Localization and SLAM Challenges

Report for Robot Perception and Learning - 01-14-2014
趙冠琳, B99901164 - 林泉亨, R0252151 - Sander Valstar, R02922149
EECS – NTU, Taipei, Taiwan
Prof. Chieh-Chih (Bob) Wang , CSIE –NTU, Taipei, Taiwan

Abstract — *In this project, the team has created two sets of software. One for the Robot Localization Problem and one for the SLAM problem. The used robot is equipped with RGBD cameras and has access to wheel odometry. The project makes use of techniques like SURF, PnP, Ransac, ICP and ICPNL. The considered and used approaches, the encountered difficulties and the end results will be the main topics of this document.*

1. Localization

1.1. Introduction to the problem

In the robot localization problem, one is given a known map of the environment. The problem is to determine the robot's location in that map from its sensor- and odometry data. In our case, the robot had access to 3 Xtion Pro Live[1] RGBDepth cameras and wheel odometry data. The challenge was to map the data from one or more RGBD camera's to the known 3D-map to determine the robot's position. The odometry could be used as initial guess.

1.2. Considered approaches

For localization we decided not to use the robot's wheel odometry, since we found it to be very inaccurate in our self-recorded data. We therefore decided to solely implement visual odometry.

1.1.2. First approach

We started off by constructing a big 3D point cloud map from the RGBD data that was given by the Teaching Assistant using Point Cloud Library[2]. The philosophy behind this, was that we would use the 3D point cloud as a map to match the robot's data with. We therefore also constructed point clouds of each frame of the robot's RGBD data. To match these point clouds to the 3D map, we implemented a 3D Scale-Invariant FeatureTtransform[3] -function. This function extracts the SIFT-keypoints of the robot images and the 3D map. Unfortunately we had not foreseen that 3D SIFT would be extremely slow and moreover we did not succeed in finding a correct transformation matrix using these SIFT-keypoints. Hence we decided to abandon the idea of using 3D point clouds.

1.1.3. Final approach

After finding out that point cloud processing for SIFT features was way too computationally expensive, we decided to use a SIFT variant for 2D RGB images called Speeded Up Robust Features[4]. We chose to use OpenCV[5] to implement this. The results of SURF were much better and, moreover, they were computed surprisingly fast.

The next step in our approach was to take the map's RGB image with the maximum number of SURF keypoint correspondences to the robot's RGB image as input for the Perspective n Points[6] algorithm. The used PnP algorithm also made use of a Ransac[7] algorithm to deal with outliers. Again, we used OpenCV to implement this all.

1.1.4. Results

We had not implemented any form of visual feedback to verify our results and we only outputted the robot position matrices. This meant that we had to compare the numbers ourselves. It would have been better if we had drawn the robot position on a 2D map, but unfortunately we did not get to that point.

The PnP worked reasonably well, so it seemed. When we started working on SLAM[8] however, it turned out that the PnP was far less accurate than we had thought was.

2. SLAM

2.1. Introduction to the problem

In the Simultaneous Localization and Mapping problem, the challenge is to let a robot wander and discover a new and unknown territory. Since the area is unknown, there is no map available and since there is no map available, the robot has to build its own representation of the world around him. The localization problem is a sub problem of this mapping problem, because having a representation of the world is useless for a robot if he has no idea where in that world he is positioned.

The biggest hill to climb in this problem, will be to construct an accurate map of the environment without having any reference other than the robot's own sensors and wheel odometry.

2.2. Considered approaches

The same sensor hardware and odometry information as was used for the localization problem is being used for our SLAM challenge. This time we tried a lot more approaches, that in most cases weren't quite accurate enough to construct a usable 3D map.

2.2.3. First approach

For SLAM it seemed useful to consider odometry again, so we started our approach to solving SLAM by constructing a 2D map from wheel odometry. For the construction of the robot position we made the assumption that the only axis that the robot would turn around would be the vertical axis. This lead to the following RT matrix:

$$\begin{vmatrix} \text{Cos}(t) & 0 & \text{Sin}(t) & y \\ 0 & 1 & 0 & 0 \\ -\text{Sin}(t) & 0 & \text{Cos}(t) & x \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Our assumption that the robot would only turn its camera horizontally, was sadly not entirely correct. The robot went over bumps every now and then and it is a bit wobbly from itself. So in the end the pictures taken by the camera were by far not all taken from a horizontal perspective. This was one of the causes leading to errors in our RT matrices. Since we used the accumulative odometry, unfortunately the error in the odometry also accumulated. This lead to a very messy map. A better idea would be to use the relative odometry between frames to correct an initial guess, but therefore we would first need an initial guess.

Later on however, we found that constructing a 3D map using odometry lead to quite alright results. We had expected them to be much worse. The results are shown in Figure 1.

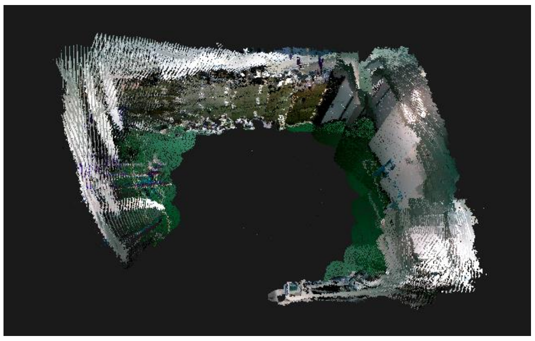
Figure 1 - 3D map constructed using only odometry

2.2.4. Second approach - PnP

We decided once again to leave the wheel odometry be and to first focus on visual odometry instead. Some of the code that was written for the localization could be reused, but now not only to provide an initial guess for the robot's position, but also for a 3D map. We reused the SURF and PnP parts of our localization code for this. This time however, we constructed 3D point clouds from the PnP results and that was when we found out that this was actually pretty inaccurate. Although the PnP constructed map looked much better than the one that was constructed with odometry, it was still very messy and we didn't believe that we could correct this with our relative wheel odometry.

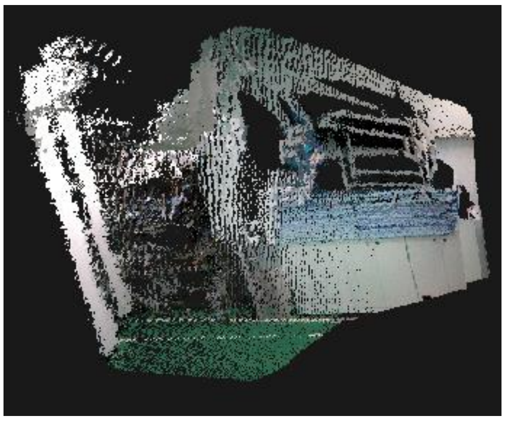
Figure 2 - Bad alignment using PnP only

2.2.5. Third approach - PnP + ICP

The most obvious method to correct our PnP-constructed map is by running an Iterative Closest Point[9] algorithm with the PnP results as initial guess. At first this sounded quite easy and straight forward, but to properly implement ICP has cost us by far the most time.

At first we tried using PCL's "IterativeClosestPoint" function, which worked pretty straight forward. However, it turned out to be unusably slow on our point clouds and moreover, the resulting alignments were far from satisfying. This lead us to trying a whole arsenal of ICP variants with varying results, but on average they were way too slow and inaccurate. We found it to be very important for ICP's performance to which value the min/max distance thresholds were set, but sadly enough we had no idea how to determine the ideal values.

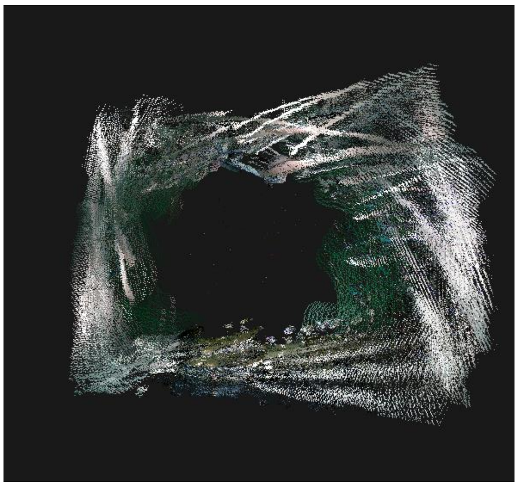

Figure 3 - An example of bad alignment using PnP + ICP

2.2.6. Final approach - PnP + ICPNL

While trying out different kinds of ICP variants we stumbled upon the "IterativeClosestPointNonlinear"[10] function of PCL. This function requires special point clouds as input, so called "PointNormal" point clouds. These point clouds do not only contain RGBD information, but also provide access to surface normals. It seemed quite plausible to us that access to surface normals could speed up the process of aligning point clouds. This method however required the computation of normals for the point clouds. We did this using a Kd-Tree for PCL's "NormalEstimation"[11] function.

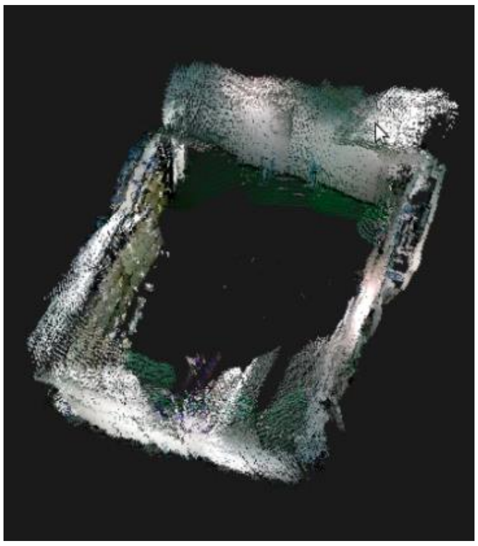

Figure 4 - Finally good alignment, with ICPNL

2.2.7. Results

PCL's ICPNL implementation was amazingly fast compared to all the other ICP implementations that we had tried. So fast even, that we could run multiple iterations and meanwhile vary the thresholds according to the results obtained from the algorithm. This enabled us to construct a point cloud from the test data within reasonable time and with a very satisfying result.

Next to this 3D map, our code produces two plain-text files. The first, called "indices", contains all the indices of the images that we have used for computation of the 3D map. The second, called "RT", contains all the RT matrices of the images corresponding to the ones listed in "indices".

3. Tasks per team member

Henry

Since all three of us were quite unfamiliar with Linux one of us had to spend a lot of time on that. Henry's main task in the localization challenge was to get all the code that was written in Windows compiled and running in Ubuntu. Next to that he contributed in research.

During the SLAM challenge he was familiar enough with Ubuntu, which enabled him to also contribute considerably more in research and coding.

Guan-Lin

During the localization challenge Guan-Lin did the major part of the coding work and also contributed in research. In the SLAM challenge Guan-Lin again did the major part of coding and implemented lots of the different ICP variants.

Sander

In the localization challenge Sander did a major part of the research and a minor part of coding work. During the SLAM challenge Sander also did a major part in coding work because of the implementation of all the various ICP algorithms. Sander did the major part of writing the report.

4. Conclusion

Despite finding much trouble in implementing functions that we assumed to be easy to implement, we managed to find a reasonable solution to the SLAM problem. Conclusively, because improving our PnP results took so much effort, the end result of our coding work is a collection of algorithms found by one, implemented by another and improved by the next person. It was very informative and satisfying to put the theoretical knowledge acquired in the course into practice.

5. Discussion

Although our results are reasonably well by now, there are still some improvements possible. For instance we have not considered the wheel odometry of the robot. There could be more precision if relative wheel odometry was used to correct the visual odometry. This is a missed opportunity especially when in the end we discovered that the odometry was much less messy than we thought.

Also, because the process of finding a suitable ICP algorithm was so intense in our case, we did not have the chance to properly finish our implementation loop detection. Improving this feature could have a major positive impact on the running time and resulting alignment of our SLAM algorithm.

6. References

(Web links verified to be operational on 01-14-2014)

[1] Asus Xtion Pro Live: http://www.asus.com/Multimedia/Xtion_PRO_LIVE/

[2] Point Cloud Library (1.6/1.7): http://pointclouds.org/

[3] SIFT: http://docs.pointclouds.org/1.7.0/classpcl_1_1_s_i_f_t_keypoint.html http://docs.opencv.org/modules/nonfree/doc/feature_detection.html

[4] SURF: http://docs.opencv.org/trunk/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html http://docs.opencv.org/modules/nonfree/doc/feature_detection.html

[5] OpenCV: http://docs.opencv.org

[6] PnP: http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

[7] Ransac: http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

[8] SLAM: Probabilistic Robotics, Sebastian Thrun, Wolfram Burgard and Dieter Fox - The MIT Press (August 19, 2005)

[9] ICP: http://pointclouds.org/documentation/tutorials/iterative_closest_point.php

[10] ICPNL Iterative Closest Point Nonlinear: http://pointclouds.org/documentation/tutorials/pairwise_incremental_registration.php

[11] NormalEstimation: http://docs.pointclouds.org/1.7.0/classpcl_1_1_normal_estimation.html