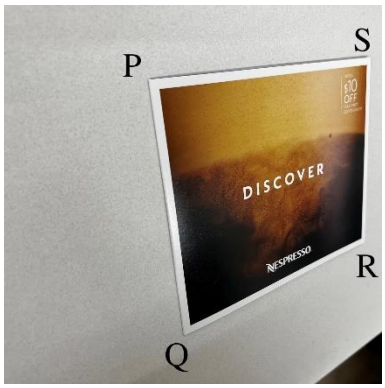
FALL 22 ECE 661 – COMPUTER VISION

Homework 2
Sahithi Kodali – 34789866
kodali1@purdue.edu

TASK – 1:

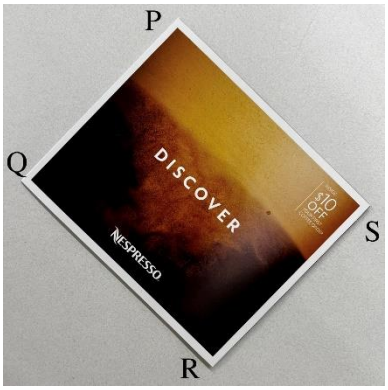
Four images are provided in which three images are cards pictures in different angles and one car image as shown below.

Q1. This task involves calculating Homographies between pairs of images to project an image onto the other i.e., image 1d to be projected on PQRS frames of 1a, 1b, 1c images respectively.



Img (1a)

Img (1b)



Img (1c)

Img (1d)

Main Concept:

To perform such tasks of projections, we must understand the concept of Homography that needs to be further implemented. To project an image onto another image frame, we must find the relationship between their corresponding points. If the homogeneous representation of a point in the domain space and the projective space 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,

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

Since, Homography matrix is a linear mapping of homogeneous coordinates we only consider the ratios of the coordinates i.e. if point (x1, x2, x3) exists then it is same as (x1/x3, x2/x3,1) which is as in the representation of the points taken $\vec{x}$ and $\vec{x}'$.

Hence, the coordinate $H_{(3,3)}$ = i = 1.

From the equations above,

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

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

$$x' = ax + by + c$$

$$y' = dx + ey + f$$

$$1 = gx + hy + 1$$

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,

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

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

To perform the task of projecting image 1d on 1a, 1b, 1c; the following steps need to be followed.

i.) Find the pixel coordinates of images using the corners/the projective frame points to find H. The tool GIMP is used to determine the pixel coordinates of the images.

ii.) Determine the Homography matrix using $H = A^{-1}B$ for the respective image pairs.

iii.) Calculate the mapping coordinates required for projecting using H.

iv.) Now using the mapping coordinates, replace each pixel coordinates in the projective space image (1a/1b/1c) with the corresponding pixel coordinates of the domain image (1d) that needs to be projected.

The resulting images obtained after performing these steps are,

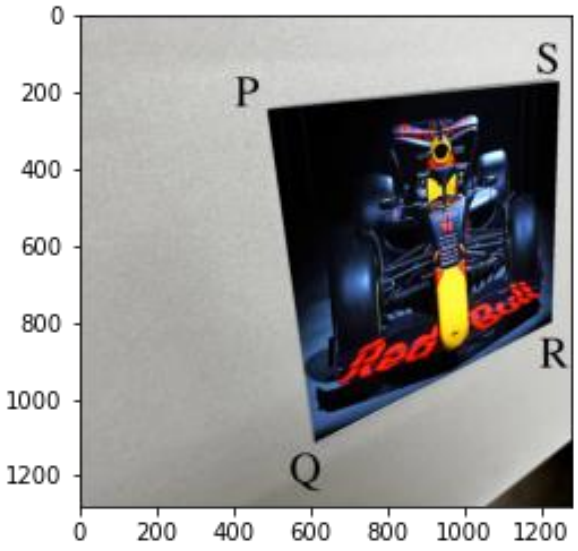


Figure 1. Projection of image 1d on 1a

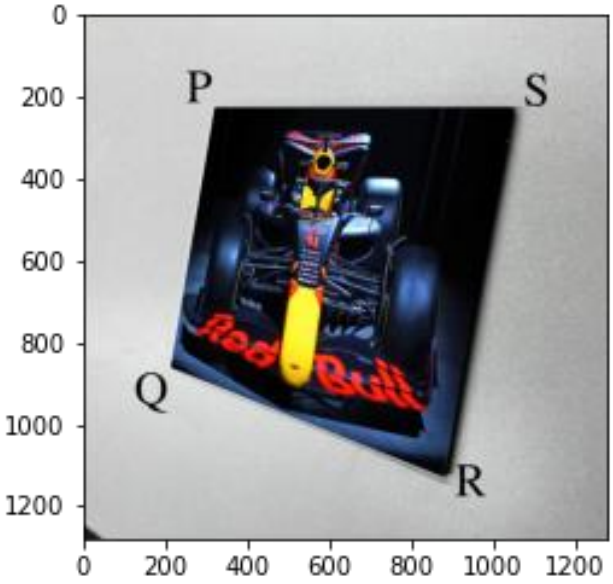


Figure 2. Projection of image 1d on 1b

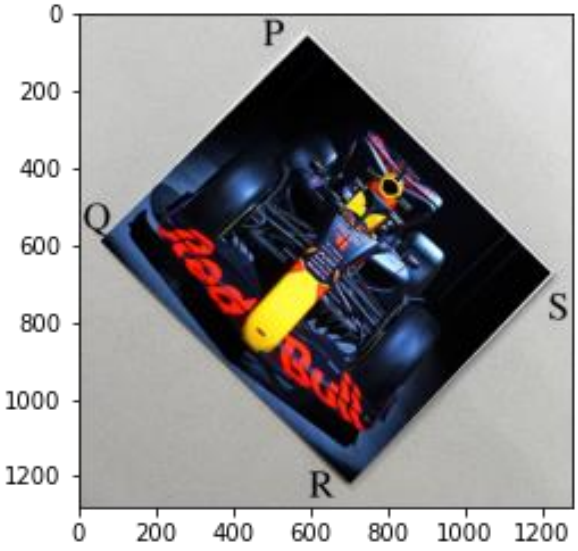


Figure 3. Projection of image 1d on 1c

Q2. To perform this task, the following steps are followed

i.) Find the Homographies between the pair of images (1a,1b) and (1b,1c). Multiply these to obtain a new H.

ii.) The new H is used to find the mapping coordinates required for projection.

iii.) Using these mapping coordinates, replace each pixel coordinates in the projective space image (1c) with the corresponding pixel coordinates of the domain image (1a) that needs to be projected.

The resulting images obtained after performing these steps are,

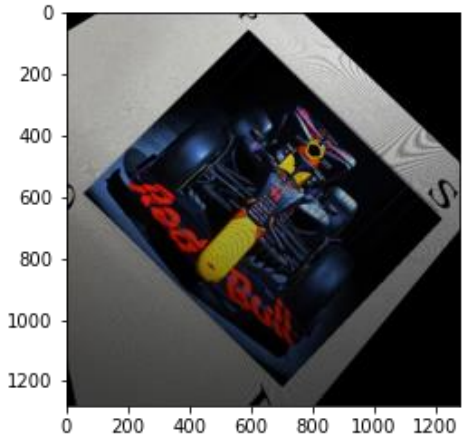


Figure 4. Projection of image 1a on 1c

Q3. This task involves calculating Affine Homographies between pairs of images to project an image onto the other i.e., image 1d to be projected on PQRS frames of 1a, 1b, 1c images respectively.

Affine Homographies are those that have the last row of matrix H as [0 0 1]. Hence, similar to the Homography concept discussed in Q1.

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

$$\text{H} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Solving this like the concept in (1), there are 6 unknowns hence at least 3 points are required to find the unknowns. The obtained matrix is,

$$\text{AH} = \text{B} => \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ x_3 & y_3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_3 & y_3 & 1 \\ x_4 & y_4 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_4 & y_4 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = \begin{bmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ x_3' \\ y_3' \end{bmatrix}$$

For this task, the matrix H here is used to map the image 1d onto 1a/1b/1c. The resulting images obtained can be seen below. We can observe that though the images look same to the ones observed in (q1), the images can change their orientation based on the three points chosen for projection. There is also a possibility of image not being projected completely based on the order of three points chosen for the task.

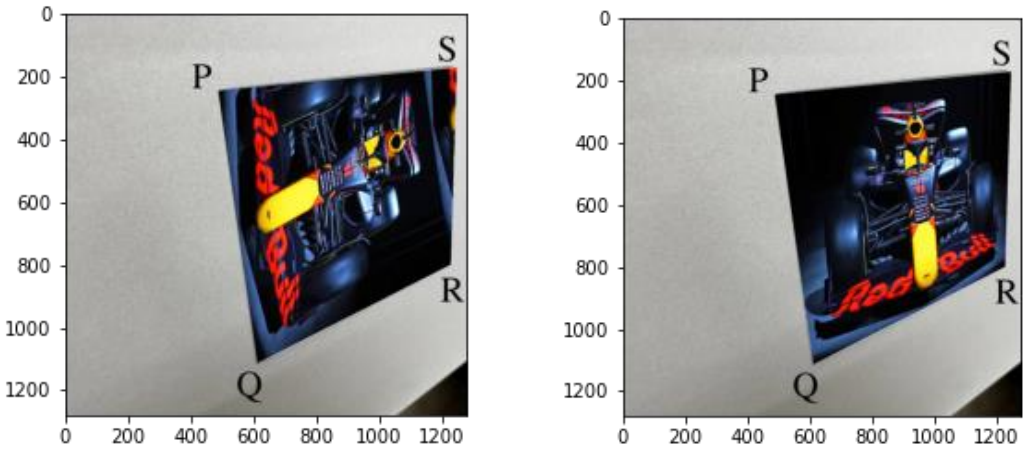


Figure 5. Affine projection of image 1d on 1a in 2 different point considerations

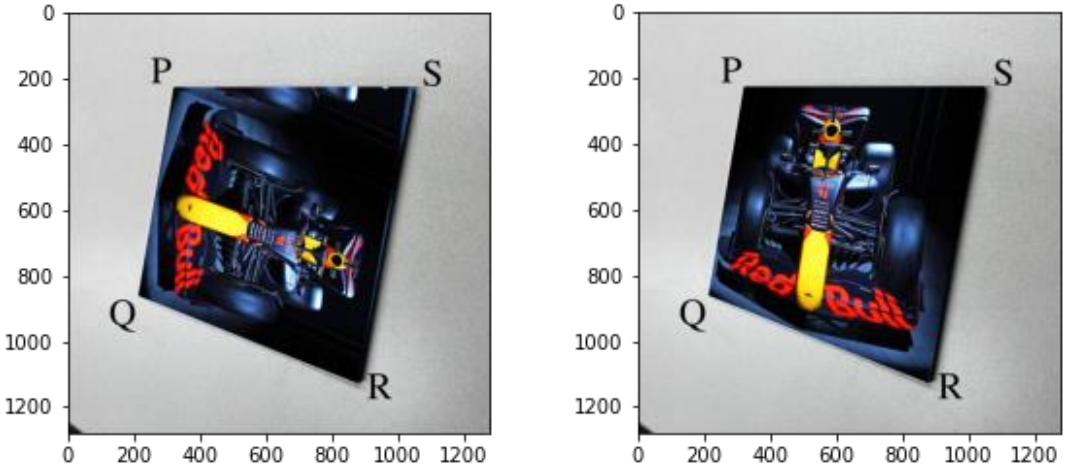


Figure 6. Affine projection of image 1d on 1b in 2 different point considerations

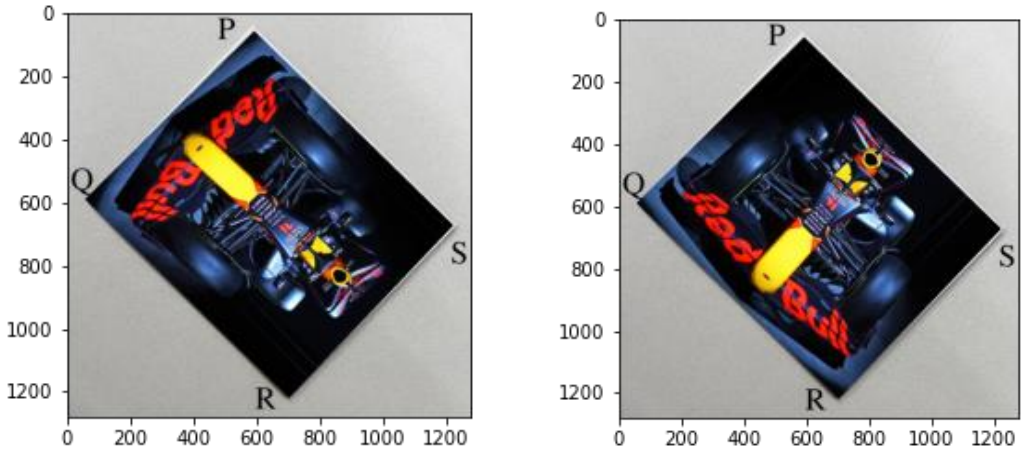
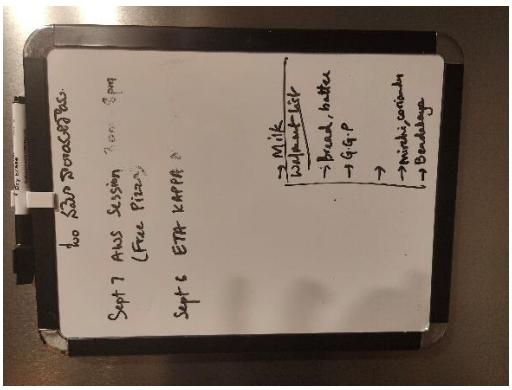


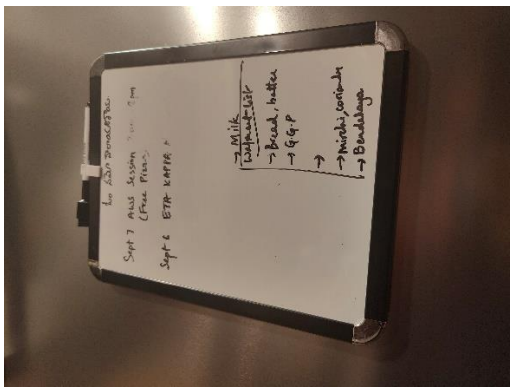
Figure 7. Affine projection of image 1d on 1c in 2 different point considerations

TASK - 2

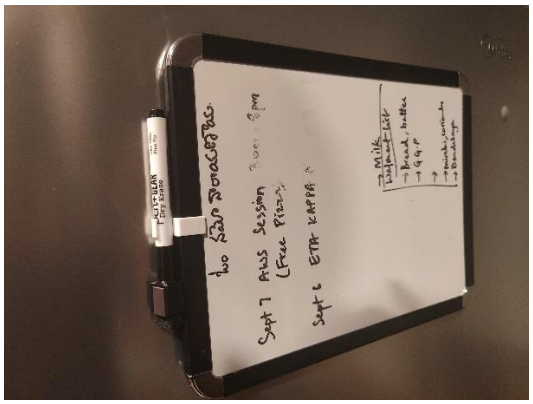
This task is performed similarly to the Task-1 (Q1) with own images chosen as below.



Img (2a)



Img (2b)



Img (2c)

Img (2d)

Performing the similar implementation as shown in task-1 (Q1), with changing only the pixel coordinates of the own image above, the resulting images obtained are as below.

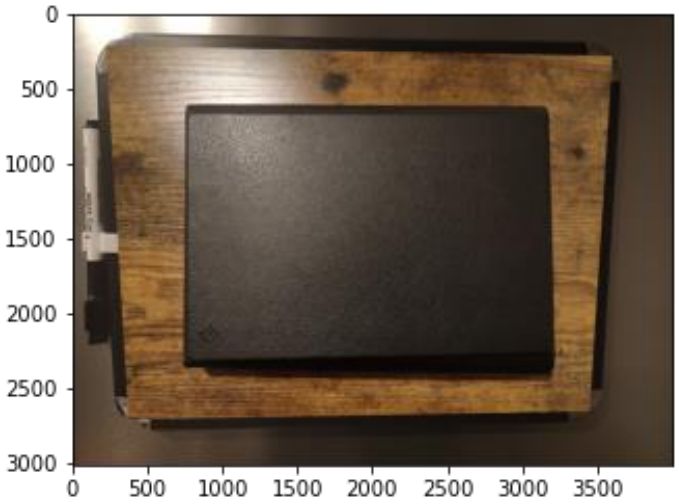


Figure 8. Projection of image 2d on 2a

Figure 9. Projection of image 2d on 2b

Figure 10. Projection of image 2d on 2c

TASK-1(Q1) (Considering 8 points instead of 4 points)

In this task, 8 points i.e 8-pixel coordinates are considered in addition to the corner points/PQRS points. These four additional points are the mid points of the points chosen. This task involves similar implementation like Tash-1 (Q1) with the only difference being the requirement of 16 equations to find 8 unknown points. Hence, int eh matrix AH = B, the matrix A has shape (16 X 8) with matric B has shape (16 X 1).

To compute $H = A^{-1}B$, we use the numpy (np) in-built function np.linalg.pinv which details can be seen in the code attached.

Performing the similar implementation as shown in task-1 (Q1), with changing 8 pixel coordinates, the resulting images obtained are as below.

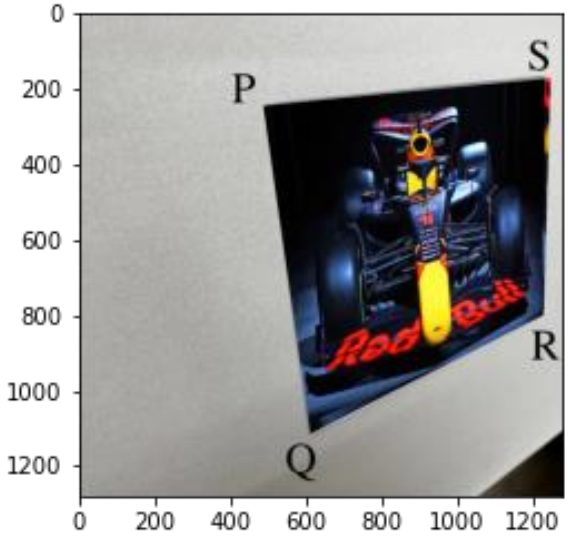


Figure 11. Projection of image 1d on 1c

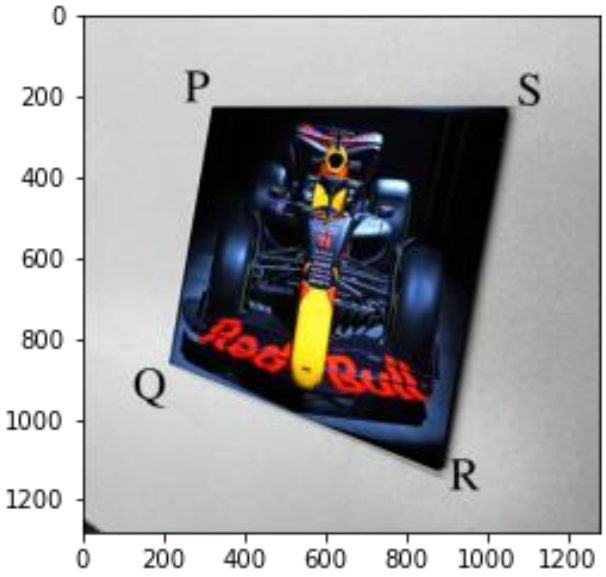


Figure 12. Projection of image 1d on 1c

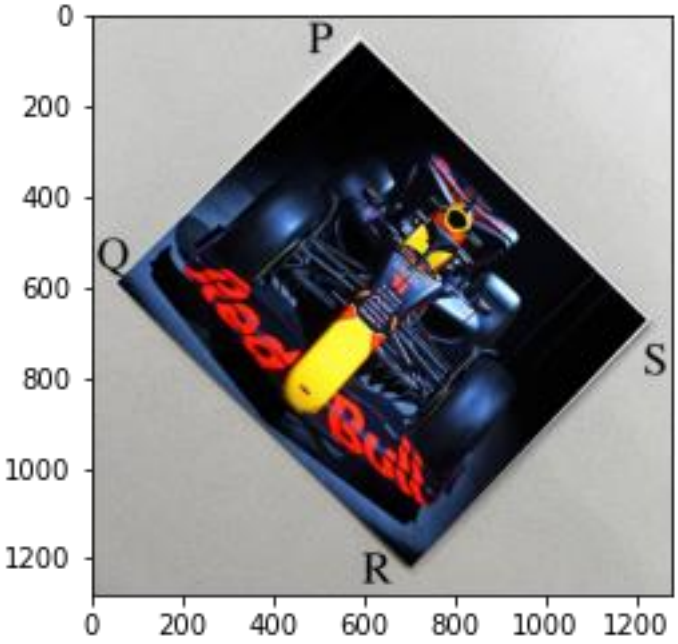


Figure 13. Projection of image 1d on 1c

SOURCE CODES:

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

# <h2><center>ECE661 COMPUTER VISION</center></h2>
# <h3><center>Homework - 2</center></h3>
# <h3><center>Sahithi Kodali - 34789866</center></h3>
# <h3><center>kodali1@purdue.edu</center></h3>

# ## Task 1 (1)

# In[1]:


# Import libraries needed

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


# In[2]:


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

car_img = sk.imread('car.jpg')
card1_img = sk.imread('card1.jpeg')
card2_img = sk.imread('card2.jpeg')
card3_img = sk.imread('card3.jpeg')


# In[3]:


# 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
# For the car (the origianl image that needs to be projected) the four corner
coordinates of the image are considered
# For the cards (the images on whcih original image is projected) the given
PQRS coordinates of the image are considered

car_points = np.array ([[0,0], [0,559],[761,559],[761,0]], np.int32)
card1_points = np.array ([[486,250], [610,1114], [1224,801], [1243,177]],
np.int32)
```

```
card2_points = np.array ([[319,231], [212,863], [875,1129], [1044,231]],
np.int32)
card3_points =  np.array ([[590,62], [55,593], [700,1220], [1219,675]],
np.int32)


# In[4]:


#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 = 20)
    return img_bb

car_bb = boundarybox_check(car_img, car_points)
card1_bb = boundarybox_check(card1_img, card1_points)
card2_bb = boundarybox_check(card2_img, card2_points)
card3_bb = boundarybox_check(card3_img, card3_points)
plt.imshow(card1_bb)


# In[5]:


# 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 original image and
the image on which it is projected

def homo_matrix(X,X_prime):
    A = np.zeros((8,8))

    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]
```

```
    B = X_prime.copy()
    B = B.reshape((-1,1))

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

    H = np.concatenate((K,np.array([[1]])),axis = 0)
    H = H.reshape((3,3))

    return H


# In[6]:


# A masking is performed to the images to determine the focus of interest i.e
the focused projected area on the image
# Below function returns an image mask (white projected area on a black
background) by taking the image and its points as inputs

def masked_image(img, points):
    mask = np.zeros(img.shape)
    return cv2.fillPoly(mask, [points.reshape(-1,1,2)], (255,255,255))


# In[7]:


# With the Homography matrix and the mask being calculated, the mapping
between the original and output frame coordinates is performed.
# Below fucntion takes the Homography matrix [H], original image (car), and
the output frame image (cards) and its mask as input
# The function returns the projected image once the cordinates are mapped
completely

def mapping(H, original, output, output_mask):

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

            if any(output_mask[i,j]) > 0:

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

                mapped_coord = np.matmul(H, coord)
                mapped_coord = (mapped_coord/mapped_coord[2,0]).astype(int)

                if (mapped_coord[0] < original.shape[1] and mapped_coord[1] <
original.shape[0]):
```

```
                    output[i,j] = original[mapped_coord[1], mapped_coord[0]]

    plt.imshow(output)


# In[8]:


#Projection of Image 1d on 1a

H_AD = homo_matrix(card1_points,car_points)
print(H_AD)

card1_mask = masked_image(card1_img, card1_points)
print(plt.imshow(card1_mask))

Image_AD = mapping(H_AD, car_img, card1_img, card1_mask)
filename = 'D:\Purdue\ECE661_CV\HW2_files\Images' + '\Image' + '_AD' +'.png'
plt.savefig(filename)


# In[9]:


#Projection of Image 1d on 1b

H_BD = homo_matrix(card2_points,car_points)
print(H_BD)

card2_mask = masked_image(card2_img, card2_points)
print(plt.imshow(card2_mask))

Image_BD = mapping(H_BD, car_img, card2_img, card2_mask)
filename = 'D:\Purdue\ECE661_CV\HW2_files\Images' + '\Image' + '_BD' +'.png'
plt.savefig(filename)


# In[10]:


#Projection of Image 1d on 1c

H_CD = homo_matrix(card3_points,car_points)
print(H_CD)

card3_mask = masked_image(card3_img, card3_points)
print(plt.imshow(card3_mask))

Image_CD = mapping(H_CD, car_img, card3_img, card3_mask)
```

```
filename = 'D:\Purdue\ECE661_CV\HW2_files\Images' + '\Image' + '_CD' +'.png'
plt.savefig(filename)


# ## TASK 1 (2)

# In[11]:


#Calculate homographies of image pairs (1a,1b) and (1b1c), then multiply them

H_12 = homo_matrix(card1_points, card2_points)
H_23 = homo_matrix(card2_points, card3_points)
H_13 = np.matmul(H_23, H_12)


# In[12]:


# The function returns the projected image once the cordinates are mapped
completely

def mapping_H13(H, original, output):
    blank = np.zeros(output.shape, dtype = 'uint8')
    for i in range(0, original.shape[0]):
        for j in range(0, original.shape[1]):
            coord = np.array([j,i,1])
            coord = coord.reshape((-1,1))

            mapped_coord = np.matmul(H, coord)
            mapped_coord = (mapped_coord/mapped_coord[2,0]).astype(int)

            if (0 <= mapped_coord[0] < output.shape[1] and 0 <=
mapped_coord[1] < output.shape[0]):
                blank[mapped_coord[1], mapped_coord[0]] = original[i,j]

    plt.imshow(blank)


# In[13]:


#Projection of Image 1a on 1c

Image_13 = mapping_H13(H_13, card1_img, card3_img)
filename = 'D:\Purdue\ECE661_CV\HW2_files\Images' + '\Image' + '_13' +'.png'
plt.savefig(filename)


# In[14]:
```

```
#Showing that it is similar to the image projected above (1a on 1c)

plt.imshow(card3_img)


# ## Task 1(3)

# In[95]:


#Calculate Affien matrix

def affine_matrix(X,X_prime):
    A = np.zeros((6,6))

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

    B = X_prime[:3,:].copy()
    B = B.reshape((-1,1))

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

    Af = np.concatenate((K,np.array([[0],[0],[1]])),axis = 0)
    Af = Af.reshape((3,3))

    return Af


# In[96]:


#Projection of Image 1d on 1a

Af_AD = affine_matrix(card1_points,car_points)

Image_Af_AD = mapping(Af_AD, car_img, card1_img, card1_mask)
filename = 'D:\Purdue\ECE661_CV\HW2_files\Images' + '\Image' + 'Af_AD' +'.png'
plt.savefig(filename)


# In[23]:
```

```
#Projection of Image 1d on 1b

Af_BD = affine_matrix(card2_points,car_points)

Image_Af_BD = mapping(Af_BD, car_img, card2_img, card2_mask)
filename = 'D:\Purdue\ECE661_CV\HW2_files\Images' + '\Image' + '_Af_BD'
+'.png'
plt.savefig(filename)


# In[24]:


#Projection of Image 1d on 1c

Af_CD = affine_matrix(card3_points,car_points)

Image_Af_CD = mapping(Af_CD, car_img, card3_img, card3_mask)
filename = 'D:\Purdue\ECE661_CV\HW2_files\Images' + '\Image' + 'Af_CD' +'.png'
plt.savefig(filename)


# ## Task - 2

# In[31]:


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

tab_img = sk.imread('tab.jpg')
board1_img = sk.imread('board1.jpg')
board2_img = sk.imread('board2.jpg')
board3_img = sk.imread('board3.jpg')
# plt.imshow(tab_img)


# In[48]:


# 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
# For the car (the origianl image that needs to be projected) the four corner
coordinates of the image are considered
# For the cards (the images on whcih original image is projected) the given
PQRS coordinates of the image are considered
```

```
tab_points = np.array ([[0,0], [0,2990],[3990,2990],[3990,0]], np.int32)
board1_points = np.array ([[220,224], [372,2704], [3436,2648],[3588,292]],
np.int32)
board2_points = np.array ([[712,444], [700,2116], [2916,2628], [3116,192]],
np.int32)
board3_points =  np.array ([[1316,280], [1480,2940], [3572,2248], [3536,588]],
np.int32)


# In[50]:


#Check points correctness through a boundary box

tab_bb = boundarybox_check(tab_img, tab_points)
board1_bb = boundarybox_check(board1_img, board1_points)
board2_bb = boundarybox_check(board2_img, board2_points)
board3_bb = boundarybox_check(board3_img, board3_points)
# plt.imshow(board3_bb)


# In[51]:


#Projection of Image 2d on 2a

H_t1 = homo_matrix(board1_points,tab_points)
print(H_t1)

board1_mask = masked_image(board1_img, board1_points)
print(plt.imshow(board1_mask))

Image_t1 = mapping(H_t1, tab_img, board1_img, board1_mask)
filename = 'D:\Purdue\ECE661_CV\HW2_files\Images' + '\Image' + '_t1' +'.png'
plt.savefig(filename)


# In[52]:


#Projection of Image 2d on 2b

H_t2 = homo_matrix(board2_points,tab_points)
board2_mask = masked_image(board2_img, board2_points)

Image_t2 = mapping(H_t2, tab_img, board2_img, board2_mask)
filename = 'D:\Purdue\ECE661_CV\HW2_files\Images' + '\Image' + '_t2' +'.png'
plt.savefig(filename)
```

```
# In[54]:


#Projection of Image 2d on 2c

H_t3 = homo_matrix(board3_points,tab_points)
board3_mask = masked_image(board3_img, board3_points)
Image_t3 = mapping(H_t3, tab_img, board3_img, board3_mask)
filename = 'D:\Purdue\ECE661_CV\HW2_files\Images' + '\Image' + '_t3' +'.png'
plt.savefig(filename)


# ## Task - 1 Using 8-points

# In[79]:


#A function to find the mid points of the points of the images take in Task1

def eight_points(points):
    m1 = (points[0] + points[1])/2
    m2 = (points[1] + points[2])/2
    m3 = (points[2] + points[3])/2
    m4 = (points[3] + points[0])/2

    new_points =
np.array([points[0],m1,points[1],m2,points[2],m3,points[3],m4],np.int32)
    return new_points


# In[80]:


#Obtain the 8 points array of the images pixel coordinates

car_8points = eight_points(car_points)
card1_8points = eight_points(card1_points)
card2_8points = eight_points(card2_points)
card3_8points = eight_points(card3_points)


# In[86]:


#Create a boundary box to check the points correctness

car_8bb = boundarybox_check(car_img, car_8points)
card1_8bb = boundarybox_check(card1_img, card1_8points)
card2_8bb = boundarybox_check(card2_img, card2_8points)
card3_8bb = boundarybox_check(card3_img, card3_8points)
```

```
# plt.imshow(card3_bb)


# In[88]:


#Calculate the Affine Transformation matrix H

def homo_8matrix(X,X_prime):
    A = np.zeros((16,8))

    for i in range(8):
        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]

    print(A.shape)
    B = X_prime.copy()
    B = B.reshape((-1,1))

    #To calculate H = A^-1B
    K = np.dot(np.linalg.pinv(A), B)

    H = np.concatenate((K,np.array([[1]])),axis = 0)
    H = H.reshape((3,3))

    return H


# In[90]:


homo_8matrix(card1_8points, car_8points)


# In[91]:


#Projection of Image 1d on 1a

H_AD_8p = homo_matrix(card1_8points,car_8points)
print(H_AD_8p)
```

```
card1_8mask = masked_image(card1_img, card1_8points)
print(plt.imshow(card1_8mask))

Image_AD_8p = mapping(H_AD_8p, car_img, card1_img, card1_8mask)
filename = 'D:\Purdue\ECE661_CV\HW2_files\Images' + '\Image' + '_AD_8p'
+'.png'
plt.savefig(filename)


# In[92]:


#Projection of Image 1d on 1b

H_BD_8p = homo_matrix(card2_8points,car_8points)
print(H_BD_8p)

card2_8mask = masked_image(card2_img, card2_8points)
print(plt.imshow(card2_8mask))

Image_BD_8p = mapping(H_BD_8p, car_img, card2_img, card2_8mask)
filename = 'D:\Purdue\ECE661_CV\HW2_files\Images' + '\Image' + '_BD_8p'
+'.png'
plt.savefig(filename)


# In[94]:


#Projection of Image 1d on 1c

H_CD_8p = homo_matrix(card3_8points,car_8points)
print(H_CD_8p)

card3_8mask = masked_image(card3_img, card3_8points)
print(plt.imshow(card1_8mask))

Image_CD_8p = mapping(H_CD_8p, car_img, card3_img, card3_8mask)
filename = 'D:\Purdue\ECE661_CV\HW2_files\Images' + '\Image' + '_CD_8p'
+'.png'
plt.savefig(filename)


# In[ ]:
```