In [148]:

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

##
## load images and match files for the first example
##

I1 = Image.open('MP4_part2_data/library1.jpg')
I2 = Image.open('MP4_part2_data/library2.jpg')
matches = np.loadtxt('MP4_part2_data/library_matches.txt')

# this is a N x 4 file where the first two numbers of each row
# are coordinates of corners in the first image and the last two
# are coordinates of corresponding corners in the second image:
# matches(i,1:2) is a point in the first image
# matches(i,3:4) is a corresponding point in the second image

N = len(matches)

##
## display two images side-by-side with matches
## this code is to help you visualize the matches, you don't need
## to use it to produce the results for the assignment
##

I3 = np.zeros((I1.size[1], I1.size[0] * 2, 3))
I3[:, :I1.size[0], :] = I1
I3[:, I1.size[0]:, :] = I2
fig, ax = plt.subplots()
ax.set_aspect('equal')
ax.imshow(np.array(I3).astype(int))
ax.plot(matches[:, 0], matches[:, 1], '+r')
ax.plot(matches[:, 2] + I1.size[0], matches[:, 3], '+r')
ax.plot([matches[:, 0], matches[:, 2] + I1.size[0]], [matches[:, 1], matches[:, 3
plt.show()
```

...

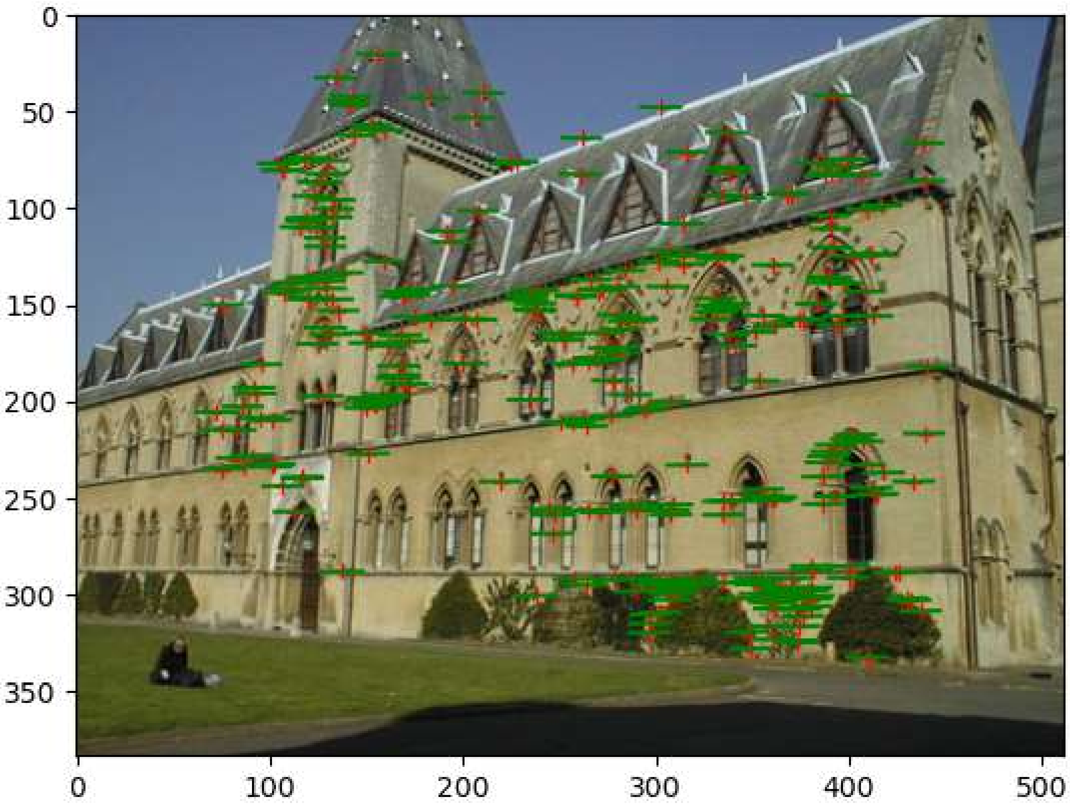
In [149]:

```python
def fit_fundamental(matches, normalize=False):
    if normalize:
        matches, T1, T2 = normalize_matches(matches)

    n = len(matches)
    rows = np.zeros((n, 9))
    for i in range(n):
        u1, v1 = matches[i, 0: 2]
        u2, v2 = matches[i, 2: 4]
        rows[i] = [u1 * u2, v1 * u2, u2, u1 * v2, v1 * v2, v2, u1, v1, 1]

    U, s, V = np.linalg.svd(rows)
    F = V[len(V) - 1].reshape(3, 3)

    # enforce rank 2
    U, s, V = np.linalg.svd(F)
    new_s = np.diag(s)
    new_s[-1] = 0
    new_F = np.dot(U, np.dot(new_s, V))
    if normalize:
        new_F = np.dot(np.dot(T2.T, new_F), T1)
    return new_F


def normalize_matches(matches):
    mean = np.mean(matches, axis=0)
    matches_mean = matches - mean
    total1 = 0
    total2 = 0
    for i in range(len(matches)):
        total1 += matches_mean[i, 0] ** 2 + matches_mean[i, 1] ** 2
        total2 += matches_mean[i, 2] ** 2 + matches_mean[i, 3] ** 2

    std1 = np.sqrt(total1 / (2 * N))
    std2 = np.sqrt(total2 / (2 * N))

    for i in range(len(matches)):
        matches_mean[i, 0] = matches_mean[i, 0] / std1
        matches_mean[i, 1] = matches_mean[i, 1] / std1
        matches_mean[i, 2] = matches_mean[i, 2] / std2
        matches_mean[i, 3] = matches_mean[i, 3] / std2

    matches = matches_mean
    T1 = np.array([[1 / std1, 0, -1 * (1 / std1) * mean[0]],
                   [0, 1 / std1, -1 * (1 / std1) * mean[1]],
                   [0, 0, 1]])
    T2 = np.array([[1 / std2, 0, -1 * (1 / std2) * mean[2]],
                   [0, 1 / std2, -1 * (1 / std2) * mean[3]],
                   [0, 0, 1]])

    return matches, T1, T2
```

In [150]:

```
##
## display second image with epipolar lines reprojected
## from the first image
##
# first, fit fundamental matrix to the matches
F = fit_fundamental(matches, True)  # this is a function that you should write
M = np.c_[matches[:, 0:2], np.ones((N, 1))].transpose()
L1 = np.matmul(F, M).transpose()  # transform points from
# the first image to get epipolar lines in the second image

# find points on epipolar lines L closest to matches(:,3:4)
l = np.sqrt(L1[:, 0] ** 2 + L1[:, 1] ** 2)
L = np.divide(L1, np.kron(np.ones((3, 1)), l).transpose())  # rescale the line
pt_line_dist = np.multiply(L, np.c_[matches[:, 2:4], np.ones((N, 1))]).sum(axis=1
closest_pt = matches[:, 2:4] - np.multiply(L[:, 0:2], np.kron(np.ones((2, 1)), pt

residual = np.mean(np.abs(pt_line_dist))
print('Residual = {}'.format(residual))

# find endpoints of segment on epipolar line (for display purposes)
pt1 = closest_pt - np.c_[L[:, 1], -L[:, 0]] * 10  # offset from the closest point
pt2 = closest_pt + np.c_[L[:, 1], -L[:, 0]] * 10

# display points and segments of corresponding epipolar lines
fig, ax = plt.subplots()
ax.set_aspect('equal')
ax.imshow(np.array(I2).astype(int))
ax.plot(matches[:, 2], matches[:, 3], '+r')
ax.plot([matches[:, 2], closest_pt[:, 0]], [matches[:, 3], closest_pt[:, 1]], 'r
ax.plot([pt1[:, 0], pt2[:, 0]], [pt1[:, 1], pt2[:, 1]], 'g')
plt.show()
```

```
Residual = 0.18359661716194822
```



In [151]:

```python
## Camera Calibration

def evaluate_points(M, points_2d, points_3d):
    """
    Visualize the actual 2D points and the projected 2D points calculated from
    the projection matrix
    You do not need to modify anything in this function, although you can if you
    want to
    :param M: projection matrix 3 x 4
    :param points_2d: 2D points N x 2
    :param points_3d: 3D points N x 3
    :return:
    """
    N = len(points_3d)
    points_3d = np.hstack((points_3d, np.ones((N, 1))))
    points_3d_proj = np.dot(M, points_3d.T).T
    u = points_3d_proj[:, 0] / points_3d_proj[:, 2]
    v = points_3d_proj[:, 1] / points_3d_proj[:, 2]
    residual = np.sum(np.hypot(u - points_2d[:, 0], v - points_2d[:, 1]))
    points_3d_proj = np.hstack((u[:, np.newaxis], v[:, np.newaxis]))
    return points_3d_proj, residual
```

In [152]:

```
def find_camera_projection_matrix(points_2d, points_3d):
    N = len(points_3d)
    rows = np.zeros((N, 12))
    for i in range(int(N / 2)):
        x = np.array([points_3d[i, 0], points_3d[i, 1], points_3d[i, 2], 1])
        rows[i * 2, 4:8] = x
        rows[i * 2, 8:12] = -1 * points_2d[i, 1] * x
        rows[i * 2 + 1, 0:4] = x
        rows[i * 2 + 1, 8:12] = -1 * points_2d[i, 0] * x

    U, s, V = np.linalg.svd(rows)
    F = V[len(V) - 1].reshape((3, 4))

    return F
```

In [153]:

```
matches = np.loadtxt('MP4_part2_data/lab_matches.txt')
lab_3d = np.loadtxt('MP4_part2_data/lab_3d.txt')
match1 = matches[:, :2]
match2 = matches[:, 2:]

matrix1 = find_camera_projection_matrix(match1, lab_3d)
matrix2 = find_camera_projection_matrix(match2, lab_3d)
print('matrix1 = \n{}'.format(matrix1))
print()
print('matrix2 = \n{}'.format(matrix2))

proj1, residual1 = evaluate_points(matrix1, match1, lab_3d)
proj2, residual2 = evaluate_points(matrix2, match2, lab_3d)
print('Residual1 = {}'.format(residual1))
print('Residual2 = {}'.format(residual2))
```

```
matrix1 =
[[ 3.09971524e-03  1.46250174e-04 -4.48354919e-04 -9.78974905e-01]
 [ 3.06744636e-04  6.36810842e-04 -2.77389022e-03 -2.03932211e-01]
 [ 1.67995219e-06  2.74565792e-06 -6.83395792e-07 -1.32842138e-03]]

matrix2 =
[[-6.88970692e-03  3.96429852e-03  1.39263702e-03  8.28289829e-01]
 [-1.53909600e-03 -1.02084411e-03  7.22962251e-03  5.60181867e-01]
 [-7.58603647e-06 -3.72293087e-06  2.03836990e-06  3.38133189e-03]]
Residual1 = 13.765505109334178
Residual2 = 17.781125905791964
```

In [154]:

```
## Camera Centers
def find_camera_center(matrix):
    U, s, V = np.linalg.svd(matrix)
    center = V.T[:, -1]
    return center / center[-1]


lab1_matrix = matrix1
lab2_matrix = matrix2
library1_matrix = np.loadtxt('MP4_part2_data/library1_camera.txt')
library2_matrix = np.loadtxt('MP4_part2_data/library2_camera.txt')

lab1_center = find_camera_center(lab1_matrix)
lab2_center = find_camera_center(lab2_matrix)
library1_center = find_camera_center(library1_matrix)
library2_center = find_camera_center(library2_matrix)
print("lab1_center = {}".format(lab1_center))
print("lab2_center = {}".format(lab2_center))
print("library1_center = {}".format(library1_center))
print("library2_center = {}".format(library2_center))
```

```
lab1_center = [305.83387882 304.20073002  30.13782356   1.        ]
lab2_center = [303.14001018 307.21161306  30.4296492    1.        ]
library1_center = [  7.28863053 -21.52118112  17.73503585   1.        ]
library2_center = [  6.89405488 -15.39232716  23.41498687   1.        ]
```

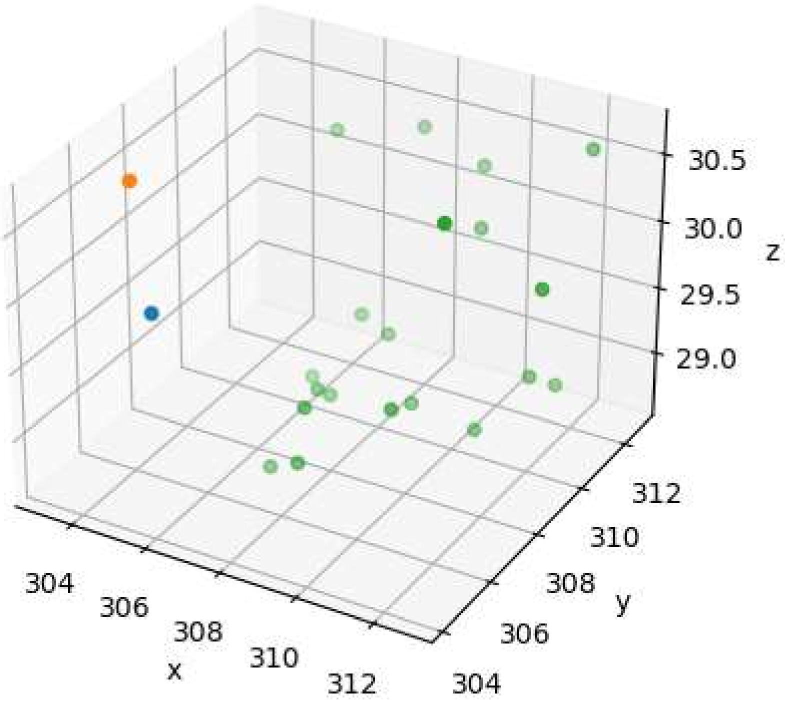
In [155]:

```
## Triangulation
def triangulate(m1, m2, matches):
    N = len(matches)

    attach = np.ones((N, 1))
    x1 = np.hstack((matches[:, : 2], attach))
    x2 = np.hstack((matches[:, 2:], attach))

    X_3d = np.zeros((N, 4))
    for i in range(N):
        x_1 = np.array([[0, -x1[i, 2], x1[i, 1]],
                        [x1[i, 2], 0, -x1[i, 0]],
                        [-x1[i, 1], x1[i, 0], 0]])
        x_2 = np.array([[0, -x2[i, 2], x2[i, 1]],
                        [x2[i, 2], 0, -x2[i, 0]],
                        [-x2[i, 1], x2[i, 0], 0]])
        A1 = x_1.dot(m1)
        A2 = x_2.dot(m2)
        A = np.vstack((A1, A2))

        U, s, V = np.linalg.svd(A)
        t = V[len(V) - 1]
        X_3d[i] = t / t[-1]

    return X_3d


def get_residual(m1, m2, X_3d, matches):
    x1 = np.dot(m1, X_3d.T).T
    x1 = x1 / x1[:, -1][:, np.newaxis]
    x2 = np.dot(m2, X_3d.T).T
    x2 = x2 / x2[:, -1][:, np.newaxis]

    res1 = np.linalg.norm(x1[:, 0:2] - matches[:, 0:2]) ** 2
    res2 = np.linalg.norm(x2[:, 0:2] - matches[:, 2:4]) ** 2

    residual1 = res1 / matches.shape[0]
    residual2 = res2 / matches.shape[0]

    return residual1, residual2


def plot(center1, center2, X_3d):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.scatter(center1[0], center1[1], center1[2])
    ax.scatter(center2[0], center2[1], center2[2])
    ax.scatter(X_3d[:, 0], X_3d[:, 1], X_3d[:, 2])

    # ax.view_init(40, 60)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('z')

    plt.show()
```

In [156]:

```
lab_matches = np.loadtxt('MP4_part2_data/lab_matches.txt')
X_3d = triangulate(lab1_matrix, lab2_matrix, lab_matches)
print(X_3d)

avg_res1, avg_res2 = get_residual(lab1_matrix, lab2_matrix, X_3d, lab_matches)
print("residual1 = {}".format(avg_res1))
print("residual2 = {}".format(avg_res2))

plot(lab1_center, lab2_center, X_3d)
```

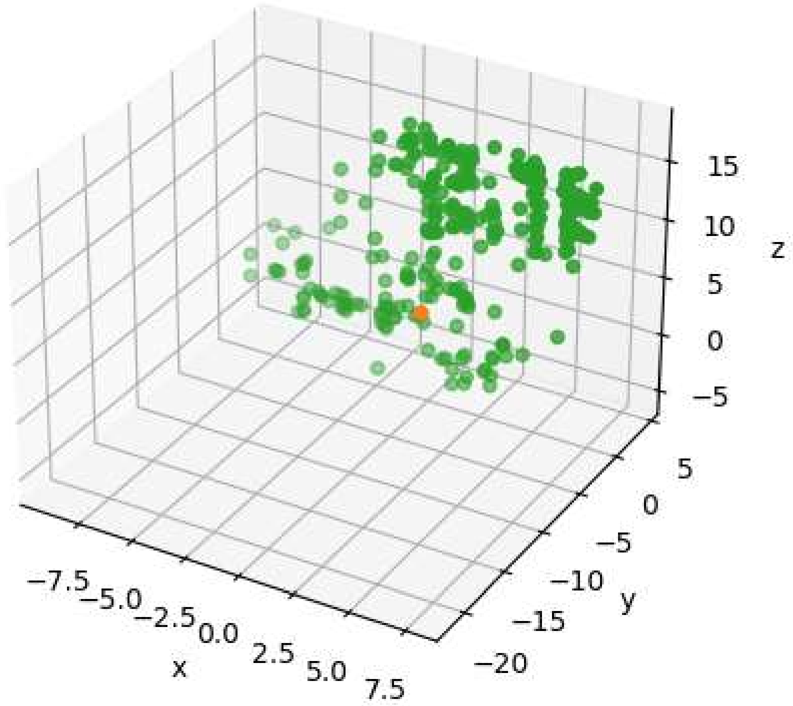
```
[[312.78291956 309.14793677  30.08825937   1.        ]
 [305.79583985 311.647791    30.35938704   1.        ]
 [307.7005947  312.37217915  30.41674895   1.        ]
 [310.13440522 307.17943144  29.30057835   1.        ]
 [311.95788273 310.12008416  29.21275648   1.        ]
 [311.20691609 307.57555977  30.67950481   1.        ]
 [307.10774467 306.88091469  28.65793142   1.        ]
 [309.28133295 312.4486935   30.23054336   1.        ]
 [307.43935436 310.15100191  29.31303864   1.        ]
 [308.24268375 306.29876816  28.88661985   1.        ]
 [306.64235566 309.2992755   28.90589268   1.        ]
 [308.0651248  306.84106044  29.19401194   1.        ]
 [309.64168792 308.81128988  29.03262552   1.        ]
 [308.27460678 309.9739103   29.25778192   1.        ]
 [307.58012218 308.62549965  28.95332987   1.        ]
 [311.08075193 309.20605436  28.88932969   1.        ]
 [307.52743213 308.18738654  29.06369628   1.        ]
 [309.93933906 311.25779992  29.99069135   1.        ]
 [312.24541544 310.81511246  29.05823813   1.        ]
 [311.98515006 312.70461737  30.51474434   1.        ]]
residual1 = 2.9569700754132855
residual2 = 0.08726077273043897
```



In [157]:

```
library_matches = np.loadtxt('MP4_part2_data/library_matches.txt')
X_3d = triangulate(library1_matrix, library2_matrix, library_matches)
print(X_3d)

res1, res2 = get_residual(library1_matrix, library2_matrix, X_3d, library_matches
print("residual1 = {}".format(res1))
print("residual2 = {}".format(res2))

plot(library1_center, library1_center, X_3d)
```

```
[[-0.6206525  -0.31364979 15.68369601  1.        ]
 [-1.11109351 -0.06468257 12.78944681  1.        ]
 [-3.79848506 -0.42359058 -0.83821848  1.        ]
 ...
 [-2.53006181 -0.16498487  6.11131626  1.        ]
 [-2.35168652 -0.20471517 12.96876516  1.        ]
 [-2.2049168  -0.50266396 15.25294161  1.        ]]
residual1 = 0.07312796424284178
residual2 = 0.2676795126176233
```



In [ ]:

```
import cv2
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial import distance
import random


def sift_descriptors(img):
    sift = cv2.xfeatures2d.SIFT_create()
    keypoints, descriptor = sift.detectAndCompute(img, None)
    return keypoints, descriptor


def get_matched_pixels(threshold, kp1, kp2, desp1, desp2):
    dist = distance.cdist(desp1, desp2, 'sqeuclidean')
    index1 = np.where(dist < threshold)[0]
    index2 = np.where(dist < threshold)[1]
    coord1 = np.array([kp1[idx].pt for idx in index1])
    coord2 = np.array([kp2[idx].pt for idx in index2])
    match_coords = np.concatenate((coord1, coord2), axis=1)

    return match_coords


def fit_homography(pairs):
    rows = []

    for i in range(pairs.shape[0]):
        p1 = np.append(pairs[i][0:2], 1)
        p2 = np.append(pairs[i][2:4], 1)

        row1 = [0, 0, 0, p1[0], p1[1], p1[2], -p2[1] * p1[0], -p2[1] * p1[1], -p2
        row2 = [p1[0], p1[1], p1[2], 0, 0, 0, -p2[0] * p1[0], -p2[0] * p1[1], -p2
        rows.append(row1)
        rows.append(row2)

    a = np.array(rows)

    U, s, V = np.linalg.svd(a)
    H = V[len(V) - 1].reshape(3, 3)
    H = H / H[2, 2]

    return H


def get_errors(matches, F):
    ones = np.ones((matches.shape[0], 1))
    p1 = np.concatenate((matches[:, 0:2], ones), axis=1)
    p2 = np.concatenate((matches[:, 2:4], ones), axis=1)

    F_p1 = np.dot(F, p1.T).T
    F_p2 = np.dot(F.T, p2.T).T

    p1_line2 = np.sum(p1 * F_p2, axis=1)[:, np.newaxis]
    p2_line1 = np.sum(p2 * F_p1, axis=1)[:, np.newaxis]
    d1 = np.absolute(p1_line2) / np.linalg.norm(F_p2, axis=1)[:, np.newaxis]
    d2 = np.absolute(p2_line1) / np.linalg.norm(F_p1, axis=1)[:, np.newaxis]

    return (d1 + d2) / 2


def normalize_matches(matches):
    mean = np.mean(matches, axis=0)
    matches_mean = matches - mean
    total1 = 0
    total2 = 0
    for i in range(len(matches)):
        total1 += matches_mean[i, 0] ** 2 + matches_mean[i, 1] ** 2
        total2 += matches_mean[i, 2] ** 2 + matches_mean[i, 3] ** 2
    N = len(matches)
    std1 = np.sqrt(total1 / (2 * N))
    std2 = np.sqrt(total2 / (2 * N))

    for i in range(len(matches)):
        matches_mean[i, 0] = matches_mean[i, 0] / std1
        matches_mean[i, 1] = matches_mean[i, 1] / std1
        matches_mean[i, 2] = matches_mean[i, 2] / std2
        matches_mean[i, 3] = matches_mean[i, 3] / std2

    matches = matches_mean
    T1 = np.array([[1 / std1, 0, -1 * (1 / std1) * mean[0]],
                   [0, 1 / std1, -1 * (1 / std1) * mean[1]],
                   [0, 0, 1]])
    T2 = np.array([[1 / std2, 0, -1 * (1 / std2) * mean[2]],
                   [0, 1 / std2, -1 * (1 / std2) * mean[3]],
                   [0, 0, 1]])

    return matches, T1, T2


def fit_fundamental(matches, normalize=False):
    if normalize:
        matches, T1, T2 = normalize_matches(matches)

    n = len(matches)
    rows = np.zeros((n, 9))
    for i in range(n):
        u1, v1 = matches[i, 0: 2]
        u2, v2 = matches[i, 2: 4]
        rows[i] = [u1 * u2, v1 * u2, u2, u1 * v2, v1 * v2, v2, u1, v1, 1]

    U, s, V = np.linalg.svd(rows)
    F = V[len(V) - 1].reshape(3, 3)

    # enforce rank 2
    U, s, V = np.linalg.svd(F)
    new_s = np.diag(s)
    new_s[-1] = 0
    new_F = np.dot(U, np.dot(new_s, V))
    if normalize:
        new_F = np.dot(np.dot(T2.T, new_F), T1)
    return new_F
```

```python
def ransac_fitting(match_coords, threshold):
    max_iteration = 1000
    num_best_inliers = 0
    avg_residual = 0

    for i in range(max_iteration):
        F = fit_fundamental(match_coords, normalize=True)

        errors = get_errors(match_coords, F)
        idx = np.where(errors < threshold)[0]
        inliers = match_coords[idx]

        num_inliers = len(inliers)
        if num_inliers > num_best_inliers:
            best_inliers = inliers.copy()
            num_best_inliers = num_inliers
            best_F = F.copy()

            avg_residual = errors[idx].sum() / num_best_inliers

    return best_inliers, best_F, avg_residual


def plot_inlier_matches(inliers, root, img_name):
    I1 = Image.open(root + 'MP4_part2_data/' + img_name + '1.jpg').convert('L')
    I2 = Image.open(root + 'MP4_part2_data/' + img_name + '2.jpg').convert('L')
    I3 = np.zeros((I1.size[1], I1.size[0] * 2))
    I3[:, :I1.size[0]] = I1
    I3[:, I1.size[0]:] = I2
    fig, ax = plt.subplots()
    ax.set_aspect('equal')
    ax.imshow(np.array(I3).astype(float), cmap='gray')
    ax.plot(inliers[:, 0], inliers[:, 1], '+r')
    ax.plot(inliers[:, 2] + I1.size[0], inliers[:, 3], '+r')
    ax.plot([inliers[:, 0], inliers[:, 2] + I1.size[0]], [inliers[:, 1], inliers[
            'r', linewidth=0.4)
    plt.axis('off')
    plt.savefig(root + 'outputs/' + img_name + '_inlier_matches.jpg', format='jpg
    plt.show()


if __name__ == '__main__':
    root = './'
    im_name = 'gaudi'
    I1 = Image.open('./MP4_part2_data/' + im_name + '1.jpg').convert('L')
    I2 = Image.open('./MP4_part2_data/' + im_name + '2.jpg').convert('L')

    im1 = np.array(I1)
    im2 = np.array(I2)
    t_match = 20000
    t_ransac = 0.4

    kp1, desp1 = sift_descriptors(im1)
    kp2, desp2 = sift_descriptors(im2)
    match_coords = get_matched_pixels(t_match, kp1, kp2,
                                      desp1, desp2)

    inliers, F, avg_residual = ransac_fitting(match_coords, t_ransac)
    print("Number of inliers: {}, Average residual: {}".format(len(inliers), avg_

    plot_inlier_matches(inliers, root, im_name)
```

In [ ]: