

Hw2 Camera Relocalization

- **Department:** Electrical Engineering
 - **Name:** Ching-Hsiang Wu (武敬祥)
 - **ID:** R11921080
-

Content

[Content](#)

[Problem1: 2D-3D Matching](#)

[Q1-1](#)

[Q1-2](#)

[Q1-3](#)

[Problem2: Augmented Reality](#)

[Package Introduction](#)

[How to Execute?](#)

[Youtube Link](#)

Problem1: 2D-3D Matching

Q1-1

Using the **P3P + RANSAC** algorithm to estimate the camera pose with respect to the world coordinates. The **pseudo-code** can be divided into two parts; the first part implements the **P3P** algorithm, and the second part implements the **P3P + RANSAC** algorithm

- **P3P** algorithm (the definition of the parameters is the same as the course slides).

```
def solveP3P(points2D, points3D):
    1. calculate the v1,v2,v3,Cab,Cbc,Rab,Rbc,Rca by points2D & points3D
    2. if Rab or Rbc or Rac ==0:
        return "P3P cannot be solved"
    3. calculate the K1,K2,G0,G1,G2,G3,G4
    4. construct the companion matrix: C of x polynomial
    5. obtain x by:
        x = eig(C)
    6. remove the complex number of x
    7. obtain a by:
        a = sqrt((Rab^2)/(1+x^2-2*x*Cab))
    8. obtain y by:
        y = (-(x^2-K1)+(x^2*(1-K2)+2*x*K2*Cab-K2))/(2(K1*Cca-x*Cbc)+2*x*Cbc*(1-K1))
    9. obtain b by:
        b = x*a
```

```

10. obtain c by:
    c = y*a
11. obtain T by trilateration method:
    Ts = Trilateration(a,b,c, points3D)
12. for each T in Ts:
    obtain lambdai (i = 1,2,3)
    calculate the rotation matrix R by:
        R = (lambdai*v1|lambdai*v2|lambdai*v3)*inv(points3D-T)
    if det(R) > 0:
        break
return R, T

```

- **P3P + RANSAC algorithm**

```

def solveP3PRANSAC(self, points2Ds, points3Ds, threshold):
    1. initialize the bestInliers=0.
    2. calculate the iterations times by:
        iterations = (1-confidence)/(log(1-(1-s)^3)) # assume s = 0.7, let confidence = 0.95
    3. for _ in length(iterations):
        select 3 random 2D-3D keypoints pairs : points2D, points3D
        solve P3P problem by:
        _, rot_temp, trans_temp = solveP3P(points2D, points3D)
        initialize the number of inliers=0.
        for _ in length(points3Ds):
            calculate the reprojectionError
            if reprojectionError < threshold:
                inliers += 1
        if inliers > bestInliers:
            bestInliers = inliers
            rot_best, trans_best = rot_temp, trans_temp
    return rot_best, trans_best

```

Assuming the percentage of the outliers(s) is 0.7 and letting confidence = 0.95 when calculating the iteration times.

Q1-2

The **median error** of 130 validation images of **translations(m)** and **rotation(degree)** are shown below. The total **elapsed time(s)** of executing the P3P+RANSAC algorithm is also provided.

```

Total time elapsed : 1053.587871313095
Rotation median error : 0.022289683159534057
Translation median error : 0.0024679094003161772

```

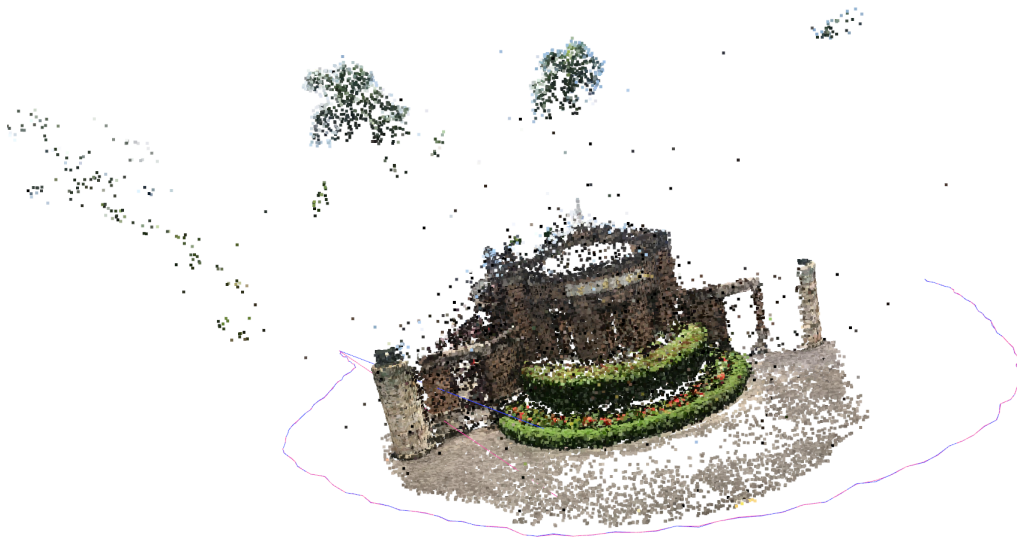
Q1-3

The camera poses of 130 validation images and NTU gate 3D points cloud can be drawn with `open3d`. The drawing procedures can be divided into 3 parts, drawing the camera

translation trajectory, drawing the camera rotation pyramid, and pasting the camera image on the camera plane.

1. Drawing the camera translation trajectory.

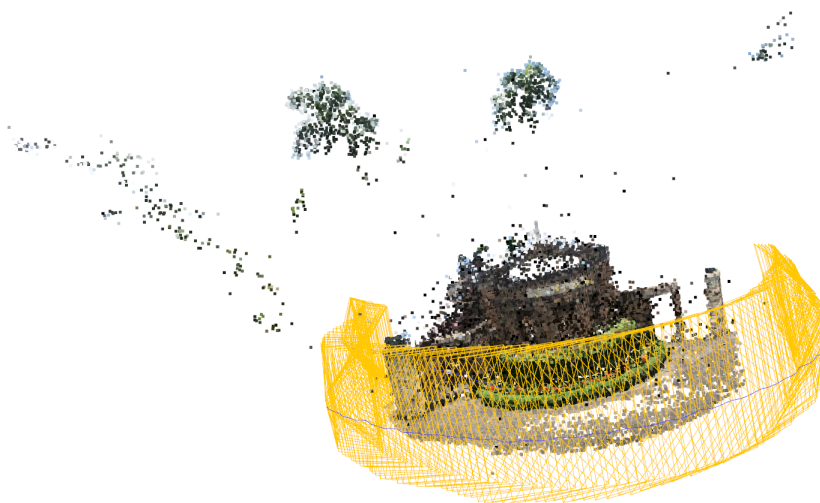
The result is shown below (Blue: estimated, Magenta: ground truth)



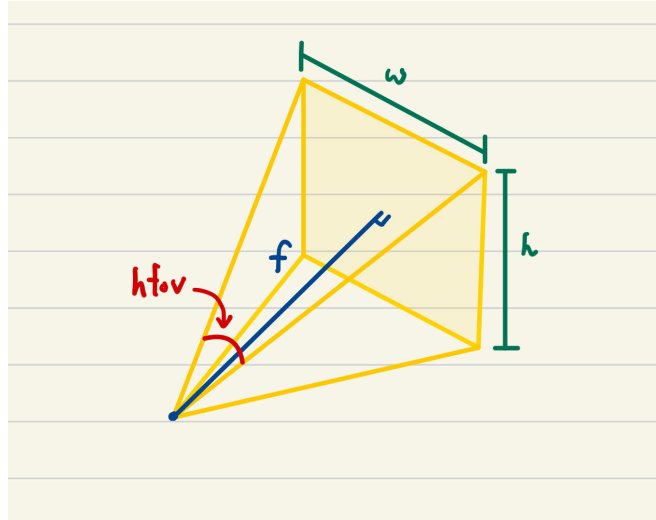
Because the order of the “**IMAGE_ID**” does not match the time sequence of the camera motion, I use `natsort` to obtain the correct order of frames first.

2. Drawing the camera rotation pyramid.

The result is shown below



Here each camera pose can be seen as a quadrangular pyramid.



the camera intrinsic matrix is

$$K = \begin{bmatrix} f_x & s & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1868.27 & 0 & 540 \\ 0 & 1869.18 & 960 \\ 0 & 0 & 1 \end{bmatrix}$$

We can know the validation image has **96 dpi** from the image information. Then assuming the **hfov** is **90** degrees. To this end, we can obtain the **w**, **h**, and **f**.

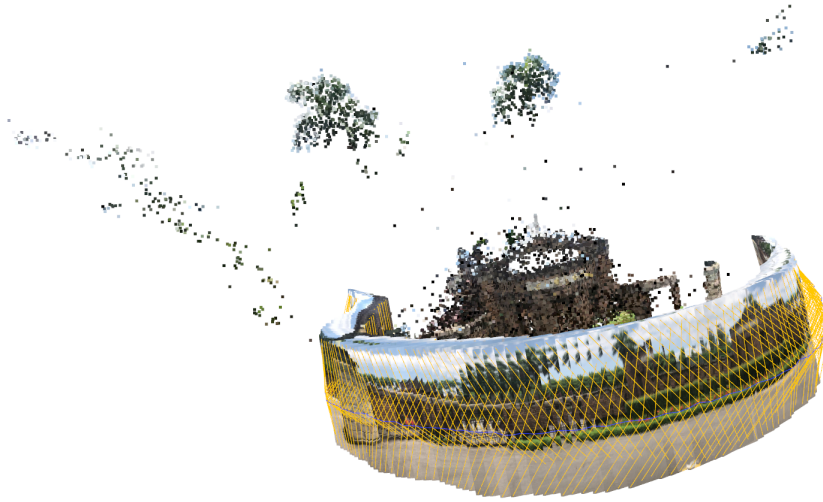
$$f = \frac{f_x}{\frac{96dpi}{0.0254}} \approx 0.5m$$

$$w = \tan\left(\frac{hfov}{2}\right) \cdot f = 0.5m$$

$$h = w \cdot \frac{o_y}{o_x} = 0.89m$$

3. Pasting the camera image on the camera plane.

The final result is shown below.



Problem2: Augmented Reality

The cube is placed on the rooftop of the NTU gate, which is shown below.



For the painter's algorithm, I created a class `Point` which stored the information of the points such as, world coordinate, camera coordinate, image coordinate, and point color. Then, sorting all the points by their z of the camera coordinate in reverse order. Last, paint the point with the largest z coordinate first.

The demo video is named as **"AR-cool.mp4"**.

Package Introduction

cv3D.py

- `poseEstimator()`

- `__init__(self, cameraMatrix, distCoeffs)`

Initialize the private parameters.

Parameters:

- **cameraMatrix**: camera intrinsic matrix.
- **distCoeffs**: camera distortion coefficients.

Return:

- None

- `featureMatching(self, query, model)`

Find the 2D - 3D key points pairs.

Parameters:

- **query**: 1 by 2, key points and descriptors pairs on the image coordinate.
- **model**: 1 by 2, key points and descriptors pairs on the world coordinate.

Return:

- **points2Ds**: key points on the image coordinates.
- **points3Ds**: key points on the world coordinates.

- `solveP3P(self, points2D, points3D)`

Find the camera pose: translation(camera → world) and rotation(world → camera), with respect to world coordinates.

Parameters:

- **points2D**: 3 by 2, key points on the image coordinates.
- **points3D**: 3 by 3, key points on the world coordinates.

Return:

- **retval**: boolean. 1 represents the solution that exists, and 0 means the opposite.
- **R**: rotation matrix (${}^C_W R$)
- **T**: translation (${}^W P_{Corg}$)

- `solveP3PRANSAC(self, points2Ds, points3Ds, iterations=None, threshold = 5, confidence = 0.95)`

Find the camera pose: translation(camera → world) and rotation(world → camera), with respect to world coordinates and using the RANSAC algorithm to remove the outliers.

Parameters:

- **points2Ds**: N by 2, key points on the image coordinates.
- **points3Ds**: N by 3, key points on the world coordinates.
- **threshold**: maximum allowed reprojection error to treat a point pair as an inlier (used in the RANSAC methods only). (default = 5).
- **confidence**: how confident that the final solution is calculated by the inliers. (default = 0.95)
- **s**: the percentage of the outliers (default = 0.7)
- **iterations**: the executing times for RANSAC to remove the outliers. (default = None). when assigned None, the iteration times will be calculated by the following formula.

$$iterations = \frac{1 - confidence}{\log(1 - (1 - s)^3)}$$

Return:

- **Rot_best**: rotation matrix (${}^C_W R$)
- **Trans_best**: translation (${}^W P_{Corg}$)

◦ `__reprojectionError(self, u, v)`

Calculate the error between the estimated image coordinates and the pair image coordinates.

Parameters:

- **u**: estimated image coordinates
- **v**: pair image coordinates

Return:

- **error**: the norm of distance between the estimated image coordinates and the pair image coordinates.

How to Execute?

Download the **Dataset** from github and put it into repository.

- **Environment**

- `Python == 3.8.12`
- `Opencv == 4.5.1`
- `Numpy == 1.24.3`
- `Open3d == 0.11.2`
- `Pandas == 2.0.3`
- `Scipy == 1.10.1`
- `Natsort == 7.1.1`

- **Problem1**

- Q1-1, Q1-2

Run the following command on the terminal: `python camera_pose_estimation.py`

(it may take a little time)

- Q1-3

Run the following command on the terminal: `python camera_trajectory.py`

- **Problem2**


Run the following command on the terminal: `python augmented_reality.py`

Youtube Link

3DCV__Homework2

This is a demonstration showing how to execute the code of 3DCV homework 2. The content includes

1. Camera pose estimation

 <https://youtu.be/gh57vc27-Gs>

