ECE661: Computer Vision

Purdue University - West Lafayette

Homework 2

Henry Namocatcat Jr
(hnamocat@purdue.edu)

September 10, 2020

Task 1.a and 2.a.

In this task, I initially picked 4pts PQRS, they can be found in my code as commented out. But I noticed from the HW2 additional instructions, that I can try 8pts ROI, and the latter is what I have been using for all of the tasks.

There other way to estimate the homography, one approach is inhomogeneous Linear Least Square, denoted by

$$Ax = b \tag{1}$$

for which we solve matrix A by doing the inverse. Equation 1 wont work when our system is homogeneous Linear Least Square (Ax = 0). Even though my system in inhomogenoeous Linear Least Square, I preferred to use SVD as out of curiosity. Singular Value Decomposition formula:

$$A = U\Sigma V \sum_{i=1}^{9} \theta_1 u_1 v_1^{\intercal} \tag{2}$$

for which I took the lowest value of V for my Homography estimate calculation. Here's a snippet of the code

```
U,S,V = np.linalg.svd(MatrixA)
H = V[8].reshape((3,3))
H = H / H[2,2]
```

In estimating the Homography, there are 8 degrees of freedom, and any addition ROI pts should not be of a problem. Hence, using those 8 correspondence, *PUQVRWST*, is describe in this manner

$$
\begin{bmatrix}
x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x'_1 & -y_1x'_1 \\
0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y'_1 & -y_1y'_1 \\
x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x'_2 & -y_2x'_2 \\
0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y'_2 & -y_2y'_2 \\
x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x'_3 & -y_3x'_3 \\
0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y'_3 & -y_3y'_3 \\
x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x'_4 & -y_4x'_4 \\
0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y'_4 & -y_4y'_4 \\
x_5 & y_5 & 1 & 0 & 0 & 0 & -x_5x'_5 & -y_5x'_5 \\
0 & 0 & 0 & x_5 & y_5 & 1 & -x_5y'_5 & -y_5y'_5 \\
x_6 & y_6 & 1 & 0 & 0 & 0 & -x_6x'_6 & -y_6x'_6 \\
0 & 0 & 0 & x_6 & y_6 & 1 & -x_6y'_6 & -y_6y'_6 \\
x_7 & y_7 & 1 & 0 & 0 & 0 & -x_7x'_3 & -y_7x'_7 \\
0 & 0 & 0 & x_7 & y_7 & 1 & -x_7y'_7 & -y_7y'_7 \\
x_8 & y_8 & 1 & 0 & 0 & 0 & -x_8x'_8 & -y_8x'_8 \\
0 & 0 & 0 & x_8 & y_8 & 1 & -x_8y'_8 & -y_8y'_8
\end{bmatrix}
\begin{bmatrix} a \\ b \\ d \\ e \\ f \\ g \\ h \\ i \end{bmatrix}
=
\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \\ x_{16} \end{bmatrix}
$$

The following images, borrowed taken from Homework 2 - Additional Notes section, describing the addition regions of interests, for which I based my criteria.

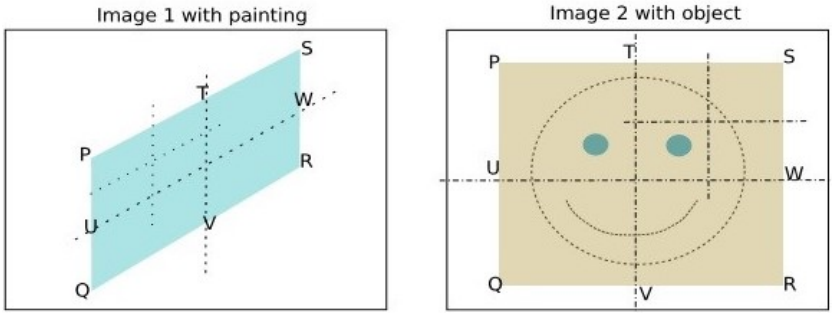


Figure 1: 8-pt Region of Interest

For extracting the color, instead of using built-in libraries like warp perpective, I used a Bilinear Interpolation, weighted average. The formula is *I(x,y) = A3*I(i,j) + A4*I(i+1,j) + A2*I(i+1,j+1) + A1*I(i,j+1)*, then average them by dividing 4 ROI from the source image. From that calculation, I did not add those extra 4-points U-V-W-T, since my intent of using those 8pts ROI were mainly for estimating Homographies.

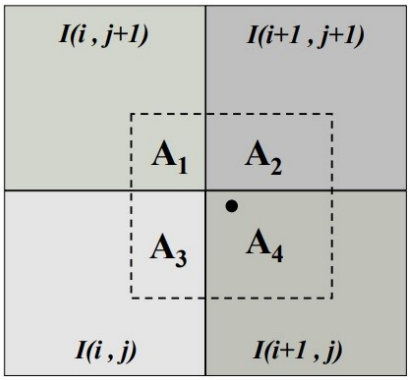


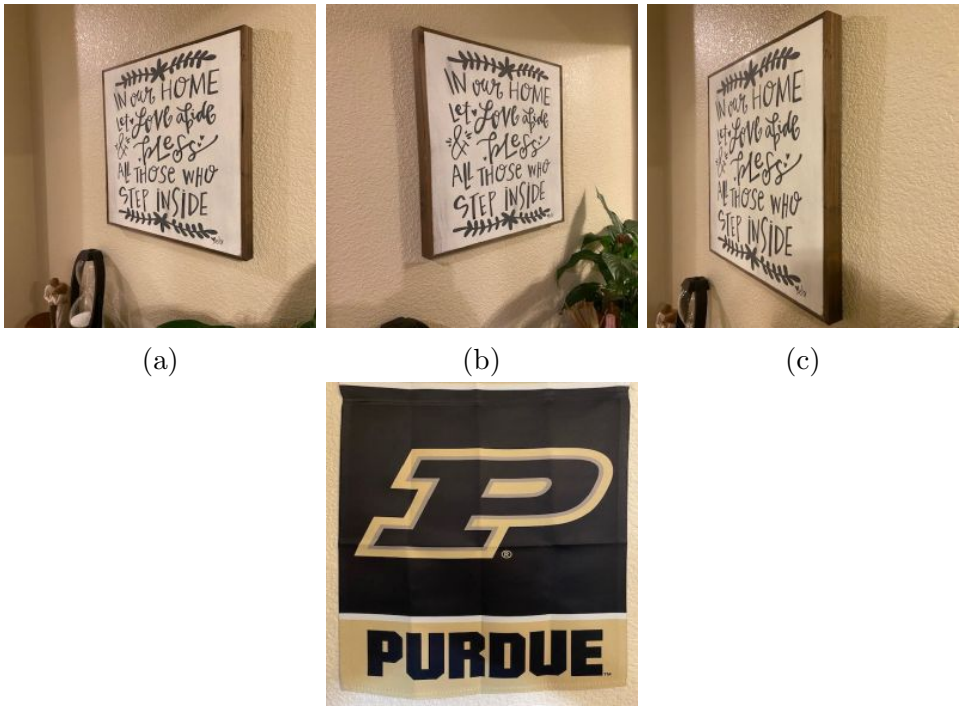
Figure 2: Extracting color from neighboring cell using Bilinear Interpolation

The process starts by blacking out the target painting with an RGB value of 255. Then iterate each pixel point (x,y), except for the area where we want to put the kitten's image. The ROI area is already pre-determined within the array of points.

(a) Painting1 (b) Painting2 (c) Painting3

Figure 3: Kitten image projected on several wall frames

For the the Task2, i.e. using our own images, I used the same program that was used to generate Figure 3. Inside the Python program, I made 6 loops, accounting 6 different wall frames, which target specific source for every 3 occurrences. Here are the generated pictures.



(a) (b) (c)

(d)

Figure 4: Personally selected images for Task 2a

(a) (b) (c)

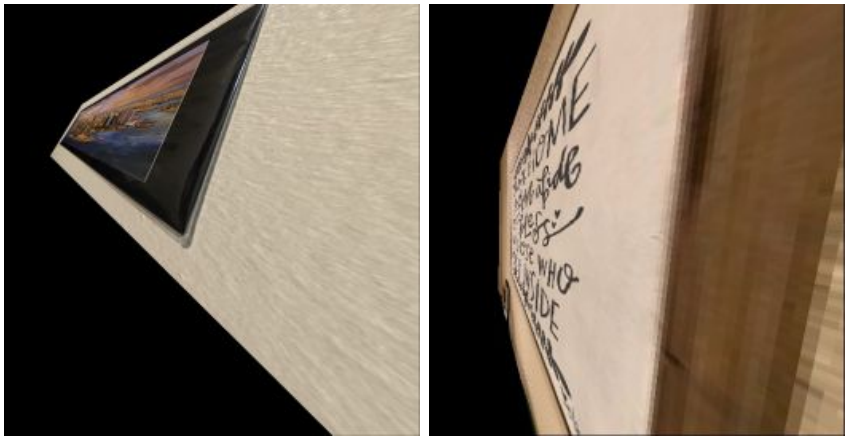
Figure 5: PurdueU projected on several HomeDecor Paintings

Task 1.b and 2.b

In this task, it is quite similar to step 1.a, but this time I used Painting 1 (step. 1b) and Painting 4(step 2b: HomeDecor) as my source to project the image on Painting 3 (for step: 1b) and Painting 6 (for step: 2b), based of the estimated product of two Homographies, namely Painting 1 and 2, then

Painting 2 and 3. Similarly, for step: 2b, I also calculated a separate two Homographies for the HomeDecor portion, namely Painting 4 and 5, then Painting 5 and 6.

The approach of calculating those Homographies were based of SVD. For the color calculation, I also used Bilinear Interpolation.



(a) (b)

Figure 6: Two separate images genrated by the same Python code

Here is my source code for Task 1a and 2a:

```
########################################
#Author: Henry Namocatcat
#ECE661: Computer Vision
#Purdue University - West Lafayette
#email: hnamocat@purdue.edu
#
#Homework 2
#Due: Septermber 10, 2020
#
########################################
import numpy as np
import cv2
import math

kitten=cv2.imread('kittens.jpeg')
purdueU=cv2.imread('PurdueU.jpeg')
#4pts PRQS
#Kitten = np.array([[0, 0], [0, 1918], [1124, 0],[1124, 1918]])

#4pts PRQS
#Kitten = np.array([[0, 0], [0, 1918], [1124, 0],[1124, 1918]])
#Frame1 = np.array([[504,291],[350,1780], [1609,233],[1835,1687]],np.int32)
#Frame2 = np.array([[683,332],[724,1892],[2338,323],[2010,1890]],np.int32)
#Frame3 = np.array([[435,102],[294,1223],[1372,106],[1872,1100],],np.int32)

#8pts PUQVRWST
Kitten = np.array([[0,
    0],[0,991],[0,1918],[505,0],[1124,0],[1124,1023],[1124,1918],[519,1918]])
Kitten_Num_Pts = len(Kitten)

Purdue = np.array([[0,
    0],[0,247],[0,480],[289,0],[639,0],[639,240],[639,480],[305,480]])
Purdue_Num_Pts = len(Purdue)

#8pts PUQVRWST
Frame_Arr =
    [[[500,289],[441,976],[344,1789],[1040,260],[1609,233],[1711,896],[1836,1691],[1055,1746]],
     \
            [[683,332],[725,1247],[724,1892],[1456,332],[2338,323],[2133,1260],[2010,1890],[1341,1883]],
             \
            [[417,80],[379,604],[269,1240],[843,106],[1372,106],[1566,503],[1890,1111],[1064,1165]],
             \
            [[134,153],[101,240],[50,362],[261,157],[430,160],[457,241],[497,365],[258,363]],
              \
            [[54,151],[95,277],[125,359],[254,158],[494,164],[451,292],[426,365],[261,361]],
             \
            [[145,95],[99,163],[27,272],[296,96],[482,97],[527,153],[621,271],[318,270]],
                np.int32]

#P-R-S-Q Sequence. Formatting inner frame boundary
InFrame_Format_Arr1 = np.array([[289,500],[233,1609],[1691,1836],[1789,344]],np.int32) #
    Painting1
InFrame_Format_Arr2 = np.array([[332,683],[323,2338],[1890,2010],[1892,724]],np.int32) #
    Painting2
InFrame_Format_Arr3 = np.array([[80,417],[106,1372],[1111,1890],[1240,269]],np.int32)  #
    Painting3
InFrame_Format_Arr4 = np.array([[153,134],[160,430],[365,497],[362,50]],np.int32) #
    HomeFrame1
InFrame_Format_Arr5 = np.array([[151,54],[164,494],[365,426],[359,125]],np.int32) #
    HomeFrame2
InFrame_Format_Arr6 = np.array([[95,145],[97,482],[271,621],[272,27]],np.int32)  #
    HomeFrame3
InFrame_Format_Arr = np.concatenate((InFrame_Format_Arr1, InFrame_Format_Arr2,
    InFrame_Format_Arr3, \
                                     InFrame_Format_Arr4, InFrame_Format_Arr5,
                                         InFrame_Format_Arr6), axis=0)

MatrixA = np.zeros((2*Purdue_Num_Pts,9))
for k in range(1, 7):
    print("+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+")
    print("Processing painting " + np.str(k))
    WallFrame=cv2.imread('painting' + np.str(k) + '.jpeg')
    inframe_blackout= np.zeros((WallFrame.shape[0],WallFrame.shape[1],3),dtype='uint8')
    Pix = InFrame_Format_Arr[(k-1)*4:(k-1)*4+4]
    Pix = Pix.reshape((-1,1,2))
    cv2.fillPoly(inframe_blackout,[Pix],(255,255,255))

    for i in range(Purdue_Num_Pts):
        if k < 4:
            X_Src_Pts = Kitten[i][0]
            Y_Src_Pts = Kitten[i][1]
```

```
            else:
                X_Src_Pts = Purdue[i][0]
                Y_Src_Pts = Purdue[i][1]
            X_Frame_Pts = Frame_Arr[k-1][i][0]
            Y_Frame_Pts = Frame_Arr[k-1][i][1]
            #Loop over each point(x,y). 4 corners
            MatrixA[2*i] = [-X_Frame_Pts, -Y_Frame_Pts, -1, 0, 0, 0, X_Frame_Pts*X_Src_Pts,
                 Y_Frame_Pts*X_Src_Pts, X_Src_Pts]
            MatrixA[2*i+1] = [0, 0, 0, -X_Frame_Pts, -Y_Frame_Pts, -1, X_Frame_Pts*Y_Src_Pts
                 , Y_Frame_Pts*Y_Src_Pts, Y_Src_Pts]

        #Take the lowest in V
        U,S,V = np.linalg.svd(MatrixA)
        H = V[8].reshape((3,3))
        H = H / H[2,2]
        print("Homography Estimate Created")

        #Looping over the image
        pt_holder = np.matrix('0 0 1',dtype=float)
        for row in range(0,WallFrame.shape[0]-1):
            for col in range(0,WallFrame.shape[1]-1):
                if inframe_blackout[row,col,1] > 0:
                    pt_holder[0,0] = row
                    pt_holder[0,1] = col
                    pt_transpose = np.transpose(H*np.transpose(pt_holder))
                    pt_transpose = pt_transpose/pt_transpose[0,2]

                    #Bilinear Interpolation Formula to extract color
                    #Formula: I(x,y) = A3*I(i,j) + A4*I(i+1,j) + A2*I(i+1,j+1) + A1*I(i,j+1)
                         , then average
                    if k < 4:
                        if pt_transpose[0, 0] > 0 and pt_transpose[0, 1] > 0 and
                             pt_transpose[0, 0] < kitten.shape[0] - 1 and pt_transpose[0, 1]
                             < kitten.shape[1] - 1:
                            kit_p = kitten[math.floor(pt_transpose[0,0]), math.floor(
                                 pt_transpose[0,1])]
                            kit_q = kitten[math.floor(pt_transpose[0,0]), math.ceil(
                                 pt_transpose[0,1])]
                            kit_r = kitten[math.ceil(pt_transpose[0,0]), math.floor(
                                 pt_transpose[0,1])]
                            kit_s = kitten[math.ceil(pt_transpose[0,0]), math.ceil(
                                 pt_transpose[0,1])]
                            kit_wp = 1/np.linalg.norm(np.array([pt_transpose[0,0],
                                 pt_transpose[0,1]]))
                            kit_wq = 1/np.linalg.norm(np.array([pt_transpose[0,0],1-
                                 pt_transpose[0,1]]))
                            kit_wr = 1/np.linalg.norm(np.array([1-pt_transpose[0,0],
                                 pt_transpose[0,1]]))
                            kit_ws = 1/np.linalg.norm(np.array([1-pt_transpose[0,0],1-
                                 pt_transpose[0,1]]))
                            WallFrame[row,col] = (kit_p*kit_wp+kit_q*kit_wq+kit_r*kit_wr+
                                 kit_s*kit_ws)/(kit_wp+kit_wq+kit_wr+kit_ws)
                    else:
                        if pt_transpose[0, 0] > 0 and pt_transpose[0, 1] > 0 and
                             pt_transpose[0, 0] < purdueU.shape[0] - 1 and pt_transpose[0, 1]
                              < purdueU.shape[1] - 1:
                            PU_p = purdueU[math.floor(pt_transpose[0,0]), math.floor(
                                 pt_transpose[0,1])]
                            PU_q = purdueU[math.floor(pt_transpose[0,0]), math.ceil(
                                 pt_transpose[0,1])]
                            PU_r = purdueU[math.ceil(pt_transpose[0,0]), math.floor(
                                 pt_transpose[0,1])]
                            PU_s = purdueU[math.ceil(pt_transpose[0,0]), math.ceil(
                                 pt_transpose[0,1])]
                            PU_wp = 1/np.linalg.norm(np.array([pt_transpose[0,0],
                                 pt_transpose[0,1]]))
                            PU_wq = 1/np.linalg.norm(np.array([pt_transpose[0,0],1-
                                 pt_transpose[0,1]]))
                            PU_wr = 1/np.linalg.norm(np.array([1-pt_transpose[0,0],
                                 pt_transpose[0,1]]))
                            PU_ws = 1/np.linalg.norm(np.array([1-pt_transpose[0,0],1-
                                 pt_transpose[0,1]]))
                            WallFrame[row,col] = (PU_p*PU_wp+PU_q*PU_wq+PU_r*PU_wr+PU_s*
                                 PU_ws)/(PU_wp+PU_wq+PU_wr+PU_ws)
            print("Estimated processed pixel: {0:.0f}% \r".format((row/(WallFrame.shape
                 [0]-1)) * 100), end='\r')
        if k < 4:
            print("Writing Painting" + np.str(k) + "_with_kittens.jpg")
            cv2.imwrite('Painting' + np.str(k) + '_with_kittens.jpg',WallFrame)
        else:
            print("Writing Home Painting" + np.str(k) + "_with_Purdue.jpg")
            cv2.imwrite('HomePainting' + np.str(k) + '_with_Purdue.jpg',WallFrame)
    cv2.destroyAllWindows()
```

Here is my source code for Task 1b and 2b:

```
########################################
#Author: Henry Namocatcat
#ECE661: Computer Vision
#Purdue University - West Lafayette
#email: hnamocat@purdue.edu
#
#Homework 2
#Due: Septermber 10, 2020
#
########################################
import numpy as np
import cv2
import math

Painting1=cv2.imread('painting1.jpeg')
Painting2=cv2.imread('painting2.jpeg')
Painting3=cv2.imread('painting3.jpeg')
Painting4=cv2.imread('painting4.jpeg')
Painting5=cv2.imread('painting5.jpeg')
Painting6=cv2.imread('painting6.jpeg')

#8pts PUQVRWST
Painting2 = np.array
    ([[683,332],[725,1247],[724,1892],[1456,332],[2338,323],[2133,1260],[2010,1890],[1341,1883]])

Painting2_Num_Pts = len(Painting2)

Painting5 = np.array
    ([[54,151],[95,277],[125,359],[254,158],[494,164],[451,292],[426,365],[261,361]])
Painting5_Num_Pts = len(Painting5)

#8pts PUQVRWST ; These Painting1 and 3
Frame_Arr =
    [[[500,289],[441,976],[344,1789],[1040,260],[1609,233],[1711,896],[1836,1691],[1055,1746]],
     \
            [[417,80],[379,604],[269,1240],[843,106],[1372,106],[1566,503],[1890,1111],[1064,1165]],
             \
            [[134,153],[101,240],[50,362],[261,157],[430,160],[457,241],[497,365],[258,363]],
             \
            [[145,95],[99,163],[27,272],[296,96],[482,97],[527,153],[621,271],[318,270]],
                np.int32]

#P-R-S-Q Sequence. Formatting frame boundary
Painting3_Boundary = np.array([[0, 0],[Painting3.shape[1]-1,0], [Painting3.shape[1]-1,
    Painting3.shape[0]-1],[0,Painting3.shape[0]-1]],np.int32)  #Painting3
Painting6_Boundary = np.array([[0, 0],[Painting6.shape[1]-1,0], [Painting6.shape[1]-1,
    Painting6.shape[0]-1],[0,Painting6.shape[0]-1]],np.int32)  #Painting6
Painting_Boundary_Arr = np.concatenate((Painting3_Boundary, Painting6_Boundary), axis=0)

MatrixA = np.zeros((2*Painting2_Num_Pts,9))
for k in range(1, 5):
    print("Processing painting " + np.str(k))
    for i in range(Painting2_Num_Pts):
        if k < 3:
            X_Painting_Pts = Painting2[i][0]
            Y_Painting_Pts = Painting2[i][1]
        else:
            X_Painting_Pts = Painting5[i][0]
            Y_Painting_Pts = Painting5[i][1]
        X_Frame_Pts = Frame_Arr[k-1][i][0]
        Y_Frame_Pts = Frame_Arr[k-1][i][1]
        #Loop over each point(x,y). 4 corners
        MatrixA[2*i] = [-X_Frame_Pts, -Y_Frame_Pts, -1, 0, 0, 0, X_Frame_Pts*
            X_Painting_Pts, Y_Frame_Pts*X_Painting_Pts, X_Painting_Pts]
        MatrixA[2*i+1] = [0, 0, 0, -X_Frame_Pts, -Y_Frame_Pts, -1, X_Frame_Pts*
            Y_Painting_Pts, Y_Frame_Pts*Y_Painting_Pts, Y_Painting_Pts]

    #Take the lowest in V from Singular Value Decomposition
    U,S,V = np.linalg.svd(MatrixA)
    H = V[8].reshape((3,3))
    H = H / H[2,2]
    print("Homography " + np.str(k) + " Estimate Created")

    if ((k == 1) or (k == 3)):
        Homography_1 = H
    if ((k == 2) or (k == 4)):
        Homography_Dot = Homography_1.dot(H)
        print('Two Homographies Multiplied')

        if (k == 2):
            WallFrame=cv2.imread('painting' + np.str(k+1) + '.jpeg')
```

```
    Pix = Painting_Boundary_Arr[0:4]
else:
    WallFrame=cv2.imread('painting' + np.str(k+2) + '.jpeg')
    Pix = Painting_Boundary_Arr[4:8]

painting_blackout= np.zeros((WallFrame.shape[0],WallFrame.shape[1],3),dtype='
    uint8')
Pix = Pix.reshape((-1,1,2))
cv2.fillPoly(painting_blackout,[Pix],(255,255,255))
print('Formatting_Boundary')

Painting1On3 = np.zeros(Painting3.shape, dtype='uint8')
Painting4On6 = np.zeros(Painting6.shape, dtype='uint8')
pt_holder = np.matrix('0_0_1',dtype=float)
for row in range(0,painting_blackout.shape[0]-1):
    for col in range(0,painting_blackout.shape[1]-1):
        if painting_blackout[row,col,1] > 0:
            pt_holder[0,0] = row
            pt_holder[0,1] = col
            pt_transpose = np.transpose(Homography_Dot*np.transpose(pt_holder))
            pt_transpose = pt_transpose/pt_transpose[0,2]

            #Bilinear Interpolation Formula to extract color
            #Formula: I(x,y) = A3*I(i,j) + A4*I(i+1,j) + A2*I(i+1,j+1) + A1*I(i,
                j+1)
            if k < 3:
                if pt_transpose[0, 0] > 0 and pt_transpose[0, 1] > 0 and
                    pt_transpose[0, 0] < Painting1.shape[0] - 1 and pt_transpose
                    [0, 1] < Painting1.shape[1] - 1:
                    P1_p = Painting1[math.floor(pt_transpose[0,0]), math.floor(
                        pt_transpose[0,1])]
                    P1_q = Painting1[math.floor(pt_transpose[0,0]), math.ceil(
                        pt_transpose[0,1])]
                    P1_r = Painting1[math.ceil(pt_transpose[0,0]), math.floor(
                        pt_transpose[0,1])]
                    P1_s = Painting1[math.ceil(pt_transpose[0,0]), math.ceil(
                        pt_transpose[0,1])]
                    P1_wp = 1/np.linalg.norm(np.array([pt_transpose[0,0],
                        pt_transpose[0,1]]))
                    P1_wq = 1/np.linalg.norm(np.array([pt_transpose[0,0],1-
                        pt_transpose[0,1]]))
                    P1_wr = 1/np.linalg.norm(np.array([1-pt_transpose[0,0],
                        pt_transpose[0,1]]))
                    P1_ws = 1/np.linalg.norm(np.array([1-pt_transpose[0,0],1-
                        pt_transpose[0,1]]))
                    RGB_Value = (P1_p*P1_wp+P1_q*P1_wq+P1_r*P1_wr+P1_s*P1_ws)/(
                        P1_wp+P1_wq+P1_wr+P1_ws)
                    Painting1On3[row,col] = (P1_p*P1_wp+P1_q*P1_wq+P1_r*P1_wr+
                        P1_s*P1_ws)/(P1_wp+P1_wq+P1_wr+P1_ws)
            else:
                if pt_transpose[0, 0] > 0 and pt_transpose[0, 1] > 0 and
                    pt_transpose[0, 0] < Painting4.shape[0] - 1 and pt_transpose
                    [0, 1] < Painting4.shape[1] - 1:
                    P4_p = Painting4[math.floor(pt_transpose[0,0]), math.floor(
                        pt_transpose[0,1])]
                    P4_q = Painting4[math.floor(pt_transpose[0,0]), math.ceil(
                        pt_transpose[0,1])]
                    P4_r = Painting4[math.ceil(pt_transpose[0,0]), math.floor(
                        pt_transpose[0,1])]
                    P4_s = Painting4[math.ceil(pt_transpose[0,0]), math.ceil(
                        pt_transpose[0,1])]
                    P4_wp = 1/np.linalg.norm(np.array([pt_transpose[0,0],
                        pt_transpose[0,1]]))
                    P4_wq = 1/np.linalg.norm(np.array([pt_transpose[0,0],1-
                        pt_transpose[0,1]]))
                    P4_wr = 1/np.linalg.norm(np.array([1-pt_transpose[0,0],
                        pt_transpose[0,1]]))
                    P4_ws = 1/np.linalg.norm(np.array([1-pt_transpose[0,0],1-
                        pt_transpose[0,1]]))
                    RGB_Value = (P4_p*P4_wp+P4_q*P4_wq+P4_r*P4_wr+P4_s*P4_ws)/(
                        P4_wp+P4_wq+P4_wr+P4_ws)
                    Painting4On6[row,col] = (P4_p*P4_wp+P4_q*P4_wq+P4_r*P4_wr+
                        P4_s*P4_ws)/(P4_wp+P4_wq+P4_wr+P4_ws)

        #print("row:{0:.0f}% col:{0:.0f}% ".format((row/(painting_blackout.shape
            [0]-1) * 100), (col/(painting_blackout.shape[1]-1) * 100)), end='\r
            ')
    print("Estimated_processed_pixel:_{0:.0f}%_\r".format((row/(
        painting_blackout.shape[0]-1)) * 100), end='\r')
if k < 3:
    print('Painting_1_on_3.jpg')
    cv2.imwrite('Painting_1_on_3.jpg',Painting1On3)
else:
```

```
            print('Painting_4_on_6.jpg')
            cv2.imwrite('Painting_4_on_6.jpg',Painting4On6)
        #cv2.imshow("Warped Source Image", Painting1On3)
        #cv2.waitKey(0)
cv2.destroyAllWindows()
```