Teammate : Zhiying He

Part 1

Q1.1 Let $k = \begin{bmatrix} f & 0 & P_x \\ 0 & f & P_y \\ 0 & 0 & 1 \end{bmatrix}$ be camara's intrinsic matrix

Let $L_1 = t\vec{d}$ be the parallel line to L

L_1 must intersect the image plane at vanishing point of L

so point $t\vec{d}$ will map to vanishing point

$$k(t\vec{d}) = t \begin{bmatrix} f & 0 & P_x \\ 0 & f & P_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} = t \begin{bmatrix} f d_x + P_x d_z \\ f d_y + P_y d_z \\ d_z \end{bmatrix} = t d_z \begin{bmatrix} f d_x / d_z + P_x \\ f d_y / d_z + P_y \\ 1 \end{bmatrix}$$

so the vanishing point is $\begin{bmatrix} f d_x / d_z + P_x \\ f d_y / d_z + P_y \end{bmatrix}$

Q1.2

Let $\begin{bmatrix} f dx_1/dz_1 + P_x \\ f dy_1/dz_1 + P_y \end{bmatrix}$ $\begin{bmatrix} f dx_2/dz_2 + P_x \\ f dy_2/dz_2 + P_y \end{bmatrix}$

be 2 different vanishing points a, b

$$a - b = f\begin{bmatrix} \frac{dx_1}{dz_1} - \frac{dx_2}{dz_2} \\ \frac{dy_1}{dz_1} - \frac{dy_2}{dz_2} \end{bmatrix}$$

if there is one entry $=0$, we can easily write it as

$a-b = s\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ or $a-b = s\begin{bmatrix} 0 \\ 1 \end{bmatrix}$ for some scaler s

so the direction vector is constant meaning vanishing points are on the same line

if two entries are all not 0

since $n_x dx + n_y dy + n_z dz = 0$ for all $\vec{d}$

$$n_x \frac{dx}{dz} + n_y \frac{dy}{dz} + n_z = 0$$

We have $\begin{cases} n_x \frac{dx_1}{dz_1} + n_y \frac{dy_1}{dz_1} + n_z = 0 \\ n_x \frac{dx_2}{dz_2} + n_y \frac{dy_2}{dz_2} + n_z = 0 \end{cases}$

$$\Rightarrow n_x\left(\frac{dx_1}{dz_1} - \frac{dx_2}{dz_2}\right) + n_y\left(\frac{dy_1}{dz_1} - \frac{dy_2}{dz_2}\right) = 0$$

$n_x = n_y = 0$ cannot be true, otherwise the plane is parallel to image plane $\Rightarrow$ no vanishing point

if one of $n_x, n_y \neq 0$ and based on our assumption, the other one $\neq 0$ as well

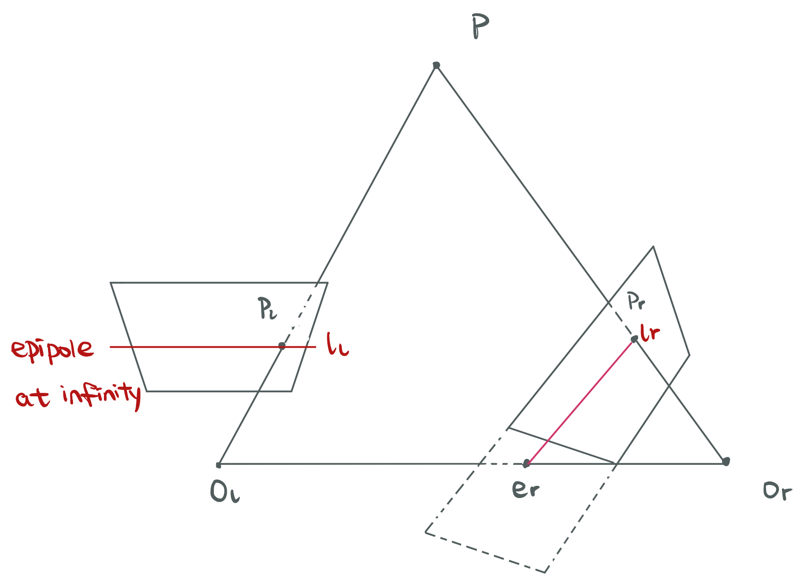
$$\text{so} \quad a - b = f\left(\frac{dy_1}{dz_1} - \frac{dy_2}{dz_2}\right)\begin{bmatrix} -\frac{n_y}{n_x} \\ 1 \end{bmatrix}$$

$$= \frac{f\left(\frac{dy_1}{dz_1} - \frac{dy_2}{dz_2}\right)}{n_x}\begin{bmatrix} -n_y \\ n_x \end{bmatrix}$$

So, the direction vector is $\begin{bmatrix} -n_y \\ n_x \end{bmatrix}$ which only depends on plane.

Thus, vanishing points of all the lines on the plane form a line

Q2



Consider a 3D point P. It intersects right image plane at P_r and left image plane at P_l

line O_lO_r intersects with right image plane at e_r.

Then, the right epipolar line will be e_rP_r

Since the left image plane is parallel to O_lO_r,

the epipole occurs at infinity.

The left epipolar line will be the line that passes through P_l and parallel to O_lO_r

Q3.1

Let $l: ax + by + m = 0$

$l': cx + dy + n = 0$

$$l \times l' = \begin{bmatrix} a \\ b \\ m \end{bmatrix} \times \begin{bmatrix} c \\ d \\ n \end{bmatrix} = \begin{bmatrix} bn - md \\ mc - an \\ ad - bc \end{bmatrix}$$

map it back to 2D $\Rightarrow \begin{bmatrix} \frac{bn-md}{ad-bc} \\ \frac{mc-an}{ad-bc} \end{bmatrix}$

Solve:

$$\begin{cases} ax + by + m = 0 & ① \\ cx + dy + n = 0 & ② \end{cases}$$

From ①

$$x = -\frac{by+m}{a} = -\frac{b}{a}y - \frac{m}{a} \quad ③$$

sub it to ②

$$-\frac{bc}{a}y - \frac{cm}{a} + dy + n = 0$$

$$\left(d - \frac{bc}{a}\right)y = \frac{cm}{a} - n$$

$$y = \frac{cm - an}{ad - bc}$$

sub it back to ③

$$x = \frac{bcm - abn}{abc - a^2d} - \frac{m(bc - ad)}{a(bc - ad)}$$

$$= \frac{adm - abn}{abc - a^2d} = \frac{dm - bn}{bc - ad}$$

Thus this point is the intersection of l, l'

Thus the intersection of l and l' is $p = l \times l'$

Q3.2

Let $p = \begin{bmatrix} a \\ b \end{bmatrix}$, $p' = \begin{bmatrix} c \\ d \end{bmatrix}$

In homogeneous coordinate

$$p \times p' = \begin{bmatrix} a \\ b \\ 1 \end{bmatrix} \times \begin{bmatrix} c \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} b-d \\ c-a \\ ad-bc \end{bmatrix}$$

$$l: (b-d)x + (c-a)y + (ad-bc) = 0$$

Solve the line

let $y = mx + n$ pass through $\begin{bmatrix} a \\ b \end{bmatrix}$ and $\begin{bmatrix} c \\ d \end{bmatrix}$

$$\begin{cases} am + n = b & ① \\ cm + n = d & ② \end{cases}$$

① − ②

$$(a-c)m = b-d$$

$$m = \frac{b-d}{a-c}$$

sub it to ①

$$a\frac{b-d}{a-c} + n = b$$

$$n = b - \frac{ab-ad}{a-c}$$

so $y = \frac{b-d}{a-c}x + b - \frac{ab-ad}{a-c}$

$$\frac{b-d}{a-c}x - y + b - \frac{ab-ad}{a-c} = 0$$

$$(b-d)x + (c-a)y + ab - bc - ab + ad = 0$$

$$l: \quad (b-d)x + (c-a)y + (ad-bc) = 0$$

So the line that goes through 2D points p and p' is

$l = p \times p'$

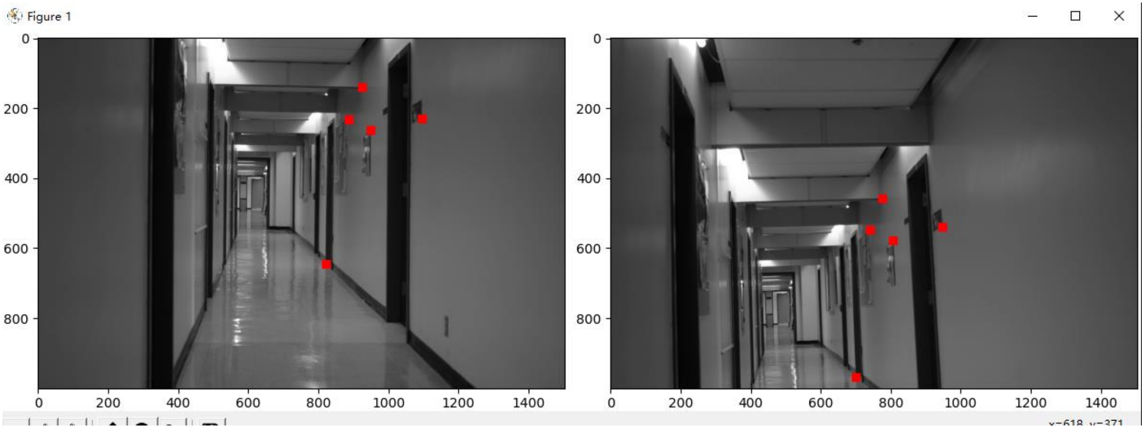
Part 2

Q4.1

This is what it should look like.

I’ve pre-selected groups of points for each case.

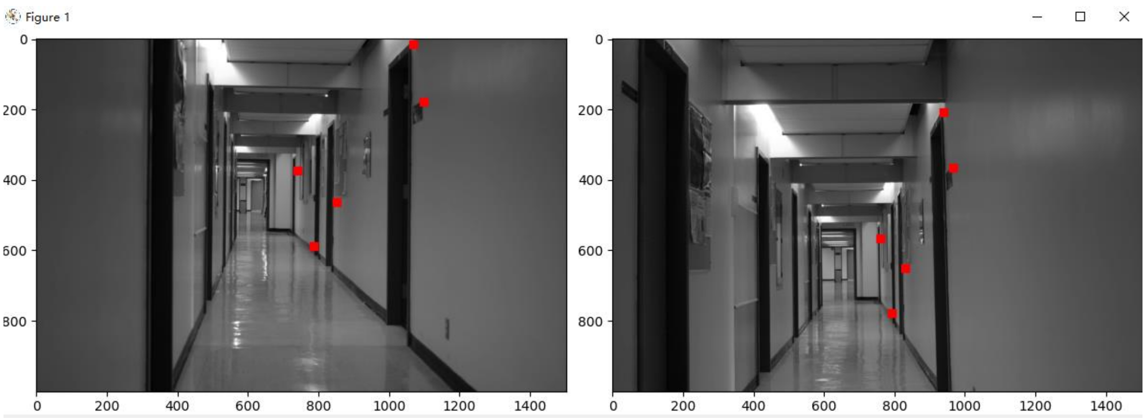
For case A, points are



points_1 = [[821, 645], [950, 263], [886, 231], [926, 141], [1097, 230]]

points_2 = [[700, 967], [807, 577], [742, 548], [777, 458], [948, 539]]

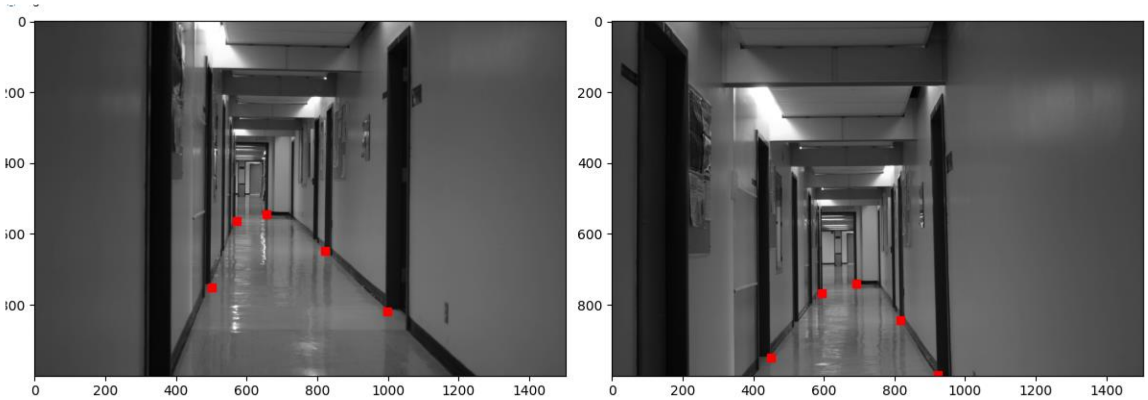
For case B, points are



points_1 = [[1069, 14], [1098, 177], [851, 462], [786, 587], [739, 374]]

points_2 = [[940, 207], [965, 366], [830, 650], [791, 777], [761, 567]]

For case C, points are



points_1 = [[997, 820], [821, 648], [501, 751], [571, 563], [657, 545]]

points_2 = [[922, 997], [817, 844], [449, 950], [595, 766], [693, 739]]

```
167    # Uncomment the line 168 and comment lines 169 - 179 to select points manually
168    # points_1, points_2 = get_points_selected(gray1, gray2)
169    if case == 'A':
170        points_1 = [[821, 645], [950, 263], [
171            886, 231], [926, 141], [1097, 230]]
172        points_2 = [[700, 967], [807, 577], [742, 548], [777, 458], [948, 539]]
173    elif case == 'B':
174        points_1 = [[1069, 14], [1098, 177], [
175            851, 462], [786, 587], [739, 374]]
176        points_2 = [[940, 207], [965, 366], [830, 650], [791, 777], [761, 567]]
177    elif case == 'C':
178        points_1 = [[997, 820], [821, 648], [501, 751], [571, 563], [657, 545]]
179        points_2 = [[922, 997], [817, 844], [449, 950], [595, 766], [693, 739]]
180
181    print(f'selected points for figure 1 are {points_1}')
182    print(f'selected points for figure 2 are {points_2}')
```

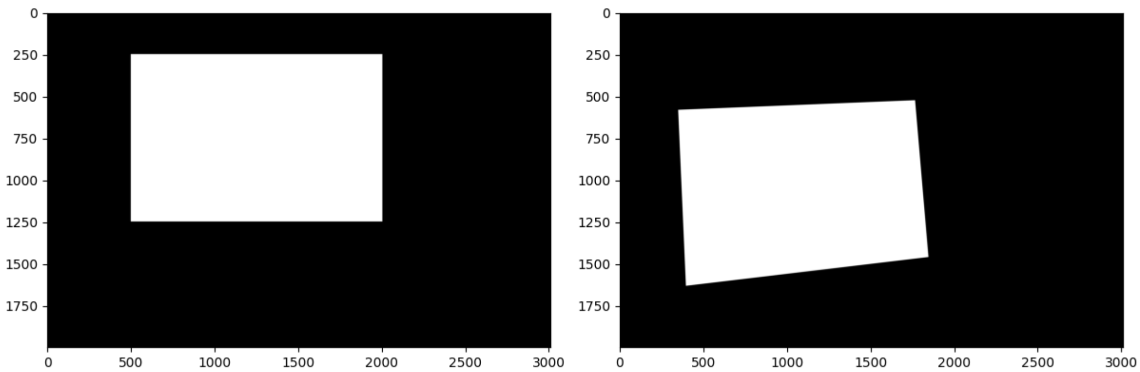
Also, I used mplcursors package to select points directly on images. You can select your own points by commenting out line 169-179 and uncomment line 168. Notice that the program will raise error when less than 5 pairs of points are selected. In line 157, case can be defined manually.

Q4.2

For case A:

```
h is [[ 2.79495076e-03  9.01363946e-05 -5.15784737e-01]
 [ 6.89975879e-06  2.59778049e-03  8.56706042e-01]
 [ 2.01450685e-07 -7.68414372e-08  2.50875770e-03]]
```

Here is its effect.

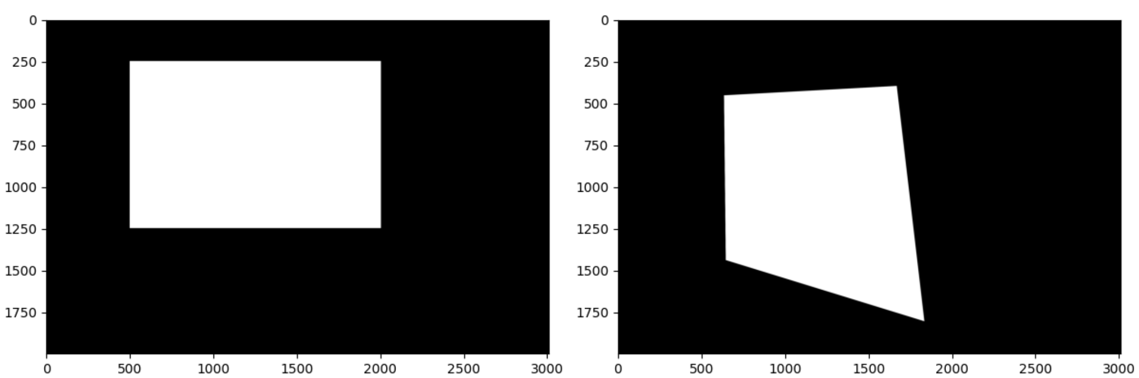


It takes the image, rotate a bit in negative direction and do a projective transformation

For case B:

```
h is [[ 7.01194995e-04 -8.88113748e-05  8.37489217e-01]
 [-2.03598002e-04  1.57043325e-03  5.46447256e-01]
 [-3.37704336e-07 -1.68525719e-07  2.04968354e-03]]
```

Here is its effect.

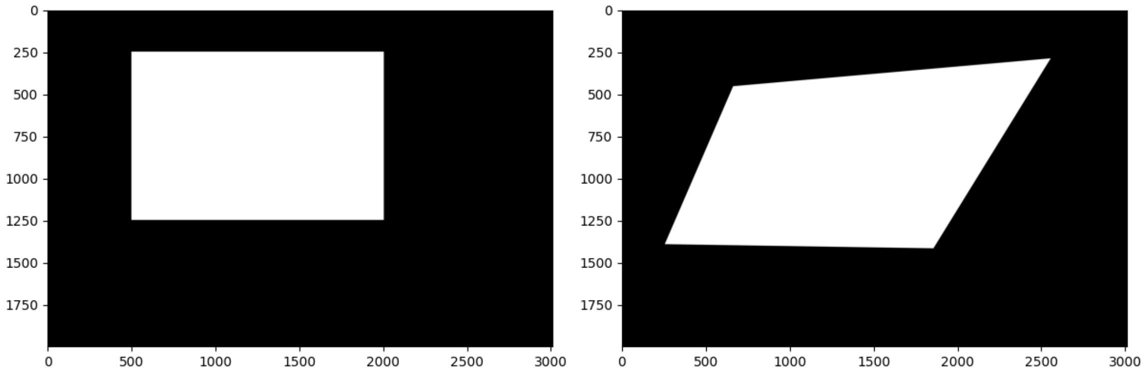


It takes the image, shears vertically and do a projective transformation.

For case C:

```
h is [[ 2.59108835e-03 -1.01791429e-03  7.31199136e-01]
 [-3.82646240e-04  2.91190491e-03  6.82146501e-01]
 [-3.03917514e-07  2.87213020e-07  2.75663207e-03]]
```

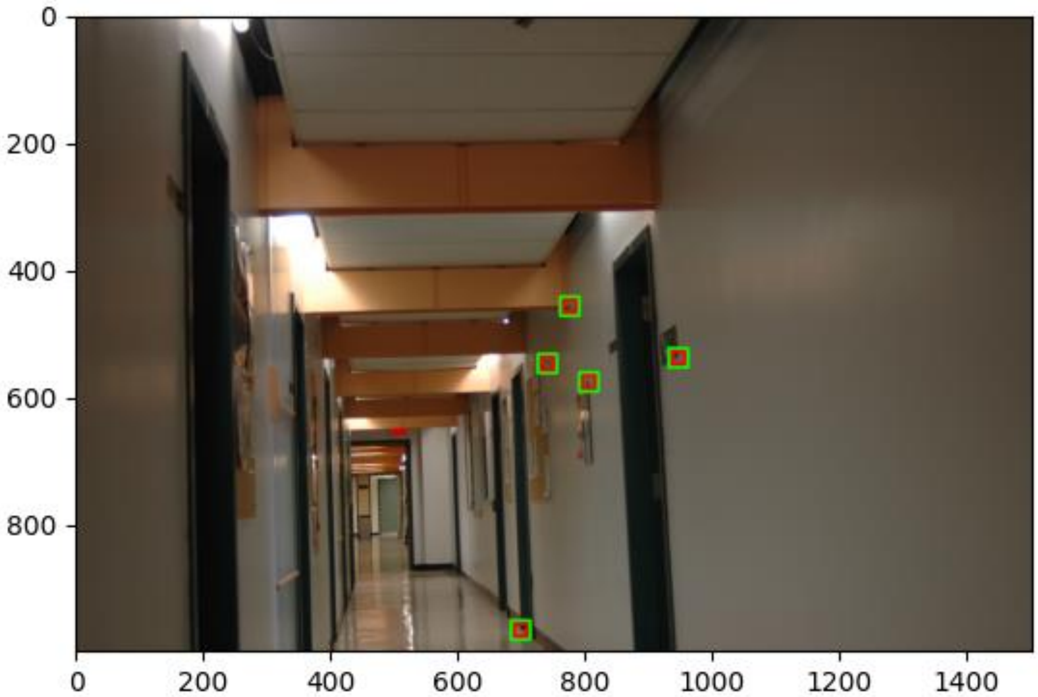
Here is its effect:



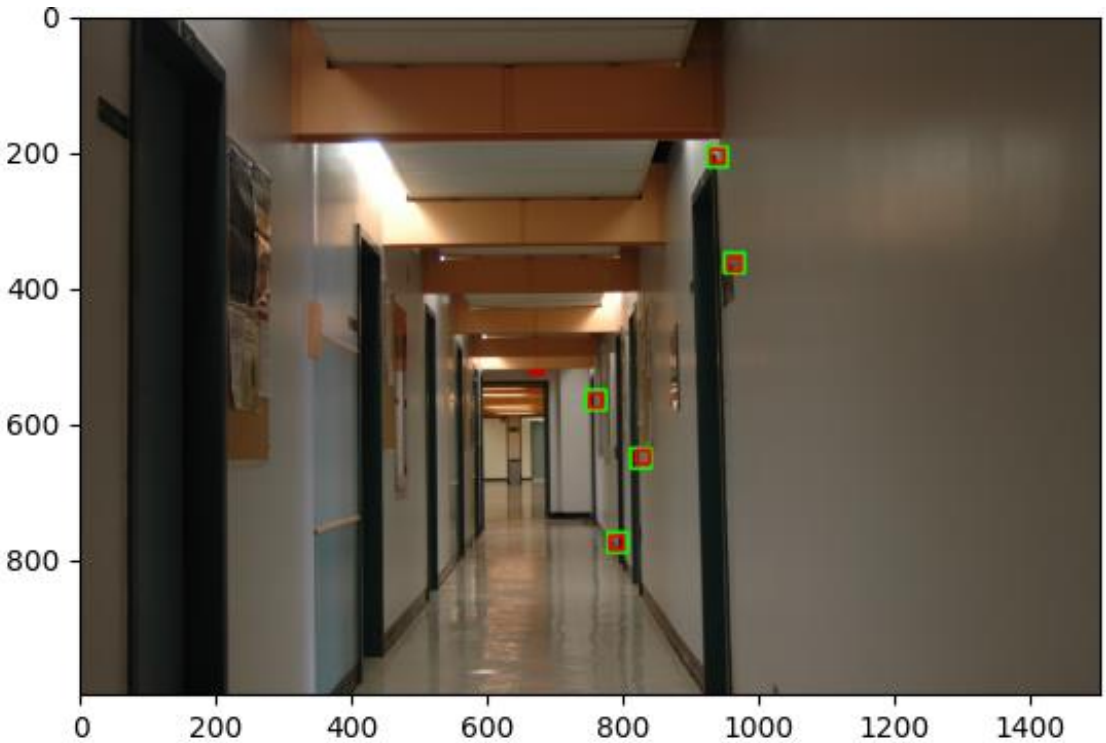
It takes the image, shears horizontally and do a projective transformation.

Q4.3

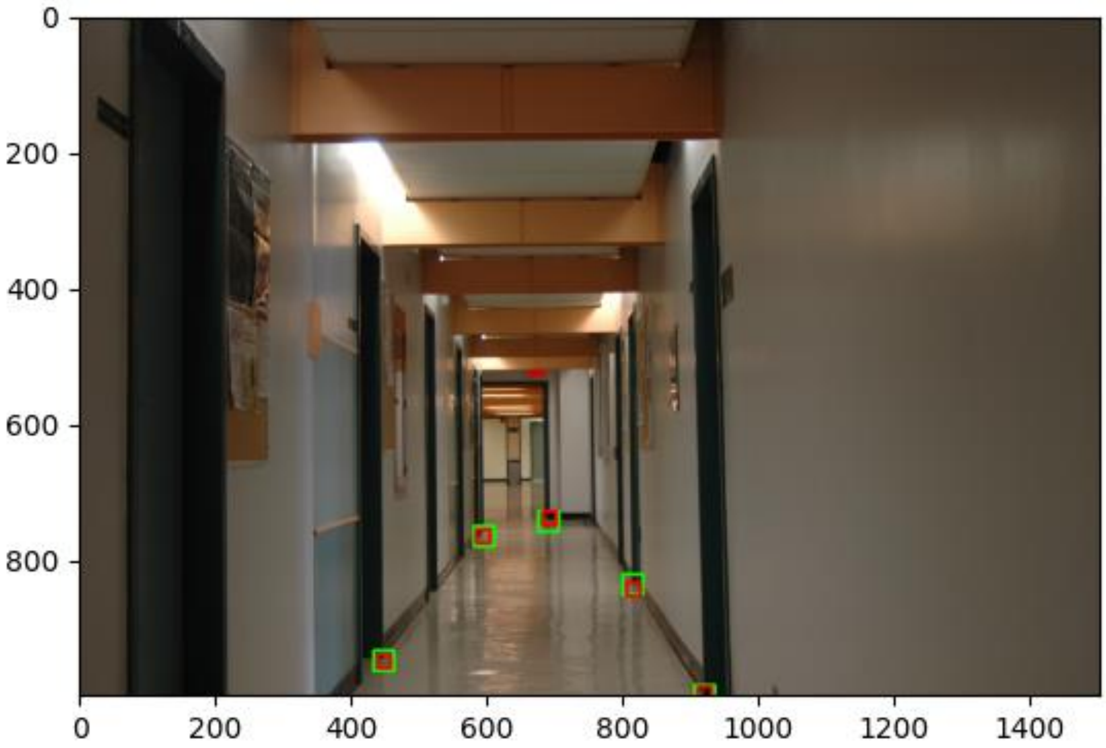
For case A:



For case B:



For case C:

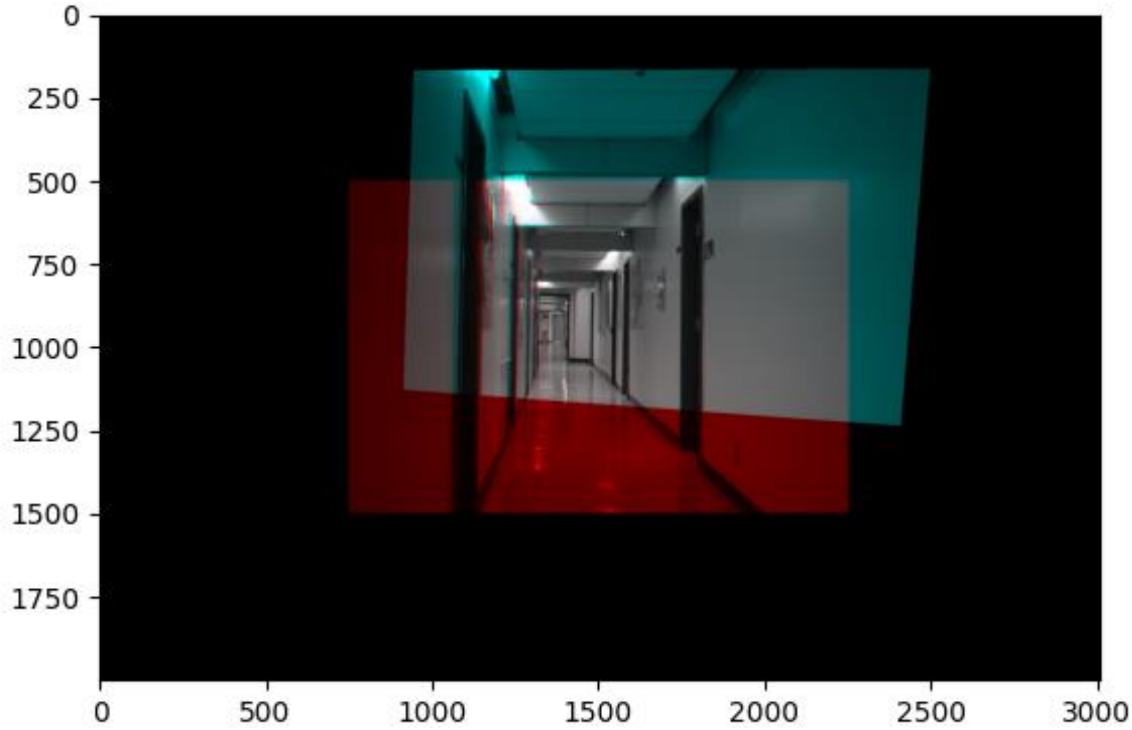


Q4.4

```python
def image_transformation(image1, image2, h):
    height, width = image1.shape[0] * 2, image1.shape[1] * 2

    result_image = np.zeros((height, width, 3), dtype=np.int32)
    # Add some offset to image so every point will appear
    points_cord = [[x, y] for y in range(-500, height - 500)
                   for x in range(-750, width - 750)]
    # Store the coordinates after applying homogeneous
    result_points = np.array(homogeneous_transformation(points_cord, h))

    for i, point in enumerate(result_points):
        if 0 <= point[0] < image2.shape[1] and 0 <= point[1] < image2.shape[0]:
            # If in range, set the Green and Blue channel to be image 2
            result_image[i // width][i % width][1] = image2[point[1]][point[0]]
            result_image[i // width][i % width][2] = image2[point[1]][point[0]]
    # Set the Red channel to be image 1
    result_image[500: image1.shape[0] + 500,
                 750: image1.shape[1] + 750, 0] = image1

    plt.imshow(result_image)
    plt.show()
    return result_image
```

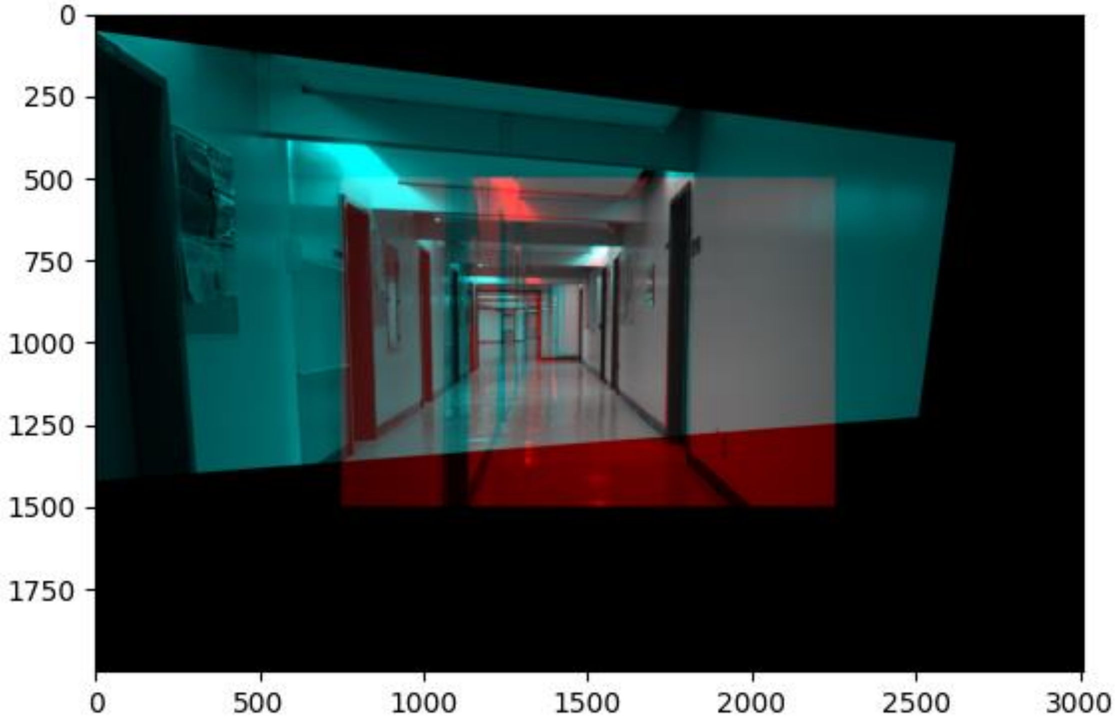
This is the function I use to get the result image. Basically, it will first create a large enough image with black pixels. I use twice of the image1's height and width to be the dimension. I also make some offset to the coordinate. Instead of staring at (0, 0), I start at (-750, -500) so that the result image would contain all transformed coordinates. Then, I shift the red channel 750 pixels right and 500 pixels down. The result images are like this.

For case A:



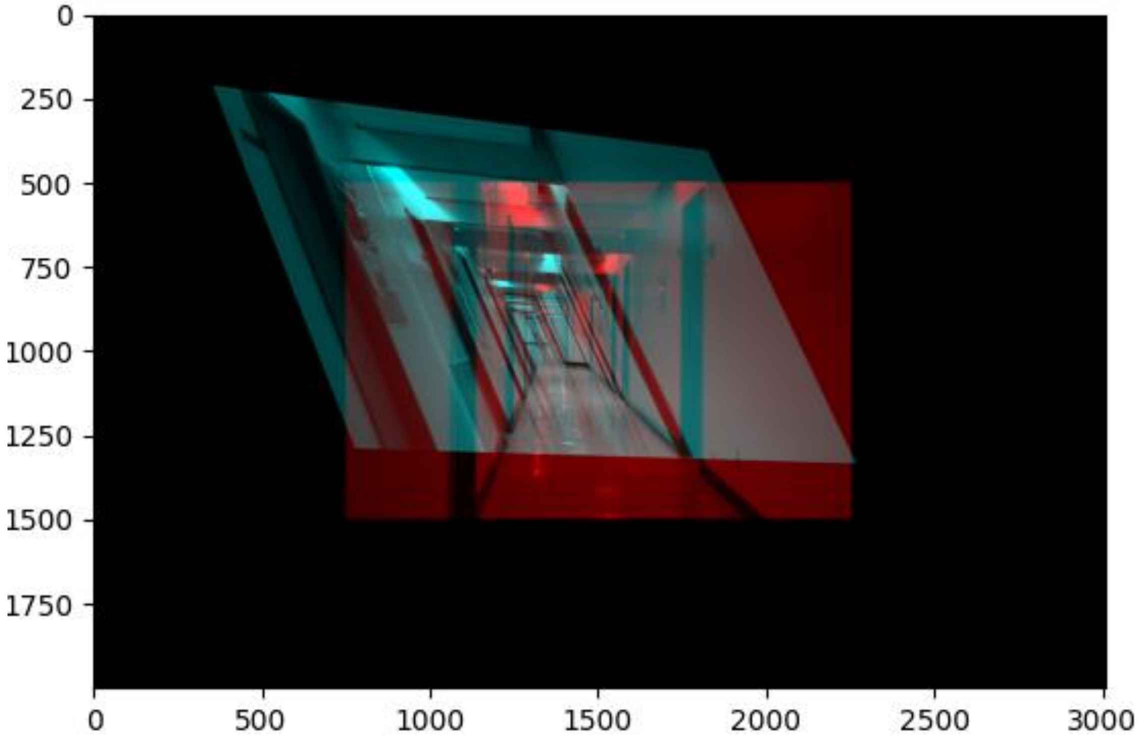
As we can see from the image, the camara rotate a bit to top right direction. Also, the right wall is more Lambertian because it occurs gray in image.

For case B:



As we can see from the image, the camara move to the right and rotate a bit to top left direction. Also, the right wall is more Lambertian because it occurs gray in image.

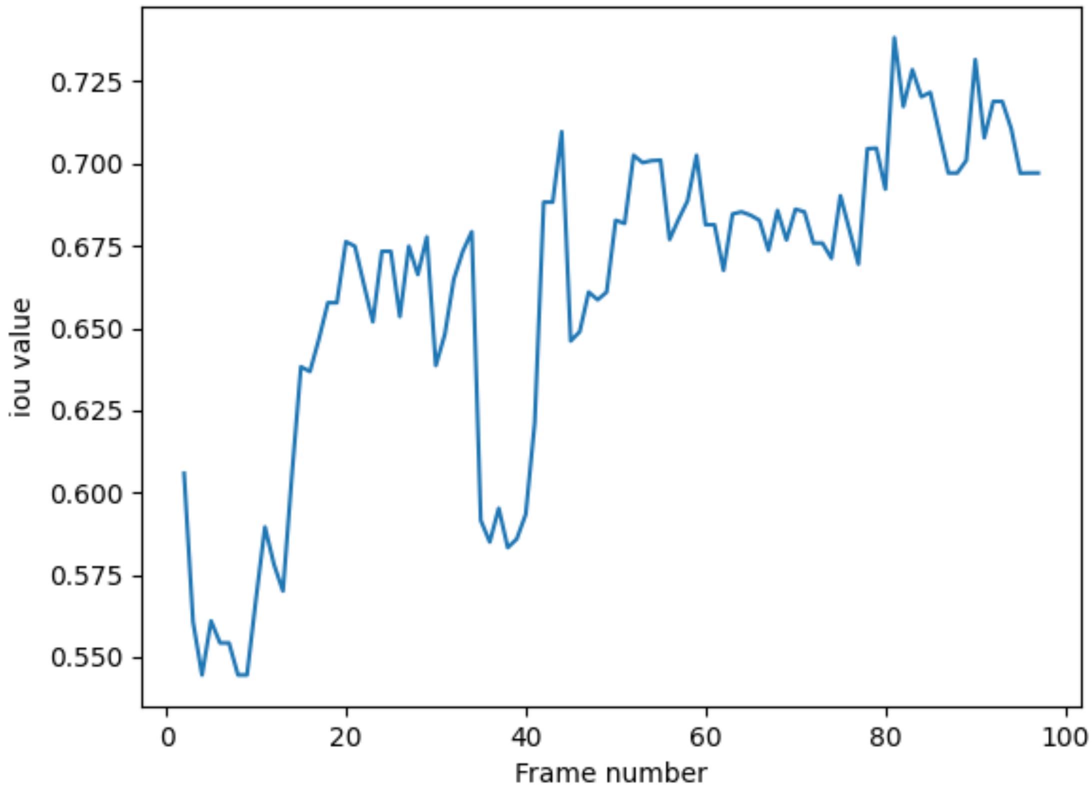
For case C:



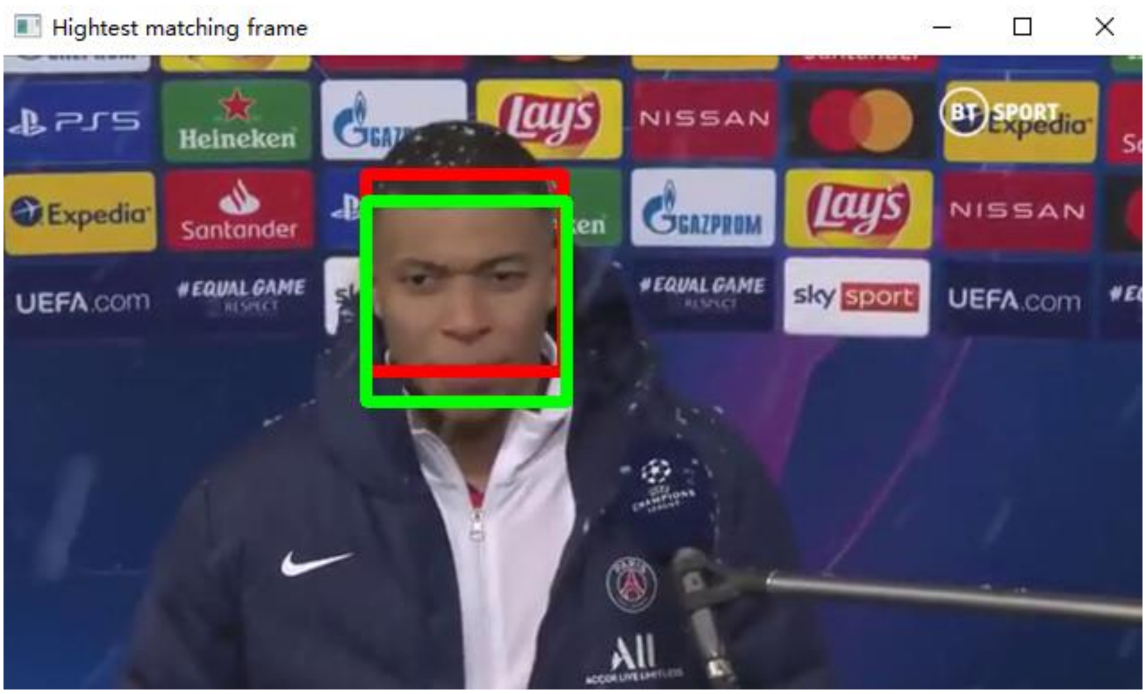
As we can see from the image, the camara move to the right and rotate a bit to top left direction. Also, the floor is less Lambertian because I cannot see any gray on it

Q5.1

This is what plot looks like:



This is the frame where they have the highest match



This is the frame where they have the lowest match

The red box is for mean shift tracking and the green box is for face detector.

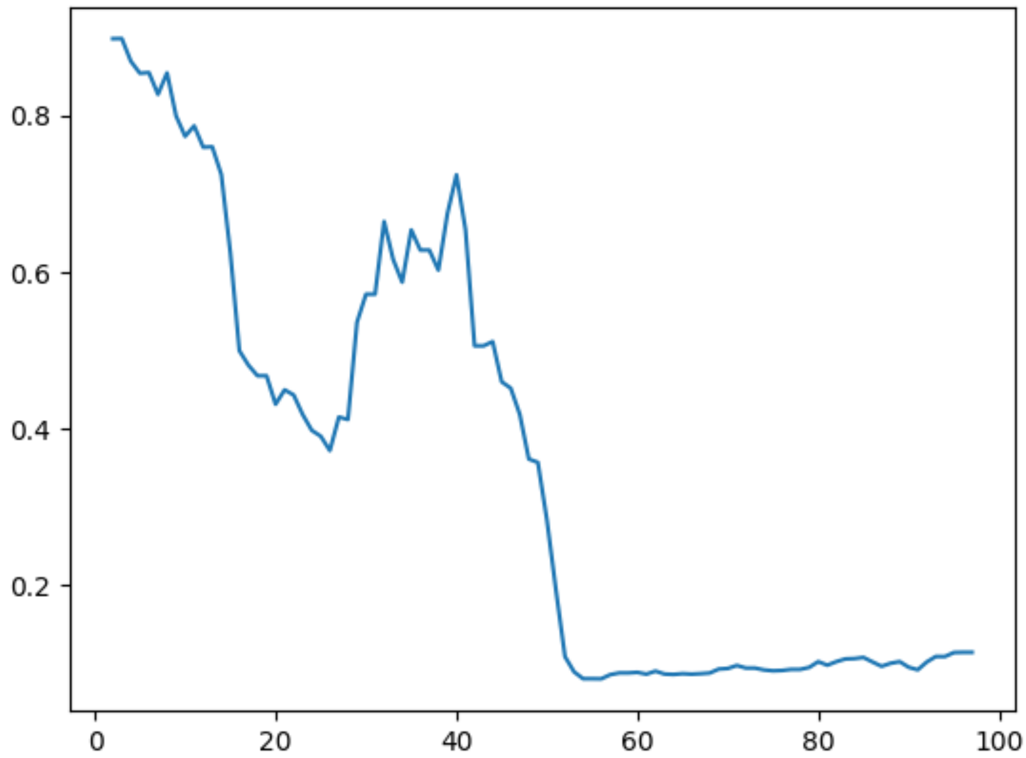
I set the lower threshold to be 0.62 and higher threshold to be 0.68.

43.75% of the frames in which iou is larger than 0.68.

Based on images, I believe that the face detector is correct more often since the red box doesn’t include the jaw part maybe that’s because it is not capable of changing the size of the box.

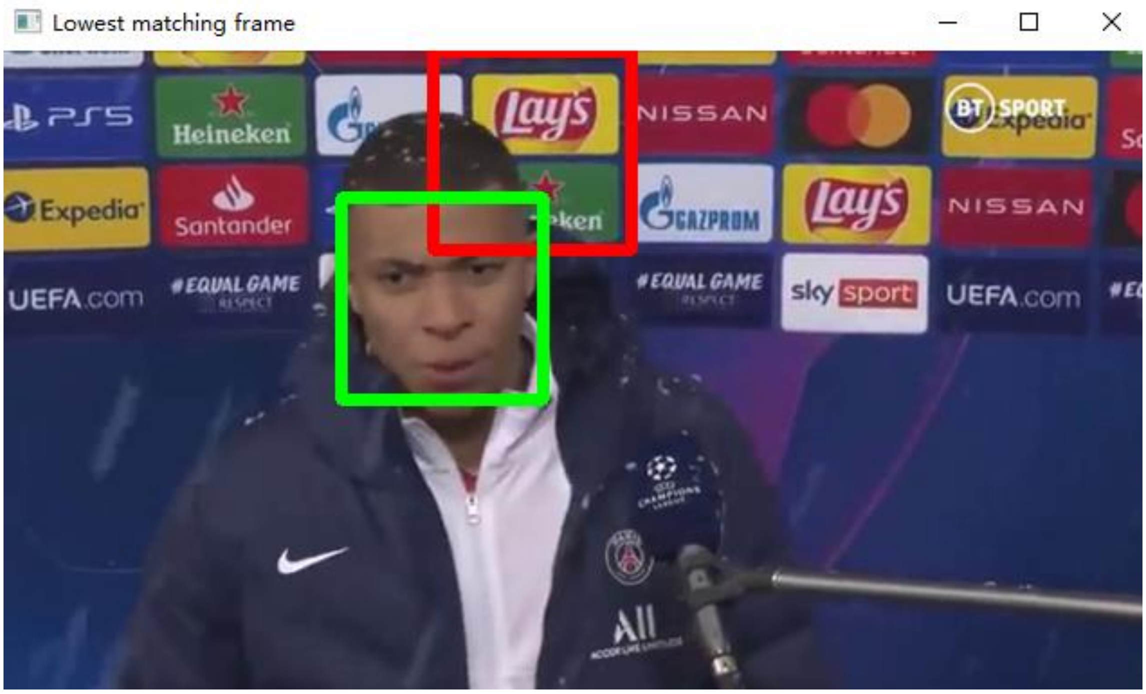
Q5.2

For this question, I use 0.05 * max magnitude as the lower threshold in computing the mask because 0.05 works the best experimentally. The plot looks like this.



This is the frame where they have the highest match. Green box represents face detector. Red box represent mean shift tracking

This is frame where they have the lowest match



31.25% of the frames in which iou is larger than 0.5