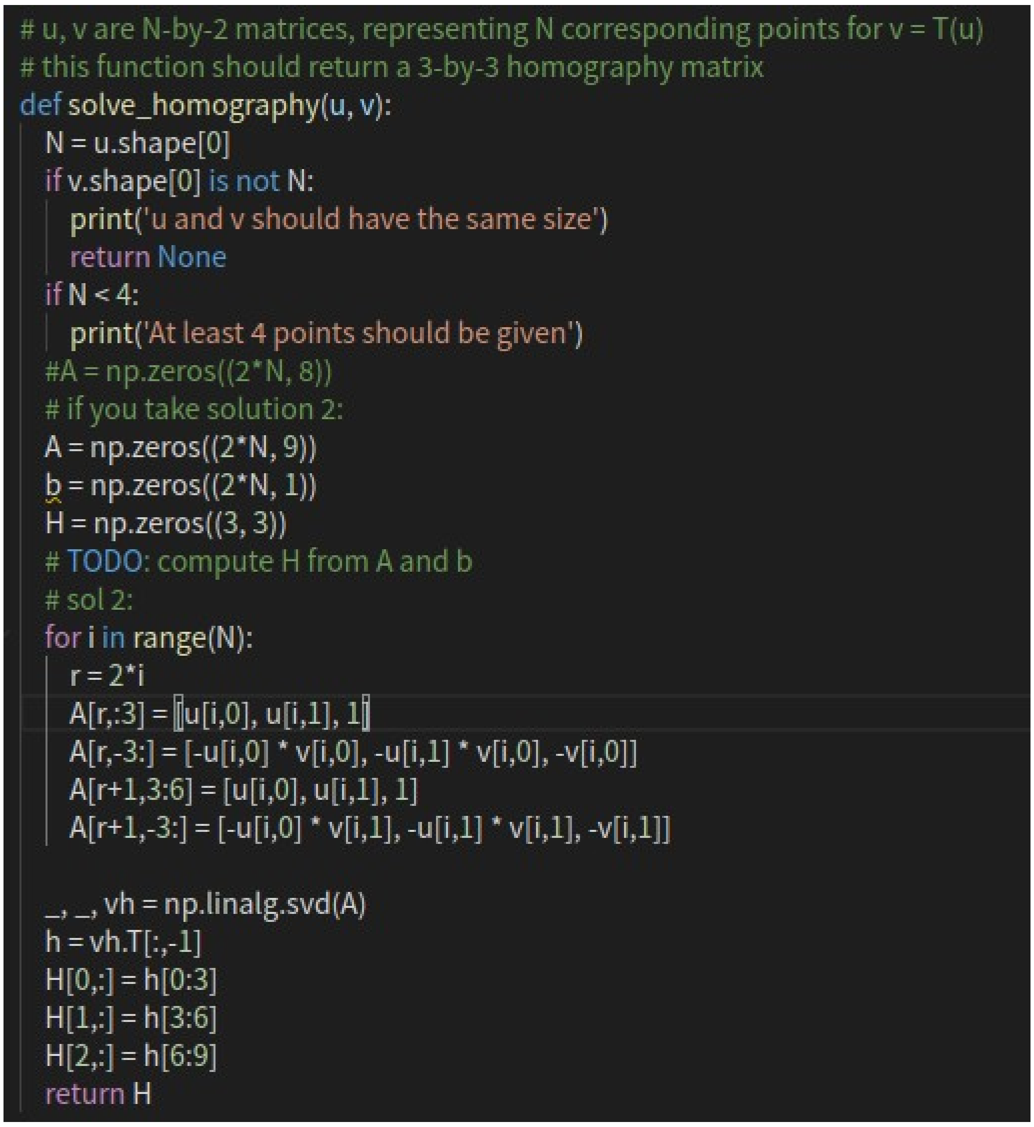
HW 3 Report

R08921040 徐均筑

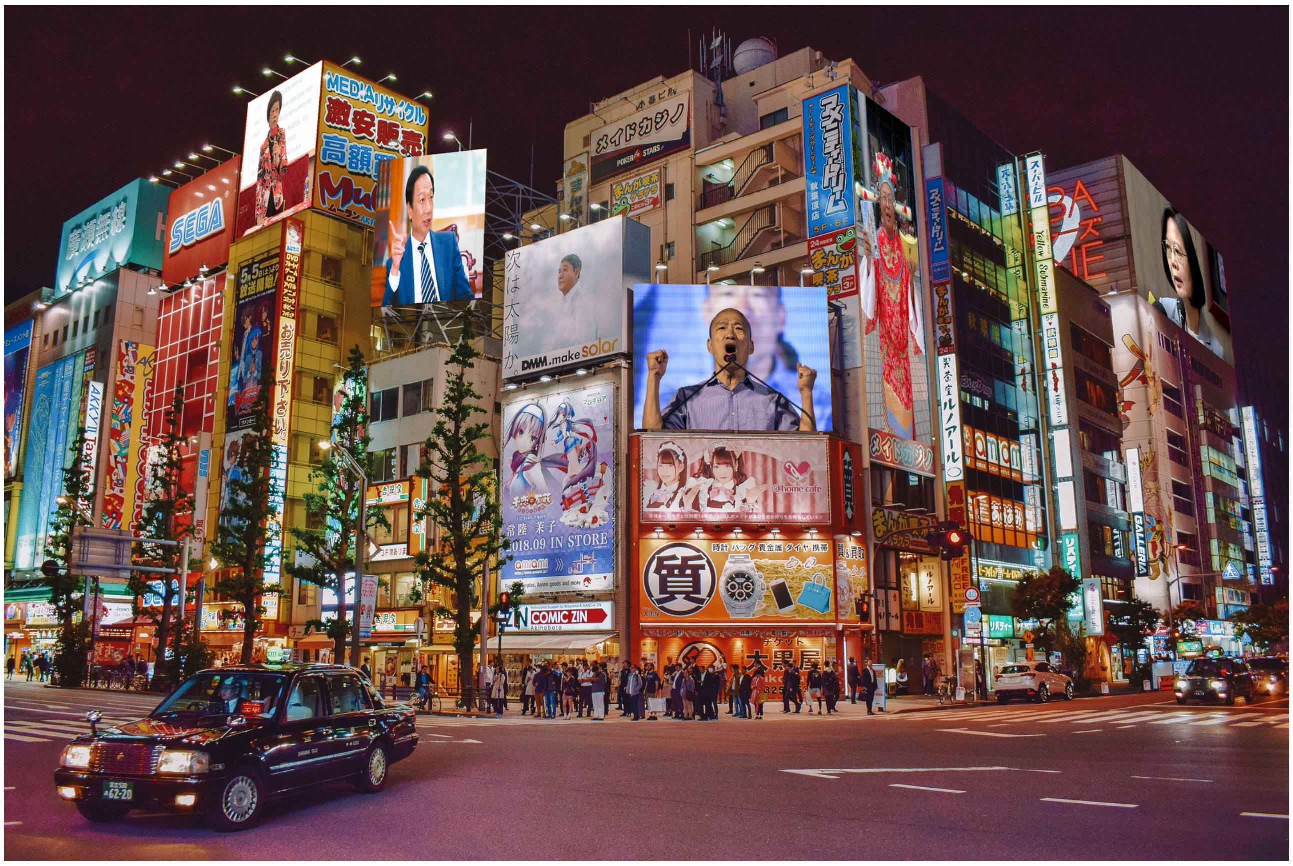
Part 1(4%)

• Include the function solve_homography(u, v) in your report.



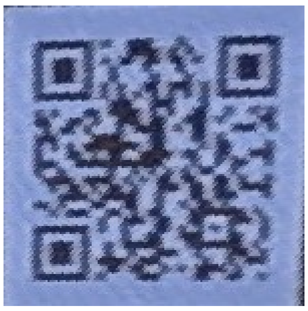
```python
# u, v are N-by-2 matrices, representing N corresponding points for v = T(u)
# this function should return a 3-by-3 homography matrix
def solve_homography(u, v):
    N = u.shape[0]
    if v.shape[0] is not N:
        print('u and v should have the same size')
        return None
    if N < 4:
        print('At least 4 points should be given')
    #A = np.zeros((2*N, 8))
    # if you take solution 2:
    A = np.zeros((2*N, 9))
    b = np.zeros((2*N, 1))
    H = np.zeros((3, 3))
    # TODO: compute H from A and b
    # sol 2:
    for i in range(N):
        r = 2*i
        A[r,:3] = [u[i,0], u[i,1], 1]
        A[r,-3:] = [-u[i,0] * v[i,0], -u[i,1] * v[i,0], -v[i,0]]
        A[r+1,3:6] = [u[i,0], u[i,1], 1]
        A[r+1,-3:] = [-u[i,0] * v[i,1], -u[i,1] * v[i,1], -v[i,1]]

    _, _, vh = np.linalg.svd(A)
    h = vh.T[:,-1]
    H[0,:] = h[0:3]
    H[1,:] = h[3:6]
    H[2,:] = h[6:9]
    return H
```

The output image:



Part 2(3%)

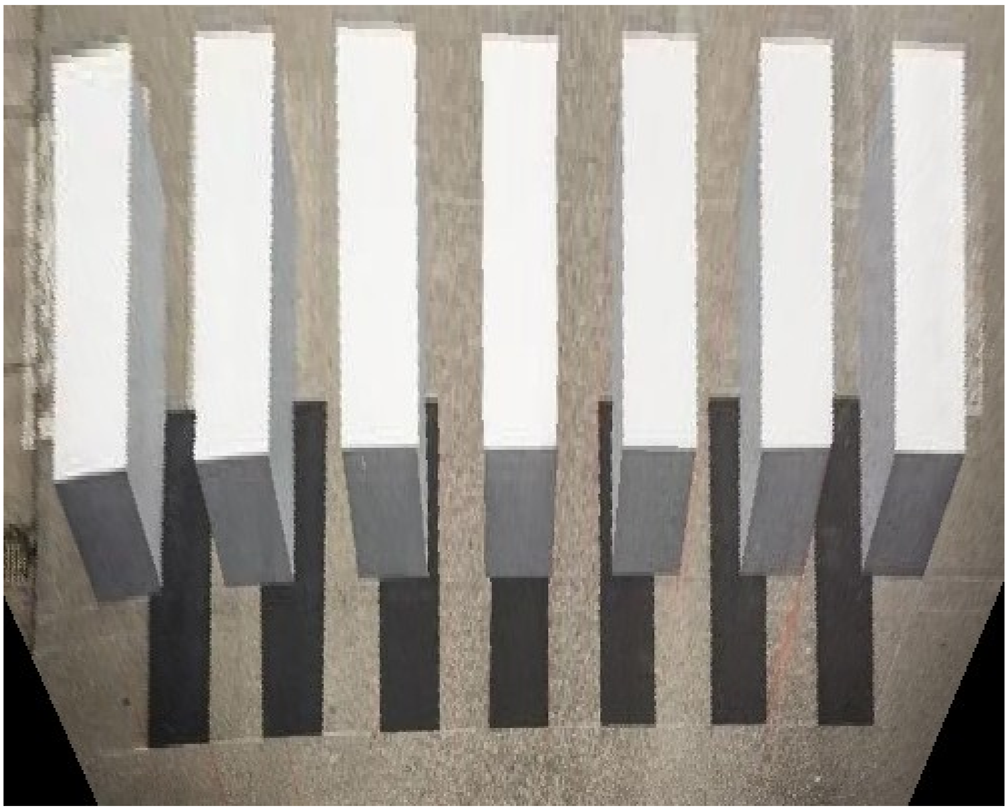
• Include the QR code and the decoded link in your report.



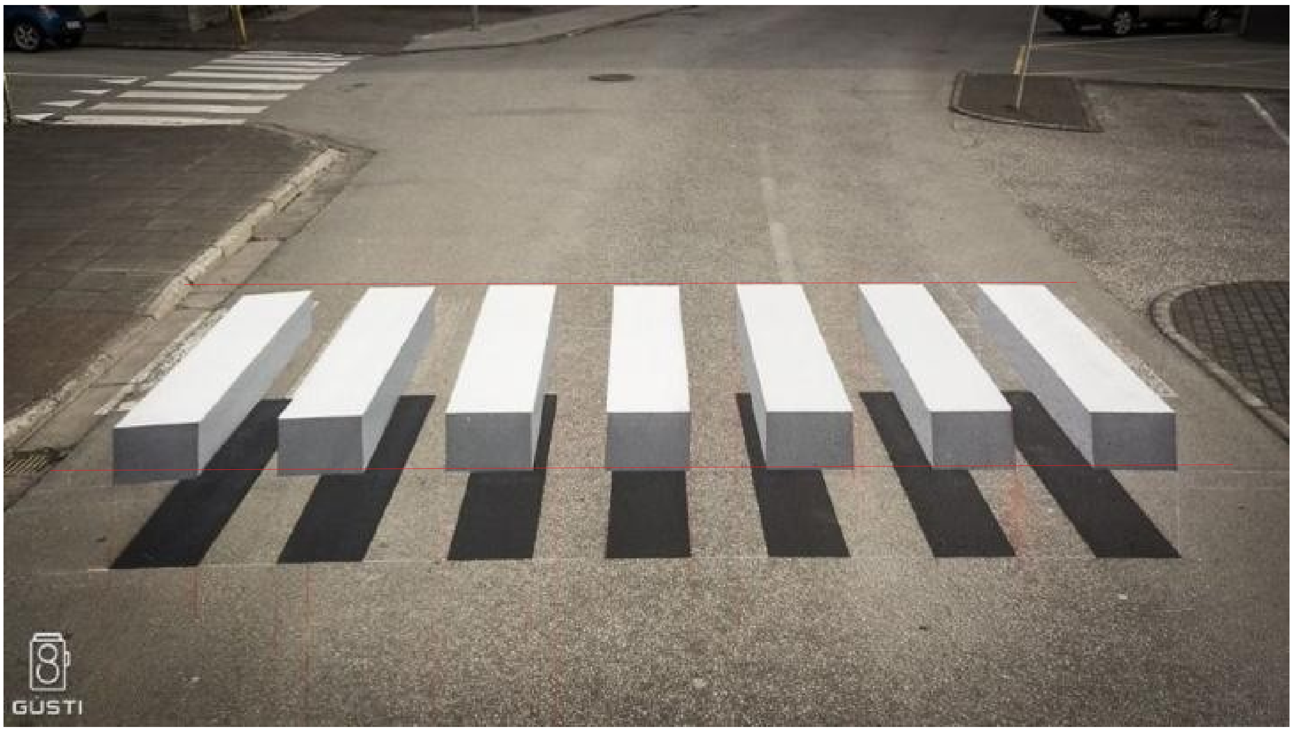
The decoded link: http://media.ee.ntu.edu.tw/courses/cv/19F/

Part 3(3%)

• Unwarp the image to the top view.



• Can you get the parallel bars from the top view?
• If not, why? Discuss in your report.



No. The top edges and the bottom edges of the bars are not aligned in the image, so it's impossible to transform them into parallel lines.

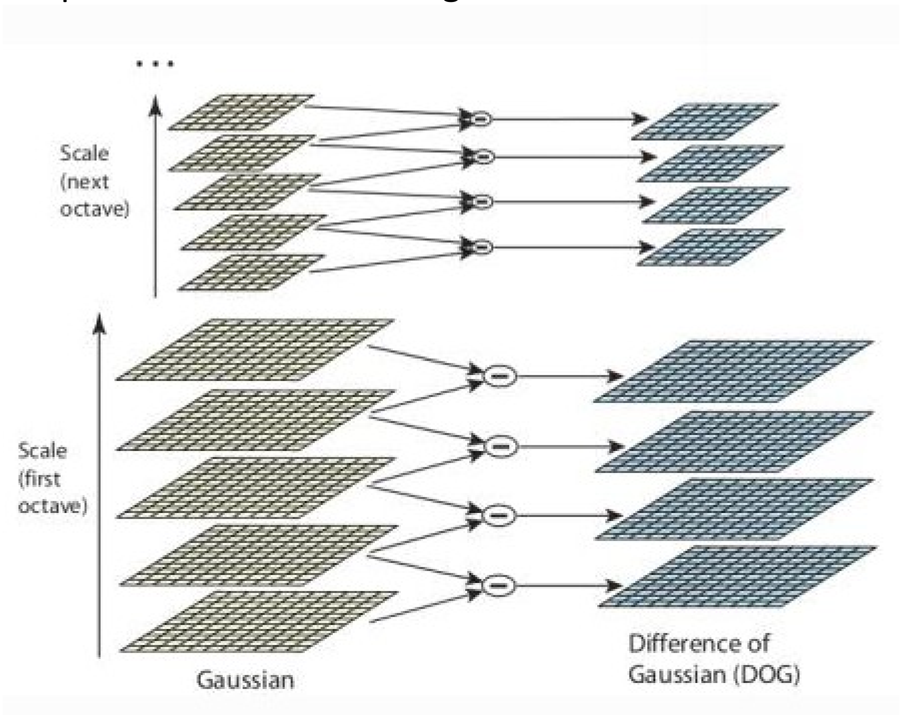
Part4(5%)
• Find the pose between the video frames and the template.

Algorithm:

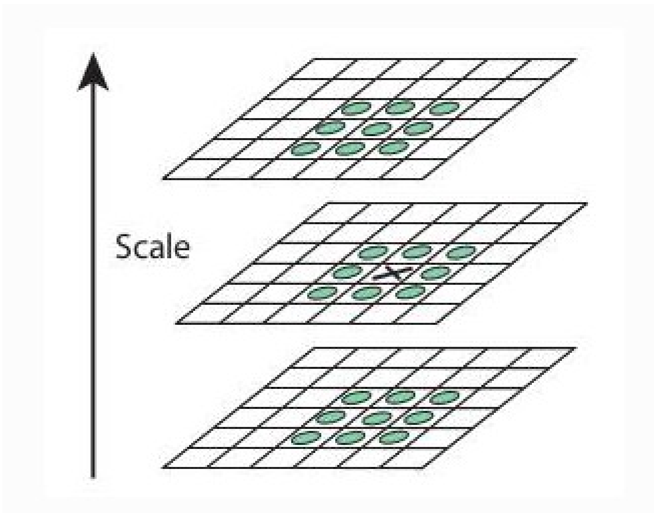
Use SIFT to feature points and find matching between the video frame and the template. Finally, we can get transformation matrix and map the desired image to the video.

SIFT:

SIFT algorithm uses Difference of Gaussians, which is obtained as the difference of Gaussian blurring of an image with two different σ, let it be σ and $k\sigma$. This process is done for different octaves of the image in Gaussian Pyramid. It is represented in below image:



Once this DoG are found, images are searched for local extrema over scale and space. For eg, one pixel in an image is compared with its 8 neighbours as well as 9 pixels in next scale and 9 pixels in previous scales. If it is a local extrema, it is a potential keypoint. It basically means that keypoint is best represented in that scale. It is shown in below image:



Once potential keypoints locations are found, they have to be refined to get more accurate results. They used Taylor series expansion of scale space to get more accurate location of extrema, and if the intensity at this extrema is less than a threshold value, it is rejected.

DoG has higher response for edges, so edges also need to be removed. For this, a concept similar to Harris corner detector is used. They used a 2x2 Hessian matrix (H) to compute the pricipal curvature. We know from Harris corner detector that for edges, one eigen value is larger than the other. So here they used a simple function, if this ratio is greater than a threshold, that keypoint is discarded.

Now keypoint descriptor is created. A 16x16 neighbourhood around the keypoint is taken. It is devided into 16 sub-blocks of 4x4 size. For each sub-block, 8 bin orientation histogram is created. So a total of 128 bin values are available. It is represented as a vector to form keypoint descriptor.

Keypoints between two images are matched by identifying their nearest neighbours. But in some cases, the second closest-match may be very near to the first. It may happen due to noise or some other reasons. In that case, ratio of closest-distance to second-closest distance is taken. If it is greater than 0.8, they are rejected.

Reference:
https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html

- Environment setting:

Python: 3.6
OpenCV2: 3.4.2