ECE549 Computer Vision: Assignment 3

Yutong Xie

NetID: yutongx6

Instructor: Prof. Saurabh Gupta

Apr 6, 2020

1. Q1 Stitching pairs of images

1.1 Putative Matches

In this part, I select all pairs that descriptor distances are below threshold 10000. The putative matches overlaid on the image pairs are shown below.

1.2 Homography Estimation and RANSAC

In this part, I implement the estimation of homography and use the RANSAC method to select great matching point. I calculate the homography between two images based on the following equation.

$$\begin{bmatrix} 0^T & \mathbf{x}_1^T & -y_1'\mathbf{x}_1^T \\ \mathbf{x}_1^T & 0^T & -x_1'\mathbf{x}_1^T \\ \cdots & \cdots & \cdots \\ 0^T & \mathbf{x}_n^T & -y_n'\mathbf{x}_n^T \\ \mathbf{x}_n^T & 0^T & -x_n'\mathbf{x}_n^T \end{bmatrix} \begin{pmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \\ \mathbf{h}_3 \end{pmatrix} = 0 \qquad \mathbf{A}\mathbf{h} = 0$$

Because H has 8 degrees of freedom, I random select 4 pairs of matching points and each match gives us two linearly independent equations. Then, I use the homogeneous least-squares method to find h that minimizing ||Ah^2||. The code is shown below.

```
A = np.array(A)
U, S, V = np.linalg.svd(A)
sv = V[-1,:].reshape((3,3))

H = sv/sv[2,2]
```

For the RANSAC part, I follow the algorithm and keep the transformation with the largest number of inliers.

RANSAC loop:

1. Randomly select a *seed group* of matches
2. Compute transformation from seed group
3. Find *inliers* to this transformation
4. If the number of inliers is sufficiently large, re-compute least-squares estimate of transformation on all of the inliers

First, I randomly selected 4 pairs of points to calculate the homography and abandon the homography if it is degenerate.

Second, I calculate the error between the estimated point calculated by H and the ground truth point.

Third, I find inliers whose error is below the threshold.

Fourth, if the number of inliers is larger than the minimum number of inliers required and the maximum number of inliers we can get so far, I will update the transformation model. If the number of inliers is equal to the maximum number of inliers we get so far, I will keep the one with the lower error. The sample code is shown below.

```
# Get all inliers fitted the current model
errors = cal_errors(H, data)
index = np.where(errors < t)[0]
inliers = data[index]

# if total inliers number is larger than d and max_inliers, we update the model
if len(inliers) >= d and len(inliers) >= max_inliers:
    curr_err = errors[index].sum() / len(inliers)
    if len(inliers) == max_inliers:
        # if the inliers number equals to max inliers number
        # check the residual
        if curr_err > best_err:
            continue

    best_err = errors[index].sum() / len(inliers)
    best_inliers = inliers.copy()
    bestmodel = H.copy()
    max_inliers = len(inliers)
```

The value of parameters I set is shown here.

RANSAC iteration times: 10000

Threshold value to determine whether a data point is an inlier: 300

Minimum number of inliers required: 15

The result for this part is here.

```
Average residual: 2.075660589668713
Inliers: 22
```

And the locations of inlier matches in both images is shown below.

1.3 Image Warping

In this part, I stitch two images based on the transformation I estimated in the last part. First, I calculate the size of the new image using H. Then, I composite the image by averaging the pixel values where the two images overlap. The result is shown below.

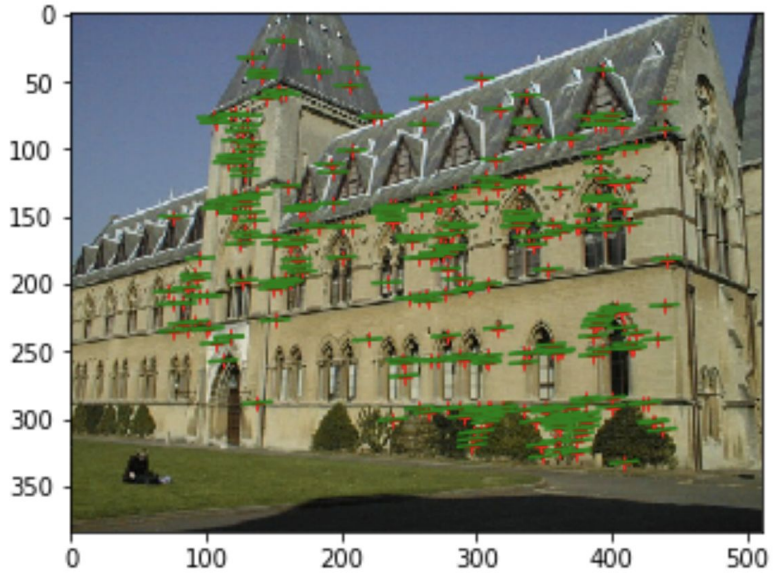
2. Q2 Fundamental Matrix Estimation, Camera Calibration, Triangulation

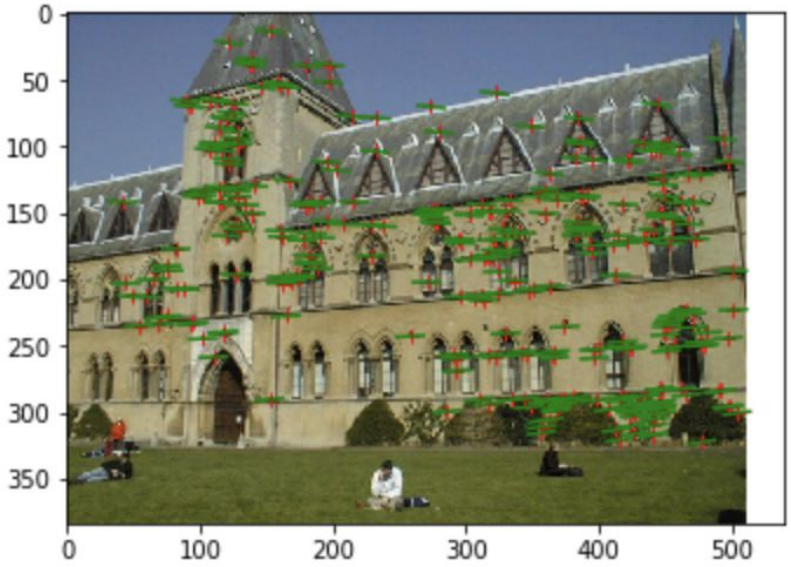
2.1 Fundamental Matrix Estimation

In this part, I find a fundamental matrix based on point matches using both normalized and the unnormalized algorithms. The results are shown below.

1. Library, unnormalized algorithms:

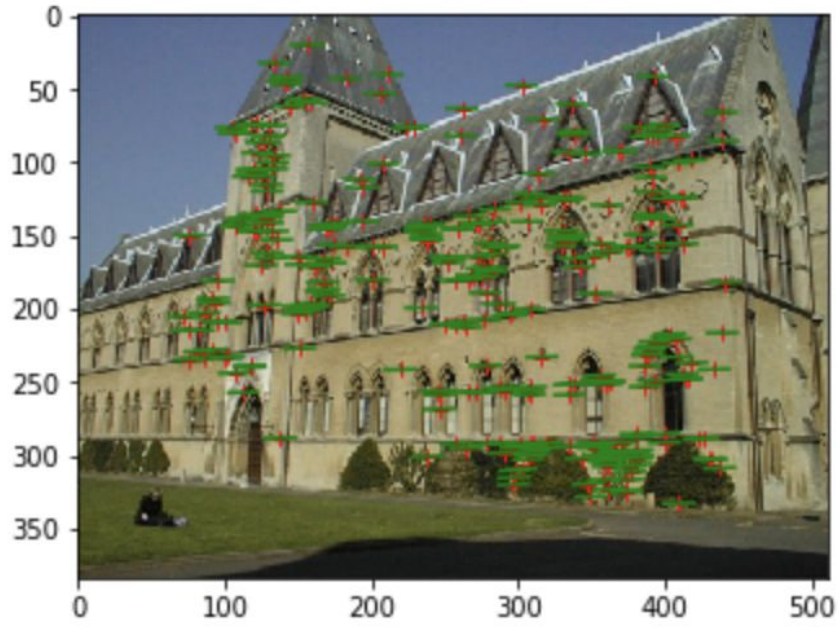
```
library: fundamental matrix (non-normalized method) =  [[ 1.32546895e-06 -1.36852466e-05  6.8
3862987e-04]
 [ 2.88625175e-05 -2.66854091e-07 -4.09703775e-02]
 [-5.63235250e-03  3.73349826e-02  1.00000000e+00]]
library: residual in frame 2 (non-normalized method) =  0.17921336679536498
library: residual in frame 1 (non-normalized method) =  0.14912309938157822
library: residual combined   (non-normalized method) =  0.16416823308847162
```

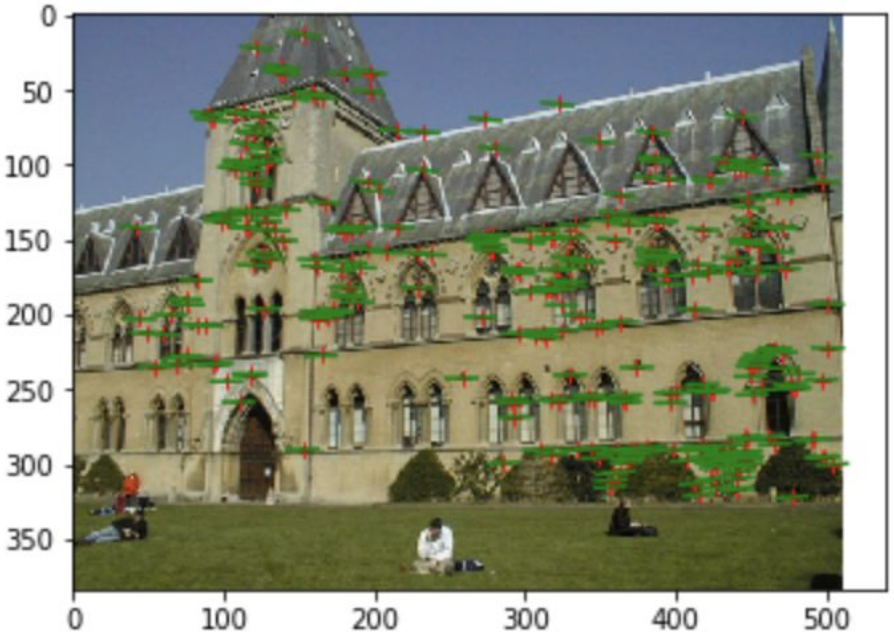




2. library, normalized method

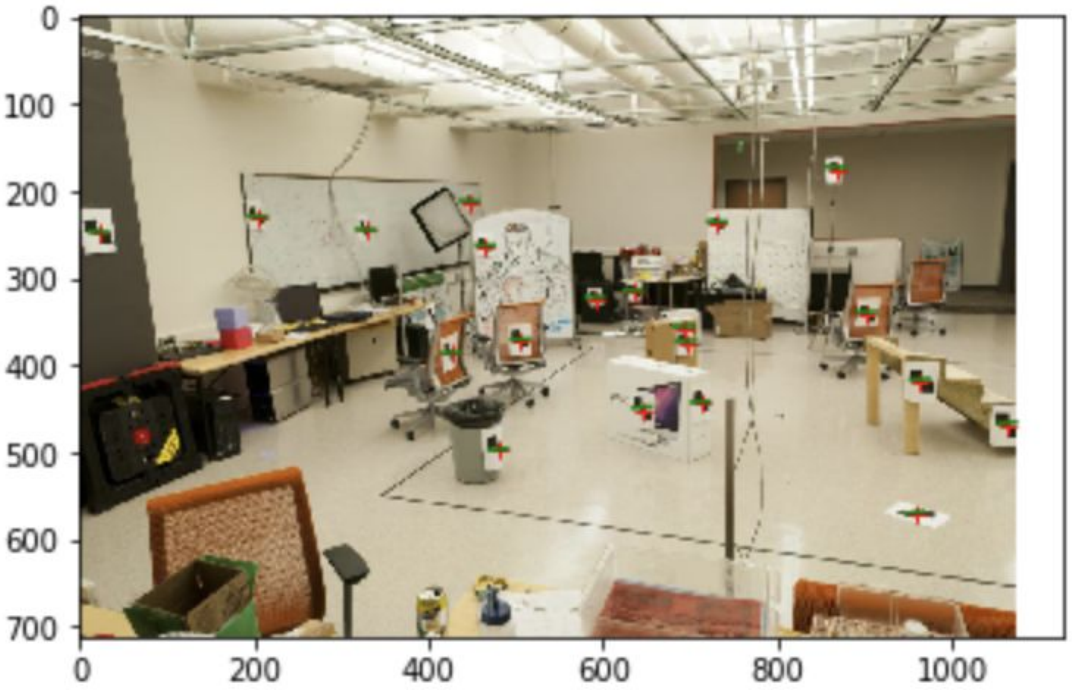
```
library: fundamental matrix (normalized method) =  [[-5.13835193e-06  7.79315652e-05 -8.72420653e-03]
 [-3.05981308e-04 -1.06772717e-06  5.29149261e-01]
 [ 6.97594921e-02 -4.77098329e-01 -1.27984131e+01]]
library: residual in frame 2 (normalized method) =  0.06077736897675683
library: residual in frame 1 (normalized method) =  0.054971268491108435
library: residual combined   (normalized method) =  0.05787431873393263
```

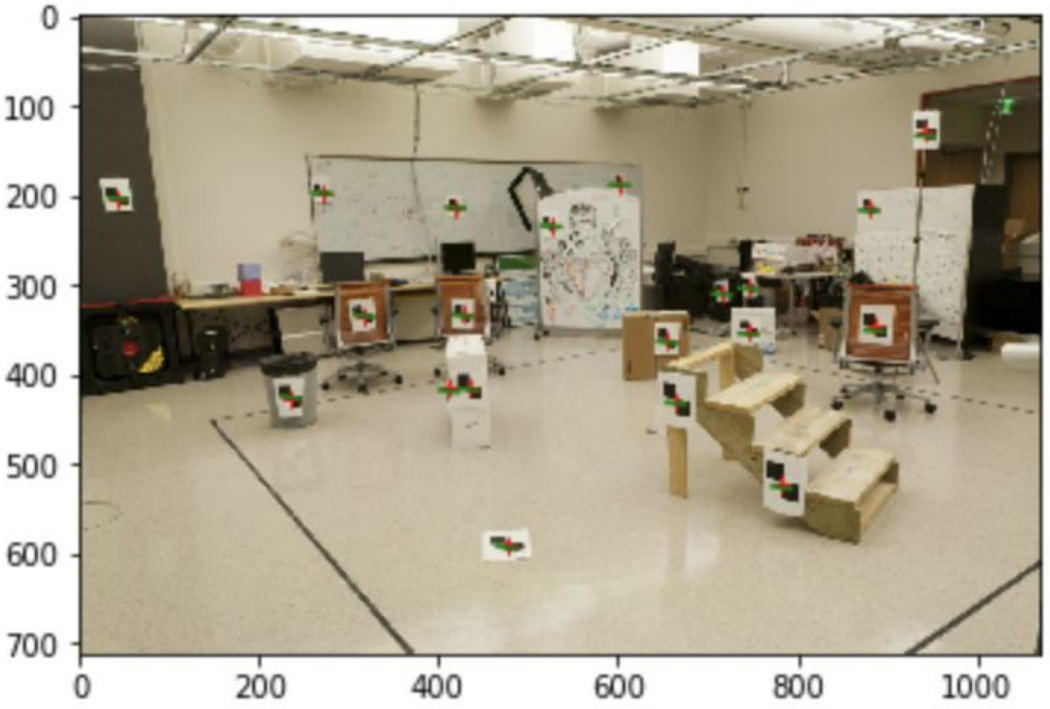




3. Lab, unnormalized method

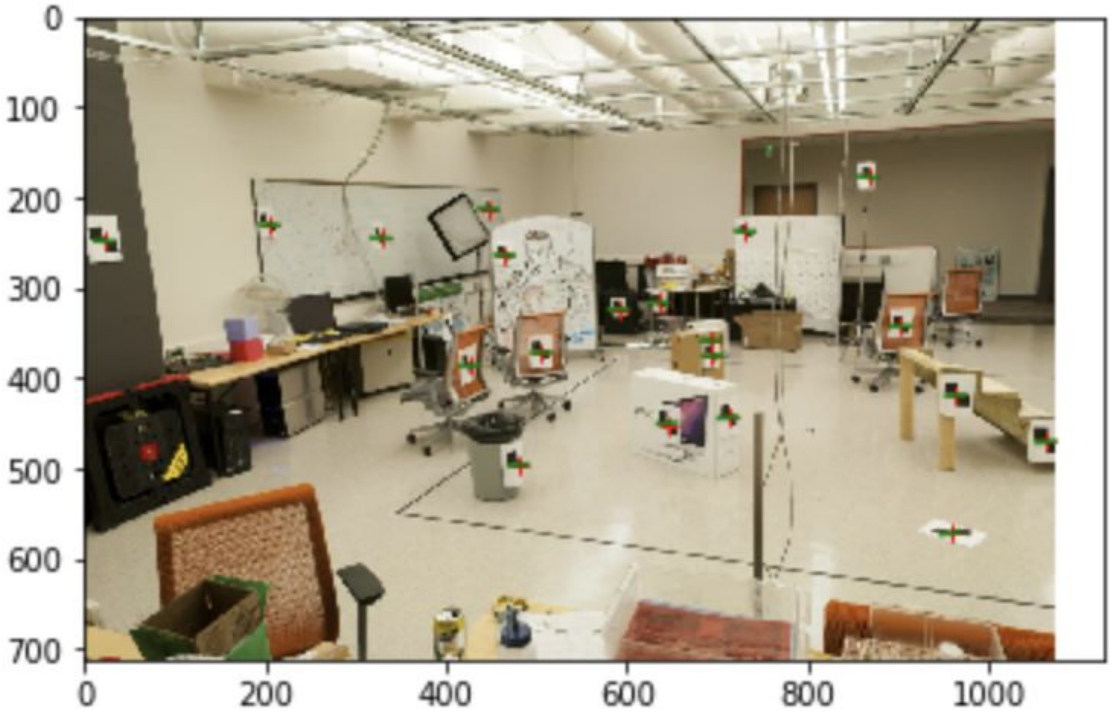
```
lab: fundamental matrix (non-normalized method) =  [[-5.36532415e-07  7.90760078e-06 -1.88694534e-03]
 [ 8.83981093e-06  1.21382365e-06  1.72419095e-02]
 [-9.07836099e-04 -2.64366809e-02  1.00000000e+00]]
lab: residual in frame 2 (non-normalized method) =  6.567091501769509
lab: residual in frame 1 (non-normalized method) =  9.760655424325847
lab: residual combined   (non-normalized method) =  8.163873463047679
```

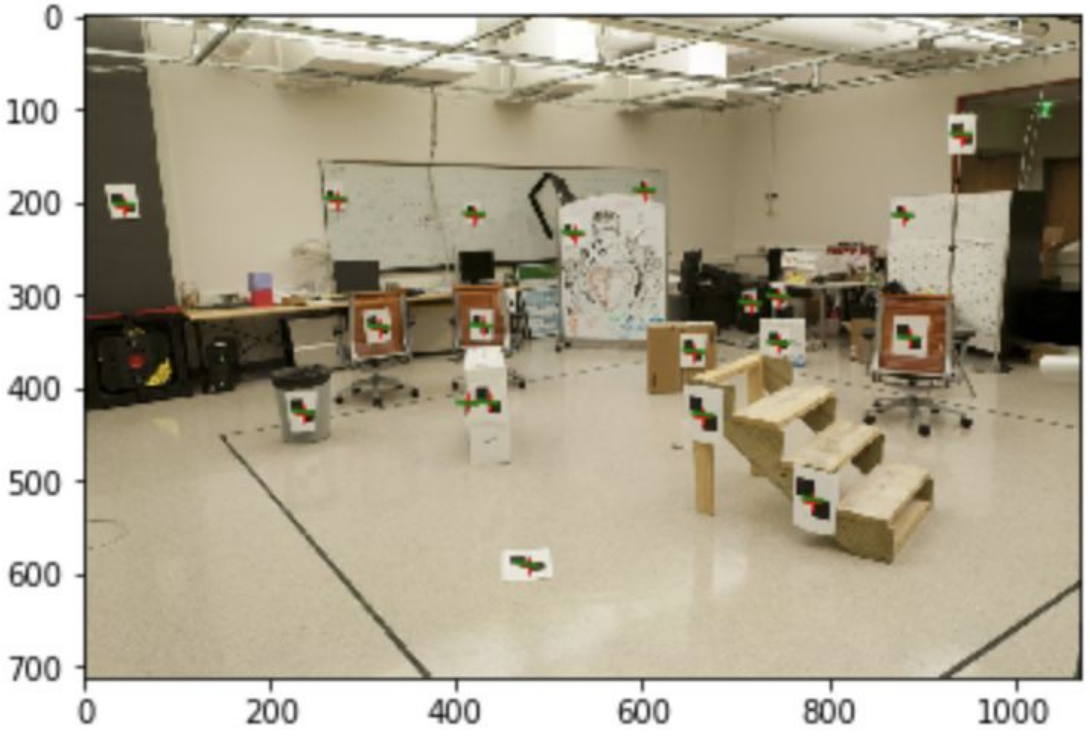




4. Lab, normalized method

```
lab: fundamental matrix (normalized method) =  [[ 7.38604202e-06 -1.01996962e-04  2.54794671e-02]
 [-7.04198557e-05  1.74306518e-05 -2.04983331e-01]
 [ 1.58540789e-03  2.81678807e-01 -6.59155675e+00]]
lab: residual in frame 2 (normalized method) =  0.527505462272854
lab: residual in frame 1 (normalized method) =  0.567076777791323
lab: residual combined   (normalized method) =  0.5472911200320885
```





2.2 Camera Calibration

In this part, I calculate the camera projection matrices by using 2D matches in both views and 3D point coordinates in lab_3d.txt. And I evaluate the projection matrices using the given evaluation function. The result is shown below.

```
lab 1 camera projection
[[-4.53187041e+03 -2.13760331e+02  6.55731767e+02  1.43125880e+06]
 [-4.48880175e+02 -9.31617589e+02  4.05512340e+03  2.98472081e+05]
 [-2.45529487e+00 -4.01727797e+00  1.00000000e+00  1.94283277e+03]]

lab 2 camera projection
[[-3.64428424e+03  2.11186971e+03  6.97164388e+02  4.34640617e+05]
 [-8.13701852e+02 -5.38648854e+02  3.82454420e+03  2.95748507e+05]
 [-4.00069963e+00 -1.95030316e+00  1.00000000e+00  1.78129302e+03]]
residuals between the observed 2D points and the projected 3D points:
residual in lab1: 13.545832904243875
residual in lab2: 15.54495344429938
```

2.3 Camera Centers

In this part, I calculate the camera centers using the projection matrices.

```
lab1 camera center [305.83276769 304.20103826  30.13699243   1.        ]
lab2 camera center [303.10003925 307.18428016  30.42166874   1.        ]
library1 camera center [  7.28863053 -21.52118112  17.73503585   1.        ]
library2 camera center [  6.89405488 -15.39232716  23.41498687   1.        ]
```

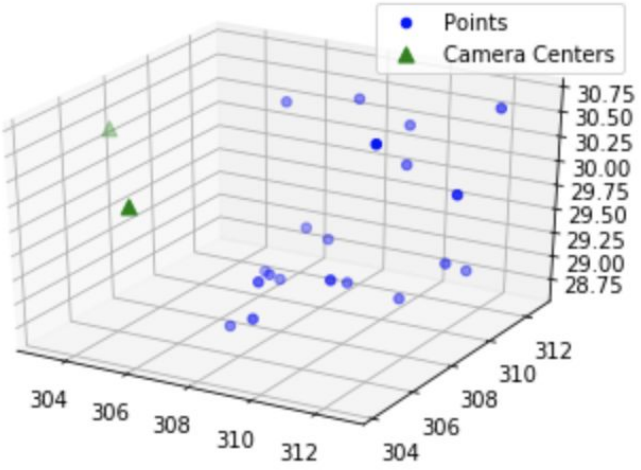
2.4 Triangulation

In this part, I use linear least squares to triangulate the 3D position of each matching pair of 2D points using the two camera projection matrices.

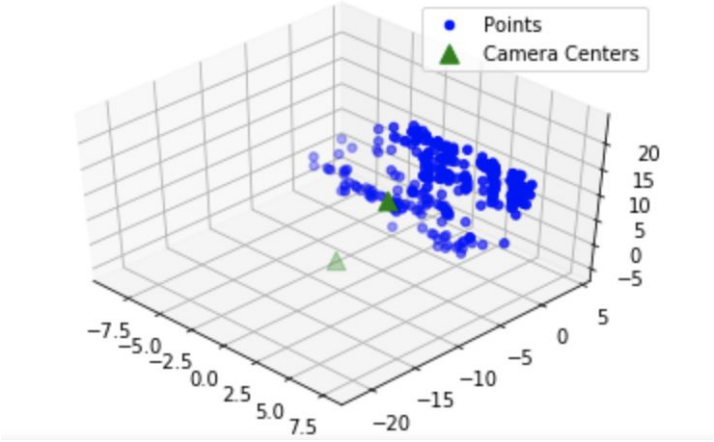
The result is shown below.

```
Mean 3D reconstuction error for the lab data:  0.39244
2D reprojection error for the lab 1 data:  11.332279608245567
2D reprojection error for the lab 2 data:  13.88877640920808
2D reprojection error for the library 1 data:  51.532271221300746
2D reprojection error for the library 2 data:  81.28132134991958
```

Also, I display the two camera centers and reconstructed points in 3D.



visualization for lab



visualization for library

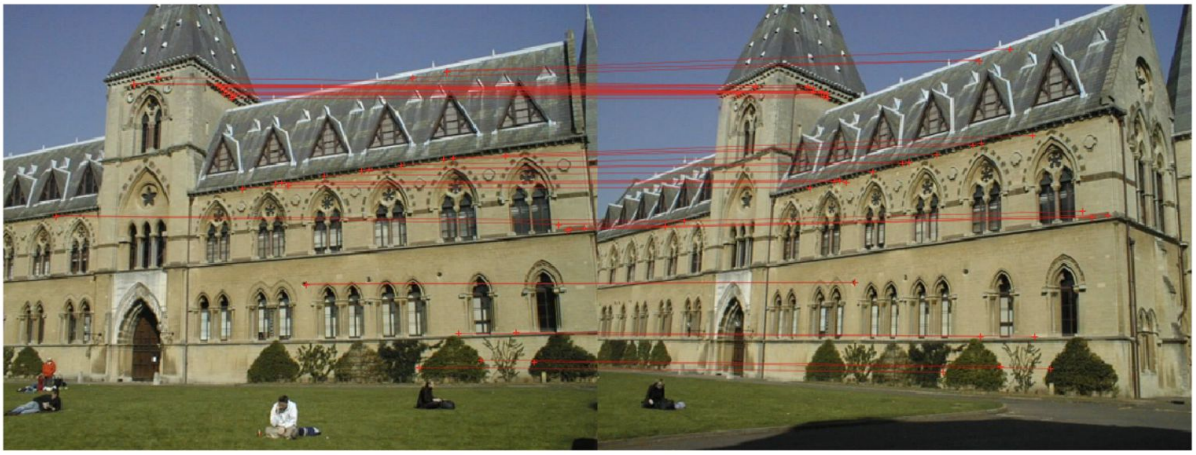
1.5 Extra Credits

In this part, I use the putative match generation and RANSAC code from question 1 to estimate fundamental matrices without ground-truth matches.

For the library one, the average residual and number of inliers are shown below.

```
Average residual: 61.33680688148416
Inliers: 44
```

And the putative match is shown as follows.



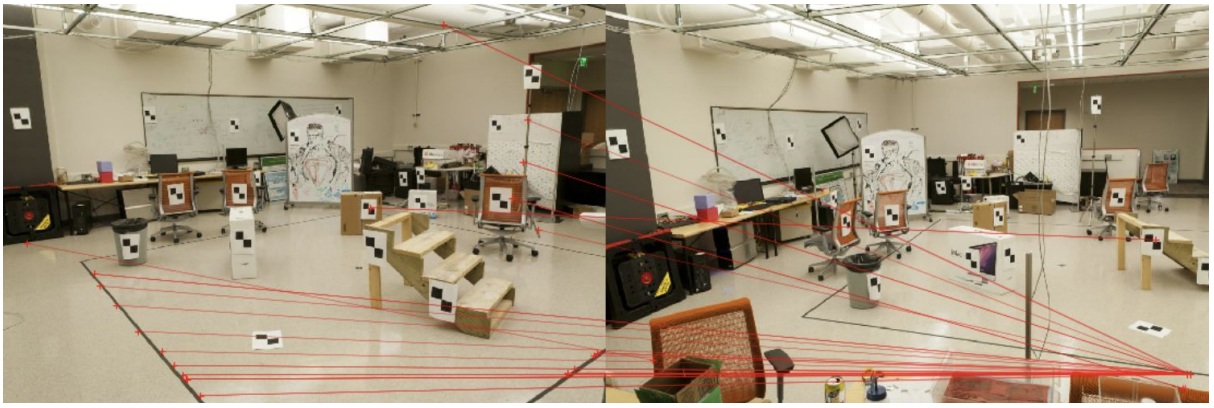
The quality of the result is not good compared to the one I get from the ground-truth matches.

```
library: residual in frame 2 (normalized method) =  19.349334057172598
library: residual in frame 1 (normalized method) =  46.36326834787548
library: residual combined   (normalized method) =  32.85630120252404
```

For the lab one, the average residual and number of inliers are shown below.

```
Average residual: 11.091227741538043
Inliers: 37
```

The putative matches are really bad.



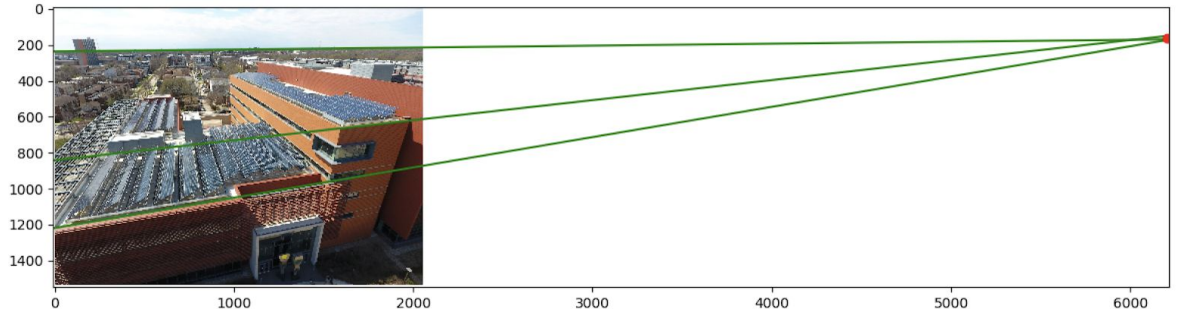
```
lab: residual in frame 2 (normalized method) =  0.22877470664963911
lab: residual in frame 1 (normalized method) =  58795.66556254393
lab: residual combined   (normalized method) =  29397.94716862529
```

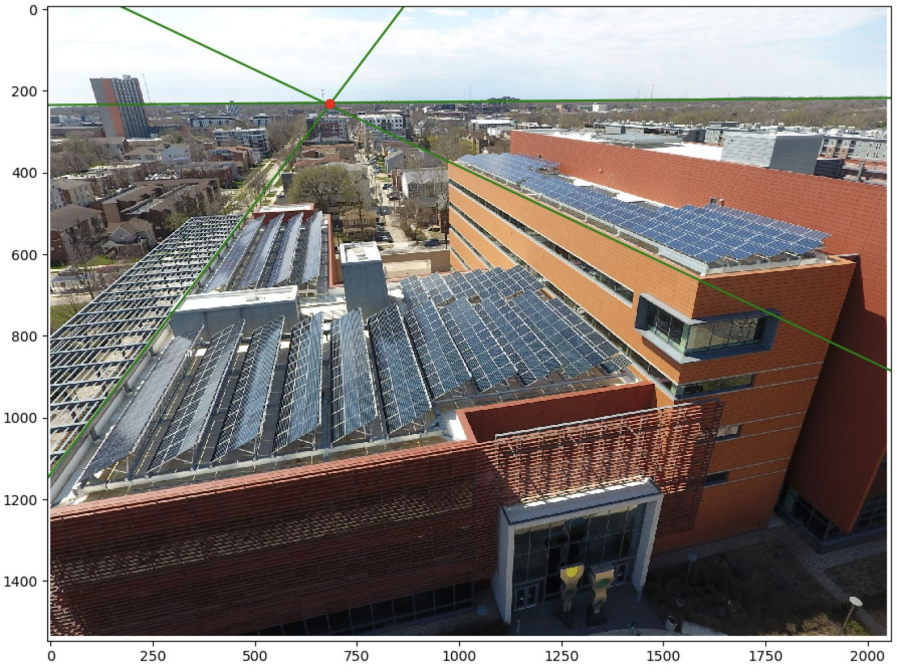
3. Q3 Single-View Geometry

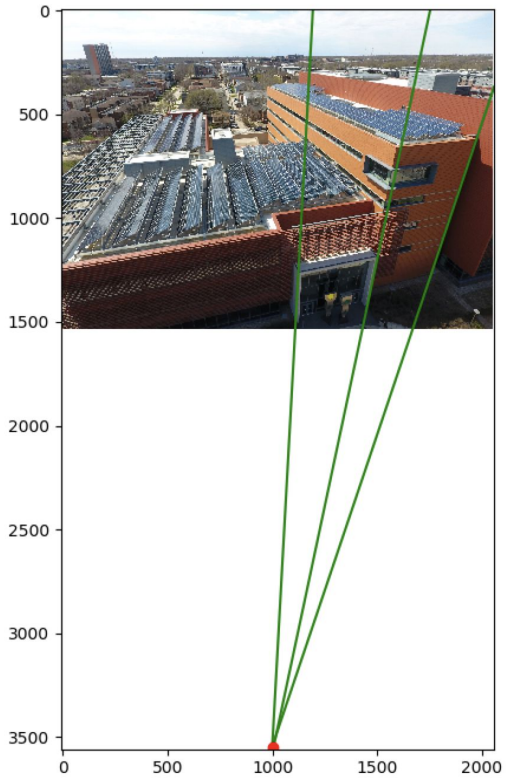
3.1 Vanishing Points

In this part, I estimate the three major orthogonal vanishing points. To keep the test result is the same for each execution, I use the data in the pickle file.

The results are shown below.







The pixel coordinates for the vanishing point is here.

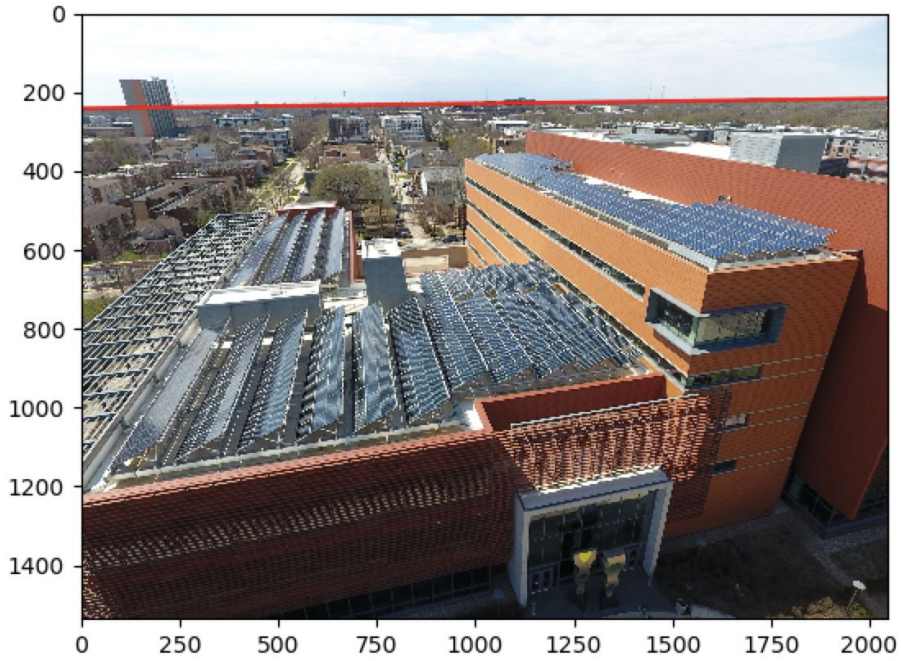
```
[[6.82625156e+02 6.20581280e+03 1.00071729e+03]
 [2.31870460e+02 1.64032644e+02 3.54755914e+03]
 [1.00000000e+00 1.00000000e+00 1.00000000e+00]]
```

3.2 Horizon

In this part, I estimate the horizon using two vanishing points I got in the last question and plot it on the image.

The function of horizontal lines in the form ax+by+c=0 is shown below, which is 0.01228x+0.999924y-0.024024.

```
[ 1.22814403e-02  9.99924580e-01 -2.40236592e+02]
```



3.3 Camera Calibration

In this part, I use the fact that the vanishing directions are orthogonal and solve the focal length and optical center of the camera.

I use the orthogonality constraint to solve the camera intrinsic parameters.

Let us align the world coordinate system with three orthogonal vanishing directions in the scene:

$$\mathbf{e}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{e}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \mathbf{e}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\lambda_i \mathbf{v}_i = \mathbf{K}\mathbf{R}\mathbf{e}_i$$

$$\mathbf{e}_i = \lambda_i \mathbf{R}^T \mathbf{K}^{-1} \mathbf{v}_i$$

Orthogonality constraint: $\mathbf{e}_i^T \mathbf{e}_j = 0$

$$\mathbf{v}_i^T \mathbf{K}^{-T} \mathbf{K}^{-1} \mathbf{v}_j = 0$$

Rotation disappears, each pair of vanishing points gives constraint on focal length and principal point

I use sympy package to create three parameters and form the K matrix first. Then I use three vanishing points and the K matrix to create three equations. Finally, I solve the equations and get the result.

```
def get_camera_parameters(vpts):
    """
    Computes the camera parameters. Hint: The SymPy package is suitable for this.
    """
    f = sympy.Symbol('f')
    u = sympy.Symbol('u')
    v = sympy.Symbol('v')
    p1 = sympy.Matrix(vpts[:,0])
    p2 = sympy.Matrix(vpts[:,1])
    p3 = sympy.Matrix(vpts[:,2])

    K = sympy.Matrix([[f, 0, u],[0, f, v],[0, 0, 1]])
    K_inv = K.inv()

    Eq1 = p1.T * K_inv.T * K_inv * p2
    Eq2 = p1.T * K_inv.T * K_inv * p3
    Eq3 = p2.T * K_inv.T * K_inv * p3

    ans = sympy.solve([Eq1, Eq2, Eq3], [f, u, v])

    f, u, v = ans[0]
    K = np.array([[f, 0, u], [0, f, v], [0, 0, 1]]).astype(np.float)
    return f, u, v, K
```

The camera calibration matrix is shown below.

```
u, v, f, K are:
965.334507693003 666.780361905498 1123.78034235151 [[-1.12378034e+03  0.00000000e+00  9.65334508e+02]
 [ 0.00000000e+00 -1.12378034e+03  6.66780362e+02]
 [ 0.00000000e+00  0.00000000e+00  1.00000000e+00]]
```

3.4 Rotation Matrix

In this part, I compute the rotation matrix for the camera according to the following method.

- Constraints on vanishing points: $\lambda_i \mathbf{v}_i = \mathbf{K}\mathbf{R}\mathbf{e}_i$
- After solving for the calibration matrix:

$$\lambda_i \mathbf{K}^{-1}\mathbf{v}_i = \mathbf{R}\mathbf{e}_i$$

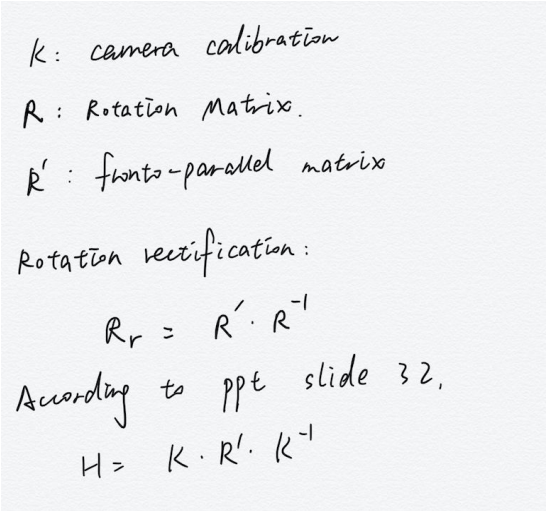
- Notice: $\mathbf{R}\mathbf{e}_1 = [\ \mathbf{r}_1 \quad \mathbf{r}_2 \quad \mathbf{r}_3] \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \mathbf{r}_1$
- Thus, $\mathbf{r}_i = \lambda_i \mathbf{K}^{-1}\mathbf{v}_i$
- Get λ_i by using the constraint $\|\mathbf{r}_i\|^2 = 1$.

The rotation I got is

```
[[-0.97349751 -0.0114418   0.2284112 ]
 [ 0.09339294 -0.93156329  0.35137957]
 [ 0.20875907  0.36339913  0.90794313]]
```

3.5 Fronto-parallel Warps

In this part, I use the estimated camera parameters and the camera rotations to compute warps that show the front, side and top views of the ECE building. The algorithm is shown below.



For the fronto-parallel matrix, I set them as follows.

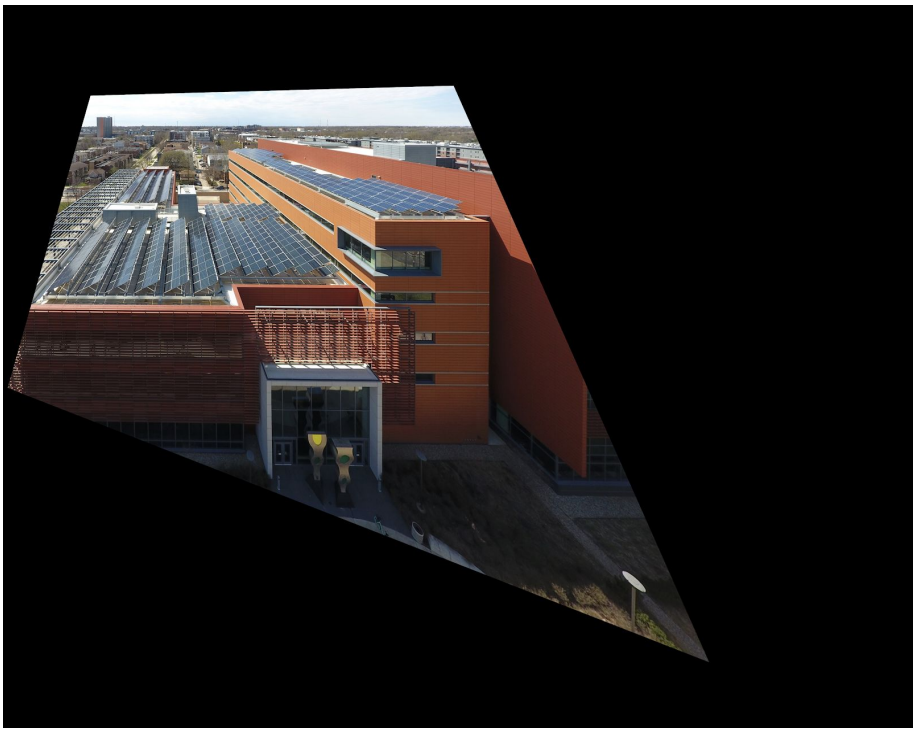
XY plane: [[1, 0, 0],[0, 1, 0],[0, 0, 1]]

XZ plane: [[0, 0, 1],[0 ,-1, 0],[1, 0, 0]]

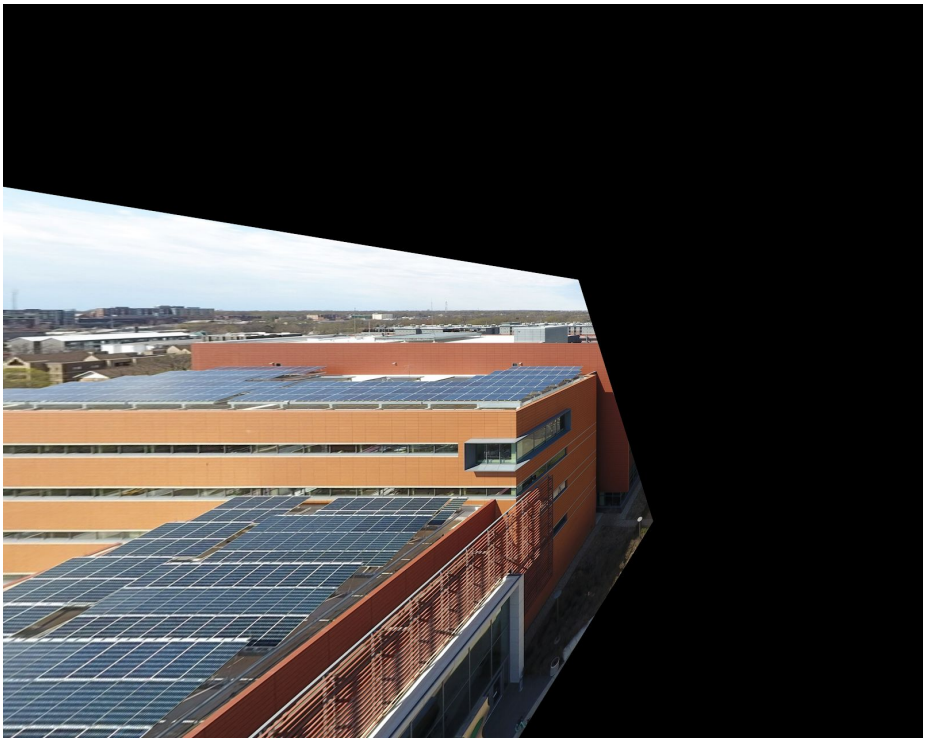
YZ plane: [[1, 0, 0],[0, 0, -1],[0, -1, 0]]

After calculating the homography, I warp the image and get the results.

Front image:



Side image:



Top image: