PURDUE UNIVERSITY

Elmore Family School of Electrical and Computer Engineering

Computer Vision

Homework 2

Adithya Sineesh

Email : asineesh@purdue.edu

Submitted: September 8, 2022

1 Theory

In this homework, estimation of homographies between different pairs of images have to be performed. These homographies are then used to transform images. Let us first look at the mathematical formulae behind them.

1.1 Finding Homography for Projective Transformation

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 Finding Homography for Affine Transformation

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,

$H =$

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

$X =$

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

$X' =$

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

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 $y' = \frac{x'_2}{x'_3}$. As we are using an affine transformation, the elements in the last row of H can be set as $h_{31} = 0, h_{32} = 0, h_{33} = 1$. Using these to facts along with the above 3 equations, we can write the physical coordinates of the domain plane as follows:

$$x' = h_{11}x + h_{12}y + h_{13}$$
$$y' = h_{21}x + h_{22}y + h_{23}$$

Therefore, as we have 6 unknowns $(h_{11}, h_{12}, h_{13}, h_{21}, h_{22}, h_{23})$, we need at least 6 equations to find them. As a single pair of corresponding points in the domain and range plane gives us 2 equations, we only need 3 pairs of corresponding points to find out all the 8 unknowns. We should also take care that no more than 2 of the 3 points lie on a straight line.

Let us take 3 points $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ from the domain plane and the 4 corresponding points $(x'_1, y'_1), (x'_2, y'_2), (x'_3, y'_3)$ from the range plane. Now we can construct the following matrix equation:

$$\begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ x_3 & y_3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_3 & y_3 & 1 \end{pmatrix} \times \begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \end{pmatrix} = \begin{pmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \end{pmatrix}$$

Since the above matrix equation is in the form of $AX = B$, we can then find the value of all the 6 unknowns present in X as follows:

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

2 Experiment

2.1 Task 1

The four images given to us are:

Figure 1: First Image of card on table

Figure 2: Second Image of card on table

Figure 3: Third Image of card on table

Figure 4: Image of "one" of the fastest cars made

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

Coordinates	**Point 1**	**Point 2**	**Point 3**	**Point 4**
Range Image 1	(527,284)	(631,1083)	(1211,789)	(1229,202)
Range Image 2	(325,250)	(224,846)	(856,1082)	(1010,260)
Range Image 3	(585,78)	(93,590)	(703,1182)	(1198,676)
Domain Image	(0,0)	(0,558)	(760,558)	(760,0)

2.2 Task 2

For this task I took photos of my kitchen cabinet at 3 different angles. For the range image, I chose an image of the Mercedes-AMG F1 W11 EQ Performance. The images can be seen below :

Figure 5: First Image of cabinet

Figure 6: Second Image of cabinet

Figure 7: Third Image of cabinet

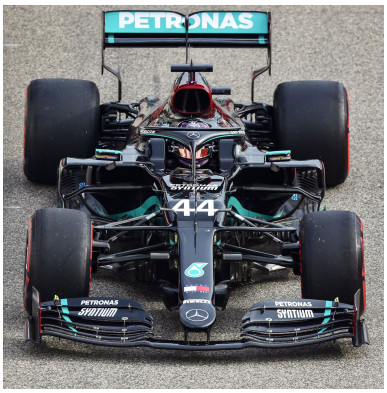


Figure 8: Image of "the" fastest car in F1

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

Coordinates	**Point 1**	**Point 2**	**Point 3**	**Point 4**
Range Image 1	(793,303)	(821,925)	(1312,878)	(1254,402)
Range Image 2	(419,249)	(391,956)	(1288,941)	(1227,245)
Range Image 3	(466,338)	(432,888)	(1084,932)	(1060,150)
Domain Image	(0,0)	(0,2048)	(2036,2048)	(2036,0)

3 Results for Task 2

3.1 Projection of the domain image into the range images

Figure 9: Projection of 1d on 1a

Figure 10: Projection of 1d on 1b

Figure 11: Projection of 1d on 1c

3.2 Applying product of the Homographies between images 1a and 1b, and between images 1b and 1c on image 1a

Figure 12: Resultant image does look like image 1c

3.3 Projection of the domain image into the range images using only affine transformations

Figure 13: Projection of 1d on 1a using only affine transformation

Figure 14: Projection of 1d on 1b using only affine transformation

Figure 15: Projection of 1d on 1c using only affine transformation. This result is the best as the image of the card on the table is taken head-on here. As a result, the shape of the card in the image is still rectangular and parallel lines will remain parallel while projecting the image 1a on it.

4 Results for Task 2

4.1 Projection of the domain image into the range images

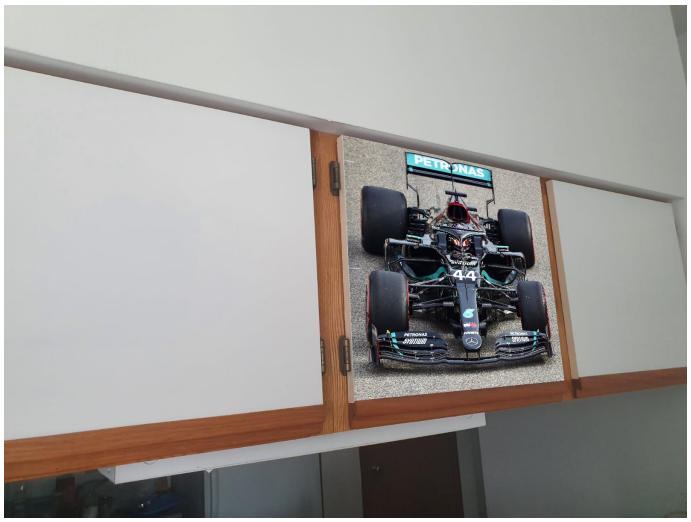


Figure 16: Projection of 2d on 2a

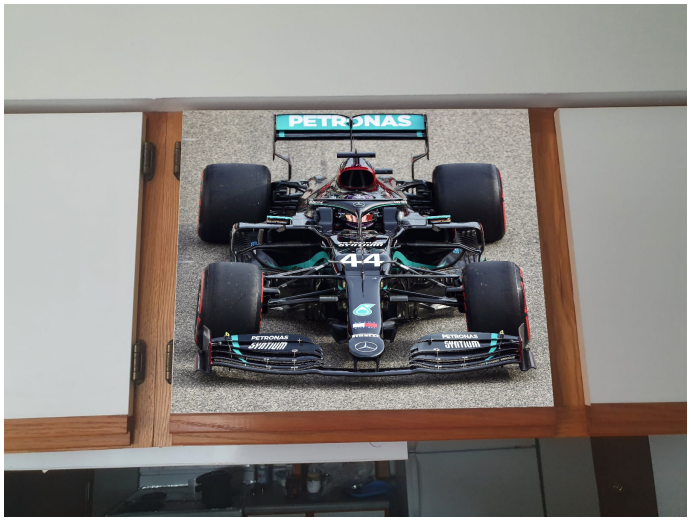


Figure 17: Projection of 2d on 2b

Figure 18: Projection of 2d on 2c

4.2 Applying product of the Homographies between images 2a and 2b, and between images 2b and 2c on image 2a

Figure 19: Resultant image does look like image 2c

4.3 Projection of the domain image into the range images using only affine transformations

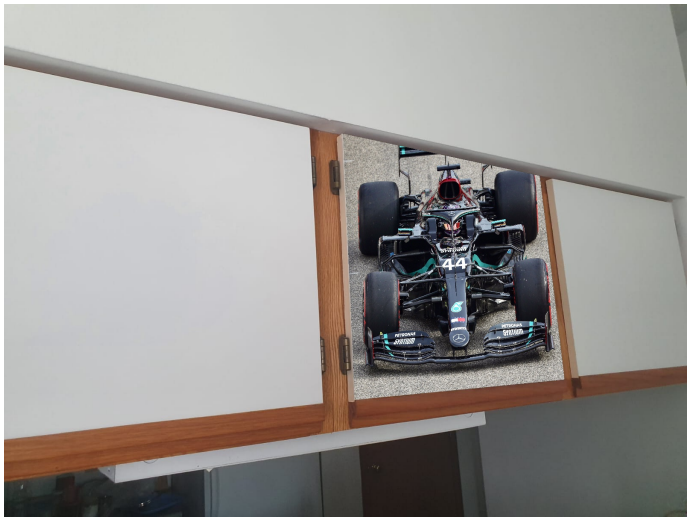


Figure 20: Projection of 2d on 2a using only affine transformation

Figure 21: Projection of 2d on 2c using only affine transformation.

Figure 22: Projection of 2d on 2b using only affine transformation. This result is the best as the image of the cabinet is taken head-on here. As a result, the shape of the card in the image is still rectangular and parallel lines will remain parallel while projecting the image 2b on it.

5 Note

For Task 1a, I initially used the domain coordinates to obtain the range coordinates with the help of the homography. I got the result below:

Figure 23: Projection of 1d on 1b using above logic

As you can see, this result is quite poor as we can still see some parts of the original range image along with the projection. This is because all the range coordinate pixels do not get a corresponding domain coordinate.

To overcome this issue, I have used the range coordinates to obtain the domain coordinates using the inverse of the homography. This ensures proper mapping and gives much better results.

Figure 24: Improved Projection of 1d on 1b

In [1]:

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

In [2]:

```
#defining the ROI coordintes for all the 4 images
def coords(n):
    if n == 0: #for domain image
        return np.array([[0,0],[0,558],[760,558],[760,0]])

    if n == 1: #for range image 1a
        return np.array([[527,284],[631,1083],[1211,789],[1229,202]])

    if n == 2: #for range image 1b
        return np.array([[325,250],[224,846],[856,1082],[1010,260]])

    if n == 3: #for range image 1c
        return np.array([[585,78],[93,590],[703,1182],[1198,676]])
```

In [3]:

```
#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 [4]:

```
#function to calculate the affine homography between two images
def affine_homography(dcord,rcord):
    A = np.zeros((6,6))
    B = np.zeros((6,1))
    H = np.zeros((3,3))

    #Generating the matrix equation AB = C using the coordinates of the ROIs
    for i in range(3):

        A[2*i,0] = dcord[i,0]
        A[2*i,1] = dcord[i,1]
        A[2*i,2] = 1

        A[2*i+1,3] = dcord[i,0]
        A[2*i+1,4] = dcord[i,1]
        A[2*i+1,5] = 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(2):
        for j in range(3):
            H[i,j]=C[3*i +j,0]

    H[2,2] = 1
    return H
```

In [5]:

```
#Creating a mask which only reveals the ROI of the range image
def masks(rimg,rcord):
    img = np.zeros_like(rimg)
    rcord = rcord.reshape(-1,1,2)
    cv2.fillPoly(img, pts = [rcord], color =(255,255,255))
    return img
```

In [6]:

```
#Function to project the domain image onto the range image ROI using homography.
# Here the domain coordinates are used to calculate the corresponding range coordinates
def hom_mapping(dimg,rimg,H,dcord):
    for i in range(dcord[1,1]):
        for j in range(dcord[2,0]):
            x  = np.array([j,i,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]
            x_bar = x_bar.astype(int)
            rimg[x_bar[1],x_bar[0]] = dimg[i,j]

    plt.imshow(rimg)
    return rimg
```

In [7]:

```
#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_mappingrev(dimg,rimg,H,rcord,rmask):
    for i in range(rimg.shape[0]):
        for j in range(rimg.shape[1]):
            if rmask[i,j,1] == 0:
                continue
            x  = np.array([j,i,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]
            x_bar = x_bar.astype(int)
            if (x_bar[0]>0 and x_bar[1]>0 and x_bar[1]<dimg.shape[0] and x_bar[0]<dimg.shap
                rimg[i,j] = dimg[x_bar[1],x_bar[0]]

    plt.imshow(rimg)
    return rimg
```

In [8]:

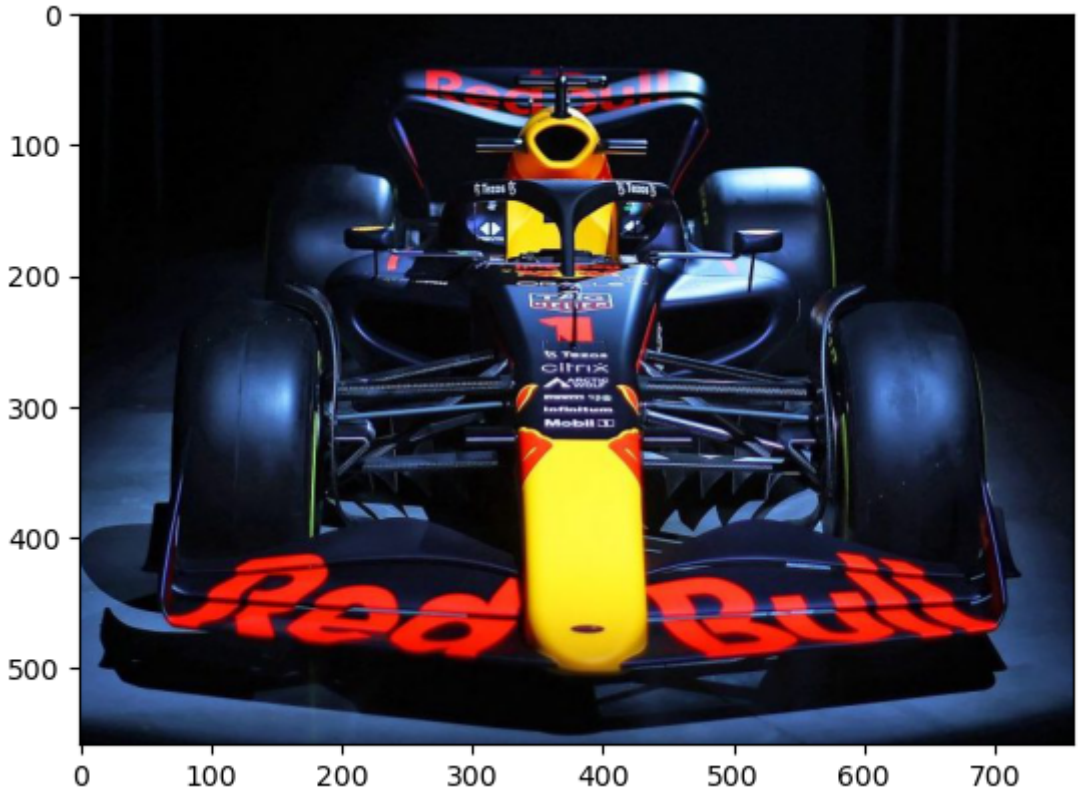
```
#Applying a homography on a full image.
def hom_mappingfull(dimg,temp,H,dcord):
    rimg = np.zeros_like(temp)
    for i in range(dimg.shape[0]):
        for j in range(dimg.shape[1]):
            x  = np.array([j,i,1])
            x_bar = np.dot(H,x)
            x_bar[0] = x_bar[0]/x_bar[2]
            x_bar[1] = x_bar[1]/x_bar[2]
            x_bar = x_bar.astype(int)
            if (x_bar[0]>0 and x_bar[1]>0 and x_bar[1]<temp.shape[0] and x_bar[0]<temp.shap
                rimg[x_bar[1],x_bar[0]] = dimg[i,j]

    plt.imshow(rimg)
    return rimg
```

In [9]:

```
#Displaying the domain image
dimg = cv2.imread ("car.jpg")
dimage = cv2.cvtColor(dimg, cv2.COLOR_BGR2RGB)
dc = coords(0)
plt.imshow(dimage)
```

Out[9]:

```
<matplotlib.figure.AxesImage at 0x21750ec5d90>
```

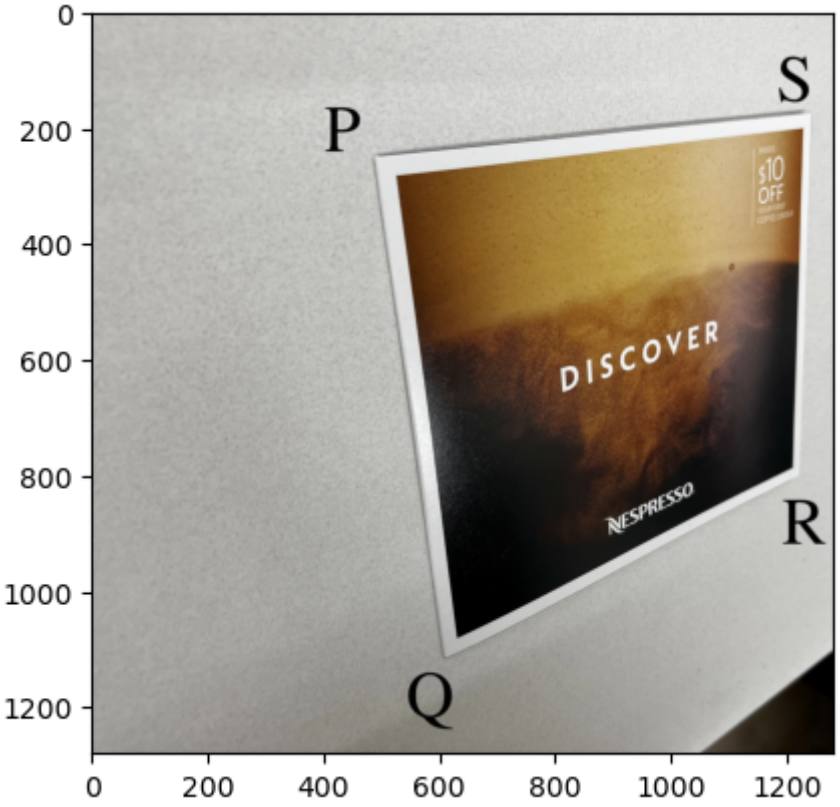


In [10]:

```
#Displaying the range image 1a
rimg1 = cv2.imread ("card1.jpeg")
rimage1 = cv2.cvtColor(rimg1, cv2.COLOR_BGR2RGB)
rc1 = coords(1)
plt.imshow(rimage1)
```

Out[10]:

```
<matplotlib.image.AxesImage at 0x21751901fd0>
```

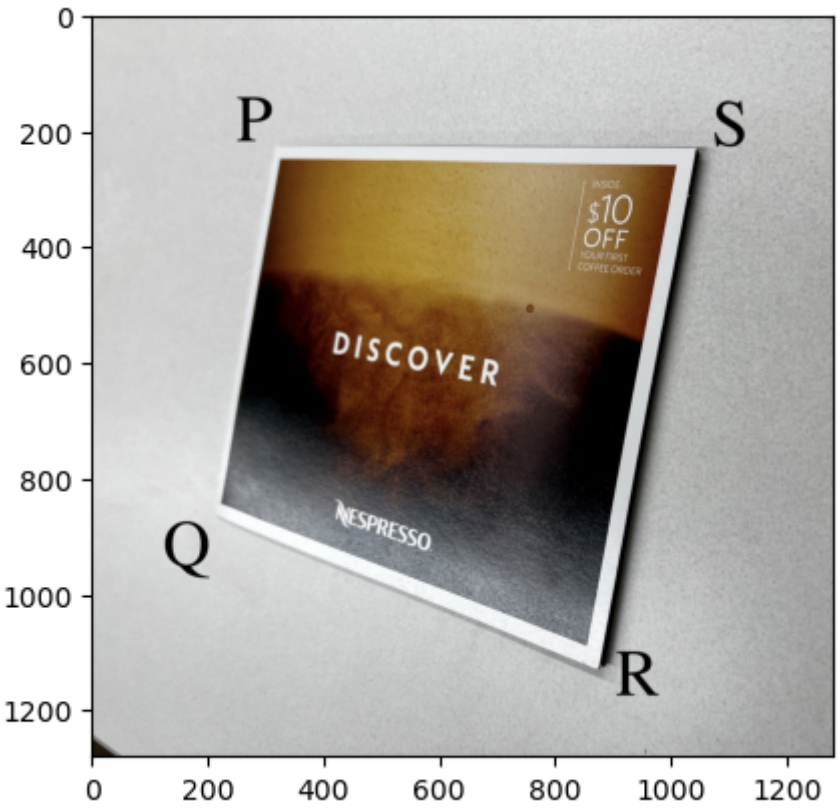


In [11]:

```
#Displaying the range image 1b
rimg2 = cv2.imread ("card2.jpeg")
rimage2 = cv2.cvtColor(rimg2, cv2.COLOR_BGR2RGB)
rc2 = coords(2)
plt.imshow(rimage2)
```

Out[11]:

```
<matplotlib.image.AxesImage at 0x217517754c0>
```

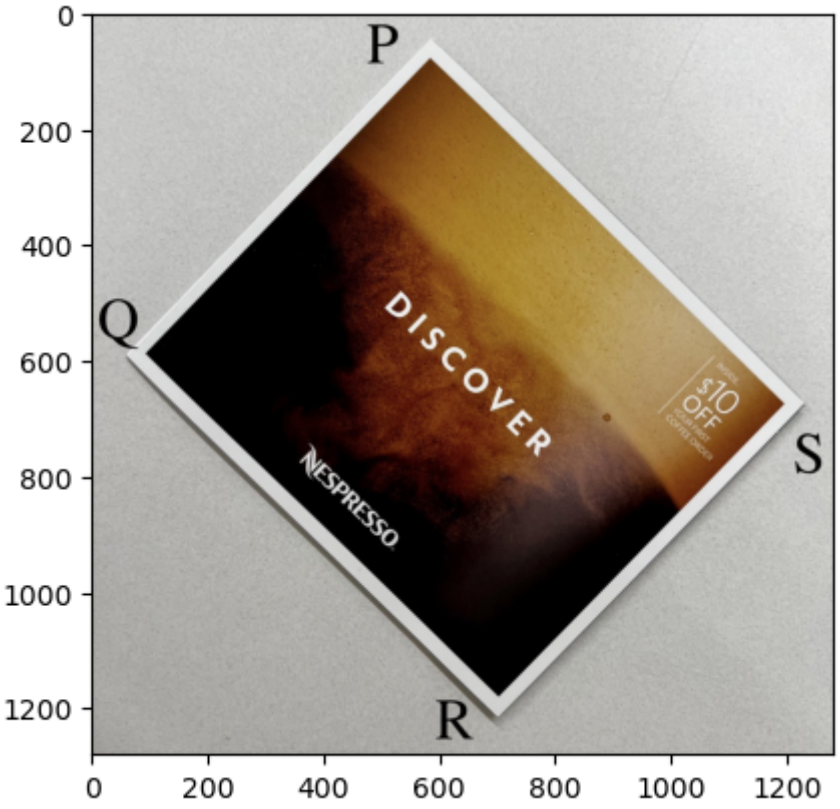


In [12]:

```
#Displaying the range image 1c
rimg3 = cv2.imread ("card3.jpeg")
rimage3 = cv2.cvtColor(rimg3, cv2.COLOR_BGR2RGB)
rc3 = coords(3)
plt.imshow(rimage3)
```

Out[12]:

```
<matplotlib.image.AxesImage at 0x217517e21c0>
```

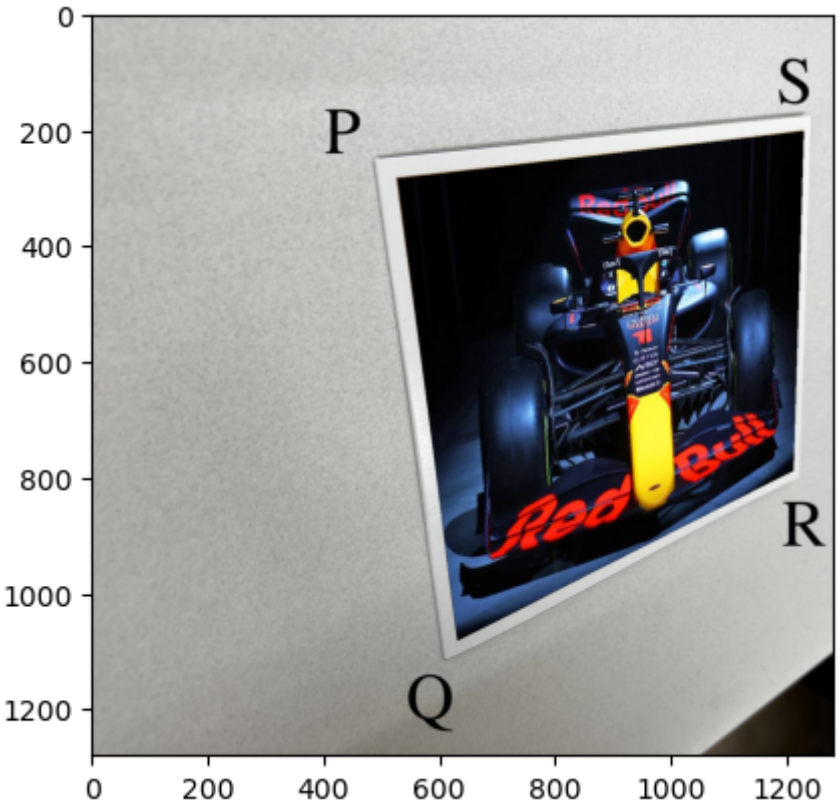


In [13]:

```
#Task 1a part i
H1 = proj_homography(dc,rc1) #Calculating the homography
rmask1 = masks(rimage1,rc1)  #Creating the mask
new_rimage1 = hom_mappingrev(dimage.copy(),rimage1.copy(),np.linalg.inv(H1),rc1,rmask1) #Ge
new_rimage1 = cv2.cvtColor(new_rimage1,cv2.COLOR_RGB2BGR)
cv2.imwrite("Proj_car_card1.jpg",new_rimage1)
```

Out[13]:

True

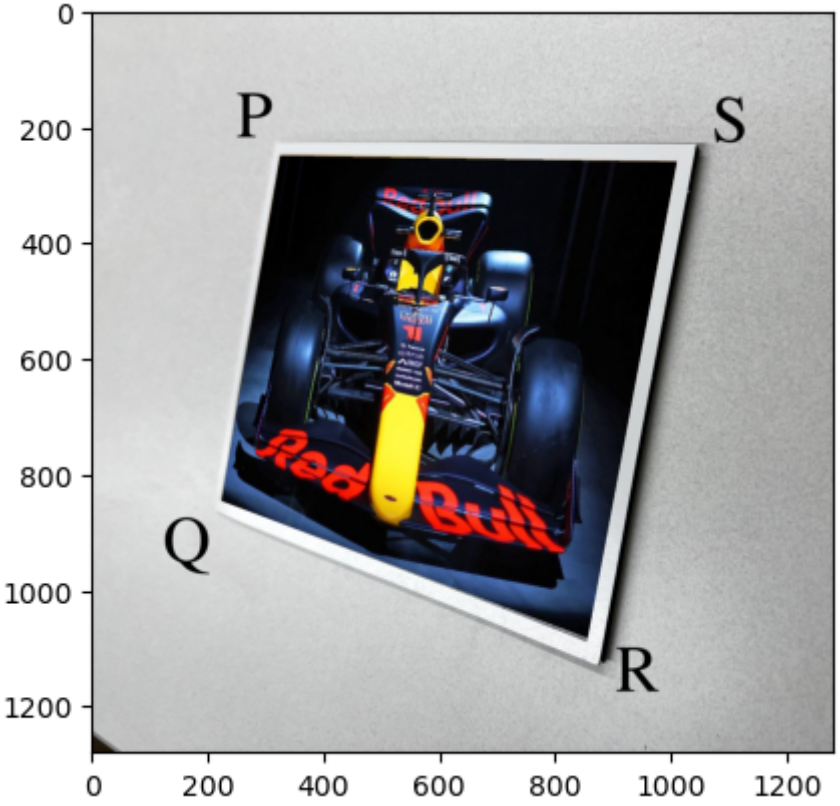


In [14]:

```
#Task 1a part ii
H2 = proj_homography(dc,rc2) #Calculating the homography
rmask2 = masks(rimage2,rc2)  #Creating the mask
new_rimage2 = hom_mappingrev(dimage.copy(),rimage2.copy(),np.linalg.inv(H2),rc2,rmask2) #Ge
new_rimage2 = cv2.cvtColor(new_rimage2,cv2.COLOR_RGB2BGR)
cv2.imwrite("Proj_car_card2.jpg",new_rimage2)
```

Out[14]:

```
True
```

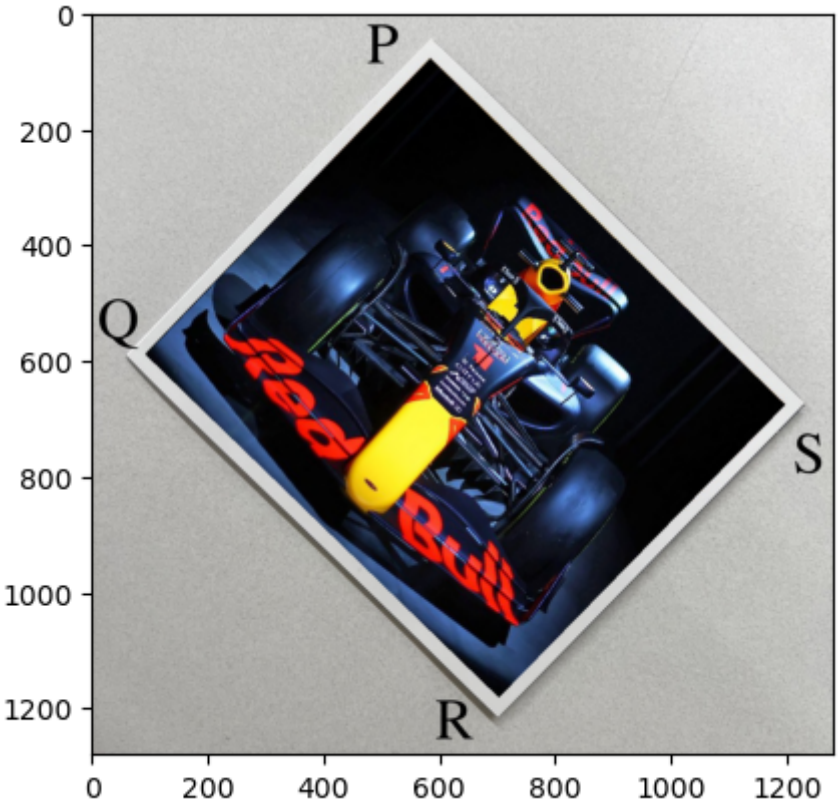


In [15]:

```
#Task 1a part iii
H3 = proj_homography(dc,rc3) #Calculating the homography
rmask3 = masks(rimage3,rc3)  #Creating the mask
new_rimage3 = hom_mappingrev(dimage.copy(),rimage3.copy(),np.linalg.inv(H3),rc3,rmask3) #Ge
new_rimage3 = cv2.cvtColor(new_rimage3,cv2.COLOR_RGB2BGR)
cv2.imwrite("Proj_car_card3.jpg",new_rimage3)
```

Out[15]:

True

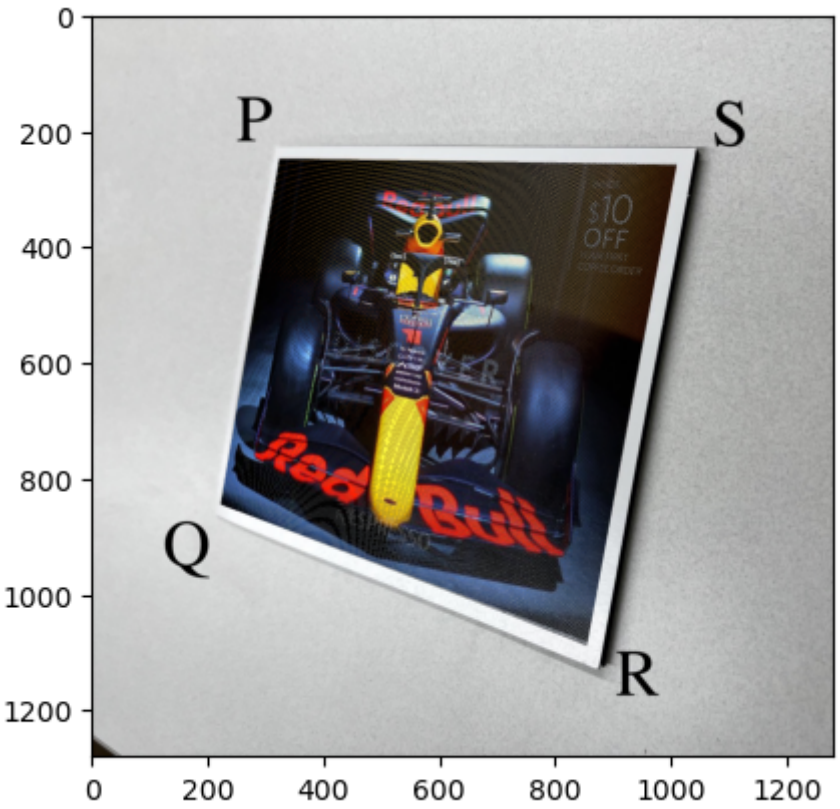


In [16]:

```
#Task 1a, using the domain coordinates to generate the range coordinates. Gives poor result
bad_rimage = hom_mapping(dimage.copy(),rimage2.copy(),H2,dc)
bad_rimage = cv2.cvtColor(bad_rimage,cv2.COLOR_RGB2BGR)
cv2.imwrite("BadProj_car_card2.jpg",bad_rimage)
```

Out[16]:

```
True
```

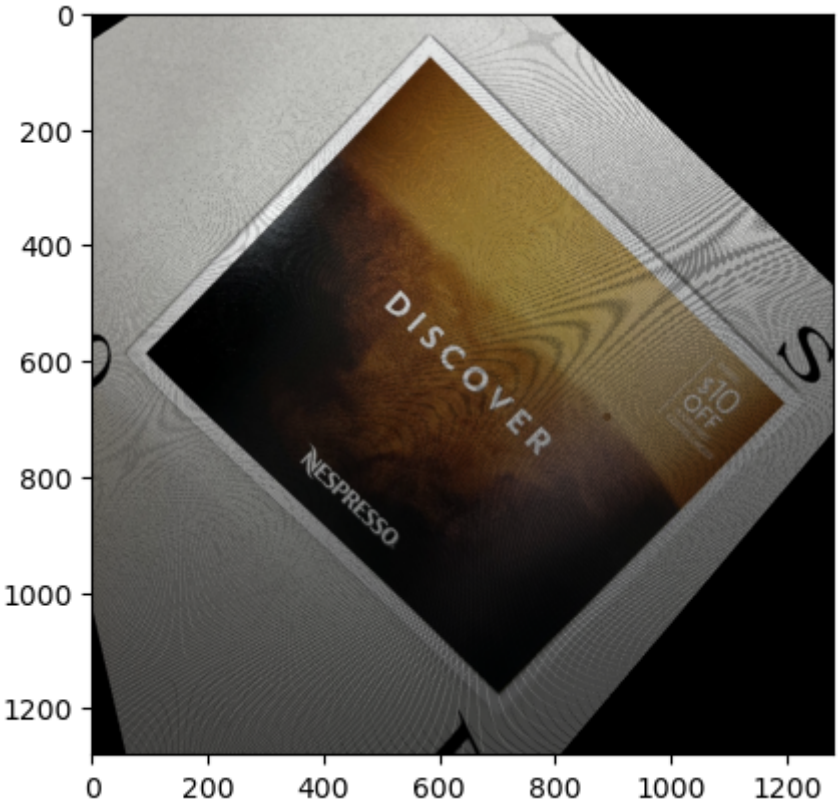


In [17]:

```
#Task 1b
#Calculating Homographies
H12 = proj_homography(rc1,rc2)
H23 = proj_homography(rc2,rc3)

fuse_rimage = hom_mappingfull(rimage1.copy(),rimage3.copy(),np.dot(H23,H12),rc1) #Generatin
fuse_rimage = cv2.cvtColor(fuse_rimage,cv2.COLOR_RGB2BGR)
cv2.imwrite("Proj_card1_card3.jpg",fuse_rimage)
```

Out[17]:

True

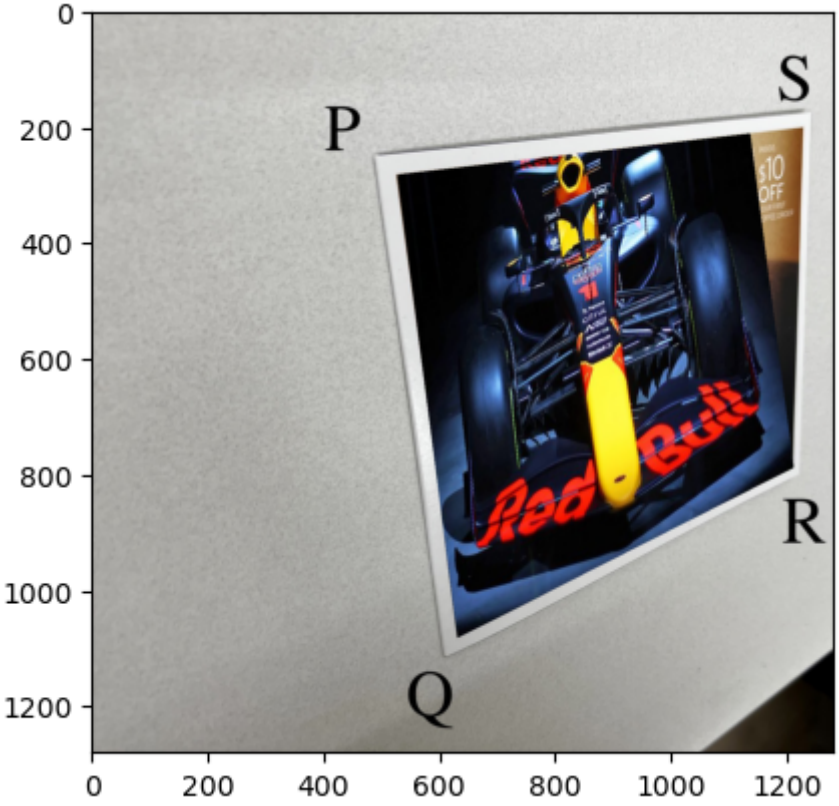


In [18]:

```
#Task 1c part i
aH1 = affine_homography(dc,rc1) #Calculating affine Homographies
anew_rimage1 = hom_mappingrev(dimage.copy(),rimage1.copy(),np.linalg.inv(aH1),rc1,rmask1) #
anew_rimage1 = cv2.cvtColor(anew_rimage1,cv2.COLOR_RGB2BGR)
cv2.imwrite("Affine_car_card1.jpg",anew_rimage1)
```

Out[18]:

```
True
```

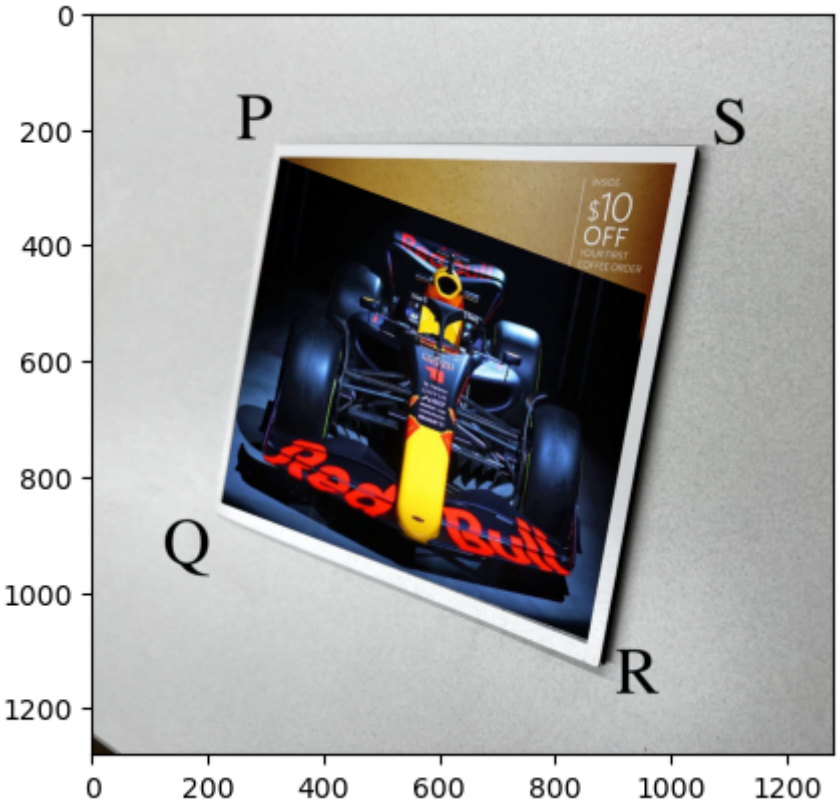


In [19]:

```
#Task 1c part ii
aH2 = affine_homography(dc,rc2) #Calculating affine Homographies
anew_rimage2 = hom_mappingrev(dimage.copy(),rimage2.copy(),np.linalg.inv(aH2),rc2,rmask2) #
anew_rimage2 = cv2.cvtColor(anew_rimage2,cv2.COLOR_RGB2BGR)
cv2.imwrite("Affine_car_card2.jpg",anew_rimage2)
```

Out[19]:

True

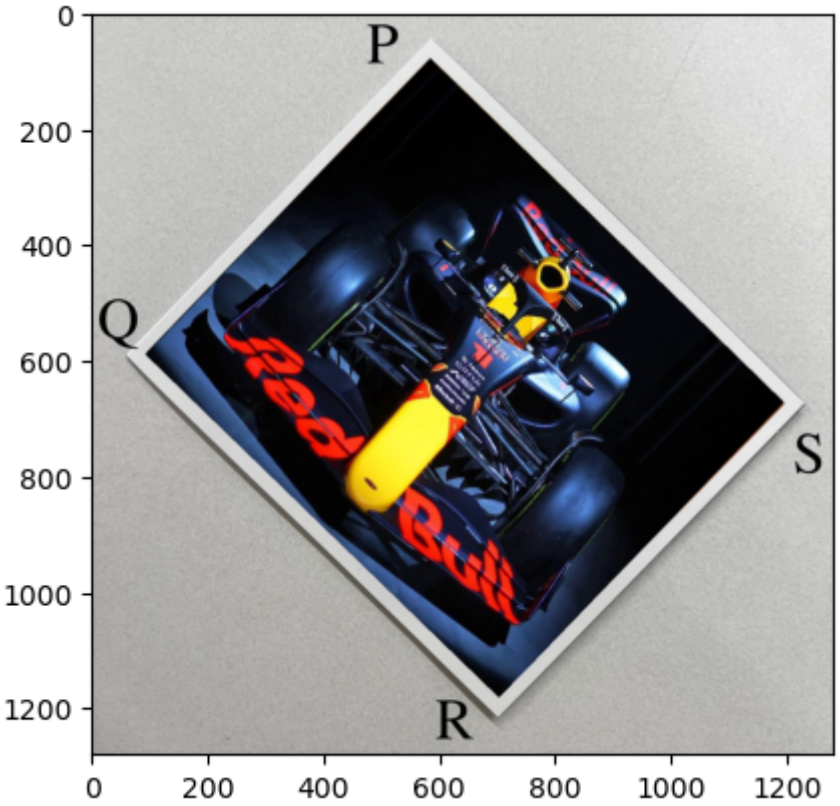


In [20]:

```
#Task 1c part iii
aH3 = affine_homography(dc,rc3) #Calculating affine Homographies
anew_rimage3 = hom_mappingrev(dimage.copy(),rimage3.copy(),np.linalg.inv(aH3),rc3,rmask3) #
anew_rimage3 = cv2.cvtColor(anew_rimage3,cv2.COLOR_RGB2BGR)
cv2.imwrite("Affine_car_card3.jpg",anew_rimage3)
```

Out[20]:

True



In [ ]:

In [ ]:

```
```

In [ ]:

```
```

In [1]:

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

In [2]:

```
#defining the ROI coordintes for all the 4 images

def coords(n):

    if n == 0: #for domain image
        return np.array([[0,0],[0,2048],[2036,2048],[2036,0]])

    if n == 1: #for range image 1a
        return np.array([[793,303],[821,925],[1312,878],[1254,402]])

    if n == 2: #for range image 1b
        return np.array([[419,249],[391,956],[1288,941],[1227,245]])

    if n == 3: #for range image 1c
        return np.array([[466,338],[432,888],[1084,932],[1060,150]])
```

In [3]:

```
#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 [4]:

```
#function to calculate the affine homography between two images
def affine_homography(dcord,rcord):
    A = np.zeros((6,6))
    B = np.zeros((6,1))
    H = np.zeros((3,3))

    #Generating the matrix equation AB = C using the coordinates of the ROIs
    for i in range(3):

        A[2*i,0] = dcord[i,0]
        A[2*i,1] = dcord[i,1]
        A[2*i,2] = 1

        A[2*i+1,3] = dcord[i,0]
        A[2*i+1,4] = dcord[i,1]
        A[2*i+1,5] = 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(2):
        for j in range(3):
            H[i,j]=C[3*i +j,0]

    H[2,2] = 1
    return H
```

In [5]:

```
#Creating a mask which only reveals the ROI of the range image
def masks(rimg,rcord):
    img = np.zeros_like(rimg)
    rcord = rcord.reshape(-1,1,2)
    cv2.fillPoly(img, pts = [rcord], color =(255,255,255))
    return img
```

In [6]:

```
#Function to project the domain image onto the range image ROI using homography.
# Here the domain coordinates are used to calculate the corresponding range coordinates
def hom_mapping(dimg,rimg,H,dcord):
    for i in range(dcord[1,1]):
        for j in range(dcord[2,0]):
            x  = np.array([j,i,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]
            x_bar = x_bar.astype(int)
            rimg[x_bar[1],x_bar[0]] = dimg[i,j]

    plt.imshow(rimg)
    return rimg
```

In [7]:

```
#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_mappingrev(dimg,rimg,H,rcord,rmask):
    for i in range(rimg.shape[0]):
        for j in range(rimg.shape[1]):
            if rmask[i,j,1] == 0:
                continue
            x  = np.array([j,i,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]
            x_bar = x_bar.astype(int)
            if (x_bar[0]>0 and x_bar[1]>0 and x_bar[1]<dimg.shape[0] and x_bar[0]<dimg.shap
                rimg[i,j] = dimg[x_bar[1],x_bar[0]]

    plt.imshow(rimg)
    return rimg
```

In [8]:

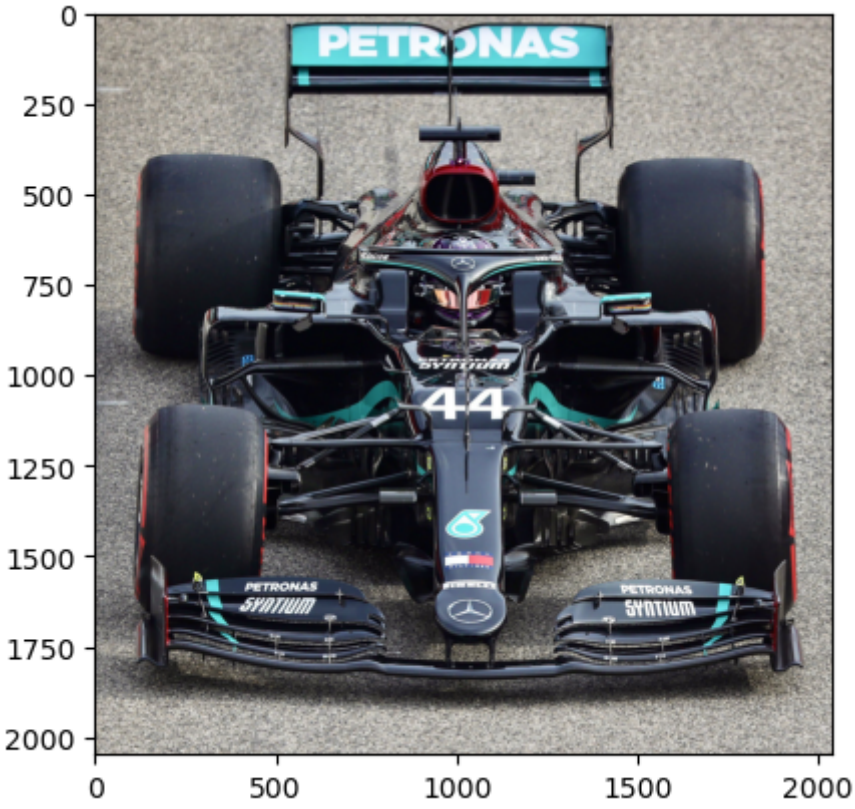
```
#Applying a homography on a full image.
def hom_mappingfull(dimg,temp,H,dcord):
    rimg = np.zeros_like(temp)
    for i in range(dimg.shape[0]):
        for j in range(dimg.shape[1]):
            x  = np.array([j,i,1])
            x_bar = np.dot(H,x)
            x_bar[0] = x_bar[0]/x_bar[2]
            x_bar[1] = x_bar[1]/x_bar[2]
            x_bar = x_bar.astype(int)
            if (x_bar[0]>0 and x_bar[1]>0 and x_bar[1]<temp.shape[0] and x_bar[0]<temp.shap
                rimg[x_bar[1],x_bar[0]] = dimg[i,j]

    plt.imshow(rimg)
    return rimg
```

In [9]:

```
#Displaying the domain image
dimg = cv2.imread ("car1.jpg")
dimage = cv2.cvtColor(dimg, cv2.COLOR_BGR2RGB)
dc = coords(0)
plt.imshow(dimage)
```

Out[9]:

```
<matplotlib.image.AxesImage at 0x17e0286da90>
```

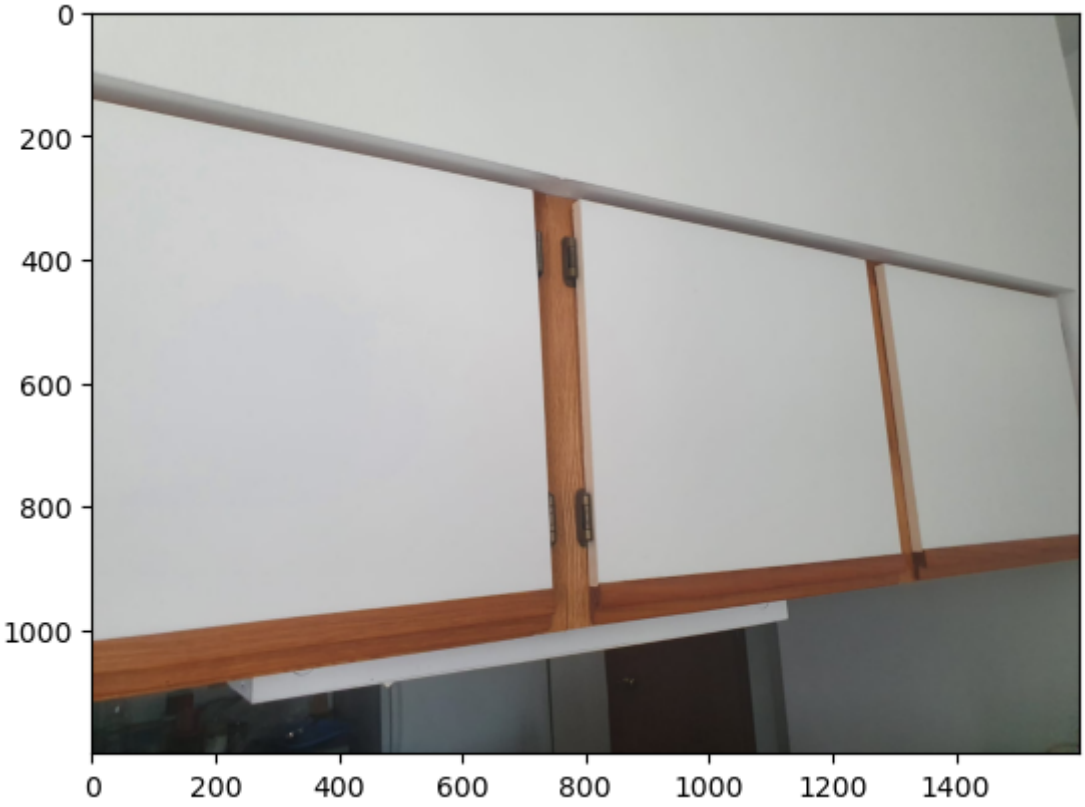


In [10]:

```
#Displaying the range image 2a
rimg1 = cv2.imread ("cabinet1.jpeg")
rimage1 = cv2.cvtColor(rimg1, cv2.COLOR_BGR2RGB)
rc1 = coords(1)
plt.imshow(rimage1)
```

Out[10]:

```
<matplotlib.image.AxesImage at 0x17e03b661c0>
```

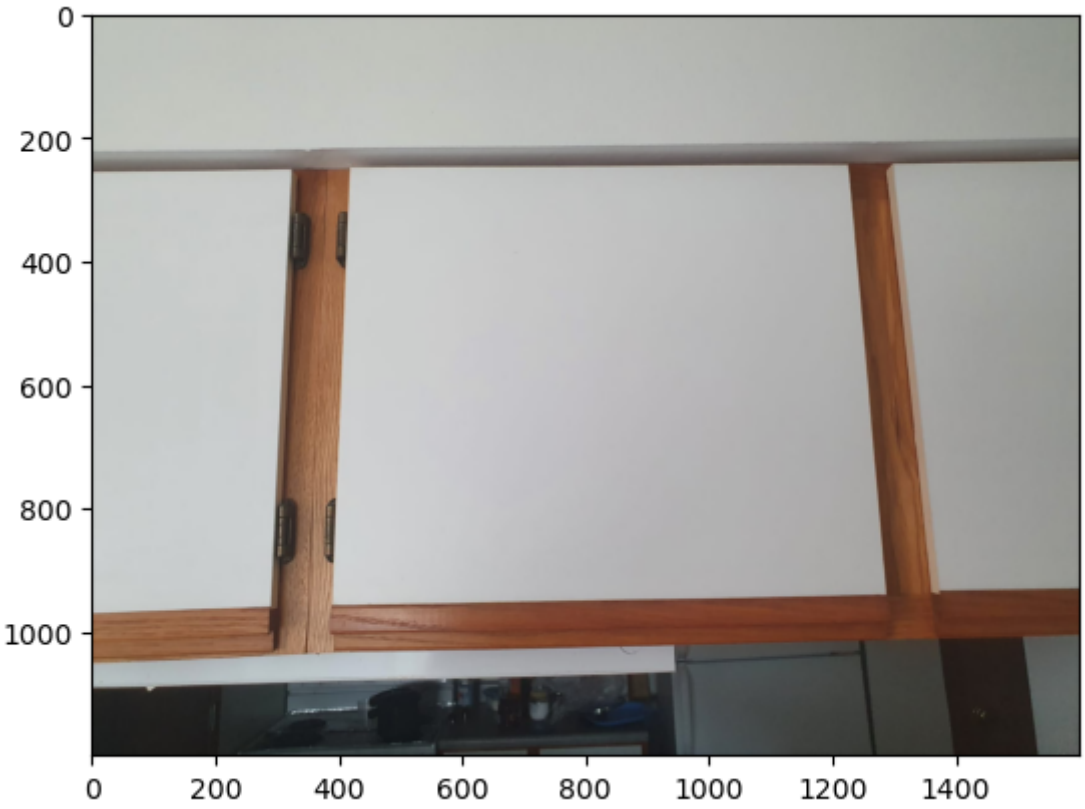


In [11]:

```
#Displaying the range image 2b
rimg2 = cv2.imread ("cabinet2.jpeg")
rimage2 = cv2.cvtColor(rimg2, cv2.COLOR_BGR2RGB)
rc2 = coords(2)
plt.imshow(rimage2)
```

Out[11]:

```
<matplotlib.image.AxesImage at 0x17e03bcc910>
```



In [12]:

```
#Displaying the range image 2c
rimg3 = cv2.imread ("cabinet3.jpeg")
rimage3 = cv2.cvtColor(rimg3, cv2.COLOR_BGR2RGB)
rc3 = coords(3)
plt.imshow(rimage3)
```

Out[12]:

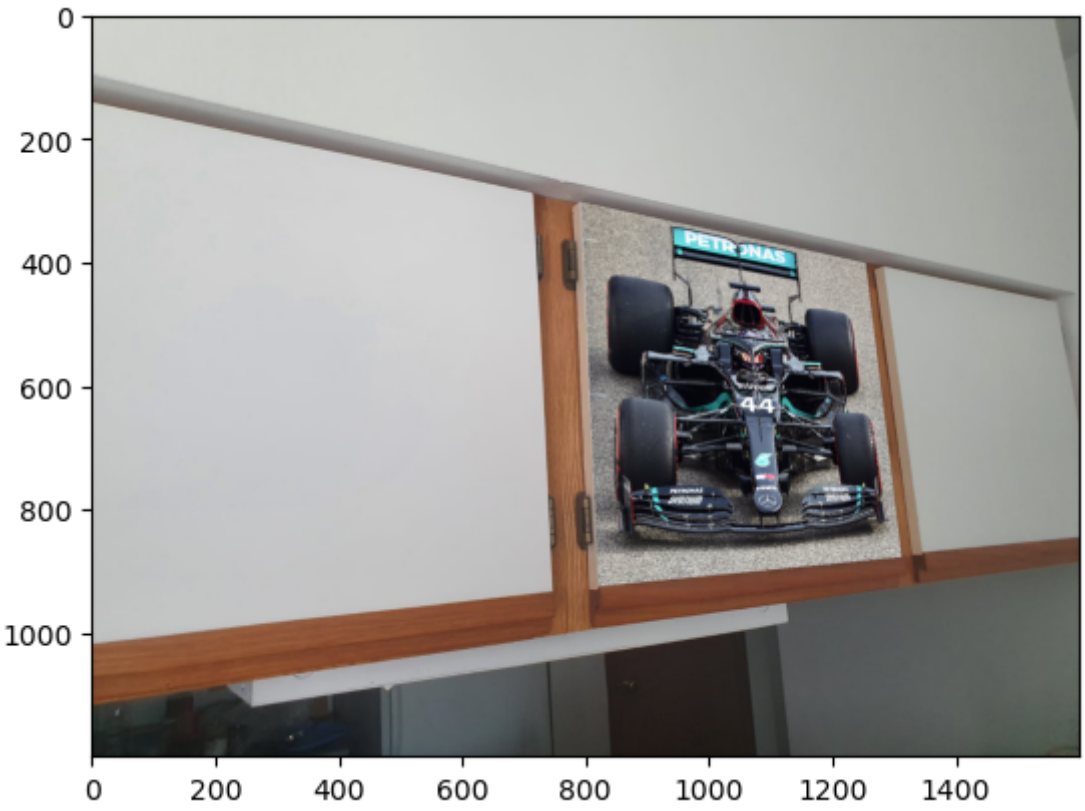
```
<matplotlib.image.AxesImage at 0x17e03c36760>
```

In [13]:

```
#Task 1a part i
H1 = proj_homography(dc,rc1) #Calculating the homography
rmask1 = masks(rimage1,rc1)  #Creating the mask
new_rimage1 = hom_mappingrev(dimage.copy(),rimage1.copy(),np.linalg.inv(H1),rc1,rmask1) #Ge
new_rimage1 = cv2.cvtColor(new_rimage1,cv2.COLOR_RGB2BGR)
cv2.imwrite("Proj_car_card1.jpg",new_rimage1)
```

Out[13]:

True

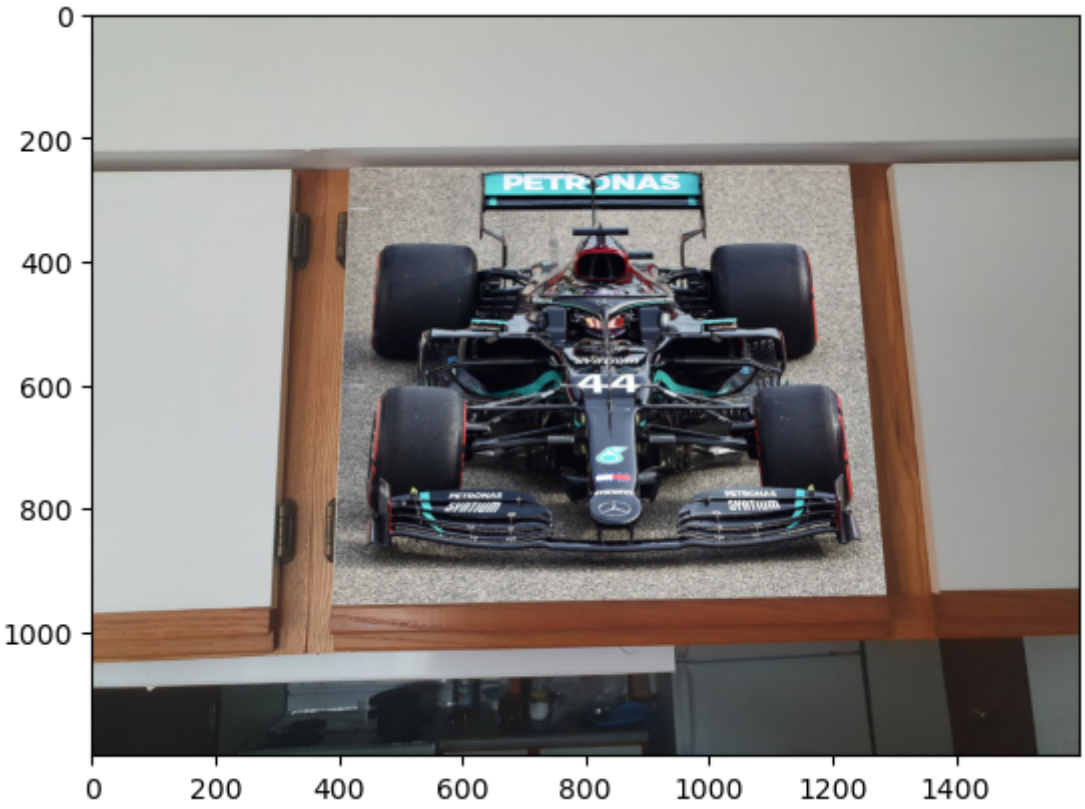


In [14]:

```
#Task 1a part ii
H2 = proj_homography(dc,rc2) #Calculating the homography
rmask2 = masks(rimage2,rc2)  #Creating the mask
new_rimage2 = hom_mappingrev(dimage.copy(),rimage2.copy(),np.linalg.inv(H2),rc2,rmask2) #Ge
new_rimage2 = cv2.cvtColor(new_rimage2,cv2.COLOR_RGB2BGR)
cv2.imwrite("Proj_car_card2.jpg",new_rimage2)
```

Out[14]:

```
True
```

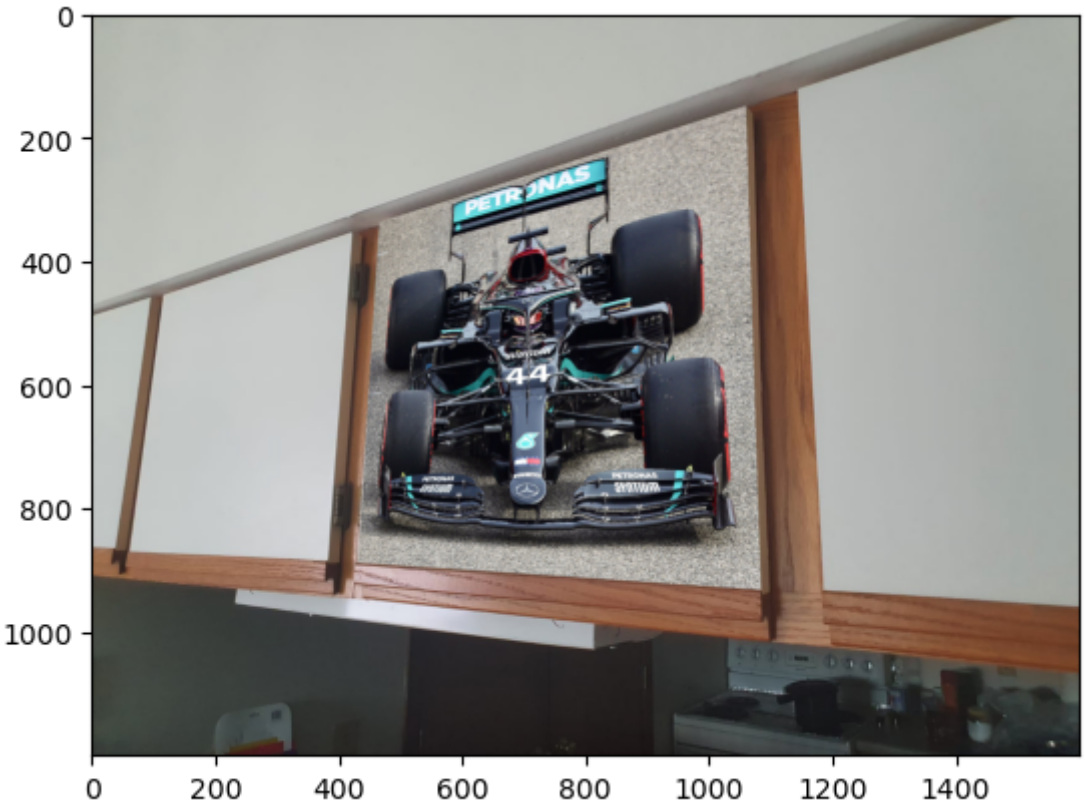


In [15]:

```
#Task 1a part iii
H3 = proj_homography(dc,rc3) #Calculating the homography
rmask3 = masks(rimage3,rc3)  #Creating the mask
new_rimage3 = hom_mappingrev(dimage.copy(),rimage3.copy(),np.linalg.inv(H3),rc3,rmask3) #Ge
new_rimage3 = cv2.cvtColor(new_rimage3,cv2.COLOR_RGB2BGR)
cv2.imwrite("Proj_car_card3.jpg",new_rimage3)
```

Out[15]:

True

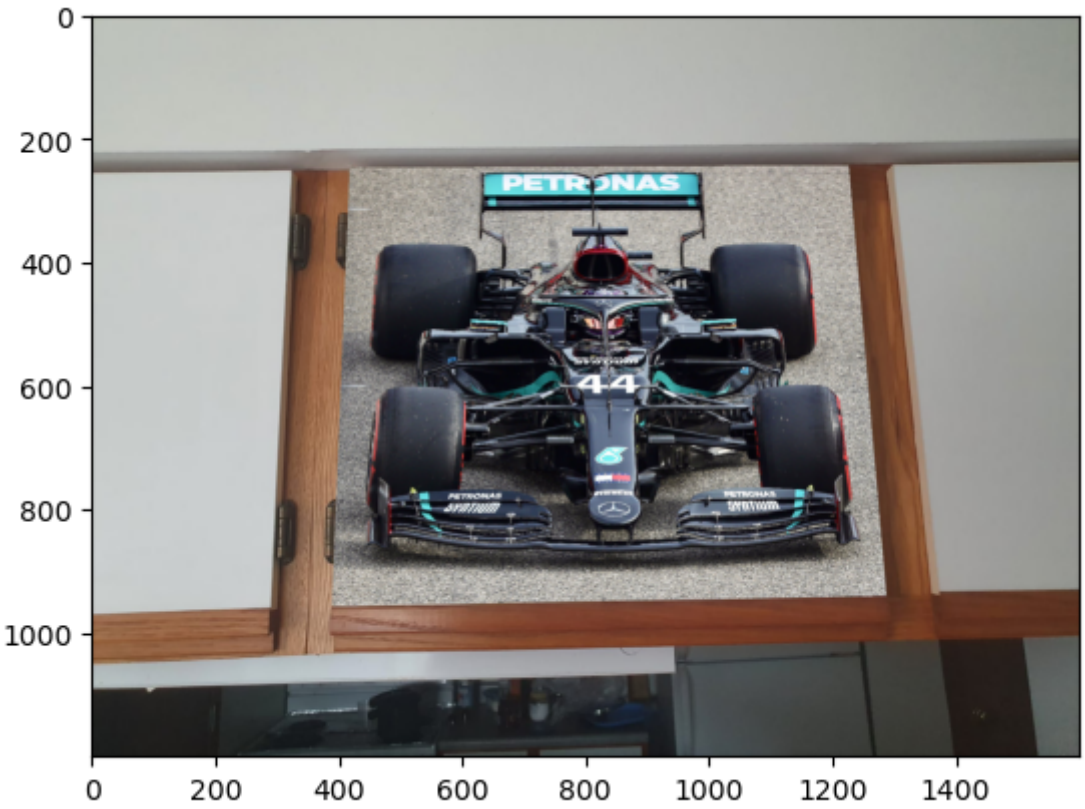


In [16]:

```
#Task 1a, using the domain coordinates to generate the range coordinates. Doesn't give poor
bad_rimage = hom_mapping(dimage.copy(),rimage2.copy(),H2,dc)
bad_rimage = cv2.cvtColor(bad_rimage,cv2.COLOR_RGB2BGR)
cv2.imwrite("BadProj_car_card2.jpg",bad_rimage)
```

Out[16]:

```
True
```

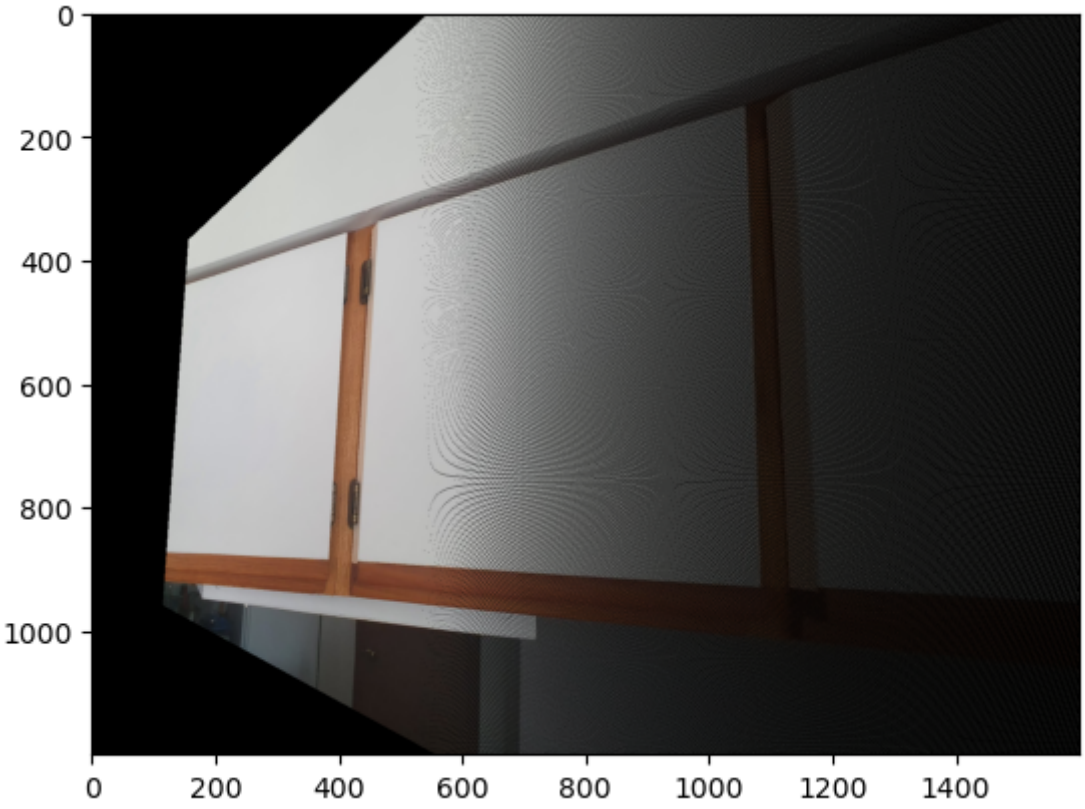


In [17]:

```
#Task 1b
#Calculating Homographies
H12 = proj_homography(rc1,rc2)
H23 = proj_homography(rc2,rc3)

fuse_rimage = hom_mappingfull(rimage1.copy(),rimage3.copy(),np.dot(H23,H12),rc1) #Generatin
fuse_rimage = cv2.cvtColor(fuse_rimage,cv2.COLOR_RGB2BGR)
cv2.imwrite("Proj_card1_card3.jpg",fuse_rimage)
```

Out[17]:

True

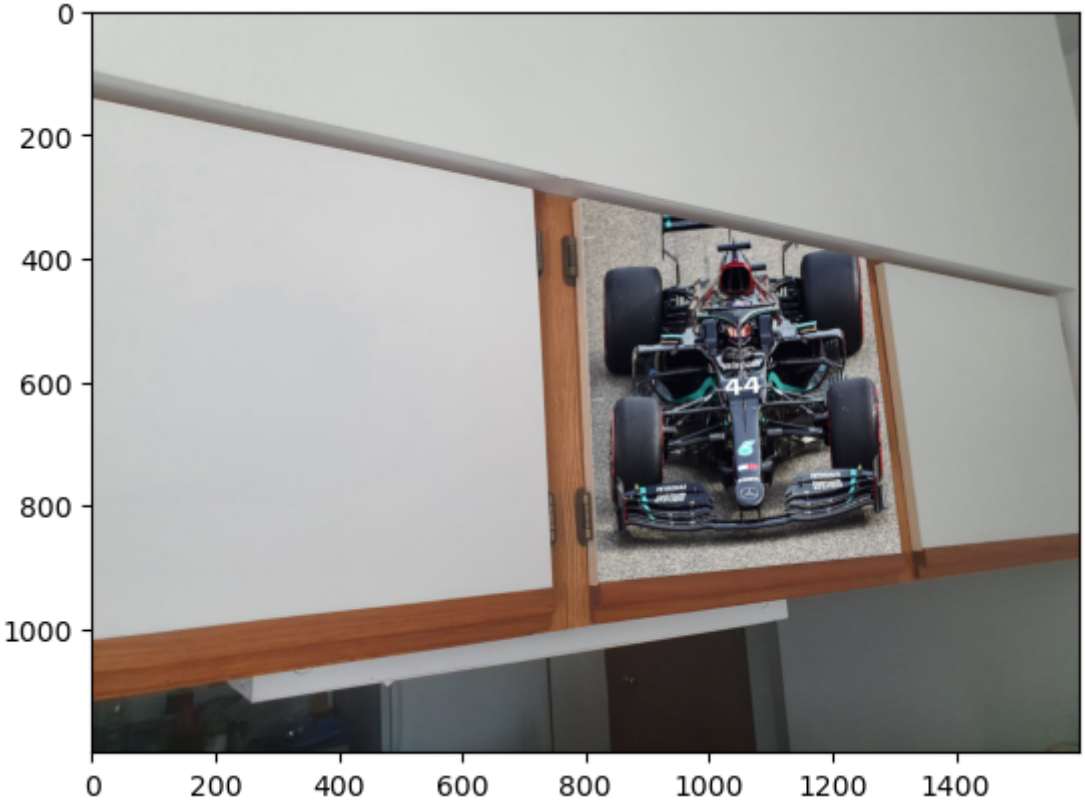


In [18]:

```
#Task 1c part i
aH1 = affine_homography(dc,rc1) #Calculating affine Homographies
anew_rimage1 = hom_mappingrev(dimage.copy(),rimage1.copy(),np.linalg.inv(aH1),rc1,rmask1) #
anew_rimage1 = cv2.cvtColor(anew_rimage1,cv2.COLOR_RGB2BGR)
cv2.imwrite("Affine_car_card1.jpg",anew_rimage1)
```

Out[18]:

True

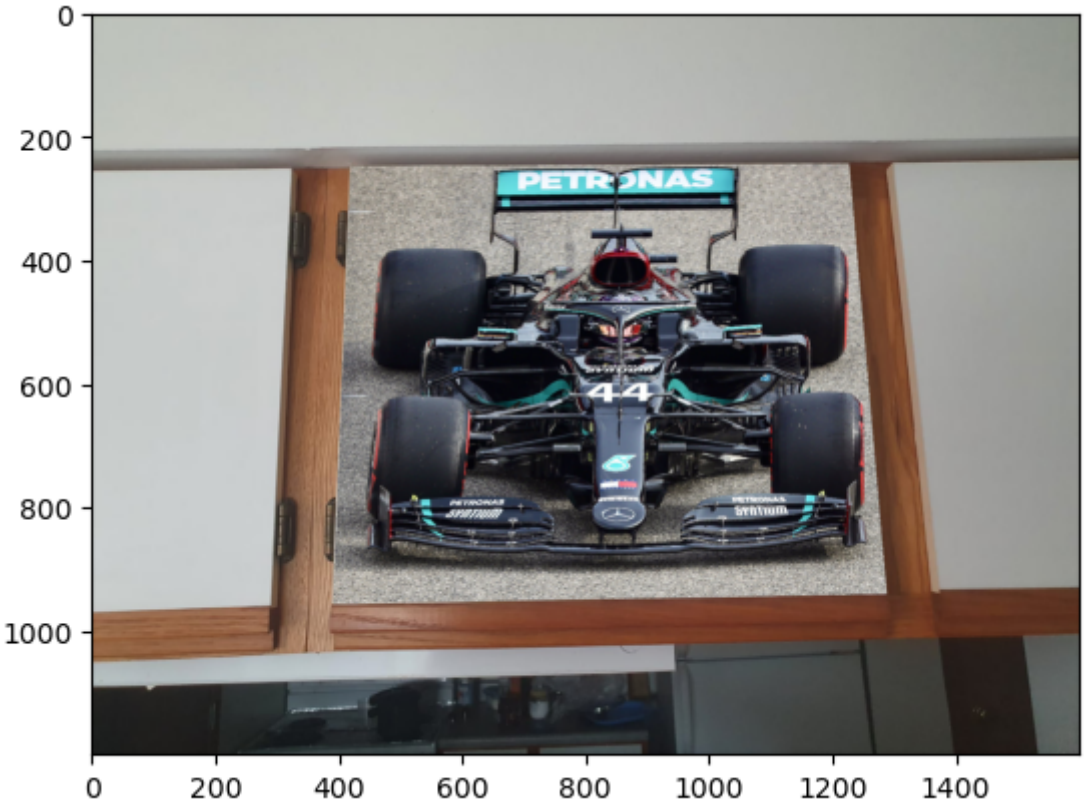


In [19]:

```
#Task 1c part ii
aH2 = affine_homography(dc,rc2) #Calculating affine Homographies
anew_rimage2 = hom_mappingrev(dimage.copy(),rimage2.copy(),np.linalg.inv(aH2),rc2,rmask2) #
anew_rimage2 = cv2.cvtColor(anew_rimage2,cv2.COLOR_RGB2BGR)
cv2.imwrite("Affine_car_card2.jpg",anew_rimage2)
```

Out[19]:

True

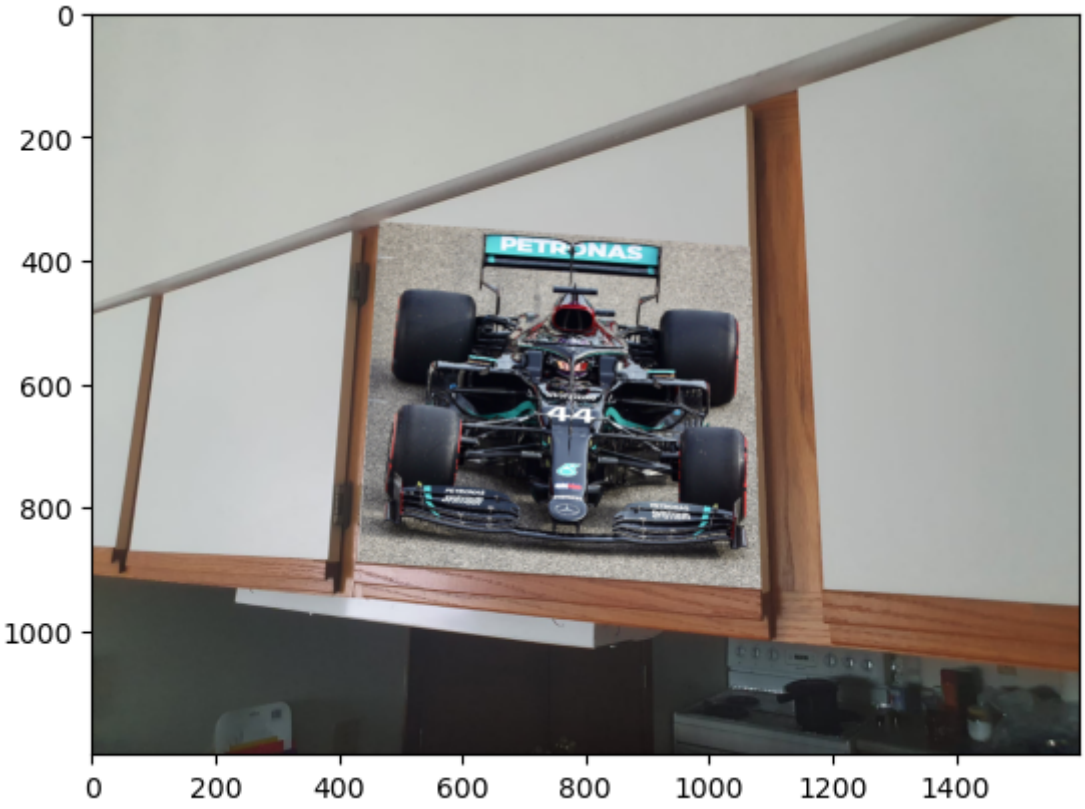


In [20]:

```
#Task 1c part iii
aH3 = affine_homography(dc,rc3) #Calculating affine Homographies
anew_rimage3 = hom_mappingrev(dimage.copy(),rimage3.copy(),np.linalg.inv(aH3),rc3,rmask3) #
anew_rimage3 = cv2.cvtColor(anew_rimage3,cv2.COLOR_RGB2BGR)
cv2.imwrite("Affine_car_card3.jpg",anew_rimage3)
```

Out[20]:

True



In [ ]:

In [ ]:

In [ ]:

```

```