Exploration of Mapping Algorithms on a TurtleBot

Heather Boortz

April 2015

1 Introduction

Using a TurtleBot, I explored 2D mapping algorithms with various conditions. The mapping algorithms were evaluated for their effectiveness given these various conditions.

2 Interfacing with TurtleBot

2.1 Setup

I used a TurtleBot with a Kobuki base running Ubuntu 14.04 and ROS Indigo. All of the hardware was left unmodified. Setting up the software involves several steps. The ROS TurtleBot tutorials [7] provide instructions for the basic setup of the TurtleBot.

I set up the onboard TurtleBot computer as both the TurtleBot and PC workstations. The network configuration worked to some extent but was not efficient to use without having a fixed IP for both machines. I set up a fixed IP for the TurtleBot and used basic SSH to teleop from a remote machine.

In order to get data from the Kinect, OpenNI drivers must be installed. Specific instructions can be found on my github wiki [2].

3 Mapping Algorithms

I saved rosbag files for three test runs. One was intended to get a full map of the area, referred to as the slow run, one an obviously incomplete map, referred to as the fast run, and one somewhere in the middle, referred to as the medium speed run. This would highlight differences between the mapping algorithms based on the completeness of the data. Each of these rosbag files was run with both GMapping and Hector mapping, and maps were generated in RViz.

3.1 Gmapping

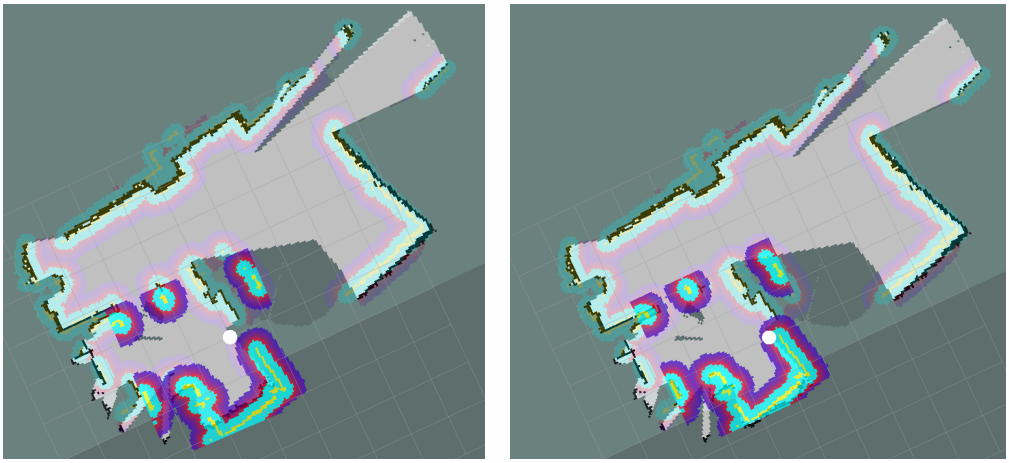
Gmapping is an implementation of a particle filter [4], and comes as a default demo on the TurtleBot. For the purposes of this project, the out-of-box code was used. Future work could include doing more work with tuning parameters, and/or modifying the algorithm slightly to optimize it for a given area.

3.2 Hector Mapping

Hector mapping is another SLAM implementation available as a ROS package. It can be used for a wider variety of applications including systems where sensors or platforms have roll and pitch, and can be used without odometry[6]. It does not come as a default demo on the TurtleBot, but can be run on the TurtleBot by editing the launch files [6]. Hector mapping is generally used with higher update rate sensor, but can be used with the laser scan data from the Kinect.

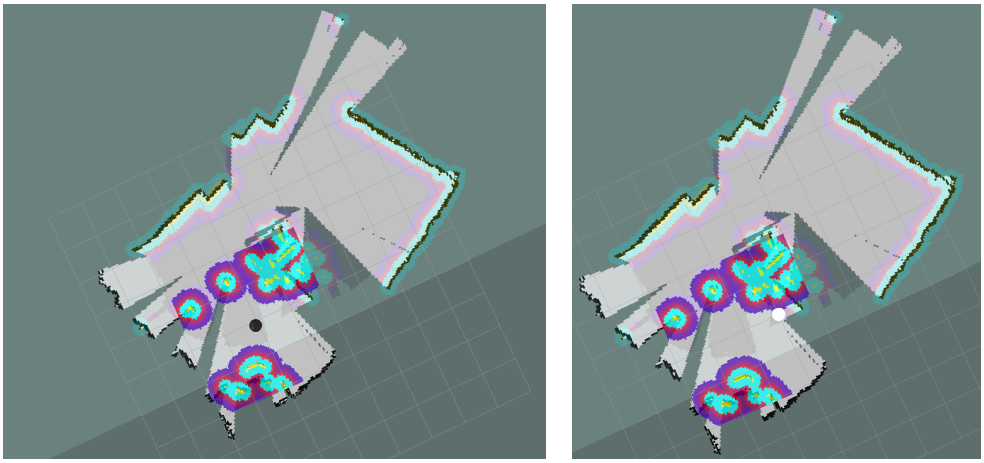
3.3 Results

Each data set is shown as a side-by-side comparison in Figures 1, 2, and 3. The blueprint of the area is shown in Figure 4 as a reference.



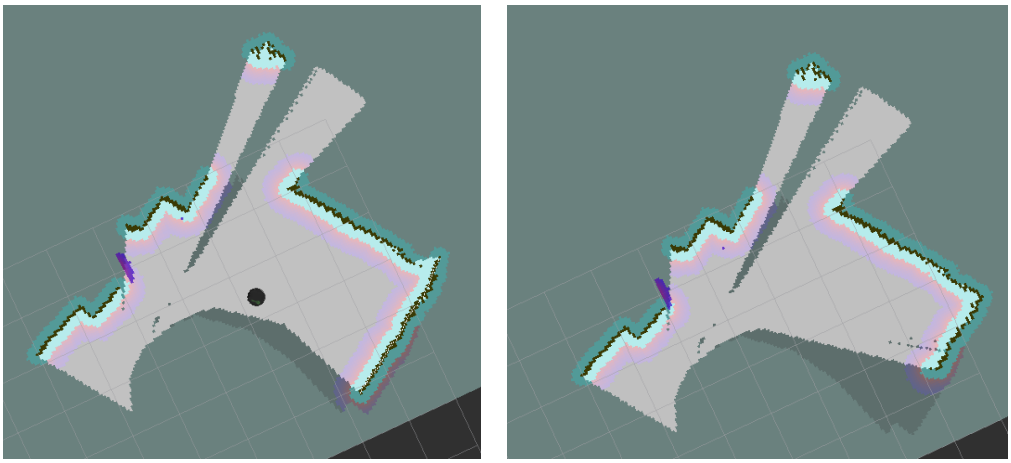
Gmapping *Hector*

Figure 1: The maps for each algorithm on the slowest run.



Gmapping *Hector*

Figure 2: The maps for each algorithm on the medium speed run.



Gmapping *Hector*

Figure 3: The maps for each algorithm on the fast run.

3.4 Comparison

For the most part, the maps look essentially the same. The only map that shows any substantive difference is the fastest run. Gmapping performs significantly better on this run. This is likely either due to Hector requiring more data in order to calculate where a wall is, or that there is a greater time lag in the Hector mapping system.

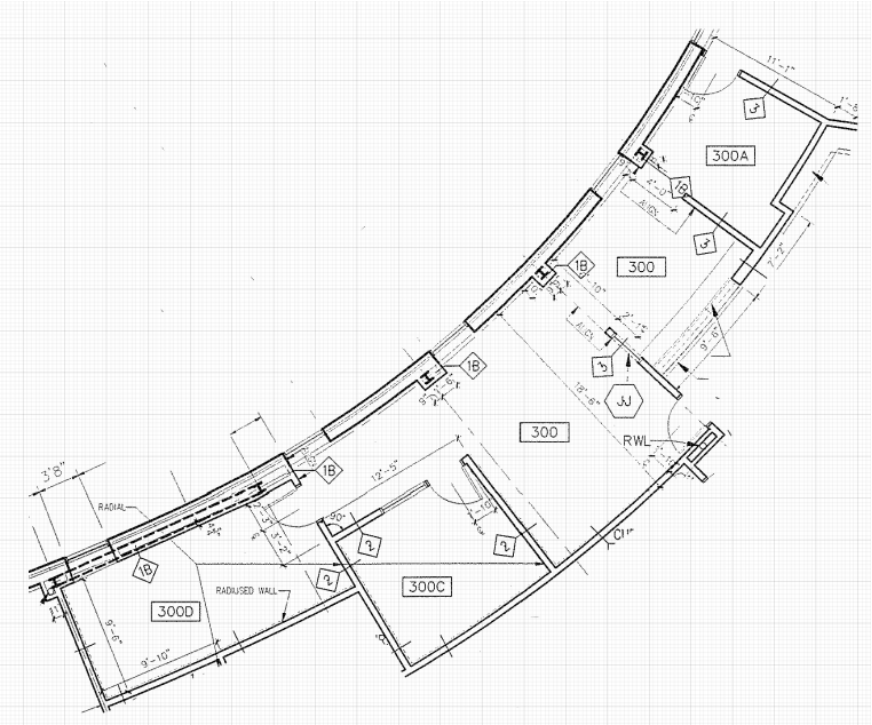


Figure 4: The blueprint of the area where the map data was gathered. Rooms 300C and the left part of 300 were the only areas the TurtleBot was driven through, though the Kinect is fairly long range and gathered data all the way to the walls of the other rooms when there was line of sight.

The direction of the Kinect influences the resulting map significantly more than the mapping algorithm does in these cases. Since the Kinect has such a narrow field of view, the TurtleBot can drive through an area and end up with an extremely inaccurate map. In order to get an accurate map with the Kinect, especially the way that it is set up, the TurtleBot has to face each wall in order to fully detect it.

4 Pitfalls

4.1 Deprecated software

The TurtleBot initially had a lot of old software running on it. Installing Ubuntu 14.04 and ROS from scratch solved many of those problems. Starting with a

fresh install is highly recommended.

4.2 Kinect Drivers

Since the Kinect is a Microsoft product, the Ubuntu drivers are somewhat finicky. The support for 14.04 is also still not very good. I ended up using a 3rd party fork of the PrimeSense drivers which can be downloaded from the avin/SensorKinect GitHub repository [3].

4.3 Simulation Time vs. Real Time

In order to play back rosbag files, ROS has to be set to use sim_time, the timestamps that appear in the rosbag file. This can be done either via command line with `rosparam set use_sim_time true` or in the launch file

4.4 Hector Mapping

I played around a lot trying to get hector mapping to work. I tried to write a script that would grab the kinect data and run that through Hector mapping, but it turns out that just modifying the launch file and includes folder in the `turtlebot_navigation` package can make the TurtleBot use Hector mapping instead of Gmapping

5 Conclusions

The ability of the TurtleBot to map an area is severely limited by the Kinect data. The mapping algorithm made negligible difference in most cases. Getting a higher update rate, 360° lidar sensor would definitely improve both the capabilities of the system as well as the ability to compare and contrast SLAM implementation differences.

References

[1] https://github.com/hboortz/TurtlebotMapping

[2] https://github.com/hboortz/TurtlebotMapping/wiki

[3] https://github.com/downloads/avin2/SensorKinect/SensorKinect093-Bin-Linux-x64-v5.1.2.1.tar.bz2

[4] http://openslam.org/gmapping.html

[5] http://wiki.ros.org/gmapping

[6] http://wiki.ros.org/hector_mapping

[7] http://wiki.ros.org/turtlebot/Tutorials/indigo/Turtlebot%20Installation