CSC 420 A4

Yinjun Zheng

Dec, 2021

1 Part 1

1.1 Q1

1.
We know that:

$$\vec{p} = \begin{pmatrix} wx \\ wy \\ wz \end{pmatrix} = K\vec{P} = \begin{pmatrix} X_0 + td_x \\ Y_0 + td_y \\ Z_0 + td_z \end{pmatrix}$$

So we get:

$$\begin{aligned} \begin{pmatrix} wx \\ wy \\ wz \end{pmatrix} &= \begin{pmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_0 + td_x \\ Y_0 + td_y \\ Z_0 + td_z \end{pmatrix} \\ &= \begin{pmatrix} fX_0 + ftd_x + p_xZ_0 + p_xtd_z \\ fY_0 + ftd_y + p_yZ_0 + p_ytd_z \\ Z_0 + td_z \end{pmatrix} \end{aligned}$$

By removing the w on the left side, we get x,y,as follows:

$$x = \frac{fX_0 + ftd_x + p_xZ_0 + p_xtd_z}{Z_0 + td_z}$$
$$y = \frac{fY_0 + ftd_y + p_yZ_0 + p_ytd_z}{Z_0 + td_z}$$

To get the vanishing points, take the limit on t:

$$\lim_{t\to\infty} x = \frac{fd_x + p_x d_z}{d_z} = \frac{fd_x}{dz} + px$$
$$\lim_{t\to\infty} y = \frac{fd_y + p_x d_z}{d_z} = \frac{fd_y}{dz} + py$$

Therefore, the vanishing point is $(\frac{fd_x}{dz} + px, \frac{fd_y}{dz} + py)$

2.
We have:

$$\vec{n}\vec{d} = 0$$

That is:

$$n_x d_x + n_y d_y = n_z d_z = 0$$
$$d_x = -\frac{1}{n_x}(n_y d_y + n_z d_z)$$

Therefore, we can find the line for vanishing point as follows:

$$\begin{aligned}
\frac{fd_x}{dz} + px &= -\frac{f}{n_x d_z}(n_y d_y + n_z d_z) + p_x \\
&= -\frac{fn_y d_y}{n_x d_z} - \frac{fn_z d_z}{n_x d_z} + p_x \\
&= -\frac{fd_y}{d_z}\frac{n_y}{n_x} - p_y\frac{n_y}{n_x} + p_y\frac{n_y}{n_x} - f\frac{n_z}{n_x} + p_x \\
&= -\frac{n_y}{n_x}(\frac{fd_y}{d_z} + p_y) + p_y\frac{n_y}{n_x} - f\frac{n_z}{n_x} + p_x
\end{aligned}$$

Therefore, if we denote the vanishing point as (V_x, V_y), we get:

$$V_x = -\frac{n_y}{n_x}V_y + p_y\frac{n_y}{n_x} - f\frac{n_z}{n_x} + p_x$$

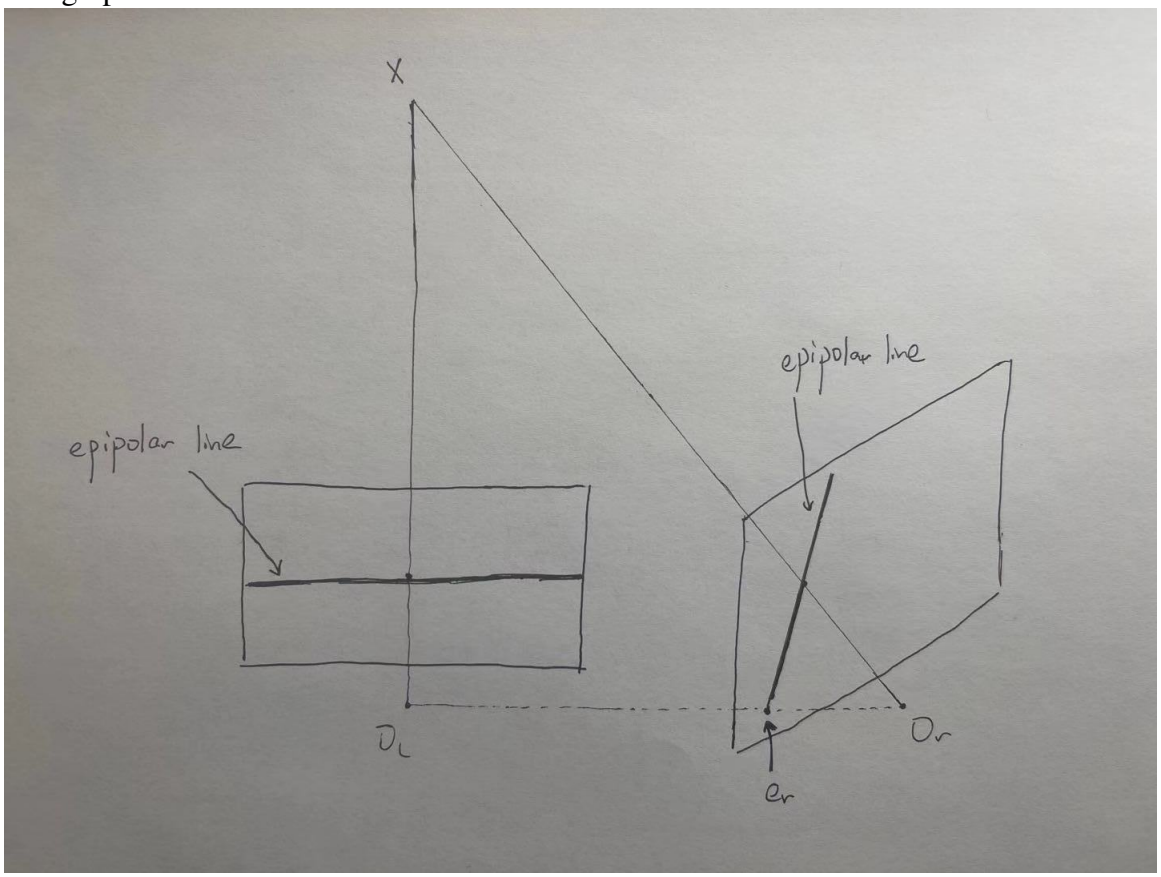
Therefore, the vanishing points are on the line:

$$n_x V_x + n_y V_y = p_y n_y + p_x n_x - f n_z$$

Note that n and p are all constants, so the vanishing points form a line.

1.2 Q2

The graph is shown as follows:



Explanation:

For the left camera, since the image plane is horizon, so the epihole is at infinity. The epipolar line passes the epihole at infinity, so the epipolar line is also horizon.

For the right camera, the epihole e_r is the intersection of line $O_l O_r$ and the right image plane. Assume the 3D object is at position X in the graph. The line that passes the center O_r and X intersects the right image plane at position. Then the epipolar line is the line that passes e_r and Y.

1.3 Q3

1.
Let the two lines be:

$$\begin{aligned} l : a_1x + b_1y + c_1 &= 0 \\ l' : a_2x + b_2y + c_2 &= 0 \end{aligned}$$

The intersection point for l and l' is:

$$\begin{aligned} x &= \frac{b_1c_2 - b_2c_1}{a_1b_2 - a_2b_1} \\ y &= \frac{a_2c_1 - a_1c_2}{a_1b_2 - a_2b_1} \\ (x, y) &= (\frac{b_1c_2 - b_2c_1}{a_1b_2 - a_2b_1}, \frac{a_2c_1 - a_1c_2}{a_1b_2 - a_2b_1}) \end{aligned}$$

Also, we get p as follows:

$$\begin{aligned} p &= l \times l' \\ &= (a_1, b_1, c_1)x(a_2, b_2, c_2) \\ &= (b_1c_2 - b_2c_1, a_2c_1 - a_1c_2, a_1b_2 - a_2b_1) \\ &\approx (\frac{b_1c_2 - b_2c_1}{a_1b_2 - a_2b_1}, \frac{a_2c_1 - a_1c_2}{a_1b_2 - a_2b_1}, 1) \\ &= (x, y, 1) \end{aligned}$$

Therefore, we get the intersection point of l and l' is exactly $p = l \times l'$.

2.
Let the two points be:

$$\begin{aligned} p &= (x_1, y_1) \\ p' &= (x_2, y_2) \end{aligned}$$

The line passing the two points p and p' is:

$$l : y = \frac{y_1 - y_2}{x_1 - x_2}x + \frac{x_1y_2 - x_2y_1}{x_1 - x_2}$$

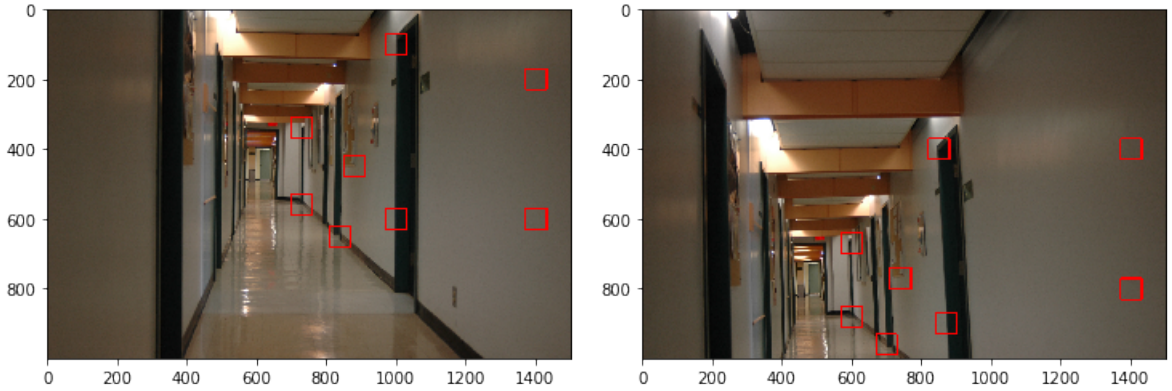
We also get the $l = p \times p'$ as follows:

$$\begin{aligned} p \times p' &= (x_1, y_1, 1) \times (x_2, y_2, 1) \\ &= (y_1 - y_2, x_2 - x_1, x_1 y_2 - x_2 y_1) \\ &\approx (\frac{y_1 - y_2}{x_1 - x_2}, \frac{x_1 y_2 - x_2 y_1}{x_1 - x_2}, 1) \end{aligned}$$

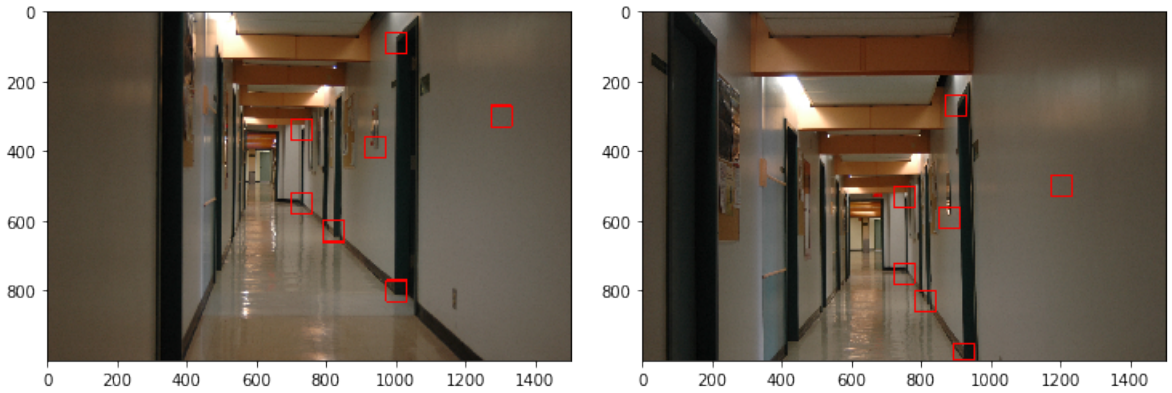
We get the same slope and intersect as above.
Therefore, the line passes p and p' is exactly $l = p \times p'$

1.4 Q4

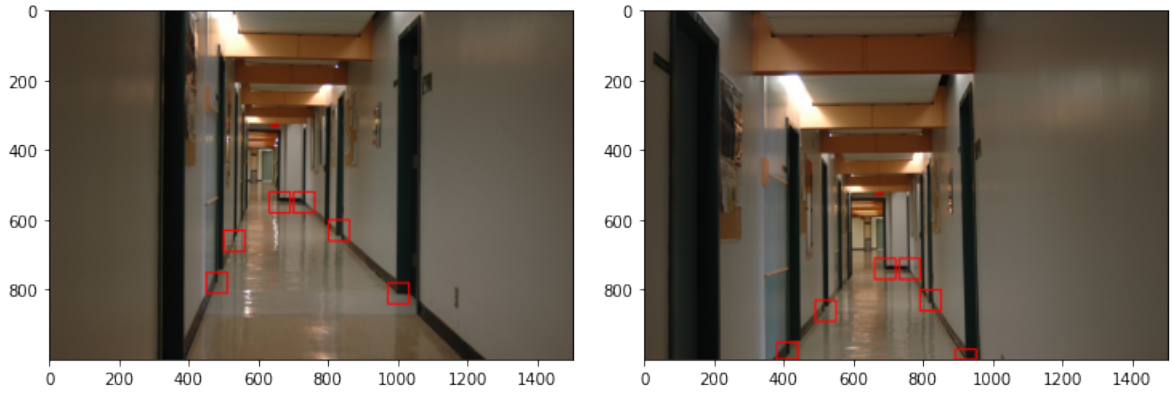
1.
For each case, I select 6 to 8 pairs of points. The selected pairs of points are shown as follows:
(1) Case A:



(2) Case B:



(3) Case C:



2.

The homography matrix H for each case are shown as follows:

(1) Case A:

$$\text{H} = \begin{pmatrix} -9.43439137e-04 & 3.56097706e-05 & -1.41730860e-01 \\ 5.88455422e-04 & -1.13663722e-03 & -9.89902394e-01 \\ 4.95429668e-07 & 6.37406871e-08 & -1.75617886e-03 \end{pmatrix}$$

Effect: Rotate the right wall of image 1. It is like rotating the right wall of image in counterclockwise direction.

(2) Case B:

$$\text{H} = \begin{pmatrix} 1.53015791e-05 & 1.92549227e-04 & 8.11243533e-01 \\ -4.37961752e-04 & 1.14788727e-03 & 5.84704691e-01 \\ -7.61735096e-07 & 1.90378864e-07 & 1.67542408e-03 \end{pmatrix}$$

Effect: It serves the functionality as scaling. Image 1 is taking a closer and bigger view of the right wall compared with image 3.

(3) Case C:

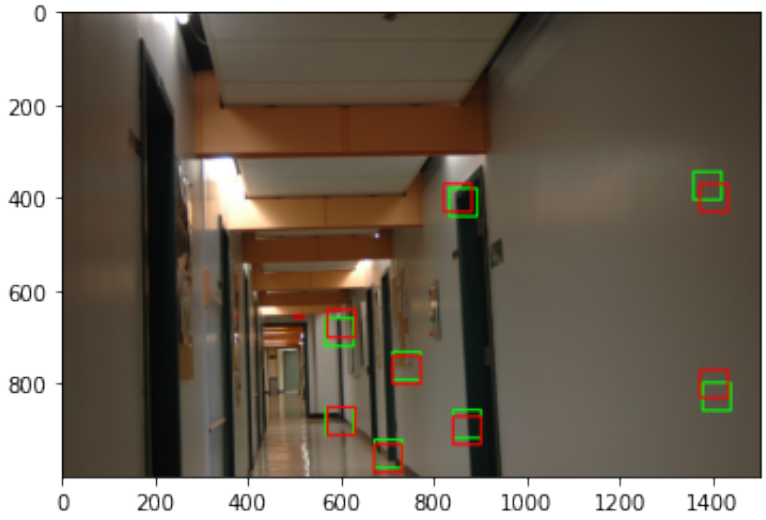
$$\text{H} = \begin{pmatrix} 6.09734251e-04 & -7.91449712e-04 & 7.18602938e-01 \\ -3.99305718e-04 & 5.63971132e-04 & 6.95417974e-01 \\ -3.97056012e-07 & -4.12709307e-07 & 1.47771243e-03 \end{pmatrix}$$

Effect: Shear the floor of image 1 to image 3. It it like keeping the position of the upper bound of the floor unchanged, while shifting the bottom bound from right to left.

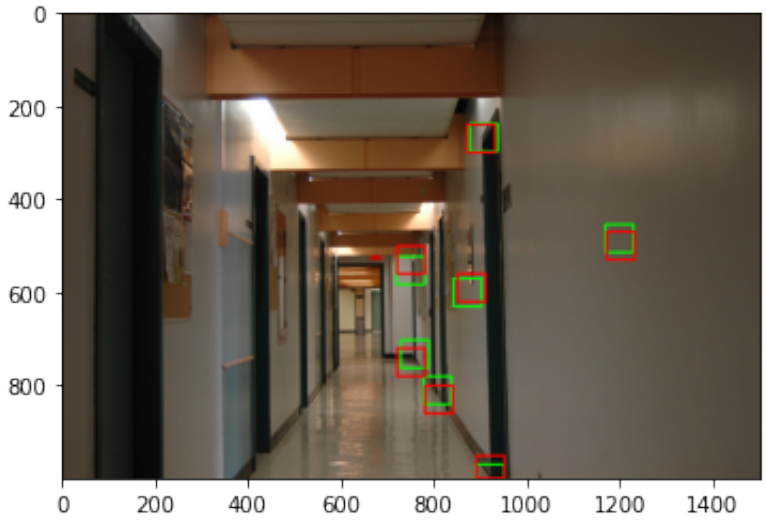
3.

The projection mapped points are shown as follows:

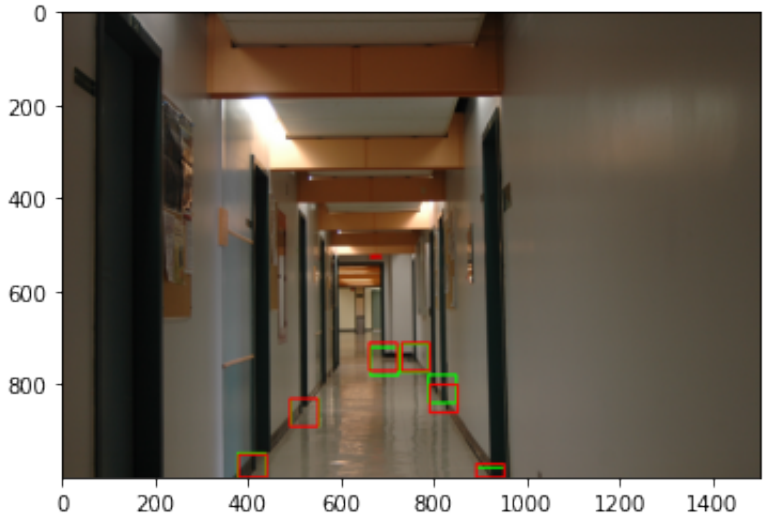
(1) Case A:



(2) Case B:



(3) Case C:



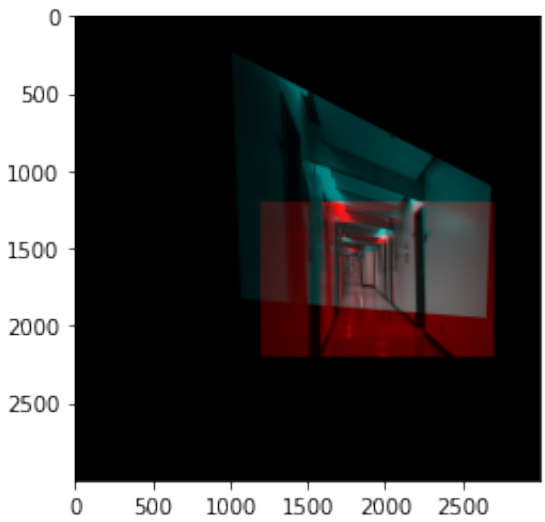
The code for these parts are shown as follows:

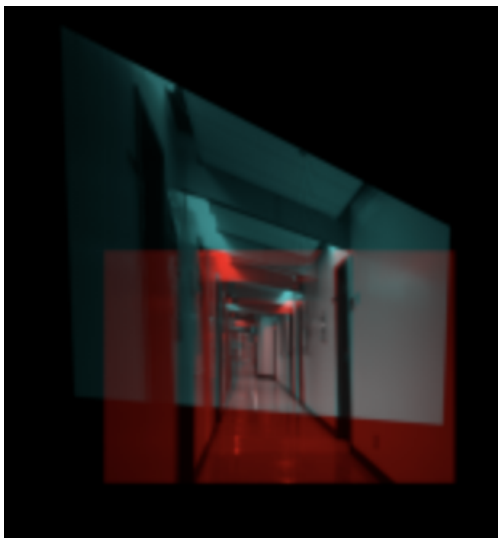
```
def homography(choose):
    img1_points, img2_points, img1_points_B, img3_points_B, img1_floor, img3_floor = choose_points()
    project_image = {"A": img2, "B": img3, "C":img3}
    if choose == "A":
        A = make_projection_matrix(img1_points, img2_points)
        H = get_H(A)[1]
        H = H.reshape(3,3)
        X_project = []
        for i in range(len(img1_points)):
            points = img1_points[i]
            proj_point = proj_points(H, points[0], points[1])
            X_project.append([proj_point[0], proj_point[1]])
    elif choose == "B":
        A = make_projection_matrix(img1_points_B, img3_points_B)
        H = get_H(A)[1]
        H = H.reshape(3,3)
        X_project = []
        for i in range(len(img1_points_B)):
            points = img1_points_B[i]
            proj_point = proj_points(H, points[0], points[1])
            X_project.append([proj_point[0], proj_point[1]])
    elif choose == "C":
        A = make_projection_matrix(img1_floor, img3_floor)
        H = get_H(A)[1]
        H = H.reshape(3,3)
        X_project = []
        for i in range(len(img1_floor)):
            points = img1_floor[i]
            proj_point = proj_points(H, points[0], points[1])
            X_project.append([proj_point[0], proj_point[1]])
```

4.

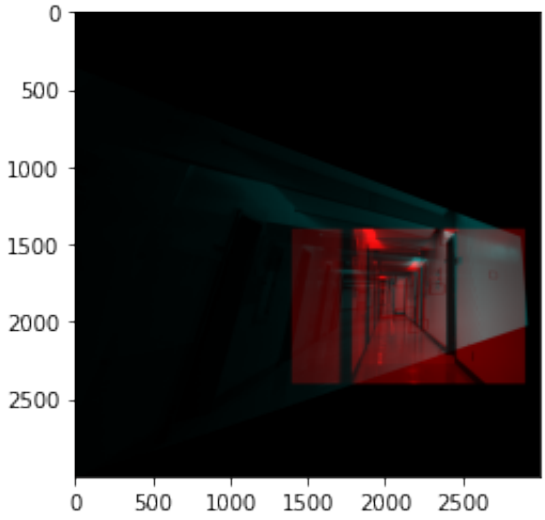
The constructed new image for each case are shown as follows:

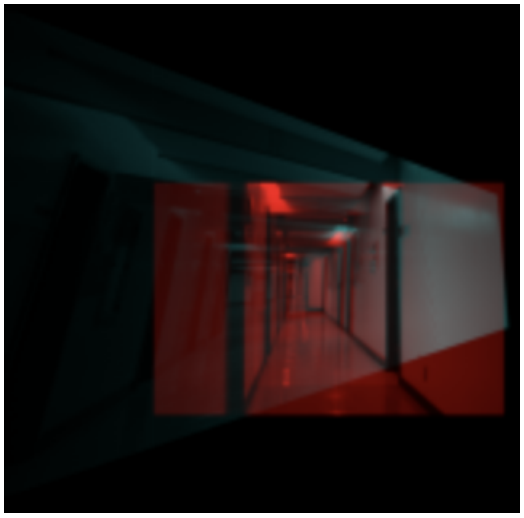
(1) Case A:



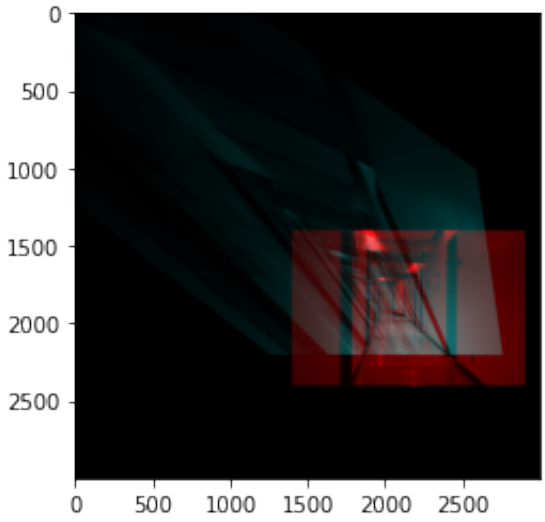


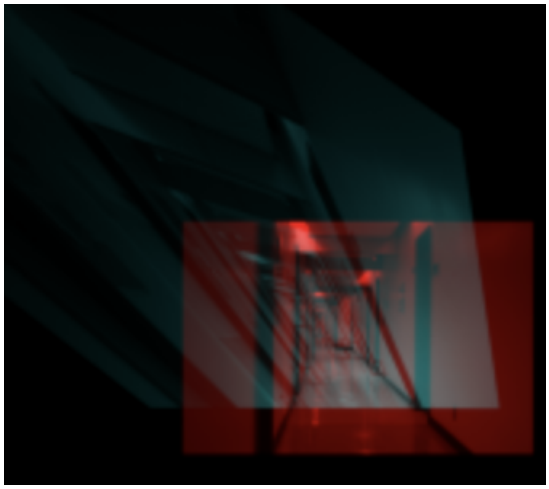
(2) Case B:





(3) Case C:





Conclusion:

The 3D relative position and the orientation of camera are using the same coordination system. We can use the homography matrix H to map the whole image into another plane, and use the inverse of H to map the image back. In this way, we will get the mapped-back point same as the original point. Based on the image above, we notice that both the right wall and the floor are planes in the image. If we map the surface plane with homography, and project back, we will get the same surface plane. This observation follows Lambertian Reflectance.

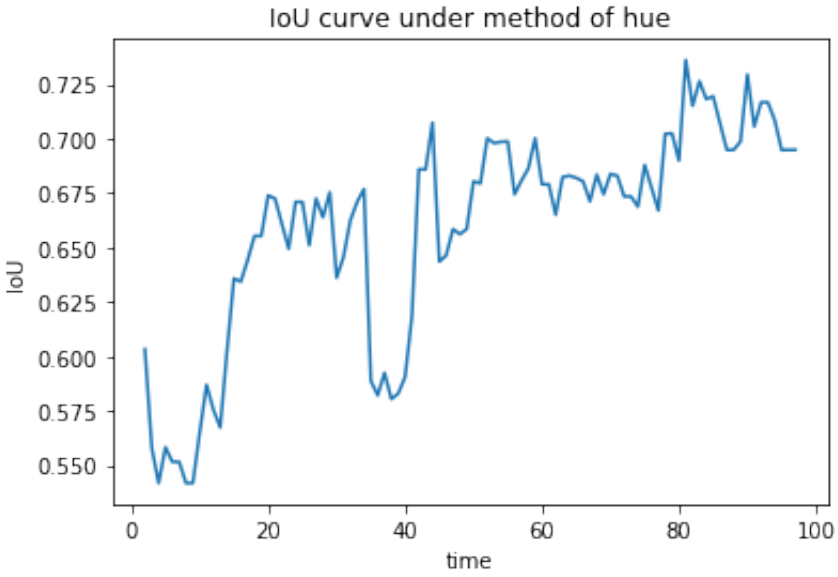
The code for part 4 is shown as follows:

```
def homography_whole_image(H, whole_img):
    m,n = whole_img.shape[0], whole_img.shape[1]
    X_project = []
    for i in range(m):
        for j in range(n):
            points = np.array([i,j])
            proj_point = proj_points(H, points[0], points[1])
            X_project.append([proj_point[0], proj_point[1]])
    return X_project


def homography_back(H, whole_img):
    H_inv = inv(H)
    m,n = whole_img.shape[0], whole_img.shape[1]
    X_project = []
    for i in range(m):
        for j in range(n):
            proj_point = homography_back_point(H_inv, j, i)
            X_project.append((proj_point[0], proj_point[1], whole_img[i][j]))
    return X_project

def homography_back_point(H_inv, x,y):
    proj_point = proj_points(H_inv, x, y)
    return proj_point
```

```
def construct_final_img(H, img_1, img_2):
    m,n = img_1.shape[0], img_1.shape[1]
    img_1 = cv2.cvtColor(img_1, cv2.COLOR_BGR2GRAY)
    img_2 = cv2.cvtColor(img_2, cv2.COLOR_BGR2GRAY)

    shift = 1400
    final_img = np.zeros((3000, 3000, 3))
    final_img[shift:shift+m, shift:shift+n,0] = img_1/255

    X_project = homography_back(H, img_2)

    for items in X_project:
        x_back, y_back,color = int(items[0]), int(items[1]), items[2]
        shift_y = y_back + shift
        shift_x = x_back + shift
        final_img[shift_y, shift_x, 1] = color/255
        final_img[shift_y, shift_x, 2] = color/255
    plt.imshow(final_img)
    return final_img

H = H_99
construct_final_img(H, img1, img3)
```

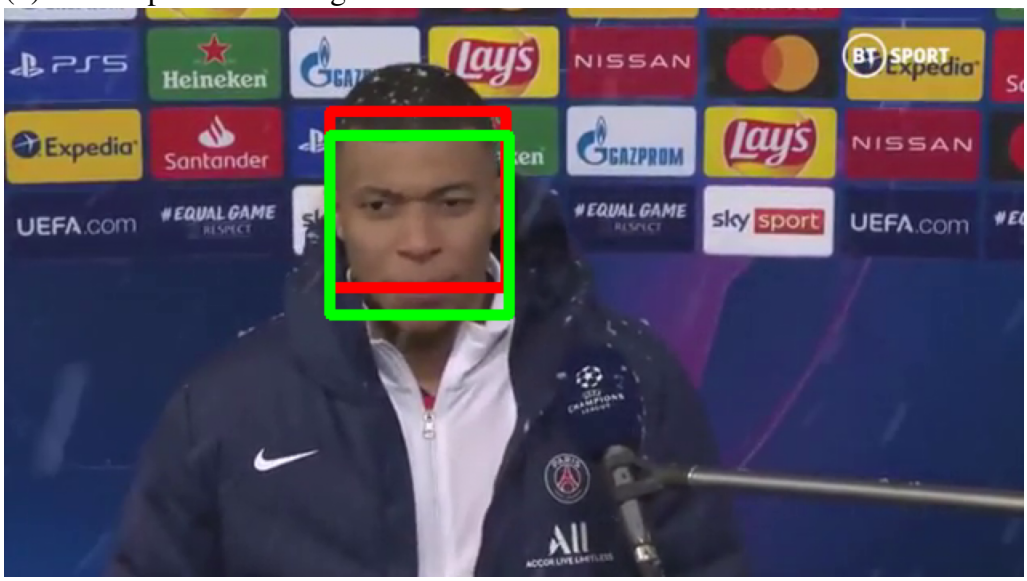
1.5 Q5

1.Use hue histogram:
The IoU changes through time as follows:



As we can see, the IoU value is always larger than 0.5, so the results of these two detection method have a large overlap. That is both method is doing a great job.

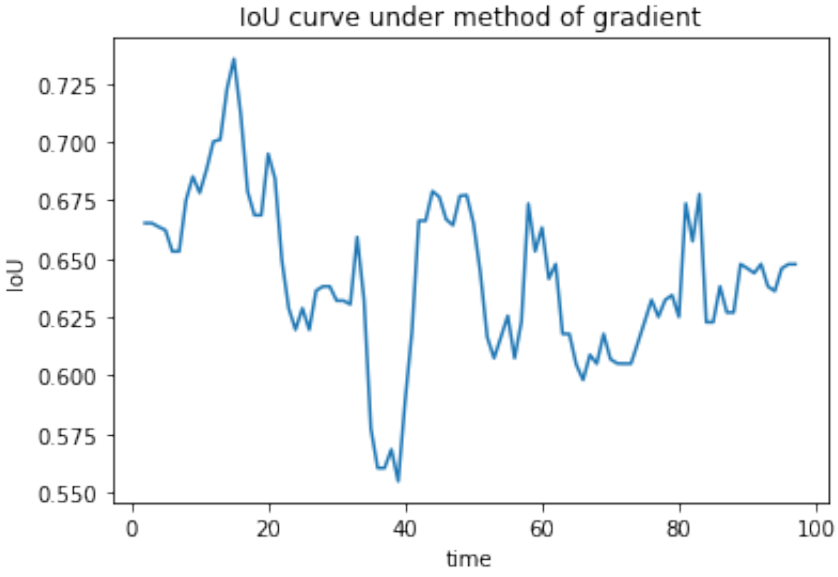
Here we take 0.7 as the upper threshold, and 0.55 as the lower threshold.
We take the red frame as Viola-Jones box, and green square as tracked box.
(1) the sample for IoU larger than 0.7 is shown as follows:



(2) the sample for IoU lower than 0.55 is shown as follows:

(3) The percentage of images that have IoU greater than 0.7 is 0.16.
The percentage of images that have IoU greater than 0.6 is 0.81.
The percentage of images that have IoU greater than 0.5 is 1.
(4) Explanation: Look at the sample image that has low IoU above. The green square (tracked box) looks better. It is probably because the face is looking at right side, which influences the detection by Viola-Jones method.

2. Use method of gradient:
The IoU changes through time as follows:



As we can see, most of the IoU values are larger than 0.5, so the results of these two detection method have a large overlap. That is both method is doing a great job.

Here we take 0.7 as the upper threshold, and 0.57 as the lower threshold.
We take the red frame as Viola-Jones box, and green square as tracked box.
(1) the sample for IoU larger than 0.7 is shown as follows:

(2) the sample for IoU lower than 0.57 is shown as follows:

(3) The percentage of images that have IoU greater than 0.5 is 1.
The percentage of images that have IoU greater than 0.7 is 0.06.

Please refer to the file Q5.ipynb for detailed code.