ECE661 Computer Vision HW2

Chengjun Guo
guo456@purdue.edu

7 September 2022

1 Logic

I defined two functions for multiple use in three questions. The code used are mostly the same for both tasks. The only difference are the variable names and values for points and images.

The function calculate_h is defined for calculating H based on the algorithm I would mention in the next section. H have two lists as inputs which contain the corresponding points from the two pictures. With this function, the homography is returned for further transformation.

The function mapping_X_to_X_prime is defined to accomplish the transformation. Image_X and Image_X_prime are defined graphs where X stands for the graph to be applied with transformation and X prime stand for the image including graph to be replaced. Four points of X prime is the corners of the area to be replaced. H is the algorithm that will be used for transformation. The result image would be returned by the image.

1.1 Steps to calculate homographies

First we know that the homography for corresponding points fulfills that:

$$\begin{bmatrix} x_1' \\ x_2' \\ x_3' \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{1}$$

After simplication of the equation, each pair of corresponding points can result in two equations:

$$xh_{11} + yh_{12} + h_{13} - xx'h_{31} - yx'h_{32} = x'h_{33} \tag{2}$$

$$xh_{21} + yh_{22} + h_{23} - xy'h_{31} - yy'h_{32} = y'h_{33} \tag{3}$$

where $x = \frac{x_1}{x_3}, y = \frac{x_2}{x_3}$. Same to x' and y'.

Since only ratio matters in homography, h_{33} can be set to 1 manually and there are 8 unknowns left, 8 equations are needed to solve for the homography which means 4 pairs of corresponding points are needed. Then we can calculate the homography by the following system.

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_1' & -y_1x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_1' & -y_1y_1' \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x_2' & -y_2x_2' \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y_2' & -y_2y_2' \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x_3' & -y_3x_3' \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y_3' & -y_3y_3' \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x_4' & -y_4x_4' \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y_4' & -y_4y_4' \end{bmatrix} \times \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = \begin{bmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ x_3' \\ y_3' \\ x_4' \\ y_4' \end{bmatrix} \tag{4}$$

With this function, we can find the rest 8 entries of H by calculating the inverse of the 8 by 8 matrix multiplying the 8 by 1 column vector.

1.2 Steps for mapping

In case there are undefined points in the result picture, I loop through all the points inside the polygon of PQRS in target picture. Then I replace all those points by the point at $H^{-1}X'$ in origin image. This would promise all the points inside the target PQRS are replaced and the points outside would not be influenced.

2 Task1

2.1 Inputs and Intermediate outputs

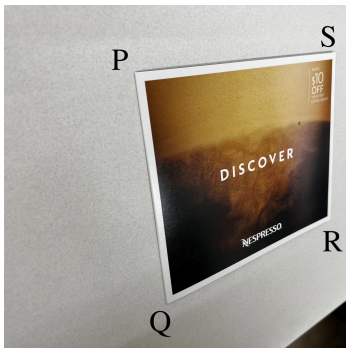
The PQRS used for card1,2,3 and car:1

	card1	card2	card3	car
P	(487,250)	(319,230)	(588,47)	(60,40)
Q	(610,1114)	(205,853)	(64,592)	(60,507)
R	(1221,799)	(873,1128)	(704,1215)	(716,507)
S	(1244,171)	(1042,232)	(1232,676)	(716,40)

Table 1: Points

The 4 picture used for task 1 :1

(a) car



(b) card1

(c) card2

(d) card3

Figure 1: input for task1

The PQRS of pictures marked with red circles:2

2.2 Result

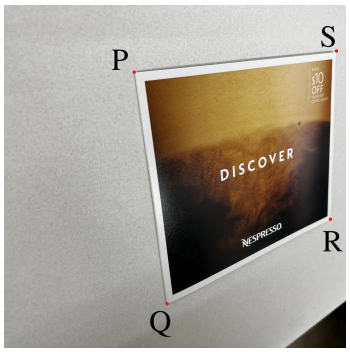
2.2.1 Result for question1

The output of the projection:3

2.2.2 Result for question 2

The output of the projection from card1 to card3:4. In this question, since I'm looping through the points from the output and find the corresponding points from the input. The way to multiply inverses of H_{ab} and H_{bc} should be reversed.

(a) carROI



(b) card1PQRS

(c) card2PQRS

(d) card3PQRS

Figure 2: intermediate output for task1

2.2.3 Result for question 3

The output of the affine homography:5. In this question, the output for card3 has the best performance. It is because the third PQRS frame is more likely to keep the property of PQ parallel to RS and PS parallel to QR, which means the transformation from car to card3 is more like a affine transformation. It can be found that the h_{31} and h_{32} of H of card3 is way less than those of H of card1,2.

3 Task2

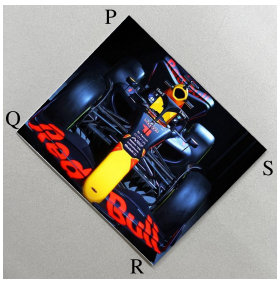
3.1 Inputs and intermediate outputs

The PQRS used for wjc1,2,3 and cat:2

The 4 picture used for task 2 :6

(a) 1.1 card1

(b) 1.1 card2



(c) 1.1 card3

Figure 3: output for 1.1

	wjc1	wjc2	wjc3	cat
P	(1090, 303)	(74, 239)	(694,170)	(0,0)
Q	(1091, 662)	(74, 574)	(694,831)	(0,1919)
R	(1530, 700)	(412, 552)	(1353,832)	(1919,1919)
S	(1531, 275)	(411, 258)	(1362,169)	(1919,0)

Table 2: Points2

The PQRS of pictures marked with red circles:7

3.1.1 Result for question1

The output of the projection:8

3.1.2 Result for question 2

The output of the projection from wjc1 to wjc3:9. In this question, since I'm looping through the points from the output and find the corresponding points from the input. The way to multiply inverses of H_{ab} and H_{bc} should be reversed.

3.1.3 Result for question 3

The output of the affine homography:**??**. In this question, the output for wjc3 has the best performance. It is because the third PQRS frame is more like a affine transformation and less projective. It can be found that the h_{31} and h_{32}

Figure 4: output for 1.2

of H of wjc3 is less than those of H of wjc1,2.

4 Code

4.1 Code for task1

```
import sys
# print(sys.version)

import numpy as np
import cv2
import sys
# read 4 images

car = cv2.imread('car.jpg')  # (559, 761, 3) H W RGB
```

(a) 1.3 card1 (b) 1.3 card2 (c) 1.3 card3

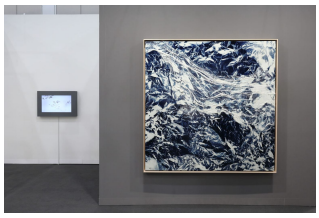
Figure 5: output for 1.3

```
card1 = cv2.imread('card1.jpeg')  # (1280, 1280, 3)
card2 = cv2.imread('card2.jpeg')  # (1280, 1280, 3)
card3 = cv2.imread('card3.jpeg')  # (1280, 1280, 3)

# car ROI: x1y1:(60,40) x2y2:(716,507)
# cv2.rectangle(car, (60, 40), (716, 507), (0,0,255), 2)
# cv2.imshow('carROI', car)

# PQRS
# card1:P (487,250) Q:(610,1114) R:(1221,799) S:(1244,171)
# card2:P (319,230) Q:(205,853) R: (873,1128) S:(1042,232)
# card3:P (588,47) Q: (64,592) R: (704,1215) S:(1232,676)

# card1 = cv2.circle(card1, (487,250), radius=0, color=(0, 0, 255), thickness=-1)
# cv2.imshow('card1P', card1)
# car = cv2.circle(car,(716,507), radius=2, color=(0,0,255), thickness=-1)
# cv2.imshow("car",car)
ROI_car = np.array([[60, 40], [60, 507], [716, 507], [716, 40]])
PQRS_card_1 = np.array([[487, 250], [610, 1114], [1221, 799], [1244, 171]])
PQRS_card_2 = np.array([[319, 230], [205, 853], [873, 1128], [1042, 232]])
PQRS_card_3 = np.array([[588, 47], [64, 592], [704, 1215], [1232, 676]])
# cv2.rectangle(car, (60, 40), (716, 507), (0,0,255), 2)
# cv2.imwrite('carROI.jpg', car)
# for i in PQRS_card_1:
#     cv2.circle(card1, i, radius=2, color=(0,0,255), thickness=8)
# cv2.imwrite('card1PQRS.jpg', card1)
# for i in PQRS_card_2:
#     cv2.circle(card2, i, radius=2, color=(0,0,255), thickness=8)
# cv2.imwrite('card2PQRS.jpg', card2)
# for i in PQRS_card_3:
#     cv2.circle(card3, i, radius=2, color=(0,0,255), thickness=8)
# cv2.imwrite('card3PQRS.jpg', card3)
```

(a) car

(b) card1

(c) card2



(d) card3

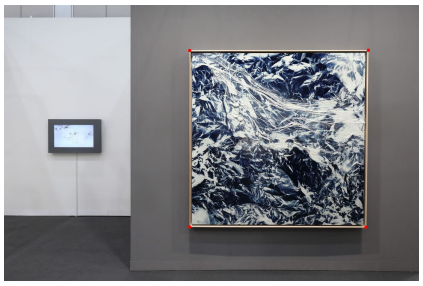
Figure 6: input for task2

```
#
# sys.exit()

# calculate for H
def calculate_h(X, X_prime):
    A = np.zeros((8, 8))
    B = np.zeros((8, 1))
    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] = -X[i][0] * X_prime[i][0]
        A[2 * i, 7] = -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] = -X[i][0] * X_prime[i][1]
        A[2 * i + 1, 7] = -X[i][1] * X_prime[i][1]
        B[2 * i] = X_prime[i][0]
```

(a) catROI

(b) wjc1PQRS

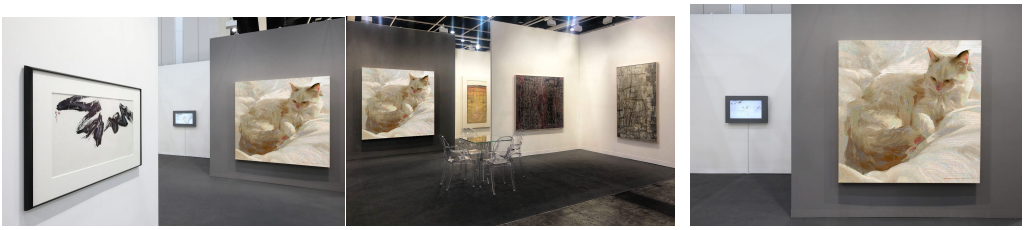
(c) wjc2PQRS



(d) wjc3PQRS

Figure 7: intermediate output for task2

```
        B[2 * i + 1] = X_prime[i][1]

    H_vec = np.ndarray.flatten(np.dot(np.linalg.inv(A), B))
    print(H_vec)
    H = np.ones((3, 3))
    H[0, :] = H_vec[0:3]
    H[1, :] = H_vec[3:6]
    H[2, :2] = H_vec[6:8]
    return H


# 1.1,1.2,1.3
H_d_a = calculate_h(ROI_car, PQRS_card_1)
H_d_b = calculate_h(ROI_car, PQRS_card_2)
H_d_c = calculate_h(ROI_car, PQRS_card_3)
print("(1)", H_d_a)
print("(2)", H_d_b)
print("(3)", H_d_c)
```



(a) 2.1 wjc1 (b) 2.1 wjc2 (c) 2.1 wjc3

Figure 8: output for 2.1

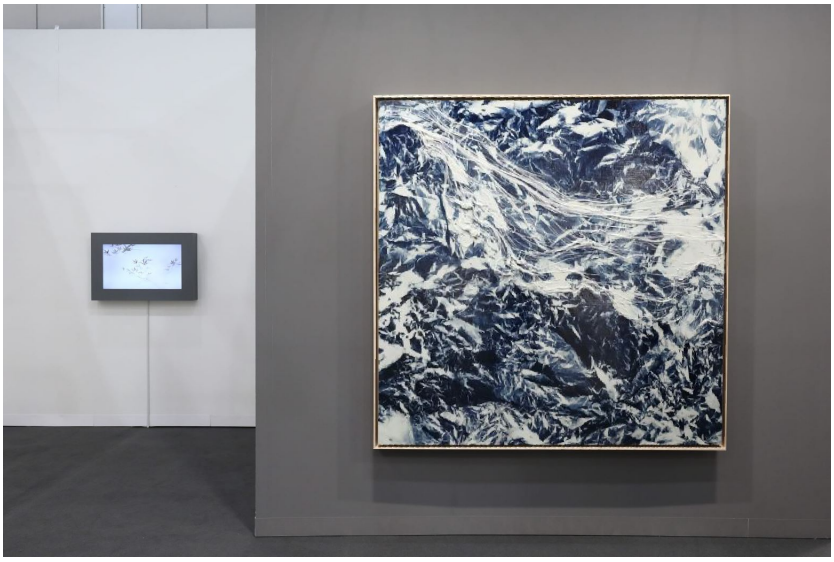
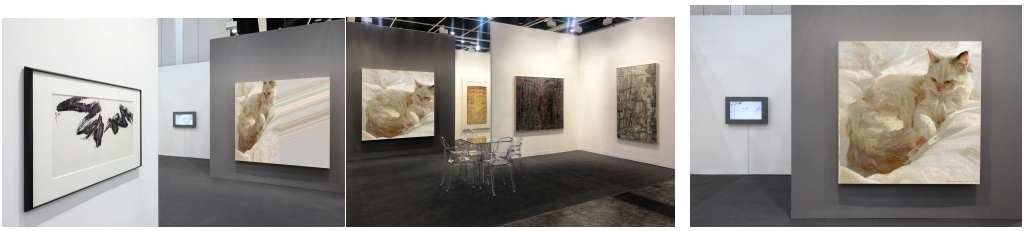


Figure 9: output for 2.2

```
#  2
H_a_b = calculate_h(PQRS_card_1, PQRS_card_2)
H_b_c = calculate_h(PQRS_card_2, PQRS_card_3)
H_a_c = np.matmul(H_b_c, H_a_b)
print("H of A to B\n", H_a_b)
print("H of B to C\n", H_b_c)
print("H of A to C\n", H_a_c)


# mapping from X to X prime

def mapping_X_to_X_prime(Image_X, Image_X_prime, four_points_X_prime, H):
    area = np.zeros(Image_X_prime.shape)
```



(a) 2.3 wjc1 (b) 2.3 wjc2 (c) 2.3 wjc3

Figure 10: output for 2.3

```
    image_to_fill = cv2.fillPoly(area, pts=[four_points_X_prime], color=(255, 255, 255))
    # print(area)
    # cv2.imshow("polygon filled with white pixel",image_to_fill)
    H_inv = np.linalg.inv(H)
    for i in range(len(image_to_fill)):
        for j in range(len(image_to_fill[i])):
            if np.all(image_to_fill[i,j] == 255):
                point_in_x_prime = np.array([j, i, 1])
                point_in_x = np.dot(H_inv, point_in_x_prime)
                x = point_in_x[0] / point_in_x[2]
                y = point_in_x[1] / point_in_x[2]
                if int(x) > 760:
                    x = 760
                if y > 558:
                    y = 558
                Image_X_prime[i,j] = Image_X[int(y),int(x)]
    # cv2.imshow("image_X_prime filled", Image_X_prime)
    return Image_X_prime



# 1
# image_d_a = mapping_X_to_X_prime(car, card1, PQRS_card_1, H_d_a)
# cv2.imwrite("from_car_to_card1.jpg",image_d_a)
# image_d_b = mapping_X_to_X_prime(car, card2, PQRS_card_2, H_d_b)
# cv2.imwrite("from_car_to_card2.jpg",image_d_b)
# image_d_c = mapping_X_to_X_prime(car, card3, PQRS_card_3, H_d_c)
# cv2.imwrite("from_car_to_card3.jpg",image_d_c)
# 2
# image_a_c = mapping_X_to_X_prime(card1, card3, PQRS_card_3, H_a_c)
# cv2.imwrite("from_card1_to_card3.jpg", image_a_c)
# H_b_c * H_a_b * X, I didnt notice the order of the Hs
# 3
# H_d_a[2,0:2] = 0
```

```
# H_d_b[2,0:2] = 0
# H_d_c[2,0:2] = 0
# image_d_a = mapping_X_to_X_prime(car, card1, PQRS_card_1, H_d_a)
# cv2.imwrite("from_car_to_card1_affine.jpg",image_d_a)
# image_d_b = mapping_X_to_X_prime(car, card2, PQRS_card_2, H_d_b)
# cv2.imwrite("from_car_to_card2_affine.jpg",image_d_b)
# image_d_c = mapping_X_to_X_prime(car, card3, PQRS_card_3, H_d_c)
# cv2.imwrite("from_car_to_card3_affine.jpg",image_d_c)


cv2.waitKey()
cv2.destroyAllWindows()
```

4.2 Code for task2

```
import sys
# print(sys.version)

import numpy as np
import cv2

# read 4 images

cat = cv2.imread('cat.jpg')  # (1920, 1920, 3) H W RGB
cyanocollage1 = cv2.imread('wjc1.jpg')  # (974, 1604, 3)
cyanocollage2 = cv2.imread('wjc2.jpg')  # (1017, 1604, 3)
cyanocollage3 = cv2.imread('wjc3.jpg')  # (1034, 1552, 3)

# cat ROI: x1y1:(0,0) x2y2:(1919,1919)
# cv2.rectangle(cat, (0, 0), (1919, 1919), (0,0,255), 2)
# cv2.imshow('catROI', cat)


# card1 = cv2.circle(card1, (487,250), radius=0, color=(0, 0, 255), thickness=-1)
# cv2.imshow('card1P', card1)
# car = cv2.circle(car,(716,507), radius=2, color=(0,0,255), thickness=-1)
# cv2.imshow("car",car)
ROI_cat = np.array([[0, 0], [0, 1919], [1919, 1919], [1919, 0]])
PQRS_cc_1 = np.array([[1090, 303], [1091, 662], [1530, 700], [1531, 275]])
```

```
PQRS_cc_2 = np.array([[74, 239], [74, 574], [412, 552], [411, 258]])
PQRS_cc_3 = np.array([[694,170], [694,831], [1353,832], [1362,169]])
# cv2.rectangle(cat, (0, 0), (1919, 1919), (0,0,255), 2)
# cv2.imwrite('catROI.jpg', cat)
# for i in PQRS_cc_1:
#     cv2.circle(cyanocollage1, i, radius=2, color=(0,0,255), thickness=8)
# cv2.imwrite('wjc1PQRS.jpg', cyanocollage1)
# for i in PQRS_cc_2:
#     cv2.circle(cyanocollage2, i, radius=2, color=(0,0,255), thickness=8)
# cv2.imwrite('wjc2PQRS.jpg', cyanocollage2)
# for i in PQRS_cc_3:
#     cv2.circle(cyanocollage3, i, radius=2, color=(0,0,255), thickness=8)
# cv2.imwrite('wjc3PQRS.jpg', cyanocollage3)
#
# sys.exit()


# calculate for H
def calculate_h(X, X_prime):
    A = np.zeros((8, 8))
    B = np.zeros((8, 1))
    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] = -X[i][0] * X_prime[i][0]
        A[2 * i, 7] = -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] = -X[i][0] * X_prime[i][1]
        A[2 * i + 1, 7] = -X[i][1] * X_prime[i][1]
        B[2 * i] = X_prime[i][0]
        B[2 * i + 1] = X_prime[i][1]

    H_vec = np.ndarray.flatten(np.dot(np.linalg.inv(A), B))
    # print(H_vec)
    H = np.ones((3, 3))
    H[0, :] = H_vec[0:3]
    H[1, :] = H_vec[3:6]
    H[2, :2] = H_vec[6:8]
    return H


# task 1 : 1.1.1,1.1.2,1.1.3
```

```
H_d_a = calculate_h(ROI_cat, PQRS_cc_1)
H_d_b = calculate_h(ROI_cat, PQRS_cc_2)
H_d_c = calculate_h(ROI_cat, PQRS_cc_3)
print("(1)", H_d_a)
print("(2)", H_d_b)
print("(3)", H_d_c)

# task 1 : 1.2
H_a_b = calculate_h(PQRS_cc_1, PQRS_cc_2)
H_b_c = calculate_h(PQRS_cc_2, PQRS_cc_3)
H_a_c = np.matmul(H_b_c, H_a_b)
print("H of A to B\n", H_a_b)
print("H of B to C\n", H_b_c)
print("H of A to C\n", H_a_c)


# mapping from X to X prime

def mapping_X_to_X_prime(Image_X, Image_X_prime, four_points_X_prime, H):
    area = np.zeros(Image_X_prime.shape)
    image_to_fill = cv2.fillPoly(area, pts=[four_points_X_prime], color=(255, 255, 255))
    # print(area)
    # cv2.imshow("polygon filled with white pixel",image_to_fill)
    H_inv = np.linalg.inv(H)
    for i in range(len(image_to_fill)):
        for j in range(len(image_to_fill[i])):
            if np.all(image_to_fill[i,j] == 255):
                point_in_x_prime = np.array([j, i, 1])
                point_in_x = np.dot(H_inv, point_in_x_prime)
                x = point_in_x[0] / point_in_x[2]
                y = point_in_x[1] / point_in_x[2]
                if int(x) > 1919:
                    x = 1919
                if y > 1919:
                    y = 1919
                Image_X_prime[i,j] = Image_X[int(y),int(x)]
    # cv2.imshow("image_X_prime filled", Image_X_prime)
    return Image_X_prime


# task 2
# 1
# image_d_a = mapping_X_to_X_prime(cat, cyanocollage1, PQRS_cc_1, H_d_a)
# cv2.imwrite("from_cat_to_painting1.jpg",image_d_a)
# image_d_b = mapping_X_to_X_prime(cat, cyanocollage2, PQRS_cc_2, H_d_b)
# cv2.imwrite("from_cat_to_painting2.jpg",image_d_b)
```

```
# image_d_c = mapping_X_to_X_prime(cat, cyanocollage3, PQRS_cc_3, H_d_c)
# cv2.imwrite("from_cat_to_painting3.jpg",image_d_c)
# 2
# image_a_c = mapping_X_to_X_prime(cyanocollage1, cyanocollage3, PQRS_cc_3, H_a_c)
# cv2.imwrite("from_painting1_to_painting3.jpg", image_a_c)
# H_b_c * H_a_b * X, I didnt notice the order of the Hs
# 3
H_d_a[2,0:2] = 0
H_d_b[2,0:2] = 0
H_d_c[2,0:2] = 0
image_d_a = mapping_X_to_X_prime(cat, cyanocollage1, PQRS_cc_1, H_d_a)
cv2.imwrite("from_cat_to_painting1_affine.jpg",image_d_a)
image_d_b = mapping_X_to_X_prime(cat, cyanocollage2, PQRS_cc_2, H_d_b)
cv2.imwrite("from_cat_to_painting2_affine.jpg",image_d_b)
image_d_c = mapping_X_to_X_prime(cat, cyanocollage3, PQRS_cc_3, H_d_c)
cv2.imwrite("from_cat_to_painting3_affine.jpg",image_d_c)


cv2.waitKey()
cv2.destroyAllWindows()
```