ECE661: Computer Vision

Homework 2

Praneet Singh
singh671@purdue.edu

September 11, 2020

1 Note :

All the code and utility functions used to perform this assignment have been attached at the end of this document. Necessary comment blocks have been added to explain the working of each code block. The task of considering more points to compute homography has been added at the end as an additional task.

Task 1:

We are given four images as seen in Figures. 1a, 1b, 1c and 1d.
1a, 1b , & 1c are images of the same painting hanging on a wall taken at different angles.
1d is the image of a kitten than needs to be mapped to the paintings.

[1a] 

[1b] 

[1c] 

[1d] 

In order to this, let us first consider the equation of the planar projective transformation of a point $\vec{X}$. This is given by ,

$$\vec{X}^t = H\vec{X} \tag{1}$$

where $\vec{X}^t$ is the transformation of the point $\vec{X}$ using a non-singular 3x3 matrix called the Homography matrix. In the representational space, we can represent the points as follows,

$$\vec{X}^t = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}, \vec{X} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Also, the Homography matrix H can be represented as follows,

$$H = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ v_1 & v_2 & \nu \end{bmatrix}$$

where $\nu = 1$.
Thus, substituting the above representations in (1) we get,

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ v_1 & v_2 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2}$$

From (2), we can get the equations of x' & y' as follows,

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11}x + a_{12}y + t_x \\ a_{21}x + a_{22}y + t_y \\ v_1x + v_2y + 1 \end{bmatrix}$$

This can be further reduced to,

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \frac{a_{11}x + a_{12}y + t_x}{v_1x + v_2y + 1} \\ \frac{a_{21}x + a_{22}y + t_y}{v_1x + v_2y + 1} \end{bmatrix} \tag{3}$$

From (3), we can see that,

$$x' = \frac{a_{11}x + a_{12}y + t_x}{v_1x + v_2y + 1} \tag{4}$$

$$y' = \frac{a_{21}x + a_{22}y + t_x}{v_1x + v_2y + 1} \tag{5}$$

On simplifying (4) and (5), we get,

$$x' = a_{11}x + a_{12}y + t_x - v_1xx' - v_2yx' \tag{6}$$

$$y' = a_{21}x + a_{22}y + t_y - v_1xy' - v_2yy' \tag{7}$$

The equations (6) & (7) can be rewritten as a system of equations as follows,

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -xx' & -yx' \\ 0 & 0 & 0 & x & y & 1 & -xy' & -yy' \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ t_x \\ a_{21} \\ a_{22} \\ t_y \\ v_x \\ v_y \end{bmatrix}$$

But, since we have 4 points, $PQRS$ in 1a, 1b, 1c & 1d, we can rewrite the above equation as follows:

$$\begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \\ x'_4 \\ y'_4 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x'_1 & -y_1x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y'_1 & -y_1y'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x'_2 & -y_2x'_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y'_2 & -y_2y'_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x'_3 & -y_3x'_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y'_3 & -y_3y'_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x'_4 & -y_4x'_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y'_4 & -y_4y'_4 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ t_x \\ a_{21} \\ a_{22} \\ t_y \\ v_x \\ v_y \end{bmatrix} \tag{8}$$

Now, we can write (8) as follows,

$$P' = KH_C \tag{9}$$

where,

$$P' = \begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \\ x'_4 \\ y'_4 \end{bmatrix}, K = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x'_1 & -y_1x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y'_1 & -y_1y'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x'_2 & -y_2x'_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y'_2 & -y_2y'_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x'_3 & -y_3x'_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y'_3 & -y_3y'_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x'_4 & -y_4x'_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y'_4 & -y_4y'_4 \end{bmatrix}, H_C = \begin{bmatrix} a_{11} \\ a_{12} \\ t_x \\ a_{21} \\ a_{22} \\ t_y \\ v_x \\ v_y \end{bmatrix}$$

We know all the parameters in P' & K and as a result we can determine all the parameters required to construct the Homography matrix H by solving $P' = KH_C$

Task 1 Code Implementation Details:

- First, GIMP was used to determine the points P, Q, R & S in images 1a, 1b, 1c. Points P', Q', R' & S' are determined in image 1d. The co-ordinates of these points are as follows:

Fig. 1a		Fig. 1b		Fig. 1c		Fig. 1d	
P	(297,509)	P	(336,692)	P	(107,441)	P'	(0,0)
Q	(237,1607)	Q	(329,2335)	Q	(117,1371)	Q'	(0,1125)
R	(1687,1833)	R	(1886,2003)	R	(1100,1865)	R'	(1920,1125)
S	(1780,354)	S	(1888,752)	S	(1220,300)	S'	(1920,0)

- Next, using function **homography()** we can estimate the Homography matrix parameters that helps map the 1d to 1a or 1b or 1c. This also gives us the Homography matrix H.
- The function **homography()** uses simple numpy functions like np.linalg.solve() or np.linalg.inv & np.dot to estimate the Homography matrix H.

$$H_{da} = \begin{bmatrix} 1.896 & 0.1036 & -615.865 \\ 0.1155 & 1.1058 & -597.1403 \\ 0.0002547 & 0.00000788 & 1 \end{bmatrix}$$

$$H_{db} = \begin{bmatrix} 0.89674 & 0.00382 & -303.95 \\ -0.02525 & 0.6533 & -443.614 \\ -0.0001466 & 0.000001 & 1 \end{bmatrix}$$

$$H_{dc} = \begin{bmatrix} 3.1318 & -0.03367 & -320.256 \\ 0.14665 & 1.1576 & -526.215 \\ 0.000692 & -0.0000895 & 1 \end{bmatrix}$$

- After this, we need to map the region $P'Q'R'S'$ from 1d using the Homography matrix H to the region $PQRS$ in 1a or 1b or 1c
- In order to do this, we can use the function **apply_homography()**. This function creates a temporary blank region in 1a or 1b or 1c and fills this region with the 1d image.

Task 1 Results

1d mapped to 1a

1d mapped to 1b

1d mapped to 1c

Task 1 Cascade Homography

In this task, we need to find the Homography between 1a & 1b. We then find the Homography between 1b & 1c. On multiplying these Homography matrices, we get a resultant Homography matrix which when applied to 1a results in 1c.

Task 1 Cascade Homography Implementation Details

- We first start of by finding the Homography H_{ab}. We also find the Homography H_{bc}.
- We then multiply the obtained Homography matrices to get H_{ac}

$$H_{ab} = \begin{bmatrix} 0.4194 & -0.030485 & 152.305 \\ -0.2064 & 0.6195 & 107.025 \\ -0.000257 & 0.00000393 & 1 \end{bmatrix}$$

$$H_{bc} = \begin{bmatrix} 3.925 & -0.07986 & -18.551 \\ 1.2367 & 1.7033 & -129.22 \\ 0.001266 & -0.000103 & 1 \end{bmatrix}$$

$$H_{ac} = \begin{bmatrix} 1.8018 & -0.1011 & 148.461 \\ 0.09149 & 1.06083 & 30.7895 \\ 0.0002621 & -0.0000759 & 1.00426 \end{bmatrix}$$

- Using the function $apply\_homography_2()$ which is very similar to the previous apply homography function, we get the necessary results

Task 1 Cascade Homography Results

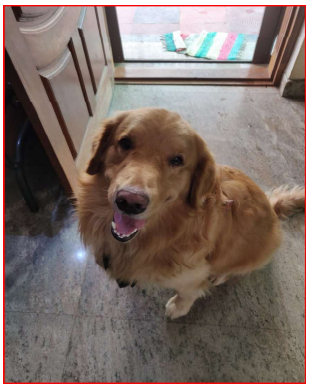
Task 2

Task 2 is very similar to Task 1. I took pictures of a painting on my wall at diffrent angles and decided to map Silver, my friend's dogs photo onto the painting.The new 1a, 1b, 1c & 1d images considered are,

[2a] 

[2b] 

[2c] 

[2d] 

Task 2 Implementation Details

The implementation details are exactly as in Task 1.
The points considered in all the images are as follows:

Fig. 2a	
P	(529,733)
Q	(645,3073)
R	(2037,3425)
S	(2137,585)

Fig. 2b	
P	(1081,413)
Q	(1089,3553)
R	(2117,2957)
S	(2217,805)

Fig. 2c	
P	(585,809)
Q	(429,3081)
R	(2041,3077)
S	(1845,285)

Fig. 2d	
P'	(0,0)
Q'	(0,1152)
R'	(864,1152)
S'	(864,0)

However, the new homography matrices are as follows:

$$H_{da} = \begin{bmatrix} 0.6852 & -0.03396 & -337.5795 \\ 0.040831 & 0.44362 & -346.777 \\ 0.0001485 & -0.0000621 & 1 \end{bmatrix}$$

$$H_{db} = \begin{bmatrix} 0.3841 & -0.0009786 & -414.850 \\ -0.08321 & 0.2411 & 9.6414 \\ -0.0002119 & -0.0000316 & 1 \end{bmatrix}$$

$$H_{dc} = \begin{bmatrix} 1.13574 & 0.07798 & -727.499 \\ 0.3532 & 0.8493 & -893.782 \\ 0.000305 & 0.0001611 & 1 \end{bmatrix}$$

Task 2 Results

2d mapped to 2a

2d mapped to 2b

2d mapped to 2c

Task 2 Cascade Homography Results

The new cascade homography matrices are:

$$H_{ab} = \begin{bmatrix} 0.3699 & 0.02314 & -42.319 \\ -0.4322 & 0.5109 & 765.018 \\ -0.000292 & -0.00000169 & 1 \end{bmatrix}$$

$$H_{bc} = \begin{bmatrix} 7.2881 & 1.1055 & -2069.7957 \\ 5.0258 & 6.2347 & -6804.176 \\ 0.002208 & 0.000698 & 1 \end{bmatrix}$$

$$H_{ac} = \begin{bmatrix} 2.7193 & 0.523784 & -965.576 \\ 1.1073 & 3.2418 & -1816.9795 \\ 0.000142 & 0.0003754 & 1.5958 \end{bmatrix}$$

Code

I've attached my entire jupyter notebook that has all the required code in it with the code explanation.

hw2

September 11, 2020

[ ]:
```
'''
This notebook contains all the code to get the results for HW2 of ECE661 taught␣
 ↪by Dr. Avinash Kak. To the best of my knowledge all the code written here␣
 ↪has been written by me.
'''
#Importing necessary libraries
import numpy as np
import sys
import os
import cv2
import matplotlib.pyplot as plt
import math
%matplotlib inline
```

[ ]:
```
plt.rcParams['figure.figsize'] = [100, 50]

''' Used GIMP to pick the pixel coordinates in paintings and images to be␣
 ↪mapped'''
#Task 1
p1_points = {"Name":"painting1.jpeg","P":(297,509),"Q":(237,1607),"R":
 ↪(1687,1833),"S":(1780,354)}
p2_points = {"Name":"painting2.jpeg","P":(336,692),"Q":(329,2335),"R":
 ↪(1886,2003),"S":(1888,752)}
p3_points = {"Name":"painting3.jpeg","P":(107,441),"Q":(117,1371),"R":
 ↪(1100,1865),"S":(1220,300),}
kittens = {"Name":"kittens.jpeg","P_":(0,0),"Q_":(0,1125),"R_":(1920,1125),"S_":
 ↪(1920,0)}

#Task2
joker1_points = {"Name":"joker1.jpg","P":(529,733),"Q":(645,3073),"R":
 ↪(2037,3425),"S":(2137,585)}
joker2_points = {"Name":"joker2.jpg","P":(1081,413),"Q":(1089,3553),"R":
 ↪(2117,2957),"S":(2217,805)}
joker3_points = {"Name":"joker3.jpg","P":(585,809),"Q":(429,3081),"R":
 ↪(2041,3077),"S":(1845,285)}
silver = {"Name":"silver.jpg","P_":(0,0),"Q_":(0,1152),"R_":(864,1152),"S_":
 ↪(864,0)}
```

```
points =␣
↪[p1_points,p2_points,p3_points,kittens,joker1_points,joker2_points,joker3_points,silver]

''' Plotting Point Coordinates on each picture to ensure correct point␣
↪selection'''
for point in points:
    painting = cv2.imread(point["Name"])
    painting = cv2.cvtColor(painting,cv2.COLOR_BGR2RGB)
    plt_points = list(point.keys())[1:]
    pt_array = []
    for i in plt_points:
        pt_array.append(point[i])
        x, y = point[i]
        painting = cv2.rectangle(painting,(x,y),(x,y),(255,0,0),10)
        if i == "P" or i == "P_":
            painting = cv2.putText(painting,i,(x-20,y-20), cv2.
↪FONT_HERSHEY_SIMPLEX,1,(255,0,0),2)
        elif i == "Q" or i == "Q_":
            painting = cv2.putText(painting,i,(x-20,y+20), cv2.
↪FONT_HERSHEY_SIMPLEX,1,(255,0,0),2)
        elif i == "R" or i == "R_":
            painting = cv2.putText(painting,i,(x+20,y+20), cv2.
↪FONT_HERSHEY_SIMPLEX,1,(255,0,0),2)
        else:
            painting = cv2.putText(painting,i,(x+20,y-20), cv2.
↪FONT_HERSHEY_SIMPLEX,1,(255,0,0),2)
    try:
        pt_array.append(point["P"])
    except:
        pt_array.append(point["P_"])
    painting = cv2.polylines(painting,np.asarray([pt_array]),False,(255,0,0),5)
    if ".jpeg" in point["Name"]:
        cv2.imwrite(point["Name"].replace(".jpeg","")+"_pqrs.jpg",cv2.
↪cvtColor(painting,cv2.COLOR_RGB2BGR))
    else:
        cv2.imwrite(point["Name"].replace(".jpg","")+"_pqrs.jpg",cv2.
↪cvtColor(painting,cv2.COLOR_RGB2BGR))
    plt.imshow(painting)
    plt.show()
```

[ ]:
```
''' Calculating Homography between the image to be mapped and the paintings'''


def midpoint(p1,p2):
    '''
```

```
    Calculates midpoint for complex_homography application
    '''
    return (int(p1[0]+p2[0]/2),int(p1[1]+p2[1]/2))



def homography(p1,p2):
    '''
    Computes the Homography parameters between given set of points and returns␣
 ↪a 3x3 Homography matrix.
    '''
    p1 = np.asarray([p1[i] for i in list(p1.keys())[1:]])
    p2 = np.asarray([p2[i] for i in list(p2.keys())[1:]])
    #Constructing point equation matrix K that can be used to determine␣
 ↪parameters of Homography
    K = np.empty((8,8))
    rows = 0
    for i in zip(p1,p2):
        x, y = list(i[1])
        x_, y_ = list(i[0])
        for row in range(2):
            if row%2 == 0:
                K[rows][0] = x
                K[rows][1] = y
                K[rows][2] = 1
                K[rows][3] = 0
                K[rows][4] = 0
                K[rows][5] = 0
                K[rows][6] = -x * x_
                K[rows][7] = -y * x_
            else:
                K[rows][0] = 0
                K[rows][1] = 0
                K[rows][2] = 0
                K[rows][3] = x
                K[rows][4] = y
                K[rows][5] = 1
                K[rows][6] = -x * y_
                K[rows][7] = -y * y_
            rows+=1
    #Reshaping point equation matrix
    K = K.reshape((8,8))
    #print(C)
    #Reshaping point matrices to allow for the solving of the system of␣
 ↪equations
    p1 = p1.reshape((8,1))
    #Because P = KH_c, we know that => H_c = K^(-1)P
```

```
    K_inv  = np.linalg.inv(K)
    #print(K_inv)
    H_c = np.dot(K_inv,p1)
    #H = np.linalg.solve(K,p1) can also be used to solve this system
    #Appending nu=1 to the parameters and reshaping to return the Homography␣
↪matrix H
    H = np.append(H_c,1)
    return H.reshape((3,3))


def complex_homography(p1,p2,mc = True):
    '''
    Very similar to the homography() above. Only thing is that it helps␣
↪consider more points to compute the Homography. The mc flag controls if we␣
↪consider 8 or 16 points rather than the usual 4
    '''
    p1 = np.asarray([p1[i] for i in list(p1.keys())[1:]])
    p2 = np.asarray([p2[i] for i in list(p2.keys())[1:]])
    #Appending more points which are the midpoints of the lines PQ,QR,RS and SP

    new_p1s = []
    new_p2s = []
    for i in range(len(p1)-1):
        new_p1s.append(p1[i])
        new_p1s.append(midpoint(p1[i],p1[i+1]))
        new_p2s.append(p2[i])
        new_p2s.append(midpoint(p2[i],p2[i+1]))

    new_p1s.append(p1[-1])
    new_p1s.append(midpoint(p1[0],p1[-1]))
    new_p2s.append(p1[-1])
    new_p2s.append(midpoint(p2[0],p2[-1]))

    p1 = np.asarray(new_p1s)
    p2 = np.asarray(new_p2s)

    if mc:
        new_p1s = []
        new_p2s = []
        for i in range(len(p1)-1):
            new_p1s.append(p1[i])
            new_p1s.append(midpoint(p1[i],p1[i+1]))
            new_p2s.append(p2[i])
            new_p2s.append(midpoint(p2[i],p2[i+1]))

        new_p1s.append(p1[-1])
        new_p1s.append(midpoint(p1[0],p1[-1]))
```

```
        new_p2s.append(p1[-1])
        new_p2s.append(midpoint(p2[0],p2[-1]))
        p1 = np.asarray(new_p1s)
        p2 = np.asarray(new_p2s)

    ''' Constructing point equation matrix but this time it is not square '''
    K = np.empty((p1.shape[0]*p1.shape[1],8))
    rows = 0
    for i in zip(p1,p2):
        x, y = list(i[1])
        x_, y_ = list(i[0])
        for row in range(2):
            if row%2 == 0:
                K[rows][0] = x
                K[rows][1] = y
                K[rows][2] = 1
                K[rows][3] = 0
                K[rows][4] = 0
                K[rows][5] = 0
                K[rows][6] = -x * x_
                K[rows][7] = -y * x_
            else:
                K[rows][0] = 0
                K[rows][1] = 0
                K[rows][2] = 0
                K[rows][3] = x
                K[rows][4] = y
                K[rows][5] = 1
                K[rows][6] = -x * y_
                K[rows][7] = -y * y_
            rows+=1
    K = K.reshape((p1.shape[0]*p1.shape[1],8))
    #print(C.shape)
    p1 = p1.reshape((p1.shape[0]*p1.shape[1],1))
    #print(p1.shape)
    K_inv = np.linalg.pinv(K)
    #print(K_inv)
    H_c = np.dot(K_inv,p1)
    #H = np.linalg.lstsq(K,p1) can also be used to solve this system as it is␣
↪not square but i sued pseudo inverse
    #Appending nu=1 to the parameters and reshaping to return the Homography␣
↪matrix H
    H = np.append(H_c,1)
    return H.reshape((3,3))

'''Calculating the homography matrix in all the three cases'''
#Task1
```

```
H1 = homography(kittens,p1_points)
print(H1)
H2 = homography(kittens,p2_points)
print(H2)
H3 = homography(kittens,p3_points)
print(H3)

#Task2
H4 = homography(silver,joker1_points)
print(H4)
H5 = homography(silver,joker2_points)
print(H5)
H6 = homography(silver,joker3_points)
print(H6)
```

[ ]:
```
'''Applying Homographies to map images to the paintings '''


def apply_homography(painting,frame_to_map,painting_frame,H):
    ''' Function helps map images to paintings. It takes in the range painting,␣
 ↪domain painting, the points
    in the range to which the domain painting needs to be mapped to and the␣
 ↪Homography matrix H.
    It uses cv2.fillPoly() to keep a track of the region in the range in which␣
 ↪the domain needs to be mapped.
    I've used math.floor to ensure mapped point stays with constraints.
    '''
    width = painting.shape[1]
    height = painting.shape[0]
    empty_painting = np.zeros(painting.shape, dtype='uint8')
    #Keep track of region
    cv2.fillPoly(empty_painting,[painting_frame],(255,255,255))
    for i in range(0,(width-1)):
        for j in range(0,(height-1)):
            #Only apply to selected region in range image
            if any(empty_painting[j][i]) > 0:
                    point_to_be_mapped = np.append(np.array([i,j]),1)
                    mapped_point = np.matmul(H, point_to_be_mapped)
                    #Convert (x',y',1) to (x',y')
                    mapped_point = np.asarray((math.floor(mapped_point[0]/
 ↪mapped_point[2]),math.floor(mapped_point[1]/mapped_point[2])))
                    mapped_point = mapped_point.astype(int)
                    if(mapped_point[0] < frame_to_map.shape[1] and␣
 ↪mapped_point[1] < frame_to_map.shape[0]):
```

```
                        painting[j][i] =␣
↪frame_to_map[mapped_point[1]][mapped_point[0]]
    return painting


def apply_homography_2(painting1,painting2,H):
    ''' Very similar to the apply_homography(). Function helps apply cascade␣
↪homographies to entire paintings. It takes in both the domain and range␣
↪paintings
    and the Homography matrix H that maps the domain painting entirely to the␣
↪range painting.
    '''
    width = painting2.shape[1]
    height = painting2.shape[0]
    empty_painting = np.zeros(painting2.shape, dtype='uint8')
    for i in range(0,(width-1)):
        for j in range(0,(height-1)):
            #Applying to entire image
                    point_to_be_mapped = np.append(np.array([i,j]),1)
                    mapped_point = np.matmul(H, point_to_be_mapped)
                    mapped_point = np.asarray((math.floor(mapped_point[0]/
↪mapped_point[2]),math.floor(mapped_point[1]/mapped_point[2])))
                    mapped_point = mapped_point.astype(int)
                    if(mapped_point[0] < painting1.shape[1] and mapped_point[1]␣
↪< painting1.shape[0]):
                        empty_painting[j][i] =␣
↪painting1[mapped_point[1]][mapped_point[0]]
    return empty_painting
```

[ ]:
```
'''
Now that we have defined all our functions, let's start getting the required␣
↪results.
'''

#Read images
#Task 1
kits = cv2.imread(kittens["Name"])
p1 = cv2.imread(p1_points["Name"])
p2 = cv2.imread(p2_points["Name"])
p3 = cv2.imread(p3_points["Name"])
#Task2
slv = cv2.imread(silver["Name"])
j1 = cv2.imread(joker1_points["Name"])
j2 = cv2.imread(joker2_points["Name"])
j3 = cv2.imread(joker3_points["Name"])
```

```
#Convert images from BGR2RGB
#Task1
kits = cv2.cvtColor(kits,cv2.COLOR_BGR2RGB)
p1 = cv2.cvtColor(p1,cv2.COLOR_BGR2RGB)
p2 = cv2.cvtColor(p2,cv2.COLOR_BGR2RGB)
p3 = cv2.cvtColor(p3,cv2.COLOR_BGR2RGB)
#Task2
slv = cv2.cvtColor(slv,cv2.COLOR_BGR2RGB)
j1 = cv2.cvtColor(j1,cv2.COLOR_BGR2RGB)
j2 = cv2.cvtColor(j2,cv2.COLOR_BGR2RGB)
j3 = cv2.cvtColor(j3,cv2.COLOR_BGR2RGB)

#Get PQRS point regions from all paintings
#Task1
p1_pts = [p1_points[i] for i in list(p1_points.keys())[1:]]
p2_pts = [p2_points[i] for i in list(p2_points.keys())[1:]]
p3_pts = [p3_points[i] for i in list(p3_points.keys())[1:]]
#Task2
j1_pts = [joker1_points[i] for i in list(joker1_points.keys())[1:]]
j2_pts = [joker2_points[i] for i in list(joker2_points.keys())[1:]]
j3_pts = [joker3_points[i] for i in list(joker3_points.keys())[1:]]

#Convert to numpy arrays
#Task1
p1_pts = np.asarray(p1_pts)
p2_pts = np.asarray(p2_pts)
p3_pts = np.asarray(p3_pts)
#Task2
j1_pts = np.asarray(j1_pts)
j2_pts = np.asarray(j2_pts)
j3_pts = np.asarray(j3_pts)

#Applying homographies and getting results
#Task1
res = apply_homography(p1,kits,p1_pts,H1)
cv2.imwrite("homography1.jpg",cv2.cvtColor(res,cv2.COLOR_RGB2BGR))
plt.imshow(res)
plt.show()
res = apply_homography(p2,kits,p2_pts,H2)
cv2.imwrite("homography2.jpg",cv2.cvtColor(res,cv2.COLOR_RGB2BGR))
plt.imshow(res)
plt.show()
res = apply_homography(p3,kits,p3_pts,H3)
cv2.imwrite("homography3.jpg",cv2.cvtColor(res,cv2.COLOR_RGB2BGR))
plt.imshow(res)
plt.show()
```

```
#Task2
res = apply_homography(j1,slv,j1_pts,H4)
cv2.imwrite("homography4.jpg",cv2.cvtColor(res,cv2.COLOR_RGB2BGR))
plt.imshow(res)
plt.show()
res = apply_homography(j2,slv,j2_pts,H5)
cv2.imwrite("homography5.jpg",cv2.cvtColor(res,cv2.COLOR_RGB2BGR))
plt.imshow(res)
plt.show()
res = apply_homography(j3,slv,j3_pts,H6)
cv2.imwrite("homography6.jpg",cv2.cvtColor(res,cv2.COLOR_RGB2BGR))
plt.imshow(res)
plt.show()
```

[ ]:
```
''' Cascade Homography: Finding Homographies between 1a and 1b, 1b & 1c and␣
 ↪mulitplying them. Applying the resultant to painting 1 to get painting 3 '''

#Compute homography from 1a to 1b, then 1b to 1c and finally multiply to get␣
 ↪cascade homography from 1a  to 1c.
#Task1
H_ab = homography(p1_points,p2_points)
H_bc = homography(p2_points,p3_points)
H_ab_bc = np.matmul(H_ab, H_bc)
p1 = cv2.imread(p1_points["Name"])
p3 = cv2.imread(p3_points["Name"])
p1 = cv2.cvtColor(p1,cv2.COLOR_BGR2RGB)
p3 = cv2.cvtColor(p3,cv2.COLOR_BGR2RGB)
res = apply_homography_2(p1,p3,H_ab_bc)
plt.imshow(res)
plt.show()
cv2.imwrite("cascade1.jpg",cv2.cvtColor(res,cv2.COLOR_RGB2BGR))

#Task2
H_ab = homography(joker1_points,joker2_points)
H_bc = homography(joker2_points,joker3_points)
H_ab_bc = np.matmul(H_ab, H_bc)
j1 = cv2.imread(joker1_points["Name"])
j3 = cv2.imread(joker3_points["Name"])
j1 = cv2.cvtColor(j1,cv2.COLOR_BGR2RGB)
j3 = cv2.cvtColor(j3,cv2.COLOR_BGR2RGB)
res = apply_homography_2(j1,j3,H_ab_bc)
plt.imshow(res)
plt.show()
cv2.imwrite("cascade2.jpg",cv2.cvtColor(res,cv2.COLOR_RGB2BGR))
```

[ ]:
```
'''Complex Homography Results'''

#Complex Homographies
H1 = complex_homography(kittens,p1_points)
print(H1)
H2 = complex_homography(kittens,p2_points)
print(H2)
H3 = complex_homography(kittens,p3_points)
print(H3)


#Task2
H4 = complex_homography(silver,joker1_points)
print(H4)
H5 = complex_homography(silver,joker2_points)
print(H5)
H6 = complex_homography(silver,joker3_points)
print(H6)
```

[ ]:
```
#Read images
#Task 1
kits = cv2.imread(kittens["Name"])
p1 = cv2.imread(p1_points["Name"])
p2 = cv2.imread(p2_points["Name"])
p3 = cv2.imread(p3_points["Name"])
#Task2
slv = cv2.imread(silver["Name"])
j1 = cv2.imread(joker1_points["Name"])
j2 = cv2.imread(joker2_points["Name"])
j3 = cv2.imread(joker3_points["Name"])

#Convert images from BGR2RGB
#Task1
kits = cv2.cvtColor(kits,cv2.COLOR_BGR2RGB)
p1 = cv2.cvtColor(p1,cv2.COLOR_BGR2RGB)
p2 = cv2.cvtColor(p2,cv2.COLOR_BGR2RGB)
p3 = cv2.cvtColor(p3,cv2.COLOR_BGR2RGB)
#Task2
slv = cv2.cvtColor(slv,cv2.COLOR_BGR2RGB)
j1 = cv2.cvtColor(j1,cv2.COLOR_BGR2RGB)
j2 = cv2.cvtColor(j2,cv2.COLOR_BGR2RGB)
j3 = cv2.cvtColor(j3,cv2.COLOR_BGR2RGB)

#Get PQRS point regions from all paintings
#Task1
p1_pts = [p1_points[i] for i in list(p1_points.keys())[1:]]
p2_pts = [p2_points[i] for i in list(p2_points.keys())[1:]]
```

```
p3_pts = [p3_points[i] for i in list(p3_points.keys())[1:]]
#Task2
j1_pts = [joker1_points[i] for i in list(joker1_points.keys())[1:]]
j2_pts = [joker2_points[i] for i in list(joker2_points.keys())[1:]]
j3_pts = [joker3_points[i] for i in list(joker3_points.keys())[1:]]

#Convert to numpy arrays
#Task1
p1_pts = np.asarray(p1_pts)
p2_pts = np.asarray(p2_pts)
p3_pts = np.asarray(p3_pts)
#Task2
j1_pts = np.asarray(j1_pts)
j2_pts = np.asarray(j2_pts)
j3_pts = np.asarray(j3_pts)

#Applying homographies and getting results
#Task1
res = apply_homography(p1,kits,p1_pts,H1)
cv2.imwrite("complex_homography1.jpg",cv2.cvtColor(res,cv2.COLOR_RGB2BGR))
plt.imshow(res)
plt.show()
res = apply_homography(p2,kits,p2_pts,H2)
cv2.imwrite("complex_homography2.jpg",cv2.cvtColor(res,cv2.COLOR_RGB2BGR))
plt.imshow(res)
plt.show()
res = apply_homography(p3,kits,p3_pts,H3)
cv2.imwrite("complex_homography3.jpg",cv2.cvtColor(res,cv2.COLOR_RGB2BGR))
plt.imshow(res)
plt.show()

#Task2
res = apply_homography(j1,slv,j1_pts,H4)
cv2.imwrite("complex_homography4.jpg",cv2.cvtColor(res,cv2.COLOR_RGB2BGR))
plt.imshow(res)
plt.show()
res = apply_homography(j2,slv,j2_pts,H5)
cv2.imwrite("complex_homography5.jpg",cv2.cvtColor(res,cv2.COLOR_RGB2BGR))
plt.imshow(res)
plt.show()
res = apply_homography(j3,slv,j3_pts,H6)
cv2.imwrite("complex_homography6.jpg",cv2.cvtColor(res,cv2.COLOR_RGB2BGR))
plt.imshow(res)
plt.show()
```

Additional Task

I tried implementing the estimation of the Homography parameters using more number of points. You can find the implementation in the function **complex_homography()** where I use either 8 or 16 points to find the Homography matrix.

In this case, because we have more points than degrees of freedoms, we have a rectangular K matrix in (9). As a result, I used np.linlag.pinv() or np.linalg.lstsq() to solve this system.

The result however doesn't seem viable.

Result of Complex Homography