1 CS4277/CS5477 Lab 4-1: Relative pose estimation with 8-point algorithm

1.0.1 Introduction

In this part, you will get to estimate the essential and fundamental matrix by using eight point algorithm. As discussed in the lecture, images taken from different views should fulfill the epipolar constraint, which can be used to estimate the fundamental and essential matrix. You will first estimate the fundamental and essential matrix with 15 correspondences provided in the dataset. Then you will decompose the essential matrix to find rotation and translation between two views. The decomposition will give 4 feasible camera poses and you will select the the correct pose by chriality check.

Honour Code. The coding assignment (Lab4-1 and Lab4-2) constitutes 10% of your final grade in CS4277/CS5477. Note that plagiarism will not be condoned! You may discuss with your classmates and check the internet for references, but you MUST NOT submit code/report that is copied directly from other sources!

References: * Lecture 7

Optional references: * Richard I. Hartley. In Defence of the 8-point Algorithm

1.0.2 Instructions

This workbook provides the instructions for the assignment, and facilitates the running of your code and visualization of the results. For each part of the assignment, you are required to **complete the implementations of certain functions in the accompanying python file** (`eight_point.py`).

To facilitate implementation and grading, all your work is to be done in that file, and **you only have to submit the .py file**.

Please note the following: 1. Fill in your name, email, and NUSNET ID at the top of the python file. 2. The parts you need to implement are clearly marked with the following:

```
```
""" YOUR CODE STARTS HERE """

""" YOUR CODE ENDS HERE """
```


, and you should write your code in between the above two lines.

3. Note that for each part, there may certain functions that are prohibited to be used. It is important **NOT to use those prohibited functions** (or other functions with similar functionality). If you are unsure whether a particular function is allowed, feel free to ask any of the TAs.

### 1.0.3 Submission Instructions

Items to be submitted:
```

• Source code (eight_point.py and pnp.py). These are where you fill in all your codes for each part.

• Report (report.pdf). This should describe your implementation and be no more than one page.

Please clearly indicate your name and student number (the one that looks like A1234567X) in the report as well as the top of your source code. Zip the three files together and name it in the following format: A1234567X_lab4.zip (replace with your student number).

Submit your assignment by 23 March 2021, 2359HRS to LumiNUS. 25% of the total score will be deducted for each day of late submission.

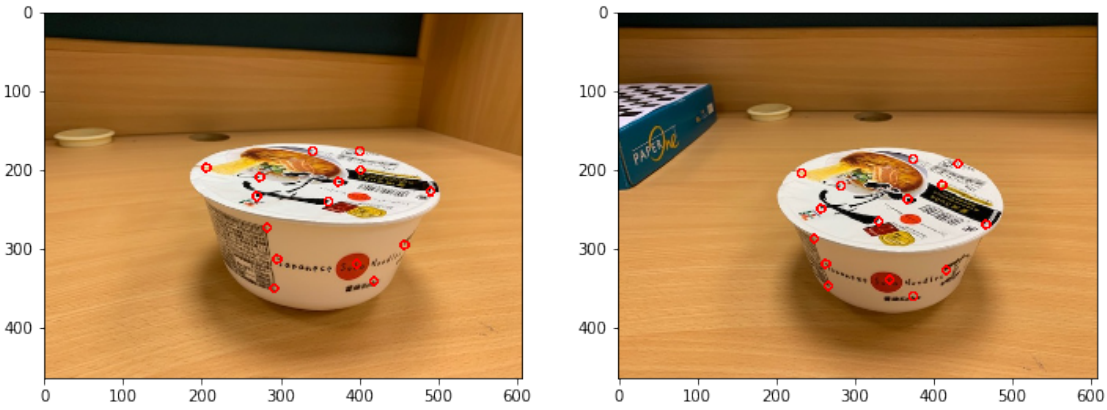
1.1 Load and Visualize Data

In this part, you will get yourself familiar with the data by visualizing it. The data includes two images of the same object (im1.jpg and img2.jpg) and 15 correpondences (correspondences.mat). You can visualize the data with the provided code below.

[ ]:
```
import cv2
import matplotlib.pyplot as plt
import numpy as np
import h5py
import scipy.io as sio
from eight_point import compute_fundamental, compute_essential, decompose_e
%matplotlib inline
```

[2]:
```
correspondences = sio.loadmat('data/correspondences_ud')
data1_ori = correspondences['movingPoints']
data2_ori = correspondences['fixedPoints']
data1 = np.concatenate([data1_ori.T, np.ones((1, data1_ori.shape[0]))], axis = 0)
data2 = np.concatenate([data2_ori.T, np.ones((1, data2_ori.shape[0]))], axis = 0)
img1 = plt.imread('data/img1.jpg')
img2 = plt.imread('data/img2.jpg')
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
for j in range(data1_ori.shape[0]):
    cv2.circle(img1, (np.int32(data1_ori[j, 0]), np.int32(data1_ori[j, 1])) , 5,␣
 ↪(255, 0, 0), 2)
plt.imshow(img1)
plt.subplot(1, 2, 2)
for j in range(data2_ori.shape[0]):
    cv2.circle(img2, (np.int32(data2_ori[j, 0]), np.int32(data2_ori[j, 1])) , 5,␣
 ↪(255, 0, 0), 2)
plt.imshow(img2)
```

[2]:
```
<matplotlib.image.AxesImage at 0x7ffb8b257208>
```



1.2 Estimate Fundamental Matrix from Point Correspondences

In this part, you will implement the 8-point algorithm to estimate the fundamental matrix. For any pair of matching points $\mathbf{x}_i \leftrightarrow \mathbf{x'}_i$ in two images, the 3×3 fundamental matrix is defined by the equation:

$$\mathbf{x'}^T \mathbf{F} \mathbf{x} = 0$$

Let $\mathbf{f}$ be the 9-vector made up of the entries of $\mathbf{F}$ in row-major order, we get:

$$(x'x, x'y, x', y'x, y'y, y', x, y, 1)\mathbf{f} = 0$$

From a set of n point matches, we obtain a set of linear equations of the form:

$$\mathbf{A}\mathbf{f} = 0$$

The solution for $\mathbf{f}$ is the singlar vector corresponding to the smallest singular value of $\mathbf{A}$. Then you will enforce the singularity constraint to $\mathbf{F}$ matrix such that the rank of $\mathbf{F}$ is 2. Note that the normalization step is very important here to for accurate estimation.

You can verify your estimation by visualizing the epipolar lines in both images, where the epiploar lines will pass through all matching points. The helper function `plot_epipolar_line()` is provided for visualization

Implement the following function(s): `cv2.findFundamentalMat()`

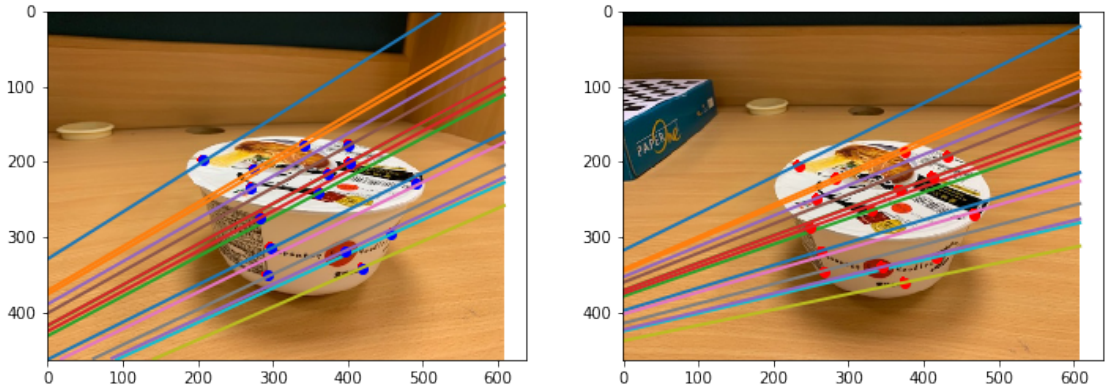
* Prohibited Functions: `cv2.findFundamentalMat()`

* You may use the following functions: `np.linalg.svd()`

[3]:
```
F = compute_fundamental(data1, data2)
plt.figure(figsize = (12, 6))
plt.subplot(1, 2, 1)
plt.imshow(img1)
for i in range(data1.shape[1]):
```

```
  plt.plot(data1[0, i], data1[1, i], 'bo')
  m, n = img1.shape[:2]
  line1 = np.dot(F.T, data2[:, i])
  t = np.linspace(0, n, 100)
  lt1 = np.array([(line1[2] + line1[0] * tt) / (-line1[1]) for tt in t])
  ndx = (lt1 >= 0) & (lt1 < m)
  plt.plot(t[ndx], lt1[ndx], linewidth=2)

plt.subplot(1, 2, 2)
plt.imshow(img2)
for i in range(data2.shape[1]):
  plt.plot(data2[0, i], data2[1, i], 'ro')
  m, n = img2.shape[:2]
  line2 = np.dot(F, data1[:, i])
  t = np.linspace(0, n, 100)
  lt2 = np.array([(line2[2] + line2[0] * tt) / (-line2[1]) for tt in t])
  ndx = (lt2 >= 0) & (lt2 < m)
  plt.plot(t[ndx], lt2[ndx], linewidth=2)
```



1.3 Estimate Essential Matrix from Point Correspondences

In this part, you will also implement the 8-point algorithm to estimate the essential matrix. The steps are the same with the fundamental matrix estimation except for that : 1. The normalization step: For each correspondence $\mathbf{x}_i \leftrightarrow \mathbf{x'}_i$, compute $\mathbf{K}^{-1}\mathbf{x}_i \leftrightarrow \mathbf{K'}^{-1}\mathbf{x'}_i$. $\mathbf{K}$ and $\mathbf{K'}$ are the camera calibration matrices which are given in the `intrinsics.h5` file. Note we only give one camera calibration matrix here because the two images are taken by the same camera.

2. The singlarity constraint: The essential matrix should have two similar singular values, and third is zero.

Implement the following function(s): `cv2.findEssentialMat`

* Prohibited Functions: `cv2.findEssentialMat()`

* You may use the following functions: `np.linalg.svd()`, `np.linalg.inv()`

Note that the your estimated essential matrix may be different from the results estimated by using `cv2.findEssentialMat()`, because the `cv2.findEssentialMat()` use a different algorithm.

[4]:
```
with h5py.File('data/intrinsics.h5', 'r') as f:
    K = f['K'][:]
E = compute_essential(data1, data2, K)
```

1.4 Two-view Relative Pose Estimation

In this part, you will extract the relative rotaion **R** and translation **t** from the essential matrix **E** accordint to:

$$\mathbf{E} = \lfloor \mathbf{t} \rfloor_{\times} R.$$

The essentrial matrix can be decomposed into 4 feasible camera poses, and you will select the correct one by cheriality check. Specifically, the 3D structure can be computed with the linear triangulation method, and the 3D points should appear in front of both cameras. Note that we assum that the rotation and translation of the first camera are identity matrix and zeros respectively.

Implement the following function(s): `cv2.recoverPose()`

* Prohibited Functions: `cv2.recoverPose()`

* You may use the following functions: `np.linalg.svd()`

[5]:
```
trans = decompose_e(E, K, data1, data2)
```