SLAM

Shu-man Lin, *Student at Udacity*

***Abstract*—A robot fitted with a Kinect RGB-D camera and hokuyo laser maps two different simulated environments created in Gazebo using the RTAB-Map tool integrated with real-time visualization in RViz. In addition, localization is launched so that the robot can know its position with respect to the map that it creates. This procedure of SLAM or simultaneous localization and mapping is explored in this project to solve the problem of navigation in unknown environments. A slam_project package is created which includes world files, launch files, urdf files, configuration files, scripts, and mesh files needed to perform SLAM with a simulated robot. Assembly of urdf files to include sensor plugins and sensor frames that control the properties and topics of sensors, building world files with Gazebo's model database, and remapping topics, setting parameters for nodes, and adding static tf transforms to launch files for effective rqt_graph communication and tf tree construction are some of the tasks that are completed to organize the slam_project package for proper execution.**

***Index Terms*—Graph-SLAM, 2-D occupancy grid mapping, 3-D point cloud mapping, RTAB-Map, RViz**

I. Introduction

FIND a robot's pose with respect to a known map and find a map with respect to a robot's known pose. These two problems in robotics: localization and mapping can be solved individually with algorithms like MCL, EKF, and occupancy grid mapping. In situations where both the robot's pose and map are unknown, the problem generalizes to what is known as the SLAM problem or simultaneous localization and mapping in which the robot is tasked with constructing an unknown environment while simultaneously keeping track of it's location within the map it makes. Given the robot's state at time t x_t, measurements from an initial time to time t $z_{1:t}$, the controls from an initial time to time t $u_{1:t}$, and the map m, the problem of localization is solved with algorithms that set out to calculate the belief of a state: $p(x_t|z_{1:t}, u_{1:t})$ which depends on measurement updates respecting the robot's placement with respect to a map or landmarks as represented by: $p(z_t|m, x_t)$ and motion updates with respect to the map as estimated by: $p(x_t|u_t, x_{t-1}, m)$. In the problem of mapping, the occupancy grid mapping algorithm calculates the probability: $p(m|z_{1:t}, x_{1:t})$. Finally, the online SLAM problem involves the estimation of the probability of the state at time t and the map given the controls and measurements over all time: $p(x_t, m|z_{1:t}, u_{1:t})$ while the full SLAM problem is formulated by finding the probability of the map and the state over the entire path given controls and measurements over all time: $p(x_{1:t}, m|z_{1:t}, u_{1:t})$ [1].

Acquiring an accurate map and a robot's place within it is difficult for several reasons. For one, the space of all possible maps as estimated by the relative position of the robot to indicator landmarks is a particularly high dimensional continuous space together with the discrete correspondence of identifying which landmarks have been previously detected. This high dimensional continuous parameter space and large number of discrete correspondence variables makes it challenging to calculate full probabilities over maps. In fact, considering discrete correspondences obtained from an initial time to time t $c_{1:t}$, the full SLAM problem becomes one in estimating the probability: $p(x_{1:t}, m, c_{1:t}|z_{1:t}, u_{1:t})$ while the online SLAM extends to calculating the probability: $p(x_t, m, c_t|z_{1:t}, u_{1:t})$. For another, learning a pose estimate together with a map estimate concurrently is hard because of the initial uncertainty of both. While the dual distinct problems of finding a position relative to a known map and finding a map relative to known positions are both solved, removing the knowledge of both robot pose and map makes it difficult to realize either to which one may ask: which estimate comes first, map or pose? Further, limitations of sensors due to short range or sensitivity to noise and misclassifying observed perceptual symmetries of the environment such as similar looking areas and cycles add to the error in mapping which augment inaccuracies that lead to confused navigation. To resolve these issues multiple solutions have been developed to solve the full SLAM or online SLAM problems to obtain 2-D and 3-D maps alongside robot localization. Examples of such methods are FastSLAM, Grid-based FastSLAM, and GraphSLAM coupled with loop closure algorithms [1], [2], [3].

In this project, RTAB-Map together with sensor information gathered from a Kinect RGB-D camera and hokuyo laser are used to obtain a 2-D occupancy grid map and 3-D point cloud of two simulated environments. RTAB-Map or Real-Time-Appearance-Based mapping is an algorithm based on visual Graph-SLAM theory and is noted for its additional features of global loop closure detection, graph optimization, and memory management [4]. The visualization tool RViz is employed in conjunction with RTAB-Map to realize robot localization and mapping in real-time thereby solving the SLAM problem. In section **II.** the background for mapping algorithms is discussed and their limitations are examined. Section **III.** explains slam_project package compilation, mapping parameter choices, sensor features, and the make of the Gazebo world. Section **IV.** includes the results for mapping processes with final 2-D and 3-D map illustrations of both Gazebo worlds. In section **V.** a discussion is provided on procedure specifics and gives a comparison of the performance of RTAB-Map in different worlds with methodologies for improvement. Finally, section **VI.** looks towards future work where the technology can be deployed in the real world and open problems in the field.

Shu-man Lin, e-mail: impetusengine@gmail.com

II. BACKGROUND

As explained in [1] EKF SLAM solves the online SLAM problem by the following steps. Recall that belief in EKF is defined by $bel(x_t) := p(x_t|z_{1:t}, u_{1:t})$ where x_t is a pose and $z_{1:t}$, $u_{1:t}$ are sensor measurements and controls respectively. EKF represents this belief by a Gaussian with center μ_t and covariance Σ_t. EKF SLAM estimates the belief of the pose and landmarks together by a single Gaussian updated by the EKF algorithm. With correspondences known, landmarks are uniquely updated without repeat. Without correspondences c_t known, an extra parameter gauging map size is used to decide whether a new landmark should be added. It is determined to be added if the Mahalanobis distance to all existing landmarks exceeds a set value. Running through the algorithm explicitly, first, the initial pose of the map is set to be the center of the map so that the initial belief of $y_0 = (x_0, m_0)$ of the pose x_0 and landmarks $m_0 = \theta_{1,0}..., \theta_{n,0}$ is represented by a Gaussian centered around an initial mean μ_0 equal to the 0 state with initial covariance Σ_0 having absolute certainty of its initial pose but absolute uncertainty of the yet to be discovered landmarks. Subsequently, estimates of the belief of $y_t = (x_t, m_t)$ of the robot pose x_t together with each landmark in $m_t = \theta_{1,t}, ..., \theta_{n,t}$, is represented by a Gaussian with center μ_t and covariance Σ_t. They are updated by linearized formulas of motion and sensor measurement to produce the next estimate via the EKF algorithm. Finally, the correspondence algorithm can be integrated if correspondences are not known. The final Gaussian representation, probabilistically estimating poses and landmarks, produces an online map and localization of the robot. A limitation to the EKF approach is its computational complexity. Sensor updates per time increment require a modification in Kalman filter covariance with $O(n^2)$ elements corresponding to the n landmarks, all of which need to be updated even if only one landmark is sighted. This limits the number of landmarks that can be recorded to be a few hundred at any given time. In addition, correspondences are only determined once for each sensor measurement which means false data association will cause mapping to fail. To ameliorate these difficulties, FastSLAM is introduced.

In FastSLAM [5], full SLAM is solved by first assuming the robot's path $x_{1:t}$ is known and the correspondence variables $c_{1:t}$ are known. These conditions allow for the splitting of the posterior:

$$p(x_{1:t}, m|z_{1:t}, u_{1:t}, c_{1:t}) = p(x_{1:t}|z_{1:t}, u_{1:t}, c_{1:t}) \prod_{k=1}^{n} p(\theta_k|x_{1:t}, z_{1:t}, u_{1:t}, c_{1:t})$$

The benefit to this method is that estimation is broken down into two parts: finding a path estimator of $p(x_{1:t}|z_{1:t}, u_{1:t}, c_{1:t})$ and finding n landmark pose estimators $p(\theta_k|x_{1:t}, z_{1:t}, u_{1:t}, c_{1:t})$, one for each landmark θ_k. FastSLAM employs a particle filter for estimating the path posterior $p(x_{1:t}|z_{1:t}, u_{1:t}, c_{1:t})$ and represents the conditional landmark estimates $p(\theta_k|x_{1:t}, z_{1:t}, u_{1:t}, c_{1:t})$ by Kalman filters. Thus, for m particles and n landmarks, there will be a total of mn Kalman filters and m samples of the robot pose as represented by the sample set:

$$S_t = \{x_{1:t}^{[i]}, \mu_1^{[i]}, \Sigma_1^{[i]}, ..., \mu_n^{[i]}, \Sigma_n^{[i]}\}_{i=1}^{m}$$

where $\mu_i^{[m]}$ and $\Sigma_i^{[m]}$ are the mean and covariance of the Gaussian representation of the i-th landmark θ_i attached to the m-th particle. Each particle $x_t^{[m]}$ represented by the probabilistic model $p(x_t|u_t, x_{t-1}^{[m]})$ is obtained by sampling from the motion model with a replacement probability proportional to the importance factor:

$$w_t^{[m]} = \frac{p(x_{1:t}^{[m]}|z_{1:t}, u_{1:t}, c_{1:t})}{p(x_{1:t}^{[m]}|z_{1:t-1}, u_{1:t}, c_{1:t})}$$

This implies an algorithm that requires $O(mn)$ time but adding a tree-based data structure can reduce the running time of FastSLAM to $O(mlog(n))$. Counting for unknown correspondences, data association is estimated per particle by setting $c_{1:t}^{[m]} = argmax_{c_{1:t}} p(z_{1:t}|x_{1:t}^{[m]}, c_{1:t}, z_{1:t-1}, u_{1:t})$ which is called the maximum likelihood estimator (ML). This is interpreted as a distance function with Mahalanobis distance and selects data associations by minimizing this distance. ML and the alternative DAS method is explained further in the reference [6]. In contrast to EKF SLAM, the particle filter approach is able to pursue multiple correspondences simultaneously so that one erroneous guess of a data association is, in the resampling process, effectively eliminated. The primary drawback of FastSLAM is that it must always assume that there are known landmark positions and therefore FastSLAM is unable to model arbitrary environments.

Even without predefined landmark positions, the SLAM problem may still be solved by extending FastSLAM to occupancy grid maps called Grid-based FastSLAM. With Grid-based FastSLAM, robot trajectory is also approximated by a particle filter, however, the map attached to each particle is then updated by solving the mapping with known poses problem using the occupancy grid mapping algorithm. In [7], Grid-based FastSLAM is presented with an added algorithm to deal with unknown correspondences. The methodology is explained in the following steps. First, a decomposition of full SLAM into probabilistic trajectory prediction and map prediction is given by:

$$p(x_{1:t}, m|z_{1:t}, u_{1:t-1}) = p(m|x_{1:t}, z_{1:t}) p(x_{1:t}|z_{1:t}, u_{1:t-1})$$

The algorithm starts with a sampling step and takes in the previous belief to calculate a collection of m particle estimates of the current pose depending on the previous control and pose estimates with the motion model $p(x_t^{[k]}|u_{t-1}, x_{t-1}^{[k]})$ representing the k-th particle for $k = 1, ..., m$. Next, a correspondence detecting algorithm is implemented based on the map $m_{t-1}^{[k]}$ starting from the initial sampling of $x_t^{[k]}$. If no correspondence is detected, the pose and weights are computed according to the motion model. If correspondence is detected, a new pose for each particle is drawn from a Gaussian distribution created from a sampling of points about the reported correspondence matcher. Next, the importance weights of the particle filter are

calculated according to the observation model $p(z_t|m^{[k]}x_t^{[k]})$ for all times t. Then the map $m^{[k]}$ of particle k is updated according to the drawn pose $x_t^{[i]}$ and the observation z_t via the grid-mapping representation of $p(m^{[k]}|x_{1:t}^{[i]}, z_{1:t})$. Finally, a resampling step is carried out where particles are drawn with replacement proportional to their importance weight.

Although the previous algorithm produces accurate mapping results with limited initial constraints on mapping assumptions, the question remains of whether offline algorithms that solve the full SLAM problem can be filtered to be fast enough to run online. GraphSLAM as introduced in [2] and [3] is an offline SLAM solution solving the full SLAM problem but can be modified to an online SLAM solution as shown with the SEIF algorithm [1]. The basic idea of the algorithm consists of constructing a graph where robot poses x_t at sequential times t and map features $m = \{m_j\}$ represent two vertex types that admit two different edge connections depending on whether a connected pair of vertices are both robot poses or a pose and a map feature. The edges of the graph and the initial vertex then correspond to equations representing an initial constraint which anchors the initial pose to the origin of the global coordinate system, motion constraints which are in one-to-one correspondence with the quantities $p(x_t|x_{t-1}, u_t)$ for each pose at time t, and measurement constraints which correspond to the quantities $p(z_t^i|x_t, m, c_t^i)$ for each pose at time t and i-th measurement where, as before, x_t, u_t, z_t, m, and c_t represent the robot pose, controls, measurement, map, and correspondences respectively at time t. In fact, these quantities decompose the problem of calculating the negative log of the target probability: $p(x_{0:t}, m|z_{1:t}, u_{1:t}, c_{1:t})$ into a sum of measurement constraints, motion constraints, and initial constraint which takes the following form with normalization constants omitted:

$$\begin{aligned} J_{GraphSLAM} = x_0^T \Omega_0 x_0 + \sum_t (x_t - g(u_t, x_{t-1}))^T \\ R_t^{-1}(x_t - g(u_t, x_{t-1})) \\ + \sum_t \sum_i (z_t^i - h(y_t, c_t^i, i))^T \\ Q_t^{-1}(z_t^i - h(y_t, c_t^i, i)) \end{aligned}$$

where g is the kinematic motion model of the robot, h is the measurement function, Q_t is the covariance of measurement noise, R_t is the covariance of motion noise, and $x_0^T \Omega_0 x_0$ is the initial constraint that anchors the absolute coordinates of the map. The corresponding summands of $J_{GraphSLAM}$ are essentially equal to:

$$\begin{aligned} & p(x_t|x_{t-1}, u_t) \\ & = \eta exp\{\frac{-1}{2}(x_t - g(u_t, x_{t-1}))^T R_t^{-1}(x_t - g(u_t, x_{t-1}))\} \\ & p(z_t^i|x_t, m, c_t^i) \\ & = \eta exp\{\frac{-1}{2}(z_t^i - h(y_t, c_t^i, i))^T Q_t^{-1}(z_t^i - h(y_t, c_t^i, i))\} \end{aligned}$$

Minimizing $J_{GraphSLAM}$ yields the most likely map and most likely robot path. To numerically solve for the path and map, the functions g and h are linearized and the resulting solution is encoded in an information matrix Ω and information vector ξ defined over the full space of map features and poses. Next, based on the known factorization:

$$\begin{aligned} p(m, x_{0:t}|z_{1:t}, u_{1:t}, c_{1:t}) = p(x_{0:t}|z_{1:t}, u_{1:t}, c_{1:t}) \\ p(m|x_{0:t}, z_{1:t}, u_{1:t}, c_{1:t}) \end{aligned}$$

Ω and ξ are reduced to $\tilde{\Omega}$ and $\tilde{\xi}$ which is defined over the space of all poses but not the map. This is achieved by removing features m_j of the map m. Finally, the mean and covariance for all poses in the robot path and a mean location estimate for all features in the map is computed. Based on the former reduction, the mean of the posterior on the robot path, $p(x_{0:t}|z_{1:t}, u_{1:t}, c_{1:t})$, is given by:

$$\tilde{\mu} = \tilde{\Omega}^{-1}\tilde{\xi}$$

To recover the second factor: $p(m|x_{0:t}, z_{1:t}, u_{1:t}, c_{1:t})$, observe the decomposition of Ω and ξ consisting of the sub-matrices $\Omega_{m,m}$ restricting to map variables and $\Omega_{m,x_{0:t}}$ connecting robot path and map variables is given by:

$$\Omega = \begin{bmatrix} \Omega_{x_{0:t},x_{0:t}} & \Omega_{x_{0:t},m} \\ \Omega_{m,x_{0:t}} & \Omega_{m,m} \end{bmatrix}$$

$$\xi = \begin{bmatrix} \xi_{x_{0:t}} \\ \xi_m \end{bmatrix}$$

so that the mean is given by:

$$\mu_m = \Omega_{m,m}^{-1}(\xi_m + \Omega_{m,x_{0:t}}\tilde{\xi})$$

However, μ_m is not necessary to find and it is enough to get by with an estimate of the mean and covariance over the entire path. Furthermore, in the case where correspondences are unknown, a correspondence test can be added to the algorithm to check for data associations.

The simulation experiments done in this paper rely on RTAB-Map's GraphSLAM based algorithm together with loop closure detection using a bag-of-words approach and graph optimization together with efficient memory management to produce real time online 3-D SLAM visualization, solving both mapping and localization as the robot navigates. We also obtain a 2-D occupancy grid through RTAB-Map's full SLAM solution.

III. Model Configuration

Successful execution of RTAB-Map to solve the SLAM problem for robots in Gazebo simulation requires the assembly of several files and the installation of three main packages. The packages rtabmap_ros and rtabmap must be installed in the workspace following the instructions in [8]. The package slam_project is constructed to include robot specifications, world specifications, launch files, a script for teleoperation, CMake file, package.xml file and meshes for sensors. The package.xml file defines properties about the package such as name, version, authors, maintainers, and dependencies [9]. The only dependency needed here is catkin. CMakeLists.txt builds the code within packages by specifying dependencies and telling where to install it. More about this can be found in the ROS Wiki [10].

The urdf folder contains a .xacro file that specifies the collision data of the robot in Gazebo and the visualization

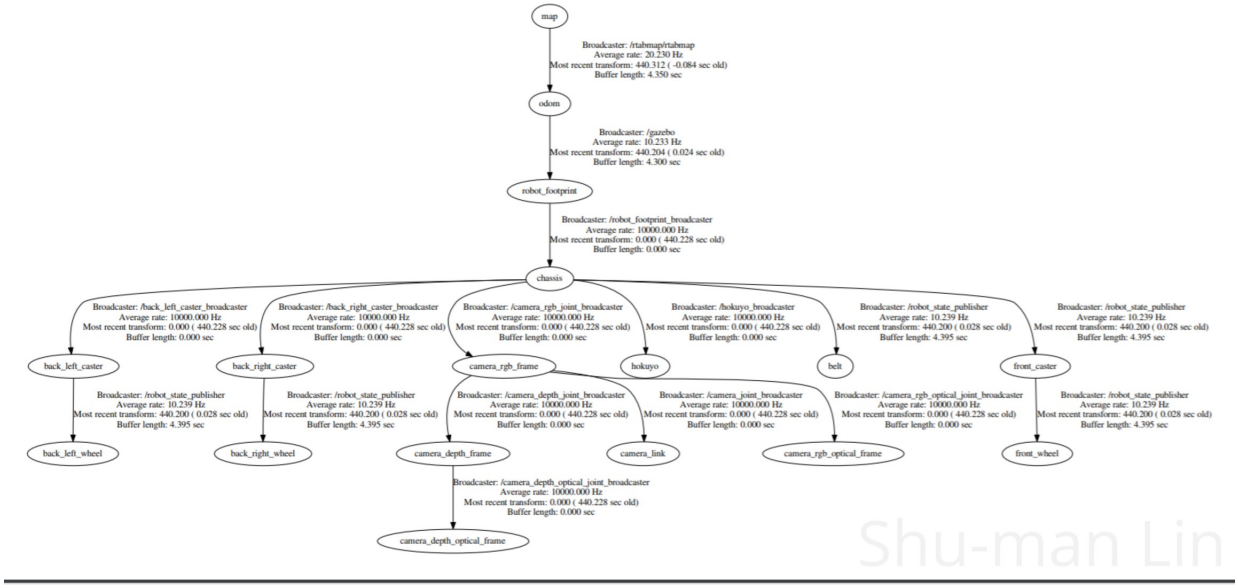

Fig. 1. tf tree

data needed for RViz. The robot's build in terms of links, joints, and sensors are all specified in this file. The .gazebo file within lists the Gazebo plugins that are needed for sensor simulation and differential drive wheel simulation. In particular, physical constraints and mapping topics for the hokuyo laser and the the RGB-D camera are specified. In addition, the meshes folder includes detailed sensor properties for the Kinect camera and hokuyo laser in .dae files which are called in the geometry tag of the .xacro urdf file when specifying the visualization for the sensors. On the other hand, the descriptor of the worlds are provided in the worlds folder where kitchen_dining.world contains the physical setup of the kitchen world and collapsed.world is a world that was custom made from Gazebo modeling features. Kitchen world imports physical models of the ground plane, sun, and kitchen dining room. Collapsed world used Gazebo gui where the sdf text associated to each object selected in Gazebo was automatically compiled. A collapsed house was selected as the base for robot exploration and objects such as mars rovers, warehouse robots, and a table were selected from the model interface to be placed within the floor plan of the house. The final construction was saved in an .world file.

The scripts folder contains a teleop.py file that contains the details of a teleop node specifying remote controls for the robot and publishes the topic /cmd_vel. Within the launch file, teleop.launch specifies a remapping of the velocity topic the teleop node publishes to, to the command topic for velocity subscription in gazebo. This in effect allows for the robot to be controlled by keyboard in Gazebo. Other contents in the launch file include a robot_description.launch file that calls the node joint_state_publisher together with the node robot_state_publisher to publish transforms for all non-fixed joints of a robot. In addition, static_transform_publisher is called to publish joint states that are fixed. Both world.launch and my_world.launch include robot_description.launch and each launch a unique Gazebo world together with a newly spawned robot as described in the urdf script: slam_project.xacro. Kitchen dining world is launched from world.launch and collapsed world is launched from my_world.launch. Furthermore, for visualization in RViz, rviz.launch opens the RViz node with configuration of RViz classes prepared for mapping provided in the robot_slam.rviz file contained in the config folder. Finally, mapping.launch sets the RTAB-Map parameters to subscribe to the sensor topics, provide the correct frame ids, and save the database path. Under the RTAB-Map node, it also remaps RTAB-Map sensor topics to Gazebo plug-in sensor topics, sets loop closure detection parameters, activates SLAM mapping, regulates the rate at which nodes are added and sets mapping to occur only when the robot is moving. In addition, a Rtabmapviz node is also added with proper subscription to scan and depth topics and remapping of standard sensor topics to the sensor topics associated to the sensors provided with Gazebo plug-ins. Definitions for relevant topics, services, and parameters in RTAB-Map are catalogued on the rtabmap_ros Wiki [11]. Also, another launch file called localization.launch mirrors the content of mapping.launch except that parameter is activated to put the robot in localization mode.

Launching the robot within the world via world.launch or my_world.launch produces a tf transform tree of the robot shown in Fig. 1. Launching world.launch, teleop.launch, mapping.launch, and rviz.launch and running rqt_graph results in the interconnection of active nodes and topics in Fig. 2. For proper RTAB-Map configurations, a tutorial is provided to set-up RTAB-Map on your robot here [12].

IV. Results

The ground truth of the kitchen dining world and collapsed world that were mapped while teleoperated in Gazebo simulation are shown in Fig. 3 and Fig. 4 respectively. In the

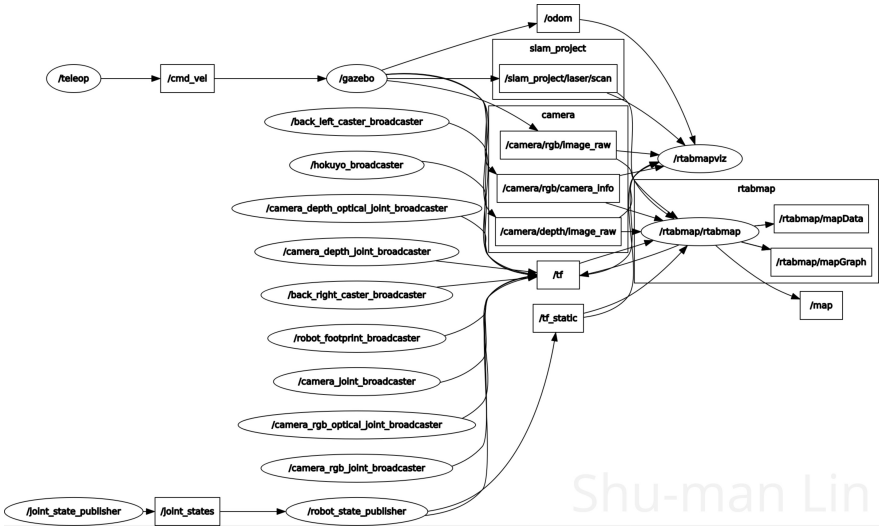

Fig. 2. Rqt graph

collapsed world, only the two right most situated rooms were mapped since there was a physical barrier on the inside of the house blocking the robot from exploring the third room.

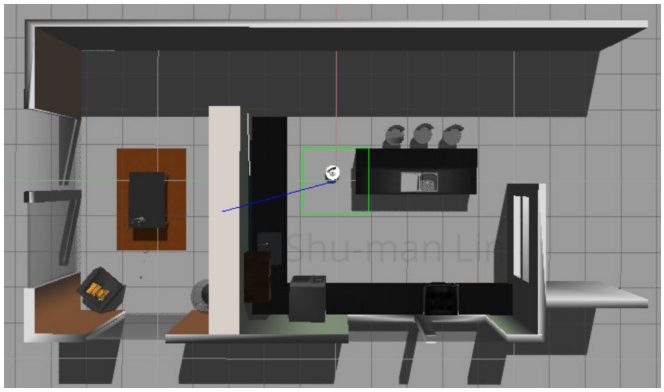

Fig. 3. Kitchen Dining World ground truth

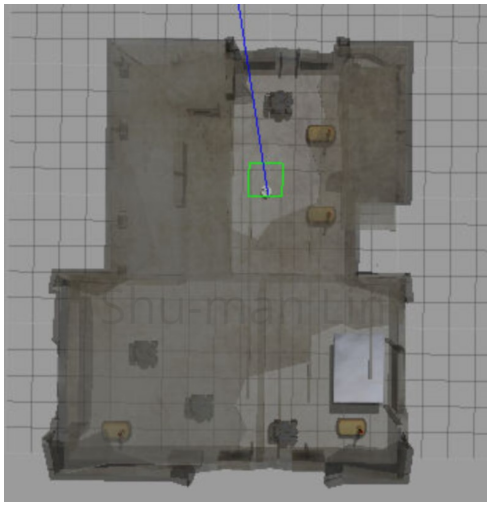

Fig. 4. Collapsed World ground truth

Collected from rtabmap.db, the two dimensional occupancy grid mapping for kitchen world and collapsed world are shown in Fig. 5 and Fig. 6

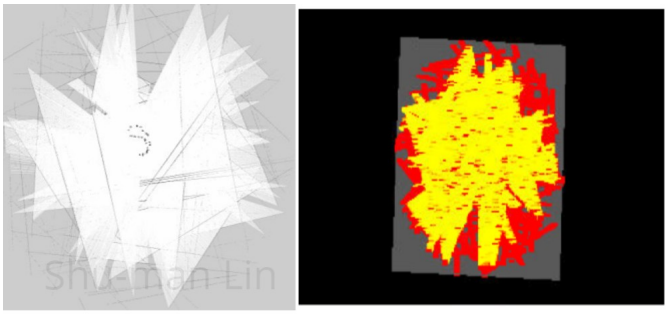

Fig. 5. Kitchen Dining World 2D Occupancy Grid from rtabmap.db

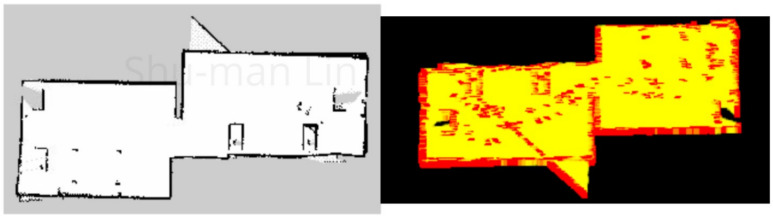

Fig. 6. Collapsed World 2D Occupancy Grid from rtabmap.db

In addition, mapping was obtained by adding the Grid/3D and Grid/FromDepth RTAB-Map parameters to be true in mapping.launch. In this case, a 2D occupancy grid mapping was obtained by projecting a 3D occupancy grid mapping to two dimensions. The results are as follows, Fig. 7 shows the kitchen dining world mapped with 3D occupancy grid mapping and 8 shows the collapsed world mapped with 3D occupancy grid mapping while Fig. 9 is the projection of the

3D mapping to 2D for kitchen dining world and Fig. 10 is the projection for collapsed world. All images here are obtained from rtabmap.db.

Fig. 7. Kitchen Dining World 3D Occupancy Grid from rtabmap.db

Fig. 8. Collapsed World 3D Occupancy Grid from rtabmap.db

Fig. 9. Kitchen Dining World 3D Occupancy Grid projection from rtabmap.db

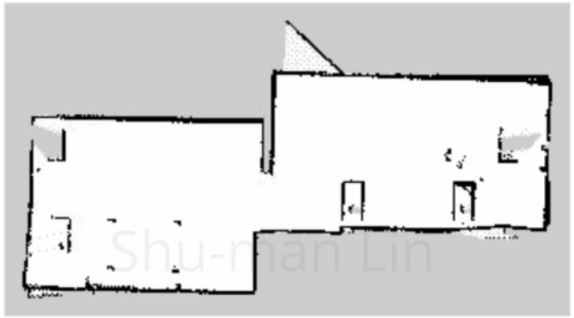

Fig. 10. Collapsed World 3D Occupancy Grid projection from rtabmap.db

Recorded in RViz, are occupancy grid mappings generated with hokuyo laser scan obtained from a remapping of the RTAB-Map published topic grid_map to the topic map. These images for kitchen dining world and collapsed world are given in Fig. 11 and Fig. 12.

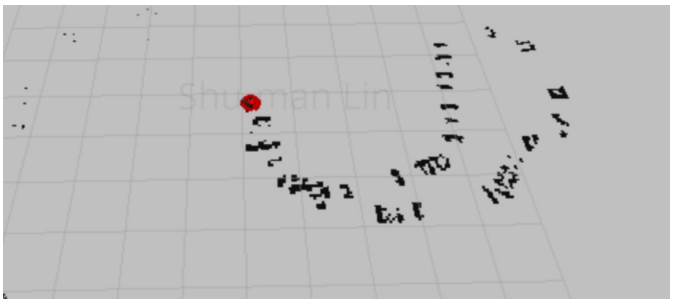

Fig. 11. Kitchen Dining World 2D Occupancy Grid in RViz

Fig. 12. Collapsed World 2D Occupancy Grid in RViz

RViz grid_map mappings when Grid/3D and Grid/FromDepth are turned to be true are given by Fig. 13 and Fig. 14.

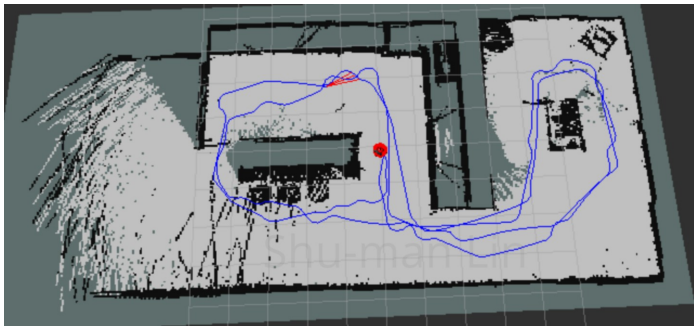

Fig. 13. Kitchen Dining World 3D to 2D Occupancy Grid in RViz

Fig. 14. Collapsed World 3D to 2D Occupancy Grid in RViz

RViz visualization of RTAB-map mapData and GraphData includes simultaneous mapping and path tracking in real-time using GraphSLAM with laser and camera data. RViz 3D cloud maps on runs in which Grid/3D and Grid/FromDepth are turned to false are given by Fig. 15 for the kitchen dining world and 16 for the collapsed world.

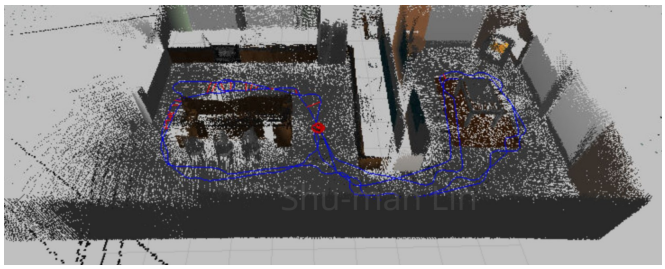

Fig. 15. Kitchen Dining World 3D cloud map in RViz without Grid/3D and Grid/FromDepth

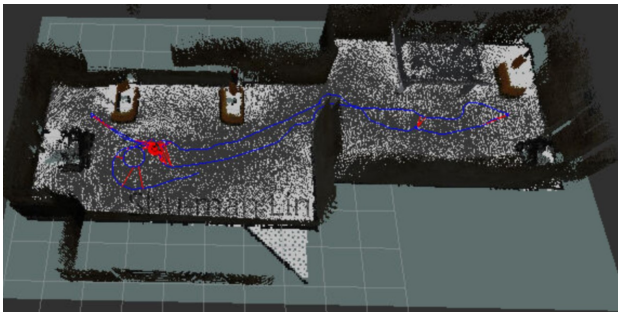

Fig. 16. Collapsed World 3D cloud map in RViz without Grid/3D and Grid/FromDepth

With Grid/3D and Grid/FromDepth turned on, RViz cloud maps turned out as portrayed in 17 and 18.

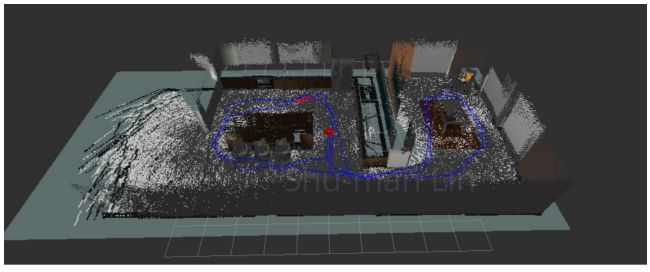

Fig. 17. Kitchen Dining World 3D cloud map in RViz with Grid/3D and Grid/FromDepth

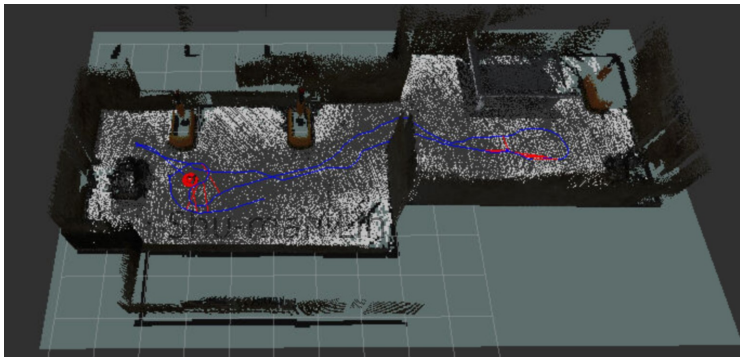

Fig. 18. Collapsed World 3D cloud map in RViz with Grid/3D and Grid/FromDepth

In addition, rtabmap.db provides a construction of a 3D map made from RGB-D camera images. The result of this 3D view for kitchen world in the footage taken without Grid/3D and Grid/FromDepth is given in Fig. 19 and the result for collapsed world is given in Fig. 20. In the case where Grid/3D and Grid/FromDepth were turned on, Fig. 21 and 22 show the outcome of each world's 3D RGB-D compiled maps.

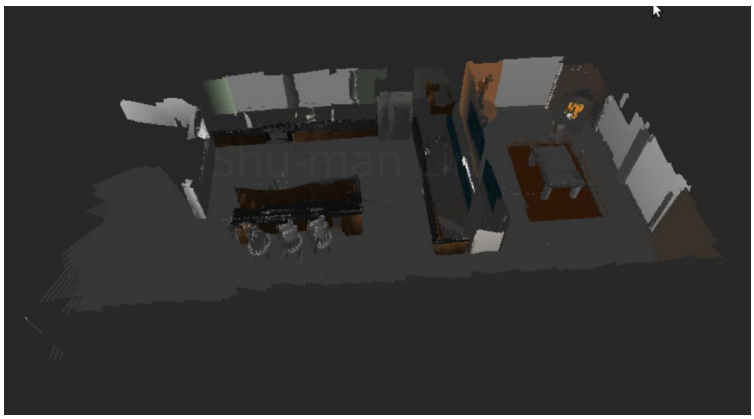

Fig. 19. Kitchen Dining World 3D RGB-D map without Grid/3D and Grid/FromDepth

Fig. 20. Collapsed World 3D RGB-D map without Grid/3D and Grid/FromDepth

Fig. 21. Kitchen Dining World 3D RGB-D map with Grid/3D and Grid/FromDepth

Fig. 22. Collapsed World 3D RGB-D map with Grid/3D and Grid/FromDepth

Finally, a report on the number of loop closures detected on each run is listed as follows. The kitchen dining world without Grid/3D and Grid/FromDepth found a total of 26 global loop closures see Fig. 23. The collapsed world without Grid/3D and Grid/FromDepth found a total of 41 global loop closures see Fig. 24. The kitchen dining world with Grid/3D and Grid/FromDepth found a total of 3 global loop closures see Fig. 25. The collapsed world with Grid/3D and Grid/FromDepth found a total of 16 global loop closures see Fig. 26.

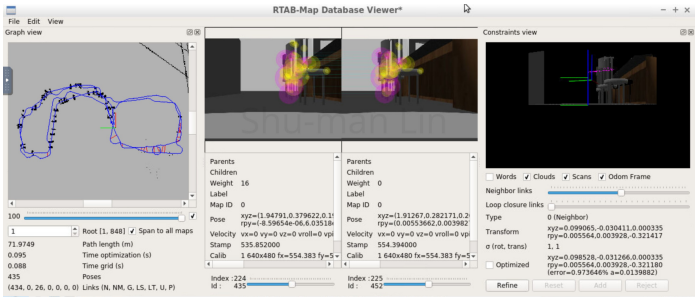

Fig. 23. Kitchen Dining World RTAB-Map Database Viewer without Grid/3D and Grid/FromDepth

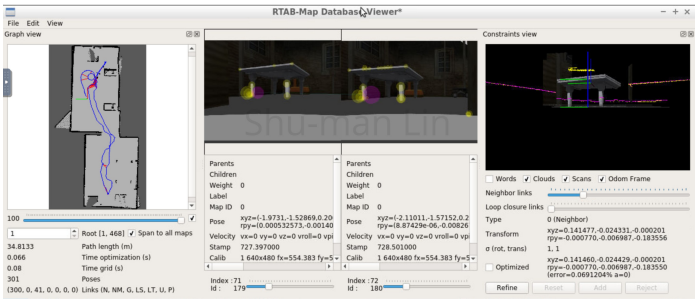

Fig. 24. Collapsed World RTAB-Map Database Viewer without Grid/3D and Grid/FromDepth

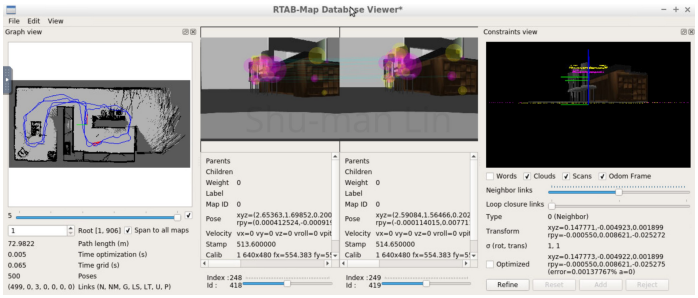

Fig. 25. Kitchen Dining World RTAB-Map Database Viewer with Grid/3D and Grid/FromDepth

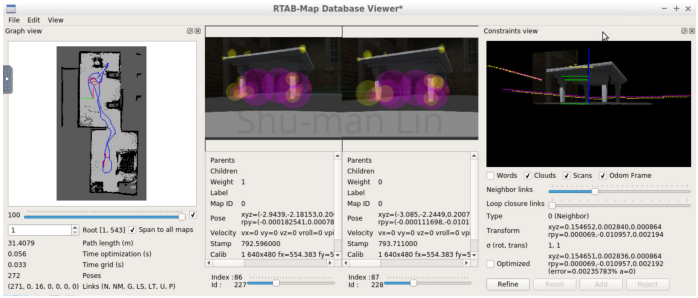

Fig. 26. Collapsed World RTAB-Map Database Viewer with Grid/3D and Grid/FromDepth

V. Discussion

Two dimensional occupancy grid mapping did not achieve much success for the kitchen world, showing a disorganized maze of disconnected possible planes of misaligned correspondences in which robot position was confusedly placed in RTAB-Map database viewer. In RViz, half of the robot's trajectory was recovered but no distinct mapping of the environment was produced. However, an upgrade to three dimensional occupancy grid mapping using Grid/3D and Grid/FromDepth with projection of such a mapping to two dimensions turned out a grid_map mapping in RViz that captured core features of the environment; included is a subscription to the topic /rtabmap/mapGraph overlaying the map to show the robot's path. On the other hand, with or without the activation of Grid/3D and Grid/FromDepth, the 2-D occupancy grid

mappings of collapsed world that were constructed were well developed and outlined the placement of obstacles within the structural floor plan of the house.

Three dimensional point cloud renderings displayed the architectural integrity of both Gazebo environments in RViz with or without Grid/3D and Grid/FromDepth pretty well during online mapping with a recognizable final result compared to the ground truth. The robot could definitely navigate without getting lost. However, cloud maps with Grid/3D and Grid/FromDepth set to true had somewhat more refined detail capture. Using Grid/3D and Grid/FromDepth definitely achieves better performance. From the point of view of RGB-D images, the development of kitchen dining world and collapsed world purely from camera images was an accurate match in the exercise of spatial mapping. In addition, the robot had to run twice around the kitchen dining world to obtain loop closures while the robot ran around once in collapsed world but with an occasional stop at the center of rooms to do a 360 degrees panning motion to achieve loop closures.

To improve accuracy, perhaps a solely RGB-D camera should be used in place of a laser, in particular as a substitute solution for creating a 2-D occupancy grid map. Even though the limitations for using RGB-D sensors come from their inability to image visually reflective and translucent material, difficulty imaging in high intensity sunlight or dim light, range constraints, and computational constraints, in [13], RGB-D sensors are given preferential treatment especially in indoor environments since they have a more robust method of obtaining depth information than stereo vision sensors, are able to recover 3-D world structure and 2-D images simultaneously as opposed to more traditional optical cameras, and recover more accurate 3-D information unlike some 3-D laser range scanners. Alternatively, a sensor fusion of RGB-D camera and 3-D LiDar could give an optimal result, integrated to make up for each others shortcomings but this mostly depends on the type of application. For example, self-driving cars require the fusion of several radar and LiDar sensors for distance sensing, RGB-D camera for traffic light detection, and GPS, inertial measurement unit, and wheel encoder for position tracking [14]. Laser based SLAM may serve as a back-up when GPS fails and maps of dynamic environments are needed for fast moving, close-by obstacle detection [15] [16]. For our purposes, in exploring an indoor static state environment with decent lighting, an RGB-D sensor is optimal, but our results show that an RGB-D sensor coupled with laser sensor readings processed with RTAB-Map is enough to solve online or full SLAM in simulation.

VI. Future Work

Real-world applications of SLAM answer this question posed in exploratory robotics: how can a vehicle know its environment and find its place within it? In varied, unexplored static home environments, cleaning robots can implement SLAM to record navigable terrain and obstacles in its path for more efficient and effective path planning by understanding the topology of the environment with loop closures [17]. In outdoor urban dynamic environments, maps of SLAM can be used for detecting moving objects and act as constraints to improve tracking performance [15]. Mars rovers solve the SLAM problem by taking stereo images from which 3-D terrain maps are generated in order to drive themselves [18]. For life or death, search and rescue challenges, rescue robots of aerial vehicle and rough terrain vehicle types use SLAM to map disaster areas [19].

For future work, open problems as cited in [20] can come from failure of algorithmic robustness due to dynamic or harsh environments or hardware related problems due to sensor or actuator degradation. Open problems include the creation of semantically more meaningful maps for SLAM systems with the aid of deep learning, deep neural network replacements for visual odometry and depth estimation, reducing the complexity and memory of map representations for long-term SLAM operation under resource constrained platforms, robust information fusion of distributed maps amongst a multi-robot SLAM system, robust recovery modes for outlier situations, providing optimal, high-level representations of objects geometry to facilitate data association, place recognition, and semantic understanding, and finding the perfect sensor-algorithm pairing for application in any given environment. Theoretical questions such as under what properties of sensors and factor graph structure is SLAM solvable globally and how to design global techniques and their verifications to be resilient to outliers are also open. The challenges the SLAM community faces in the future are manifold where models have yet to match the effectiveness of a human's.

References

[1] S. Thrun, W. Burgard, D. Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press. 2005.

[2] S. Thrun, M. Montemerlo. *The GraphSLAM Algorithm with Applications to Large-Scale Mapping of Urban Structures*. The International Journal of Robotics Research. Vol. 25, No. 5-6. online: http://robot.cc/papers/thrun.graphslam.pdf, June 2006.

[3] G. Grisetti, R. Kummerle, C. Stachniss, W. Burgard. *A Tutorial on Graph-Based SLAM*. IEEE Intelligent Transportation Systems Magazine. Vol. 2. Issue: 4. online: http://www2.informatik.uni-freiburg.de/~stachnis/pdf/grisetti10titsmag.pdf, Winter 2015.

[4] M. Labbe, F. Michaud. *Online Global Loop Closure Detection for Large-Scale Multi-Session Graph-Based SLAM*. In Proc. IEEE/RSJ Intelligent Robots and Systems (IROS). online: https://introlab.3it.usherbrooke.ca/mediawiki-introlab/images/e/eb/Labbe14-IROS.pdf, 2014.

[5] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit. *FastSLAM: A factorized solution to the simultaneous localization and mapping problem*. Proceeding eighteenth national conference on Artificial Intelligence. online: http://robots.stanford.edu/papers/montemerlo.fastslam-tr.pdf, 2018.

[6] S. Thrun, M. Montemerlo, D. Koller, B. Wegbreit, J. Nieto, E. Nebot. *FastSLAM: An efficient solution to the simultaneous localization and mapping problem with unknown data association*. Journal of Machine Learning Research. online: https://ai.stanford.edu/~koller/Papers/Thrun+al:04b.pdf, May 2004.

[7] G. Grisetti, C. Stachniss, W. Burgard. *Improved techniques for grid mapping with Rao-Blackwellized particle filters*. IEEE Transactions on Robotics. online: http://ais.informatik.uni-freiburg.de/publications/papers/grisetti07tro.pdf, Feb. 2007.

[8] *RTAB-Map's ROS package*. GitHub. online: https://github.com/introlab/rtabmap_ros#build-from-source. April 2018.

[9] *Wiki: catkin/package.xml*. ROS wiki. online: http://wiki.ros.org/catkin/package.xml. January 2018.

[10] *Wiki: catkin/CmakeLists.txt*. ROS wiki. online: http://wiki.ros.org/catkin/CMakeLists.txt. July 2017.

[11] *Wiki: rtabmap_ros*. ROS Wiki. online: http://wiki.ros.org/rtabmap_ros. Sept. 2017

[12] *Wiki: rtabmap_ros/Tutorials/SetupOnYourRobot*. ROS wiki. online: http://wiki.ros.org/rtabmap_ros/Tutorials/SetupOnYourRobot Sept. 2016.

[13] T. Chong, X. Tang, C. Leng, M. Yogeswaran, O. Ng, Y. Chong. *Sensor Technologies and Simultanesous Localization and Mapping (SLAM)* Procedia Computer Science Vol. 76. online: https://doi.org/10.1016/j.procs.2015.12.336 2015.

[14] E. Guizzo. *How Google's Self-Driving Car Works*. IEEE Spectrum. online: https://spectrum.ieee.org/automaton/robotics/artificial-intelligence/how-google-self-driving-car-works. Oct. 2011.

[15] C. Wang, C. Thorpe, S. Thurn. *Online Simultaneous Localization and Mapping with Detection and Tracking of Moving Objects: Theory and Results from a Ground Vehicle in Crowded Urban Areas*. Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on. online: http://robots.stanford.edu/papers/wang.icra-slam-datmo.pdf, 2003.

[16] J. Levinson, M. Montemerlo, S. Thrun. *Map-Based Precision Vehicle Localization in Urban Environments*. Robotics: Science and Systems, The MIT Press. online: http://www.roboticsproceedings.org/rss03/p16.pdf, 2007.

[17] W. Knight. *The Roomba now sees and maps a home*. MIT Technology Review. online: https://www.technologyreview.com/s/541326/the-roomba-now-sees-and-maps-a-home/, Sept. 2015.

[18] *In-situ Exploration and Sample Return: Autonomous Planetary Mobility*. NASA JPL, California Institute of Technology. online: https://mars.nasa.gov/mer/technology/is_autonomous_mobility.html.

[19] Y. Liu, G. Nejat. *Robotic Urban Search and Rescue: A Survey from the Control Perspective*. Journal of Intelligent & Robotic Systems. Vol. 72, Issue 2. March 2013.

[20] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, J. Leonard. *Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age*. IEEE Transactions on Robotics Vol. 32 No. 6. online: http://rpg.ifi.uzh.ch/docs/TRO16_cadena.pdf, Dec. 2016.