CSCI 677 Homework 4

Jingyun Yang

Student ID: 8357011225

1 Introduction

In this assignment, we write and execute a program to perform the "structure from motion" task. In section 2, we provide a summary of the program we write, including the directory structure and all the sources used in our program; in section 3, we show the outputs produced by our program on the given input image data; in section 4, we provide insights to our results by analyzing them qualitatively.

2 Summary of Program

2.1 Directory Structure

Under the program directory, `hw4.py` is the main python program that reads the input images, performs SFM, and outputs results to directories `result_a`, `result_b`, and `result_c`; `run.sh` is the executable file that will send each pair of corresponding input images into the program and execute SFM; `HW4_Data` is the provided folder for input image data. The directory tree, with `hw4` as the base directory name, is shown below.

```
hw4/
|-- hw4.py
|-- run.sh
|-- result_a/
|-- result_b/
|-- result_c/
|-- HW4_Data/
```

2.2 Executing the Program

To execute the program, run `sh run.sh` in the base directory. Outputs will be saved to directories `result_a`, `result_b`, and `result_c`.

2.3 Source Listing

Now we list the source files for our program. The edits I made based on the given skeleton code is outlined in the first 10 lines of the code.

```
''' hw4.py '''

##########################################
#                                        #
#  Edits Made to the File by Jingyun Yang  #
#   1. Refactoring                       #
#   2. Edit intrinsic matrix             #
#   3. Added code to plot point cloud    #
#                                        #
##########################################
```

```
import numpy as np
import cv2
import os
import argparse

from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

MIN_MATCH_COUNT = 10

''' Print a numpy matrix (or vector) with name.

Args:
    name: name of the matrix to print
    matrix: matrix object to be printed
    latex: whether to print the matrix in latex format
'''
def print_matrix(name, matrix, latex=False):
    print("{}:".format(name))
    if latex:
        numbers = matrix.tolist()
        print('\\\\'.join(['&'.join(["{:.4f}".format(s) for s in row]) for row in numbers]))
    else:
        print(matrix)

''' Safe mkdir that checks directory before creation.

Args:
    dir: directory to make
'''
def mkdir(dir):
    if not os.path.exists(dir):
        os.makedirs(dir)

''' Perform SFM on a pair of images.

Args:
    img1: first image
    img2: second image
    outdir: output directory for results
    latex: whether to print matrices in latex format
    show_plot: whether to show plot or not
'''
def process_image_pair(img1, img2, outdir, latex=False, show_plot=False):
    ############################
    #   0. Intrinsic Matrix K   #
    ############################

    K = np.array([[518.86,     0., 285.58],
                  [    0., 519.47, 213.74],
                  [    0.,     0.,     1.]])

    ##############################
    #   1. SIFT Feature Matching   #
    ##############################

    # detect sift features for both images
    sift = cv2.xfeatures2d.SIFT_create()
    kp1, des1 = sift.detectAndCompute(img1, None)
    kp2, des2 = sift.detectAndCompute(img2, None)

    # use flann to perform feature matching
    FLANN_INDEX_KDTREE = 0
    index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
    search_params = dict(checks = 50)

    flann = cv2.FlannBasedMatcher(index_params, search_params)
```

```
matches = flann.knnMatch(des1, des2, k=2)

# store all the good matches as per Lowe's ratio test.
good = []
for m,n in matches:
    if m.distance < 0.7 * n.distance:
        good.append(m)

if len(good) > MIN_MATCH_COUNT:
    p1 = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
    p2 = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)

draw_params = dict(matchColor = (0, 255, 0), # draw matches in green color
                   singlePointColor = None,
                   flags = 2)

img_siftmatch = cv2.drawMatches(img1, kp1, img2, kp2, good, None, **draw_params)
cv2.imwrite(os.path.join(outdir, 'sift_match.png'), img_siftmatch)

##########################
#   2. Essential Matrix   #
##########################

E, mask = cv2.findEssentialMat(p1, p2, K, cv2.RANSAC, 0.999, 1.0);

matchesMask = mask.ravel().tolist()

draw_params = dict(matchColor = (0,255,0), # draw matches in green color
                   singlePointColor = None,
                   matchesMask = matchesMask, # draw only inliers
                   flags = 2)

img_inliermatch = cv2.drawMatches(img1, kp1, img2, kp2, good, None, **draw_params)
cv2.imwrite(os.path.join(outdir, 'inlier_match.png'), img_inliermatch)
print_matrix("Essential matrix", E, latex=latex)

######################
#   3. Recover Pose   #
######################

points, R, t, mask = cv2.recoverPose(E, p1, p2)
print_matrix("Rotation", R, latex=latex)
print_matrix("Translation", t, latex=latex)

# (R*p2+t)-p1
p1_tmp = np.ones([3, p1.shape[0]])
p1_tmp[:2,:] = np.squeeze(p1).T
p2_tmp = np.ones([3, p2.shape[0]])
p2_tmp[:2,:] = np.squeeze(p2).T
print_matrix("(R*p2+t)-p1", (np.dot(R, p2_tmp) + t) - p1_tmp, latex=latex)

######################
#   4. Triangulation   #
######################

# calculate projection matrix for both camera
M_r = np.hstack((R, t))
M_l = np.hstack((np.eye(3, 3), np.zeros((3, 1))))

P_l = np.dot(K, M_l)
P_r = np.dot(K, M_r)

# undistort points
p1 = p1[np.asarray(matchesMask)==1,:,:]
p2 = p2[np.asarray(matchesMask)==1,:,:]
p1_un = cv2.undistortPoints(p1,K,None)
p2_un = cv2.undistortPoints(p2,K,None)
p1_un = np.squeeze(p1_un)
```

```
    p2_un = np.squeeze(p2_un)

    # triangulate points this requires points in normalized coordinate
    point_4d_hom = cv2.triangulatePoints(M_l, M_r, p1_un.T, p2_un.T)
    point_3d = point_4d_hom / np.tile(point_4d_hom[-1, :], (4, 1))
    point_3d = point_3d[:3, :].T

    # print point locations
    print_matrix("Point Locations", point_3d, latex=latex)

    ##############################
    #   5. Output 3D Point Cloud   #
    ##############################

    # plot 3D points
    fig = plt.figure()
    ax = Axes3D(fig)

    X = point_3d[:, 0].flatten()
    Y = point_3d[:, 1].flatten()
    Z = point_3d[:, 2].flatten()

    ax.scatter(X, Y, Z)

    # save figure
    plt.savefig(os.path.join(outdir, 'point_cloud.png'), format='png')

    # show plot if requested
    if show_plot:
        plt.show()

def main():
    # setup argparser
    parser = argparse.ArgumentParser(formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('--imgdir', type=str, default=None, help='directory to source images.')
    parser.add_argument('--img1', type=str, default=None, help='filename of the 1st image.')
    parser.add_argument('--img2', type=str, default=None, help='filename of the 2nd image.')
    parser.add_argument('--outdir', type=str, default=None, help='directory to save output data.'
                                                                 )
    parser.add_argument('--latex', help='show matrices in latex format', action='store_true')
    parser.add_argument('--show_plot', help='show image results', action='store_true')
    args = parser.parse_args()

    img1 = cv2.imread(os.path.join(args.imgdir, args.img1))
    img2 = cv2.imread(os.path.join(args.imgdir, args.img2))

    mkdir(args.outdir)
    process_image_pair(img1, img2, args.outdir, args.latex, args.show_plot)

if __name__ == '__main__':
    main()
```

```
: 'run.sh'
IMGDIR='./HW4_data'

python3 hw4.py --imgdir=$IMGDIR --img1=a1.png --img2=a2.png --outdir=result_a
python3 hw4.py --imgdir=$IMGDIR --img1=b1.png --img2=b2.png --outdir=result_b
python3 hw4.py --imgdir=$IMGDIR --img1=c1.png --img2=c2.png --outdir=result_c
```

3 Results

3.1 Feature Matches before and after RANSAC

We used SIFT feature matching and RANSAC to find corresponding points between pairs of images. Visualizations of feature matches before and after RANSAC are displayed in Figure 1.

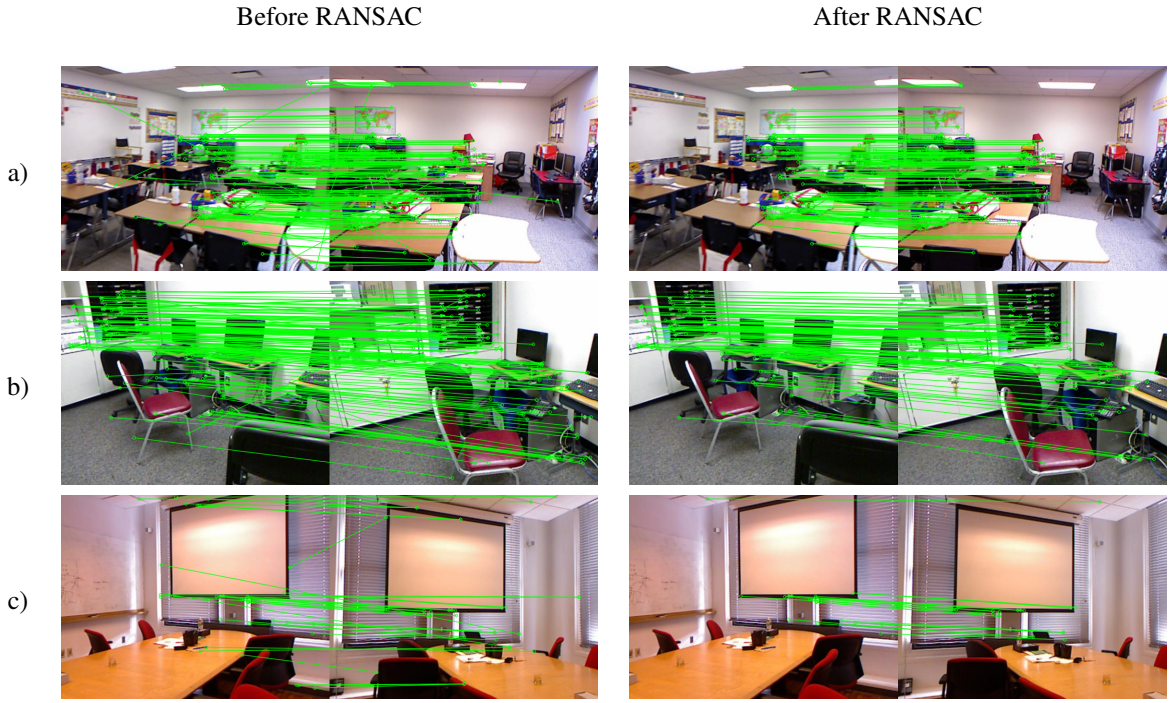


Figure 1: Matches Found before and after RANSAC (Better Viewed by Zooming In)

3.2 Recovered Essential Matrix, R, and t

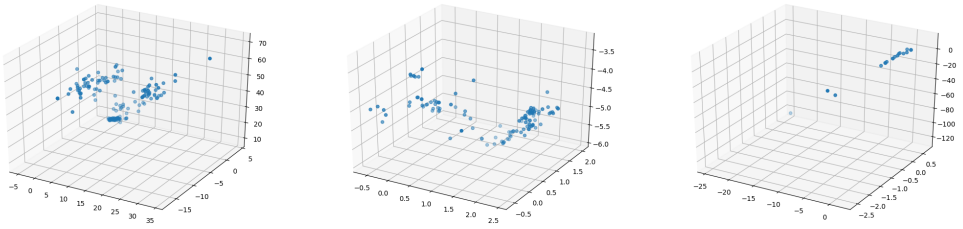
Key matrices recovered by our program are shown in Table 1. For recovered point locations, we leave these matrices to the appendix section at the end of the article because the matrices are very large.

	Essential Matrix	Rotation Parameters	Translation Parameters
$a)$	$\begin{pmatrix} 0.0353 & -0.6429 & 0.0059 \\ 0.4934 & 0.0286 & -0.5055 \\ -0.0264 & 0.2913 & 0.0080 \end{pmatrix}$	$\begin{pmatrix} 0.9309 & 0.0478 & -0.3621 \\ -0.0472 & 0.9988 & 0.0105 \\ 0.3622 & 0.0073 & 0.9321 \end{pmatrix}$	$\begin{pmatrix} -0.4133 \\ -0.0192 \\ -0.9104 \end{pmatrix}$
$b)$	$\begin{pmatrix} 0.0259 & -0.5914 & -0.2769 \\ 0.6406 & -0.0683 & -0.0700 \\ 0.2957 & 0.2383 & 0.0933 \end{pmatrix}$	$\begin{pmatrix} 0.9427 & -0.1067 & 0.3161 \\ 0.1076 & 0.9941 & 0.0145 \\ -0.3158 & 0.0203 & 0.9486 \end{pmatrix}$	$\begin{pmatrix} 0.3820 \\ -0.4001 \\ 0.8331 \end{pmatrix}$
$c)$	$\begin{pmatrix} 0.0047 & 0.3105 & 0.0060 \\ 0.3972 & -0.0394 & 0.5836 \\ -0.0234 & -0.6340 & -0.0325 \end{pmatrix}$	$\begin{pmatrix} 0.9897 & -0.0123 & -0.1425 \\ -0.0202 & -0.9984 & -0.0536 \\ -0.1416 & 0.0560 & -0.9883 \end{pmatrix}$	$\begin{pmatrix} 0.8983 \\ 0.0152 \\ 0.4390 \end{pmatrix}$

Table 1: Recovered Matrices

3.3 Point Cloud

We plot the 3D point cloud for the recovered 3D positions of the matched points. We show our results in Figure 2.



a b c

Figure 2: 3D Point Clouds for Reconstructed 3D Positions of Matched Points (Better Viewed by Zooming In)

4 Qualitative Analysis

Overall, the program works fine in finding pairs of corresponding points in the two images, reconstructing essential matrix and camera parameters, and reconstructing the 3D points in the scene.

The individual results generated by image pair a, b, and c, though, have different qualities. The program, on input image pair a, for example, is able to find a large number of matching points and produce 3D points that almost form surfaces. The outputs of image pairs b and c, in comparison, don't look as good. In particular, the program, when executed with input image pair c, is only able to find a small number of corresponding point pairs. Even among these matching points, some are outliers. In the end, the program is only able to reconstruct a small number of 3D points in the scene.

By looking at the properties of three input image pairs, one might be able to obtain insights on why there are such differences in the results. In image pair a, the two images are taken in very similar angle. Thus, it is very easy to find matching points and reconstruct 3D points from them. In image pair b, the two images are taken from different vertical angles. Such a difference will trigger result in different occlusions of objects in the scene, which makes the job of matching points harder. In image pair c, the two images are taken in very different angles, which results in bad results.

In summary, we attribute the variable performance of the method on different input image pairs to the following reasons the difference in viewing angles, the difference in occlusions of objects in the scenes, and the difference in quality and quantity of found matching points in the images.

A Appendix: Recovered 3D Point Locations

Below is the recovered 3D points for image pair a):

$$
\begin{pmatrix}
-4.7915 & -2.9082 & 47.0657 \\
-3.0126 & -4.7405 & 41.1540 \\
-2.8751 & -4.5577 & 39.6518 \\
-2.6712 & -4.7176 & 37.1499 \\
-1.3965 & -1.0789 & 26.3160 \\
-0.3764 & 1.9797 & 9.0271 \\
-0.3764 & 1.9797 & 9.0271 \\
-0.3544 & 1.6187 & 11.0757 \\
-0.7774 & -3.4205 & 37.6844 \\
-0.1452 & 1.6317 & 10.6816 \\
-0.1434 & 1.8421 & 10.6649 \\
-0.1668 & -2.3250 & 35.5551 \\
0.0029 & 1.9500 & 10.3543 \\
0.0141 & 1.2709 & 10.9433 \\
0.1249 & -1.7235 & 38.8971 \\
0.0351 & 1.9870 & 10.2824 \\
0.1804 & -2.8307 & 32.8377 \\
0.2698 & -5.6471 & 42.2697 \\
0.1017 & 1.8550 & 10.3360 \\
0.1142 & 1.6589 & 10.4094 \\
0.5458 & -10.0271 & 43.7815 \\
0.1467 & 1.5571 & 10.2879 \\
0.6507 & -5.6300 & 44.4390 \\
0.7276 & -9.6550 & 46.4347 \\
0.7336 & -7.1044 & 43.1353 \\
0.1960 & 1.8957 & 10.1409 \\
0.9056 & -2.2649 & 46.4789 \\
0.8915 & -7.8802 & 44.9269 \\
0.9151 & -5.1452 & 44.0893 \\
0.2371 & 1.9794 & 9.9497 \\
1.1623 & -5.7510 & 42.9965 \\
1.4059 & -6.1373 & 48.3104 \\
1.5197 & -6.7418 & 48.5774 \\
1.5197 & -6.7418 & 48.5774 \\
1.2199 & -3.0633 & 37.3775 \\
1.6296 & -5.5179 & 49.1818 \\
0.3383 & 1.6127 & 9.9879 \\
0.3612 & 1.8871 & 10.0058 \\
1.6050 & -2.9662 & 42.7010 \\
1.6816 & -5.5818 & 42.8201 \\
1.8371 & -8.3366 & 44.2839 \\
2.1013 & -8.4976 & 43.0501 \\
0.5608 & 1.9462 & 10.3934 \\
2.0102 & 1.2657 & 36.8298 \\
2.5314 & 1.0559 & 46.2400 \\
2.5314 & 1.0559 & 46.2400 \\
0.6103 & 1.0085 & 11.1209 \\
2.8748 & -5.7355 & 47.5130 \\
0.6345 & 1.7444 & 10.3163 \\
2.7787 & 1.5208 & 42.7905 \\
2.9704 & -2.9922 & 45.0596 \\
2.8623 & -8.9337 & 43.2145 \\
0.7969 & 1.2932 & 11.7477 \\
0.7478 & 1.9551 & 10.5297 \\
0.7928 & 2.0142 & 10.6240 \\
0.7974 & 1.8924 & 10.6101 \\
0.7898 & 1.7277 & 10.2223 \\
3.4523 & -9.3374 & 44.5645 \\
0.9405 & 1.3633 & 11.5434 \\
0.8610 & 1.9289 & 10.5215 \\
0.8178 & 1.8979 & 9.9211 \\
0.9574 & 1.7483 & 11.5600 \\
4.1365 & -0.1148 & 49.6545 \\
3.9104 & -2.9399 & 45.4549 \\
3.8988 & -3.4120 & 44.4313 \\
4.2219 & -7.7786 & 46.9281 \\
3.7422 & -4.4752 & 39.4956 \\
1.1907 & 1.3702 & 12.2146 \\
1.1838 & 1.2090 & 11.8471 \\
4.5689 & -10.3990 & 43.8400 \\
4.5792 & -3.0454 & 43.5985 \\
4.5792 & -3.0454 & 43.5985 \\
5.8396 & -17.3893 & 54.0244 \\
5.8396 & -17.3893 & 54.0244 \\
3.1528 & 0.3336 & 29.1491 \\
4.3039 & -12.3768 & 37.8201 \\
1.2576 & 1.9026 & 10.9960 \\
2.4314 & 0.5366 & 20.0116 \\
5.6304 & -2.4199 & 46.3945 \\
6.1956 & -6.6863 & 49.9001 \\
5.6517 & -1.3029 & 44.2586 \\
5.8512 & -2.0873 & 45.4954 \\
1.4244 & 1.5179 & 10.8489 \\
5.7610 & -0.4253 & 42.1224 \\
1.4442 & 1.7033 & 10.4856 \\
1.4442 & 1.7033 & 10.4856 \\
1.7075 & 1.4454 & 12.3524 \\
6.7278 & -0.9691 & 48.5343 \\
1.6530 & 1.3669 & 11.7147 \\
6.6939 & -5.4970 & 47.2222 \\
2.6660 & 1.0517 & 18.2140 \\
3.5099 & 0.2978 & 23.9535 \\
1.7611 & 1.5724 & 11.9779 \\
1.5643 & 1.7327 & 10.4994 \\
1.6456 & 2.0471 & 10.8822 \\
1.7755 & 1.6906 & 11.6257 \\
1.9555 & 1.7325 & 11.5978 \\
1.9555 & 1.7325 & 11.5978 \\
1.9670 & 2.1829 & 11.4416 \\
1.6124 & 2.6166 & 8.7847 \\
2.1622 & 1.9040 & 11.5233 \\
2.1429 & 1.2731 & 11.3883 \\
2.3937 & 1.4344 & 12.3756 \\
2.4619 & 1.5114 & 12.4764 \\
2.2592 & 2.0433 & 11.1508 \\
2.6791 & 1.4646 & 13.0930 \\
2.6791 & 1.4646 & 13.0930 \\
4.3707 & 0.9412 & 21.1542 \\
5.0489 & -0.0321 & 24.1408 \\
2.6343 & 1.2516 & 12.5568 \\
2.4478 & 2.0080 & 11.3367 \\
2.4454 & 2.0096 & 11.3152 \\
4.5775 & 0.5460 & 21.0943 \\
2.6568 & 1.1772 & 12.0158 \\
5.5919 & 0.3369 & 24.7003 \\
2.5725 & 2.1995 & 11.3380 \\
2.9444 & 1.3818 & 12.7625 \\
5.4074 & 0.5055 & 23.0627 \\
3.5134 & 1.3305 & 14.4274 \\
3.2676 & 1.4838 & 12.5118 \\
3.4119 & 1.5010 & 12.2910 \\
11.4824 & -2.2894 & 38.9751 \\
6.3151 & 1.0538 & 21.3226 \\
4.8113 & 2.0573 & 15.9997 \\
2.9758 & 1.9306 & 9.3777 \\
8.6040 & 0.9034 & 26.9696 \\
15.4583 & -1.0651 & 47.4697 \\
13.7406 & -1.8666 & 40.1605 \\
3.7521 & 2.9198 & 10.8542 \\
3.9087 & 2.9022 & 11.2468 \\
3.9695 & 2.4997 & 11.4021 \\
14.1820 & -1.4114 & 40.5830 \\
3.8219 & 2.9935 & 10.8495 \\
15.0386 & -2.6581 & 42.3492 \\
15.0386 & -2.6581 & 42.3492 \\
15.9143 & -2.4757 & 44.5383 \\
12.6910 & 0.1185 & 35.3180 \\
12.4796 & -0.5405 & 34.1757 \\
14.9134 & -2.2311 & 40.5823 \\
8.1919 & 2.5633 & 22.3038 \\
13.2899 & -1.0039 & 35.6124 \\
13.2899 & -1.0039 & 35.6124 \\
18.6559 & -4.8139 & 49.9730 \\
15.5511 & -2.1176 & 41.4587 \\
15.5511 & -2.1176 & 41.4587 \\
14.1394 & -0.3128 & 37.1459 \\
14.2519 & -0.6763 & 37.0467 \\
10.2036 & 1.7126 & 25.9153 \\
13.6935 & -0.5550 & 34.6063 \\
16.0614 & -4.2684 & 40.5626 \\
16.0614 & -4.2684 & 40.5626 \\
14.5034 & -0.9157 & 36.5251 \\
15.1242 & -1.2992 & 38.0906 \\
20.7546 & -4.9465 & 51.3519 \\
16.1395 & -0.4954 & 39.7618 \\
18.8944 & -1.0053 & 46.0182 \\
9.9286 & 2.3571 & 24.1467 \\
11.9055 & 3.4562 & 28.8067 \\
14.8269 & 0.4808 & 34.8592 \\
22.1917 & -5.7507 & 51.2544 \\
25.6093 & -4.4289 & 58.7503 \\
15.1232 & -0.7939 & 34.3557 \\
8.7748 & 1.7361 & 19.6839 \\
12.1270 & 4.0210 & 26.5069 \\
12.7872 & 3.2527 & 27.8409 \\
12.7872 & 3.2527 & 27.8409 \\
23.5630 & -2.3173 & 51.1980 \\
15.8422 & 0.7739 & 34.2401 \\
17.5974 & -0.2671 & 36.8540 \\
20.0090 & -1.9041 & 41.8898 \\
34.0444 & -3.3437 & 70.5831
\end{pmatrix}
$$

Below is the recovered 3D points for image pair b):

$$
\begin{pmatrix}
2.4491 & 1.1765 & -4.7053 \\
2.3111 & 0.7082 & -4.4604 \\
2.3007 & 0.6519 & -4.4725 \\
2.2772 & 0.6795 & -4.4665 \\
2.3730 & 1.4464 & -4.7076 \\
2.2436 & 0.6656 & -4.4699 \\
2.3754 & 1.4801 & -4.7532 \\
2.3754 & 1.4801 & -4.7532 \\
2.3518 & 1.2025 & -4.7486 \\
2.3374 & 1.3387 & -4.7822 \\
2.2094 & 0.7156 & -4.5394 \\
2.3110 & 1.3221 & -4.7917 \\
2.2380 & 1.3350 & -4.6744 \\
2.1444 & 0.8364 & -4.5228 \\
2.1444 & 0.8364 & -4.5228 \\
2.0822 & 1.5517 & -4.4234 \\
2.1424 & 0.6809 & -4.6548 \\
2.1424 & 0.6809 & -4.6548 \\
2.1190 & 0.9667 & -4.6232 \\
2.0942 & 1.0340 & -4.6065 \\
2.0877 & 0.7500 & -4.6415 \\
2.0474 & 1.0211 & -4.5806 \\
2.0737 & 0.7354 & -4.6921 \\
2.0440 & 0.9154 & -4.6370 \\
2.0584 & 0.6843 & -4.6856 \\
2.0209 & 0.6491 & -4.7433 \\
2.0934 & 1.3207 & -5.1664 \\
2.0034 & 1.1763 & -5.1390 \\
2.0034 & 1.1763 & -5.1390 \\
1.9970 & 1.1300 & -5.1498 \\
1.9202 & 1.6617 & -4.9744 \\
1.9202 & 1.6617 & -4.9744 \\
1.9844 & 1.2099 & -5.1440 \\
1.9314 & 1.5028 & -5.0148 \\
1.9705 & 1.1160 & -5.1630 \\
1.9986 & 0.8951 & -5.2661 \\
1.9372 & 1.1519 & -5.1094 \\
1.8771 & 1.6064 & -5.0237 \\
1.9412 & 1.1352 & -5.2415 \\
1.9412 & 1.1352 & -5.2415 \\
1.8801 & 1.2678 & -5.0897 \\
1.8801 & 1.2678 & -5.0897 \\
1.9114 & 0.9000 & -5.2083 \\
1.8954 & 1.0183 & -5.1933 \\
1.8544 & 1.2168 & -5.2331 \\
1.8946 & 1.4635 & -5.3860 \\
1.5600 & -0.6043 & -4.4667 \\
1.8253 & 1.5506 & -5.2421 \\
1.7022 & 1.7049 & -5.1747 \\
1.7485 & 1.4782 & -5.3549 \\
1.7130 & 1.0602 & -5.3115 \\
1.6599 & 1.8476 & -5.1648 \\
1.6443 & 1.5572 & -5.2765 \\
1.6349 & 1.5167 & -5.3319 \\
1.6349 & 1.5167 & -5.3319 \\
1.5582 & 1.9284 & -5.1741 \\
1.5979 & 1.8639 & -5.3438 \\
1.0556 & 0.5506 & -3.8792 \\
1.3957 & 1.7496 & -5.4820 \\
1.4086 & 1.3930 & -5.5861 \\
1.3919 & 1.4541 & -5.6050 \\
1.3919 & 1.4541 & -5.6050 \\
1.3963 & 1.3268 & -5.6606 \\
1.4004 & 1.2838 & -5.7404 \\
1.3929 & 1.0999 & -5.7202 \\
1.3597 & 1.4090 & -5.6209 \\
1.3190 & 2.0371 & -5.4901 \\
1.3709 & 1.3966 & -5.7286 \\
1.3394 & 1.1223 & -5.7277 \\
1.3445 & 1.3163 & -5.7608 \\
1.3285 & 1.2199 & -5.8035 \\
1.2464 & 1.5262 & -5.7064 \\
1.2516 & 1.3025 & -5.9012 \\
1.2516 & 1.3025 & -5.9012 \\
0.8759 & 0.0585 & -4.3073 \\
1.1406 & 1.1611 & -5.8745 \\
1.0144 & 1.5619 & -5.6165 \\
0.8482 & 0.5714 & -4.7486 \\
1.0338 & 1.3690 & -5.8330 \\
0.7454 & 0.4045 & -4.2258 \\
0.5580 & -0.3899 & -3.2677 \\
0.8563 & 0.7405 & -5.0308 \\
0.9939 & 1.1248 & -5.9180 \\
0.8093 & 0.6069 & -4.8723 \\
0.8562 & 0.8203 & -5.3331 \\
0.9099 & 1.0621 & -5.7353 \\
0.9099 & 1.0621 & -5.7353 \\
0.6674 & 0.3247 & -5.3966 \\
0.2723 & 1.0908 & -5.0485 \\
0.2331 & 0.0262 & -4.3123 \\
0.2331 & 0.0262 & -4.3123 \\
0.2361 & 0.5793 & -4.6722 \\
0.2170 & 0.4654 & -4.5764 \\
0.2106 & 0.5446 & -4.6501 \\
0.2129 & 0.6015 & -4.6957 \\
0.1797 & -0.0199 & -4.2561 \\
0.2049 & 0.6323 & -4.7115 \\
0.1283 & 0.6120 & -4.7646 \\
0.1357 & 0.7321 & -4.9918 \\
0.1217 & 0.7722 & -4.9648 \\
0.1023 & 0.2246 & -4.6080 \\
0.0959 & 0.0772 & -4.3268 \\
0.0962 & 0.5161 & -4.6724 \\
0.0930 & 0.7599 & -4.9466 \\
0.0696 & 0.6231 & -4.9046 \\
0.0742 & 0.8031 & -5.0658 \\
-0.0138 & 0.7157 & -5.0309 \\
-0.0366 & 0.5771 & -4.7946 \\
-0.0913 & 0.0905 & -4.5050 \\
-0.1072 & 0.6447 & -4.8563 \\
-0.1039 & 0.4130 & -3.9999 \\
-0.1474 & 0.1445 & -4.6436 \\
-0.1789 & 0.6444 & -4.9258 \\
-0.1910 & -0.5364 & -4.5473 \\
-0.1867 & 0.4156 & -4.0211 \\
-0.2685 & -0.4913 & -4.4595 \\
-0.2443 & 0.4516 & -4.0289 \\
-0.2443 & 0.4516 & -4.0289 \\
-0.3633 & 0.5251 & -4.0914 \\
-0.4166 & -0.4782 & -4.4151 \\
-0.5094 & -0.2324 & -4.9730 \\
-0.4637 & 0.6447 & -4.1341 \\
-0.4637 & 0.6447 & -4.1341 \\
-0.5981 & -0.4677 & -4.5429
\end{pmatrix}
$$

Below is the recovered 3D points for image pair c):

$$\begin{pmatrix} -1.5488 & -2.5792 & 6.4992 \\ -0.1916 & -0.0134 & 1.8746 \\ 0.3011 & -0.0171 & 4.0552 \\ 0.4128 & 0.3982 & 4.6170 \\ 0.6677 & -0.0485 & 7.4419 \\ 0.9216 & -0.0605 & 9.0986 \\ 1.1831 & 0.0458 & 6.7987 \\ 2.0147 & -0.0043 & 10.2102 \\ -24.6015 & 0.7122 & -124.4006 \\ 2.0649 & 0.1723 & 10.3119 \\ 2.0649 & 0.1723 & 10.3119 \\ -3.6107 & -1.8099 & -17.4809 \\ -1.2634 & -0.3782 & -4.2203 \\ -0.8333 & 0.0447 & -2.4005 \\ -0.9292 & -0.2607 & -2.6080 \\ -0.7861 & 0.0421 & -2.1196 \\ -0.8958 & -0.2801 & -1.9537 \\ -0.9082 & -0.2070 & -1.9580 \end{pmatrix}$$