Mapping the College of Computing using a Kinect with an Existing SLAM Algorithm

Andrey Kurenkov, David Zhang, Daniel Keyes

April 19, 2013

1 Work Done

We have acquired a Kinect on a loan, set up ROS-groovy and the icthzasl_icp_mapper package, and have verified we could acquire and store point-clouds. Then, we used this algorithm to acquire measurements of the College of Computing Building (CoC) with five different paths to evaluate the quality of evaluation and mapping.

2 Algorithm Used

The ethzasl_icp_mapper library is not just a SLAM algorithm but in fact a more ambitious framework. It allows an icp algorithm to be run in a modular nature, so that several steps of the process can be implemented in different ways. The default provided approach uses libpointmatcher (an ICP framework[**?**]) and libnabo (an octree nearest neighbors library[**?**]). These topics are essentially what we learned in class with respect to ICP.

The default algorithm takes two point clouds, one of them a model and the other sensor data, and tries to find a set of common points between them to join them together coherently[**?**]. It does so through iterative error minimization, and if it cannot find a sufficiently good match, it throws out the sensor data. Otherwise, adds the sensor data to the model. The algorithm steadily builds up a model in this fashion until it has a coherent image. In addition to creating a model, the algorithm also uses the offset of the sensor data to estimate the sensor motion.

3 Software Implementation

In order to record Kinect data and run the SLAM algorithm, we had to first set up ROS. This proved easy on Ubuntu, but either impossible or problematic on others OSes or VMs. After setting up ROS, we had to install the Openni package of drivers for Kinect as well as the algorithm package. Although the installation was as easy as using get-apt install, some additional effort was needed to get Kinect to work. After everything was set up, we could simply use the ROS commands roslaunch and rosbag record to run the algorithm and store data as outlined below.

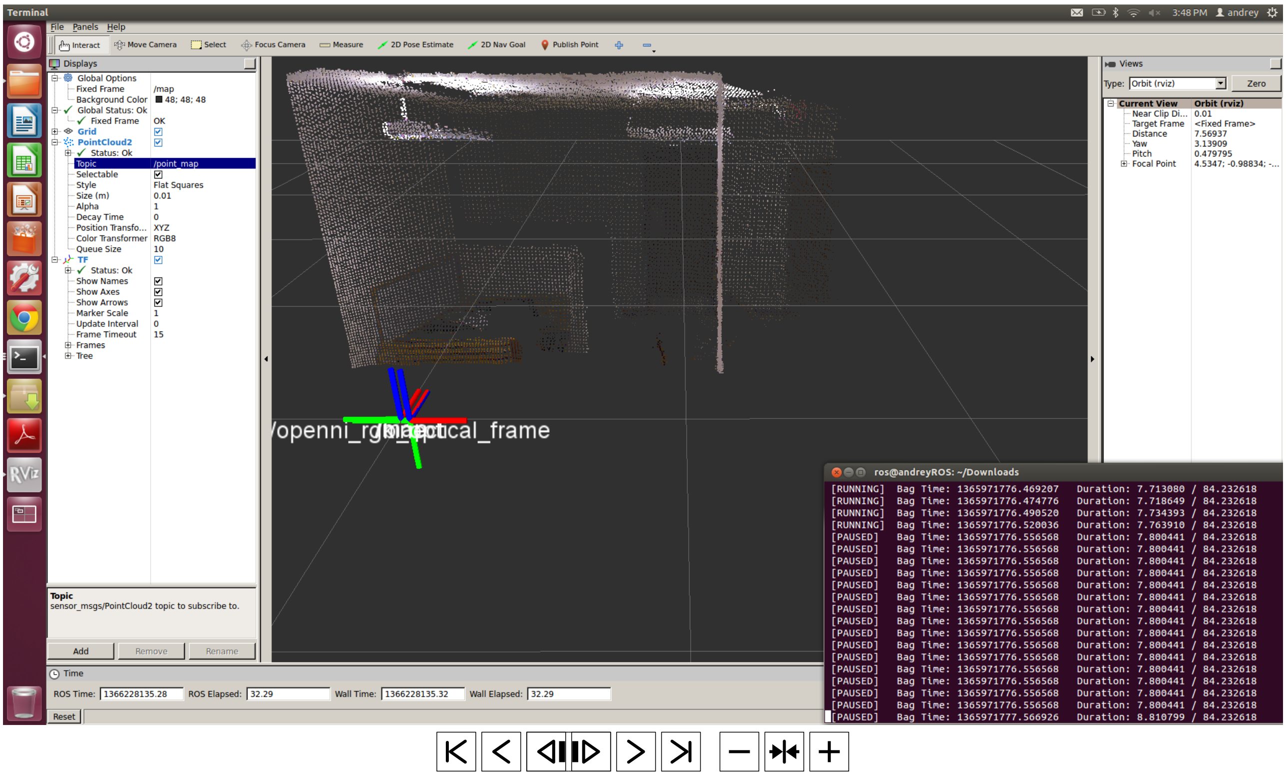

Figure 1: Development of a point cloud with movement to the right

4 Data Collected

In order to evaluate the effectiveness of this algorithm, we collected a total of five sets of data with different traversed paths within the CoC. One person held the Kinect at about chest height and walked steadily through the same section of the CoC in five different paths. Data was collected by another person who ran roslaunch with a provided launch files that came with the icp_mapper package in conjunction with rosbad record. The algorithm ran with data coming straight from the kinect. We attempted to store Kinect readings and run the algorithm with those, but the algorithm required the timestamp of the data to be the present and so we had no choice but to simultaneously collect and process data.

Since different paths were walked while carrying the kinect, the localization quality of the algorithm can be evaluated with respect to different types of motion. The distance traversed by the person holding the Kinect was measured in tiles for convenience. A point cloud was succesively built during each run, for a total of five maps.

5 Evaluation

Table 1 presents the quantitative results of our multiple runs, with run five being divided into 3 phases of forward movement, rotation, and further movement. Relative errors were calculated to represent the significane of error for differnt runs. Absolute error are also included for cases where actual displacement is zero, and show that the error is still significant. It should be noted that that measurement of actual displacements are

Run	Person Movement	SLAM Movement	Absolute Error	Relative Error
1	(0m,0m,90°)	(1.19m,1.19m,45°)	(1.19m,1.19m,45°)	(NaN,NaN,0.50)
2	(3.3528m,0m, 0°)	(3m,0m,0°)	(0.353m,0m,0°)	(0.11,0,0)
3	(0m,2.7432m,0°)	(0m,2.379m,0°)	(0m,0.365m,0°)	(0,0.13,0)
4	(1.3716m,1.3716m,0°)	(1.66m,1.66m,0°)	(0.29m,0.29m,0°)	(0.21,0.21,0)
5.1	(-1.524m,0m,0°)	(-1.46m,0m,0°)	(0.06m,0.29m,0°)	(0.031,0,0)
5.2	(0m,0m,90°)	(-2m,0m,70°)	(0m,0m,20°)	(NaN,0,0.22)
5.3	(0m,1.524m,0°)	(0m,1.7m,0°)	(0m,0.176m,0°)	(0,0.12,0)

Table 1: Localization results for experiments

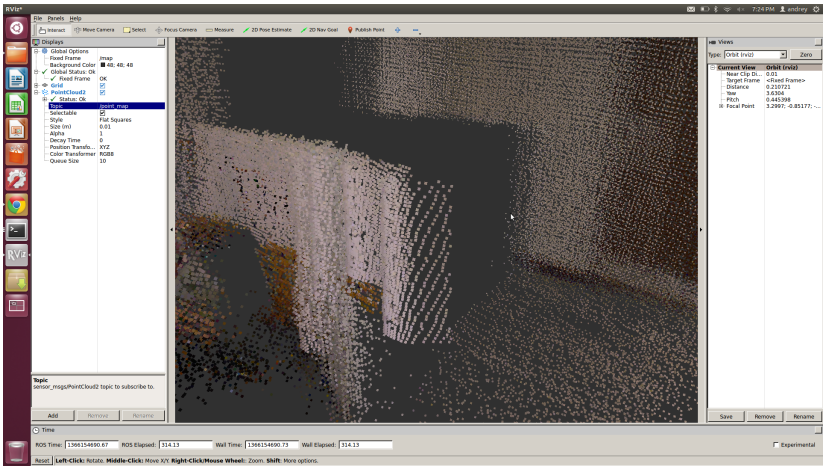

Figure 2: Error in 3D map with due to overlapping wall surfaces

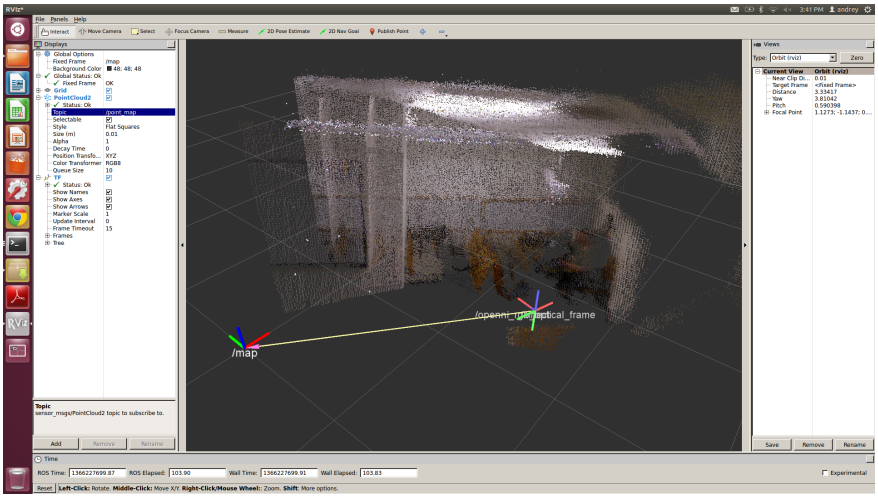

Figure 3: Larger error that resulted during a rotation-only run

not exact, since they were originally based on tiles passed while walking. However, the error margin is not great. Figures 2 and 3 show portions of pointclouds with clear error to be discussed later, figure 4 once again showed the development of a pointcloud for a different path from figure 1, and figures 5 and 6 show a map of the enviroment and an actual photo of the actual location for comparison.

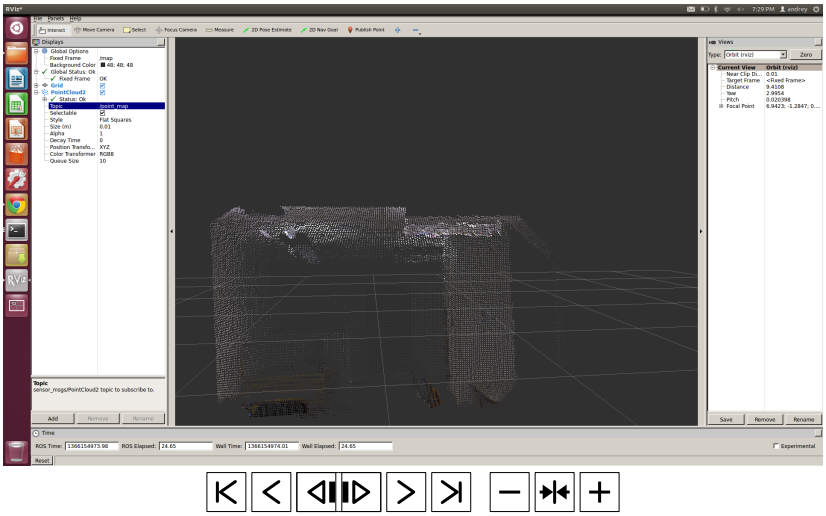

Figure 4: Progression of point cloud for diagonal movement

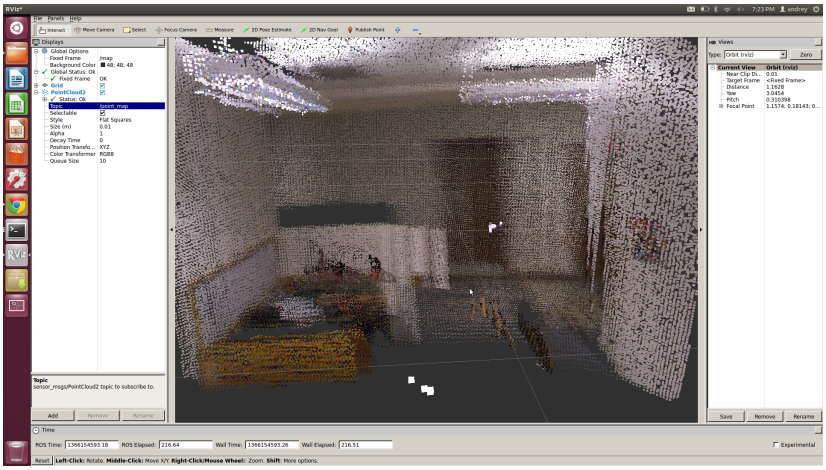

Figure 5: Final result of 3D map generation

Figure 6: Photo presenting the enviroment mapped

6 Discussion

The methodology of our tests limits the trustworthiness of our data, since the camera was held only roughtly steady and distanced traversed was not measured very precisely.

If done again, we would probably use a pushcart to hold the robo and precisely delineate the distances. However, we are still able to reach some overwhelmnigly clear conclusions from our data. Mainly, this algorithm works best when involving only translation and has large amounts of error for rotation. The algorithm only succesfully registers small amount of rotation, and larger amounts inevitabely lead to mistaken translation along with insuficient rotation. A possible reason for this is the fact that rotation to the right with a fixed Z axis leads to translation to the left in the image, which could be interpreted by as translation by the algorithm. Another possibility is that for large rotations, the change in the scene causes the algorithm to interpret the items in the background as different sizes instead of the image becoming skewed.

The mapping component is not as clearly flawed as localization, and is harder to judge quantitively. Figures 2 and 3 display the typical errors that occur in point clouds during mapping, with figure 3 display especially large error. A major source of error that can be seen in the generated maps is the tendency to almost overlay surfaces but with slight translations or rotations(as in figure 3). Large errors in localization lead to errors such as seen in figure 3, through smaller errors can be due to problems with point cloud scaling where part of the sensor cloud fits well with the model cloud which causes another portion of the sensor cloud to be slightly misaligned. Another problem occurs during perceptual aliasing, when the camera is moved very quickly, which causes the new area to become overlayed over an old one which clutters the a previously good model with garbage. This is likely because the camera does not have a good sense of its actually location, so it has no way of telling if a similar scene is new or old if the camera is moved quickly.

Additional tests to collect more data could have been done, including varying the speed of movement and the rate of rotation to see how it affects the quality of localization and mapping. Likewise, a more steady approach to moving the kinect and better measurement capabilities would improve the viability of our data. Still, the data we did collect clearly delineates the strenghts and weaknesses of the algorithm

References

[1] http://publications.asl.ethz.ch/files/pomerleau11tracking.pdf

[2] http://www.ros.org/wiki/ethzasl_icp_mapper

[3] http://stephane.magnenat.net/publications/Comparison%20of%20nearest-neighbor-search%20strategies%20and%20implementations%20for%20efficient%20shape%20registration%20-%20Elseberg%20et%20al.%20-%20JOSER%20-%202012.pdf