ECE 661: Homework 2

Shane Fernandes
ferna202@purdue.edu

September 8, 2022

1 Logic

The goal of this homework is to calculate a homography between two images by manually recording the pixel coordinates of a set of corresponding points in two images and using these coordinates to calculate the unknown elements of the homography that relates the images.

We are interested in finding the Homography matrix, which is a 3 by 3 non-singular matrix represented by H. Consider X which represents the homogeneous coordinates of a point in the domain space and X' which represents the homogeneous coordinates of the point in the range space. These two points can be related by the equation $X^{'} = HX$.

To estimate H from point correspondences:

$$X^{'} = HX$$

Writing it in matrix representation we have:

$$\begin{pmatrix} x'_1 \\ x'_2 \\ x'_3 \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

Expanding this we get:

$$x'_1 = h_{11}x_1 + h_{12}x_2 + h_{13}x_3$$
$$x'_2 = h_{21}x_1 + h_{22}x_2 + h_{23}x_3$$
$$x'_3 = h_{31}x_1 + h_{32}x_2 + h_{33}x_3$$

To get x' and y' we divide x'_1 by x'_3 and x'_2 by x'_3 respectively,

$$x' = \frac{x'_1}{x'_3} = \frac{h_{11}x_1 + h_{12}x_2 + h_{13}x_3}{h_{31}x_1 + h_{32}x_2 + h_{33}x_3}$$

$$y' = \frac{x'_2}{x'_3} = \frac{h_{21}x_1 + h_{22}x_2 + h_{23}x_3}{h_{31}x_1 + h_{32}x_2 + h_{33}x_3}$$

Dividing throughout by x_3 and setting $h_{33} = 1$, we get the physical coordinates:

$$x' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + 1}$$

$$y' = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + 1}$$

We can rearrange these equations and write them as follows:

$$xh_{11} + yh_{12} + h_{13} - xx'h_{31} - yx'h_{32} - x' = 0$$

$$xh_{21} + yh_{22} + h_{23} - xy'h_{31} - yy'h_{32} - y' = 0$$

Here we have lesser equations(2) than unknowns(8). We would require at least 8 equations to solve this system of equations. Each point correspondence gives us 2 equations each, and hence for 8 equations we would require 4 point correspondences. Stacking 8 such equations and rewriting them in matrix form,

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x'_1 & -y_1x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y'_1 & -y_1y'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x'_2 & -y_2x'_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y'_2 & -y_2y'_2 \\ x_3 & y_3 & 1 & 0 & 0_5 & 0 & -x_3x'_3 & -y_3x'_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y'_3 & -y_3y'_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x'_4 & -y_4x'_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y'_4 & -y_4y'_4 \end{bmatrix} \begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{pmatrix} = \begin{pmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \\ x'_4 \\ y'_4 \end{pmatrix}$$

which can be represented as $Ah = b$. We are interested in the h vector. $h = A^{-1}b$ gives us the solution. (Solved in code using *np.dot(np.linalg.inv(A), b)*). We can append 1 at the end and reshape this vector into a matrix to obtain the homography matrix.

For task 1.3, where we have to find the affine homography, I used a similar approach. In an affine homography, the last row of the homography matrix is [0, 0, 1]. Hence we have,

$$\begin{pmatrix} x'_1 \\ x'_2 \\ x'_3 \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

After rewriting and rearranging the equations as above we can obtain a matrix as below. (Note: I chose 4 point correspondences in my code and hence the matrix is as such.)

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ x_3 & y_3 & 1 & 0 & 0_5 & 0 \\ 0 & 0 & 0 & x_3 & y_3 & 1 \\ x_4 & y_4 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_4 & y_4 & 1 \end{bmatrix} \begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \end{pmatrix} = \begin{pmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \\ x'_4 \\ y'_4 \end{pmatrix}$$

which can be represented as $Ap = b$. We are interested in the p vector. $p = A^{-1}b$ gives us the solution and we can reshape and append [0, 0, 1] to form the affine homography matrix P.

2 Tasks

2.1 Task 1: Using provided images

(a) View 1 of card image (b) View 2 of card image (c) View 3 of card image

Figure 1: Card Images

Figure 2: Image to be projected

2.1.1 Task 1.1: Calculating Homography and projecting image to ROI

Initially, I obtained bad results after projecting the car image onto the card image. The card image had pixels in the Region of Interest(ROI) which which did not have pixels from the car image.This was because while applying the homography, many points in the car image would get mapped to decimal pixel values in the ROI which aren't valid. To alleviate this, I performed inverse homography. First I created a ROI map which is basically an image where pixels that are in the ROI are marked as 255. This was achieved using *draw.polygon* from the *skimage* library. Then while applying the homography, I traversed only through the points in the ROI and multiplied the points by H^{-1} instead of H. With this I got the each point in the car image that corresponded to each pixel location in the ROI. Examples of the mapping without inverse homography are shown below.

(a) Projected View 1 (b) Projected View 2 (c) Projected View 3

Figure 3: Car Projected onto ROI without inverse homography

Figure 4: Card Images

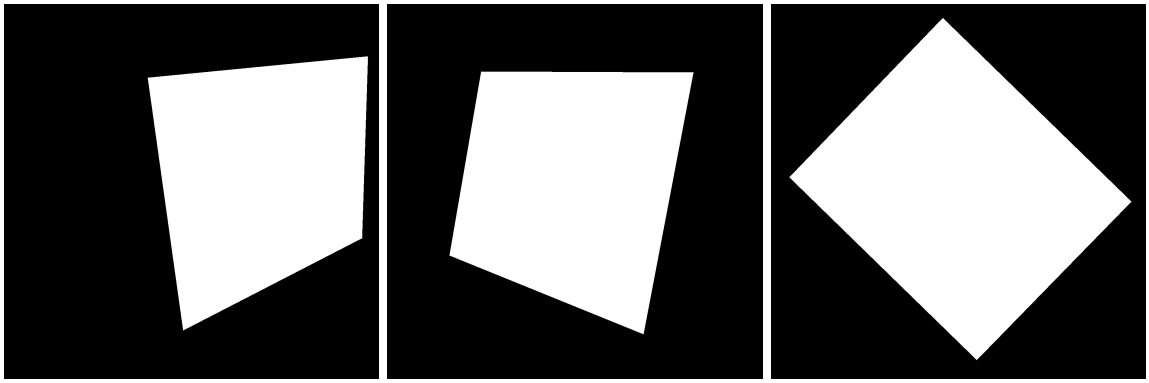


Figure 5: Region of Interest Maps

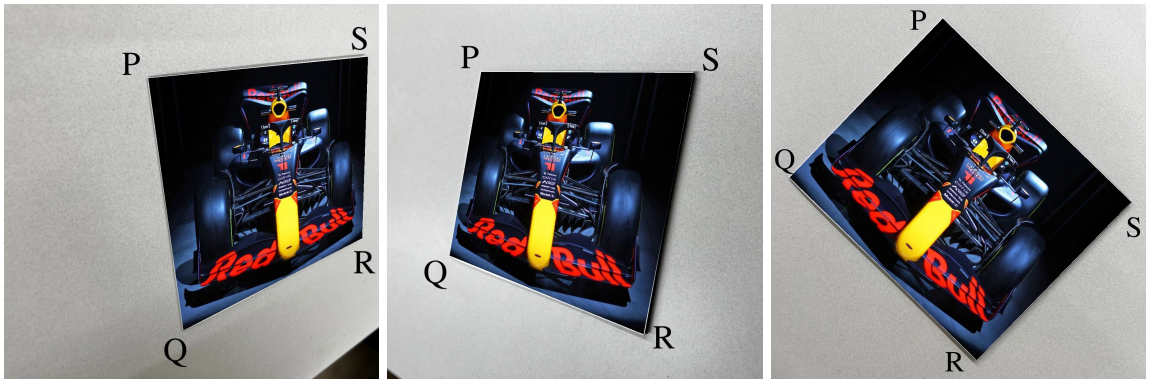


Figure 6: Car Projected onto ROI with inverse homography

2.1.2 Task 1.2: Finding Homographies between images

(a) View 1 of card image (b) View 3 of card image (c) Card 1 projected as card 3

Figure 7: Homography between Images

2.1.3 Task 1.3: Using affine homographies to project the image

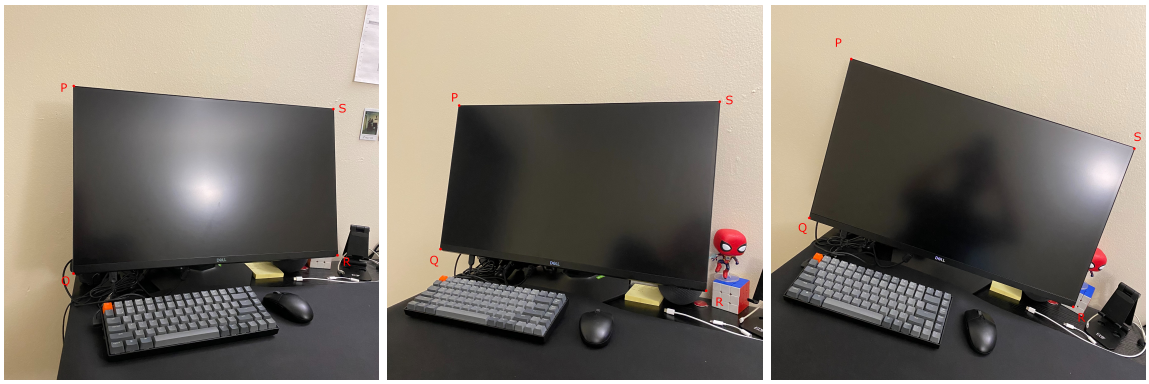
(a) Projected View 1 (b) Projected View 2 (c) Projected View 3

Figure 8: Car projected onto card using affine homography

For the affine homography results, I applied the same inverse homography method. The results are not good except for the third projected view, which still isn't perfect. I guess this is due to the fact that the 4 points in the ROI of the third view makes a polygon that had parallel edges. The inverse affine homography was able to preserve these parallel edges while picking points from the car image. The first two views did not have any parallel edges in the ROI and hence the projection is distorted.

2.2 Task 2: Using captured images

Task 2 consists of capturing and using our own images and repeating the steps in Task 1.

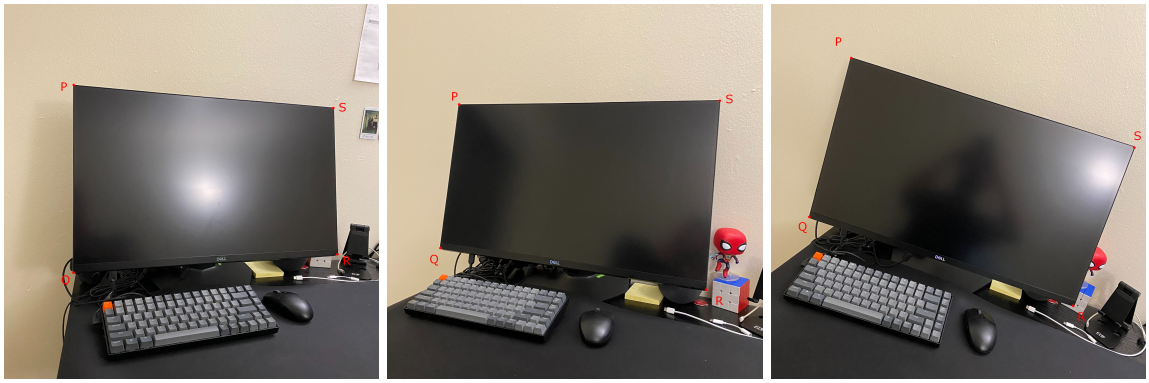


(a) View 1 of my image (b) View 2 of my image (c) View 3 of my image

Figure 9: Card Images

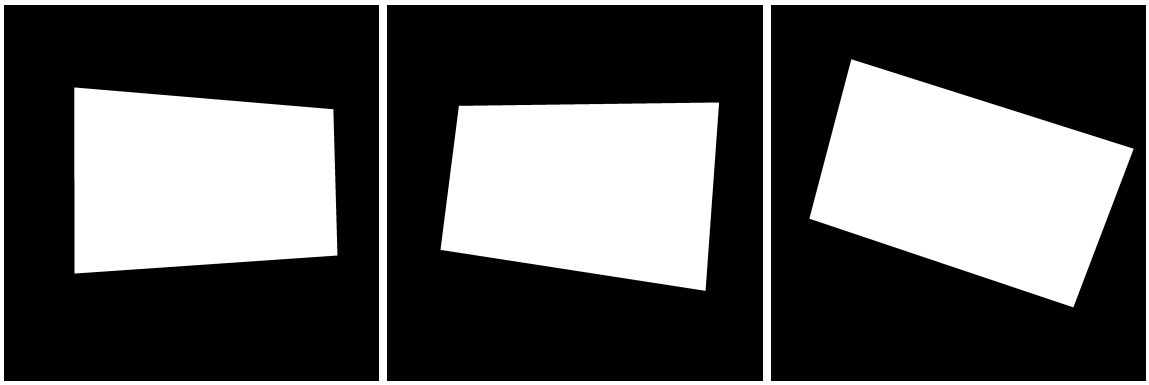
Figure 10: Image to be projected

2.2.1 Task 2.1



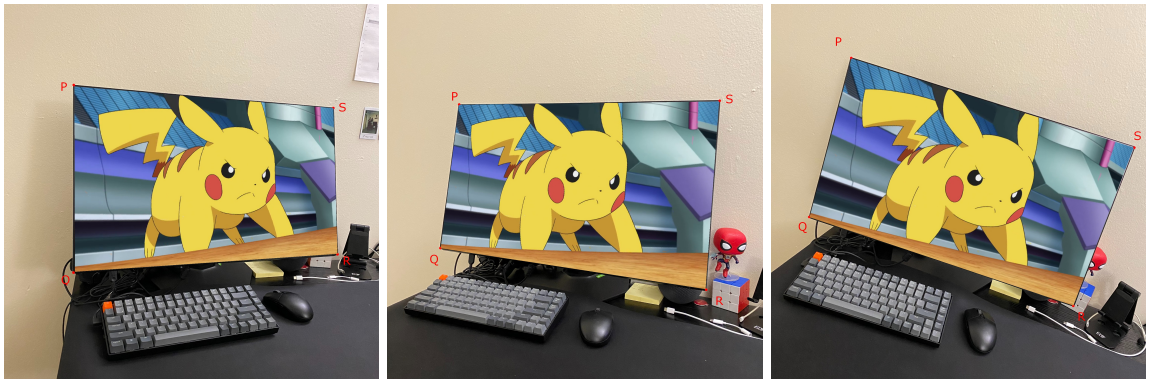
(a) View 1 of my image (b) View 2 of my image (c) View 3 of my image

Figure 11: My Images



(a) ROI Map 1 (b) ROI Map 2 (c) ROI Map 3

Figure 12: Region of Interest Maps



(a) Projected View 1 (b) Projected View 2 (c) Projected View 3

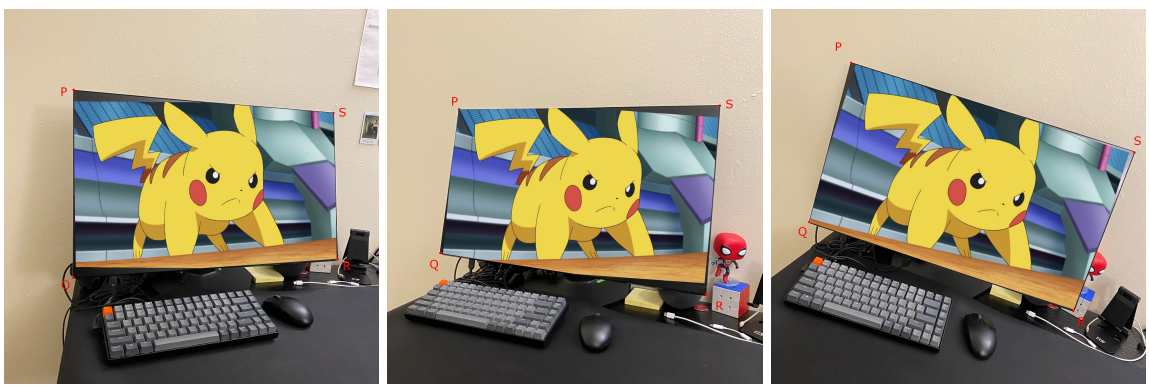
Figure 13: Pikachu Projected onto ROI

2.2.2 Task 2.2: Finding Homographies between images

(a) View 1 of my image (b) View 3 of my image (c) Image 1 projected as Image 3

Figure 14: Homography between Images

2.2.3 Task 2.3: Using affine homographies to project the image



(a) Projected View 1 (b) Projected View 2 (c) Projected View 3

Figure 15: Pikachu projected onto image using affine homography

3 Code Listing

```
from skimage.draw import polygon
import cv2
import numpy as np


# calculate homography matrix
def calc_homography(x_dash, x):
    # Ah=b
    A = np.zeros((8, 8))
    b = x_dash
    # construct the A matrix
    for i in range(0, 4):
        A[2 * i:2 * i + 2][:] = np.array(
```

```
            [[x[2 * i], x[2 * i + 1], 1, 0, 0, 0, -1 * x[2 * i] * x_dash[2 * i], -1 * x[2
            ↪  * i + 1] * x_dash[2 * i]]
                , [0, 0, 0, x[2 * i], x[2 * i + 1], 1, -1 * x[2 * i] * x_dash[2 * i + 1],
                   -1 * x[2 * i + 1] * x_dash[2 * i + 1]]])
    # compute h=(A^-1)b and reshape to form homography matrix
    h = np.dot(np.linalg.inv(A), b)
    H = np.reshape(np.append(h, 1), (3, 3))
    return H


# calculate affine matrix
def calc_affine(x_dash, x):
    # Ap=b
    A = np.zeros((8, 6))
    b = x_dash
    # construct the A matrix
    for i in range(0, 4):
        A[2 * i:2 * i + 2][:] = np.array(
            [[x[2 * i], x[2 * i + 1], 1, 0, 0, 0]
                , [0, 0, 0, x[2 * i], x[2 * i + 1], 1]])

    # compute p=(A^-1)b and reshape to form affine matrix
    p_ = np.dot(np.linalg.pinv(A), b)
    P = np.reshape(np.append(p_, [0, 0, 1]), (3, 3))
    return P


# create a region of interest map which joins the vertices of the roi and fills it with
↪  white
def get_roi_map(pts, dest_img):
    roi_map = np.zeros((dest_img.shape[0], dest_img.shape[1]))
    col = np.array([pts[0], pts[2], pts[4], pts[6]])
    row = np.array([pts[1], pts[3], pts[5], pts[7]])
    rows, cols = polygon(row, col)
    roi_map[rows, cols] = 255
    cv2.imwrite('roi' + str(pts) + '.jpg', roi_map)
    return roi_map


# we apply the inverse homography on points inside the roi map and picks pixel values
# from the source image to paste to the destination image
def apply_transform(dest_img, source_img, H, roi_map=None):
    dest = np.copy(dest_img)
    (W_s, H_s) = source_img.shape[0:2]
    (W_d, H_d) = dest_img.shape[0:2]
    if roi_map is None:
        for i in range(H_d):
            for j in range(W_d):
                x_hc = [i, j, 1]
                xp_hc = np.dot(np.linalg.inv(H), x_hc)  # inverting H for inverse
                ↪  homography
                xp = (xp_hc / xp_hc[2]).astype(int)
                # check if coordinates are within the source image
                if 0 < xp[1] < W_s and 0 < xp[0] < H_s:
```

```
                    dest[j, i] = source_img[xp[1], xp[0]]
    else:
        for i in range(H_d):
            for j in range(W_d):
                if roi_map[j, i] == 255:
                    x_hc = [i, j, 1]
                    xp_hc = np.dot(np.linalg.inv(H), x_hc)  # inverting H for inverse
                    ↪  homography
                    xp = (xp_hc / xp_hc[2]).astype(int)
                    # check if coordinates are within the source image
                    if 0 < xp[1] < W_s and 0 < xp[0] < H_s:
                        dest[j, i] = source_img[xp[1], xp[0]]
    return dest


if __name__ == "__main__":
    # Task 1: Using provided images
    car = cv2.imread('car.jpg')
    card1 = cv2.imread('card1.jpeg')
    card2 = cv2.imread('card2.jpeg')
    card3 = cv2.imread('card3.jpeg')

    # 4 points for car image
    p = [0, 0]
    q = [0, 558]
    r = [760, 558]
    s = [760, 0]
    car_pts = np.array([p, q, r, s]).flatten()

    # 4 points for card1 image
    p_dash1 = [490, 251]
    q_dash1 = [611, 1114]
    r_dash1 = [1222, 799]
    s_dash1 = [1242, 178]
    card1_pts = np.array([p_dash1, q_dash1, r_dash1, s_dash1]).flatten()

    # 4 points for card2 image
    p_dash2 = [320, 230]
    q_dash2 = [212, 858]
    r_dash2 = [873, 1127]
    s_dash2 = [1043, 233]
    card2_pts = np.array([p_dash2, q_dash2, r_dash2, s_dash2]).flatten()

    # 4 points for card3 image
    p_dash3 = [586, 47]
    q_dash3 = [62, 591]
    r_dash3 = [702, 1215]
    s_dash3 = [1230, 674]
    card3_pts = np.array([p_dash3, q_dash3, r_dash3, s_dash3]).flatten()

    # Task 1.1: Calculating Homography and projecting image to ROI
    H1 = calc_homography(card1_pts, car_pts)
    roi1 = get_roi_map(card1_pts, card1)
    res1 = apply_transform(card1, car, H1, roi1)
```

```
cv2.imwrite('card1_res.jpg', res1)

H2 = calc_homography(card2_pts, car_pts)
roi2 = get_roi_map(card2_pts, card2)
res2 = apply_transform(card2, car, H2, roi2)
cv2.imwrite('card2_res.jpg', res2)

H3 = calc_homography(card3_pts, car_pts)
roi3 = get_roi_map(card3_pts, card3)
res3 = apply_transform(card3, car, H3, roi3)
cv2.imwrite('card3_res.jpg', res3)

# Task 1.2: Finding Homographies between images
H4 = calc_homography(card2_pts, card1_pts)
H5 = calc_homography(card3_pts, card2_pts)
H6 = np.matmul(H5, H4)
temp_res = np.zeros_like(card1)
res4 = apply_transform(temp_res, card1, H6)
cv2.imwrite('res4.jpg', res4)

# Task 1.3: Using affine homographies to project the image
Haff_1 = calc_affine(card1_pts, car_pts)
res1a = apply_transform(card1, car, Haff_1, roi1)
cv2.imwrite('cardaff1_res.jpg', res1a)

Haff_2 = calc_affine(card2_pts, car_pts)
res2a = apply_transform(card2, car, Haff_2, roi2)
cv2.imwrite('cardaff2_res.jpg', res2a)

Haff_3 = calc_affine(card3_pts, car_pts)
res3a = apply_transform(card3, car, Haff_3, roi3)
cv2.imwrite('cardaff3_res.jpg', res3a)

# Task 2: Using captured images
pik = cv2.imread('pikachu.png')
myimg1 = cv2.imread('myimg1.jpg')
myimg2 = cv2.imread('myimg2.jpg')
myimg3 = cv2.imread('myimg3.jpg')

# 4 points for my images
# pikachu image
p1 = [0, 0]
q1 = [0, 674]
r1 = [1199, 674]
s1 = [1199, 0]
pik_pts = np.array([p1, q1, r1, s1]).flatten()

# myimage1
p1_dash1 = [374, 438]
q1_dash1 = [376, 1428]
r1_dash1 = [1778, 1332]
s1_dash1 = [1756, 554]
myimg1_pts = np.array([p1_dash1, q1_dash1, r1_dash1, s1_dash1]).flatten()
```

```
# myimage2
p1_dash2 = [382, 536]
q1_dash2 = [284, 1302]
r1_dash2 = [1694, 1520]
s1_dash2 = [1766, 518]
myimg2_pts = np.array([p1_dash2, q1_dash2, r1_dash2, s1_dash2]).flatten()

# myimage 3
p1_dash3 = [428, 288]
q1_dash3 = [204, 1136]
r1_dash3 = [1612, 1608]
s1_dash3 = [1934, 764]
myimg3_pts = np.array([p1_dash3, q1_dash3, r1_dash3, s1_dash3]).flatten()

# Task 2.1: Calculating Homography and projecting image to ROI
H2_1 = calc_homography(myimg1_pts, pik_pts)
roi2_1 = get_roi_map(myimg1_pts, myimg1)
res2_1 = apply_transform(myimg1, pik, H2_1, roi2_1)
cv2.imwrite('img1_res.jpg', res2_1)

H2_2 = calc_homography(myimg2_pts, pik_pts)
roi2_2 = get_roi_map(myimg2_pts, myimg2)
res2_2 = apply_transform(myimg2, pik, H2_2, roi2_2)
cv2.imwrite('img2_res.jpg', res2_2)

H2_3 = calc_homography(myimg3_pts, pik_pts)
roi2_3 = get_roi_map(myimg3_pts, myimg3)
res2_3 = apply_transform(myimg3, pik, H2_3, roi2_3)
cv2.imwrite('img3_res.jpg', res2_3)

# Task 2.2: Finding Homographies between images
H2_4 = calc_homography(myimg2_pts, myimg1_pts)
H2_5 = calc_homography(myimg3_pts, myimg2_pts)
H2_6 = np.matmul(H2_5, H2_4)
temp_res2 = np.zeros_like(myimg1)
res5 = apply_transform(temp_res2, myimg1, H2_6)
cv2.imwrite('res5.jpg', res5)

# Task 2.3: Using affine homographies to project the image
Haff2_1 = calc_affine(myimg1_pts, pik_pts)
res2_1a = apply_transform(myimg1, pik, Haff2_1, roi2_1)
cv2.imwrite('myaff1_res.jpg', res2_1a)

Haff2_2 = calc_affine(myimg2_pts, pik_pts)
res2_2a = apply_transform(myimg2, pik, Haff2_2, roi2_2)
cv2.imwrite('myaff2_res.jpg', res2_2a)

Haff2_3 = calc_affine(myimg3_pts, pik_pts)
res2_3a = apply_transform(myimg3, pik, Haff2_3, roi2_3)
cv2.imwrite('myaff3_res.jpg', res2_3a)
```