PURDUE UNIVERSITY

Elmore Family School of Electrical and Computer Engineering

Computer Vision

Homework 3

Adithya Sineesh

Email : asineesh@purdue.edu

Submitted: September 20, 2022

1 Theory

In this homework, we are implementing different approaches to remove purely projective and affine distortions. This process can also be referred to as metric rectification. Let us first look at the mathematical formulae behind these different methods.

1.1 Point to Point correspondences

This is similar to what ah been done in the previous homework. Given a point X corresponding to (x, y) in the domain plane R^2, X' corresponding to (x', y') in the range plane R^2 and H being their corresponding mapping, the relation between them is as follows:

$$X' = HX$$

where,

$H =$

$$\begin{pmatrix} \mathrm{h}_{11} & \mathrm{h}_{12} & \mathrm{h}_{13} \\ \mathrm{h}_{21} & \mathrm{h}_{22} & \mathrm{h}_{23} \\ \mathrm{h}_{31} & \mathrm{h}_{32} & \mathrm{h}_{33} \end{pmatrix}$$

$X =$

$$\begin{pmatrix} \mathrm{x}_1 \\ \mathrm{x}_2 \\ \mathrm{x}_3 \end{pmatrix}$$

$X' =$

$$\begin{pmatrix} \mathrm{x'}_1 \\ \mathrm{x'}_2 \\ \mathrm{x'}_3 \end{pmatrix}$$

By expanding the above equation we get,

$$x'_1 = h_{11}x_1 + h_{12}x_2 + h_{13}x_3$$
$$x'_2 = h_{21}x_1 + h_{22}x_2 + h_{23}x_3$$
$$x'_3 = h_{31}x_1 + h_{32}x_2 + h_{33}x_3$$

We know that the physical coordinates can obtained from the representative coordinates as $x = \frac{x_1}{x_3}$, $y = \frac{x_2}{x_3}$, $x' = \frac{x'_1}{x'_3}$ and $y' = \frac{x'_2}{x'_3}$. And as H is homogeneous, only the ratio of the elements of H are important, so we can set $h_{33} = 1$. Using these to facts along with the above 3 equations, we can write the physical coordinates of the domain plane as follows:

$$x' = \frac{h_{11}x+h_{12}y+h_{13}}{h_{31}x+h_{32}y+1}$$
$$y' = \frac{h_{21}x+h_{22}y+h_{23}}{h_{31}x+h_{32}y+1}$$

Simplifying the above 2 equations we get:

$$h_{11}x + h_{12}y + h_{13} - h_{31}xx' - h_{32}yx' = x'$$
$$h_{21}x + h_{22}y + h_{23} - h_{31}xy' - h_{32}yy' = y'$$

Therefore, as we have 8 unknowns $(h_{11}, h_{12}, h_{13}, h_{21}, h_{22}, h_{23}, h_{31}, h_{32})$, we need at least 8 equations to find them. As a single pair of corresponding points in the domain and range plane gives us 2 equations, we only need 4 pairs of corresponding points to find out all the 8 unknowns. We should also take care that no more than 2 of the 4 points lie on a straight line.

Let us take 4 points $(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)$ from the domain plane and the 4 corresponding points $(x'_1, y'_1), (x'_2, y'_2), (x'_3, y'_3), (x'_4, y'_4)$ from the range plane. Now we can construct the following matrix equation:

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

Since the above matrix equation is in the form of $AX = B$, we can then find the value of all the 8 unknowns present in X as follows:

$$X = A^{-1}B$$

1.2 The Two step method

As the name suggests, we initially remove the purely projective distortion from the image and then we remove the purely affine transformation from the image.

1.2.1 Removing purely Projective distortion

A homography is affine if and only if the line at infinity in the domain image is mapped to a line at infinity in the range image. Therefor if we apply a homography to an image that sends the vanishing line back to infinity, the remaining distortion in the image will be purely affine i.e. we have removed the purely projective distortion.

We start by picking two pairs of lines that will be parallel after removing the projective distortion. We then find the point of intersection of these pair of lines by applying cross product to their homogeneous coordinates. We then find the line that joins these two points by again applying cross product to their homogeneous product. This line is the vanishing line.

If the vanishing line is given as $l_\infty = (l_1, l_2, l_3)^T$, the homography that pushes the vanishing line back to infinity can be written as follows:

$H =$

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

1.2.2 Removing purely Affine distortion

After the applying the homography obtained using the first step, we get an image that only suffers from affine distortion. After correcting the purely affine distortion, the angles between any pairs of lines should be the same as it was in the original scene.

Given two lines in the domain image $l = (l_1, l_2, l_3)^T$ and $m = (m_1, m_2, m_3)^T$, the angle between them can be written as

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

where $C_\infty^\star$ is the dual degenerate conic at infinity.

We now write the above equation using the corresponding range terms. As we know, $l = H^T l'$ and $m = H^T m'$. By taking $\cos\theta = 0$ i.e. orthogonal lines, the equation will be

$$l'^T H C_\infty^\star H^T m = 0$$

where H and $C_\infty^\star$ are

$H =$

$$\begin{pmatrix} \mathrm{A} & 0 \\ 0^T & 1 \end{pmatrix}$$

$C_\infty^\star =$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

So the equation can be simplified and written as follows:

$$s_{11}l'_1m'_1 + s_{12}(l'_1m'_2 + l'_2m'_1) + s_{22}l'_2m'_2 = 0$$

where S is a symmetric matrix and $S = AA^T =$

$$\begin{pmatrix} \mathrm{s}_{11} & \mathrm{s}_{12} \\ \mathrm{s}_{21} & \mathrm{s}_{22} \end{pmatrix}$$

The above equation has 3 unknowns (s_{11},s_{12},s_{22}), but since all the information is contained in the ratios, we can set $s_{22} = 0$. Now, we only need 2 equations to solve for S. This means we need a minimum of 2 pairs of orthogonal lines to obtain S. So taking two pairs of orthogonal lines l_a,m_a and l_b,m_b, we can write the following matrix equation.

$$\begin{pmatrix} \mathrm{l'}_{a1}m'_{a1} & \mathrm{l'}_{a2}m'_{a1} + l'_{a1}m'_{a2} \\ \mathrm{l'}_{b1}m'_{b1} & \mathrm{l'}_{b2}m'_{b1} + l'_{b1}m'_{b2} \end{pmatrix} \times \begin{pmatrix} \mathrm{s}_{11} \\ \mathrm{s}_{11} \end{pmatrix} = - \begin{pmatrix} \mathrm{l'}_{a2}m'_{a2} \\ \mathrm{l'}_{b2}m'_{b2} \end{pmatrix}$$

Since the above matrix equation is in the form of $AX = B$, we can then find the value of the 2 unknowns present in X as follows:

$$X = A^{-1}B$$

Now, we have the value of the S matrix. Since we know $S = AA^T$, and the eigen decomposition of A can be written as $A = VDV^T$. Therefore

$$S = AA^T$$
$$S = (VDV^T)(VDV^T)^T$$
$$S = VD^2V$$

This shows that S and A share eigenvectors and the eigenvalues of A are the positive square root of the eigenvalues S (both S and A are positive definite). Therefore we can construct A from S and obtain H subsequently.

1.3 The One step method

As the name suggests, we remove both the purely projective and purely affine distortions in one step.

Given two lines in the domain image $l = (l_1, l_2, l_3)^T$ and $m = (m_1, m_2, m_3)^T$, the angle between them can be written as

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

where $C_\infty^\star$ is the dual degenerate conic at infinity.

We now write the above equation using the corresponding range terms. As we know, $l = H^T l'$, $m = H^T m'$ and $C'^\star = HC_\infty^\star H^T$. By taking $\cos\theta = 0$ i.e. orthogonal lines, the equation will be

$$l'^T C'^\star m = 0$$

where $C'^\star$ is $C'^\star =$

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

As all the information is present in the ratios, we can set $f = 1$ and thus need a minimum of 5 equations to get the 5 unknowns (a,b,c,d,e). This means we need a minimum of 5 pairs of orthogonal lines to obtain $C'^{\star}$. So taking five pairs of orthogonal lines l_a,m_a, l_b,m_b, l_c,m_c, l_d,m_d and l_e,m_e, we can write the following matrix equation

$$\begin{pmatrix} \text{l'}_{a1}m'_{a1} & \text{l'}_{a2}m'_{a1} + l'_{a1}m'_{a2} & \text{l'}_{a2}m'_{a2} & \text{l'}_{a3}m'_{a1} + l'_{a1}m'_{a3} & \text{l'}_{a3}m'_{a2} + l'_{a2}m'_{a3} \\ \text{l'}_{b1}m'_{b1} & \text{l'}_{b2}m'_{b1} + l'_{b1}m'_{b2} & \text{l'}_{b2}m'_{b2} & \text{l'}_{b3}m'_{b1} + l'_{b1}m'_{b3} & \text{l'}_{b3}m'_{b2} + l'_{b2}m'_{b3} \\ \text{l'}_{c1}m'_{c1} & \text{l'}_{c2}m'_{c1} + l'_{c1}m'_{c2} & \text{l'}_{c2}m'_{c2} & \text{l'}_{c3}m'_{c1} + l'_{c1}m'_{c3} & \text{l'}_{c3}m'_{c2} + l'_{c2}m'_{c3} \\ \text{l'}_{d1}m'_{d1} & \text{l'}_{d2}m'_{d1} + l'_{d1}m'_{d2} & \text{l'}_{d2}m'_{d2} & \text{l'}_{d3}m'_{d1} + l'_{d1}m'_{d3} & \text{l'}_{d3}m'_{d2} + l'_{d2}m'_{d3} \\ \text{l'}_{e1}m'_{e1} & \text{l'}_{e2}m'_{e1} + l'_{e1}m'_{e2} & \text{l'}_{e2}m'_{e2} & \text{l'}_{e3}m'_{e1} + l'_{e1}m'_{e3} & \text{l'}_{e3}m'_{e2} + l'_{e2}m'_{e3} \end{pmatrix} \times \begin{pmatrix} \text{a} \\ \text{b/2} \\ \text{c} \\ \text{d/2} \\ \text{e/2} \end{pmatrix} = - \begin{pmatrix} \text{l'}_{a3}m'_{a3} \\ \text{l'}_{b3}m'_{b3} \\ \text{l'}_{c3}m'_{c3} \\ \text{l'}_{d3}m'_{d3} \\ \text{l'}_{e3}m'_{e3} \end{pmatrix}$$

Since the above matrix equation is in the form of $AX = B$, we can then find the value of the 5 unknowns present in X as follows:

$$X = A^{-1}B$$

Now as we have the value of the conic $C'^{\star}$, we can write the following,

$$C'^{\star} = HC^{\star}_{\infty}H^T$$

$$\text{C'}^{\star} = \begin{pmatrix} \text{A} & 0 \\ \text{v}^T & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} \text{A}^T & \text{v} \\ 0^T & 1 \end{pmatrix} = \begin{pmatrix} \text{AA}^T & \text{Av} \\ 0\text{v}^T A^T & 0\text{v}^T v \end{pmatrix}$$

So we can write A and Av as follows:

$A =$

$$\begin{pmatrix} \text{a} & \text{b/2} \\ \text{b/2} & \text{c} \end{pmatrix}$$

$Av =$

$$\begin{pmatrix} \mathrm{d}/2 \\ \mathrm{e}/2 \end{pmatrix}$$

Since we have A and v, the resultant H^{-1} is the homography that can remove both the purely projective and purely affine distortions in one step.

2 Experiment

2.1 Task 1

The two images given to us are:

Figure 1: First Image

Figure 2: Second Image

The following are the points I used on each image to obtain the homographies:

Coordinates	Point 1	Point 2	Point 3	Point 4	Point 5	Point 6	Point 7
Image 1	(200,240)	(370,236)	(374,295)	(214,297)	(228,348)	(381,347)	(384,386)
Image 2	(181,77)	(653,79)	(621,805)	(221,804)	(152,56)	(680,57)	(641,827)

2.2 Task 2

The two images are the ones that I have taken:

Figure 3: Third Image

Figure 4: Fourth Image

The following are the points I used on each image to obtain the homographies:

Coordinates	**Point 1**	**Point 2**	**Point 3**	**Point 4**	**Point 5**	**Point 6**	**Point 7**
Image 3	(142,224)	(501,304)	(357,633)	(34,566)	(156,354)	(436,416)	(372,566)
Image 4	(95,76)	(202,259)	(105,454)	(31,269)	(143,212)	(171,262)	(123,364)

For both the tasks, I took the first 4 points to do part A as they form the vertices of a rectangle in the undistorted image. In addition to these 4 points, points 5,6,7 are used for part B and part C as they form 2 orthogonal lines intersecting at point 6

3 Results for Task 1

3.1 Using Point to Point Correspondences

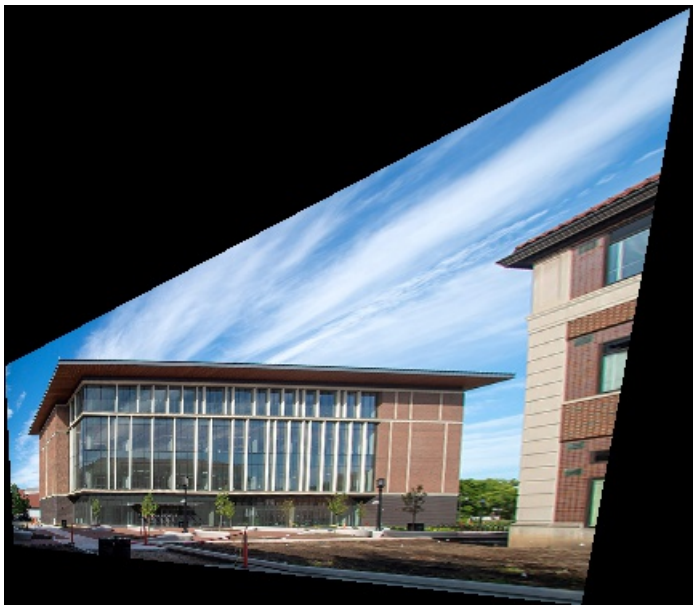


Figure 5: Undistorted Image 1

Figure 6: Undistorted Image 2

3.2 Using Two step Method

Figure 7: Image 1 without purely projective distortion

Figure 8: Image 2 without purely projective distortion

Figure 9: Image 1 without purely projective and affine distortion

Figure 10: Image 2 without purely projective and affine distortion

3.3 Using One step Method

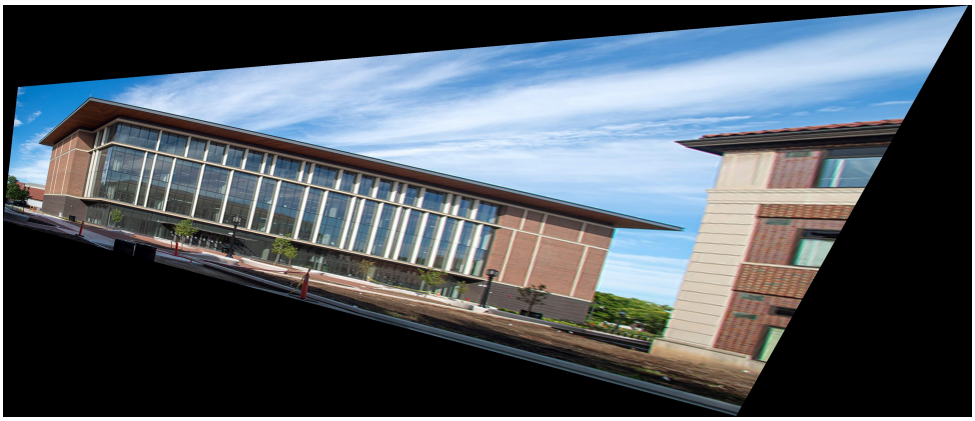


Figure 11: Undistorted Image 1

Figure 12: Undistorted Image 2

4 Results for Task 2

4.1 Using Point to Point Correspondences

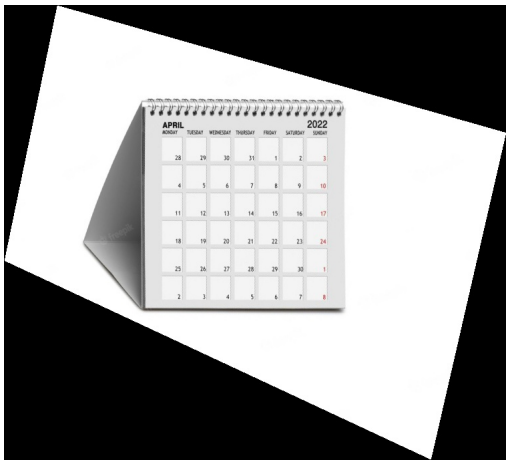


Figure 13: Undistorted Image 3

Figure 14: Undistorted Image 4

4.2 Using Two step Method

Figure 15: Image 3 without purely projective distortion

Figure 16: Image 4 without purely projective distortion

Figure 17: Image 3 without purely projective and affine distortion

Figure 18: Image 4 without purely projective and affine distortion

4.3 Using One step Method

Figure 19: Undistorted Image 3

Figure 20: Undistorted Image 4

5 Note

For the previous homework, the data type of domain coordinates obtained for the corresponding range coordinates using the homography was floating point. Since the indices of a matrix are whole numbers, the floating point coordinates won't have a corresponding RGB value. So I had just truncated the floating value to get an integer value and thereby get the RGB value. But this method leads to under-sampling and therefore is not good practice.

For this homework, I used bilinear interpolation to overcome this problem. It utilized the four integer coordinates around the floating point coordinates to obtain the corresponding RGB value for it.

In [1]:

```
#importing the libraries
import numpy as np
import cv2
import matplotlib.pyplot as plt
import math
```

In [38]:

```
#defining the ROI coordintes for all the 4 images
def coords(n):
    if n == 0: #for undistorted resultant image1
        return np.array([[0,0],[45,0],[45,15],[0,15]])

    if n == 1: #for distorted input image1
        return np.array([[200,240],[370,236],[374,295],[214,297]])

    if n == 2: #for undistorted resultant image2
        return np.array([[0,0],[85,0],[85,150],[0,150]])

    if n == 3: #for distorted input image2
        return np.array([[181,77],[653,79],[621,805],[221,804]])
```

In [39]:

```
#extra coordinates
def extracoords(n):
    if n == 1: #for distorted input image1
        return np.array([[200,240],[370,236],[374,295],[214,297],[228,348],[381,347],[384,3

    if n == 2: #for distorted input image2
        return np.array([[181,77],[653,79],[621,805],[221,804],[152,56],[680,57],[641,827]]
```

In [19]:

```
#drawing a box aroudnt the ROI
def box(rimg,rcord):
    rcord[:,[1,0]] = rcord[:,[0,1]]
    rcord = rcord.reshape(-1,1,2)
    cv2.polylines(rimg, pts = [rcord], isClosed = True, color =(255,0,0),thickness=1)
    plt.imshow(rimg)
    new_rimage = cv2.cvtColor(rimg,cv2.COLOR_RGB2BGR)
    cv2.imwrite("box.jpg",new_rimage)
```

In [20]:

```
#function to calculate the projective homography between two images
def proj_homography(dcord,rcord):
    A = np.zeros((8,8))
    B = np.zeros((8,1))
    H = np.ones((3,3))

    #Generating the matrix equation AB = C using the coordinates of the ROIs
    for i in range(4):
        A[2*i,0] = dcord[i,0]
        A[2*i,1] = dcord[i,1]
        A[2*i,2] = 1
        A[2*i,6] = -1*dcord[i,0]*rcord[i,0]
        A[2*i,7] = -1*dcord[i,1]*rcord[i,0]

        A[2*i+1,3] = dcord[i,0]
        A[2*i+1,4] = dcord[i,1]
        A[2*i+1,5] = 1
        A[2*i+1,6] = -1*dcord[i,0]*rcord[i,1]
        A[2*i+1,7] = -1*dcord[i,1]*rcord[i,1]

        B[2*i,0] = rcord[i,0]
        B[2*i+1,0] = rcord[i,1]


    C = np.dot(np.linalg.inv(A),B);

    #Obtaining the Homography H
    for i in range(3):
        for j in range(3):
            if i==2 & j ==2:
                break
            H[i,j]=C[3*i +j,0]

    return H
```

In [21]:

```
#Creating a mask of the domain image
def masks(rimg,rcord):
    img = np.zeros_like(rimg)
    rcord[:,[1,0]] = rcord[:,[0,1]]
    rcord = rcord.reshape(-1,1,2)
    cv2.fillPoly(img, pts = [rcord], color =(255,255,255))
    for i in range(rimg.shape[0]):
        for j in range(rimg.shape[1]):
            if img[i,j,0] == 0 and img[i,j,1] == 0 and img[i,j,2] == 0:
                continue
            img[i,j] = rimg[i,j]
    return img
```

In [22]:

```
#Performing bilinear interpolation to obtain the RGB values at a floating point coordinate
def bilinear_inter(dmask,x_bar):

    I1 = dmask[math.floor(x_bar[0]),math.floor(x_bar[1])]
    I2 = dmask[math.floor(x_bar[0]),math.ceil(x_bar[1])]
    I3 = dmask[math.ceil(x_bar[0]),math.ceil(x_bar[1])]
    I4 = dmask[math.ceil(x_bar[0]),math.floor(x_bar[1])]

    return (((math.ceil(x_bar[0])-x_bar[0])*I1 + (x_bar[0]-math.floor(x_bar[0]))*I4)*(math.
```

In [23]:

```
#Function to project the domain image onto the range image ROI using homography.
# Here the range coordinates are used to calculate the corresponding domain coordinates
def hom_mapping_bi(dimg,H,rcord,dmask):
    rimg = np.zeros((rcord[2,0],rcord[2,1],3),dtype=np.uint8)
    for i in range(rimg.shape[0]):
        for j in range(rimg.shape[1]):
            x  = np.array([i,j,1])
            x_bar = np.dot(H,x)
            x_bar[0] = x_bar[0]/x_bar[2]
            x_bar[1] = x_bar[1]/x_bar[2]
            i_bar = bilinear_inter(dmask,np.array([x_bar[0],x_bar[1]]))
            if  i_bar[0]!= 0 and i_bar[1]!= 0 and i_bar[2]!= 0 :
                rimg[i,j] = i_bar

    plt.imshow(rimg)
    return rimg
```

In [24]:

```
# Here the range coordinates are used to calculate the corresponding domain coordinates for
def hom_mapping_full(dimg,H):
    dcord = np.array([[0,0],[dimg.shape[0],0],[dimg.shape[0],dimg.shape[1]],[0,dimg.shape[1
    ones =  np.ones((dcord.shape[0],1))

    hdcord = np.append(dcord,ones,1)
    rcorners = np.ones((dcord.shape[0],2))
    for k in range(dcord.shape[0]):
        temp = np.dot(np.linalg.inv(H),hdcord[k])
        temp[0] = temp[0]/temp[2]
        temp[1] = temp[1]/temp[2]
        rcorners[k] = np.array([temp[0],temp[1]])

    xmin = np.min(rcorners,0)[0]
    ymin = np.min(rcorners,0)[1]
    xmax = np.max(rcorners,0)[0]
    ymax = np.max(rcorners,0)[1]

    rimg = np.zeros((math.floor(xmax-xmin),math.floor(ymax-ymin),3),dtype=np.uint8)

    #To calculate offset
    xoffset = -1*xmin
    yoffset = -1*ymin


    for i in range(rimg.shape[0]):
        for j in range(rimg.shape[1]):
            x  = np.array([i-xoffset,j-yoffset,1])
            x_bar = np.dot(H,x)
            x_bar[0] = x_bar[0]/x_bar[2]
            x_bar[1] = x_bar[1]/x_bar[2]

            if (x_bar[0]>0 and x_bar[1]>0 and x_bar[0]<dimg.shape[0]-1 and x_bar[1]<dimg.sh
                i_bar = bilinear_inter(dimg,np.array([x_bar[0],x_bar[1]]))
                rimg[i,j] = i_bar

    plt.imshow(rimg)
    return rimg
```

In [45]:

```
def range_image_size(dimg,H):
    dcord = np.array([[0,0],[dimg.shape[0],0],[dimg.shape[0],dimg.shape[1]],[0,dimg.shape[1
    ones =  np.ones((dcord.shape[0],1))

    hdcord = np.append(dcord,ones,1)
    rcorners = np.ones((dcord.shape[0],2))
    for k in range(dcord.shape[0]):
        temp = np.dot(np.linalg.inv(H),hdcord[k])
        temp[0] = temp[0]/temp[2]
        temp[1] = temp[1]/temp[2]
        rcorners[k] = np.array([temp[0],temp[1]])

    xmin = np.min(rcorners,0)[0]
    ymin = np.min(rcorners,0)[1]
    xmax = np.max(rcorners,0)[0]
    ymax = np.max(rcorners,0)[1]

    rimg = np.zeros((math.floor(xmax-xmin),math.floor(ymax-ymin),3),dtype=np.uint8)
    return rimg.shape
```

In [73]:

```
#Returns the HC of 5 pairs of orthogonal lines
def five_ortho_lines(dcord):

    ones =  np.ones((dcord.shape[0],1))
    hdcord = np.append(dcord,ones,1)
    l1 = np.cross(hdcord[0],hdcord[1])
    l2 = np.cross(hdcord[1],hdcord[2])
    l3 = np.cross(hdcord[2],hdcord[3])
    l4 = np.cross(hdcord[0],hdcord[3])
    l5 = np.cross(hdcord[4],hdcord[5])
    l6 = np.cross(hdcord[5],hdcord[6])
    l1 = l1/l1[2]
    l2 = l2/l2[2]
    l3 = l3/l3[2]
    l4 = l4/l4[2]
    l5 = l5/l5[2]
    l6 = l6/l6[2]
    ans = [[l1,l2],[l2,l3],[l3,l4],[l1,l4],[l5,l6]]
    return ans
```

In [26]:

```
#Returns the HC of the vanishing line
def vanishing_line(dcord):
    ones =  np.ones((dcord.shape[0],1))
    hdcord = np.append(dcord,ones,1)
    l1 = np.cross(hdcord[0],hdcord[1])
    l2 = np.cross(hdcord[1],hdcord[2])
    l3 = np.cross(hdcord[2],hdcord[3])
    l4 = np.cross(hdcord[3],hdcord[0])
    p1 = np.cross(l1,l3)
    p2 = np.cross(l2,l4)
    vl = np.cross(p1,p2)
    return vl
```

In [27]:

```
#Function that returns H to remove purely projective distortion for two step method
def two_homography_a(dcord):
    vl = vanishing_line(dcord)
    vl = vl/vl[2]
    H = np.eye(3)
    for i in range(3):
        H[2,i] = vl[i]
    return H
```

In [91]:

```
#Function that returns H to remove purely affine distortion for two step method
def two_homography_b(dcord,H_old):
    ones =  np.ones((dcord.shape[0],1))
    hdcord = np.append(dcord,ones,1)
    new_hdcord = np.zeros_like(hdcord)
    for i in range(7):
        new_hdcord[i] = np.dot(H_old,hdcord[i])

    l1 = np.cross(new_hdcord[0],new_hdcord[1])
    l2 = np.cross(new_hdcord[1],new_hdcord[2])
    l3 = np.cross(new_hdcord[2],new_hdcord[3])
    l4 = np.cross(new_hdcord[3],new_hdcord[0])
    l5 = np.cross(new_hdcord[4],new_hdcord[5])
    l6 = np.cross(new_hdcord[5],new_hdcord[6])

    l1 = l1/l1[2]
    l2 = l2/l2[2]
    l3 = l3/l3[2]
    l4 = l4/l4[2]
    l5 = l5/l5[2]
    l6 = l6/l6[2]

    A = np.zeros((5,2))
    B = np.zeros((5,1))
    S = np.ones((2,2))
    H = np.zeros((3,3))

    A[0,0] = l1[0]*l2[0]
    A[0,1] = l1[1]*l2[0] + l1[0]*l2[1]
    A[1,0] = l2[0]*l3[0]
    A[1,1] = l2[1]*l3[0] + l2[0]*l3[1]
    A[2,0] = l3[0]*l4[0]
    A[2,1] = l3[1]*l4[0] + l3[0]*l4[1]
    A[3,0] = l3[0]*l4[0]
    A[3,1] = l3[1]*l4[0] + l3[0]*l4[1]
    A[4,0] = l5[0]*l6[0]
    A[4,1] = l5[1]*l6[0] + l5[0]*l6[1]

    B[0] = -1*l1[1]*l2[1]
    B[1] = -1*l2[1]*l3[1]
    B[2] = -1*l3[1]*l4[1]
    B[3] = -1*l4[1]*l5[1]
    B[4] = -1*l5[1]*l6[1]


    C = np.linalg.pinv(A)@B;
    C = C/np.abs(C).max()


    S[0,0] = C[0]
    S[0,1] = C[1]
    S[1,0] = C[1]

    ve,w,temp = np.linalg.svd(S)
    wpos = w
    a = ve@np.diag(np.sqrt(wpos))@np.linalg.inv(ve)

    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 [97]:

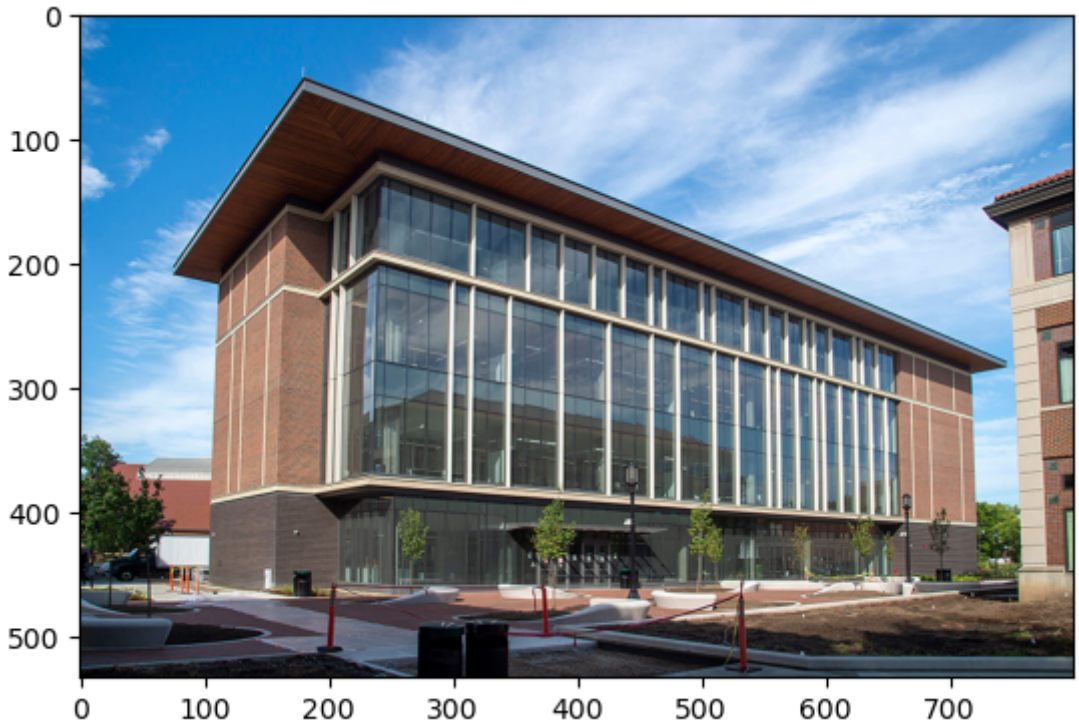
```
#Function that returns H for one step method
def one_homography(dcord):
    ans = five_ortho_lines(dcord)
    A = np.zeros((5,5))
    B = np.zeros((5,1))
    aat = np.ones((2,2))
    av = np.ones((2,1))
    H = np.zeros((3,3))

    for i in range(5):
        A[i,0] = ans[i][0][0]*ans[i][1][0]
        A[i,1] = (ans[i][0][1]*ans[i][1][0] + ans[i][0][0]*ans[i][1][1])
        A[i,2] = ans[i][0][1]*ans[i][1][1]
        A[i,3] = (ans[i][0][2]*ans[i][1][0] + ans[i][0][0]*ans[i][1][2])
        A[i,4] = (ans[i][0][2]*ans[i][1][1] + ans[i][0][1]*ans[i][1][2])
        B[i,0] = -(ans[i][0][2]*ans[i][1][2])

    res = np.dot(np.linalg.pinv(A),B);
    res = res/np.abs(res).max()

    aat[0,0] = res[0]
    aat[0,1] = res[1]
    aat[1,0] = res[1]
    aat[1,1] = res[2]

    av[0,0] = res[3]
    av[1,0] = res[4]

    ve,w,temp = np.linalg.svd(aat)
    wpos = w
    a = ve@np.diag(np.sqrt(wpos))@ve.T
    v = np.dot(np.linalg.pinv(a),av)

    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

    return H
```

In [41]:

```
#Displaying the first image
dimg1 = cv2.imread ("building.jpg")
dimage1 = cv2.cvtColor(dimg1, cv2.COLOR_BGR2RGB)
rc1 = coords(0)
dc1 = coords(1)
dc1c = extracoords(1)
plt.imshow(dimage1)
```

Out[41]:

```
<matplotlib.image.AxesImage at 0x2e6ce3fb8b0>
```



In [42]:

```
#Displaying the second image
dimg2 = cv2.imread ("nighthawks.jpg")
dimage2 = cv2.cvtColor(dimg2, cv2.COLOR_BGR2RGB)
rc2 = coords(2)
dc2 = coords(3)
dc2c = extracoords(2)
plt.imshow(dimage2)
```

Out[42]:

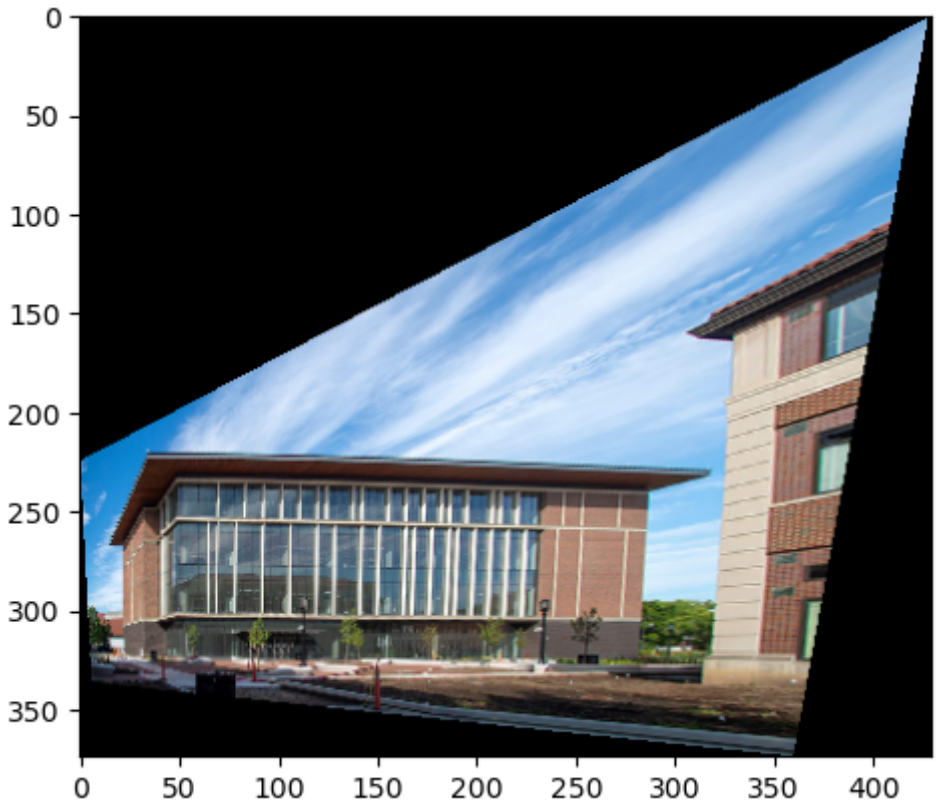
```
<matplotlib.image.AxesImage at 0x2e6ce461940>
```

In [43]:

```
#Task 1a part i
H1a = proj_homography(dc1,rc1) #Calculating the homography
dmask1 = masks(dimage1,dc1)  #Creating the mask
new_rimage1a = hom_mapping_full(dimage1.copy(),np.linalg.inv(H1a)) #Generating the projecti
new_rimage1a = cv2.cvtColor(new_rimage1a,cv2.COLOR_RGB2BGR)
cv2.imwrite("PTtoPT_image1.jpg",new_rimage1a)
```

Out[43]:

```
True
```



In [33]:

```
#Task 1a part ii
H2a = proj_homography(dc2,rc2) #Calculating the homography
dmask2 = masks(dimage2,dc2)  #Creating the mask
new_rimage2a = hom_mapping_full(dimage2.copy(),np.linalg.inv(H2a)) #Generating the projecti
new_rimage2a = cv2.cvtColor(new_rimage2a,cv2.COLOR_RGB2BGR)
cv2.imwrite("PTtoPT_image2.jpg",new_rimage2a)
```

Out[33]:

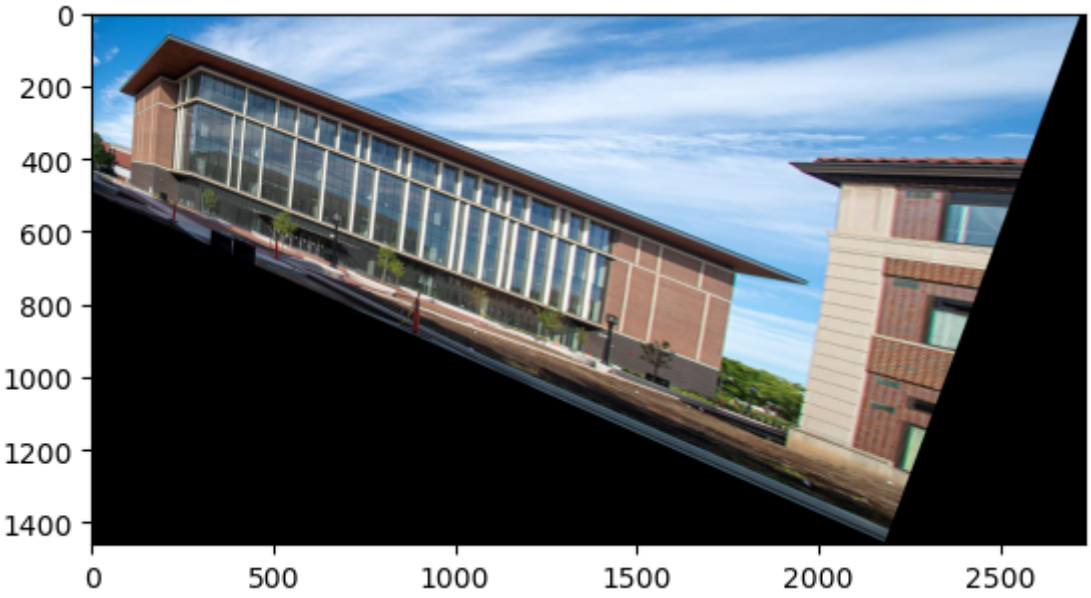
True

In [44]:

```
#Task 1b part i a
H1b = two_homography_a(dc1c) #Calculating the homography
new_rimage1b = hom_mapping_full(dimage1.copy(),np.linalg.inv(H1b)) #Generating the projecti
new_rimage1b = cv2.cvtColor(new_rimage1b,cv2.COLOR_RGB2BGR)
cv2.imwrite("twoa_image1.jpg",new_rimage1b)
```

Out[44]:

```
True
```



In [49]:

```
#Task 1b part i b

H2b = two_homography_a(dc2c) #Calculating the homography
new_rimage2b = hom_mapping_full(dimage2.copy(),np.linalg.inv(H2b)) #Generating the projecti
new_rimage2b = cv2.cvtColor(new_rimage2b,cv2.COLOR_RGB2BGR)
cv2.imwrite("twoa_image2.jpg",new_rimage2b)
```

Out[49]:

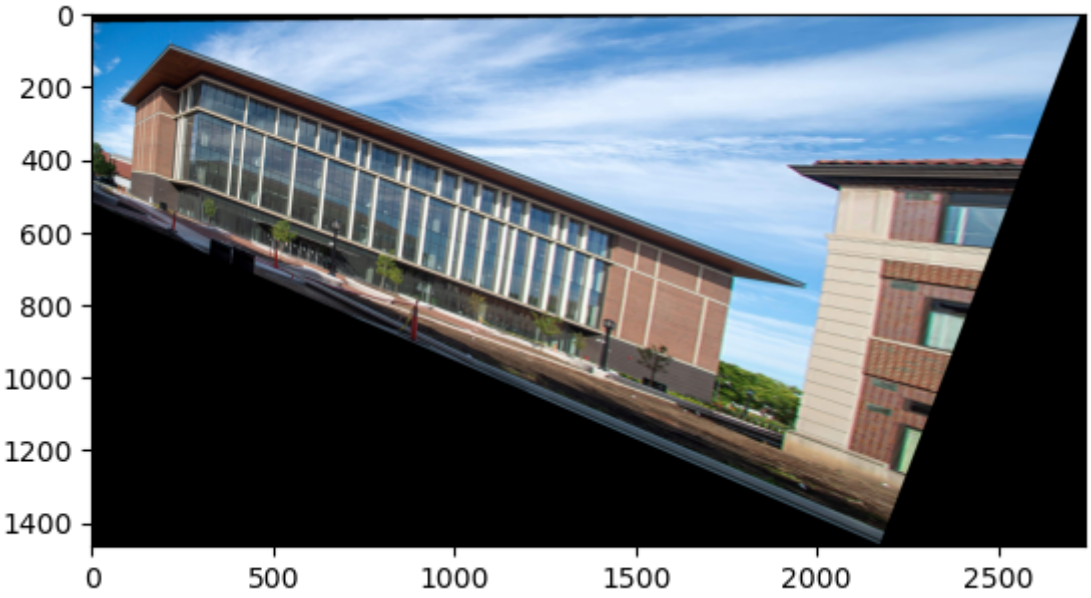
True

In [93]:

```
#Task 1b part ii a
H1bb = two_homography_b(dc1c,H1b)
new_rimage1bb = hom_mapping_full(dimage1.copy(),np.linalg.inv(H1bb@H1b)) #Generating the pr
new_rimage1bb = cv2.cvtColor(new_rimage1bb,cv2.COLOR_RGB2BGR)
cv2.imwrite("twob_image1.jpg",new_rimage1bb)
```

Out[93]:

```
True
```



In [92]:

```
#Task 1b part ii b
H2bb = two_homography_b(dc2c,H2b)
new_rimage2bb = hom_mapping_full(dimage2.copy(),np.linalg.inv(H2bb@H2b)) #Generating the pr
new_rimage2bb = cv2.cvtColor(new_rimage2bb,cv2.COLOR_RGB2BGR)
cv2.imwrite("twob_image2.jpg",new_rimage2bb)
```

Out[92]:

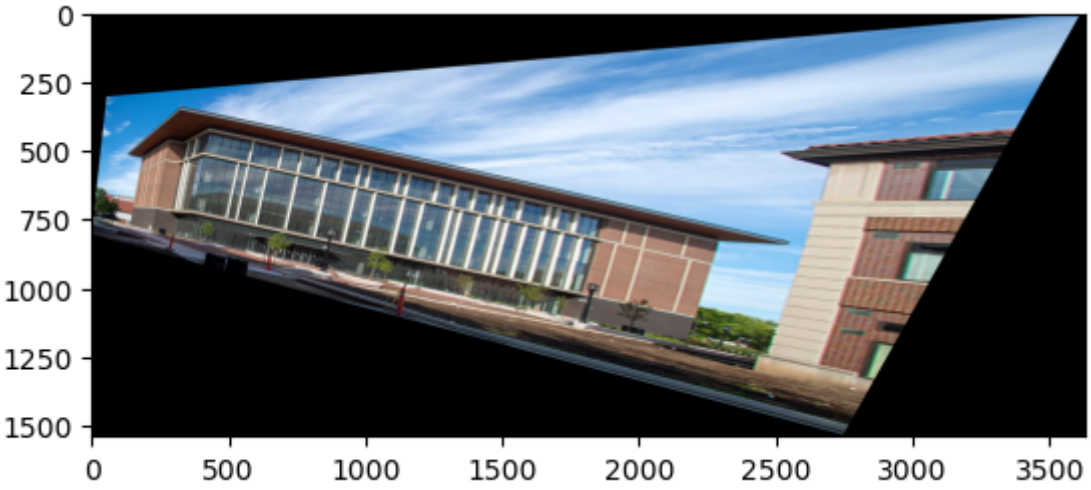
True

In [104]:

```
#Task 1c part i
H1c = one_homography(dc1c) #Calculating the homography
new_rimage1 = hom_mapping_full(dimage1.copy(),H1c) #Generating the projection
new_rimage1 = cv2.cvtColor(new_rimage1,cv2.COLOR_RGB2BGR)
cv2.imwrite("one_image1.jpg",new_rimage1)
```

Out[104]:

```
True
```

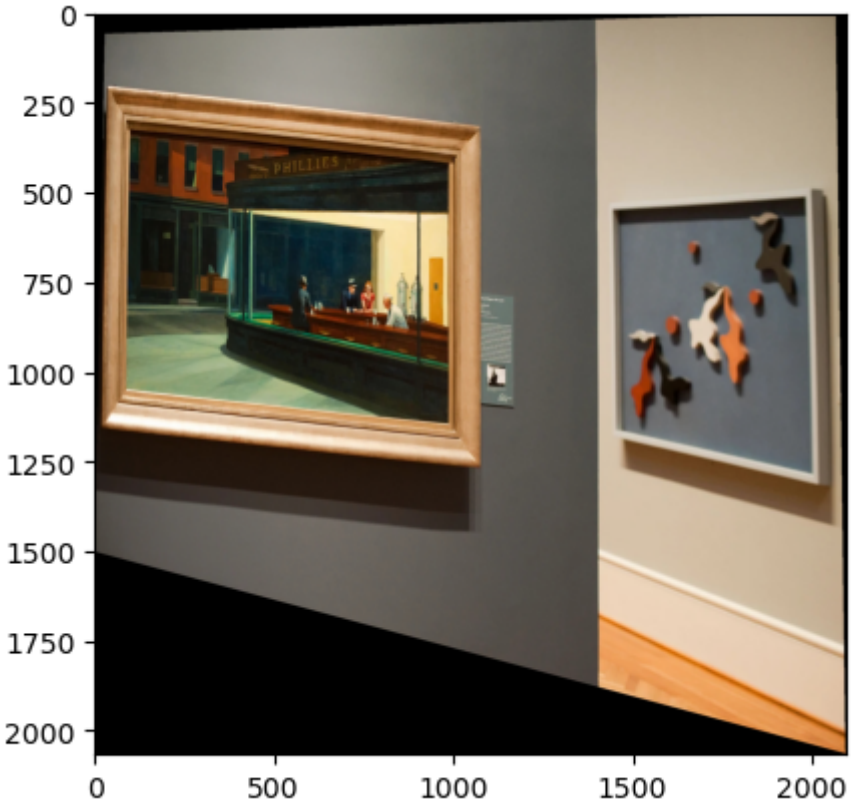


In [101]:

```
#Task 1c part ii
H2c = one_homography(dc2c) #Calculating the homography
new_rimage2 = hom_mapping_full(dimage2.copy(),H2c) #Generating the projection
new_rimage2 = cv2.cvtColor(new_rimage2,cv2.COLOR_RGB2BGR)
cv2.imwrite("one_image2.jpg",new_rimage2)
```

Out[101]:

True



In [ ]:

In [2]:

```
#importing the libraries
import numpy as np
import cv2
import matplotlib.pyplot as plt
import math
```

In [42]:

```
#defining the ROI coordintes for all the 4 images
def coords(n):
    if n == 0: #for undistorted resultant image1
        return np.array([[0,0],[300,0],[300,300],[0,300]])

    if n == 1: #for distorted input image1
        return np.array([[142,224],[501,304],[357,633],[34,566]])

    if n == 2: #for undistorted resultant image2
        return np.array([[0,0],[85,0],[85,85],[0,85]])

    if n == 3: #for distorted input image2
        return np.array([[95,76],[202,259],[105,454],[31,269]])
```

In [46]:

```
#extra coordinates
def extracoords(n):
    if n == 1: #for distorted input image1
        return np.array([[142,224],[501,304],[357,633],[34,566],[156,354],[436,416],[372,56

    if n == 2: #for distorted input image2
        return np.array([[95,76],[202,259],[105,454],[31,269],[143,212],[171,262],[123,364]
```

In [5]:

```
#drawing a box aroudnt the ROI
def box(rimg,rcord):
    rcord[:,[1,0]] = rcord[:,[0,1]]
    rcord = rcord.reshape(-1,1,2)
    cv2.polylines(rimg, pts = [rcord], isClosed = True, color =(255,0,0),thickness=1)
    plt.imshow(rimg)
    new_rimage = cv2.cvtColor(rimg,cv2.COLOR_RGB2BGR)
    cv2.imwrite("box.jpg",new_rimage)
```

In [6]:

```
#function to calculate the projective homography between two images
def proj_homography(dcord,rcord):
    A = np.zeros((8,8))
    B = np.zeros((8,1))
    H = np.ones((3,3))

    #Generating the matrix equation AB = C using the coordinates of the ROIs
    for i in range(4):
        A[2*i,0] = dcord[i,0]
        A[2*i,1] = dcord[i,1]
        A[2*i,2] = 1
        A[2*i,6] = -1*dcord[i,0]*rcord[i,0]
        A[2*i,7] = -1*dcord[i,1]*rcord[i,0]

        A[2*i+1,3] = dcord[i,0]
        A[2*i+1,4] = dcord[i,1]
        A[2*i+1,5] = 1
        A[2*i+1,6] = -1*dcord[i,0]*rcord[i,1]
        A[2*i+1,7] = -1*dcord[i,1]*rcord[i,1]

        B[2*i,0] = rcord[i,0]
        B[2*i+1,0] = rcord[i,1]


    C = np.dot(np.linalg.inv(A),B);

    #Obtaining the Homography H
    for i in range(3):
        for j in range(3):
            if i==2 & j ==2:
                break
            H[i,j]=C[3*i +j,0]

    return H
```

In [7]:

```
#Creating a mask of the domain image
def masks(rimg,rcord):
    img = np.zeros_like(rimg)
    rcord[:,[1,0]] = rcord[:,[0,1]]
    rcord = rcord.reshape(-1,1,2)
    cv2.fillPoly(img, pts = [rcord], color =(255,255,255))
    for i in range(rimg.shape[0]):
        for j in range(rimg.shape[1]):
            if img[i,j,0] == 0 and img[i,j,1] == 0 and img[i,j,2] == 0:
                continue
            img[i,j] = rimg[i,j]
    return img
```

In [8]:

```
#Performing bilinear interpolation to obtain the RGB values at a floating point coordinate
def bilinear_inter(dmask,x_bar):

    I1 = dmask[math.floor(x_bar[0]),math.floor(x_bar[1])]
    I2 = dmask[math.floor(x_bar[0]),math.ceil(x_bar[1])]
    I3 = dmask[math.ceil(x_bar[0]),math.ceil(x_bar[1])]
    I4 = dmask[math.ceil(x_bar[0]),math.floor(x_bar[1])]

    return (((math.ceil(x_bar[0])-x_bar[0])*I1 + (x_bar[0]-math.floor(x_bar[0]))*I4)*(math.
```

In [9]:

```
#Function to project the domain image onto the range image ROI using homography.
# Here the range coordinates are used to calculate the corresponding domain coordinates
def hom_mapping_bi(dimg,H,rcord,dmask):
    rimg = np.zeros((rcord[2,0],rcord[2,1],3),dtype=np.uint8)
    for i in range(rimg.shape[0]):
        for j in range(rimg.shape[1]):
            x  = np.array([i,j,1])
            x_bar = np.dot(H,x)
            x_bar[0] = x_bar[0]/x_bar[2]
            x_bar[1] = x_bar[1]/x_bar[2]
            i_bar = bilinear_inter(dmask,np.array([x_bar[0],x_bar[1]]))
            if  i_bar[0]!= 0 and i_bar[1]!= 0 and i_bar[2]!= 0 :
                rimg[i,j] = i_bar

    plt.imshow(rimg)
    return rimg
```

In [10]:

```
# Here the range coordinates are used to calculate the corresponding domain coordinates
def hom_mapping_full(dimg,H):
    dcord = np.array([[0,0],[dimg.shape[0],0],[dimg.shape[0],dimg.shape[1]],[0,dimg.shape[1
    ones =  np.ones((dcord.shape[0],1))

    hdcord = np.append(dcord,ones,1)
    rcorners = np.ones((dcord.shape[0],2))
    for k in range(dcord.shape[0]):
        temp = np.dot(np.linalg.inv(H),hdcord[k])
        temp[0] = temp[0]/temp[2]
        temp[1] = temp[1]/temp[2]
        rcorners[k] = np.array([temp[0],temp[1]])

    xmin = np.min(rcorners,0)[0]
    ymin = np.min(rcorners,0)[1]
    xmax = np.max(rcorners,0)[0]
    ymax = np.max(rcorners,0)[1]

    rimg = np.zeros((math.floor(xmax-xmin),math.floor(ymax-ymin),3),dtype=np.uint8)

    #To calculate offset
    xoffset = -1*xmin
    yoffset = -1*ymin


    for i in range(rimg.shape[0]):
        for j in range(rimg.shape[1]):
            x  = np.array([i-xoffset,j-yoffset,1])
            x_bar = np.dot(H,x)
            x_bar[0] = x_bar[0]/x_bar[2]
            x_bar[1] = x_bar[1]/x_bar[2]

            if (x_bar[0]>0 and x_bar[1]>0 and x_bar[0]<dimg.shape[0]-1 and x_bar[1]<dimg.sh
                i_bar = bilinear_inter(dimg,np.array([x_bar[0],x_bar[1]]))
                rimg[i,j] = i_bar

    plt.imshow(rimg)
    return rimg
```

In [11]:

```
def range_image_size(dimg,H):
    dcord = np.array([[0,0],[dimg.shape[0],0],[dimg.shape[0],dimg.shape[1]],[0,dimg.shape[1
    ones =  np.ones((dcord.shape[0],1))

    hdcord = np.append(dcord,ones,1)
    rcorners = np.ones((dcord.shape[0],2))
    for k in range(dcord.shape[0]):
        temp = np.dot(np.linalg.inv(H),hdcord[k])
        temp[0] = temp[0]/temp[2]
        temp[1] = temp[1]/temp[2]
        rcorners[k] = np.array([temp[0],temp[1]])

    xmin = np.min(rcorners,0)[0]
    ymin = np.min(rcorners,0)[1]
    xmax = np.max(rcorners,0)[0]
    ymax = np.max(rcorners,0)[1]

    rimg = np.zeros((math.floor(xmax-xmin),math.floor(ymax-ymin),3),dtype=np.uint8)
    return rimg.shape
```

In [12]:

```
#Returns the HC of 5 pairs of orthogonal lines
def five_ortho_lines(dcord):

    ones =  np.ones((dcord.shape[0],1))
    hdcord = np.append(dcord,ones,1)
    l1 = np.cross(hdcord[0],hdcord[1])
    l2 = np.cross(hdcord[1],hdcord[2])
    l3 = np.cross(hdcord[2],hdcord[3])
    l4 = np.cross(hdcord[0],hdcord[3])
    l5 = np.cross(hdcord[4],hdcord[5])
    l6 = np.cross(hdcord[5],hdcord[6])
    l1 = l1/l1[2]
    l2 = l2/l2[2]
    l3 = l3/l3[2]
    l4 = l4/l4[2]
    l5 = l5/l5[2]
    l6 = l6/l6[2]
    ans = [[l1,l2],[l2,l3],[l3,l4],[l1,l4],[l5,l6]]
    return ans
```

In [13]:

```
#Returns the HC of the vanishing line
def vanishing_line(dcord):
    ones =  np.ones((dcord.shape[0],1))
    hdcord = np.append(dcord,ones,1)
    l1 = np.cross(hdcord[0],hdcord[1])
    l2 = np.cross(hdcord[1],hdcord[2])
    l3 = np.cross(hdcord[2],hdcord[3])
    l4 = np.cross(hdcord[3],hdcord[0])
    p1 = np.cross(l1,l3)
    p2 = np.cross(l2,l4)
    vl = np.cross(p1,p2)
    return vl
```

In [14]:

```
#Function that returns H to remove purely projective distortion for two step method
def two_homography_a(dcord):
    vl = vanishing_line(dcord)
    vl = vl/vl[2]
    H = np.eye(3)
    for i in range(3):
        H[2,i] = vl[i]
    return H
```

In [15]:

```
#Function that returns H to remove purely affine distortion for two step method
def two_homography_b(dcord,H_old):
    ones =  np.ones((dcord.shape[0],1))
    hdcord = np.append(dcord,ones,1)
    new_hdcord = np.zeros_like(hdcord)
    for i in range(7):
        new_hdcord[i] = np.dot(H_old,hdcord[i])

    l1 = np.cross(new_hdcord[0],new_hdcord[1])
    l2 = np.cross(new_hdcord[1],new_hdcord[2])
    l3 = np.cross(new_hdcord[2],new_hdcord[3])
    l4 = np.cross(new_hdcord[3],new_hdcord[0])
    l5 = np.cross(new_hdcord[4],new_hdcord[5])
    l6 = np.cross(new_hdcord[5],new_hdcord[6])

    l1 = l1/l1[2]
    l2 = l2/l2[2]
    l3 = l3/l3[2]
    l4 = l4/l4[2]
    l5 = l5/l5[2]
    l6 = l6/l6[2]

    A = np.zeros((5,2))
    B = np.zeros((5,1))
    S = np.ones((2,2))
    H = np.zeros((3,3))

    A[0,0] = l1[0]*l2[0]
    A[0,1] = l1[1]*l2[0] + l1[0]*l2[1]
    A[1,0] = l2[0]*l3[0]
    A[1,1] = l2[1]*l3[0] + l2[0]*l3[1]
    A[2,0] = l3[0]*l4[0]
    A[2,1] = l3[1]*l4[0] + l3[0]*l4[1]
    A[3,0] = l3[0]*l4[0]
    A[3,1] = l3[1]*l4[0] + l3[0]*l4[1]
    A[4,0] = l5[0]*l6[0]
    A[4,1] = l5[1]*l6[0] + l5[0]*l6[1]

    B[0] = -1*l1[1]*l2[1]
    B[1] = -1*l2[1]*l3[1]
    B[2] = -1*l3[1]*l4[1]
    B[3] = -1*l4[1]*l5[1]
    B[4] = -1*l5[1]*l6[1]


    C = np.linalg.pinv(A)@B;
    C = C/np.abs(C).max()


    S[0,0] = C[0]
    S[0,1] = C[1]
    S[1,0] = C[1]

    ve,w,temp = np.linalg.svd(S)
    wpos = w
    a = ve@np.diag(np.sqrt(wpos))@np.linalg.inv(ve)

    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 [36]:

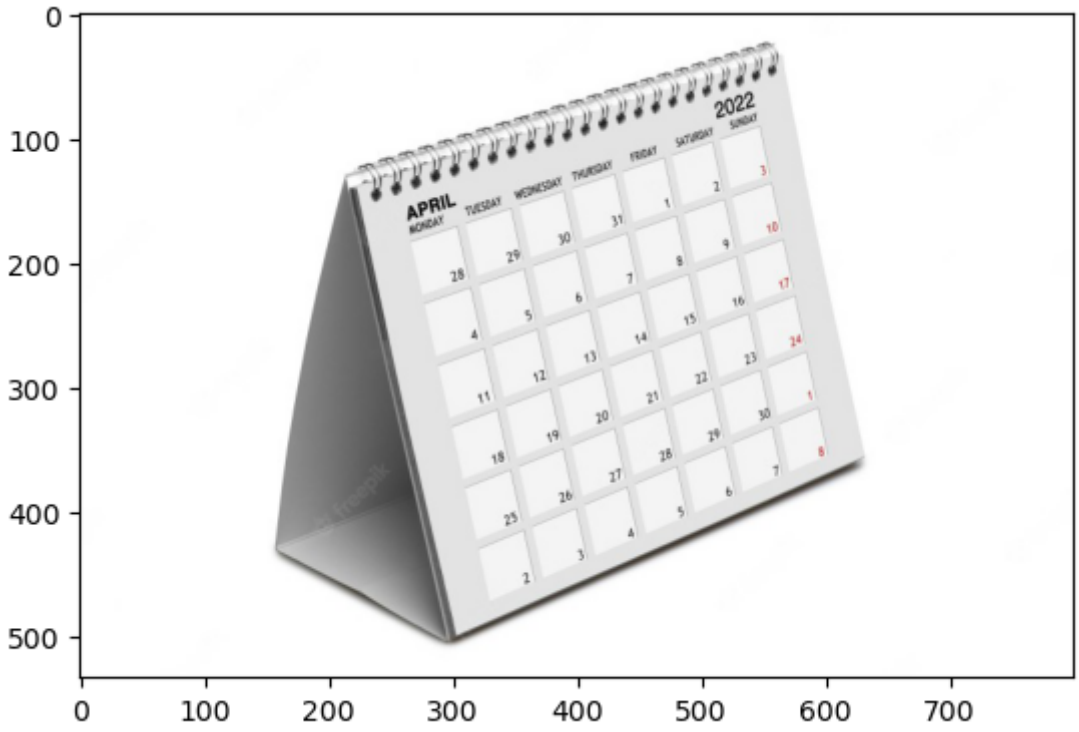
```
#Function that returns H for one step method
def one_homography(dcord):
    ans = five_ortho_lines(dcord)
    A = np.zeros((5,5))
    B = np.zeros((5,1))
    aat = np.ones((2,2))
    av = np.ones((2,1))
    H = np.zeros((3,3))

    for i in range(5):
        A[i,0] = ans[i][0][0]*ans[i][1][0]
        A[i,1] = (ans[i][0][1]*ans[i][1][0] + ans[i][0][0]*ans[i][1][1])
        A[i,2] = ans[i][0][1]*ans[i][1][1]
        A[i,3] = (ans[i][0][2]*ans[i][1][0] + ans[i][0][0]*ans[i][1][2])
        A[i,4] = (ans[i][0][2]*ans[i][1][1] + ans[i][0][1]*ans[i][1][2])
        B[i,0] = -(ans[i][0][2]*ans[i][1][2])

    res = np.dot(np.linalg.pinv(A),B);
    res = res/np.abs(res).max()

    aat[0,0] = res[0]
    aat[0,1] = res[1]
    aat[1,0] = res[1]
    aat[1,1] = res[2]

    av[0,0] = res[3]
    av[1,0] = res[4]

    ve,w,temp = np.linalg.svd(aat)
    wpos = w
    a = ve@np.diag(np.sqrt(wpos))@ve.T
    v = np.dot(np.linalg.pinv(a),av)

    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

    return H
```

In [47]:

```
#Displaying the first image
dimg1 = cv2.imread ("Image1.jpg")
dimage1 = cv2.cvtColor(dimg1, cv2.COLOR_BGR2RGB)
rc1 = coords(0)
dc1 = coords(1)
dc1c = extracoords(1)
plt.imshow(dimage1)
```

Out[47]:

```
<matplotlib.image.AxesImage at 0x2424979a610>
```



In [30]:

```
#Displaying the second image
dimg2 = cv2.imread ("Image3.jpg")
dimage2 = cv2.cvtColor(dimg2, cv2.COLOR_BGR2RGB)
rc2 = coords(2)
dc2 = coords(3)
dc2c = extracoords(2)
plt.imshow(dimage2)
```

Out[30]:

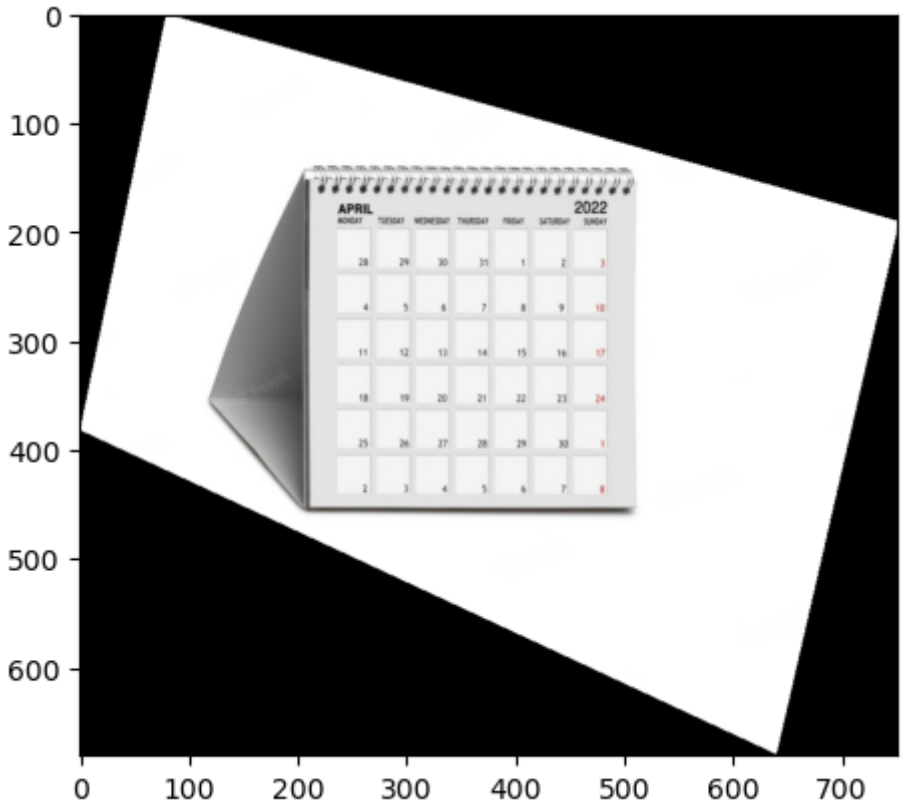
```
<matplotlib.image.AxesImage at 0x241aac3a8e0>
```

In [48]:

```
#Task 2a part i
H1a = proj_homography(dc1,rc1) #Calculating the homography
dmask1 = masks(dimage1,dc1)  #Creating the mask
new_rimage1a = hom_mapping_full(dimage1.copy(),np.linalg.inv(H1a)) #Generating the projecti
new_rimage1a = cv2.cvtColor(new_rimage1a,cv2.COLOR_RGB2BGR)
cv2.imwrite("PTtoPT_image1.jpg",new_rimage1a)
```

Out[48]:

```
True
```

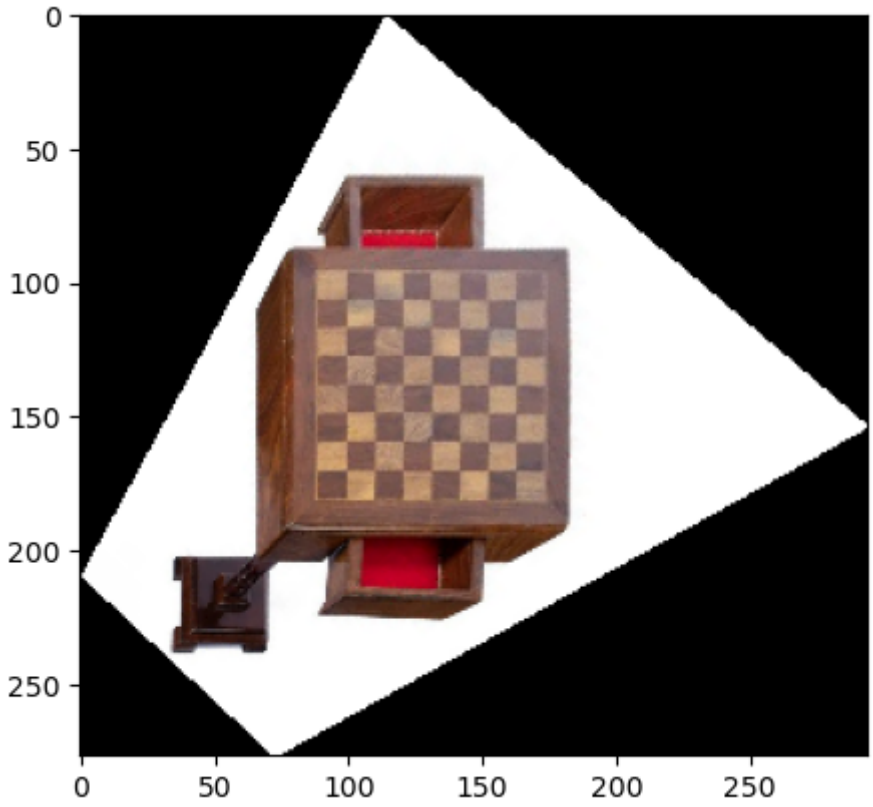


In [31]:

```
#Task 2a part ii
H2a = proj_homography(dc2,rc2) #Calculating the homography
dmask2 = masks(dimage2,dc2)  #Creating the mask
new_rimage2a = hom_mapping_full(dimage2.copy(),np.linalg.inv(H2a)) #Generating the projecti
new_rimage2a = cv2.cvtColor(new_rimage2a,cv2.COLOR_RGB2BGR)
cv2.imwrite("PTtoPT_image2.jpg",new_rimage2a)
```

Out[31]:

True

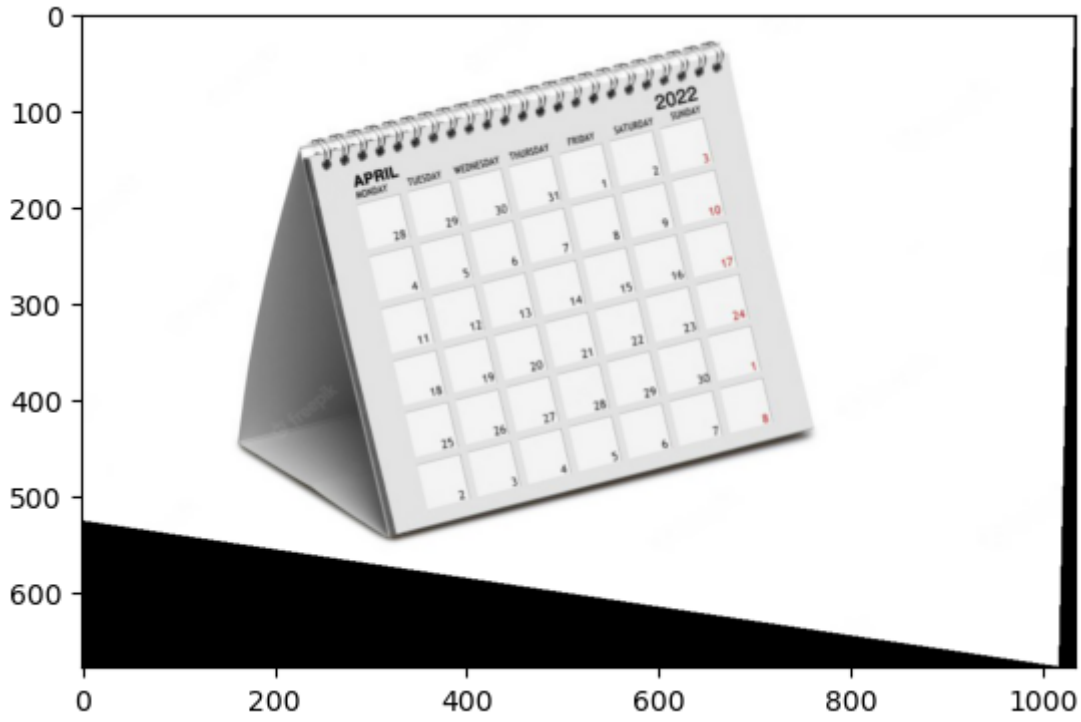


In [49]:

```
#Task 2b part i a
H1b = two_homography_a(dc1c) #Calculating the homography
new_rimage1b = hom_mapping_full(dimage1.copy(),np.linalg.inv(H1b)) #Generating the projecti
new_rimage1b = cv2.cvtColor(new_rimage1b,cv2.COLOR_RGB2BGR)
cv2.imwrite("twoa_image1.jpg",new_rimage1b)
```

Out[49]:

```
True
```

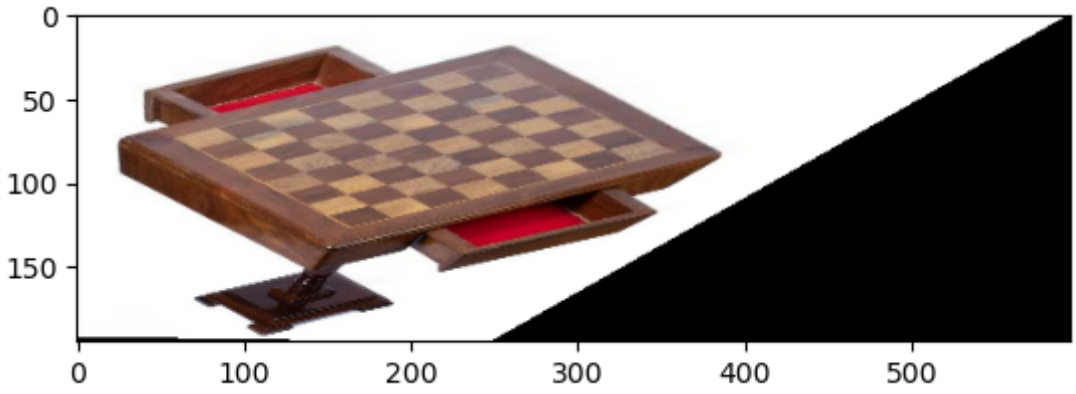


In [32]:

```
#Task 2b part i b

H2b = two_homography_a(dc2c) #Calculating the homography
new_rimage2b = hom_mapping_full(dimage2.copy(),np.linalg.inv(H2b)) #Generating the projecti
new_rimage2b = cv2.cvtColor(new_rimage2b,cv2.COLOR_RGB2BGR)
cv2.imwrite("twoa_image2.jpg",new_rimage2b)
```

Out[32]:

```
True
```

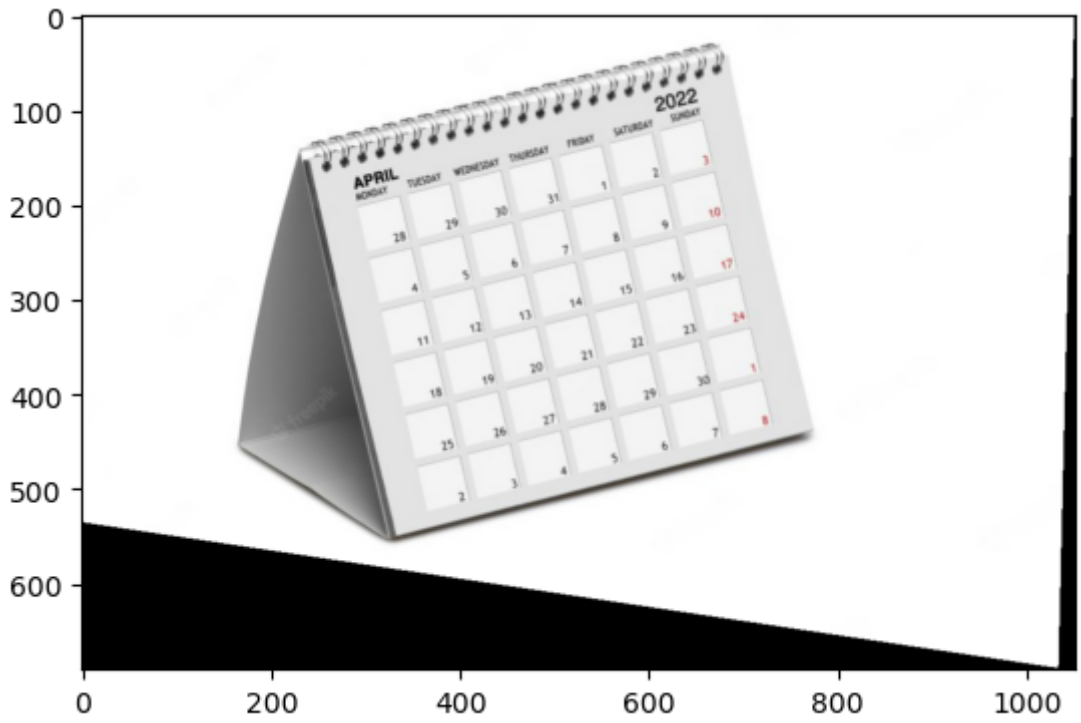


In [50]:

```
#Task 2b part ii a
H1bb = two_homography_b(dc1c,H1b)
new_rimage1bb = hom_mapping_full(dimage1.copy(),np.linalg.inv(H1bb@H1b)) #Generating the pr
new_rimage1bb = cv2.cvtColor(new_rimage1bb,cv2.COLOR_RGB2BGR)
cv2.imwrite("twob_image1.jpg",new_rimage1bb)
```

Out[50]:

```
True
```

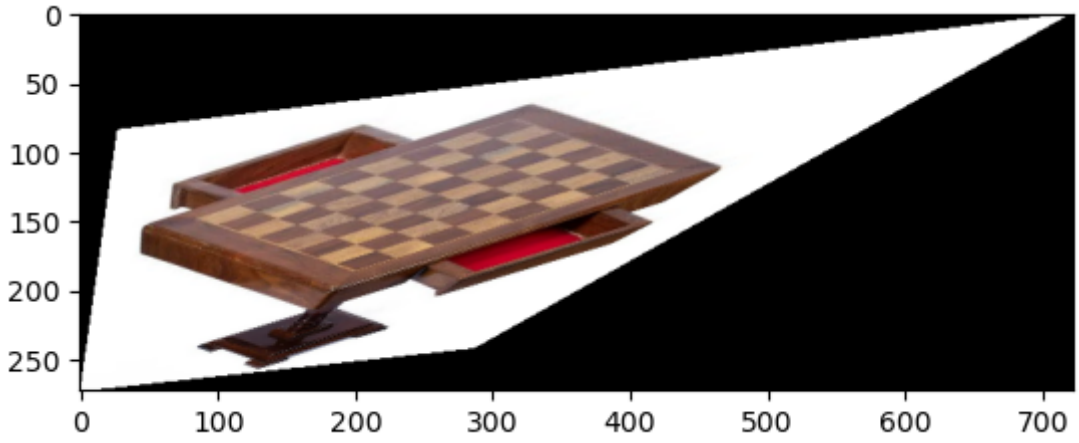


In [33]:

```
#Task 2b part ii b
H2bb = two_homography_b(dc2c,H2b)
new_rimage2bb = hom_mapping_full(dimage2.copy(),np.linalg.inv(H2bb@H2b)) #Generating the pr
new_rimage2bb = cv2.cvtColor(new_rimage2bb,cv2.COLOR_RGB2BGR)
cv2.imwrite("twob_image2.jpg",new_rimage2bb)
```

Out[33]:

True

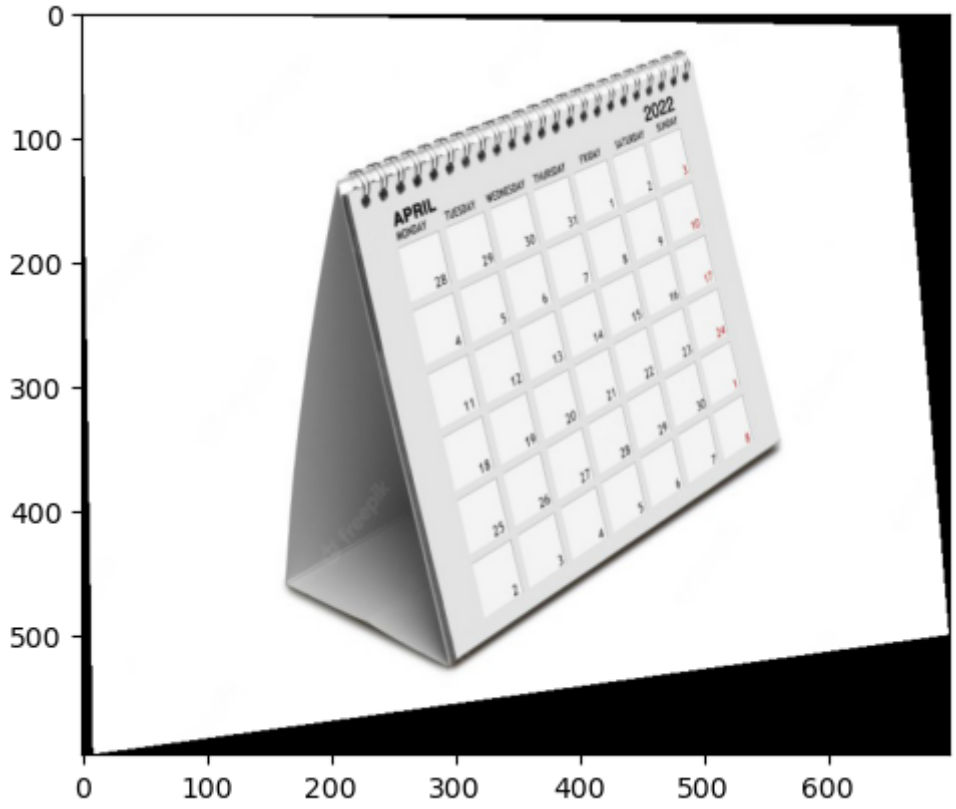


In [51]:

```
#Task 2c part i
H1c = one_homography(dc1c) #Calculating the homography
new_rimage1 = hom_mapping_full(dimage1.copy(),np.linalg.pinv(H1c)) #Generating the projecti
new_rimage1 = cv2.cvtColor(new_rimage1,cv2.COLOR_RGB2BGR)
cv2.imwrite("one_image1.jpg",new_rimage1)
```

Out[51]:

True

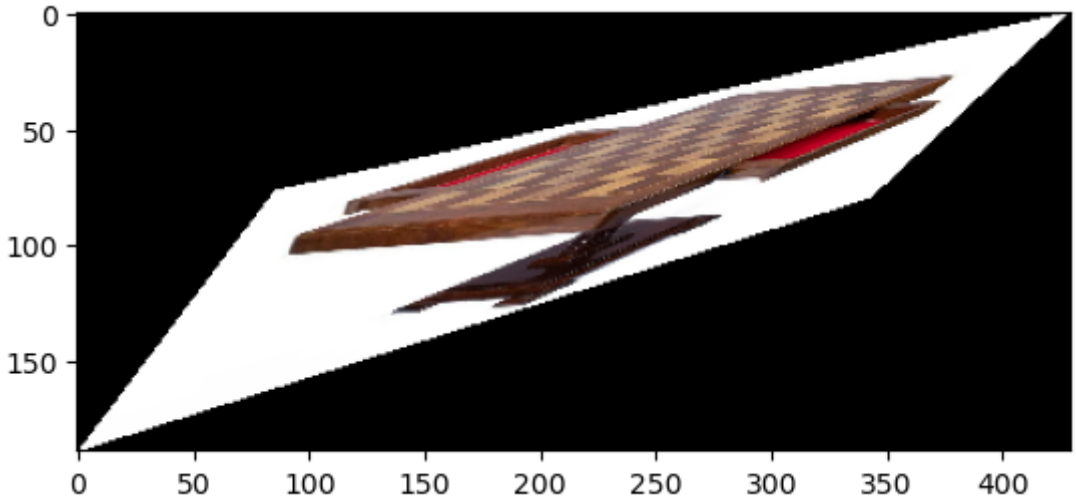


In [41]:

```
#Task 2c part ii
H2c = one_homography(dc2c) #Calculating the homography
new_rimage2 = hom_mapping_full(dimage2.copy(),np.linalg.pinv(H2c)) #Generating the projecti
new_rimage2 = cv2.cvtColor(new_rimage2,cv2.COLOR_RGB2BGR)
cv2.imwrite("one_image2.jpg",new_rimage2)
```

Out[41]:

```
True
```



In [ ]: