# Map My World Robot


## Abstract

The project applies Simultaneous Localization And Mapping (SLAM) algorithm to localize robot and get mapping simultaneously. SLAM is an important and challenge task for robot-like objects. Robot, drone, and self-driving all face this kind of chicken-and-egg problems to find out poses and mapping.

The project uses Real-Time Appearance Based Mapping (RTAB-Map) that is a RGB-D SLAM approach based on a global loop closure detector with real-time constraints. RTAB-Map can create 2D occupancy grid map and 3D Octomap for navigation.


## Background

SLAM is to handle challenge problems for robotics that neither environment map is available nor is the knowledge of its own pose. Without environment map and known pose is a typical chicken-and-egg problem. First, we are going to review couple of algorithms.

### 1. Occupancy Grid Mapping Algorithm

Occupancy Grid Mapping Algorithm is based on a known pose of robot. It uses binary Bayes filter to loop through every cell with previous belief plus log odd of the inverse measurements model minus initial belief. Fig. 1 shows a diagram with perceptual view from a robot with occupancy grid mapping algorithm. The white cells are for free for robot to navigate.

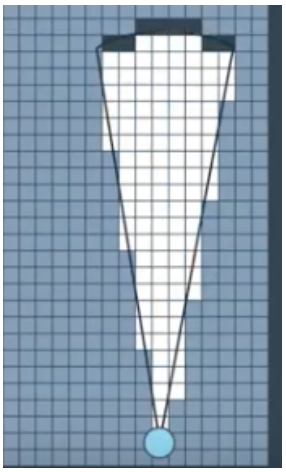


Fig.1 perceptual view by occupancy grid mapping algorithm

### 2. FastSlam Algorithms

There are two forms of SLAM, Online SLAM and full SLAM.
Online SLAM solves the posterior represented by the instantaneous pose at time t with the map given the measurement and control at time t only.
Full SLAM solves the posterior represented by the instantaneous pose at time t with the map given the measurement and control throughout the robot travel time.
The FastSlam algorithm solves the full SLAM problem with known correspondences.

- Estimating the trajectory using a particle filter, Monte Carlo Localization (MCL)
- Estimating the Map using a low dimensional Extended Kalman Filter (EKF) to solve independent features of the map which are modeled with local Gaussian.

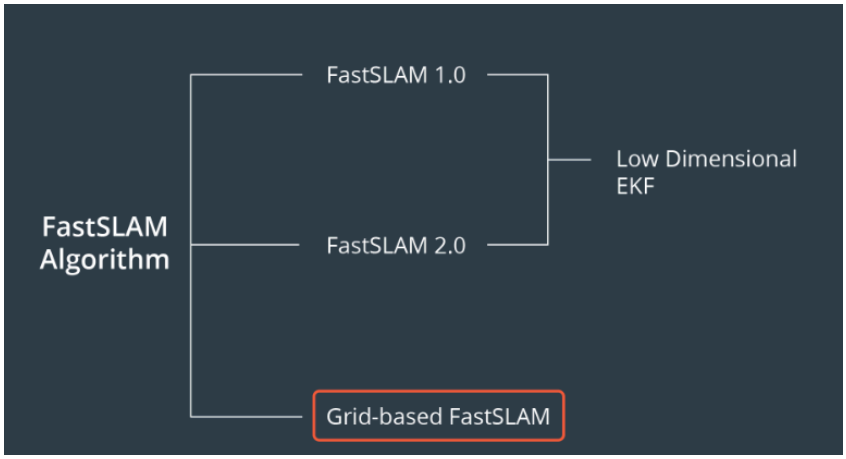



Fig. 2 Three Types of FastSLAM Algorithms

FastSLAM 1.0 is simple and easy to implement, known to be inefficient since particle filters generate sample inefficiency.

FastSLAM 2.0 overcomes the inefficiency of FastSLAM 1.0 by imposing a different distribution, which results in a low number of particles. Both 1.0 and 2.0 use low dimensional EKF to estimate the posterior over the map features.

Grid-based FastSLAM adapts FastSLAM to grid maps by modeling the environment using grid maps without predefining any landmark position. By extending the FastSLAM algorithm to occupancy grid maps, Grid-based FastSLAM can solve the SLAM problem in an arbitrary environment.

Each particle maintains its own map. The grid-based FastSLAM algorithm will update each particle by solving the mapping with known poses using occupancy mapping algorithm.

We need three techniques to adapt FastSLAM to grid mapping.

- Sampling Motion: estimates the current pose given the k-th particle previous pose and the current controls u.
- Map Estimation: estimates the current map given current measurements, the current k-th particle pose, and the previous k-th particle map.
- Important Weight: estimates the current likelyhood of measurement given the current k-th particle pose and the current k-th particle map.

### 3. GraphSlam

GraphSLAM solves the full SLAM problem by covering full trajectory instead of the most recent pose and map. GraphSLAM improves accuracy over FastSLAM. FastSLAM uses particles, while there might not be enough particles in certain place needed. With GraphSLAM, it'll be able to optimize over the full trajectory along with constraints.

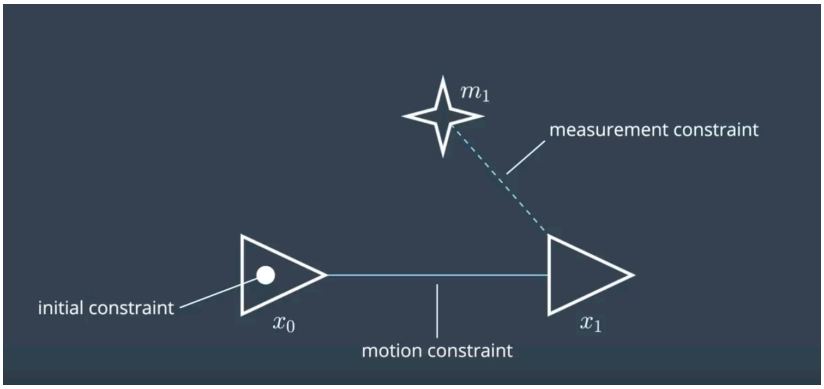



Fig. 3 A Simple Graph

Fig. 3 shows a simple graph. Triangle stands for pose. Star stands for an environment feature like landmark. Solid line represents a motion constraint. Dashed line represents a measurement constraint.

GraphSLAM usually have two parts, frond-end and back-end. The front-end of GraphSLAM looks at how to construct the graph using the odometer and sensory measurements collected by the robot. The front-end can differ greatly from application to application depending on the desired goal, sensors, accuracy, and environment. The back-end is similar to most applications, taking output of whole trajectory from the front-end. The back-end performs the optimization process that takes the constraints and find the system configuration that produces the smallest errors.

At the core of GraphSLAM is graph optimization - the process of minimizing the error present in all of the constraints in the graph.

Both motion constraints and measurement constraints can be non-linear. To simplify computation, Taylor Series can be used to linearize them. To reduce errors due to linearization, every constraint must occur as close to the true location of the pose or measurement relating to the constraint as possible. An iterative loop can be used to improve it with every iteration. After certain iteration, a reasonable estimation can be an acceptable solution for the true locations of all robot  poses and features.

RTAB-Map

Real-Time Appearance-Based Mapping (RTAB-Map) is a RGB-D Graph SLAM approach based on global Bayesian loop closure detector. The loop closure detector uses a bag-of-words approach to eliminate how likely a new image comes from a previous location or a new location.

Fig. 4 shows front-end and back-end of RTAB-Map. The front-end takes inputs from sensors and odometer, construct graph, and detect loop closure. The back-end performs graph optimization and generate 2D/3D map.

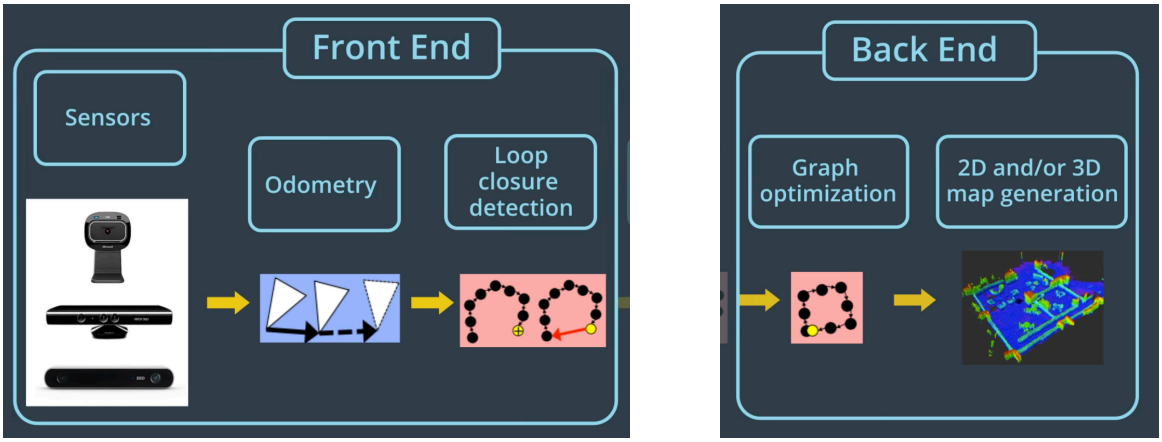



Fig. 4 Front End and Back End of RTAB-Map

## Results

The Robot that made in **localization project** has been used for the project. The camera has been enhanced with RGBD capability from Kinect and libgazebo_ros_openni_kinect.so library. Camera has been moved up 0.05m in the z-axis to get a better view. Fig. 5 shows frames of the robot.

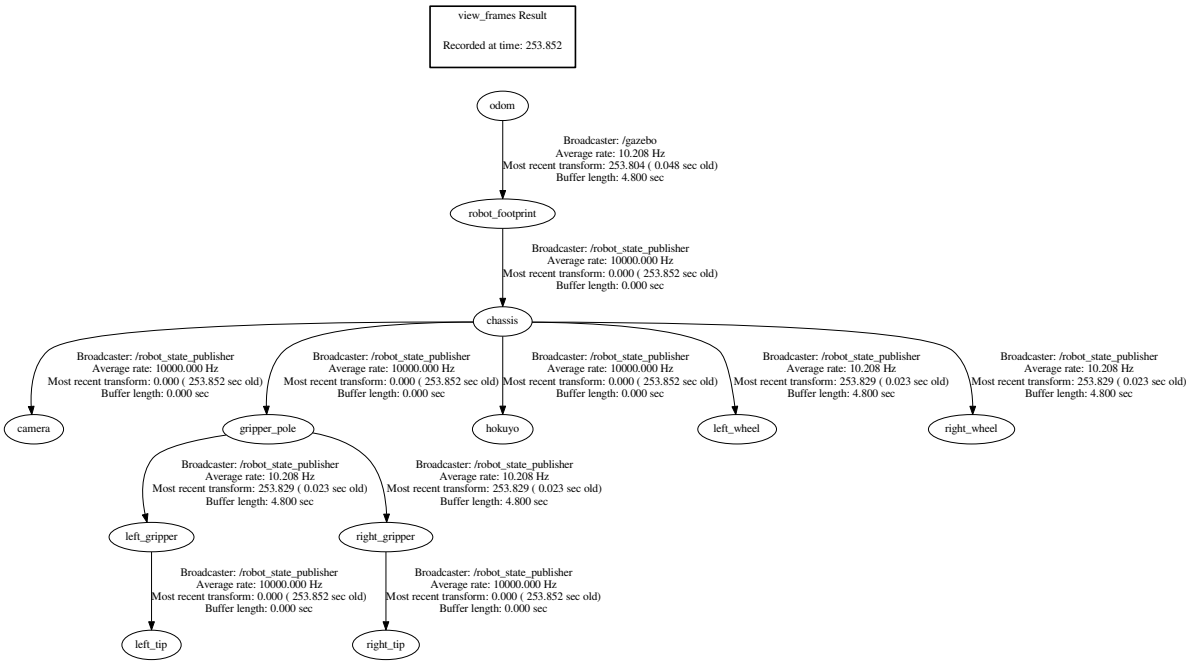



Fig. 5 Robot Frames

There are two tests one for the provided kitchen dining room and the pother from custom layout.

Fig. 6 shows the kitchen dining rooms in Gazebo. Fig. 7 shows three screenshots for the corresponding Rtabmap views with Occupancy Grid, 3D View, Graph View, and Constraints View. Fig. 8 shows rViz for the corresponding test.

14 Global Loop Closures were detected in the test.

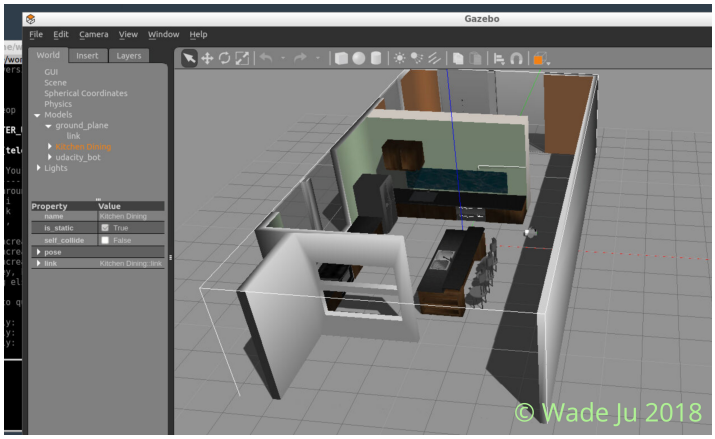
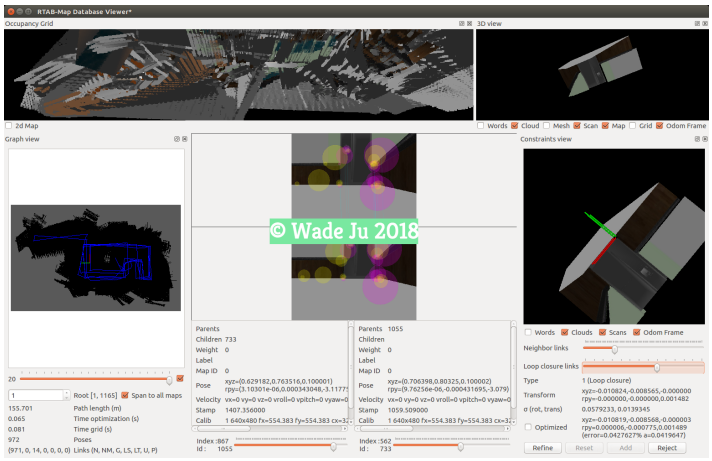
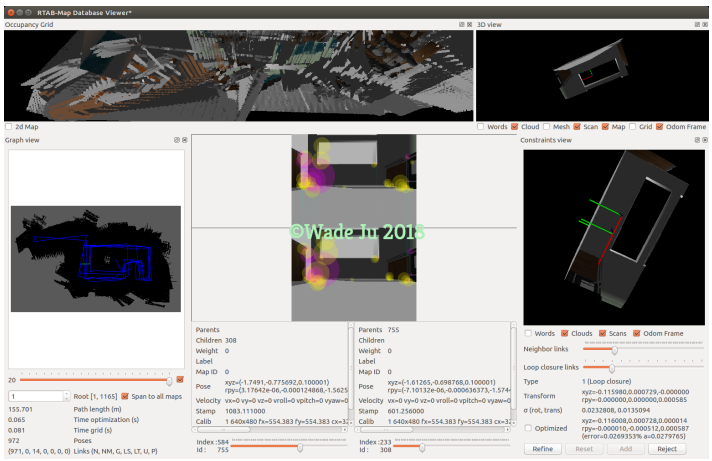



Fig. 6 Kitchen Dining in Gazebo



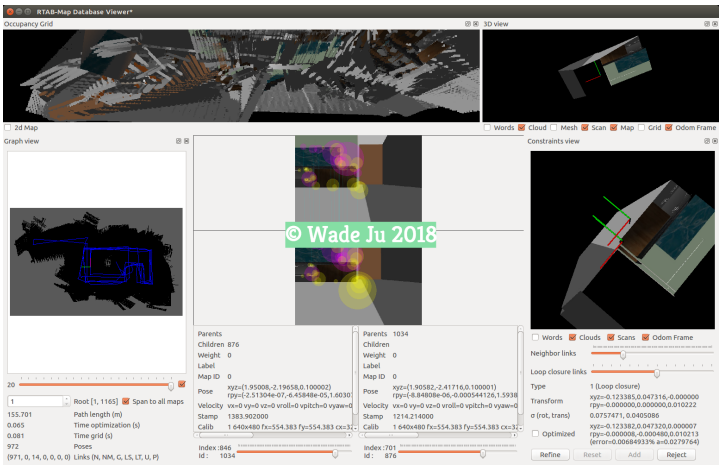



Fig. 7 Rtabmap for Kitchen Dining Room

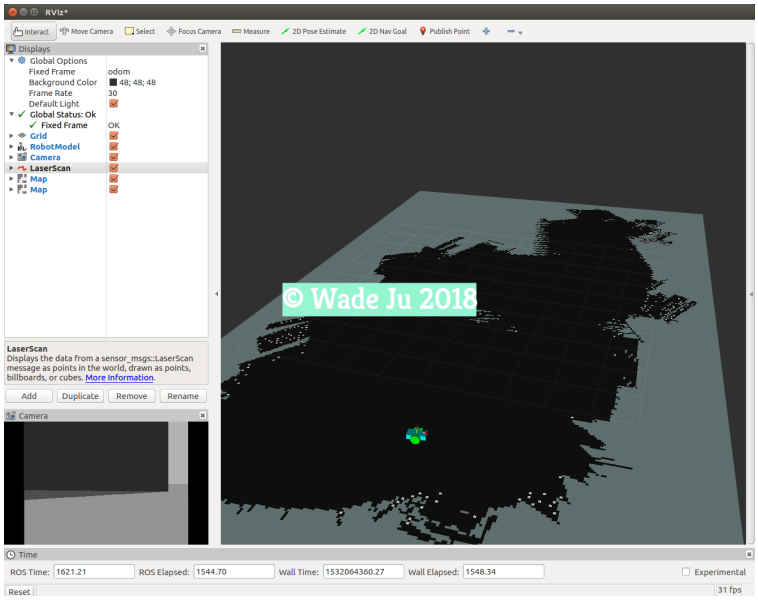



Fig. 8 Rtabmap for Kitchen Dining Room in RViz

Fig. 9 shows a custom rectangle shape structure on Gazebo. The custom building was built with four walls in different colors or textures. Couple of furnitures were added including table, bookshelf, lumber, and a cylinder to make it feature rich for better loop detection.

Fig. 10 shows the corresponding RTabmap view. There were 24 global loop closures detected in the testing. Fig. 11 shows the corresponding testing on rViz.

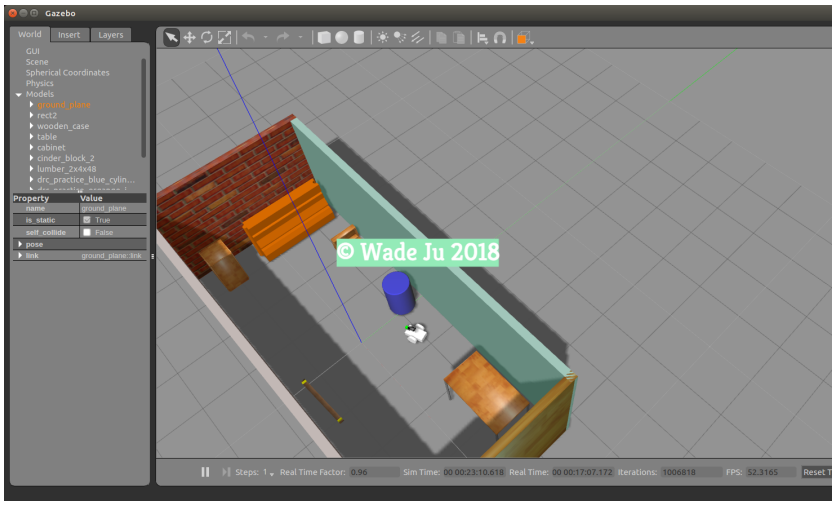



Fig. 9 Custom Building with Rectangle-Shape Building in Testing

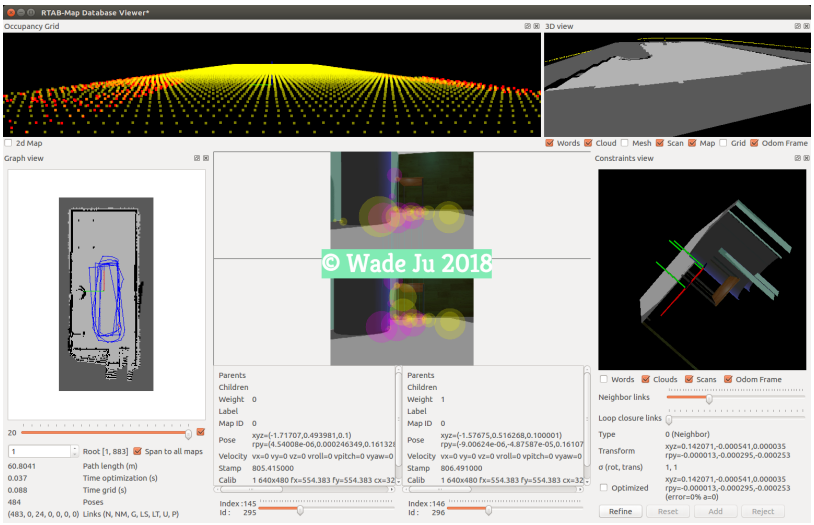



Fig. 10 RtabMap View for the Custom Building

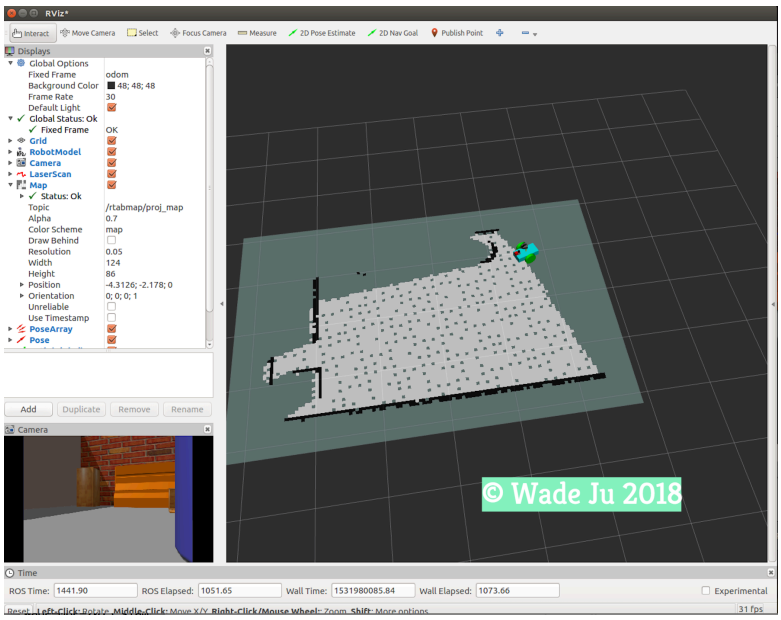



Fig. 11 rViz for the Custom Building

**Discussion**

By manually moving robot to get mapping is a labor intensive task with questionable reliability. I continuously ran into swinging robot which turned too much left or too right. It tuned out getting different images from different angles.

The provided kitchen and dining room needs more computation power. Parameter “Vis/MinInliers" was lowered to 10 in the testing to get results shown above.

I have tried couple times for my rectangle shape to get results as shown above. First, I set different colors and textures around four walls and added a ball and a cylinder. It never detected loop closure. Then I added several furnitures including table, bookshelf, lumber, etc to make it feature rich. It turned out getting 10 more loop closures than Kitchen and Dining room. As it takes less computation power, it’s tested with Vis/MinInliers equal to 15.

The laser range finder doesn’t seem to work in long distance. In kitchen and dining rooms, there are so many furnitures and close walls to bounce back to get better results. I had another U shape world. It didn’t work well. For open space, laser range finder doesn’t seem to work as well.

The positions of camera and laser ranger finder are also important. I moved up camera 0.05m up in the z-axis to get a better view although it worked well in the prior localization project.

**Future**

There are more sensors available for robot, self-driving, and drone. Prices are going down. It'll help SLAM. The project is simple in both seniors and environment with Gazebo simulator.

For 3D, by integrating with object recognition and detection, robot can potentially do more work besides traditional mapping assuming enough processing power is provided. By integrating with Computer Vision and Machine Learning, a robot will be able to do lots of intelligent works.

To have a auto navigating robot feature is definitely a big win. I got tired of navigating robot around with keyboard.

To work with hardware and more sensors will bring SLAM to next level.

**Reference**

Sebastian Thrun and  Michael Montemerlo
The GraphSLAM Algorithm with Applications to Large-Scale Mapping of Urban Structures, https://disco.ethz.ch/courses/fs14/seminar/paper/Pascal/1.pdf

Cyrill Stachniss, Robot Mapping, http://ais.informatik.uni-freiburg.de/teaching/ws12/mapping/pdf/slam20-frontends-4.pdf

https://medium.com/@NickHortovanyi/map-my-world-robot-66cbcc2241f8

ROS RTAB-Map Wiki, http://wiki.ros.org/rtabmap

Udacity Robotic Nanodegree lectures