CS284 HW3 Report: Camera Calibration

Hongchen Cao
2019533114
caohch1@shanghaitech.edu.cn

***Abstract*—This report covers a basic description of the implementation, performance, and qualitative observations on the results of 3 sub-tasks in HW3.**

I. Project

A. Description of the algorithm

The algorithm for task1 can be summarized as follow:

ALGORITHM 1: Algorithm-1

$M = P[:, 0:3]$
$K, R = RQ(M)$
$C = (-M)^{-1}P[:, 3])$
return K, R, C

The algorithm for task2 can be summarized as follow:

ALGORITHM 2: Algorithm-2

$points = (xyz_1, uv_1), (xyz_2, uv_2), ..., (xyz_3, uv_3)$
$P = DLT(points)$
$K, R, C = Algorithm - 1(p)$
return K, R, C

The algorithm for task3 can be summarized as follow:

ALGORITHM 3: Algorithm-3

\# Use FAST
$points1 = feature\_extract(img1)$
$points2 = feature\_extract(img2)$
\# Use BRIEF and Brute-force
$point\_pairs = match(points1, points2)$
$E = cal\_essen(point\_pairs, K)$
return E

I use *python* to implement the algorithm and visualize the result.

B. Structure of the project

CS284_hw3_data contains the raw data.
CS284_hw3.pdf is the introduction for HW3.
vis contains the visualization of results.
main.py is the implementation and visualization of 3 sub-tasks.

```
Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
d-----        2021/10/29     20:26          1 .idea
d-----        2021/10/19     11:53          1 CS284_hw3_data
d-----        2021/10/29     20:26          1 vis
-a----        2021/10/24     18:53     944.80KB CS284_hw3.pdf
-a----        2021/10/29     20:26      11.72KB main.py
```

Fig. 1: Structure of the project

C. Dependencies

python == 3.6.13
matplotlib == 3.3.4
numpy == 1.19.5
opencv-python == 3.4.2.16

D. Instructions

Run `python main.py`

II. Performance

All the results can be visualized by run *main.py*.

A. Efficieny

The time consumed for Task1 is $0.00099s$;
The time consumed for Task2 is $1.04320s$;
The time consumed for Task3 is $4.93277s$;
The time calculated includes the process of reading raw data, the main calculating part, printing, saving and so on.

B. Accuracy and Visualization

Accuracy

For task2, I calulate the error for each single image and the error between every two images.

For each single image, I use the result matrix(i.e., camera projection matrix) to calculate points $[uv'_1, uv'_2, ..., uv'_6]$ and compute the error between them with the original input $[uv_1, uv_2, ..., uv_6]$.

for the error between every two images, I simply sum up the distance between 2 matrix (i.e., $err = \sum_{i=1}^{9} K[i] - K'[i]$)

	img1	img2	img3
error_single	5.94846	5.23800	0.75065
	img1&2	img2&3	img3&1
error_two	51.47314	22.11678	29.35636

TABLE I: Error for task2

For task3, there are 6 result essential matrix and I further calculate R and C. Then I compute the error between transform matrix computed by R, C and results in task2 by suming up the distance between 2 matrix.

	img1-2	img1-3	img2-3	img2-1	img3-1	img3-2
error	2.55132	7.60498	6.45939	5.90951	6.37876	0.69612

TABLE II: Error for task3

For detail matrix value, you can see the output information in terminal by runing the *main.py*.

Visualization The visualization of all the results are stored in *./vis*.

Here is 2 examples for selected corners of img1 in task2 (See Fig.I) and matched points of img3-2 in task3 (See Fig. 2).

Fig. 2: Selected corners of img1 in task2

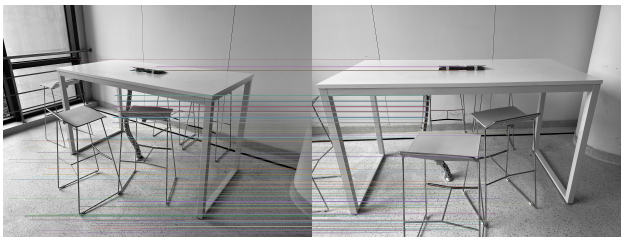

Fig. 3: Matched points of img3-2 in task 3

III. Q&A

Q1: Quantitative and qualitative observations

A1: Quantitative results can be seen in Sec. II-B. The error-single of task2 is small but the error-two is large. One possible reason is that the point pairs are manually marked by myself, and the second possible reason is that the photo itself is distorted.

For task3, the main source of error may be inaccurate feature extraction. In Fig. 3, we can see that there are a lot of feature points on the floor, chairs and pipe under the table, and they are matched to the wrong place.