Computer Vision HW3

R08942065 楊宗山

- **Part 1**

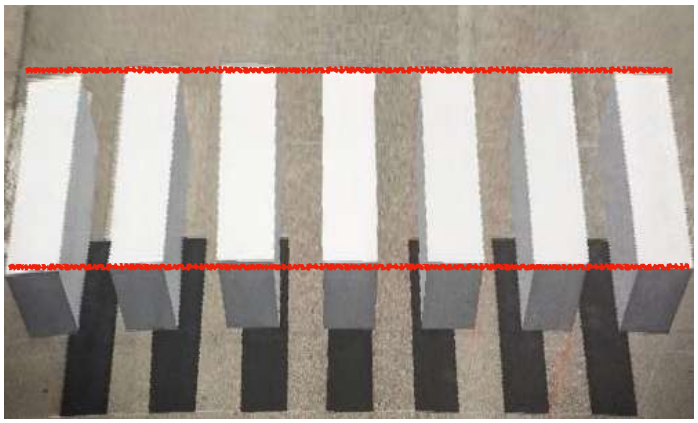
```python
def solve_homography(u, v):
    N = u.shape[0]
    if v.shape[0] is not N:
        print('u and v should have the same size')
        return None
    if N < 4:
        print('At least 4 points should be given')
    A = np.zeros((2*N, 9))
    b = np.zeros((2*N, 1))
    H = np.zeros((3, 3))
    u = np.concatenate([u, np.ones((N, 1))], axis=1)
    # Construct MatrixA
    for row in range(N):
        A[2*row, 0:3] = u[row]
        A[2*row+1, 3:6] = u[row]
        A[2*row:2*row+2, 6:9] = -np.matmul(v[row, np.newaxis].T, u[row, np.newaxis])
    U, _, _ = np.linalg.svd(np.matmul(A.T, A))
    H = U[:, -1].reshape((3, 3))
    return H
```

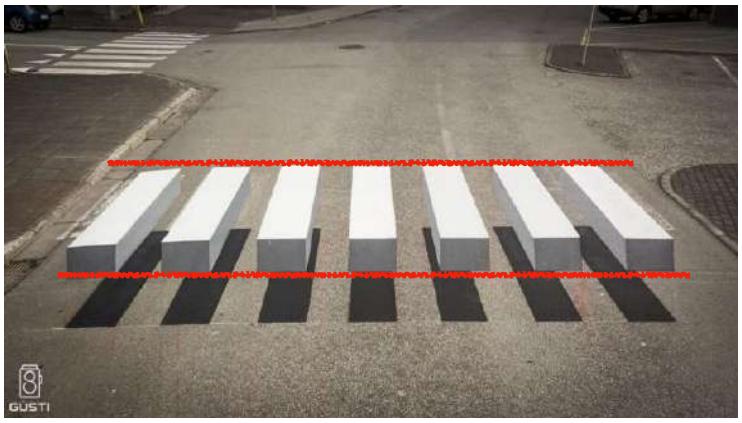
I follow the math type of homography transformation to get matrix H, and speed up the construction progress with `numpy` manipulation.

- **Part 2**

The link is `ee.ntu.edu.tw`

- **Part 3**





We can't project the front-view picture to match the top-view ground truth. As the labeled red lines, the leftmost block are not parallel to each other, so the reprojected picture won't match to the given top-view.

- **Algorithm to the simple AR**

(With OpenCV 3.4.2)

We can resize the `mark` to the same shape with the **`handsome guy`**, and the projection can be simplified to the two coordinate mapping problem.

There are three main step in `part4.py`:

Feature Matching

```
## check whethere the frame is legal, i.e., there still exists a frame
## Feature Matching
kp1, des1 = sift.detectAndCompute(template,None)
kp2, des2 = sift.detectAndCompute(frame,None)
matches = flann.knnMatch(des1,des2,k=2)
good = []
for m,n in matches:
    if m.distance < 0.85*n.distance:
        good.append(m)
```

I implement this part with `cv2.xfeatures2d.SIFT_create()` and `cv2.FlannBasedMatcher()`

Find Homography Matrix

```
## Homography
if len(good)>MIN_MATCH_COUNT:
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1, 1, 2)
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1, 1, 2)

    H, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
    matchesMask = mask.ravel().tolist()
```

I implement this part with `cv2.findHomography()` , and get the projection matrix.

Projection

```
    height, width = template.shape[:2]

    x_coor, y_coor = np.meshgrid(np.arange(width), np.arange(height).T)
    pad = np.ones(x_coor.shape)
    coor = np.stack((x_coor, y_coor, pad), axis=2)
    coor = coor.reshape((-1, 3))

    new_coor = np.matmul(H, coor.T)
    new_coor = np.stack([np.divide(new_coor[0, :], new_coor[2, :]), \
                         np.divide(new_coor[1, :], new_coor[2, :])], axis=1)
    new_coor = new_coor.astype(int)
    new_coor[new_coor[:, 0] >= film_w, 0] = film_w - 1
    new_coor[new_coor[:, 0] <= 0, 0] = 0
    new_coor[new_coor[:, 1] >= film_h, 1] = film_h - 1
    new_coor[new_coor[:, 0] <= 0, 0] = 0

    frame[new_coor[:, 1], new_coor[:, 0]] \
    = ref_image[y_coor.reshape(-1), x_coor.reshape(-1)]
```

Using refactorization trick for the x_coor and y_coor to accelerate the computation.
I only implement direct mapping, and limit the out-of-boundary index by the conditions marked in the red box.