3DCV HW2

電信碩一 黃郁珊 r09942089

Problem 1: 2D-3D Matching

Q1-1 Implement PnP algorithm

I implemented the **Epnp** algorithm. The pseudo code is as follow:

```
3D points: (Num_3d, x_w, y_w, z_w)
For each image,
    2D image points: (M, u, v)
    Use knn feature matching to form corresponding 2D-3D pairs
    #Epnp
    Choose 4 control points in WCS
    Compute alphas of all reference points
    Solve MX = 0 by minimizing MX (X simplifies 4 control points in CCS)
    ⇨ Get 4 v  s.t.  $x = \sum_{i=1}^{N} \beta_i v_i$
    For N =1,2,3,4:
        Find beta and do Gaussian Newton Optimization
        Compute R, t
    Choose N, betas, R, t with smallest reprojection error
```

1. Choose 4 control points in World coordinate system. (WCS)
   One control points is the centroid of 3D pints. The others are the eigenvalues of $Pw0^T Pw0$ where $Pw0 = (\text{3D\_points} - \text{centroid})$.
2. Compute alpha (the barycentric coordinate parameter) of all reference points

$$\begin{bmatrix} \alpha_{i,j=1} \\ \alpha_{i,j=2} \\ \alpha_{i,j=3} \\ \alpha_{i,j=4} \end{bmatrix} = \begin{bmatrix} c^w_{x,j=1} & c^w_{x,j=2} & c^w_{x,j=3} & c^w_{x,j=4} \\ c^w_{y,j=1} & c^w_{y,j=2} & c^w_{y,j=3} & c^w_{y,j=4} \\ c^w_{z,j=1} & c^w_{z,j=2} & c^w_{z,j=3} & c^w_{z,j=4} \\ 1 & 1 & 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} P^w_{x,i} \\ P^w_{y,i} \\ P^w_{z,i} \\ 1 \end{bmatrix},$$

3. Solve MX = 0 by minimizing MX (X simplifies 4 control points in CCS)

$$\begin{aligned} &\sum_{j=1}^{4} \alpha_{ij} f_u x_j^c + \alpha_{ij}(u_c - u_i) z_j^c = 0 \\ &\sum_{j=1}^{4} \alpha_{ij} f_v y_j^c + \alpha_{ij}(v_c - v_i) z_j^c = 0 \end{aligned} \quad \xrightarrow{\text{concatenate n points}} \quad \mathbf{Mx} = \mathbf{0}$$

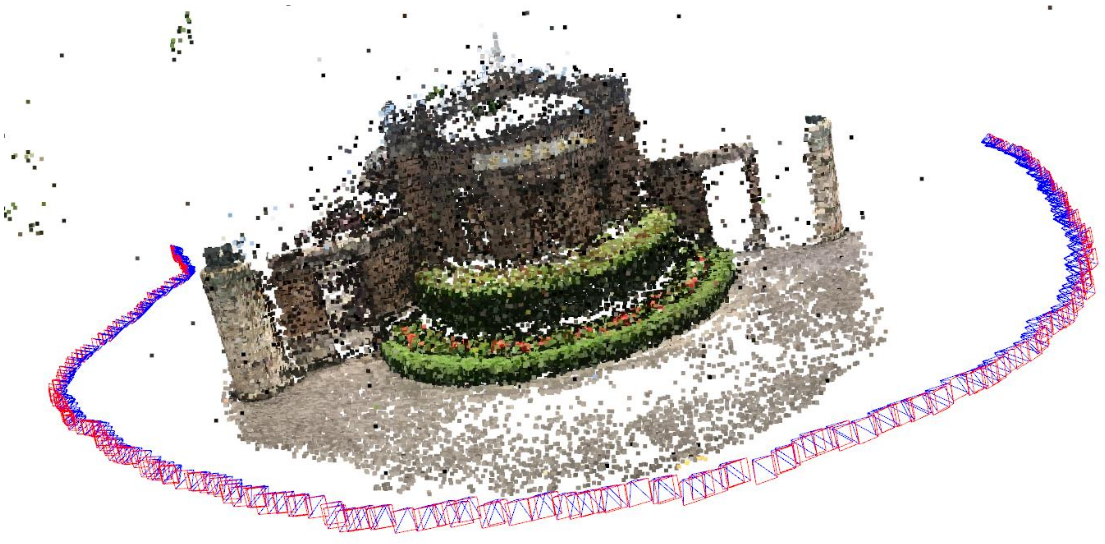
$$\mathrm{M} \sim [2\mathrm{n}, 12]$$

I solved X by computing eigenvectors of M^TM, got 4 column vectors with smallest eigenvalue and used them to represent solution X as:

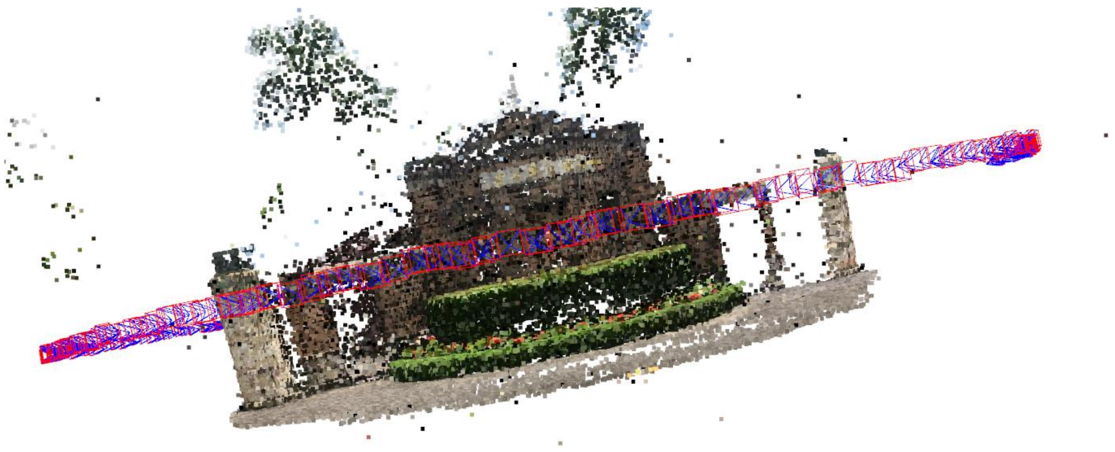
$$\mathrm{x} = \sum_{i=1}^{N} \beta_i v_i$$

4. For N = 1, 2, 3, 4, calculate their corresponding Beltas and optimized those Beltas by doing Gauss Newton Optimization. Having Beltas means getting the X, that is, the camera coordinate system (CCS) of control points. Then I can compute R, t between CCS and WCS because I already have the CCS and WCS pairs of the 4 control points.
5. Calculating reprojection error on all 2D-3D pairs and choose the best R, t

Q1-2 Camera Pose Visualization



From lower view:



Implementation

For each image, I have a set (R, t)

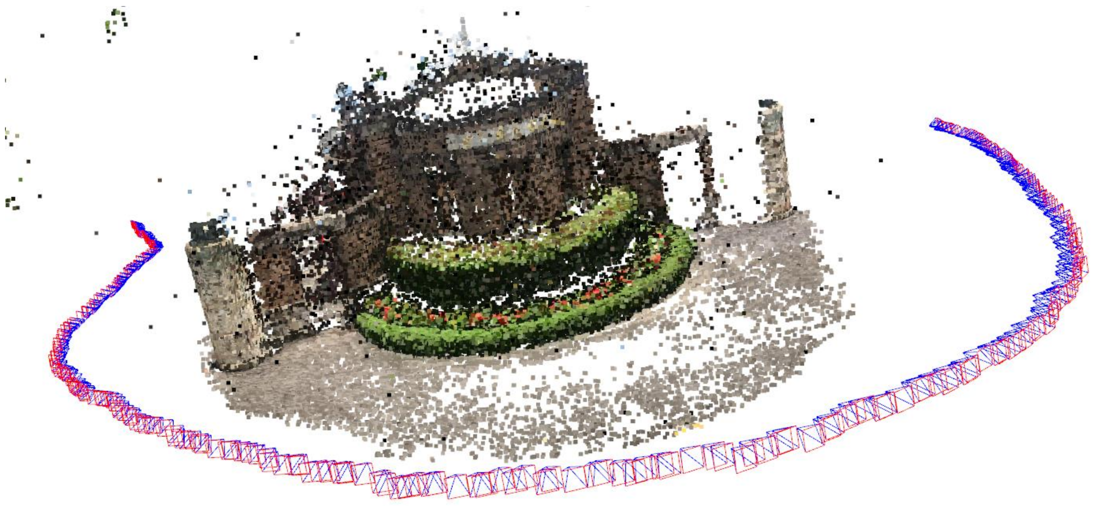
$$H = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$$

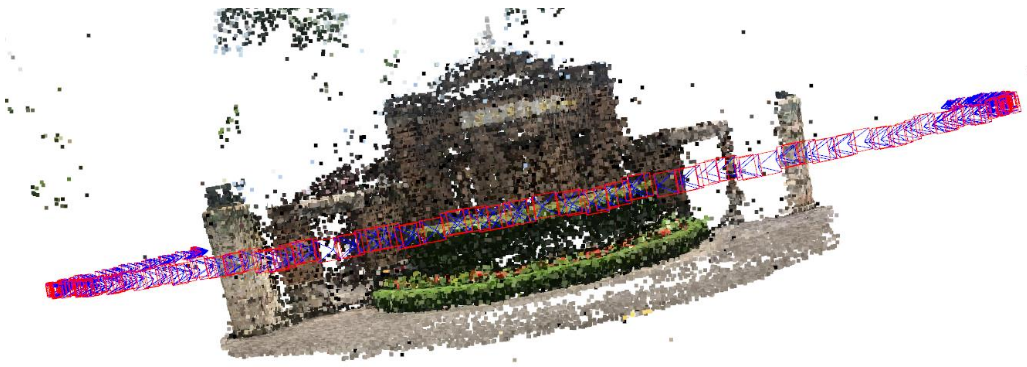
For each point p in pyramid:

$$p' = H^{-1}p \text{ where p' is WCS of each point}$$

Discussion

From the visualized result, I’m confident that I indeed get a great result. Which is really similar to ground truth (as shown below)





Q1-3 Median Pose Error

Median Rotation error	Median Translation error
0.0013619	0.0221212

The implementation of calculating rotation error:

For $R_{gt} = d_r {R_q}^T$(here R is in the format of rotation matrix), then

$$d_r = R_{gt}({R_q}^T)^{-1}$$

Then transform d_r to rotation vector and its norm length is the rotation error.

Discussion

1. I compare the results with R, t from *cv2.solvePnPRansac*

	Median Rotation error	Median Translation error
My EPnP	0.0013619	0.0221212
Cv2 PnP	$3.98 \times e^{-5}$	0.0001219

From the comparison result, my implement method still can be improved.

2. When computing rotation error, it' important to make sure the order of Quaternion is QX, QY, QZ, QW.

Problem 2: Augmented Reality

Q2-1 Placing Virtual Cube in Video

[Video Link] https://drive.google.com/file/d/12ferurXodU5z7ZJJUmxsF8EL6TqccFlO/view?usp=sharing

(The cube shows up at around 0:04)

Implementation

I sample N*N points in each plane. N=8

Painter's Algorithm: 1. Sort each voxel by depth 2. Place each voxel from the furthest to the closest. The point is: how to get the depth? The depth is the distance between point X and z=0 in Camera Coordinate System, that is, the z value of X(x,y,z). So we transform vertices of cube from WCS to CCS, then sorted them based on their z value. The vertices with biggest z are drawn first.

Discussion

The results are not very stable and highly depends on the initial position of cube in World coordinate system. Especially if the cube is far from the origin point (0,0,0), it will look unreasonable because the hiding situation is not considered.

Bonus:

1. Use linear regression to **implement RANSAC -> outlier rejection strategies**

First randomly sample some pairs of (2D point, 3Dpoint).

I use linear regression to fit 2D point (served as X) and 3D point (served as Y).

$$\text{Distance} = \left\| 3D - 3D_{predicted} \right\|^2$$

Only those pairs whose distance<θ can be view as inliers.

After several iteration, chose the iteration which has most inliers.

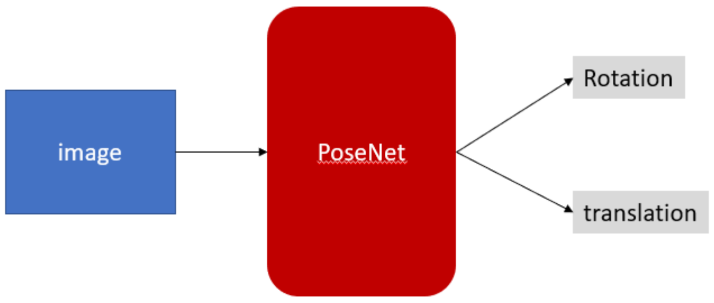
	Median Rotation error	Median Translation error
My EPnP	0.0013619	0.0221212
My EPnP(w/oRansan)	0.003189	0.0204574

*The RANSAC code can be found in utils.py

2. PoseNet

I implemented PoseNet to predict rotation and translation and used

InceptionV3 as backbone structure.



Hyperparameters:

Learning rate: 0.0001 -> 0.00001(finetune)

Batch size: 32

Optimizer: Adam

Loss: Mean square error (MSE)

$$MSE_R = \frac{1}{n}\sum_{i=1}^{n}(R_i - \widehat{R_i})^2$$

$$MSE_t = \frac{1}{n}\sum_{i=1}^{n}(t_i - \widehat{t_i})^2$$

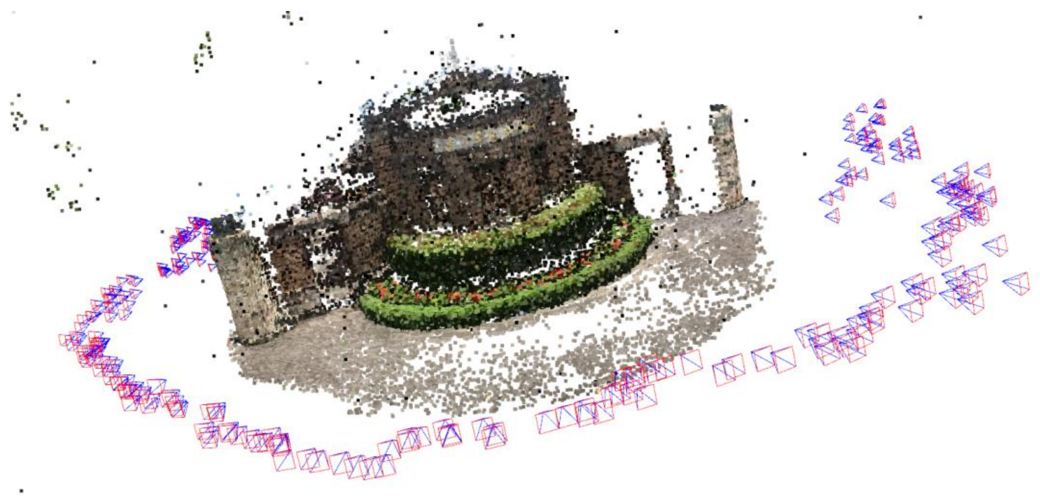
Training strategy:

At the beginning, since the imbalance value size between rotation and translation. I need to weight the term.

However, it seemed that rotation turned out to converg first. So, I need to freeze the rotation after several training iteration to make translation layer updated more.

Result

	Rotation error	Translation error
Epnp	0.0013619	0.0221212
PoseNet	0.02133556	0.1916



Time consuming

	Running time per image
Epnp	3.4 s
PoseNet	0.04s
Cv2 pnp	2 s

Discussion

From the result I found that the deep learning method performed worse but the result is already close to ground truth.

Since this task used absolute pose regression with only image as input. It's reasonable to fail achieving 100% correct. Besides, it takes only little time to finish estimation.

References

1. https://github.com/DunFenTiao/EPNP (written in C++, I implemented the Python version of Epnp based on this script)
2. https://github.com/WeiyanCai/EPnP_Python/tree/021b6876db13cf723314ef47e725d2f5ae34463d
3. *EPnP: Efficient Perspective-n-Point Camera Pose Estimation*
4. *PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization*