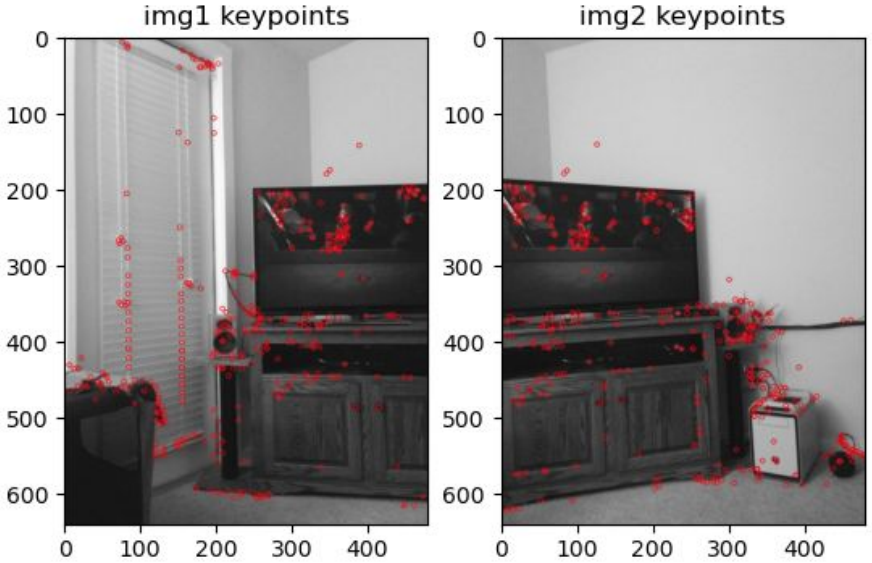
CV HW3 Report

Group 13

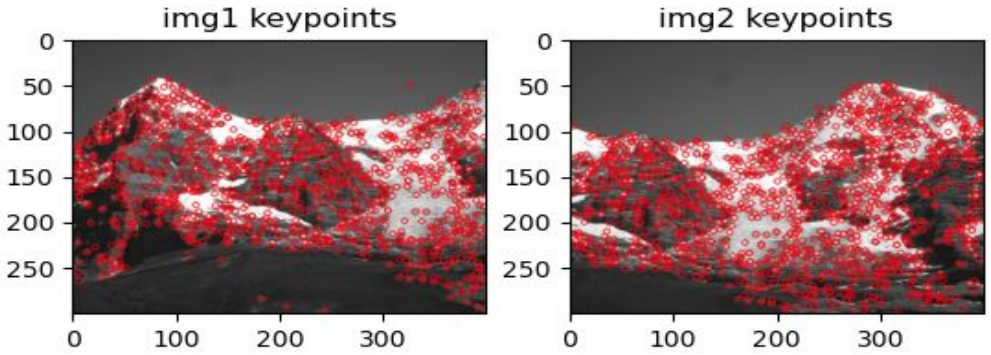
Interest points detection & feature description by SIFT

In the first, we convert the input images to grayscale images and use the opencv SIFT library to get the keypoints and feature descriptors

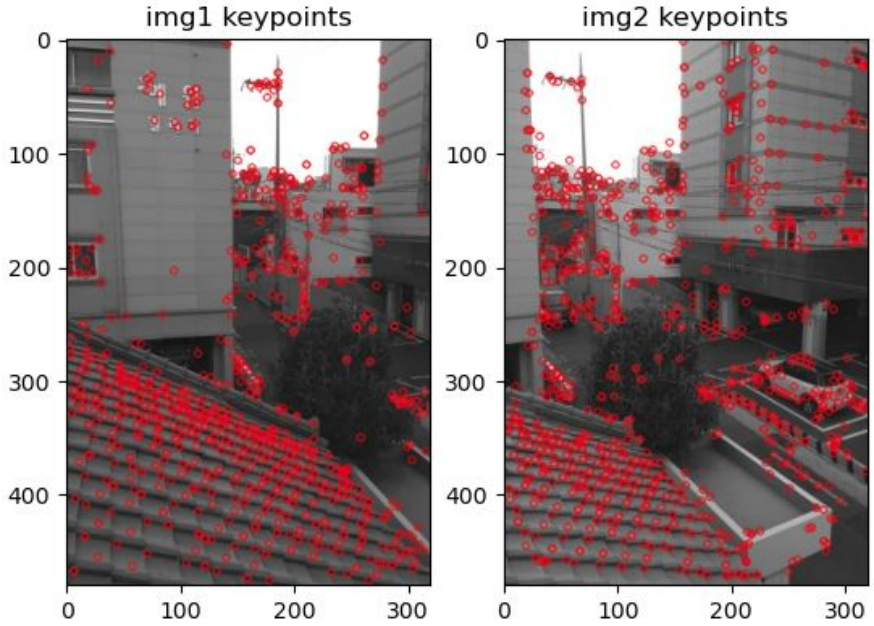
Interest points detection & feature description by SIFT



Interest points detection & feature description by SIFT



Interest points detection & feature description by SIFT

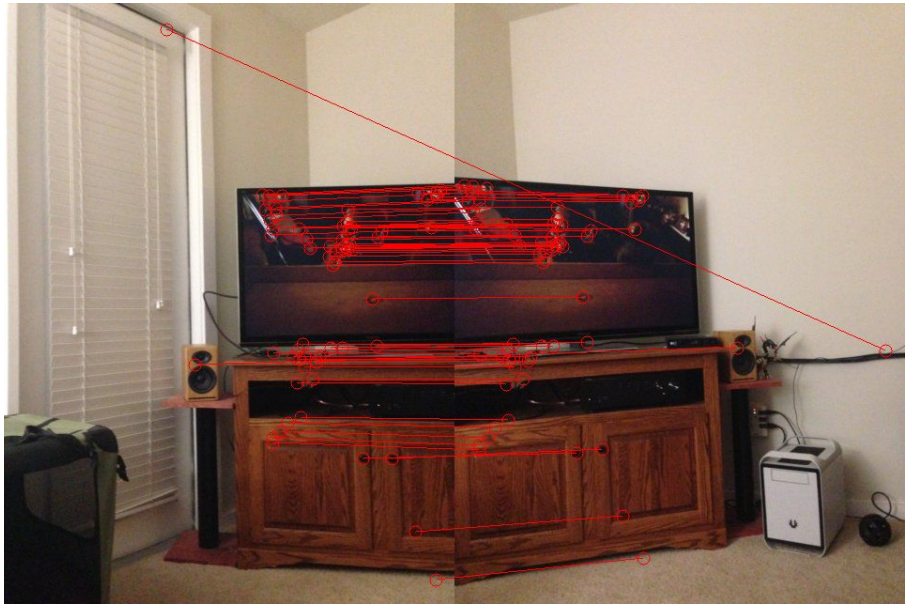


Feature matching by SIFT features

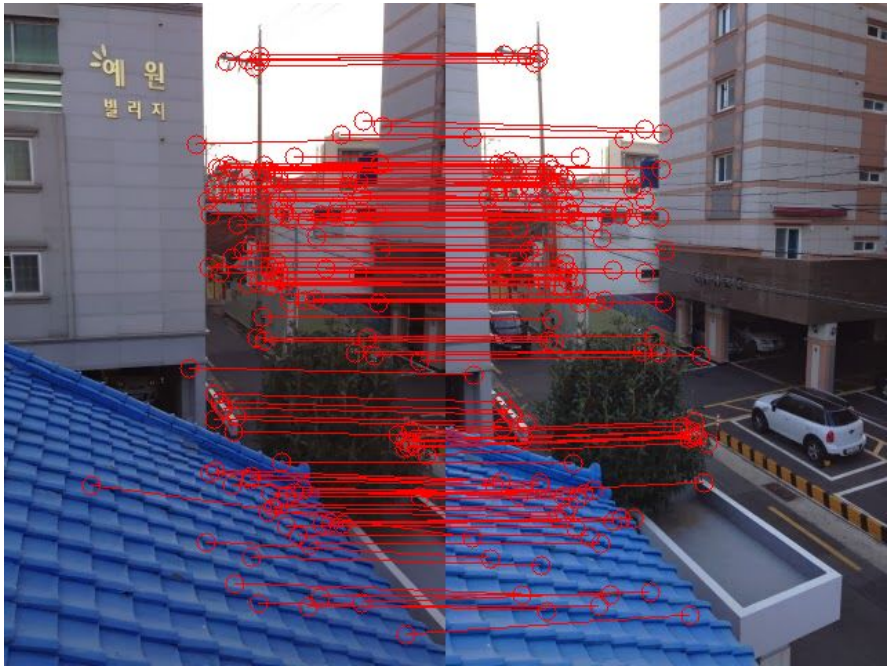
We use the ratio distance between the first nearest pair and the second nearest keypoint pair. If the ratio distance is smaller than the threshold, we put them to the matching list.

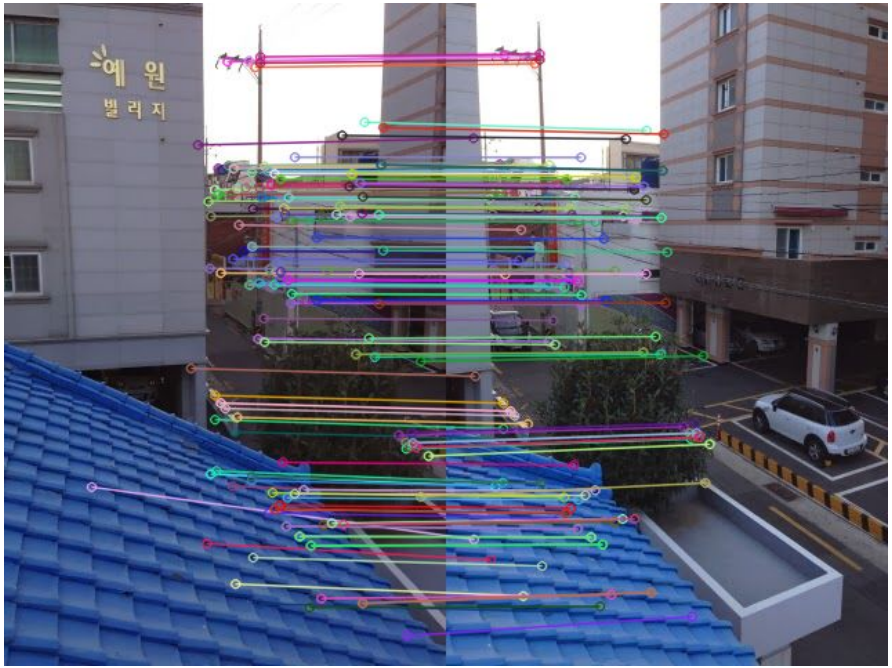
Next we will show the experiment result of feature matching and compare the result of opencv. The left image is our result and the right is the result of opencv

Feature matching by SIFT features

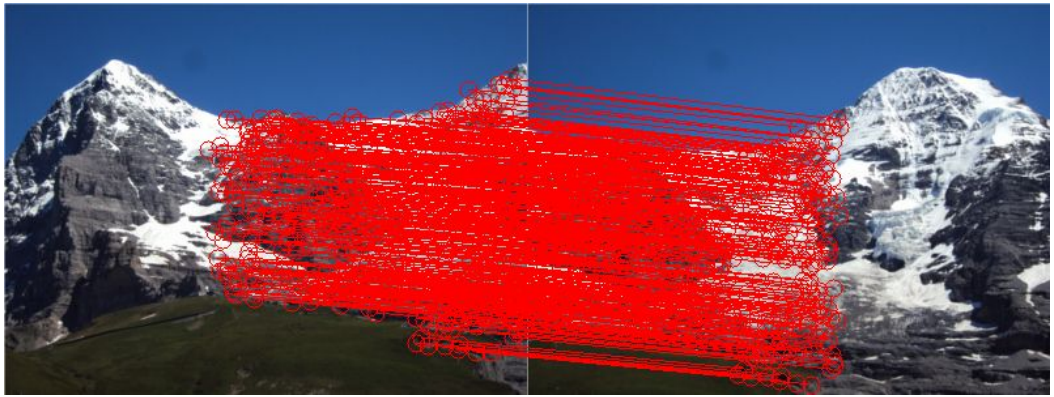


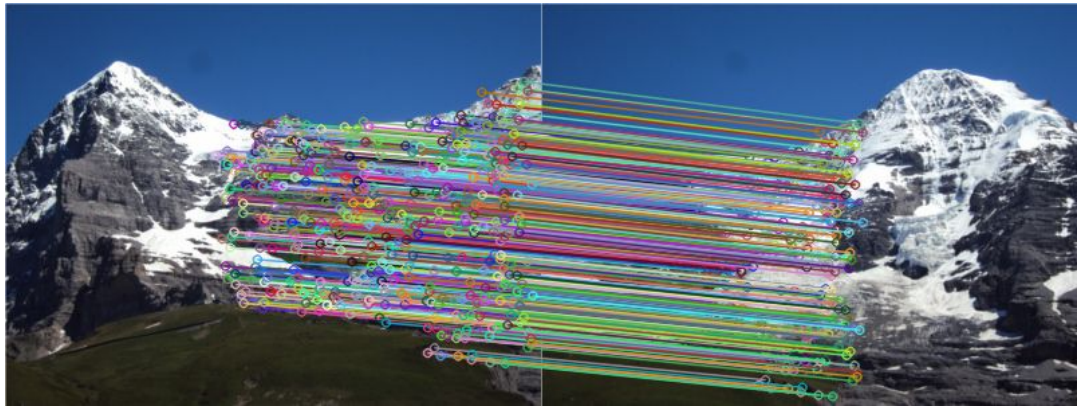
Feature matching by SIFT features





Feature matching by SIFT features





Introduction- RANSAC to find homography matrix H

It is a reliable homography estimation that uses RANSAC (a statistical method to suppress outliers).

In computer vision*, Homography* is a matrix that maps coordinates from one plane to the same plane that has been rotated or translated or transformed in any other way in space.

From motion estimation to create panoramic images, there are many applications of *Homography*.

Inplementation Procedure

1. We first use previous work to calculate SIFT key points.

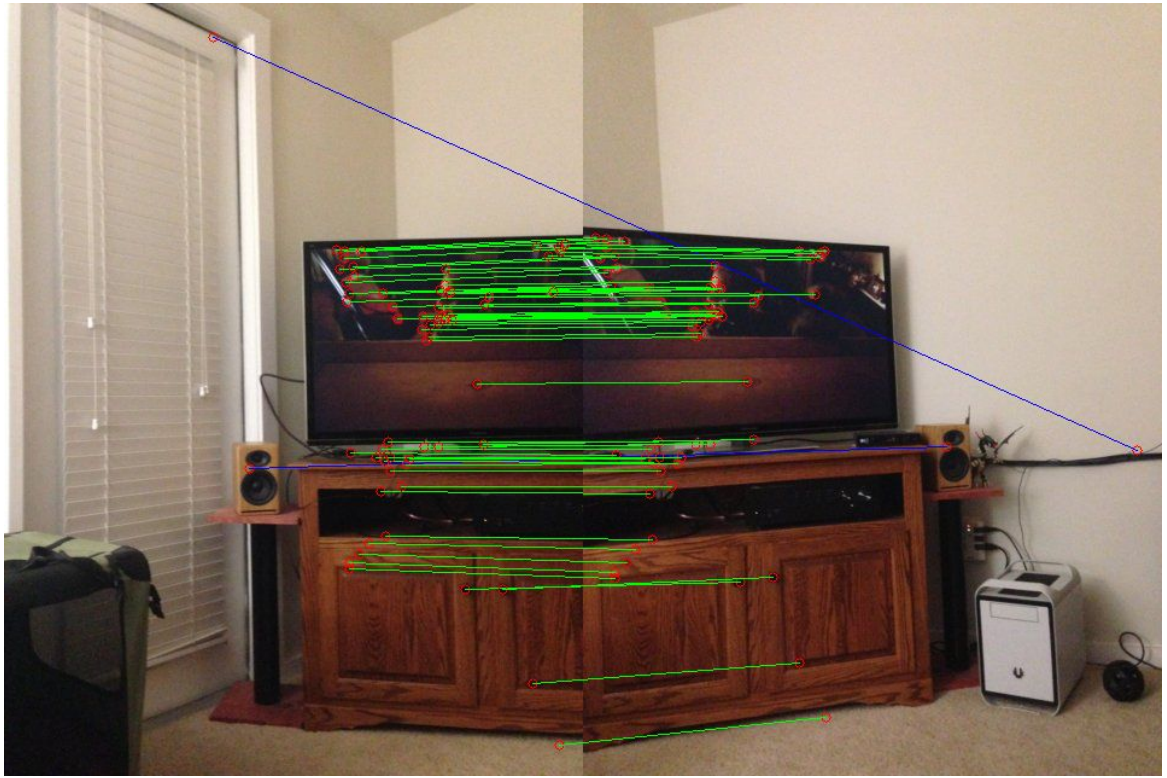
2. The four correspondences are passed into a function, which then calculates the homography.

3. RANSAC will select four random correspondences, calculate the homography, calculate the number of inside lines, and make the homography better than any found homography.

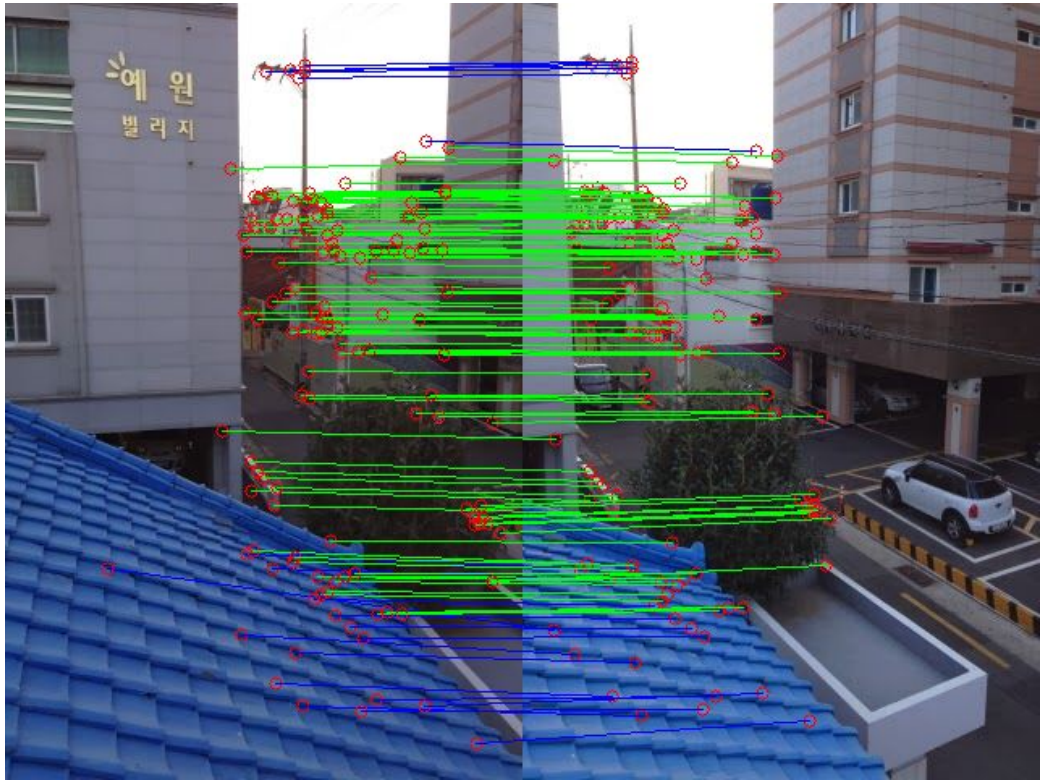
4. This function also has a threshold parameter, which uses a floating-point number, which sets the minimum percentage of image points occupied by the current best homography before RANSAC stops.

5. The default threshold value accounts for 0.6 of the corresponding point. If the threshold is not reached, the function will loop 1000 times.

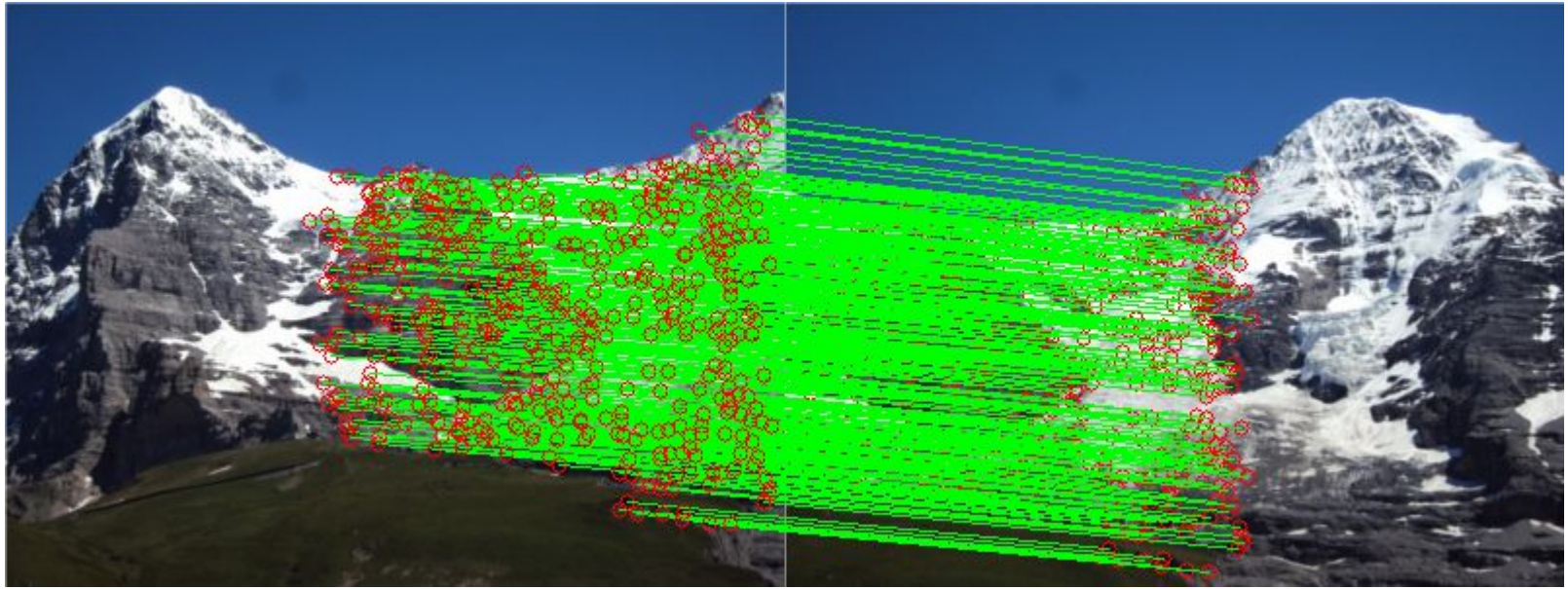
Experimental Result



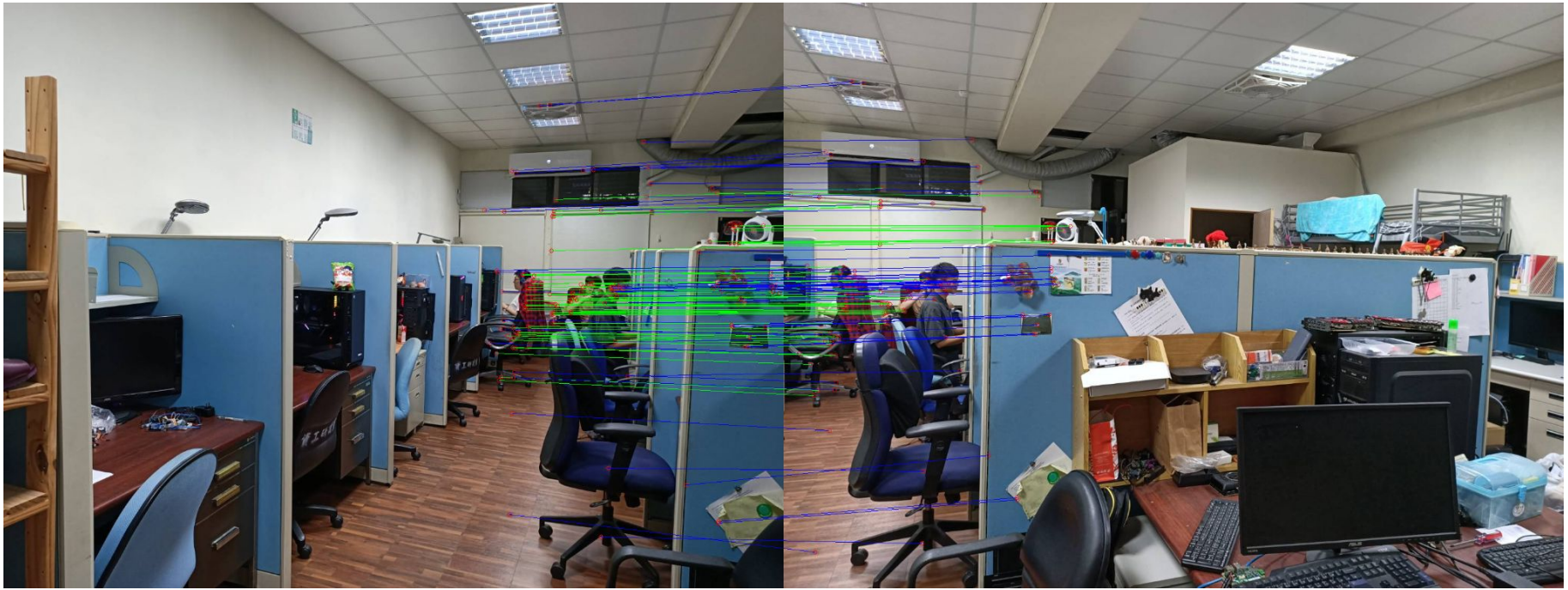
Experimental Result



Experimental Result



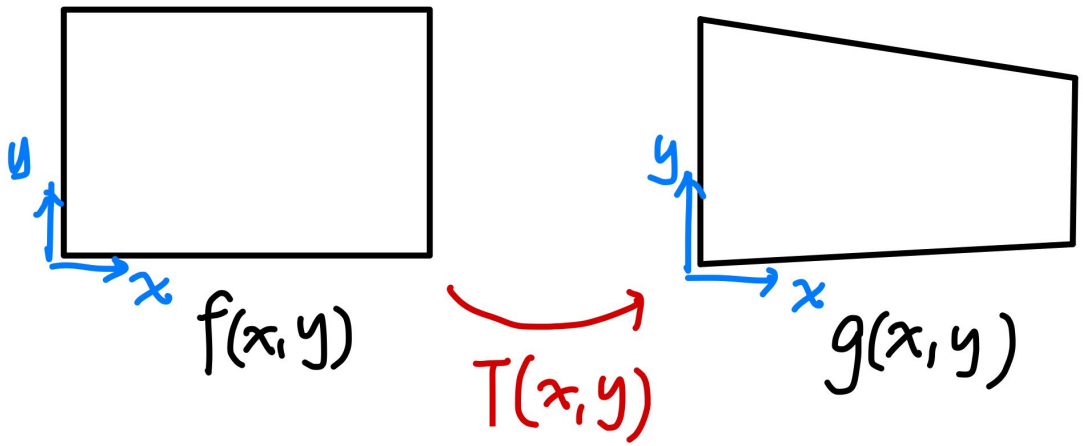
Experimental Result



blending and wrapping

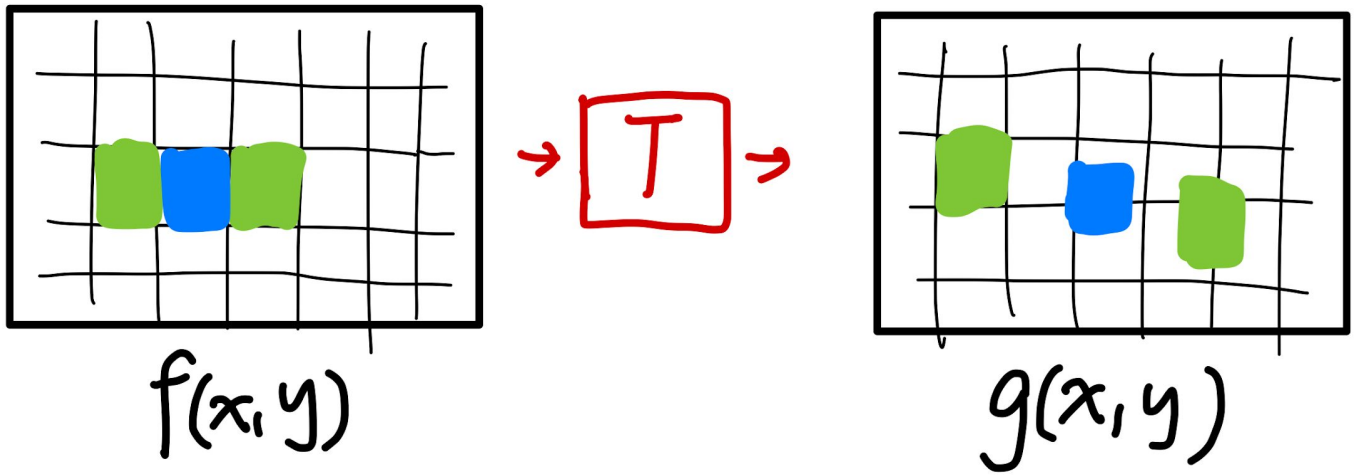
Given a transformation T and a image f(x, y), compute the transformed image g(x,y)

$$g(x, y) = f(T(x, y))$$



Send each pixel (x, y) in f(x, y) to its corresponding location T(x, y) in g(x, y)

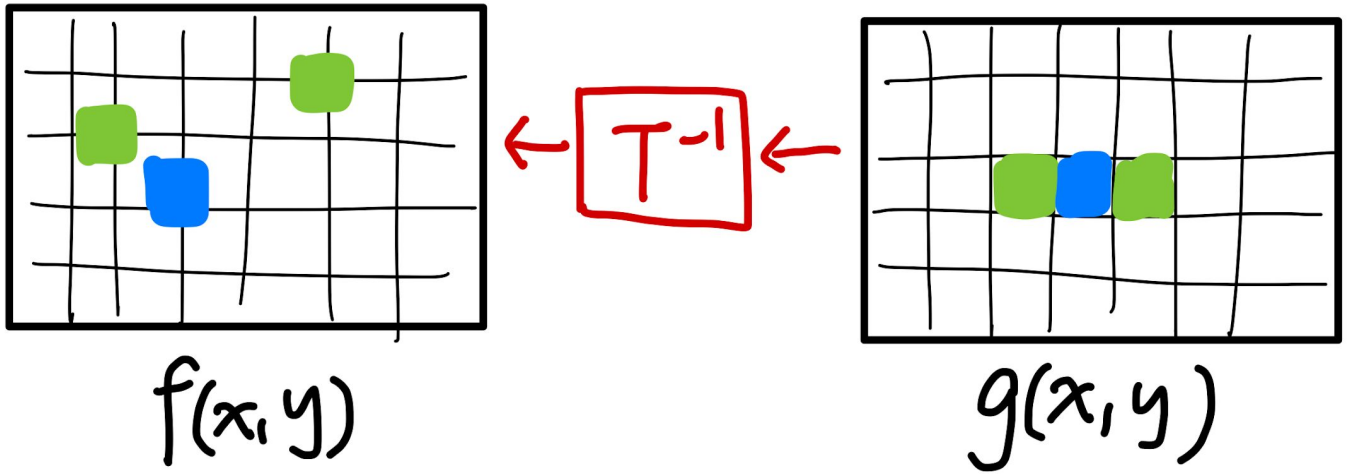
$$g(x, y) = f(T(x, y))$$



Backword wraping

Get each pixel (x, y) in g(x,y) from its corresponding location $T^{-1}(x, y)$ in f(x, y)

$$g(x, y) = f(T(x, y))$$



Why blending

Once that the homography matrix is computed, the homography can be applied to the source image, so that it will overlap to the target image.

Differences: exposure, white balance control...

Solutions: images blending,

make smoother the discontinuities

between the two overlapped pictures.

create blending mask

```
def make_weighting(img1, img2, side):
    h_blended = img1.shape[0]
    w_blended = img1.shape[1] + img2.shape[1]
    offset = int(100 / 2) # size of smoothing windows
    barrier = img1.shape[1] - int(100 / 2)
    mask = np.zeros((h_blended, w_blended))

    # using different weighting for left and right
    if side == 'left':
        mask[:, barrier - offset:barrier + offset] = np.tile(np.linspace(1, 0, 2 * offset).T, (h_blended, 1))
        mask[:, :barrier - offset] = 1
    else:
        mask[:, barrier - offset:barrier + offset] = np.tile(np.linspace(0, 1, 2 * offset).T, (h_blended, 1))
        mask[:, barrier + offset:] = 1
    return cv2.merge([mask, mask, mask])
```

Results

Results

Results

our dataset