Exercise Sheet 4

Anselm Coogan

November 28, 2019

1 Exercise 3

A robust loss function should be robust against outliers, i.e. a few large outliers shouldn't affect the loss, and hence the learner, drastically. Additionally, it should converge strongly to a minimum. The Huber loss does this by combining the absolute loss for large residuals (greater than some threshold) and the squared loss for residuals closer to the optimum. In doing so it assures that outliers do not contribute quadratically to the loss and that it still converges at the same speed when it is close to the optimum as the squared loss. We want such robust loss functions when we expect to have outliers far greater than inliers. During calibration we do not expect to have large outliers mainly because our ground truth dataset is a regular corner grid detected with the help of a robust aprilgrid and not a set of irregular arbitrary features. Moreoever, as we are optimizing over a set of cameras that may be far away from each other, the discrepancy during reprojection may also be large.

2 Exercise 4

2.1 Outlier filters:

- Large reprojection error
- Too close to camera
- Too small z-coordinate

Large reprojection errors are likely the result of a feature mismatch, i.e. feature track x is incorrectly detected by camera y. Too small z-coordinates (e.g. negative, behind the camera) are likely related to errors in the camera to world transformations coupled with features that are far away from the camera. Outliers that are to close to the camera are also likely to be due to abberations in the camera-world transformations.

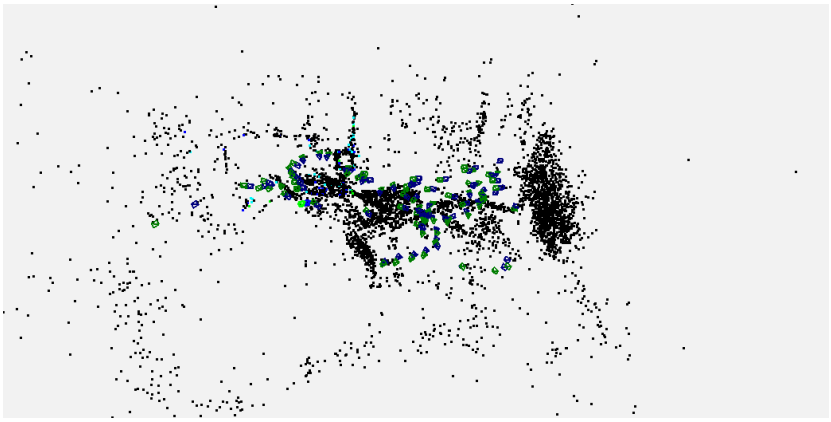

Figure 1: the fully built map

3 Exercise 5

- number of cameras added to map: 151 of 162
- total optimization time: 56756ms → roughly one minute
- what takes longest: optimization takes the longest by far - about half a second per optimization

3.1 Suggestions for speeding up process

Since the bundle adjustment step in the pipeline takes the longest by a factor of 10 compared to all other steps, we will need to speed up this step. One possible way may be to parallize this process and optimize several smaller batches in parallel. Then we can do a final optimization on the entire data. This should converge faster as our initialization should be better. Whether this two-step process is actually faster, one would have to try out.

Another way to speed up bundle adjustment would be to run it on a gpu as the required matrix operations would be significantly faster.