Project: Map My World Robot - Thomas Hatting

Abstract

Simultaneous Localization and Mapping (SLAM) is a field of robotics dealing with situations where the robot does not have access to a map of the environment nor have knowledge of its own pose. Well-known approaches to SLAM are EKF, FastSLAM and GraphSLAM. In this project, a variant of graph-based SLAM is used called RTAB-Map. A robot has been configured in ROS, based on a previous project. Two simulated environments have been created: a kitchen-dining environment based on existing template and an own designed environment. The robot is navigated around the two environments to generate 2D and 3D maps, and the results are compared.

1. Introduction

Simultaneous Localization and Mapping (SLAM) is a field of robotics dealing with problems that occur when the robot does not have access to a map of the environment nor have knowledge of its own pose. This is a fundamental problem in robotics and appears to be the classical “chicken-and-egg” problem, however there are number of algorithms to solve this scenario, in particular, Extended Kalman Filters (EKF), FastSLAM and GraphSLAM. These algorithms have been employed in self-driving cars, unmanned aerial vehicles, new emerging domestic robotics and even inside the human body (Thrun et al. 2006, Wikipedia 2018).

In this project, a graph-based variant of SLAM is used called RTAB-Map. This algorithm appears to be the best solution in SLAM to develop robots that can map environments in 3D (Udacity 2018). The algorithm is fast and provides good memory management. Also, the tools for information analysis are well-developed, and the system is well-documented. In the project, an API in ROS called rtabmap_ros is being used to interact with the RTAB-Map package.

The goal of this project is to develop a robot that can interface with RTAB-Map in a simulated environment. The robot is launched in the simulated environment without

knowledge of the map or poses. The robot is then navigated around the area to generate 2D and 3D maps.

2. Background / Formulation

A somewhat less challenging form of mapping in robotics occurs when you assume that the robot poses are known. A popular family of algorithms to deal with this problem is called occupancy grid mapping. These algorithms address the problem of generating consistent maps despite noise in motion and perception data, under the assumption that the robot's poses are known.

A more challenging form of mapping is called SLAM (Simultaneous Localization and Mapping). SLAM is more challenging than occupancy grid mapping, since the poses are unknown and have to be estimated along the way. Furthermore, SLAM is more difficult than localization, such as using the Monte Carlo algorithm, since the map is unknown and has to be estimated along the way (Thrun et al. 2006).

There are various approaches to SLAM. The earliest algorithms were based on the Extended Kalman Filter (EKF). Maps in this algorithm are feature-based, meaning they are composed of landmarks. The EKF algorithm uses a linear Gaussian-based approach to the noise generated by the robot's motion and perception data. However, in real-life situations, this is not always true, and under certain circumstances this assumption can lead to an excess of errors and inconsistencies. Another implementation of SLAM is called FastSLAM. This is a particle filter approach where each particle contains an estimate of the robot's pose. The advantage of FastSLAM is that particle filters can cope with non-linear models of motion in contrast to the EKF algorithm. Also, FastSLAM offers computational advantages over EKF (Thrun et al. 2006).

There are two main forms of SLAM. One is known as the online SLAM problem. This involves estimating the posterior over the momentary pose along with the map. The second SLAM problem is called the full SLAM problem. In full SLAM, the posterior is calculated over the entire path along with the map instead of just the robot's current pose.

In contrast to EKF, the so-called GraphSLAM algorithm solves the full SLAM problem (Thrun et. al. 2006). In this algorithm a graph is used to represent the problem. Every node in the graph corresponds to a pose of the robot during the mapping. Every edge between two nodes corresponds to the spatial constraints between them. The goal of this algorithm is to

find a configuration of the nodes that minimize the error introduced by the constraints (Abbeel 2013).

The current challenge in robotic mapping is to develop accurate, efficient and fast algorithms, which do not require excessive computational resources. One high-performing algorithm is RTAB-Map (Real-Time Appearance-Based Mapping). This is a variation of GraphSLAM and is based on an incremental loop closure detector. The algorithm is depicted in figure 1. The loop closure detector uses a bag-of-words to determine how likely a new image comes from a previous location or new location. When a loop closure hypothesis is accepted, a new constraint is added to the map's graph, then a graph optimizer minimizes the errors in the map. Furthermore, a memory management approach is used to limit the number of locations used for loop closure detection and graph optimization (IntRoLab 2018a).

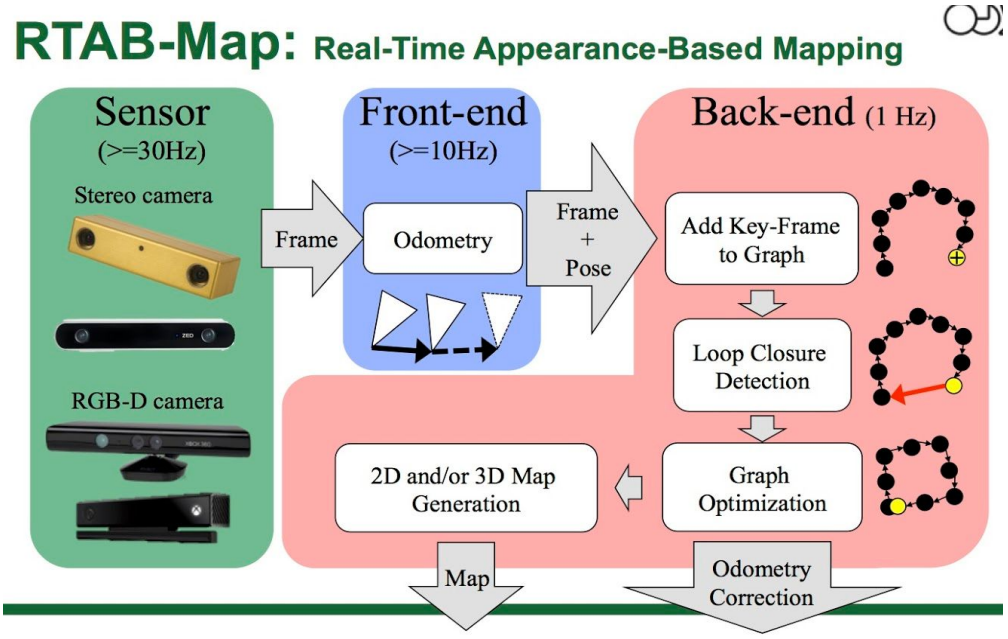

Figure 1:Depiction of how RTAB-Map works (IntRoLab 2018a)

In robotics, it is important to have the ability to generate 2D maps, such as grid occupancy maps. However, in many robotic applications, it would not be enough to generate 2D models. Several applications also require 3D models of the environment. 3D models are needed in many airborne, underwater, outdoor or planetary missions. Also, the use of 3D models is relevant in domestic tasks such as mobile manipulation. Although, 3D mapping is an important component of robotic systems, in the past, there have been a limited number of readily available, reliable and efficient implementations. To deal with these challenges, open-source 3D mapping frameworks have been developed (Hornung et al. 2012; Octomap

2018). One such open-source framework is the Octomap library that implements a 3D occupancy grid mapping approach, providing data structures and mapping algorithms in C++ particularly suited for applications in robotics. This approach is able to model arbitrary environments without prior assumptions about the environment. Moreover, it is possible to add new information or sensor readings at any time to the map, and the extent of the map does not have to be known in advance. Instead, the map is dynamically expanded as needed. This type of map can be stored efficiently, both in memory and on disk (Octomap 2018).

3. Scene and Robot Configuration

A previously developed robot (from project: “Where Am I”) has been adapted to this project. In table 1, the configuration of the robot has been shown:

Link or Joint	**Origin**	**Comment**
Chassis (link)	xyz = (0, 0, 0) rpy = (0, 0, 0)	cylinder length 0.1, radius 0.1, mass 15.0
Back caster (link)	xyz = (-0.15, 0, -0.05) rpy = (0, 0, 0)	sphere radius 0.05
Front caster (link)	xyz = (0.15, 0, -0.05) rpy = (0, 0, 0)	sphere radius 0.05
Left wheel (link)	xyz = (0, 0, 0) rpy = (0, 1.5707, 1.5707)	cylinder radius 0.1, length 0.05, mass 5.0
Right wheel (link)	xyz = (0, 0, 0) rpy = (0, 1.5707, 1.5707)	cylinder radius 0.1, length 0.05, mass 5.0
Left wheel hinge (joint)	xyz = (0, 0.15, 0) rpy = (0, 0, 0)	parent link: chassis; child link: left wheel
Right wheel hinge (joint)	xyz = (0, -0.15, 0) rpy = (0, 0, 0)	parent link: chassis; child link: right wheel
Camera (link)	xyz = (0, 0, 0) rpy = (0, 0, 0)	box size 0.05x0.05x0.05, mass 0.1
Camera (joint)	xyz = (0.2, 0, 0) rpy = (0, 0, 0)	parent link: chassis; child link: camera_link

Camera RGB frame	xyz = (0, 0, 0) rpy = (0, 0, 0)	link added to match the plugin for Kinect camera
Camera RGB joint	xyz = (0.001, 0, 0.001) rpy = (0, 0, 0)	parent link: camera_link; child link: camera_rgbd_frame
Camera optical link	xyz = (0, 0, 0) rpy = (0, 0, 0)	link added to account for depth cloud being upside down in RViz
Camera optical joint	xyz = (0, 0, 0) rpy = (-1.57, 0, -1.57)	parent link: camera_rgbd_frame; child link: camera_optical_link
Hokuyo (link)	xyz = (0, 0, 0) rpy = (0, 0, 0)	box size 0.1x0.1x0.1, mass 0.1
Hokuyo (joint)	xyz = (0.15, 0, 0.1) rpy = (0, 0, 0)	parent link: chassis; child link: hokuyo

Table 1: Configuration of the robot in URDF

The “camera RGB joint” (frame-name: camera_rgbd_frame) was added when following warning was observed:

[WARN] [1520423753.974848992, 1085.039000000]: Could not get transform from base_footprint to camera_rgbd_frame after 0.200000 seconds (for stamp=1084.477000)! Error="canTransform: source_frame camera_rgbd_frame does not exist.. canTransform returned after 0.203 timeout was 0.2.".
[ERROR] [1520423753.974951294, 1085.039000000]: TF of received image 0 at time 1084.477000s is not set!
[ERROR] [1520423753.975003677, 1085.039000000]: Could not convert rgb/depth msgs! Aborting rtabmap update…

This warning indicated that the transforms could not be completed because a frame was missing in the URDF configuration yet had been defined in the Gazebo plugin. Also, a “camera optical joint” was added (frame-name: camera_optical_link). This was to correct for the fact that the depth cloud was pointing in the wrong direction in RViz.

The robot configuration has been stored under filenames, *slam_project.gazebo* and *slam_project.xacro*, and has been located in directory: ***~catkin_ws/src/slam_project/urdf***.

The robot located inside the kitchen-dining environment has been shown in figure 2.

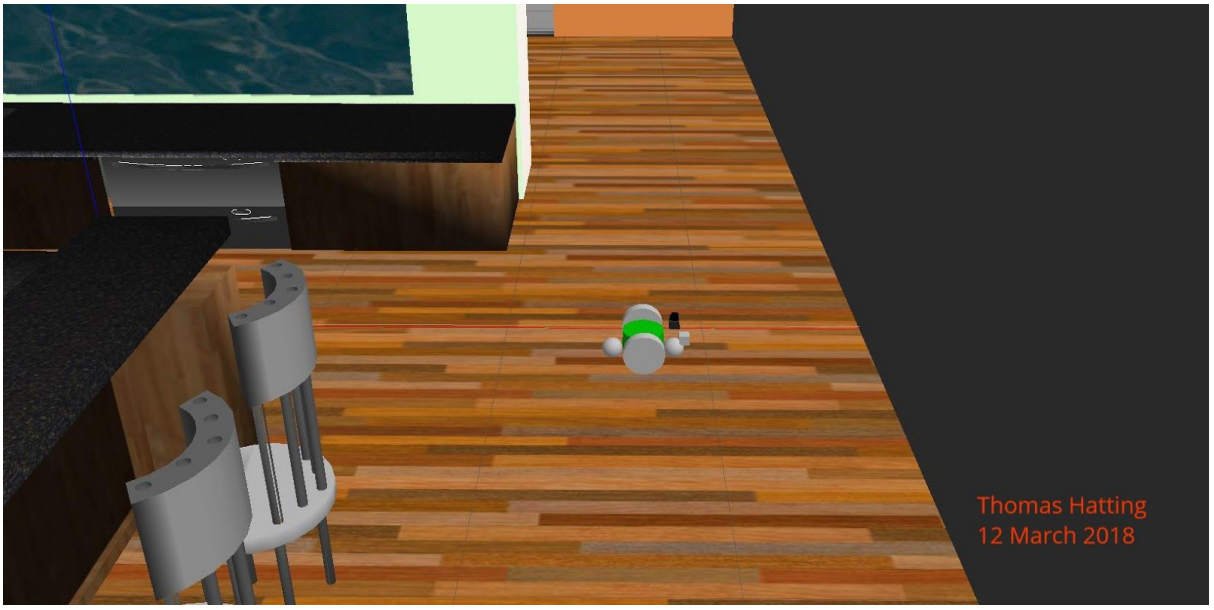

Figure 2: The robot in the kitchen-dining environment

In the project, a default file was provided for the kitchen-dining environment. During the simulation, it was discovered that only manually inserted objects gave off scan data. The walls and built-in objects did not provide any scan data with following fault occurring on a regular basis:

[INFO] [1520612167.224209170, 367.894000000]: rtabmap (153): Rate=1.00s, Limit=0.000s, RTAB-Map=0.0746s, Maps update=0.0008s pub=0.0005s (local map=1, WM=1)
[WARN] (2018-03-09 16:16:08.379) SensorData.cpp:733::uncompressDataConst() Requested laser scan data, but the sensor data (-1) doesn't have laser scan.

This was due to the fact that collisions were turned off in the default kitchen-dining world. This was solved by opening a blank gazebo page and inserting the kitchen-dining object. This world was identical to the default kitchen-dining environment but had collisions turned on, and consequently laser scan data could be received from the walls.

The own simulated environment has been shown in figure 3, including the robot shown to the left being ready to start the mapping process. The environment was created in Gazebo. It consisted of brick walls, a door, openings for windows and a number of objects. The objects were picked amongst the built-in objects in Gazebo. These were objects like shelves, tables, postboxes, TurtleBots, car tyres, cans and cardboard boxes. The objects were placed along walls and corridors to generate a feature rich environment, thus increasing the likelihood of loop-closures.

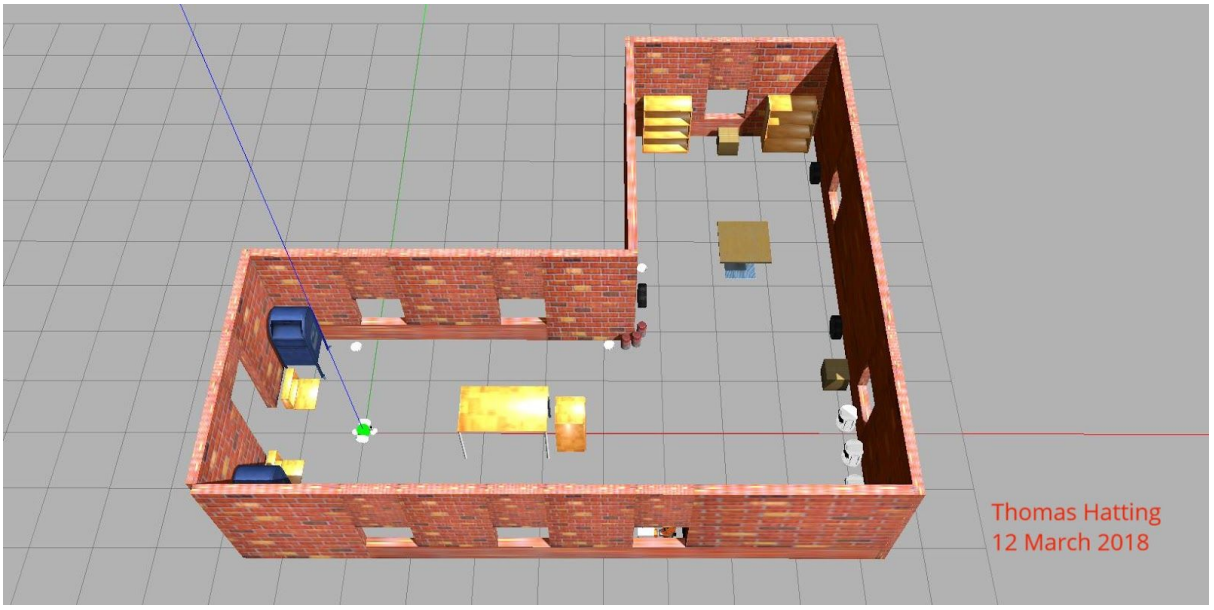

Figure 3: Own simulated environment created in Gazebo

The two environments have been stored under filenames, *kitchen_dining.world* and *thomas_hatting.world*, and have been located in ***~catkin_ws/src/slam_project/worlds***.

The created launch files have been stored as, *world.launch, teleop.launch, mapping.launch, rviz.launch, localization.launch and robot_description.launch*, and have been located in directory: ***~catkin_ws/src/slam_project/launch***.

4. Results

The tf-tree of the system has been shown in figure 4. As can be seen, there is a transform from camera_link to camera_rgbd_frame. The name, camera_rgbd_frame, corresponds to the name used in the Gazebo plugin for the Kinect camera.

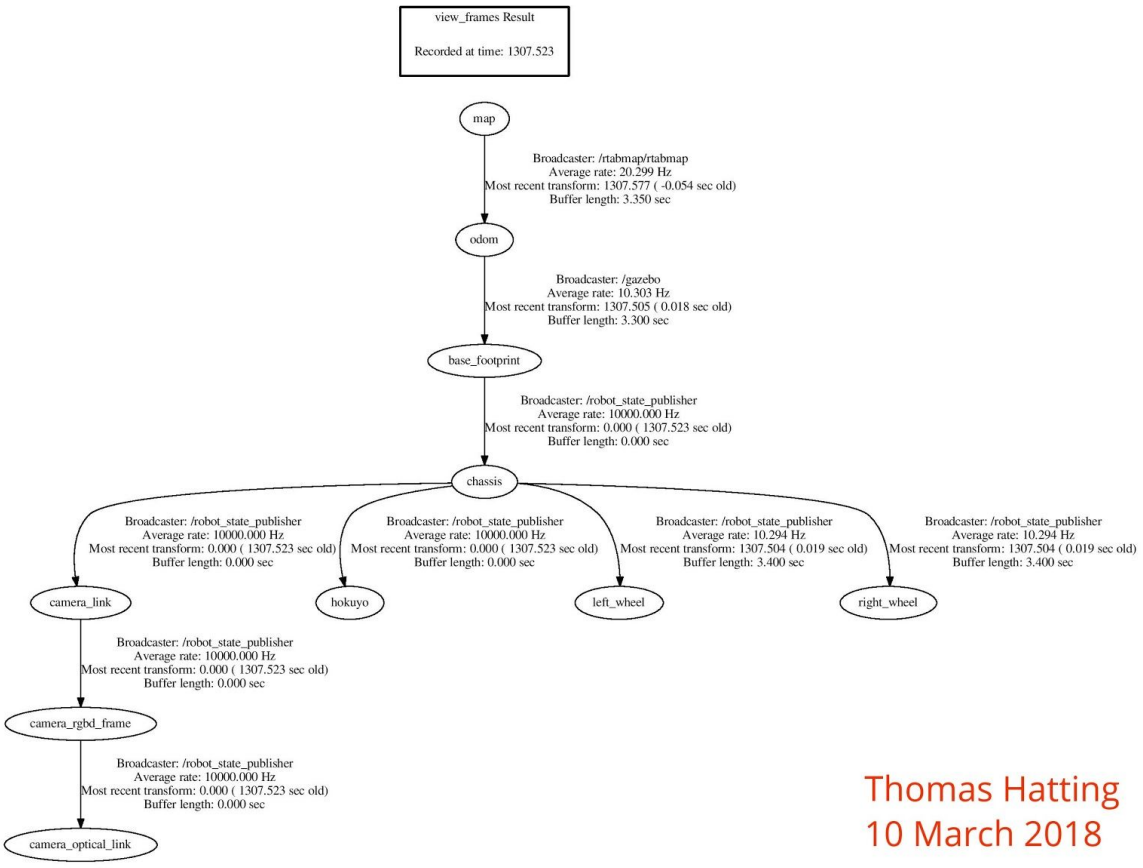

Figure 4: Tf-tree for the system

Connections for topics and nodes have been shown in figure 5. As can be seen, the node /gazebo publishes to the 3 topics, **/camera/depth/image/raw**, **/camera/rgb/camera_info**, **/camera/rgb/image_raw**, for the camera and topic **/scan** for the laser scanner. The node **/rtabmap/rtabmap** created by the RTAB-Map package then subscribes to these four topics. Also, the robot publishes its states to topic /joint_states, which is received by node **/rtabmap/rtabmap** via topic /tf.

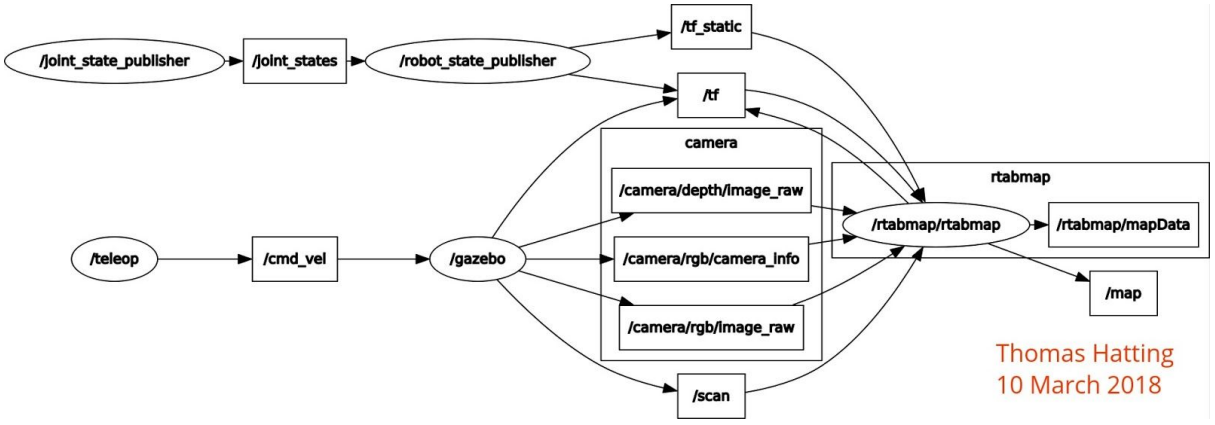

Figure 5: Visualization of connections with rqt_graph

The initial view of the depth cloud has been shown in figure 6. It was necessary to rotate the depth cloud in order to correctly visualize it in RViz. This was done by means of the frame camera_optical_link (see tf-tree in figure 4), which caused a rotation of -90 degrees about the x-axis and -90 degrees about the z-axis (Gazebo 2018)

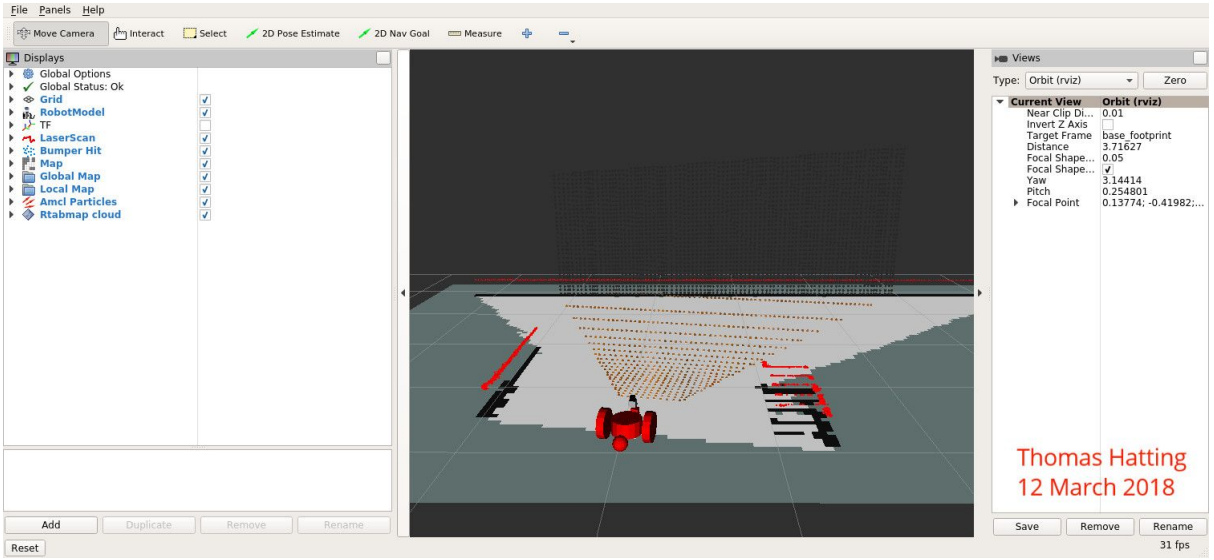

Figure 6: Initial view of depth cloud in RViz

To begin with, simulations were carried out with the robot making 3 loops around the central object in the first room and then making 3 loops around the central object in the second room. The results have been shown as a 2D occupancy grid map in figure 7a and, furthermore, as a 3D octomap map in figure 7b. As can be seen there was quite a lot of distortion, and the maps did not come out well. Due to the distortion, the algorithm somehow misinterpreted the information and merged the two rooms into one. The corresponding screenshot from rtabmap-databaseViewer has been shown in figure 7c. The number of loop closures can be seen at bottom left. As can be seen, there were 231 global loop closures.

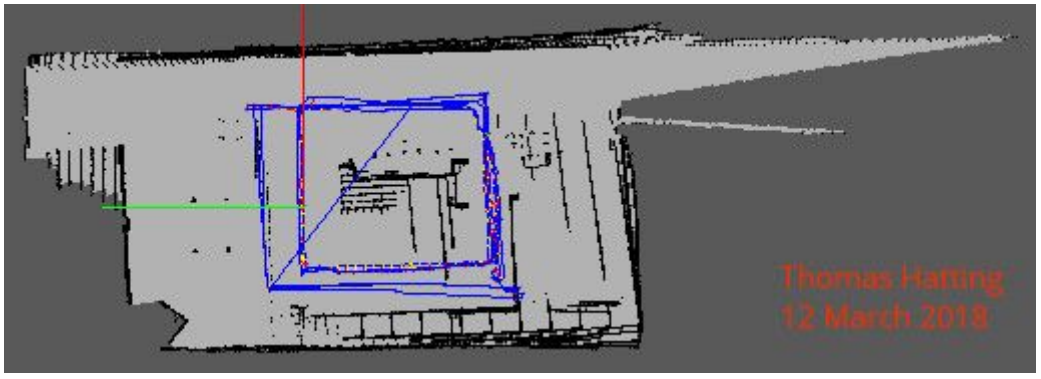

Figure 7a: 2D map of the kitchen-dining environment with the robot navigating in loops around central objects in each room

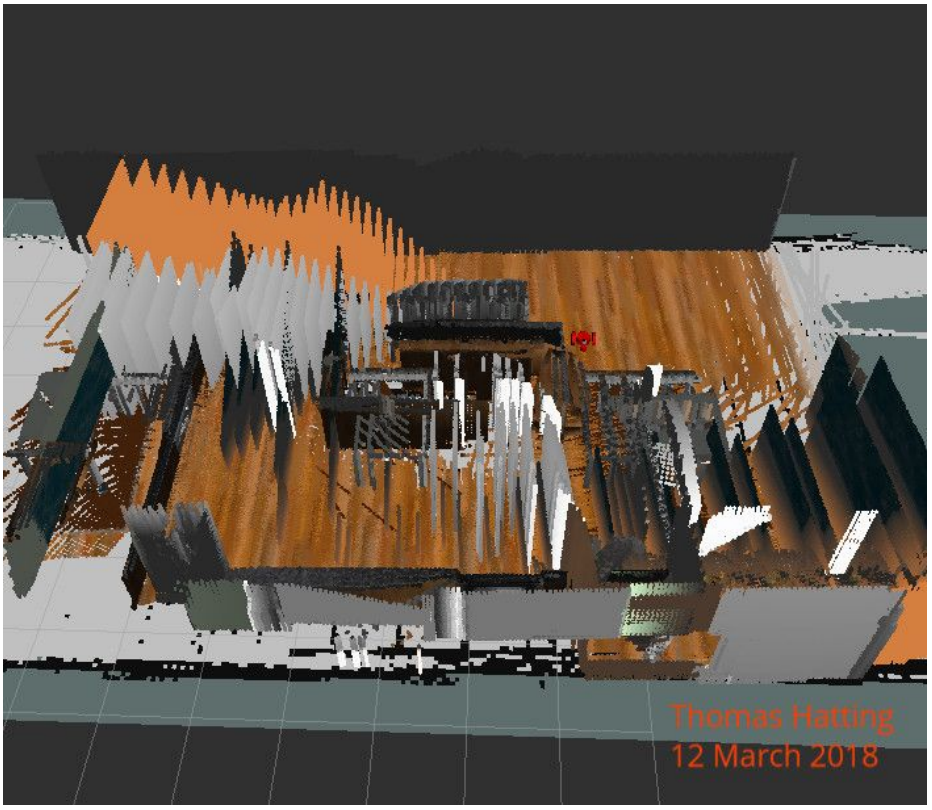

Figure 7b: 3D map in RViz of the kitchen-dining environment with the robot navigating in loops around central objects in each room

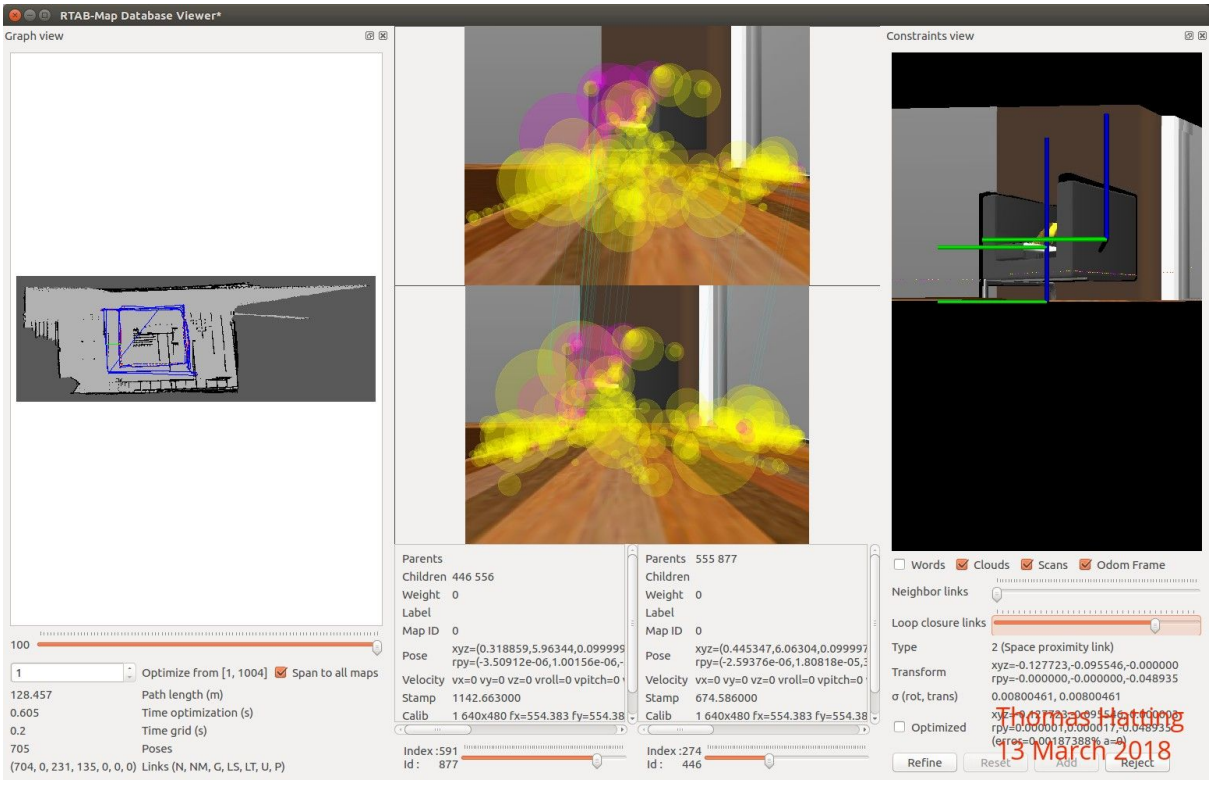

Figure 7c: Screenshot from rtabmap-databaseViewer for kitchen-dining environment with the robot navigating in loops around central objects in each room

Instead of navigating the robot around in loops, the robot was navigated up and down three times in straight lines along the central corridor connecting the two rooms. This has been shown in figures 8a, 8b and 8c. In this case the maps turned out better with less distortion. There is a “shadow” behind the kitchen table but apart from that the maps appear to be accurate. These were the most accurate maps obtained during the simulation and are therefore the final maps of the kitchen-dining environment. Once the robot navigated around the tables and back, as shown in the previous simulation, the maps started to get distorted and inaccurate. As can be seen in figure 8c, there were 79 global loop closures, which was less than the previous simulation, however, the maps turned out better.

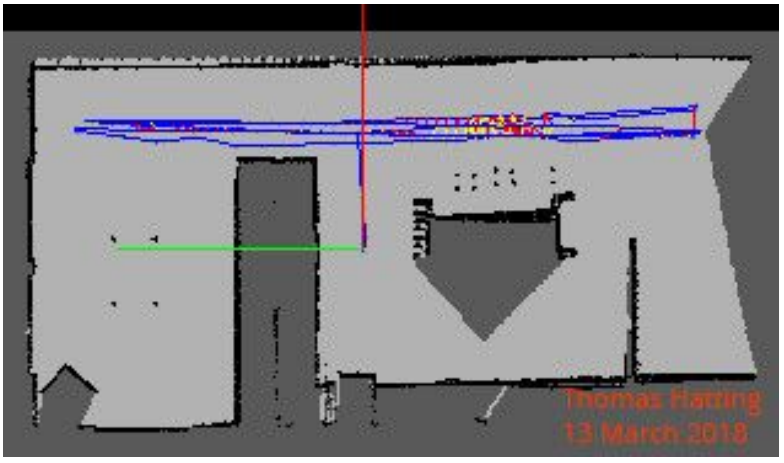

Figure 8a: Final 2D map of the kitchen-dining environment with the robot navigating in linear paths along the corridor

Figure 8b: Final 3D map in RViz of the kitchen-dining environment with the robot navigating in linear paths along the corridor

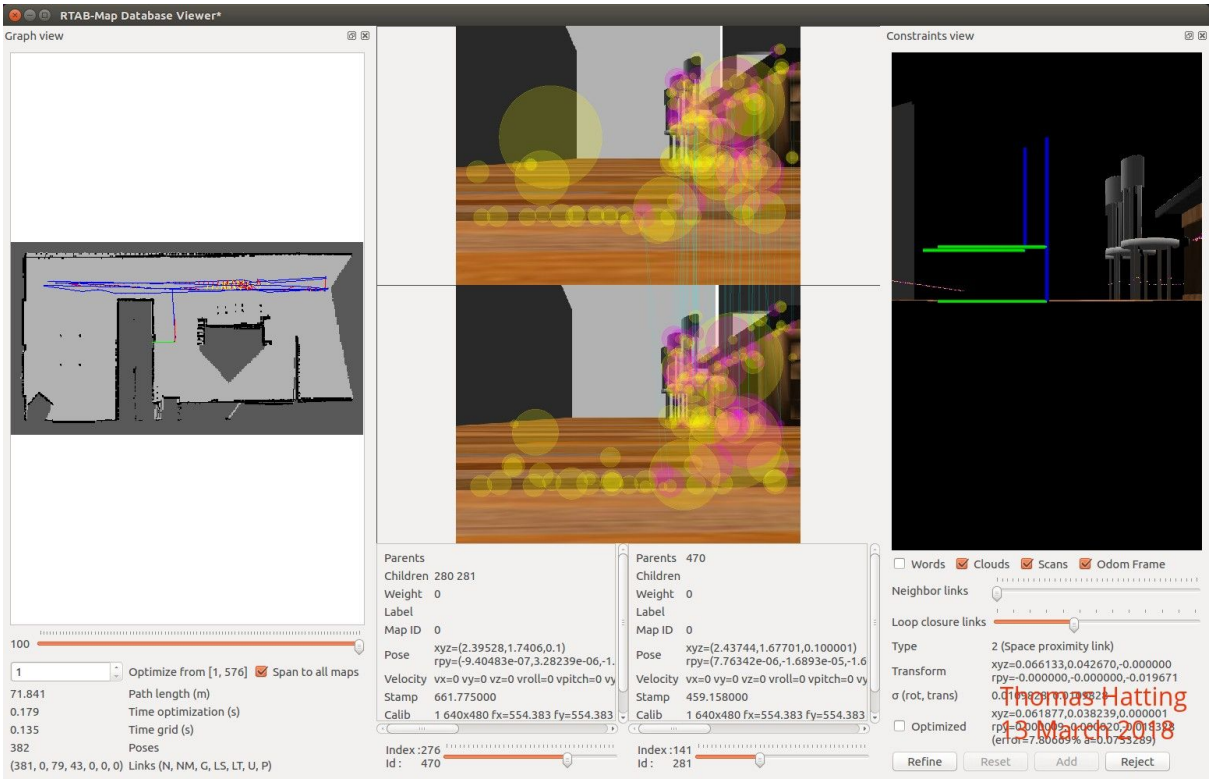

Figure 8c: Screenshot from rtabmap-databaseViewer for the kitchen-dining environment with the robot navigating in linear paths along the corridor

Next, the simulation was carried out for the own generated environment. The robot was navigated around this area three times. The results have been shown as a 2D occupancy grid map in figure 9a and as a 3D octomap in figure 9b. The result came out much better this time, perhaps, due to the fact that the environment was less complex than the kitchen-dining environment. As can be seen, the RTAB-Map algorithm clearly mapped out the environment. There were in all 178 global loop closures, as shown in figure 9c.

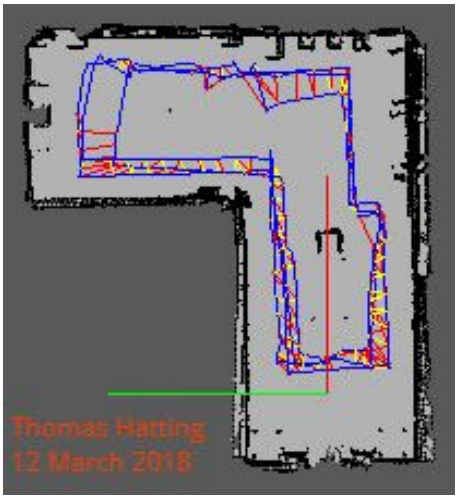

Figure 9a: Final 2D map of own environment

Figure 9b: Final 3D map in RViz for own simulated environment

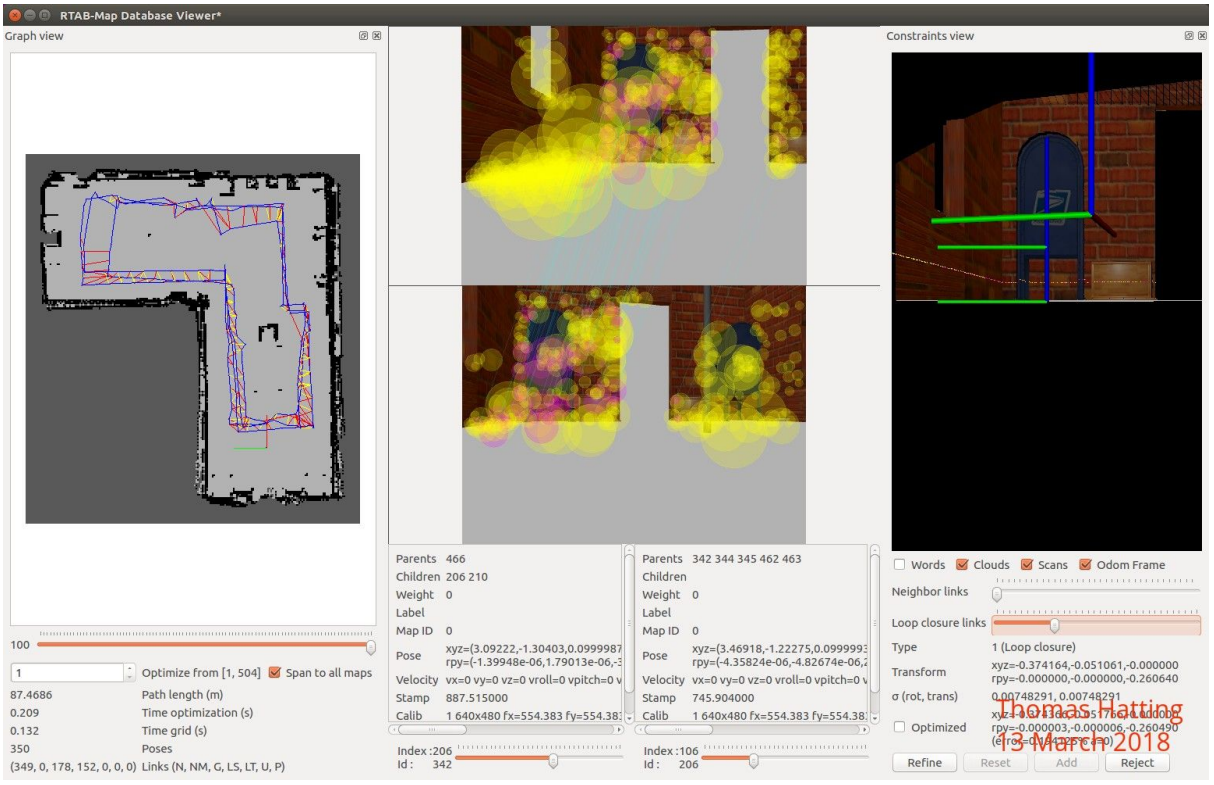

Figure 9c: Screenshot from rtabmap-databaseViewer for own simulated environment

An attempt was made to run the ROS package, developed in this project, on a NVIDIA Jetson TX2 platform. The Jetson TX2 was flashed with Jetpack 3.1 followed by installation of ROS and RTAB-Map. It was possible to successfully launch the files “world.launch” and “teleop.launch” to bring up environments and navigate the robot. The environments opened up fine in Gazebo, and the robot could be navigated around with the keyboard. However, it was not possible to launch the third file: mapping.launch. The mapping process failed with exit code -11:

```
nvidia@tegra-ubuntu:~/catkin_ws$ roslaunch slam_project mapping.launch
... logging to /home/nvidia/.ros/log/7295db48-26f0-11e8-a4c9-00044ba7dade/roslaunch-tegra-ubuntu-10686.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.
started roslaunch server http://tegra-ubuntu:42007/
..
..
[ INFO] [1520967603.712989015, 332.431000000]: rtabmap 0.15.0 started...
[rtabmap/rtabmap-1] process has died [pid 10704, exit code -11, cmd
/home/nvidia/catkin_ws/devel/lib/rtabmap_ros/rtabmap --delete_db_on_start scan:=/scan
rgb/image:=/camera/rgb/image_raw depth/image:=/camera/depth/image_raw
rgb/camera_info:=/camera/rgb/camera_info grid_map:=/map __name:=rtabmap
__log:=/home/nvidia/.ros/log/7295db48-26f0-11e8-a4c9-00044ba7dade/rtabmap-rtabmap-1.log].
log file: /home/nvidia/.ros/log/7295db48-26f0-11e8-a4c9-00044ba7dade/rtabmap-rtabmap-1*.log
all processes on machine have died, roslaunch will exit
shutting down processing monitor...
... shutting down processing monitor complete
done
```

It was not possible within the timeframe of the project to resolve this issue. In the end, the simulations were done in the student workspace only.

5. Discussion

When navigating the kitchen-dining environment, distortion appeared in the images. Especially, navigating the robot in loops seemed to lead to this problem. Although, there were plenty of global loop closures when the robot moved like this, these loop closures did not seem to be enough to generate accurate and coherent 2D and 3D maps. During the simulations these warnings appeared:

```
[ WARN] (2018-03-13 10:05:18.134) Rtabmap.cpp:1860::process() Rejected loop closure 281 -> 1237: Not
enough inliers 11/15 (matches=21) between 281 and 1237
```

[INFO] [1520935518.212715998, 1519.147000000]: rtabmap (1237): Rate=1.00s, Limit=0.000s, RTAB-Map=0.2890s, Maps update=0.0326s pub=0.0047s (local map=929, WM=929)

[WARN] (2018-03-13 10:05:16.031) Rtabmap.cpp:2299::process() Rejecting all added loop closures (1) in this iteration because a wrong loop closure has been detected after graph optimization, resulting in a maximum graph error of 1.190409 m (edge 1234->1235, type=0). The maximum error parameter is 1.000000 m.
[WARN] (2018-03-13 10:05:16.031) Rtabmap.cpp:2303::process() Loop closure 1235->432 rejected!

The first warning appeared regularly at the beginning but after a while appeared less often. The warning is likely to be okay and just means that the robot had not collected enough data at that particular point in time to create a loop closure. The second warning might also be okay, as it states the algorithm detected a wrong loop closure after graph optimization and therefore rejected all added loop closures since the point of detection. In other words, it is making a correction for a mistake made in the past.

When navigating the entire length of the kitchen-dining area in straight lines, the 2D and 3D maps were significantly more accurate. However, the best result was achieved when the robot navigated in the own generated environment. This environment was less complex than the kitchen-dining area, and the RTAB-Map algorithm produced the most accurate 2D and 3D maps.

One possible way to reduce distortion could be to investigate whether depth calibration of the RGB-D camera might help (IntRoLab 2018b). The gazebo plugin for the RGB-D camera (Kinect camera) has a number of parameters that could be tuned (this plugin was provided as part of the project).

6. Conclusion and Future Work

In this project, a graph-based variant of SLAM was used called RTAB-Map. Two simulated environments were created, a kitchen-dining environment based on an existing template and an own designed environment. The robot was navigated around the two environments to generate 2D and 3D maps, which were then compared.

For the kitchen-dining environment, there was some distortion in the maps, especially when navigating the robot in loops around the room. A better result was achieved when navigating the robot in straight lines along the entire length of the kitchen-dining area. The robot performed best in the own generated environment, possibly due to the environment being less complex.

In terms of future work, the next step is to get the RTAB-Map algorithm to work on a hardware platform and ultimately build a full robotic system. During the project, an attempt was made to get RTAB-Map to work on the NVIDIA Jetson TX2, but there were issues during the launch of the algorithm, which could not be resolved within the timeframe of the project. These issues need to be investigated, and it might be necessary to completely rebuild the TX2 (flash again with JetPack 3.1 and reinstall ROS & RTAB-Map).

The second step is to acquire a suitable robotics platform. One such platform is the TurtleBot, which is well-supported and documented in ROS and Gazebo (ROS 2018). Another suitable platform is iRobot Create 2. Although this platform does not appear to be as well supported in ROS as the TurtleBot, it offers other advantages such as compact size and a self-charging home-base (iRobot 2018). The Jetson TX2 needs to be mounted somehow on either of these platforms.

In addition, a suitable RGB-D camera should be acquired. During the project's simulations, a Gazebo plugin for the Microsoft Kinect was used, however the actual hardware has gone out of production (Microsoft 2018), but perhaps a second-hand Kinect camera could be purchased. An alternative solution would be to acquire an Intel RealSense depth camera (Intel 2018). A laser scanner could also be acquired, but it is not necessary to have this piece of equipment to implement RTAB-Map (Udacity 2018). So, the laser scanner could be left out to lower costs.

The robot built in hardware could then be used as a prototype for a number of applications. One application would be to use the robot as a public library assistant. The robot would drive around the library to generate 2D and 3D maps of the place. The robot could then be used as a customer-service agent to help guide library users to the right parts of the library in order to find a particular book. Another application could be as a museum guide that supplies information and conducts tours for visitors in a museum or gallery. The robot would need good mapping capabilities, such as RTAB-Map, to carry out such a job.

References

Abbeel, P. (2013) “GraphSLAM”. UC Berkeley EECS. Available online: https://people.eecs.berkeley.edu/~pabbeel/cs287-fa13/slides/GraphSLAM.pdf

Gazebo (2018) “Gazebo 1.5: libgazebo_openni_kinect.so: pointcloud data is rotated and flipped?”. Gazebo Answers. Available online: http://answers.gazebosim.org/question/4266/gazebo-15-libgazebo_openni_kinectso-pointcloud-data-is-rotated-and-flipped/

IntRoLab (2018a) “RTAB-Map: Real-Time Appearance-Based Mapping”. Université de Sherbrooke. Available online: http://introlab.github.io/rtabmap/

IntRoLab (2018b) “Depth Calibration”. Available online: https://github.com/introlab/rtabmap/wiki/Depth-Calibration

iRobot (2018) “iRobot Create® 2 Programmable Robot”. Available online: http://www.irobot.com/About-iRobot/STEM/Create-2.aspx

Intel (2018) “The Newest in Depth Cameras”. Intel Software Developer Zone. Available online: https://software.intel.com/realsense

Hornung, A., Wurm, K.M., Bennewitz, M., Stachiss, C. and Burgard, W. (2012) “OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees”. Preprint submitted to publication Autonomous Robots (2013). Available online: http://www2.informatik.uni-freiburg.de/~hornunga/pub/hornung13auro.pdf

Microsoft (2018) “Kinect for Windows”. Available online: https://developer.microsoft.com/en-us/windows/kinect

Octomap (2018) “OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees”. Available online: https://octomap.github.io/

ROS (2018) “Robots/ TurtleBot”. TurtleBot documentation in ros.org. Available online: http://wiki.ros.org/Robots/TurtleBot

Thrun, S., Burgard, W. and Fox, D. (2006) “Probabilistic Robotics”. Massachusetts Institute of Technology. MIT Press.

Udacity (2018) , Classroom Material, Term 2, Sections 15-18 + P, Robotics Software Engineer Nanodegree Program, March 2018.

Wikipedia (2018) “Simultaneous localization and mapping ”. Article available online: https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping