3dcv hw2

余承諺 r12922135

2023 / 10 / 23

Problem 1-1

To estimate the camera extrinsic, normalized DLT method is used. Consider the 3d to 2d matching trasnformation.

$$u_i = \lambda_i \mathbf{K} \left[\mathbf{R}|\mathbf{T}\right] x_i = \lambda_i \mathbf{H} x_i$$

The intrinsic of camera, $\mathbf{K}$, is already known. We can consider there is a homography $\mathbf{H}$ from x_i to u_i. After employing normalized DLT method to acquire $\mathbf{H}$, the remaining works are how to separate $\mathbf{K}$, $\mathbf{R}$ and $\mathbf{T}$.

First, we can directly left multiply $\mathbf{K}^{-1}$ to $\mathbf{H}$ to eliminate the influence of $\mathbf{K}$. Then we will get the matrix of $\lambda_i \left[\mathbf{R}'|\mathbf{T}\right]$ which is 3×4.

To determine λ_i, we can consider the left part of $\lambda_i \left[\mathbf{R}'|\mathbf{T}\right]$. Choose $k = det(\lambda_i \mathbf{R})^{-1/3}$ to make $det(k\lambda_i \mathbf{R}')$ be 1. After this, the scale of $[\mathbf{R}'|\mathbf{T}]$ has been determined and $\mathbf{T}$ is acquired.

However, the orthonormal property of $\mathbf{R}$ is not ensured. Therefore, we apply SVD to $\mathbf{R}' = U\Sigma V^H$ and only choose U and V^H. Then let $\mathbf{R} = UV^H$.

Finally, we acquire the extrinsic of camera, $\mathbf{R}$ and $\mathbf{T}$.

Since the process of point matching and solve pnp are very slow, "2d3dmatching_all.py" is used to acquire extrinsic for each frame then save them to "rt.np" as rotation vectors and translation vectors which is used to following problems.

Problem 1-2

To determine the median error of camera rotation and translation. "extract_rt_gt.py" is used to extract the groundtruth of camera extrinsic, $\mathbf{R}$ and $\mathbf{T}$, from the dataset. After comparing, the median error of rotation and translation between groundtruth and estimation is $1.53992e-3$ rad and $5.63158e-3$ respectively.

Problem 1-3

In order to draw the trajectory and cone, the first thing is to acquire the position of camera. Consider the 3d to 2d transformation.

$$u_i = \lambda_i \mathbf{K} R^{-1} [I | -T] x_i = \lambda_i \mathbf{H} x_i$$
$$u_i = \lambda_i \mathbf{K} [\mathbf{R} | \mathbf{T}] x_i = \lambda_i \mathbf{H} x_i$$

Note that the difference between of R, T and $\mathbf{R}$, $\mathbf{T}$. R, T denotes the rotation and translation of camera cone. R and T can be acquired by.

$$R = \mathbf{R}^{-1}$$
$$T = -\mathbf{R}^{-1}\mathbf{T} = -R\mathbf{T}$$

In order to draw the cone, it is necessary to acquire the 4 points at the top-left, top-right, bottom-right and bottom-left of the screen under rest-pose camera whose $\mathbf{R}$ is identity and $\mathbf{T}$ is 0. Then Consider this 4 points transformation.

$$[0, 0, 1]^T \mathbf{K}^{-1} = x_{tl}$$
$$[W, 0, 1]^T \mathbf{K}^{-1} = x_{tr}$$
$$[W, H, 1]^T \mathbf{K}^{-1} = x_{br}$$
$$[0, H, 1]^T \mathbf{K}^{-1} = x_{bl}$$

H and W are the height and width of screen. x_{tl}, $x_{x_{tr}}$, x_{br} and x_{bl} denote the 4 points. Note that we assume the camera origin is at $[0, 0, 0]^T$ and only consider these 4 points when depth $= 1$. Several experiments are made before this step to confirm these equations are appropriate. Some camera define the depth should be less than 0 and point which has greater depth is more close to camera. Some camera defines the first entries of u_i denote the index of height and second entries denote the index of width.

The result can be checked by executing "python problem_1_3.py". The trajectory is drawn in green. And the cone is drawn in red. There is only one cone drawn at a time. User can use "A" to see previous cone and "D" to see next cone.

Problem 2

Without the ability of detecting the depth of objects in the scene, the whole frame is set as background and only occlusions from the cube itself are considered. Transforming all 3d points of the cube to 2d, note that we can acquire the depths simultaneously. Already known that a point which has greater depth is more further from the screen. To avoid wrong occlusions, painter's algorithm can be used, draw points from furthest to closest. OpenCV instead of Open3D which is too complicated is used to draw frame-to-frame then connect them to one video.

Demo Video: https://youtu.be/Wk-UuUnc8IM

AR Vidoe: https://youtube.com/shorts/b-a6CWRGN9w?feature=share