1 CS4277/CS5477 Lab 3-2: Absolute Pose Estimation

1.0.1 Introduction

In this assignment, you will get to estimate the rotation and translation of a camera by using the linear n-point camera pose determination algorithm. As discussed in the lecture, You will need at least four 2d-to-3d correspondencs to get a unique solution. We will provide ten 2d-to-3d correspondences and the camera intrinsics in the dataset.

This assignment is worth **10%** of the final grade.

References: * Lecture 7

Optional references:

* Long Quan, Zhong-Dan Lan. Linear N-point Camera Pose Determination.

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** (`pnp.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

Zip your completed `pnp.py` and `eight_point.py` and upload onto the relevant work bin in Luminus.

---

## 1.1 Part 1: Absolute Pose Estimation

In this part, you will implement the linear n-point camera pose determination algorithm. You will estimate the camera postion and orientation given a calibrated camera and ten 2d-to-3d correspondences. Each pair of correspondences $\mathbf{p}_i \leftrightarrow \mathbf{u}_i$ and $\mathbf{p}_j \leftrightarrow \mathbf{u}_j$ gives a constraint on the unknown
```

camera-point distances:

$$d_{ij}^2 = x_i^2 + x_j^2 - 2x_i x_j cos\theta_{ij},$$

where $d_{ij} = \|\mathbf{p}_i - \mathbf{p}_j\|$ and θ_{ij} is the inter-point distance and angle. The quadratic constraint can be written as :

$$f_{ij}(x_i, x_j) = x_i^2 + x_j^2 - 2x_i x_j \cos\theta_{ij} - d_{ij}^2 = 0.$$

For $n = 3$, we can obtain three constraints

$$\begin{cases} f_{12}(x_1, x_2) = 0 \\ f_{13}(x_1, x_3) = 0 \\ f_{23}(x_2, x_3) = 0 \end{cases}$$

for the three unknown distances x_1, x_2, x_3. The elimination of x_2, x_3 gives an eighth degree polynomial in x_1:

$$g(x) = a_5 x^4 + a_4 x^3 + a_3 x^2 + a_2 x + a_1 = 0,$$

where $x = x_1^2$. Thus, given ten 2d-to-3d correspondences in the dataset, you will get $\frac{9\times 8}{2} = 36$ constraints for each unknown x_i. The matrix equation can be written as:

$$\begin{bmatrix} a_1 & a_2 & a_3 & a_4 & a_5 \\ a_1^2 & a_2^2 & a_3^2 & a_4^2 & a_5^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_1^{36} & a_2^{36} & a_3^{36} & a_4^{36} & a_5^{36} \end{bmatrix} \begin{bmatrix} 1 \\ x \\ x_2 \\ x^3 \\ x^4 \end{bmatrix} = \mathbf{A}_{36\times 5}\mathbf{t}_5 = 0.$$

The vector $\mathbf{t}_5$ is obtained from the singular value decomposition of $\mathbf{A}_{36\times 5}$. Then x can be calculated as :

$$x = \text{average}(t_1/t_0, t_2/t_1, t_3/t_2, t_4/t_3),$$

and the final depth is $x_i = \sqrt{x}$. We will repeat the same process for all other points.

Here are some steps you will follow during the implementation:

1. Construct the matrix $\mathbf{A}$, which is made up of the coefficients of the polynomial. We provide a help function `extract_coeff()` to extract the coefficients of a polynomial. An example of how to use the function is given below. Note that you will compute the `cos_theta12, cos_theta23, cos_theta_13` and `d12, d23, d13` using the real data.

2. Compute the camera-point distance x_i for each point by taking SVD of matrix $\mathbf{A}$.

3. Reconsruct the 3d coordinates of each point by using the helper function `reconstruct_3d()`. Note that the 2d points should be in the homogeneous coordinate in this function.

4. Recover the camera rotationa and translation by using the ICP algorithm. We provid the helper function `icp()`,where the inputs are the 3d coordinates of all points under the world and camera coordinates. Note that you may need the `np.squeeze()` to convert the data into the required format.

After you get the rotation and translation of the camera, you can check your results by reprojecting all 3d points into image space and compare with the ground truth. You will find that the reprojections of the 3d points are close to the ground truth pixels if your estimations are correct (As shown below).

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

* You may use the following functions:

`np.linalg.svd(), np.linalg.inv(), combinations()`.

In [ ]:
```
import sympy as sym
x1, x2, x3 = sym.symbols('x1, x2, x3')
cos_theta12, cos_theta23, cos_theta_13 = 0.0, 0.0, 0.0
d12, d23, d13 = 0.0, 0.0, 0.0
a = extract_coeff(x1, x2, x3, cos_theta12, cos_theta23, cos_theta13, d12, d23, d13)
```

In [1]:
```
%matplotlib inline
%load_ext autoreload
%autoreload 1
import numpy as np
import sympy as sym
import scipy.io as sio
from sympy.polys import subresultants_qq_zz
from itertools import combinations
import cv2
import matplotlib.pyplot as plt
from pnp import pnp_algo, visualize
np.set_printoptions(precision=6)

data = sio.loadmat('data/data_pnp.mat')
points2d = data['points2d']
points3d = data['points3d']
K = data['k']
r, t = pnp_algo(K, points2d, points3d)
points2d = np.squeeze(points2d)
points3d = np.squeeze(points3d)
visualize(r, t, points3d, points2d, K)
```