fcosta
19-931-989

Homework 4

Computer Vision
December 3, 2021

1 RANdom SAmple Consesus

1.1 Least-squares Solutions

There is nothing special about this function. I use the lstsq function from numpy to solve the problem as metioned in the assignement.

1.2 RANSAC

The process is the same for each iterations. First I sample some points. Then I compute the least square solution. Once I am done, I compute the number of inliners and compare with the best previous solution.

1.3 Results

As we can see on the picture, the least square regression is not the same as the RANSAC version. The RANSAC version clearly avoid outliers while the classic version is biased by these outliers.

Output in the terminal :

```
Estimated coefficients (true, linear regression, RANSAC): 1 10 0.6159656578755459 8.96172714144364
0.9987449792570883 9.99700975879101
```

2 Multi-View Stereo

2.1 RGB Image

This is simple image processing with PIL and numpy

2.2 Camera parameters

This is a simple text processing method. There is nothing special to say about that.

2.3 Feature extraction

I create the NN like in the table 3.1. I create a custom ConvBnReLU block and I used it to create the architecture.

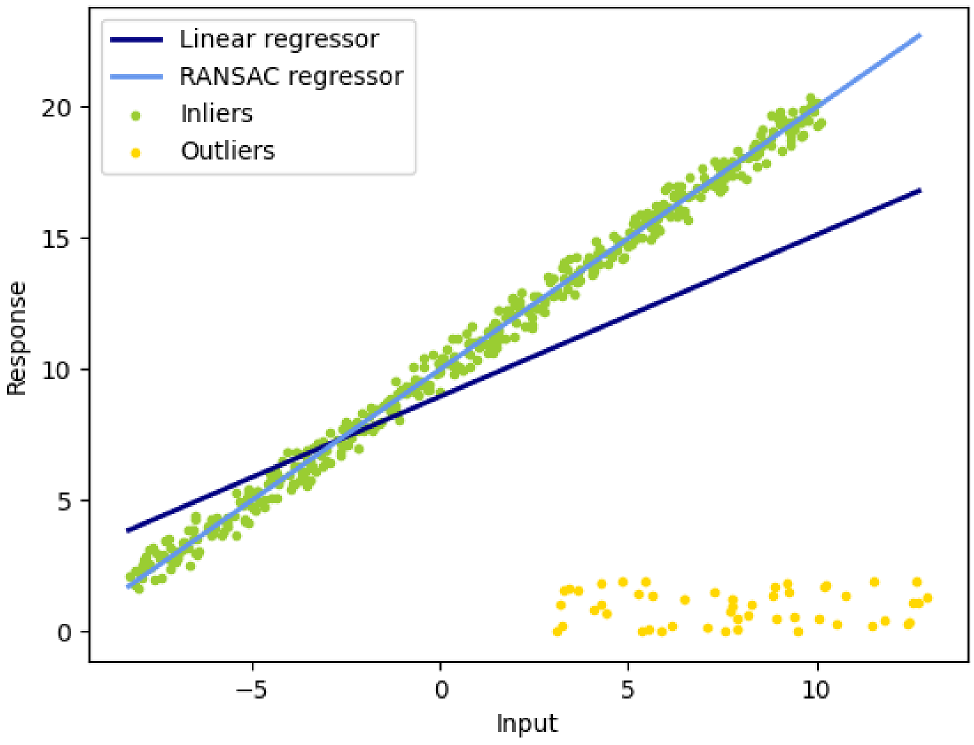


Figure 1: Calibration camera

2.4 Differential Warping

It was clearly the hardest part. The equation is the same as the equation from this paper : https://arxiv.org/pdf/2012.01411.pdf I took a screenshot of the equation. There are some difference in the sense that the intrinsics matrix is included in the projection and matrix. The projection was therefore a bit easier than this equation.

$$\mathbf{p}_{i,j} = \mathbf{K}_i \cdot (\mathbf{R}_{0,i} \cdot (\mathbf{K}_0^{-1} \cdot \mathbf{p} \cdot d_j) + \mathbf{t}_{0,i}).$$

Figure 2: Projection of the point

2.5 Similarity computation

I implemented the first function. First I implemented it without vector processing and for loops. It took too much time to backpropagate and I created a vectorized version. The NN was not a very hard task. I followed the architecture of table 3.2

2.6 Similarity computation

Simple tensor processing.

2.7 Loss

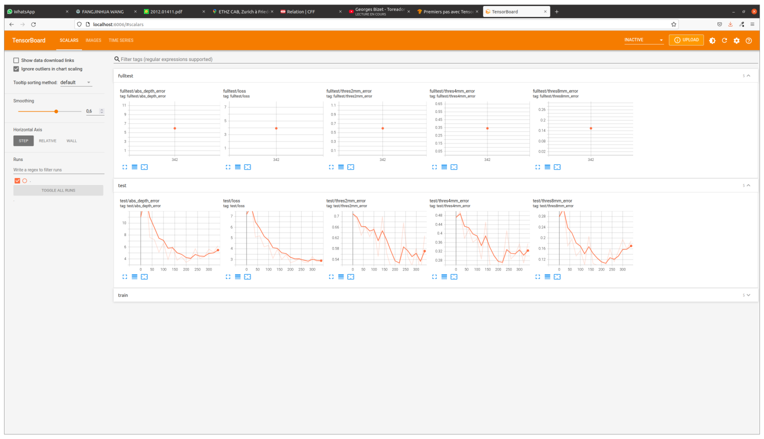
Implementation of a simple loss. Nothing special

2.8 Train

I trained the nn only one iteration because It takes a lot of time and I had not enough time left

2.9 Test

Geometric consistency filtering is all about re-projecting the points and then compare the error. If the error is small enough then we can consider that it is a match. If enought points are matching (3 in the code) then we can consider this point as correct. This step in the pipeline allow to reduce the number of strange points.



localhost:6006/#scalars

- [ ] Show data download links
- [x] Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing: 0,6

Horizontal Axis: [✓ STEP] RELATIVE WALL

Figure 3: Tensorboard

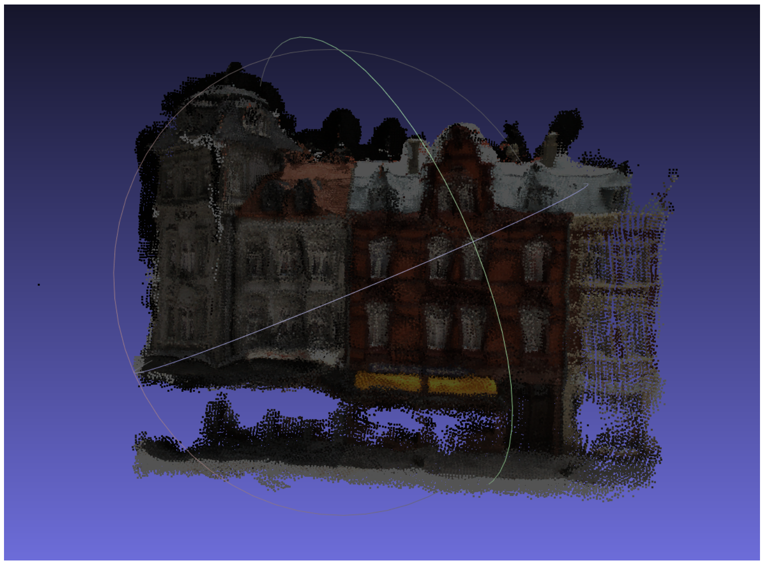


Figure 4: First scene

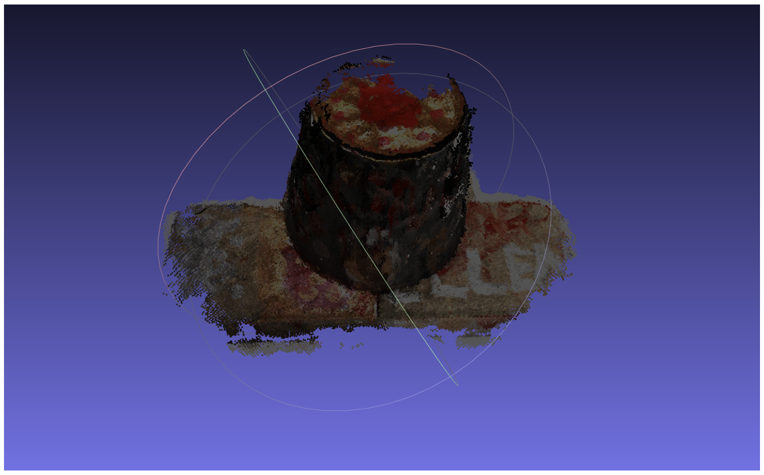


Figure 5: Second scene

2.10 Questions

1) If we sample uniformly in the range then the samples are logarithmic distributed in the inverse range. We can safely assume that in large scale datasets the images have less points that are near the camera. Hence uniformly in the inverse range would make more sense in my opinion.

2) I think it can be challenging for some situation. If we have occlusion, we could have pixel that are very similar even if the principal colour is not the same. It mean that the estimated distance is going to be the average of the distance of the points. Hence this is not robust to all situation in my opinion.