Student: Ninh DO
SID#**25949105**

Due: Mon, April 2nd, 2018, 11:55PM
Late: 1 day

CS 280 – Homework #3

Problem 1: Multi-View Reconstruction

1. `fundamental_matrix.py`: see Appendix

How to calculate F:

Step 1: Normalize points. The transformation matrix is:

$$T_j = \begin{bmatrix} 1/\sigma_x^{(j)} & 0 & -\dfrac{\mu_x^{(j)}}{\sigma_x^{(j)}} \\ 0 & 1/\sigma_y^{(j)} & -\dfrac{\mu_y^{(j)}}{\sigma_y^{(j)}} \\ 0 & 0 & 1 \end{bmatrix}$$

where, μ is mean, σ is standard variation, (j) is 1 or 2 corresponding to the set of points in first or second image.

Step 2: Rewrite $x_2^t F x_1 = 0$ into $Af = 0$, where f is formed from the entries of F stacked to a 9-element vector row-wise and A is a N x 9 dimensional matrix. In particular, the i-th row of A is equal to

$$A_i = \begin{bmatrix} x_1^{(i)} x_2^{(i)} & y_1^{(i)} x_2^{(i)} & x_2^{(i)} & x_1^{(i)} y_2^{(i)} & y_1^{(i)} y_2^{(i)} & y_2^{(i)} & x_1^{(i)} & y_1^{(i)} & 1 \end{bmatrix}$$

Step 3: Solve

$$\min_f \|Af\|_2 \quad s.t. \quad \|f\|_2 = 1$$

f is the right singular vector of A associated with the smallest singular value. Rearrange f into 3x3 matrix F^*.

Step 4: Solve

$$\min_F \|F - F^*\|_F \quad s.t. \quad rank(F) = 2$$

The solution is $F = U\hat{S}V^t$, where $F^* = USV^t$, $S = diag(s_1, s_2, s_3)$ with $s_1 > s_2 > s_3$, and $\hat{S} = diag(s_1, s_2, 0)$

Step 5: Denormalize F

$$F \leftarrow T_2^t F T_1$$

Results:

Sample Input 1: house
Fundamental matrix:

```
[[ -6.20566675e-08   1.45875690e-06   1.33031446e-04]
 [  6.45620291e-06  -5.56578733e-07  -1.65983285e-02]
 [ -1.04085120e-04   1.52331793e-02  -1.01024540e-02]]
```

Residual:
2.63803682452e-06
('Residual in F = %f', 2.6380368245245054e-06)

Sample Input 2: library
Fundamental matrix:
```
[[ -4.86210796e-08   9.98237781e-07  -1.47341026e-04]
 [ -5.80698246e-06  -5.65060960e-08   1.07254893e-02]
 [  1.38164930e-03  -9.63586245e-03  -2.60674702e-01]]
```
Residual:
1.01446714511e-05
('Residual in F = %f', 1.014467145111905e-05)

The residual is NOT what we are directly optimizing using SVD when solving the homogeneous system. The objective is the scaled distance between the points in the two images. Because the residual is the mean squared distance between the points, minimizing the scaled distance leading to minimizing the residual.

2. `find_rotation_translation.py`: see Appendix

Results:

Sample Input 1: house
Possible rotation matrices:
[array([[0.99473827, 0.02940012, -0.09813974],
       [0.03041134, -0.99949852, 0.00882362],
       [-0.0978311 , -0.01176176, -0.99513353]]), array([[0.98576115, 0.06875218, -0.15345389],
       [-0.07016042, 0.99752858, -0.00377408],
       [0.15281517, 0.01448673, 0.9881486]])]
Possible translation vector:
[array([-0.99941372, 0.02005846, -0.02774645]), array([0.99941372, -0.02005846, 0.02774645])]

Sample Input 2: library
Possible rotation matrices:
[array([[0.95733976, 0.02659659, -0.28773807],
       [-0.02579906, 0.9996456 , 0.00656393],
       [0.28781067, 0.00113946, 0.95768665]]), array([[0.91769622, 0.01543813, -0.39698276],
       [0.01559333, -0.99987439, -0.00283703],
       [-0.39697669, -0.00358675, -0.91782168]])]
Possible translation vector:
[array([-0.99829473, 0.00543367, 0.05812148]), array([0.99829473, -0.00543367, -0.05812148])]

3. `find_3d_points.py`: see Appendix

How to calculate the reconstruction error:

The reconstruction error is defined as the mean distance between the 2D points and the projected 3D points in the two images. So after we obtain the 3D points X_i's and the correct P2, we reconstruct the 2D points x_1, x_2 by:

$$x^{(1)}_{i\_reconstructed} = P_1 X_i \quad and \quad x^{(2)}_{i\_reconstructed} = P_2 X_i$$

then

$$rec\_err = \frac{1}{2n} \sum_{j=1}^{2} \sum_{i=1}^{n} \left\| x_i^{(j)} - x^{(j)}_{i\_reconstructed} \right\|_2$$

Results:

Sample Input 1: house
('Reconstruction error = %f', 0.23782105637693882)
Reconstruction error:
0.237821056377

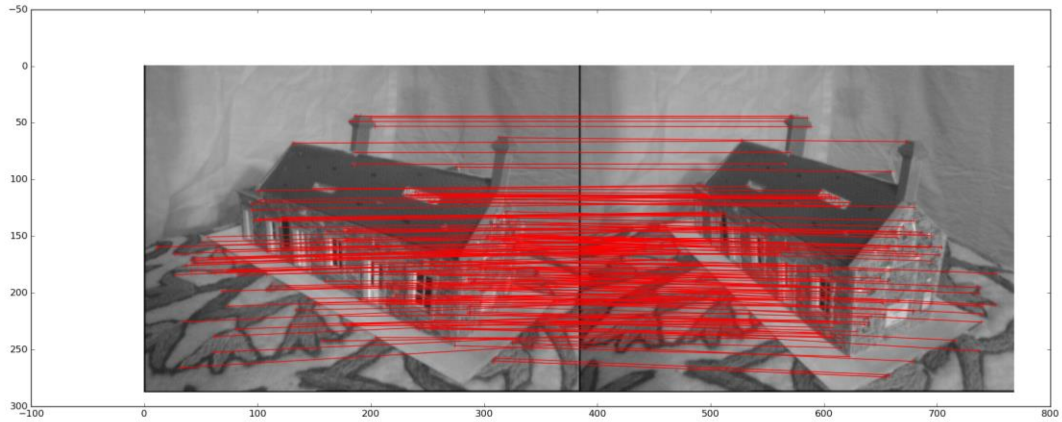
Sample Input 2: library
('Reconstruction error = %f', 0.3113685873637913)
Reconstruction error:
0.311368587364

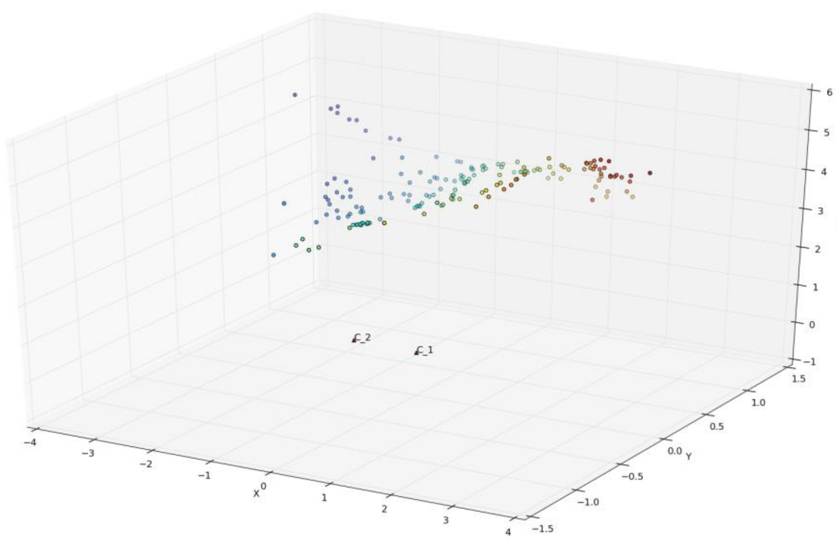
Student: Ninh DO
SID#**25949105**

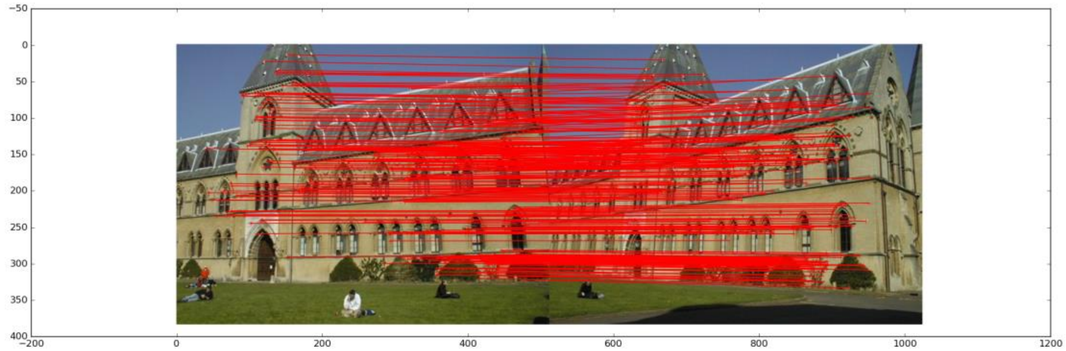
Due: Mon, April 2nd, 2018, 11:55PM
Late: 1 day

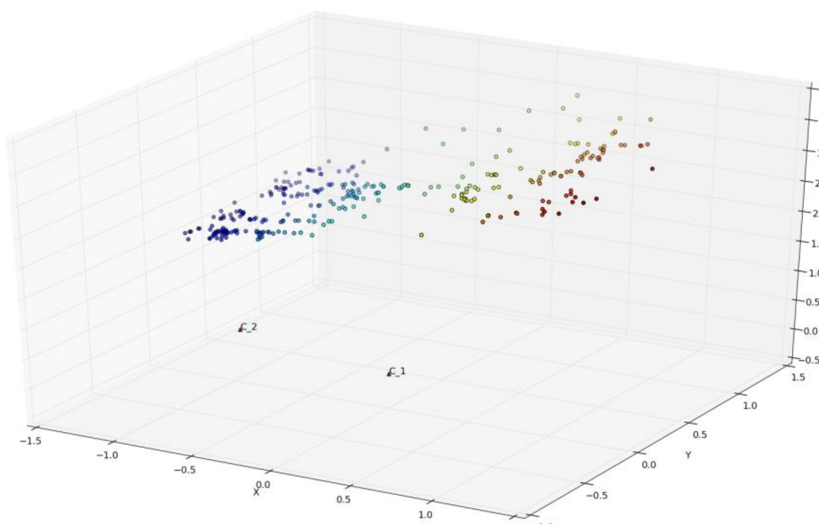
4. `plot_3d.py`: see Appendix

Results:








C_2
C_1
X
Y
Z

Appendix

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import scipy.io as scio
import sklearn.preprocessing as skpp
import numpy.linalg as nla
import sys


def reconstruct_3d(name):
    """ Homework 3: 3D reconstruction from two Views

    This function takes as input the name of the image pairs (i.e. 'house' or
    library') and returns the 3D points as well as the camera matrices... but
    some functions are missing.

    NOTES
    (1) The code has been written so that it can be easily understood. It has
    not been written for efficiency.
    (2) Don't make changes to this main function since I will run my
    reconstruct_3d.m and not yours. I only want from you the missing functions
    and they should be able to run without crashing with my reconstruct_3d.m
    (3) Keep the names of the missing functions as they are defined here,
    otherwise things will crash
    """

    ##--------Load images, K matrices and matches--------
    data_dir = "../data/" + name

    # images
    I1 = plt.imread(data_dir + '/' + name + "1.jpg")
    I2 = plt.imread(data_dir + '/' + name + "2.jpg")

    # K matrices
    qq = scio.loadmat(data_dir + '/' + name + "1_K.mat")
    K1 = qq['K']
    del qq
    qq = scio.loadmat(data_dir + '/' + name + "2_K.mat")
    K2 = qq['K']
    del qq

    # corresponding points
    matches = np.loadtxt(data_dir + '/' + name + "_matches.txt")
    # this is a N x 4 where:
```

```
# matches(i, 0:2) is a point in the first image
# matches(i, 2:) is the corresponding point in the second image

# visualize matches (disable or enable this whenever you want)
if True:
    plt.figure()
    plt.imshow(np.concatenate((I1, I2), axis=1))
    plt.plot(matches[:, 0], matches[:, 1], '+r')
    plt.plot(matches[:, 2] + I1.shape[1], matches[:, 3], '+r')

    for i in range(0, matches.shape[0]):
        X = np.concatenate((matches[i, [0]], matches[i, [2]] + I1.shape[1]))
        Y = matches[i, [1, 3]]
        plt.plot(X, Y, 'r')

    plt.show()

#-----------------------------------------------------------------------
##--------Find fundamental matrix--------

# F: the 3x3 fundamental matrix,
# res_err: mean squered distance between points in the two images and their
# corresponding epipolar lines
F, res_err = fundamental_matrix(matches);

print("Residual in F = %f", res_err)

# the essential matrix
E = K2.T.dot(F).dot(K1)

#-----------------------------------------------------------------------
##--------Rotation and translation of camera 2--------

# R: list of possible rotation matrices of second camera
# t: list of the possible translation vectors of second camera
R, t = find_rotation_translation(E)

# Find R2, t2 from R, t such that largest number of points lie in front of
# the image planes of the two cameras

P1 = K1.dot( np.concatenate( (np.identity(3), np.zeros( (3, 1) )), axis=1) )

# the number of points
num_points = np.zeros( (len(t), len(R)) )

# the reconstruction error for all combinations
```

```
    errs = np.full( (len(t), len(R)), np.inf )

    for it in range(len(t)):
        t2 = t[it]
        for ir in range(len(R)):
            R2 = R[ir]

            P2 = K2.dot( np.concatenate( (R2, t2.reshape(-1, 1)), axis=1 ) )

            points_3d, errs[it, ir] = find_3d_points(matches, P1, P2)

            Z1 = points_3d[:, 2]
            Z2 = R2[2, :].dot(points_3d.T) + t2[2]
            Z2 = Z2.T
            num_points[it, ir] = np.sum( np.logical_and(Z1 > 0, Z2 > 0) )

    its, irs = np.where( num_points == np.max(num_points) )

    # pick onw out the best combinations
    j = 0

    print("Reconstruction error = %f", errs[its[j], irs[j]])

    t2 = t[its[j]]
    R2 = R[irs[j]]
    P2 = K2.dot( np.concatenate( (R2, t2.reshape(-1, 1)), axis=1 ) )

    # compute the 3D points with the final P2
    points, _ = find_3d_points(matches, P1, P2)

    ##--------plot points and centers of cameras--------

    plot_3d(points, R2, t2)

    return points, P1, P2


def linear_transformation(X):
    means = np.mean(X, axis=1)
    standard_deviations = np.std(X, axis=1)
    standard_deviations[2] = -1
    transformation_matrix = np.diag(1 / standard_deviations)
    transformation_matrix[:, 2] = (-1) * means / standard_deviations
    X_standardized = np.dot(transformation_matrix, X)
    return X_standardized, transformation_matrix
```

```
def fundamental_matrix(matches):
    n = matches.shape[0]
    X1 = np.concatenate((matches[:, [0,1]].T, np.ones((1, n))), axis=0)
    X2 = np.concatenate((matches[:, [2,3]].T, np.ones((1, n))), axis=0)
    X1_standardized, T1 = linear_transformation(X1)
    X2_standardized, T2 = linear_transformation(X2)
    A = np.concatenate((X1_standardized[[0], :] * X2_standardized[[0], :],
                        X1_standardized[[1], :] * X2_standardized[[0], :],
                        X2_standardized[[0], :],
                        X1_standardized[[0], :] * X2_standardized[[1], :],
                        X1_standardized[[1], :] * X2_standardized[[1], :],
                        X2_standardized[[1], :],
                        X1_standardized[[0], :],
                        X1_standardized[[1], :],
                        np.ones((1, n))), axis=0)
    A = A.T
    U_A, s_A, V_A_transpose = nla.svd(A)
    f = V_A_transpose.T[:, -1]
    F_fullrank = f.reshape(3, 3)
    U_F, s_F, V_F_transpose = nla.svd(F_fullrank)
    s_F[-1] = 0
    F_rank2 = U_F.dot(np.diag(s_F)).dot(V_F_transpose)
    F_ret = T2.T.dot(F_rank2).dot(T1)

    residual = 0
    for i in range(n):
        x1 = X1[:, i]
        x2 = X2[:, i]
        residual += x1.dot(F_ret.T).dot(x2) ** 2 / nla.norm(F_ret.T.dot(x2)) ** 2 \
                + x2.dot(F_ret).dot(x1) ** 2 / nla.norm(F_ret.dot(x1)) ** 2

    residual /= 2 * n

    print("Fundamental matrix:")
    print(F_ret)
    print("Residual:")
    print(residual)

    return F_ret, residual


def find_rotation_translation(E):
    U, s, V_transpose = nla.svd(E)
    t = U[:, -1]
    t_list = [t, -t]
```

```
    R_pos_90_degree = np.array([[0, -1, 0],
                                [1,  0, 0],
                                [0,  0, 1]])
    R_neg_90_degree = np.array([[ 0, 1, 0],
                                [-1, 0, 0],
                                [ 0, 0, 1]])
    R_list = []
    R1 = U.dot(R_pos_90_degree.T).dot(V_transpose)
    if abs(nla.det(R1) - 1) <= 1e-5:
        R_list.append(R1)

    R2 = R1 * (-1)
    if abs(nla.det(R2) - 1) <= 1e-5:
        R_list.append(R2)

    R3 = U.dot(R_neg_90_degree.T).dot(V_transpose)
    if abs(nla.det(R3) - 1) <= 1e-5:
        R_list.append(R3)

    R4 = R3 * (-1)
    if abs(nla.det(R4) - 1) <= 1e-5:
        R_list.append(R4)

    print("Possible rotation matrices:")
    print(R_list)
    print("Possible translation vector:")
    print(t_list)

    return R_list, t_list


def find_3d_points(matches, P1, P2):
    n = matches.shape[0]
    points = np.ones( (n, 4) )
    P_pair = [P1, P2]

    for i in range(n):
        x_pair = [matches[i, 0:2], matches[i, 2:]]

        A = []
        b = []
        for j in range(2):
            P = P_pair[j]
            x = x_pair[j]
            A.append( [ P[0, 0] - x[0] * P[2, 0],
```

```
                P[0, 1] - x[0] * P[2, 1],
                P[0, 2] - x[0] * P[2, 2] ] )
        A.append( [ P[1, 0] - x[1] * P[2, 0],
                P[1, 1] - x[1] * P[2, 1],
                P[1, 2] - x[1] * P[2, 2] ] )
        b.append(x[0] * P[2, 3] - P[0, 3])
        b.append(x[1] * P[2, 3] - P[1, 3])

        A = np.array(A)
        b = np.array(b)
        points[i, :-1] = nla.inv( A.T.dot(A) ).dot(A.T).dot(b)

    rec_err = 0
    for i in range(n):
        x1 = matches[i, 0:2]
        x2 = matches[i, 2:]
        x1_rec_homo = P1.dot(points[i, :])
        x1_rec_homo /= x1_rec_homo[-1]
        x1_rec = x1_rec_homo[:-1]
        x2_rec_homo = P2.dot(points[i, :])
        x2_rec_homo /= x2_rec_homo[-1]
        x2_rec = x2_rec_homo[:-1]
        rec_err += nla.norm(x1 - x1_rec) + nla.norm(x2 - x2_rec)

    rec_err /= 2 * n

    print("Reconstruction error:")
    print(rec_err)

    return points[:, :-1], rec_err


def plot_3d(points, R2, t2):
    C1 = np.zeros(3)
    C2 = -nla.inv(R2).dot(t2)

    ax = plt.axes(projection='3d')
    ax.scatter(points[:, 0], points[:, 1], points[:, 2], c=points[:, 0])

    ax.scatter(C1[0], C1[1], C1[2], c='r', marker='^')
    ax.text(C1[0], C1[1], C1[2], 'C_1')
    ax.scatter(C2[0], C2[1], C2[2], c='r', marker='^')
    ax.text(C2[0], C2[1], C2[2], 'C_2')

    ax.set_xlabel('X')
    ax.set_ylabel('Y')
```

```
    ax.set_zlabel('Z')

    plt.show()


if __name__ == "__main__":

    if len(sys.argv) > 1:
        reconstruct_3d(sys.argv[1])
    else:
        print("Sample Input 1: house")
        reconstruct_3d("house")
        print()
        print("Sample Input 2: library")
        reconstruct_3d("library")


##-----------------------------------------------------------------------------##
## Output

$ python reconstruct_3d.py
Sample Input 1: house
Fundamental matrix:
[[ -6.20566675e-08   1.45875690e-06   1.33031446e-04]
 [  6.45620291e-06  -5.56578733e-07  -1.659832855e-02]
 [ -1.04085120e-04   1.52331793e-02  -1.01024540e-02]]
Residual:
2.63803682452e-06
('Residual in F = %f', 2.638036824524 5054e-06)
Possible rotation matrices:
[array([[ 0.99473827,  0.02940012, -0.09813974],
       [ 0.03041134, -0.99949852,  0.00882362],
       [-0.0978311 , -0.01176176, -0.99513353]]), array([[ 0.98576115,  0.06875218, -
0.15345389],
       [-0.07016042,  0.99752858, -0.00377408],
       [ 0.15281517,  0.01448673,  0.9881486 ]])]
Possible translation vector:
[array([-0.99941372,  0.02005846, -0.02774645]), array([ 0.99941372, -0.02005846,
0.02774645])]
Reconstruction error:
0.237921196992
Reconstruction error:
0.237821056377
Reconstruction error:
0.237921196992
Reconstruction error:
```

```
0.237821056377
('Reconstruction error = %f', 0.23782105637693882)
Reconstruction error:
0.237821056377
()
Sample Input 2: library
Fundamental matrix:
[[ -4.86210796e-08   9.98237781e-07  -1.47341026e-04]
 [ -5.80698246e-06  -5.65060960e-08   1.07254893e-02]
 [  1.38164930e-03  -9.635862450-03  -2.60674702e-01]]
Residual:
1.01446714511e-05
('Residual in F = %f', 1.014467145111905e-05)
Possible rotation matrices:
[array([[ 0.95733976,  0.02659659, -0.28773807],
      [-0.02579906,  0.9996456 ,  0.00656393],
      [ 0.28781067,  0.00113946,  0.95768665]]), array([[ 0.91769622,  0.01543813, -
0.39698276],
      [ 0.01559333, -0.99987439, -0.00283703],
      [-0.39697669, -0.00358675, -0.91782168]])]
Possible translation vector:
[array([-0.99829473,  0.00543367,  0.05812148]), array([ 0.99829473, -0.00543367, -
0.05812148])]
Reconstruction error:
0.311368587364
Reconstruction error:
0.311711926583
Reconstruction error:
0.311368587364
Reconstruction error:
0.311711926583
('Reconstruction error = %f', 0.3113685873637913)
Reconstruction error:
0.311368587364
```