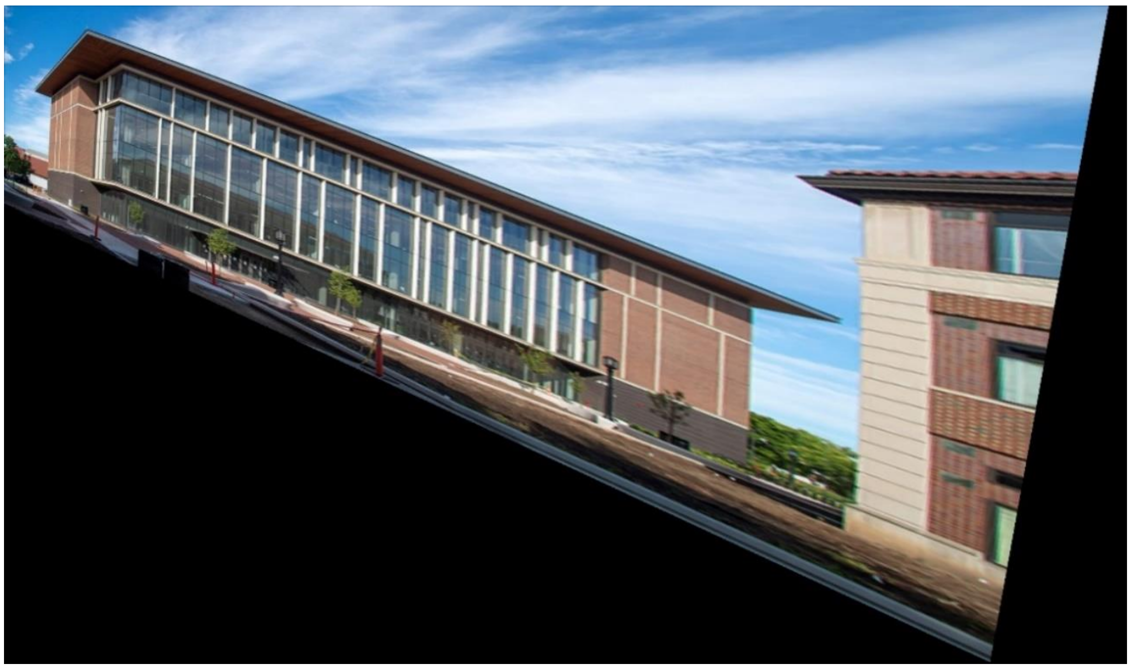

Figure 5 – Image obtained after removing projective distortion only in the Image 1

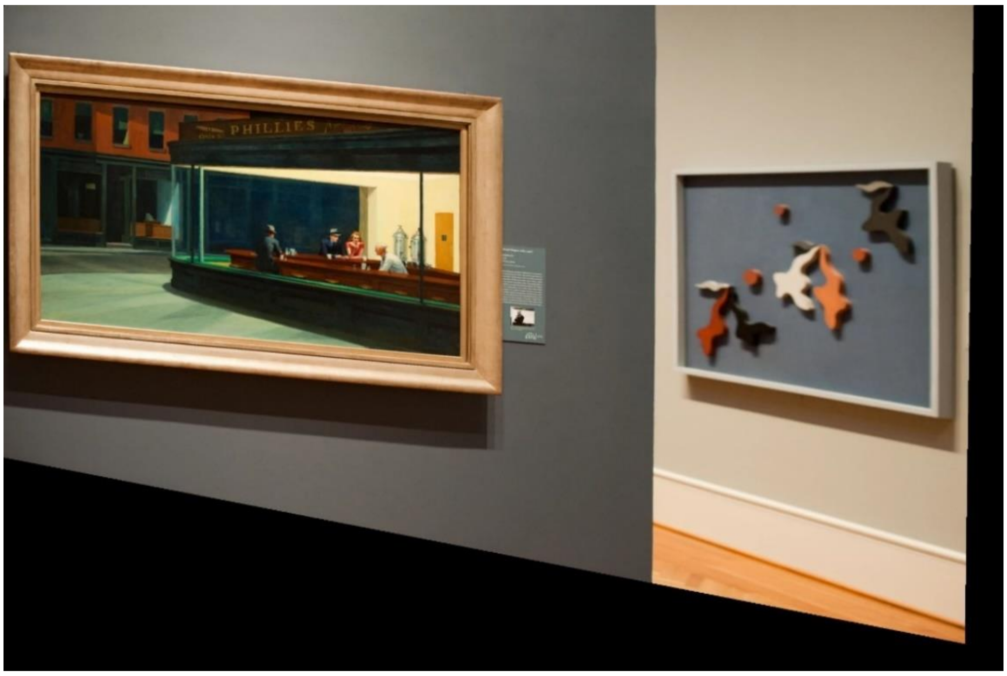

Figure 6 – Image obtained after removing projective distortion only in the Image 2

We can observe that the parallel lines are restored after removing the projective distortion, however the angles are still stretched or shrinked.

1.2.2 Removing Affine Distortion

Removing projective distortion from the image got rid of the converging lines and the parallel lines of the image are restored. However, these parallel lines are not orthogonal i.e., the angle between these lines is either shrinked or stretched. To bring back the orthogonality between the lines we must remove the affine distortion. This can be done by the cosine theta method given as,

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

We know that, for affine Homography H the conic is transformed as $C_\infty^{*\prime} = HC_\infty^*\, H^T$

To obtain Homography that brings orthogonality between the lines, we substitute $\cos\theta = 0$ since $\theta = 90°$, giving us the transformed when solved as below,

$$L^{T\prime}\, C_\infty^{*\prime}\, M' = 0 => L^{T'}\, HC_\infty^*\, H^T\, M' = 0$$

Where, H is Affine transformation = $\begin{bmatrix} A & t \\ 0^T & 1 \end{bmatrix}$ and solving further gives us,

$$(l_1{}' \quad l_2{}' \quad l_3{}') \begin{bmatrix} AA^T & 0 \\ 0^T & 0 \end{bmatrix} \begin{pmatrix} m_1{}' \\ m_2{}' \\ m_3{}' \end{pmatrix} = 0$$

Since AA^T is a symmetric matrix, let it be S = $\begin{bmatrix} S_{11} & S_{12} \\ S_{12} & S_{22} \end{bmatrix}$, giving us

$$(l_1' \quad l_2')\begin{bmatrix} S_{11} & S_{12} \\ S_{12} & S_{22} \end{bmatrix}\begin{pmatrix} m_1' \\ m_2' \end{pmatrix} = 0$$

$$=> S_{11}l_1'm_1' + S_{12}\,(\,l_1'm_2' \;+\; l_2'm_1'\,) + \;S_{22}l_2'm_2' \;=\; 0$$

Here, S_{22} = 0 since we only care about ratios and at least two equations are required to find the three unknowns. Hence, we choose two pairs of orthogonal lines from the image to determine the matrix S. Once we find S, we perform Singular Value Decomposition (SVD) of S to obtain A. Here A is a non-singular and assumed to be positive definite which can be proven.

A = $V\,D\,V^T$, solving S = AA^T = $V\,D^2\,V^T$ we get the values of D and further solving gives us the affine Homography matrix H = $\begin{bmatrix} A & 0 \\ 0^T & 1 \end{bmatrix}$.

To get rid of the affine distortion, we multiply the projective Homography with affine and apply it to the image to get rid of both the distortions. After getting the Homography, we map the coordinates from image plane to world plane as in previous method to give an image free from both affine and projective distortion.

The orthogonal lines chosen for this task are the adjacent sides of the rectangle PQRS of the input images shown in previous method and the steps followed are same as in previous method.

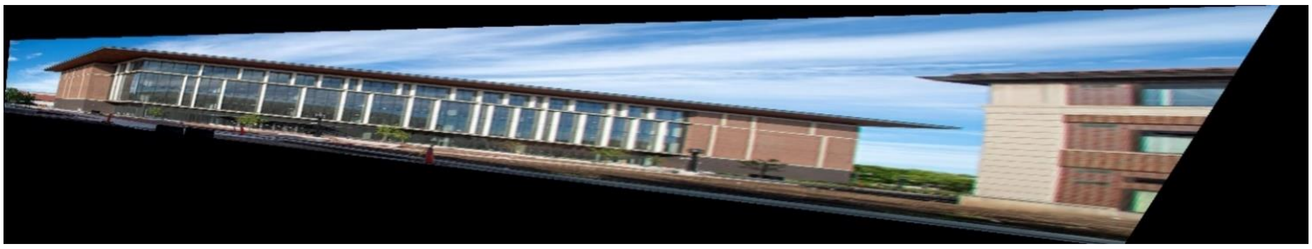

Figure 7 – Image obtained after removing both projective & affine distortion in the Image 1

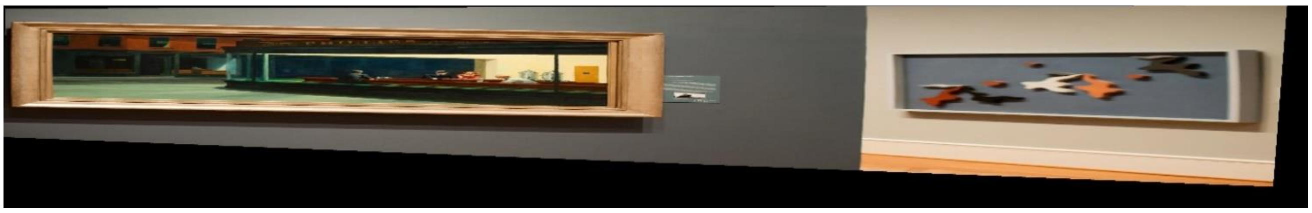

Figure 8 – Image obtained after removing both projective & affine distortion in the Image 2

We can observe that the projective and affine distortion are removed from the images i.e. the parallel lines and orthogonality between the lines are restored.

1.3 The One-Step method

This one- step method removes both affine and projective distortion in a single step using the dual degenerate conic $C_\infty^{*\prime}$.

If $C_\infty^{*\prime}$ is a projection of C_∞^{*} then,

$$C_\infty^{*\prime} = HC_\infty^* H^T, \text{ where } H = \begin{bmatrix} A & 0 \\ v^T & 1 \end{bmatrix}$$

The dual degenerate conic $C_\infty^{*\prime}$ can be represented as,

$$C_\infty^{*\prime} = \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix}$$

Here f =1 since we only care about the ratios giving us 5 unknowns a, b, c, d, e. To obtain these unknowns, we need five orthogonal line pairs that give five equations.

As solved in previous 2-step method, $L^{T\prime}\, C_\infty^{*\prime}\, M' = 0$, where L, M are orthogonal lines.

$$\begin{pmatrix} l_1' & l_2' & 1 \end{pmatrix} \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix} \begin{pmatrix} m_1' \\ m_2' \\ 1 \end{pmatrix} = 0$$

Solving for this equation further we get,

$$\begin{pmatrix} l_1'm_1' & l_1'm_2' + l_2'm_1' & l_2'm_2' & l_1' + m_1' & l_2' + m_2' \end{pmatrix} \begin{pmatrix} a \\ b/2 \\ c \\ d/2 \\ e/2 \end{pmatrix} = -1$$

Similarly, we need 5 equations of 5 pairs of orthogonal lines to obtain $C_\infty^{*\prime}$. As in the 2-step method, S = AA^T = $\begin{bmatrix} a & b/2 \\ b/2 & c \end{bmatrix}$ obtained by performing SVD. Then calculate v from Av = $\begin{bmatrix} d/2 \\ e/2 \end{bmatrix}$ = > v = A$^{-1}\begin{bmatrix} d/2 \\ e/2 \end{bmatrix}$

After obtaining the A, v values, estimate the Homography matrix H that is used to get rid of both affine and projective distortion in the image. The orthogonal lines chosen are the adjacent pairs and diagonal pairs of the input image frame PQRS shown in p2p method. The steps followed are similar to the p2p method, with the only change in the fashion of computing Homography matrix.

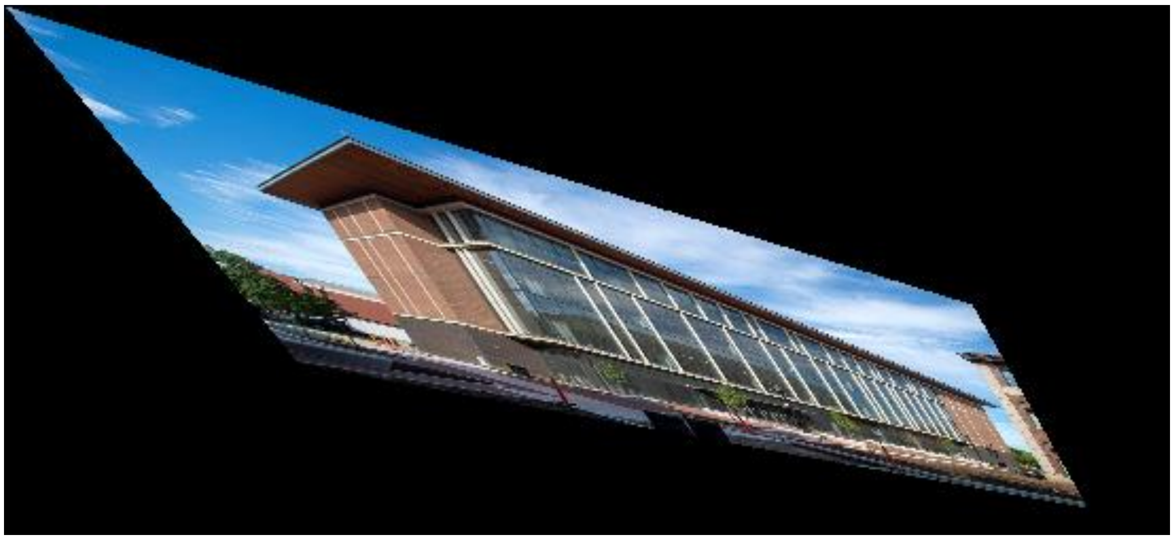

Figure 9 – Image obtained after removing both projective & affine distortion using 1-step method in the Image 1

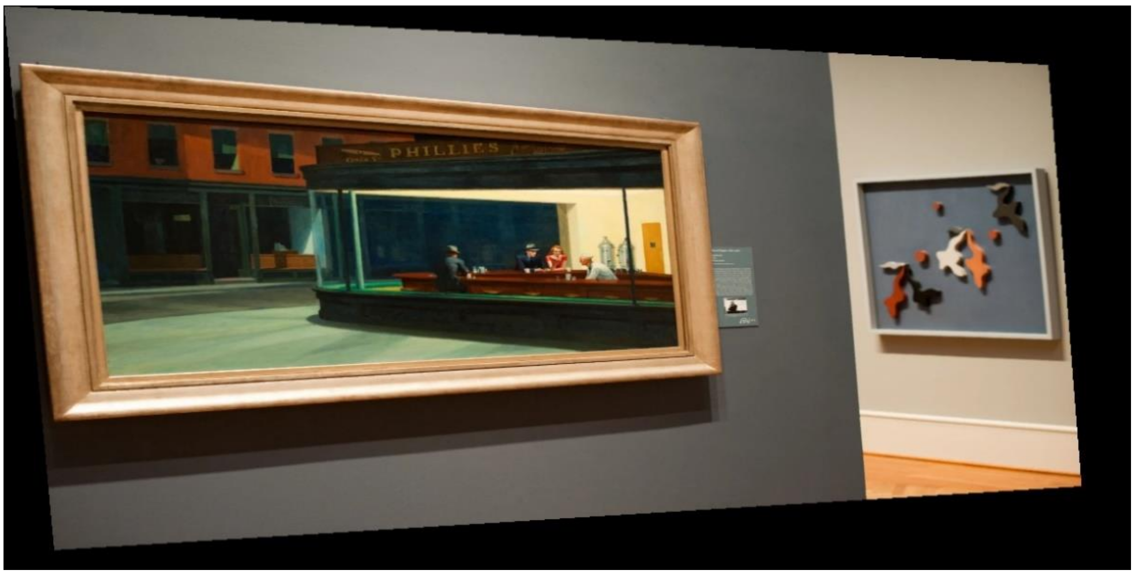

Figure 10 – Image obtained after removing both projective & affine distortion using 1-step method in the Image 2

We can finally observe from the outputs that the parallel lines and the orthogonal angles between these lines in the images are restored, thus proving that the projective and affine distortion are removed from the images.

TASK – 2:

The same methods performed in Task 1 are implemented with two of our own images shown below.

Image 3

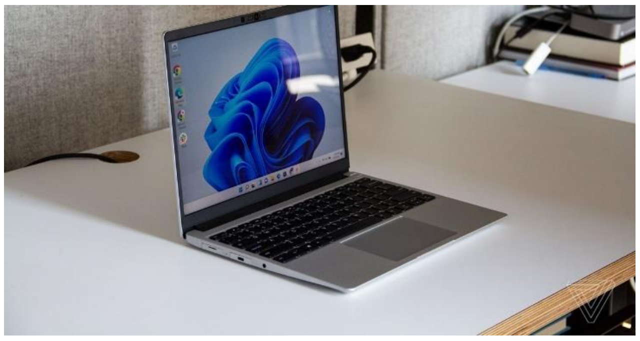

Image 4

2.1 Point-to-point Correspondence

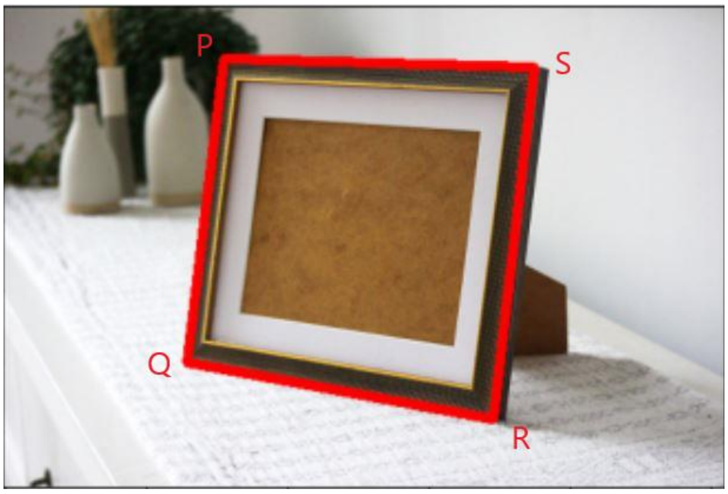

Figure 11 – Frame PQRS chosen in the Image 3

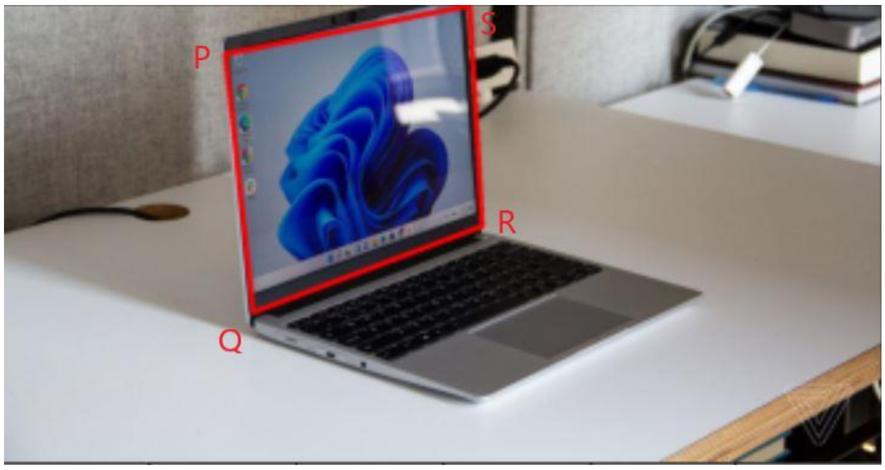

Figure 12 – Frame PQRS chosen in the Image 4

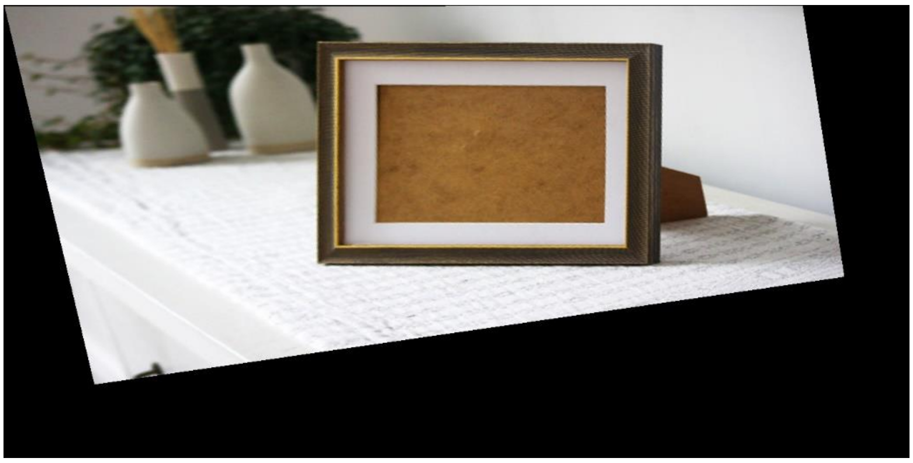
Figure 13 – Image obtained after performing point-to-point correspondence method to remove distortion in the Image 3

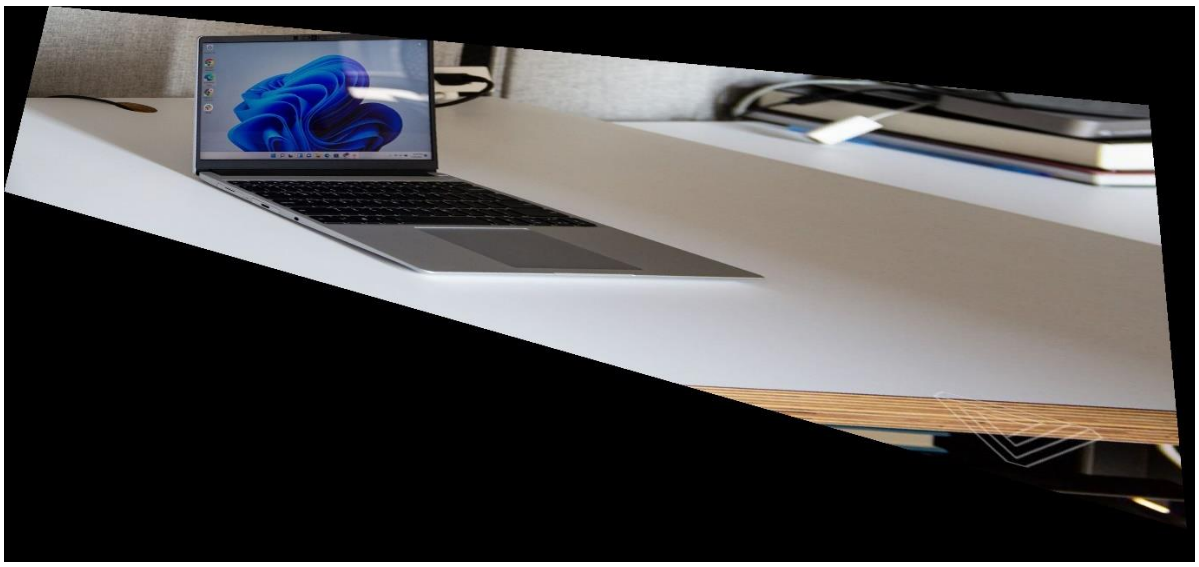
Figure 14 – Image obtained after performing point-to-point correspondence method to remove distortion in the Image 4

We can finally observe from the outputs that the parallel lines and the orthogonal angles between these lines in the images are restored, thus proving that the projective and affine distortion are removed from the images.

2.2 The Two-Step Method

2.2.1 Removing Projective Distortion

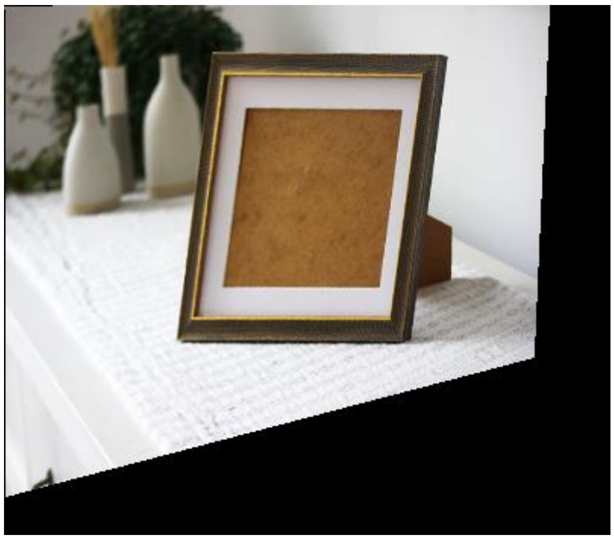

Figure 15 – Image obtained after removing projective distortion only in the Image 3

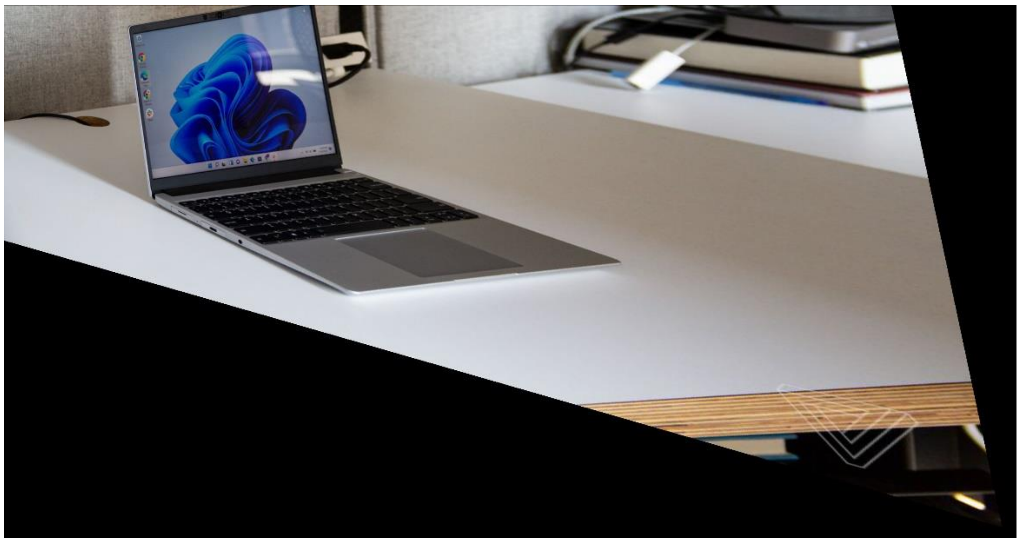

Figure 16 – Image obtained after removing projective distortion only in the Image 4

We can observe that the parallel lines are restored after removing the projective distortion, however the angles are still stretched or shrinked.

2.2.2 Removing Affine Distortion

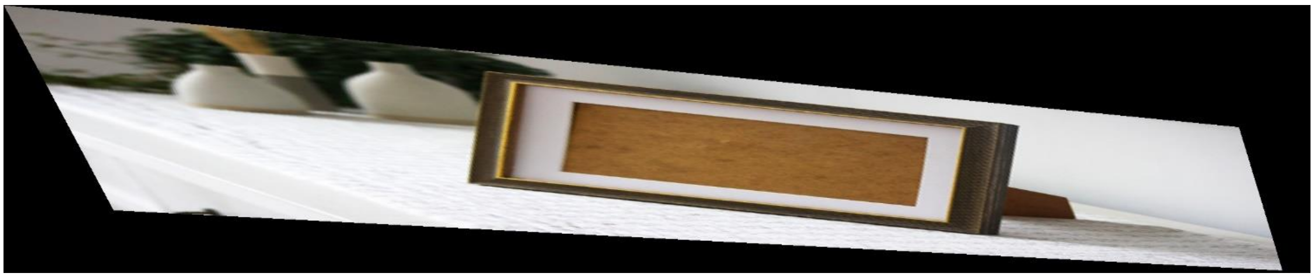

Figure 17 – Image obtained after removing both projective & affine distortion in the Image 3

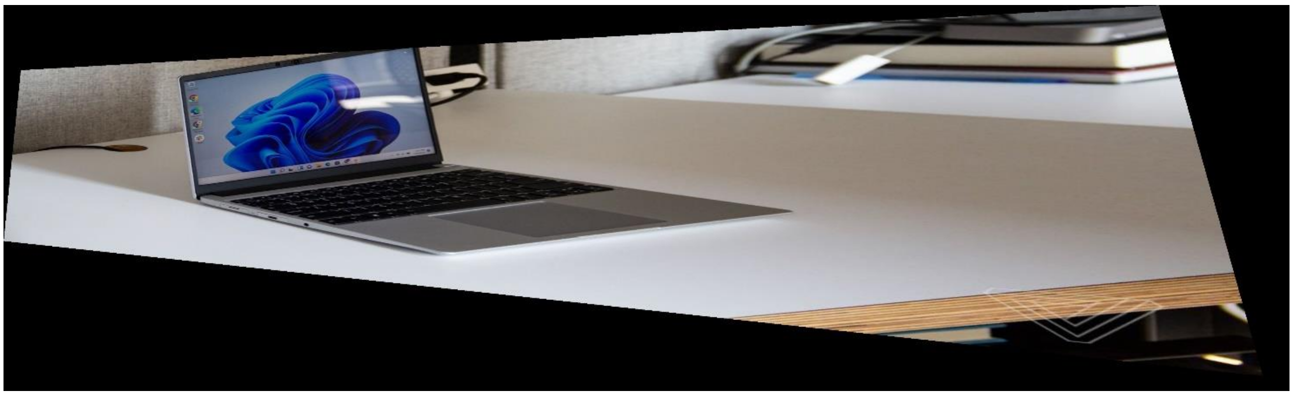

Figure 18 – Image obtained after removing both projective & affine distortion in the Image 4

We can finally observe from the outputs that the parallel lines and the orthogonal angles between these lines in the images are restored, thus proving that the projective and affine distortion are removed from the images.

2.3 The One-Step method

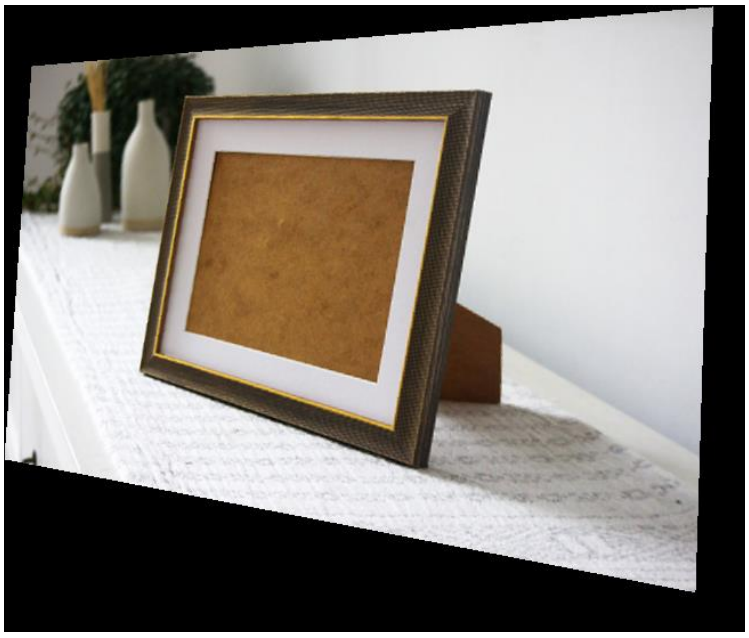

Figure 19 – Image obtained after removing both projective & affine distortion using 1-step method in the Image 3

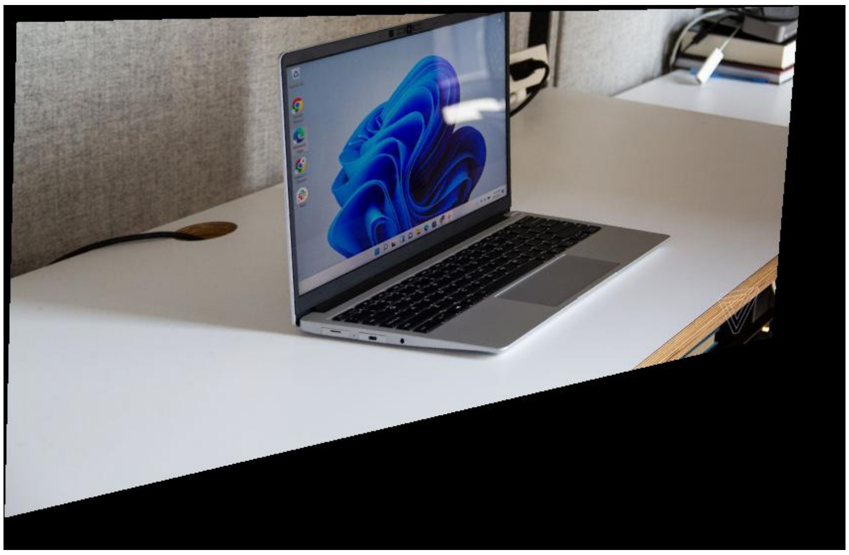

Figure 20– Image obtained after removing both projective & affine distortion using 1-step method in the Image 4

We can finally observe from the outputs that the parallel lines and the orthogonal angles between these lines in the images are restored, thus proving that the projective and affine distortion are removed from the images.

SOURCE CODE:

```
#!/usr/bin/env python
# coding: utf-8

# <h2><center>ECE661 COMPUTER VISION</center></h2>
# <h3><center>Homework - 3</center></h3>
# <h3><center>Sahithi Kodali - 34789866</center></h3>
# <h3><center>kodali1@purdue.edu</center></h3>

# ## Task 1
# ### Task 1.1 (Point-to-point correspndence method)

# In[139]:


# Import libraries needed

import numpy as np
import matplotlib.pyplot as plt
import skimage.io as sk
import cv2
import PIL
import math
```

```
# In[140]:


# Read the images from uplaoded files/directly by giving path in the computer

building_img = sk.imread('building.jpg')
canvas_img = sk.imread('nighthawks.jpg')


# In[141]:


# Create an array of pixel corodinates located at the four corners of the
Region of Interest (RoI) in the images
# Here the GIMP - GUI based tool is used to estimate the pixel ordinates of
the images

# The target coordinates are chosen based on the height and width of the
images

building_points = np.array ([[240,119], [240,457],[723,450],[717,291]],
np.int32)
building_target_points = np.array([[0,0], [0,533],[800,533],[800,0]],
np.int32)

canvas_points = np.array ([[15,103], [15,727], [865,678], [862,162]],
np.int32)
canvas_target_points = np.array([[0,0], [0,958],[1440,958],[1440,0]],
np.int32)


# In[142]:


#To check that the pixel coordinates obtained are as desired, we draw a
boundary box using these corodinates
#Below function takes the image and its pixel coordinates as input and returns
the image with boundary box

def boundarybox_check(img, points):
    img_bb = cv2.polylines(img.copy(), [points.reshape(-1,1,2)], True,
(255,0,0), thickness = 5)
    return img_bb


# In[143]:


#checking the validity of boundary box by displaying
```

```
building_bb = boundarybox_check(building_img, building_points)
canvas_bb = boundarybox_check(canvas_img, canvas_points)
plt.imshow(building_bb)


# In[144]:


# Now that the coordinates are verified, the Homography matrix [H] in the derived equation AH = B is estimated
# Below function returns the [H] by taking the points of image plane and the world plane on which it is projected

def homo_matrix(X,X_prime):

    #create an empty matrix
    A = np.zeros((8,8))

    #iterate throuogh each element and add values in A
    for i in range(4):
        A[2*i,0] = X[i,0]
        A[2*i,1] = X[i,1]
        A[2*i,2] = 1
        A[2*i,6] = -1*X[i,0]*X_prime[i,0]
        A[2*i,7] = -1*X[i,1]*X_prime[i,0]
        A[2*i+1,3] = X[i,0]
        A[2*i+1,4] = X[i,1]
        A[2*i+1,5] = 1
        A[2*i+1,6] = -1*X[i,0]*X_prime[i,1]
        A[2*i+1,7] = -1*X[i,1]*X_prime[i,1]

    #make a copy of target coordinates and reshape to required dimension
    B = X_prime.copy()
    B = B.reshape((-1,1))

    #calculate H = inv(A)B
    K = np.dot(np.linalg.inv(A), B)

    #complete the homography matrix H
    H = np.concatenate((K,np.array([[1]])),axis = 0)
    H = H.reshape((3,3))

    return H


# In[145]:


#Calculate the Homo matrix for the image plane and world plane points
```

```
H_building = homo_matrix(building_points, building_target_points)
H_canvas = homo_matrix(canvas_points, canvas_target_points)


# In[146]:


#find the minimum and maximum of the mapped coordinates to define an offset
while taking the world plane size

def find_min_max(image, homo_matrix):

    #initialize min and max of numbers
    xmin = math.inf
    xmax = -1 * math.inf

    ymin = math.inf
    ymax = -1 * math.inf

    #iterate through the image coordinates to compute mapped coordinates
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):

            coord = np.array([j,i,1])
            coord = coord.reshape((-1,1))

            #calculate mapped coordinates by finding HX
            mapped_coord = np.matmul(homo_matrix, coord)
            mapped_coord = mapped_coord/mapped_coord[2,0]

            #determine the max and min values of x and y
            if mapped_coord[0,0]<xmin:
                xmin = mapped_coord[0,0]

            if mapped_coord[0,0]>xmax:
                xmax = mapped_coord[0,0]

            if mapped_coord[1,0]<ymin:
                ymin = mapped_coord[1,0]

            if mapped_coord[1,0]>ymax:
                ymax = mapped_coord[1,0]

    return xmin,xmax,ymin,ymax


# In[147]:
```

```
#Initialize a zero matrix as the world plane based on min and max values of
the coordinates

output_img_p2p_building = np.zeros((2400,1400,3), dtype = 'uint8')
output_img_p2p_canvas = np.zeros((2100,2900,3), dtype = 'uint8')


# In[148]:


#finding the min and max of image1
find_min_max(building_img, H_building)


# In[149]:


#finding the min and max of image2
find_min_max(canvas_img, H_canvas)


# In[150]:


# Calculate mapped coordinates from image plane to the world plane

def mapping_func_weighted(image, homo_matrix, output_img, offset_x, offset_y):

    for i in range(0, output_img.shape[0]):
        for j in range(0, output_img.shape[1]):

            coord = np.array([j-offset_x,i-offset_y,1])
            coord = coord.reshape((-1,1))

            mapped_coord = np.matmul(np.linalg.pinv(homo_matrix), coord)
            mapped_coord = (mapped_coord/mapped_coord[2,0])

            mapped_coord_new = mapped_coord.reshape((3,1))

            #find the weighted pixels value
            if (0 <= mapped_coord_new[0] < image.shape[1]-1 and 0 <=
mapped_coord_new[1] < image.shape[0]-1):
                yf = int(math.floor(mapped_coord_new[0]))
                yc = int(math.ceil(mapped_coord_new[0]))
                xf = int(math.floor(mapped_coord_new[1]))
                xc = int(math.ceil(mapped_coord_new[1]))

                wff = 1/np.linalg.norm(np.array([mapped_coord_new[1] - xf,
mapped_coord_new[0] - yf]))
```

```
                wfc = 1/np.linalg.norm(np.array([mapped_coord_new[1] - xf,
mapped_coord_new[0] - yc]))
                wcf = 1/np.linalg.norm(np.array([mapped_coord_new[1] - xc,
mapped_coord_new[0] - yf]))
                wcc = 1/np.linalg.norm(np.array([mapped_coord_new[1] - xc,
mapped_coord_new[0] - yc]))

                pixel_value = wff*image[xf,yf,:] + wfc*image[xf,yc,:] +
wcf*image[xc,yf,:] + wcc*image[xc,yc,:]
                weights = wff + wcc + wfc + wcf

                output_img[i,j,:] = pixel_value/weights

    #map the corresponsing pixel value and clip to RGB range
    output_img = np.clip(output_img, 0.0, 255.0)
    output_img = output_img.astype(np.uint8)

    return output_img


# In[15]:


#Apply the mapping function for Image1(building) to showcase the output world
plane image and save the image to local computer

output_p2p_building_weighted = mapping_func_weighted(building_img, H_building,
output_img_p2p_building, 149, 1466)
plt.imshow(output_p2p_building_weighted)
filename = 'D:\Purdue\ECE661_CV\HW3_outputs' + '\Image' +
'_output_p2p_building_weighted' +'.png'
sk.imsave(filename, output_p2p_building_weighted)


# In[17]:


#Apply the mapping function for Image2(canvas) to showcase the output world
plane image and save the image to local computer

output_p2p_canvas_weighted = mapping_func_weighted(canvas_img, H_canvas,
output_img_p2p_canvas, 21, 440)
plt.imshow(output_p2p_canvas_weighted)
filename = 'D:\Purdue\ECE661_CV\HW3_outputs' + '\Image' +
'_output_p2p_canvas_weighted' +'.png'
sk.imsave(filename, output_p2p_canvas_weighted)


# ### Task 1.2 (Two- step method)
# #### Removing Projective distortion

# In[151]:


#Change the points to HC representation by adding z axis coordinate

building_points_3 = np.array ([[240,119,1],
[240,457,1],[723,450,1],[717,291,1]], np.int32)
canvas_points_3 = np.array ([[15,103,1], [15,727,1], [865,678,1],
[862,162,1]], np.int32)


# In[152]:


#compute the homography function that gets rid of projective distortion

def proj_homo_matrix(points):

    #find l1(passing through P,S points) and l2 (passing through Q,R points)
parallel lines in PQRS frame
    l1 = np.cross(points[0],points[3])
    l2 = np.cross(points[1],points[2])

    #find l3(passing through P,Q points) and l4 (passing through S,R points)
parallel lines in PQRS frame
    l3 = np.cross(points[0],points[1])
    l4 = np.cross(points[3],points[2])

    #find the vanishing point of l1 and l2
    vp1 = np.cross(l1, l2)
    vp1 = vp1/vp1[2]

    #find the vanishing point of l3 and l4
    vp2 = np.cross(l3, l4)
    vp2 = vp2/vp2[2]

    #find the vanishing line from the two vanishing points
    vl = np.cross(vp1,vp2)
    print(vl)
    vl = vl/vl[2]
    print(vl)

    #compute H
    H = np.zeros((3,3))
    H[0][0] = 1
    H[1][1] = 1
    H[2] = vl
    return H


# In[153]:


#Calculate the projective Homo matrix for the image plane and world plane
points

H_proj_building = proj_homo_matrix(building_points_3)
H_proj_canvas = proj_homo_matrix(canvas_points_3)


# In[154]:


#finding the min and max of images1 and 2

find_min_max(building_img, H_proj_building)
find_min_max(canvas_img, H_proj_canvas)


# In[155]:


##Initialize a zero matrix as the world plane based on min and max values of
the coordinates

output_img_proj_building = np.zeros((1700,2900,3), dtype = 'uint8')
output_img_proj_canvas = np.zeros((1400,2100,3), dtype = 'uint8')


# In[26]:


#Apply the mapping function for Image1(building) to showcase the output world
plane image and save the image to local computer

image_proj_building = mapping_func_weighted(building_img, H_proj_building,
output_img_proj_building, 0, 0)
plt.imshow(image_proj_building)
filename = 'D:\Purdue\ECE661_CV\HW3_outputs' + '\Image' + '_proj_building'
+'.png'
sk.imsave(filename, image_proj_building)


# In[27]:
```

```
#Apply the mapping function for Image2(canvas) to showcase the output world
plane image and save the image to local computer

image_proj_canvas = mapping_func_weighted(canvas_img, H_proj_canvas,
output_img_proj_canvas, 0, 0)
plt.imshow(image_proj_canvas)
filename = 'D:\Purdue\ECE661_CV\HW3_outputs' + '\Image' + '_proj_canvas'
+'.png'
sk.imsave(filename, image_proj_canvas)


# #### Removing Affine distortion

# In[156]:


#compute the homography function that gets rid of affine distortion by
following the concept explained in the report

def affine_homo_matrix(points):

    #find the pair of orthogonal lines of PQRS frame i.e, PQ&PS, QR&RS pairs
    l1 = np.cross(points[0],points[1])
    l1 = l1/l1[2]

    m1 = np.cross(points[0],points[3])
    m1 = m1/m1[2]

    l2 = np.cross(points[1],points[2])
    l2 = l2/l2[2]

    m2 = np.cross(points[2],points[3])
    m2 = m2/m2[2]

    #create lists to append the linear equations
    mat1 = []
    mat2 = []

    mat1.append([ l1[0]*m1[0], l1[0]*m1[1] + l1[1]*m1[0] ])
    mat1.append([ l2[0]*m2[0], l2[0]*m2[1] + l2[1]*m2[0] ])

    mat2.append([ -l1[1]*m1[1] ])
    mat2.append([ -l2[1]*m2[1] ])

    mat1 = np.asarray(mat1)
    mat2 = np.asarray(mat2)

    #compute S matrix
    S_temp = np.dot(np.linalg.pinv(mat1), mat2)
    S = np.zeros((2,2))
    S[0][0] = S_temp[0]
    S[0][1] = S[1][0] = S_temp[1]
    S[1][1] = 1

    #Perform a Singular Value Decomposition on S and compute A from its values
    u,s,vh = np.linalg.svd(S)
    D = np.sqrt(np.diag(s))
    A = np.dot(np.dot(u,D),np.transpose(u))

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

    H[0][0] = A[0][0]
    H[0][1] = A[0][1]
    H[1][0] = A[1][0]
    H[1][1] = A[1][1]
    H[2][2] = 1

    return H


# In[157]:


#Calculate the affine Homo matrix for the image plane and world plane points

H_affine_building = affine_homo_matrix(building_points_3)
H_affine_canvas = affine_homo_matrix(canvas_points_3)


# #### Removing both Projective and Affine distortion

# In[158]:


#Calculate the homography to get rid of both affine and projective distortions
#Multiply Affine homography matrix with projective homography matrix obtained
for the images

H_affine_proj_building = np.dot(np.linalg.pinv(H_affine_building),
H_proj_building)
H_affine_proj_canvas = np.dot(np.linalg.pinv(H_affine_canvas), H_proj_canvas)


# In[159]:
```

```
#finding the min and max of images1 and 2

find_min_max(building_img, H_affine_proj_building)
find_min_max(canvas_img, H_affine_proj_canvas)


# In[160]:


#Initialize a zero matrix as the world plane

output_img_affine_proj_building = np.zeros((1700,9300,3), dtype = 'uint8')
output_img_affine_proj_canvas = np.zeros((1500,9700,3), dtype = 'uint8')


# In[49]:


#Apply the mapping function for Image1(building) to showcase the output world
plane image and save the image to local computer

image_affine_proj_building = mapping_func_weighted(building_img,
H_affine_proj_building, output_img_affine_proj_building, 46, 250)
plt.imshow(image_affine_proj_building)
filename = 'D:\Purdue\ECE661_CV\HW3_outputs' + '\Image' +
'_affine_proj_building' +'.png'
sk.imsave(filename, image_affine_proj_building)


# In[71]:


#Apply the mapping function for Image2(canvas) to showcase the output world
plane image and save the image to local computer

image_affine_proj_canvas = mapping_func_weighted(canvas_img,
H_affine_proj_canvas, output_img_affine_proj_canvas, 12, 25)
plt.imshow(image_affine_proj_canvas)
filename = 'D:\Purdue\ECE661_CV\HW3_outputs' + '\Image' +
'_affine_proj_canvas' +'.png'
sk.imsave(filename, image_affine_proj_canvas)


# ### Task 1.3 (One- step method)

# In[202]:


#Compute the one step method matrix based on the concept discussed in the
report to remove affine and projective distortions
```

```python
def onestep_homo_matrix(points):

    # Compute the pair of orthogonal lines from points of PQRS consdiering
adjacent sides for each pair
    # Choose a diagonal for the last orthogonal pair

    l1 = np.cross(points[0],points[1])
    l1 = l1/l1[2]

    m1 = np.cross(points[0],points[3])
    m1 = m1/m1[2]

    l2 = np.cross(points[0],points[1])
    l2 = l2/l2[2]

    m2 = np.cross(points[1],points[2])
    m2 = m2/m2[2]

    l3 = np.cross(points[1],points[2])
    l3 = l3/l3[2]

    m3 = np.cross(points[2],points[3])
    m3 = m3/m3[2]

    l4 = np.cross(points[2],points[3])
    l4 = l4/l4[2]

    m4 = np.cross(points[3],points[0])
    m4 = m4/m4[2]

    l5 = np.cross(points[0],points[2])
    l5 = l5/l5[2]

    m5 = np.cross(points[1],points[3])
    m5 = m5/m5[2]

    #give two matrixes to append the linear equations of the orthogonal lines
according to the concept in the report
    mat1 = []
    mat2 = []

    mat1.append([ l1[0]*m1[0], (l1[0]*m1[1] + l1[1]*m1[0])/2, l1[1]*m1[1],
(l1[0] + m1[0])/2, (l1[1] + m1[1])/2 ])
    mat1.append([ l2[0]*m2[0], (l2[0]*m2[1] + l2[1]*m2[0])/2, l2[1]*m2[1],
(l2[0] + m2[0])/2, (l2[1] + m2[1])/2 ])
    mat1.append([ l3[0]*m3[0], (l3[0]*m3[1] + l3[1]*m3[0])/2, l3[1]*m3[1],
(l3[0] + m3[0])/2, (l3[1] + m3[1])/2 ])
```

```
    mat1.append([ l4[0]*m4[0], (l4[0]*m4[1] + l4[1]*m4[0])/2, l4[1]*m4[1],
(l4[0] + m4[0])/2, (l4[1] + m4[1])/2 ])
    mat1.append([ l5[0]*m5[0], (l5[0]*m5[1] + l5[1]*m5[0])/2, l5[1]*m5[1],
(l5[0] + m5[0])/2, (l5[1] + m5[1])/2 ])


    mat2.append([ -l1[2]*m1[2] ])
    mat2.append([ -l2[2]*m2[2] ])
    mat2.append([ -l3[2]*m3[2] ])
    mat2.append([ -l4[2]*m4[2] ])
    mat2.append([ -l5[2]*m5[2] ])


    mat1 = np.asarray(mat1)
    mat2 = np.asarray(mat2)

    #Compute c_infinity to find their respective coefficents and determine S
    C_inf = np.dot(np.linalg.pinv(mat1), mat2)
    C_inf = C_inf/np.max(C_inf)

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

    #compute SVD of matrix S
    u,s,vh = np.linalg.svd(S)
    D = np.sqrt(np.diag(s))

    #compute matrix A from the SVD values computed
    A = np.dot(np.dot(u,D),np.transpose(u))

    #Compute matrix V from A
    V = np.dot(np.linalg.pinv(A), np.array([C_inf[3]/2, C_inf[4]/2]))

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

    H[0][0] = A[0][0]
    H[0][1] = A[0][1]
    H[1][0] = A[1][0]
    H[1][1] = A[1][1]
    H[2][0] = V[0]
    H[2][1] = V[1]
    H[2][2] = 1
    print(H)

    return H
```

```
# In[203]:


#Calculate the 1-step method Homo matrix for the image plane and world plane
points

H_1step_building = onestep_homo_matrix(building_points_3)
H_1step_canvas = onestep_homo_matrix(canvas_points_3)


# In[204]:


#finding the min and max of image 1

find_min_max(building_img, H_1step_building)


# In[205]:


#finding the min and max of image 2

find_min_max(canvas_img, H_1step_canvas)


# In[209]:


# Initialize a zero matrix as the world plane based on min and max values of
the coordinates

output_img_1step_building = np.zeros((250,550,3), dtype = 'uint8')
output_img_1step_canvas = np.zeros((600,1200,3), dtype = 'uint8')


# In[210]:


#Apply the mapping function for Image1(building) to showcase the output world
plane image and save the image to local computer

image_1step_building = mapping_func_weighted(building_img, H_1step_building,
output_img_1step_building, 0, 0)
plt.imshow(image_1step_building)
filename = 'D:\Purdue\ECE661_CV\HW3_outputs' + '\Image' + '_1step_building'
+'.png'
sk.imsave(filename, image_1step_building)
```

```
# In[211]:


#Apply the mapping function for Image2(canvas) to showcase the output world
plane image and save the image to local computer

image_1step_canvas = mapping_func_weighted(canvas_img, H_1step_canvas,
output_img_1step_canvas, 0, 0)
plt.imshow(image_1step_canvas)
filename = 'D:\Purdue\ECE661_CV\HW3_outputs' + '\Image' + '_1step_canvas'
+'.png'
sk.imsave(filename, image_1step_canvas)


# ## Task-2
# ### Task 2.1 (Point-to-point correspondence method)

# In[113]:


# Read the images from uplaoded files/directly by giving path in the computer

frame_img = sk.imread('frame.jpg')
lappy_img = sk.imread('lappy.jpg')


# In[114]:


# Create an array of pixel corodinates located at the four corners of the
Region of Interest (RoI) in the images
# Here the GIMP - GUI based tool is used to estimate the pixel ordinates of
the images

# The target coordinates are chosen based on the height and width of the
images

frame_points = np.array ([[156,37], [130,251], [346,291], [375,43]], np.int32)
frame_target_points = np.array([[0,0], [0,339],[509,339],[509,0]], np.int32)

lappy_points = np.array ([[306,73], [344,424], [656,307], [639,7]], np.int32)
lappy_target_points = np.array([[0,0], [0,628],[1200,628],[1200,0]], np.int32)


# In[115]:


#verify that the points are as desired using boundary box function

frame_bb = boundarybox_check(frame_img, frame_points)
```

```
lappy_bb = boundarybox_check(lappy_img, lappy_points)
plt.imshow(lappy_bb)


# In[116]:


#Calculate the Homo matrix of images between the image and world plane

H_frame = homo_matrix(frame_points, frame_target_points)
H_lappy = homo_matrix(lappy_points, lappy_target_points)


# In[118]:


#finding min and max of coordinates of frame image

find_min_max(frame_img, H_frame)


# In[119]:


#finding min and max of coordinates of lappy image

find_min_max(lappy_img, H_lappy)


# In[120]:


#Initialize a zero matrix as the world plane based on min and max values of
the coordinates

output_img_p2p_frame = np.zeros((700,1400,3), dtype = 'uint8')
output_img_p2p_lappy = np.zeros((2800,6000,3), dtype = 'uint8')


# In[121]:


#Apply the mapping function for Image3(frame) to showcase the output world
plane image and save the image to local computer

output_p2p_frame_weighted = mapping_func_weighted(frame_img, H_frame,
output_img_p2p_frame, 488, 60)
plt.imshow(output_p2p_frame_weighted)
filename = 'D:\Purdue\ECE661_CV\HW3_outputs' + '\Image' +
'_output_p2p_frame_weighted' +'.png'
```

```
sk.imsave(filename, output_p2p_frame_weighted)


# In[122]:


#Apply the mapping function for Image4(lappy) to showcase the output world
plane image and save the image to local computer

output_p2p_lappy_weighted = mapping_func_weighted(lappy_img, H_lappy,
output_img_p2p_lappy, 986, 191)
plt.imshow(output_p2p_lappy_weighted)
filename = 'D:\Purdue\ECE661_CV\HW3_outputs' + '\Image' +
'_output_p2p_lappy_weighted' +'.png'
sk.imsave(filename, output_p2p_lappy_weighted)


# ### Task 2.2 (Two - step method)
# #### Removing projective distortion

# In[123]:


#Change the points to HC representation by adding z axis coordinate

frame_points_3 = np.array ([[156,37,1], [130,251,1], [346,291,1], [375,43,1]],
np.int32)
lappy_points_3 = np.array ([[306,73,1], [344,424,1], [656,307,1], [639,7,1]],
np.int32)


# In[124]:


#Calculate the projective Homo matrix of images between the image and world
plane

H_proj_frame = proj_homo_matrix(frame_points_3)
H_proj_lappy = proj_homo_matrix(lappy_points_3)


# In[125]:


#finding min and max of coordinates of frame image

find_min_max(frame_img, H_proj_frame)


# In[126]:
```

```
#finding min and max of coordinates of lappy image

find_min_max(lappy_img, H_proj_lappy)


# In[127]:


#Initialize a zero matrix as the world plane based on min and max values of
the coordinates

output_img_proj_frame = np.zeros((350,400,3), dtype = 'uint8')
output_img_proj_lappy = np.zeros((1500,2850,3), dtype = 'uint8')


# In[128]:


#Apply the mapping function for Image3(frame) to showcase the output world
plane image and save the image to local computer

image_proj_frame = mapping_func_weighted(frame_img, H_proj_frame,
output_img_proj_frame, 0, 0)
plt.imshow(image_proj_frame)
filename = 'D:\Purdue\ECE661_CV\HW3_outputs' + '\Image' + '_proj_frame'
+'.png'
sk.imsave(filename, image_proj_frame)


# In[136]:


#Apply the mapping function for Image4(lappy) to showcase the output world
plane image and save the image to local computer

image_proj_lappy = mapping_func_weighted(lappy_img, H_proj_lappy,
output_img_proj_lappy, 0, 0)
plt.imshow(image_proj_lappy)
filename = 'D:\Purdue\ECE661_CV\HW3_outputs' + '\Image' + '_proj_lappy'
+'.png'
sk.imsave(filename, image_proj_lappy)


# #### Removing affine distortion

# In[129]:
```

```
#compute the affine homography for images 3 and 4 (frame and lappy)

H_affine_frame = affine_homo_matrix(frame_points_3)
H_affine_lappy = affine_homo_matrix(lappy_points_3)


# #### Removing affine & projective distortion

# In[130]:


#Calculate the homography to get rid of both affine and projective distortions
#Multiply Affine homography matrix with projective homography matrix obtained
for the images

H_affine_proj_frame = np.dot(np.linalg.pinv(H_affine_frame), H_proj_frame)
H_affine_proj_lappy = np.dot(np.linalg.pinv(H_affine_lappy), H_proj_lappy)


# In[131]:


#finding min and max of coordinates of frame image

find_min_max(frame_img, H_affine_proj_frame)


# In[132]:


#finding min and max of coordinates of lappy image

find_min_max(lappy_img, H_affine_proj_lappy)


# In[133]:


#Initialize a zero matrix as the world plane based on min and max values of
the coordinates

output_img_affine_proj_frame = np.zeros((450,2200,3), dtype = 'uint8')
output_img_affine_proj_lappy = np.zeros((1500,5000,3), dtype = 'uint8')


# In[135]:


#Apply the mapping function for Image3(frame) to showcase the output world
plane image and save the image to local computer
```

```
image_affine_proj_frame = mapping_func_weighted(frame_img,
H_affine_proj_frame, output_img_affine_proj_frame, 0, 0)
plt.imshow(image_affine_proj_frame)
filename = 'D:\Purdue\ECE661_CV\HW3_outputs' + '\Image' + '_affine_proj_frame'
+'.png'
sk.imsave(filename, image_affine_proj_frame)


# In[137]:


#Apply the mapping function for Image4(lappy) to showcase the output world
plane image and save the image to local computer

image_affine_proj_lappy = mapping_func_weighted(lappy_img,
H_affine_proj_lappy, output_img_affine_proj_lappy, 67, 253)
plt.imshow(image_affine_proj_lappy)
filename = 'D:\Purdue\ECE661_CV\HW3_outputs' + '\Image' + '_affine_proj_lappy'
+'.png'
sk.imsave(filename, image_affine_proj_lappy)


# ### Task 2.3 (One-step method)

# In[212]:


#Calculate the 1-step method Homo matrix for the image plane and world plane
points

H_1step_frame = onestep_homo_matrix(frame_points_3)
H_1step_lappy = onestep_homo_matrix(lappy_points_3)


# In[213]:


#finding the min and max of image 1

find_min_max(frame_img, H_1step_frame)


# In[214]:


#finding the min and max of image 1

find_min_max(lappy_img, H_1step_lappy)
```

```
# In[215]:


# Initialize a zero matrix as the world plane based on min and max values of
the coordinates

output_img_1step_frame = np.zeros((550,650,3), dtype = 'uint8')
output_img_1step_lappy = np.zeros((550,850,3), dtype = 'uint8')


# In[216]:


#Apply the mapping function for Image3(frame) to showcase the output world
plane image and save the image to local computer

image_1step_frame = mapping_func_weighted(frame_img, H_1step_frame,
output_img_1step_frame, 24, 53)
plt.imshow(image_1step_frame)
filename = 'D:\Purdue\ECE661_CV\HW3_outputs' + '\Image' + '_1step_frame'
+'.png'
sk.imsave(filename, image_1step_frame)


# In[217]:


#Apply the mapping function for Image4(lappy) to showcase the output world
plane image and save the image to local computer

image_1step_lappy = mapping_func_weighted(lappy_img, H_1step_lappy,
output_img_1step_lappy, 13, 17)
plt.imshow(image_1step_lappy)
filename = 'D:\Purdue\ECE661_CV\HW3_outputs' + '\Image' + '_1step_lappy'
+'.png'
sk.imsave(filename, image_1step_lappy)
```

****