HOMEWORK 2

Alexandre Olivé Pellicer

aolivepe@purdue.edu

Computing the homography matrix H

We know that we can map a point from the domain plain to the range plain by using homogeneous coordinates and applying a homograpky H as:

$$\begin{pmatrix} x_1' \\ x_2' \\ x_3' \end{pmatrix} = H \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

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

From this product of matrices, we can write the following equations:

$$x_1' = ax_1 + bx_2 + cx_3$$

$$x_2' = dx_1 + ex_2 + fx_3$$

$$x_3' = gx_1 + hx_2 + ix_3$$

Knowing that $x' = \frac{x_1'}{x_3'}$ and $y' = \frac{x_2'}{x_3'}$ we can write:

$$x' = \frac{x_1'}{x_3'} = \frac{ax_1 + bx_2 + cx_3}{gx_1 + hx_2 + ix_3}$$

$$y' = \frac{x_2'}{x_3'} = \frac{dx_1 + ex_2 + fx_3}{gx_1 + hx_2 + ix_3}$$

Dividing numerator and denominator by 3 and setting i=1 since we are just interested in ratios:

$$x' = \frac{ax + by + c}{gx + hy + 1}$$

$$y' = \frac{dx + ey + f}{gx + hy + 1}$$

We can write this as:

$$x' = ax + by + c - gxx' + hyx'$$

$$y' = dx + ey + f - gxy' - hyy'$$

In these 2 equations we have in total 8 unknowns corresponding to the 9 coefficients of the matrix H since we have set i=1. To find these 8 unknowns, we need 8 equations. By taking 4 points from the domain plain and looking at which would be their corresponding points in the range plain, we will get 8 equations that will allow us to find the remaining 8 coefficients of the matrix H. We can write these 8 equations as a product of matrixes where x_i' and y_i' correspond to the points in the 2D range plain and x_i and y_i correspond to the points in the 2D domain plain:

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

We can represent this product of matrices as $\mathrm{Ah} = \mathrm{b}$. Thus, if we solve $h = A^{-1}b$ we will find the remaining coefficients of the matrix H. This method will be used for tasks 1.1 and 1.2

Computing the affine homography matrix H

For task 1.3, we know that the affine transformation can be described as

$$\begin{pmatrix} x_1' \\ x_2' \\ x_3' \end{pmatrix} = H \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

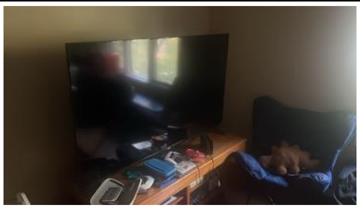
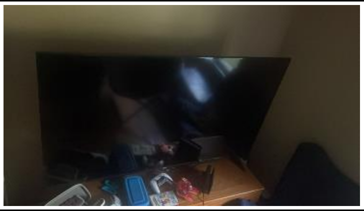
$$H = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}$$

Therefore, now we just need to find 6 coefficients of the matrix H. Following the same steps as done before, we can obtain these coefficients solving the following product of matrices:

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

Task 1 and 2

The explanation is done by talking about Alex Honnold's image and images a), b), c) provided in the instructions. Nevertheless, the same procedure is followed for Task 2 using the image of the celebrity Pitbull and the three images of a TV captured with my phone.

Pitbull's image	Image TV 1	Image TV 2	Image TV 3
			

These are the P, Q, R, S points that has been used for Task 1:

Alex Honnold's image	Image a	Image b	Image c

p = [0, 0]
q = [783, 0]
r = [783, 665]
s = [0, 665]
	p_prima1 = [500, 870]
	q_prima1 = [2529, 900]
	r_prima1 = [2382, 2282]
	s_prima1 = [700, 3135]
		p_prima2 = [528, 1442]
		q_prima2 = [1847, 844]
		r_prima2 = [1940, 2803]
		s_prima2 = [473, 2674]
			p_prima3 = [1217, 569]
			q_prima3 = [2923, 2310]
			r_prima3 = [1790, 3178]
			s_prima3 = [263, 1796]

These are the P, Q, R, S points that has been used for Task 2:

Pitbull's image	Image TV 1	Image TV 2	Image TV 3

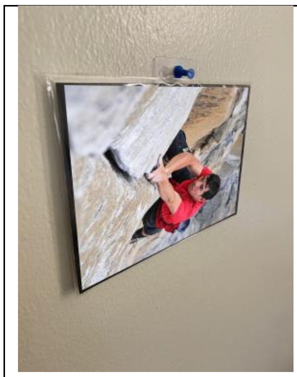
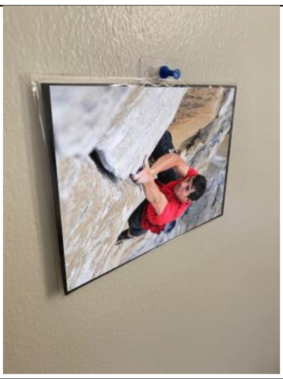
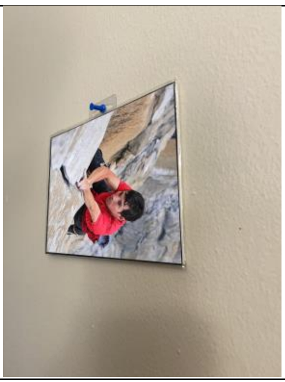
p = [0, 0]
q = [1200, 0]
r = [1200, 747]
	p_prima1 = [377, 230]	p_prima2 = [196, 279]	p_prima3 = [223, 142]

s = [0, 747]	q_prima1 = [1303, 170]
	r_prima1 = [1256, 673]
	s_prima1 = [450, 1069]
		q_prima2 = [1644, 315]
		r_prima2 = [1410, 845]
		s_prima2 = [247, 1051]
			q_prima3 = [1633, 123]
			r_prima3 = [1553, 1029]
			s_prima3 = [391, 797]

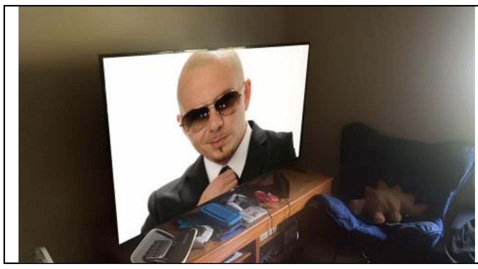
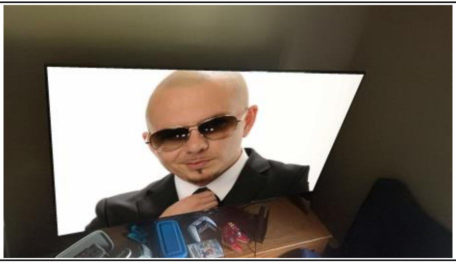
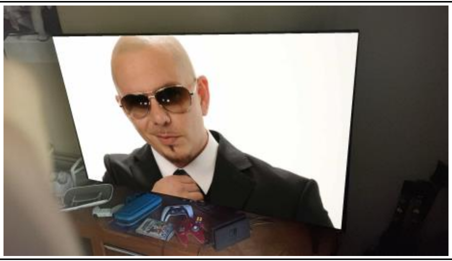
Task 1.1 and 2.1

These are the steps that we follow in order to project the image of Alex Honnold over the images a), b) and c) provided:

a. Record the pixel coordinates PQRS from images a), b), c) using the 4th approach proposed in the instructions explained in "Displaying the coordinates of the points clicked on the image using Python-OpenCV." For the case of the Alex Honnold's image we pick the extremes of the image
b. Compute the H matrix as described in section "Computing the homography matrix H". For the 3 homographies, we will use the points p, q, r, s as the points in the domain plain and the points p_prima_i, q_prima_i, r_prima_i and s_prima_i as the points in the range plain for i = 1, 2, 3 according to the homography that we are calculating.
c. We create a mask for each of the background images a), b) and c). It will have the same dimensions as the background images and it has been created using the Python library "polygon". The mask is characterized by being "True" in the region where Alex Honnold's image will be projected and "False" for the remaining pixels.
d. We apply the homography. We determine the (x, y) point in the source image for each point (x', y') in the PQRS area using H^{-1}. Basically, we multiply the pixels of the Alex Honnold's image by H^{-1}. We will only apply the homography to those points that are part of the area with "True" pixels of the mask. Finally, we set the pixel value at (x', y') to the pixel value at (x, y).
e. These are the obtained results:
   Task 1:

Task 2:

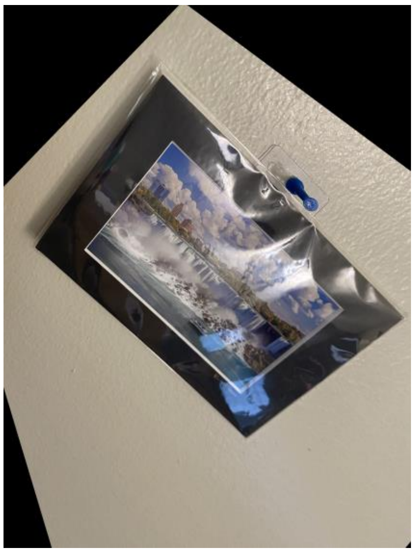
		

Task 1.2 and 2.2

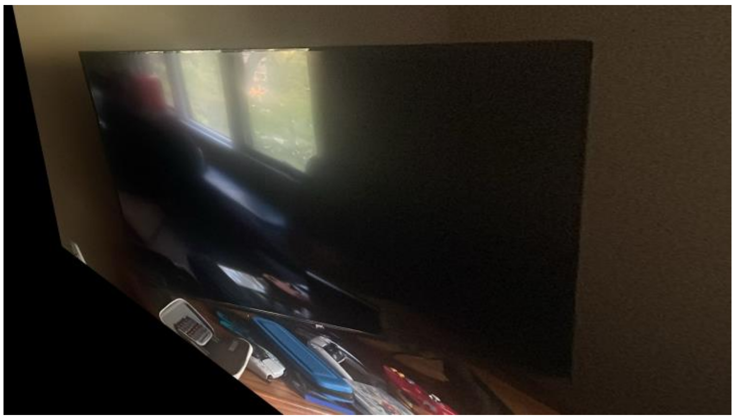
These are the steps that we follow:

a. We first compute the homography between image a) and image b) using the points p_prima_1, q_prima_1, r_prima_1 and s_prima_1 as the points in the domain plain and the points p_prima_2, q_prima_2, r_prima_2 and s_prima_2 as the points in the range plain.
b. Then we compute the homography between image b) and image c) using the points p_prima_2, q_prima_2, r_prima_2 and s_prima_2as the points in the domain plain and the points p_prima_3, q_prima_3, r_prima_3 and s_prima_3 as the points in the range plain.
c. We multiply both homographies and we obtain the homography that when applied in image a) will deformate it so that it looks like image c).
d. We apply the final homography following the same steps as described in Task 1.1 and Task 2.1
e. These are the obtained results:

   Task 1:



   Task 2:

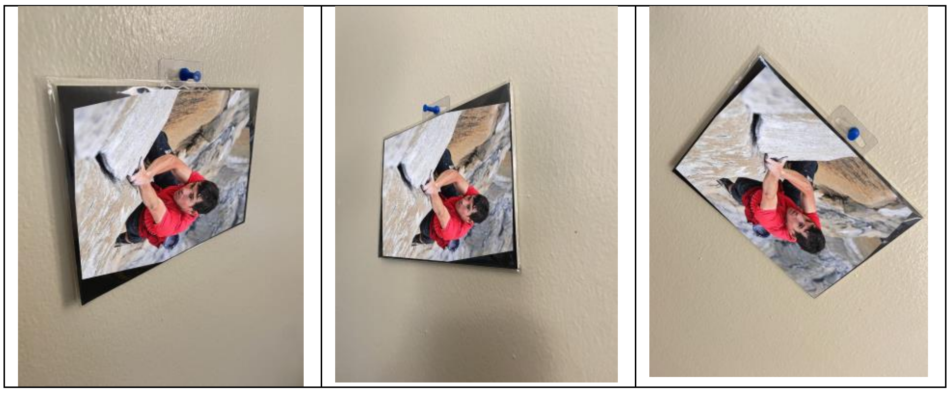


We can see that the resulting images are similar to image c) for task 1 and image TV 3 for task 2.

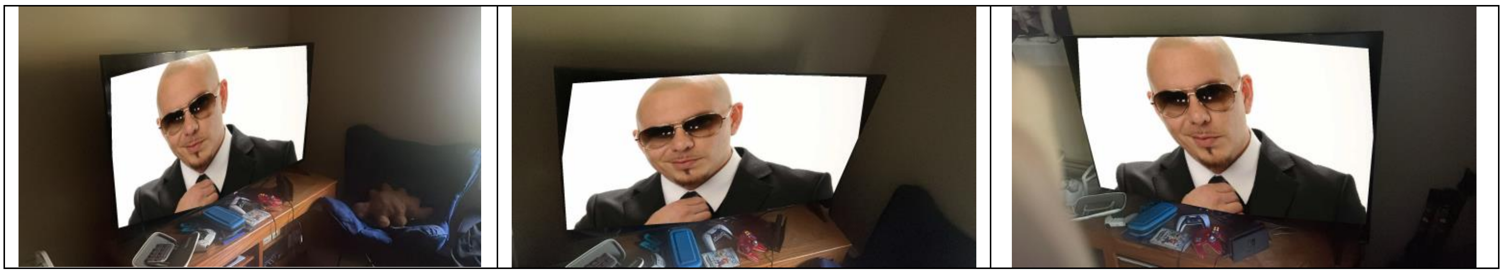
<u>Task 1.3 and 2.3</u>

Here we use the same steps as described in 1). The only difference is that now, instead of computing the projective homography that fits better, we will compute the affine homography that fits better in the region of interest. In order to do it, we will compute H as described in the section "Computing the affine homography matrix H". The other steps are exactly the same as used in Task 1.1 and Task 1.2. These are the obtained results:

Task 1:



Task 2:



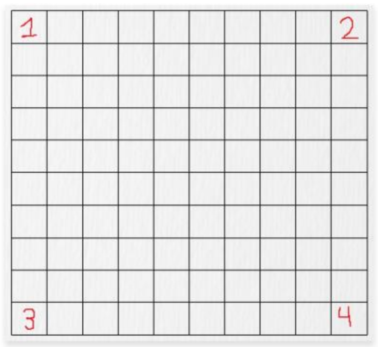
<u>Analysis of results obtained from task 1.3:</u>

The results are not good. The only one that offers a better performance is for image c). For images a) and b) there are considerable big gaps of the ROI that have not been covered. This is because the ROI of image c) is the one where the edges are more parallel. Therefore, when applying the affine transformation, in which parallel lines map into parallel lines, the performance is still decent and the number of gaps is reduced. For images a) and b) the edges are not that parallel and thus there are bigger gaps. We can conclude that the more parallel the edges of the ROI, the better will be the performance when just using affine homographies.

Analysis of results obtained from task 2.3:
In this case we see that the performance is low in all three images since in none of them the ROI is limited by parallel lines. Therefore, in the three images we can see gaps that are not covered by the projected image. The reasoning is the same as described for the results using the images from task 1.3.

Extra credit:

Original input image:



Formulating the parameterized homographies:

These are the parameterized homographies that we will use for each of the transformations described in the instructions

Parameterized homography for rotation:

$$H(\alpha) = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Looking at the parameterized homography for rotation, the 2x2 matrix formed by the cosine and sinus is the one that will apply the rotation. The remaining coefficients represent the homogeneous coordinates of the system for affine transformation and projection. As when rotating the image we are just doing similarity transform where angles keep the same, we just need the 2x2 matrix already explained.

Parameterized homography for vertical tilt:

$$H(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & \text{-}\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix}$$

Parameterized homography for horizontal tilt

$$H(\alpha) = \begin{bmatrix} \cos(\alpha) & 0 & \text{-}\sin(\alpha) \\ 0 & 1 & 0 \\ \sin(\alpha) & 0 & \cos(\alpha) \end{bmatrix}$$

The process that is used to apply the transformation is the following: 1) Shift the image to the origin by using a normalization matrix, denoted as $\mathrm{H}_{\mathrm{norm}}$. 2) Implement the transformation matrix on the shifted image. 3)Move the image back to its initial position by applying a denormalization matrix, $\mathrm{H}_{\text{de-norm}}$

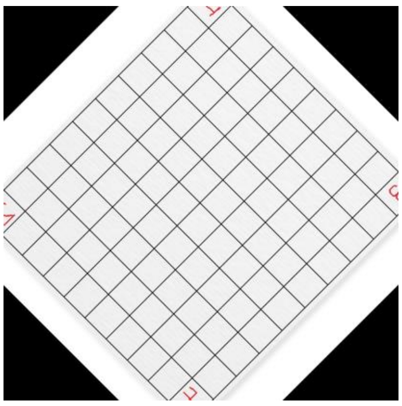
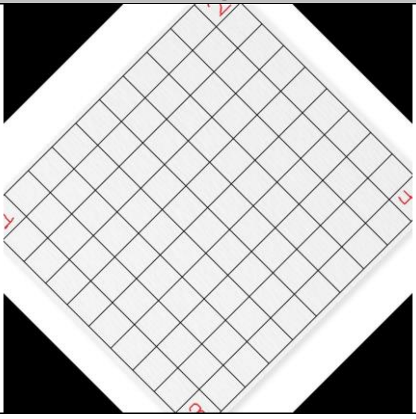
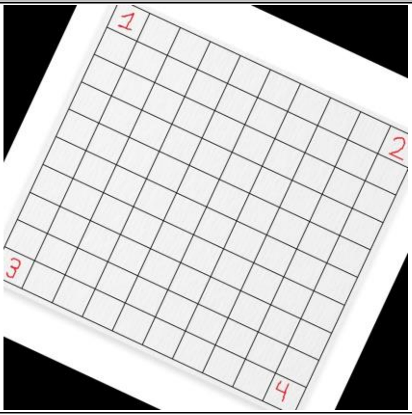
The resulting combined transformation matrix H_{c} is expressed as:

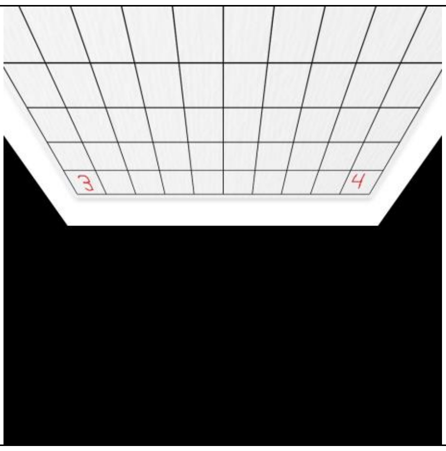
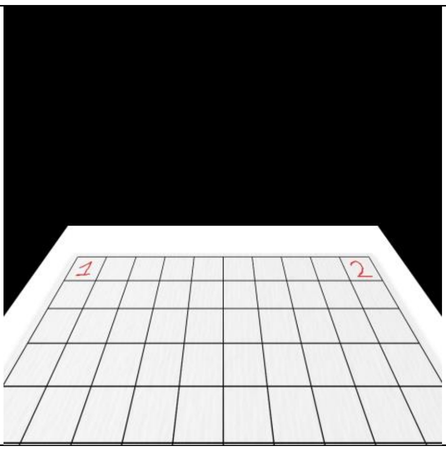
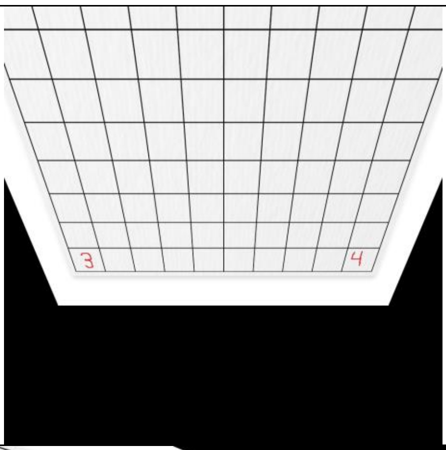
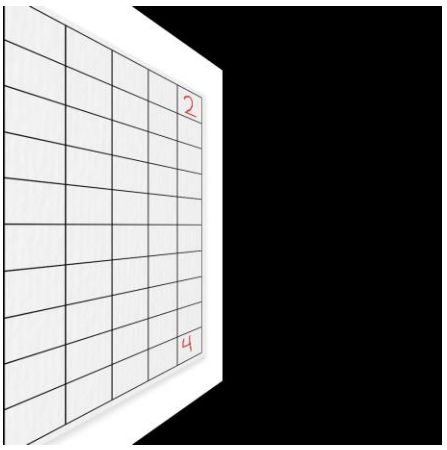
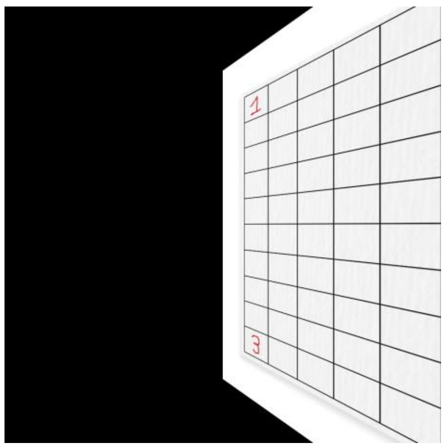
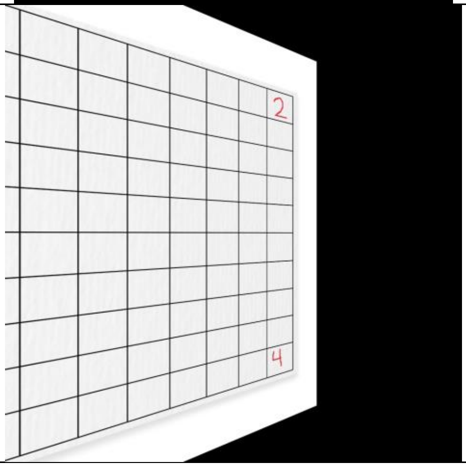
$$\mathrm{H}_{\mathrm{c}} = \mathrm{H}_{\text{de-norm}} H(\alpha) \mathrm{H}_{\mathrm{norm}}$$

We finally use the cv2.warpPerspective() function to create the final output image.

Application of the homographies:

These are some results that we have obtained when taking different values of alpha:

	45 degrees	315 degrees	25 degrees
Rotation			

Vertical			
Horizontal			

Analysis of the effects:

Analyzing the obtained results and the changes in direction of the lines when applying the homographies we obtain the following conclutions:

a) When applying rotation, as shown in the first of the table row above, vertical and horizontal lines change direction. This is because we are rotating the entire image.
b) When applying vertical tilt, as shown in the second row of the table above, vertical lines change direction while horizontal lines do not change direction. This is because the image is only rotated vertically. Horizontal lines remain parallel while vertical lines do not and would intersect in a vanishing point.
c) When applying horizontal tilt, as shown in the third row of the table above, horizontal lines change direction while vertical lines do not change direction. This is because the image is only rotated horizontally. Vertical lines remain parallel while horizontal lines do not and would intersect in a vanishing point.

Task 1 and 2 CODE:

```
from skimage.draw import polygon
```

```
import cv2

import numpy as np


## COORDINATES ---------------------

# function to display the coordinates of

# of the points clicked on the image

def click_event(event, x, y, flags, params):


    # checking for left mouse clicks

    if event == cv2.EVENT_LBUTTONDOWN:


        # displaying the coordinates

        # on the Shell

        print(x, ' ', y)


        # displaying the coordinates

        # on the image window

        font = cv2.FONT_HERSHEY_SIMPLEX

        cv2.putText(img, str(x) + ',' +

                    str(y), (x,y), font,

                    1, (255, 0, 0), 2)

        cv2.imshow('image', img)


    # checking for right mouse clicks

    if event==cv2.EVENT_RBUTTONDOWN:


        # displaying the coordinates

        # on the Shell
        print(x, ' ', y)

        # displaying the coordinates
        # on the image window
        font = cv2.FONT_HERSHEY_SIMPLEX
        b = img[y, x, 0]
        g = img[y, x, 1]
        r = img[y, x, 2]
        cv2.putText(img, str(b) + ',' +
                    str(g) + ',' + str(r),
                    (x,y), font, 1,
                    (255, 255, 0), 2)
        cv2.imshow('image', img)

# driver function
if __name__=="__main__":

    # reading the image
    img = cv2.imread('/mnt/c/Users/Hp/Desktop/MSECE/Computer-
Vision/HW2/HW2_images/img3.jpg', 1)

    # displaying the image
    cv2.imshow('image', img)

    # setting mouse handler for the image
    # and calling the click_event() function
    cv2.setMouseCallback('image', click_event)
    # wait for a key to be pressed to exit
    cv2.waitKey(0)

    # close the window
    cv2.destroyAllWindows()

#TASK 1 images and points ------------------------------
alex = cv2.imread('/mnt/c/Users/Hp/Desktop/MSECE/Computer-
Vision/HW2/HW2_images/alex_honnold.jpg')
p = [0, 0]
q = [783, 0]
r = [783, 665]
s = [0, 665]

img1 = cv2.imread('/mnt/c/Users/Hp/Desktop/MSECE/Computer-
Vision/HW2/HW2_images/img1.jpg')

p_prima1 = [500, 870]
q_prima1 = [2529, 900]
r_prima1 = [2382, 2282]
s_prima1 = [700, 3135]

img2 = cv2.imread('/mnt/c/Users/Hp/Desktop/MSECE/Computer-
Vision/HW2/HW2_images/img2.jpg')
p_prima2 = [528, 1442]
q_prima2 = [1847, 844]
r_prima2 = [1940, 2803]
s_prima2 = [473, 2674]
```

```
img3 = cv2.imread('/mnt/c/Users/Hp/Desktop/MSECE/Computer-Vision/HW2/HW2_images/img3.jpg')

p_prima3 = [1217, 569]

q_prima3 = [2923, 2310]

r_prima3 = [1790, 3178]

s_prima3 = [263, 1796]


#TASK 2 images and points ----------------------------------------

alex = cv2.imread('/mnt/c/Users/Hp/Desktop/MSECE/Computer-Vision/HW2/HW2_images/pitbull.jpg')

p = [0, 0]

q = [1200, 0]

r = [1200, 747]

s = [0, 747]


img1 = cv2.imread('/mnt/c/Users/Hp/Desktop/MSECE/Computer-Vision/HW2/HW2_images/mine_img1.jpg')


p_prima1 = [377, 230]

q_prima1 = [1303, 170]

r_prima1 = [1256, 673]

s_prima1 = [450, 1069]


img2 = cv2.imread('/mnt/c/Users/Hp/Desktop/MSECE/Computer-Vision/HW2/HW2_images/mine_img2.jpg')

p_prima2 = [196, 279]

q_prima2 = [1644, 315]

r_prima2 = [1410, 845]

s_prima2 = [247, 1051]
```

```
img3 = cv2.imread('/mnt/c/Users/Hp/Desktop/MSECE/Computer-
Vision/HW2/HW2_images/mine_img3.jpg')

p_prima3 = [223, 142]

q_prima3 = [1633, 123]

r_prima3 = [1553, 1029]

s_prima3 = [391, 797]

# ---------------------------


pqrs = [p[0], p[1], s[0], s[1], r[0], r[1], q[0], q[1]]

pqrs_prima1 = [p_prima1[0], p_prima1[1], s_prima1[0], s_prima1[1], r_prima1[0],
r_prima1[1], q_prima1[0], q_prima1[1]]

pqrs_prima2 = [p_prima2[0], p_prima2[1], s_prima2[0], s_prima2[1], r_prima2[0],
r_prima2[1], q_prima2[0], q_prima2[1]]

pqrs_prima3 = [p_prima3[0], p_prima3[1], s_prima3[0], s_prima3[1], r_prima3[0],
r_prima3[1], q_prima3[0], q_prima3[1]]


# TASK 1 code -----------------------------------------------------

def obtain_h(x_prima, x):

  # compute h following the mathematics of the report

  a = [

  [x[0], x[1], 1, 0, 0, 0, -x[0]*x_prima[0], -x[1]*x_prima[0]],

  [0, 0, 0, x[0], x[1], 1, -x[0]*x_prima[1], -x[1]*x_prima[1]],

  [x[2], x[3], 1, 0, 0, 0, -x[2]*x_prima[2], -x[3]*x_prima[2]],

  [0, 0, 0, x[2], x[3], 1, -x[2]*x_prima[3], -x[3]*x_prima[3]],

  [x[4], x[5], 1, 0, 0, 0, -x[4]*x_prima[4], -x[5]*x_prima[4]],

  [0, 0, 0, x[4], x[5], 1, -x[4]*x_prima[5], -x[5]*x_prima[5]],

  [x[6], x[7], 1, 0, 0, 0, -x[6]*x_prima[6], -x[7]*x_prima[6]],

  [0, 0, 0, x[6], x[7], 1, -x[6]*x_prima[7], -x[7]*x_prima[7]]
```

```
    ]
    b = x_prima


    a_inverse = np.linalg.inv(a)


    # h = A^{-1}b
    h = np.dot(a_inverse, b)


    #Add known coefficient and reshape
    h = np.append(h, 1)
    h = h.reshape(3, 3)
    return h


# compute mask where true is the area where we want to project alex's img
def obtain_mask(pqrs, new):
    mask = np.zeros((new.shape[0], new.shape[1]), dtype=bool)
    w = pqrs[0::2]
    h = pqrs[1::2]
    hs, ws = polygon(h, w)
    mask[hs, ws] = True
    return mask


# Determine the (x, y) point in the original img for each point (x', y') in the PQRS area
def obtain_point(x_, y_, h_inverse):
    x = [x_, y_, 1]
    new_point_3d = np.dot(h_inverse, x)
    new_x = int(new_point_3d[0] / new_point_3d[2])
    new_y = int(new_point_3d[1] / new_point_3d[2])
    return new_x, new_y


# apply projection of alex's image onto background image
def generate_final_image(new, original, h, mask, idx):
    h_inverse = np.linalg.inv(h)


    width_original = original.shape[0]
    height_original = original.shape[1]
    width_destination = new.shape[0]
    height_destination = new.shape[1]


    for col in range(height_destination):
        for row in range(width_destination):
            # check if the point is in the area where we want to project
            if mask[row, col]:
                new_x, new_y = obtain_point(col, row, h_inverse)
                if 0 < new_y < width_original and 0 < new_x < height_original:
                    new[row, col] = original[new_y, new_x]
                else:
                    print("DONT REMOVE")


    cv2.imwrite(f'task_2_1_img_{idx}.jpg', new)


#compute homography
h = obtain_h(pqrs_prima1, pqrs)
#obtain mask of points of interest in range domain
mask = obtain_mask(pqrs_prima1, img1)
#apply homography
```

```
generate_final_image(img1, alex, h, mask, 1)


#compute homography
h = obtain_h(pqrs_prima2, pqrs)
#obtain mask of points of interest in range domain
mask = obtain_mask(pqrs_prima2, img2)
#apply homography
generate_final_image(img2, alex, h, mask, 2)


#compute homography
h = obtain_h(pqrs_prima3, pqrs)
#obtain mask of points of interest in range domain
mask = obtain_mask(pqrs_prima3, img3)
#apply homography
generate_final_image(img3, alex, h, mask, 3)


# ------------------------------------------------------------


# TASK 2 code ------------------------------------------------
def obtain_h(x_prima, x):
  # compute h following the mathematics of the report
  a = [
  [x[0], x[1], 1, 0, 0, 0, -x[0]*x_prima[0], -x[1]*x_prima[0]],
  [0, 0, 0, x[0], x[1], 1, -x[0]*x_prima[1], -x[1]*x_prima[1]],
  [x[2], x[3], 1, 0, 0, 0, -x[2]*x_prima[2], -x[3]*x_prima[2]],
  [0, 0, 0, x[2], x[3], 1, -x[2]*x_prima[3], -x[3]*x_prima[3]],
  [x[4], x[5], 1, 0, 0, 0, -x[4]*x_prima[4], -x[5]*x_prima[4]],
  [0, 0, 0, x[4], x[5], 1, -x[4]*x_prima[5], -x[5]*x_prima[5]],
  [x[6], x[7], 1, 0, 0, 0, -x[6]*x_prima[6], -x[7]*x_prima[6]],
  [0, 0, 0, x[6], x[7], 1, -x[6]*x_prima[7], -x[7]*x_prima[7]]
  ]
  b = x_prima

  a_inverse = np.linalg.inv(a)

  # h = A^{-1}b
  h = np.dot(a_inverse, b)

  #Add known coefficient and reshape
  h = np.append(h, 1)
  h = h.reshape(3, 3)
  return h

# Determine the (x, y) point in the original img for each point (x', y') in the PQRS area
def obtain_point(x_, y_, h_inverse):
  x = [x_, y_, 1]
  new_point_3d = np.dot(h_inverse, x)
  new_x = int(new_point_3d[0] / new_point_3d[2])
  new_y = int(new_point_3d[1] / new_point_3d[2])
  return new_x, new_y

# apply projection of alex's image onto background image
def generate_final_image(new, original, h):
  h_inverse = np.linalg.inv(h)

  width_original = original.shape[0]
  height_original = original.shape[1]

  width_destination = new.shape[0]

  height_destination = new.shape[1]


  for col in range(height_destination):

      for row in range(width_destination):

          new_x, new_y = obtain_point(col, row, h_inverse)

          if 0 < new_y < width_original and 0 < new_x < height_original:

              new[row, col] = original[new_y, new_x]

          else:

              print("DONT REMOVE")


  cv2.imwrite(f'task_2_2.jpg', new)


#compute homography to go from img a) to b)

h1 = obtain_h(pqrs_prima2, pqrs_prima1)

#compute homography to go from img b) to c)

h2 = obtain_h(pqrs_prima3, pqrs_prima2)

#apply homography resulting of the product of h1 and h2. It will be the homography to
go from img a) to c)

generate_final_image(np.zeros_like(img1), img1, np.matmul(h2, h1))

# -------------------------------------------------------------


# TASK 3 code -----------------------------------------------

def obtain_h_affine(x_prima, x):

  # compute affine h following the mathematics of the report

  a = [

  [x[0], x[1], 1, 0, 0, 0],
  [0, 0, 0, x[0], x[1], 1],
  [x[2], x[3], 1, 0, 0, 0],
  [0, 0, 0, x[2], x[3], 1],
  [x[4], x[5], 1, 0, 0, 0],
  [0, 0, 0, x[4], x[5], 1],
  [x[6], x[7], 1, 0, 0, 0],
  [0, 0, 0, x[6], x[7], 1]
  ]

  b = x_prima

  a_inverse = np.linalg.pinv(a)

  # h = A^{-1}b
  h = np.dot(a_inverse, b)

  #Add known coefficients and reshape
  h = np.append(h, (0, 0, 1))
  h = h.reshape(3, 3)
  return h

# compute mask where true is the area where we want to project alex's img
def obtain_mask(pqrs, new):
  mask = np.zeros((new.shape[0], new.shape[1]), dtype=bool)
  w = pqrs[0::2]
  h = pqrs[1::2]
  hs, ws = polygon(h, w)
  mask[hs, ws] = True
  return mask


# Determine the (x, y) point in the original img for each point (x', y') in the PQRS area
def obtain_point(x_, y_, h_inverse):
  x = [x_, y_, 1]
  new_point_3d = np.dot(h_inverse, x)
  new_x = int(new_point_3d[0] / new_point_3d[2])
  new_y = int(new_point_3d[1] / new_point_3d[2])
  return new_x, new_y


# apply projection of alex's image onto background image
def generate_final_image(new, original, h, mask, idx):
  h_inverse = np.linalg.inv(h)


  width_original = original.shape[0]
  height_original = original.shape[1]
  width_destination = new.shape[0]
  height_destination = new.shape[1]


  for col in range(height_destination):
      for row in range(width_destination):
          # check if the point is in the area where we want to project
          if mask[row, col]:
              new_x, new_y = obtain_point(col, row, h_inverse)
              if 0 < new_y < width_original and 0 < new_x < height_original:
                  new[row, col] = original[new_y, new_x]
              else:
                  print("DONT REMOVE")
    cv2.imwrite(f'task_2_3_img_{idx}.jpg', new)

#compute affine homography
h = obtain_h_affine(pqrs_prima1, pqrs)
#obtain mask of points of interest in range domain
mask = obtain_mask(pqrs_prima1, img1)
#apply homography
generate_final_image(img1, alex, h, mask, 1)

#compute affine homography
h = obtain_h_affine(pqrs_prima2, pqrs)
#obtain mask of points of interest in range domain
mask = obtain_mask(pqrs_prima2, img2)
#apply homography
generate_final_image(img2, alex, h, mask, 2)

#compute affine homography
h = obtain_h_affine(pqrs_prima3, pqrs)
#obtain mask of points of interest in range domain
mask = obtain_mask(pqrs_prima3, img3)
#apply homography
generate_final_image(img3, alex, h, mask, 3)
# ------------------------------------------------------------
```

Extra credit CODE:

```
import numpy as np
import cv2
```

```
def Rotation(img, alpha):
    height = img.shape[0]
    width = img.shape[1]

    # Compute cosine and sinus of alpha
    cos_alpha = np.cos(np.deg2rad(alpha))
    sin_alpha = np.sin(np.deg2rad(alpha))

    # Create parameterized homography and compute the composite homography
    h_alpha = np.array([[cos_alpha,-sin_alpha, 0],[sin_alpha, cos_alpha, 0],[0, 0, 1]])
    h_norm = np.array([[2/width, 0,-1],[0, 2/height,-1],[0, 0, 1]])
    h_denorm = np.array([[width/2, 0, width/2],[0, height/2, height/2],[0, 0, 1]])
    h_c_alpha = np.matmul(np.matmul(h_denorm, h_alpha), h_norm)

    # Create output image
    rotation_image = cv2.warpPerspective(img, h_c_alpha, (height, width))
    cv2.imwrite('/mnt/c/Users/Hp/Desktop/MSECE/Computer-
Vision/HW2/HW2_output_images/extracredit_rotation_image.jpg', rotation_image)

def Vertical(img, alpha):
    height = img.shape[0]
    width = img.shape[1]

    # Compute cosine and sinus of alpha
    cos_alpha = np.cos(np.deg2rad(alpha))
    sin_alpha = np.sin(np.deg2rad(alpha))
    # Create parameterized homography and compute the composite homography
    h_alpha = np.array([[1, 0, 0], [0, cos_alpha,-sin_alpha],[0, sin_alpha, cos_alpha]])
    h_norm = np.array([[2/width, 0,-1],[0, 2/height,-1],[0, 0, 1]])
    h_denorm = np.array([[width/2, 0, width/2],[0, height/2, height/2],[0, 0, 1]])
    h_c_alpha = np.matmul(np.matmul(h_denorm, h_alpha), h_norm)

    # Create output image
    vertical_image = cv2.warpPerspective(img, h_c_alpha, (height, width))
    cv2.imwrite('/mnt/c/Users/Hp/Desktop/MSECE/Computer-
Vision/HW2/HW2_output_images/extracredit_vertical_image.jpg', vertical_image)

def Horizontal(img, alpha):
    height = img.shape[0]
    width = img.shape[1]

    # Compute cosine and sinus of alpha
    cos_alpha = np.cos(np.deg2rad(alpha))
    sin_alpha = np.sin(np.deg2rad(alpha))

    # Create parameterized homography and compute the composite homography
    h_alpha = np.array([[cos_alpha, 0,-sin_alpha],[0, 1, 0],[sin_alpha, 0, cos_alpha]])
    h_norm = np.array([[2/width, 0,-1],[0, 2/height,-1],[0, 0, 1]])
    h_denorm = np.array([[width/2, 0, width/2],[0, height/2, height/2],[0, 0, 1]])
    h_c_alpha = np.matmul(np.matmul(h_denorm, h_alpha), h_norm)

    # Create output image
    horizontal_image = cv2.warpPerspective(img, h_c_alpha, (height, width))
    cv2.imwrite('/mnt/c/Users/Hp/Desktop/MSECE/Computer-
Vision/HW2/HW2_output_images/extracredit_horizontal_image.jpg',
horizontal_image)


# Load your image
img = cv2.imread('/mnt/c/Users/Hp/Desktop/MSECE/Computer-
Vision/HW2/HW2_input_images/extracredit.jpg')
alpha = 45
Rotation(img, alpha)
Vertical(img, alpha)
Horizontal(img, alpha)
```