Map My World Project

Ross Lloyd

Abstract—A popular SLAM algorithm, RTABMap, which uses an appearance based loop closure approach to create 3D and 2D environment maps, is applied and assessed in three environments, two simulated and one real. A six-wheeled differential drive robot is used to map the virtual environments, whilst a two wheeled differential drive robot was built to explore the real environments. RTABMap proved to be a robust SLAM tool in all environments, though results were slightly poorer in the real space. Reasons for this and possible improvements are discussed.

Index Terms—SLAM, RTABMap, Udacity, ROS, Gazebo

1 Introduction

In this project, a novel form of SLAM, known as Real Time Appearance Based Mapping (RTABMap) will be used to create maps of both supplied and original gazebo world models. This will be accomplished by creating appropriate launch files in ROS, creating a URDF file and STL models for an RGBD camera equipped robot to provide 3d mapping, using the ROS teleoperation node to move the virtual robot in the world. An *rtabmap.db* database containing the extracted occupancy grid map, octomap and data to create a point cloud representation is the intended output. Additionally a real-world robot was built in order to create a database for a home environment.

2 Background

2.1 General Principles of SLAM

As described above, the essence of the SLAM problem is to carry out mapping of an unknown environment, which requires a known path (set of poses) in order to estimate the environment. Simultaneously, we also want to accomplish Localization within that map, but that requires a known map to estimate the pose, which is usually unavailable especially for changing or unexplored environments. The robot has access to its sensor data and odometry data and must somehow correlate what it measures and what its control inputs are, to some map and set of accurate poses.

Fig. 1. Probability equations for types of robotic problem .

Occupancy Grid Mapping

The ultimate output for all of these calculations is an Occupancy Grid Map, in either 2d or 3d. An OGM is a discrete representation of a continuous space. Occupancy Grid Mapping (OGM) creates a map representing occupied, empty or unknown grid cells. However, an algorithm dedicated to achieving OGM must be able to contend with a number of fundamental challenges. Firstly, it must deal with both discrete (e.g. "this is a door") and continuous data (the input from an RGB image). Secondly, we have the already mentioned unknown poses and unknown map. Thirdly, there is an extremely large hypothesis space, or the total number of possible maps. For a map of $\boldsymbol{n}$ cells, there are 2^n maps. Our goal is to estimate the posterior of the map over this space, which is potentially a computationally heavy burden especially outdoors or in geometrically complex spaces. In fact, standard Bayes filters used in localization will diverge. Thus algorithms for OGM must allow for the high dimensionality. Fourth on our list of challenges is sensor implementation. For example, a laser range finder has a limited range. A camera has a limited field of view. As the robot moves around it collects data about obstacles and walls. From one frame to the next it collects overlapping data, each an 'instantaneous map', that may include previously seen features. A number of challenges exist in condensing the overall map from this data:

1) Environment size. As most robots use a form of microcontroller and are limited in memory, the large amount of data (instantaneous poses and obstacles) must be able to cope with it.
2) The map may be larger than the robots perceptual range
3) Sensor data may be noisy.
4) Perceptual ambiguity. This occurs when two places look alike, such as a series of similar corridors in a building.
5) Cycles. This is where the robot moves back and forth repeatedly, causing accumulated error in odometry.

The OGM process can be described as follows. During SLAM the robot builds a map and localizes itself relative to it. After SLAM, the OGM Algorithm must a) filter the exact poses from SLAM and then b) with known poses and noisy measurements, generate the map. The map is then *generated in post processing* using a form of the function

$$(P(m|z_{1:t}, x_{1:t})$$

where m is the map,

$$z_{1:t}$$

and

$$x_{1:t}$$

represent observations and poses, respectively, from *time = 1* to *time = t*. (Note that the controls, *u*, are not included here. However they are used during SLAM to estimate the poses). The map is populated with zeroes and ones, according to whether the space is occupied or unoccupied. Due to the computational burden of computing the posterior over high dimensionality maps, the problem can be reconceptualised as computing the instantaneous map

$$m_i$$

at each time step, and using the product of marginals, given by

$$\prod_i p(m_i|z_{1:t}, x_{1:t})$$

. This addresses the dependencies between neighboring cells and reduces the memory requirement. It thus becomes a *binary estimation problem*, a function of the measurements only, given by

$$bel_t(x) = p(x|z1:t$$

. Each grid cell holds a *static state* and does not change during sensing. As a result, a *Binary Bayes* estimate can be used, applying the *log odds ratio*. This belief, known as the *inverse measurement model* (IMM), is updated for each grid cell dependent on measurements. The IMM is a good option when the complexity of the measurements is greater than the binary static state, for example the situation described above where a camera views a door (complex, continuous measurement) and must assess whether it is open or closed (simple, discrete). Bayes solves the IMM by using the log odds ratio representation. The new belief is thus represented by

$$l_t = l_{t-1} + log\frac{p(x|z_t)}{1-p(x|z_t)} - log\frac{p(x)}{1-p(x)}$$

The three expressions on the right side of the equation are the previous log ratio odds belief, the new log odds of the IMM, and the initial belief, or the state of the system prior to acquiring new data. (Note that the log odds ratio calculation may include additional expressions to process the specific sensor model used by the robot, such as the Inverse Sensor Model) The algorithm for OGM is given by:

The algorithm contains a loop that cycles through all the grid cells, updating the ones that are covered by whichever sensor model is in use. In the case that the cell is not covered by the sensor model, the value of that cell remains unchanged.

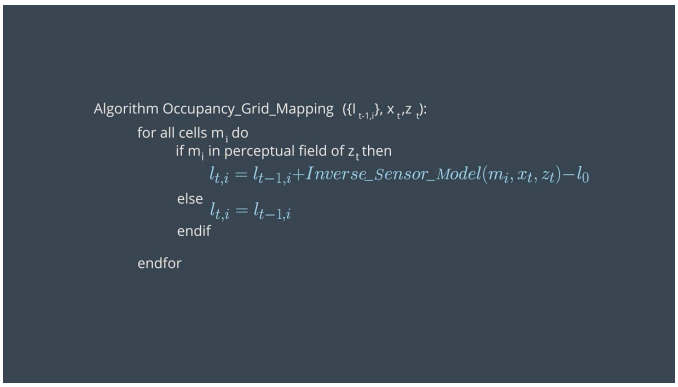

Fig. 2. Occupancy Grid Mapping Algorithm Credit: Udacity .

2.1.1 3D data representation

For the purposes of this project both a 3D map will be produced (though in so doing, a 2D occupancy grid map may also be derived). Though such maps require a great deal more processing power, they provide certain advantages over a 2D map, particularly in the areas of obstacle avoidance, motion and path planning, as well as being of particular use when dealing with airborne robots. Many types of 3D sensors exist, and here an RGBD sensor will be used, in the virtual case a Kinect v1 type sensor (simulated in ROS) and in the case of the real robot, a kinect V2 type sensor. For such mapping, a probabilistic representation is needed, as this can allow for both sensor noise and changes in the environment. The map must distinguish between free and unknown cells so that the robot is able to plan a path that is clear of obstacles. The mapping process must be efficient in terms of memory and accessible in the main memory of the robot. Again there are many types of representation, including Point clouds, Voxels, Octrees, Elevation Maps and Multi-level surface maps. However this project will focus on using a specific type of octree (see section on RTABMAP) and point clouds. The latter is memory efficient and uses voxels which can be subdivided recursively as needed. Empty space need not consume memory as with point clouds, as the program will only assign additional cells in the event that a measurement is made in that cell. This also means that the size of the map does not need to be known prior to commencing mapping as the cells do not need to be initialized until needed.

2.1.2 Online SLAM vs Full SLAM

SLAM types can be further subdivided into two broad headings - Online SLAM and Full SLAM. Online SLAM, when finding the posterior of the map and pose, only uses the controls and measurements at each time step, given by:

$$p(x_t, m|z_{1:t}, u_{1:t})$$

. On the other hand, the Full SLAM problem estimates the posterior over the entire path:

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

by integrating all the poses from the online SLAM problem (See Probabilistic Robotics; Thrun et al).

2.1.3 Discrete and Continuous natures

As alluded to above, there are two 'natures' of SLAM data , *discrete* - Correspondences and *continuous* - Objects, Locations and Robot poses. As poses and object data are continually sensed, so they are continuous data. However, a robot must also sense if it has visited a location before, or seen an object before. This *correspondence* question is a 'yes or a no', and thus, binary in nature. Correspondence must

be included in our calculations. Here the updated posterior for the Full SLAM problem is considered again:

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

, where c is a correspondence value that establishes a relation between objects. As above the poses are then integrated over the full path.

2.1.4 SLAM Challenges

As the robot explores the environment, so the number of *continuous* variables to keep track of increases. This space is highly dimensional. Additionally, as the robot must keep track of all of the correspondences between these objects, the *discrete* space is also highly dimensional. The complexity increases exponentially, and as it is not possible to compute the posterior under unknown correspondence, the SLAM algorithms must use an approximation of correspondence. In this way memory use can be more conservative. One of the main challenges in modern SLAM is accomplishing mapping at speed. This is especially important in spaces where safety is important. Presently this is addressed by separating robots from humans and using rigid paths for robots to follow. However this limits flexibility of robotic applications.

2.2 FastSLAM

FastSLAM is a SLAM algorithm that uses particle filters. However due to the high dimensionality described above, the size of the MCL matrix will increase exponentially. The FASTSLAM algorithm is one solution to this problem. It solves the Full SLAM problem with known correspondences, using a specialized Probability Function. Particles are employed to estimate the posterior over the entire robot path at the same time as the map. At the same time, each particle contains a map and a local guassian can be used to represent each feature of the map. Thus there are two independent problems, each estimating features of the map. To do this, FastSLAM uses a low dimensional EKF, and the dependency exists between the robot pose uncertainty whilst map features are treated independently. This approach is known as the *Rao-Blackwellized Particle Filter*. In essence, with the MCL, FastSLAM estimates the robot trajectory, and with the low dimensional EKF, FastSLAM estimates the features of the map. It can solve both the online and Full SLAM problems. Further, Grid Based FastSLAM (GBFS) adapts FS to grid maps. The reason for this is that the standard FS approach requires known landmark positions, which may not always be the case. GBFS extends the FS algorithm to OGMs by using the OCM algorithm instead of the EKF to estimate features of the map. Now, each particle contains the trajectory of the robot, in addition to the local map. It then solves the mapping with known poses problem using the OGM algorithm.

2.2.1 Grid Based Fast SLAM techniques

Three probability functions represent the three methods needed to adapt FS to grid mapping: Sampling Motion (MCL) given by:

$$p(x_t|x_{t-1}^{(k)}, u_t)$$

, Map Estimation (OGM) given by:

$$p(m_t|z_t, x_t^{(k)}, m_{t-1}^{(k)})$$

and Importance Weight (MCL) given by:

$$p(z_t|x_t^{(k)}, m^{(k)})$$

.

2.2.2 Disadvantages of FastSLAM

FastSLAM uses the diversity of its particle set to implicitly maintain the dependencies in the estimate of feature locations. Complex environments can cause problems with convergence in the FastSLAM algorithm, a problem that guassian SLAM techniques suffer considerably less from. Though FastSLAM 2.0 and GBFS can close larger loops than FS1, there is still a fundamental limitation to the algorithm due to its reliance on the number of particles used.

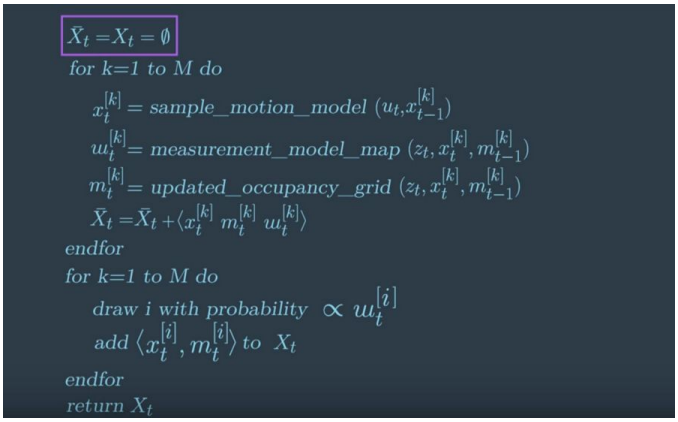

Fig. 3. Grid BAsed Fast SLAM Algorithm Credit: Udacity .

2.3 GraphSLAM

GraphSLAM has a number of advantages. It recovers the entire path and map, rather than just the most recent pose and map. As a result it can consider the dependencies that exist between the current pose under consideration, and poses it has already occupied. GraphSLAM has a reduced processing overhead. It also resolves the problem presented by FastSLAM, where there may be insufficient particles available at the location required by the algorithm. This is especially true in large mapping spaces. GraphSLAM additionally has all accumulated data available to it in order to compute the optimal solution.

2.3.1 Graphs and Constraints

The figure below illustrates the concept of the graph in GraphSLAM (GS). At each timestep, new nodes are added. These nodes represent the robot's pose (triangular shapes) and observations it makes (star shapes). Linking each of the nodes is an 'edge', also known as a *soft spatial constraint*. The term 'soft' is applied due to the inherently uncertain nature of motion and measurement data.

The solid line represents a *motion constraint* between one robot pose and the next. The *dashed line* represents a measurement constraint between the poses of the robot and

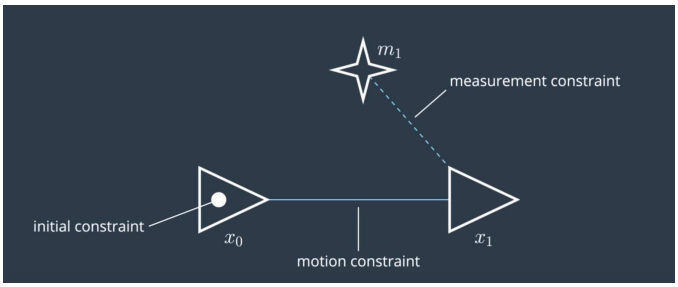

Fig. 4. GraphSLAM Constraints.

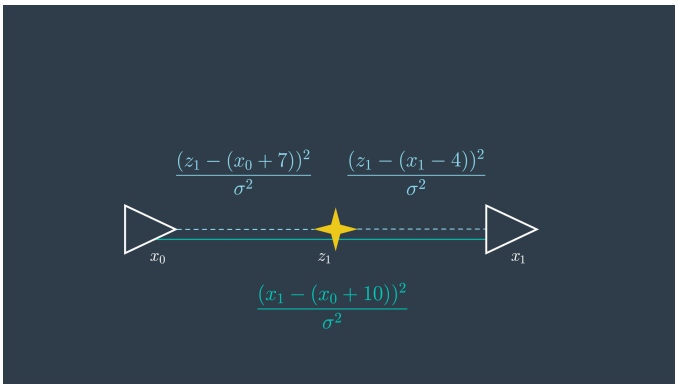

Fig. 5. Example nodes and representative simplified negative log likelihood error.

some feature observed by the robot's sensors, such as an edge or other definable geometric feature. The GS algorithm can store and process a large quantity of features. Naturally due to inaccuracy, these constraints will display error relative to one another and thus conflict. The overall aim of the GS algorithm is to find the most likely 'balance' of these constraints, often compared to a system of springs finding their equilibrium, effectively finding the configuration that reduces the overall error to a minimum.

2.3.2 Front End and Back End Processing

There are two architectural elements to graphSLAM, both of which contribute to finding the poses and features, and then creating the map and retrieving the path. These are the Front End and Back End. The **Front End** is responsible for constructing the graph, creating the nodes and edges using odometry and sensory measurements. It must solve the data association problem, that is identifying features that it has seen before. Across different applications and sensor types, the front end can be very different. The **Back End** receives the completed graph from the Front End, and returns the configuration of robot poses and measured features that is most probable. As described above, it is a process of optimization whose aim is to return the configuration of lowest error. The front end and back end can be performed in succession or iteratively. In the latter case, the completed optimzed graph can be fed back to the front end.

In contrast to the front end, there is much greater commonality across applications, goals and sensor types when it comes to the back end.

2.3.3 Maximum Likelihood Estimation (MLE)

Optimizing the graph, that is, minimising the error contained within all of the constraints, lies at the heart of the Graph Slam algorithm. MLE is the means to solve this optimization. **Likelihood** is a complementary principle to probability. In SLAM the MLE, given the observations and measurements, tries to find the most likely arrangement of objects and poses. The figure below shows an example configuration of nodes and constraints, with their representative simplified negative log likelihood error is shown:

To optimize the set of constraints, it is necessary to minimize their sum, given by the following *error function*:

$$J_{graphSLAM} = \frac{(z_1-7)^2}{\sigma^2} + \frac{(x_1-(x_0+10))^2}{\sigma^2} + \frac{(z_1-(x_1-4))}{\sigma^2}$$

To do so, the first derivative of the function is taken and set equal to zero. However the ensuing system of equations that must be solved as a result becomes more and more complex to calculate as the map grows. For this reason the equations can be solved *numerically*, rather than *analytically*. An algorithm is applied to find the local minimum of the error function. In SLAM, effective algorithms can include the gradient descent, Levenberg-Marquardt, and conjugate gradient methods. In *n-dimensional* graphs, matrices and covariances are used to represent the constraints. The equations thus become

$$(x_t - g(u_t, x_{t-1}))^T R_t^{-1} (x_t - g(u_t, x_{t-1}))$$

for the motion constraints and

$$(z_t - h(x_t, m_j))^T Q_t^{-1} (z_t - h(x_t, m_j))$$

for the measurement constraints, Where h()h() and g()g() represent the measurement and motion functions, and Q and R are the covariances of the measurement and motion noise. The multidimensional formula for the sum of all constraints is thus written:

$$J_{GraphSLAM} =$$
$$x_0^T \Omega x_0 + \Sigma_t (x_t - g(u_t, x_{t-1}))^T R_t^{-1} (x_t - g(u_t, x_{t-1}))$$
$$+\Sigma_t (z_t - h(x_t, m_j))^T Q_t^{-1} (z_t - h(x_t, m_j))$$

The first element sets the first robot pose at the origin of the map and is the "initial constraint", with complete confidence as given by omega.

2.3.4 Information Matrix and Vector

The above matrices and equations are unweildy to work with. For this reason, GraphSLAM employs The Information Matrix and the Information Vector.

The *Information Matrix* (IM) is the inverse of the *Covariance Matrix*:

$$\Omega = \Sigma^{-1}$$

and higher values in the IM indicate higher certainty.

The principle of the IM and IV can be seen from the figure above. Broadly, associations are made between nodes, which are shown as occupying a cell. For example, an occupied cell at (x1, x2) would illustrate a constraint between pose 1 and 2; An occupied cell at (x3, m1) would be a constraint between pose 3 and measurement 1; An occupied cell at (m2, m3) is a constraint between measurement 2

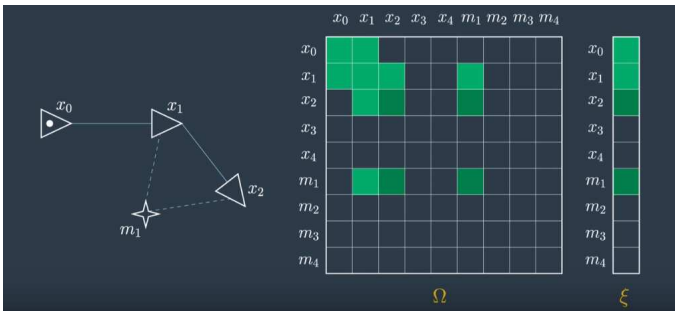

Fig. 6. Information Matrix and Information Vector.

and measurement 3. As new nodes are added, so each cell is updated additively via their negative log likelihoods. The corresponding cells are also updated in the IV. If no information is available about a cell, its value is 0. Most of the off diagonals are zero in fact, a feature known as *sparsity*. This is a useful property of the matrix which offsets the high number of features commonly found.

2.3.5 Inference

From a fully populated IM and IV, the path can be extracted using the following equation

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

Where the mu is a vector. It is defined over all the poses and features and contains the most accurate estimate of each. As the size of environment increases, so too do the vectors. As there is a matrix inversion involved, efficiency is a concern with this process. As most environments and applications are cyclical (the robot revisits areas more than once), and the revisited areas are separated in time, the matrix is harder to recover as non-zero cells cannot be reordered to the diagonal so readily as in the linear case. As a result, a variable elimination algortihm is applied to simplify the matrix. In this way the inversion and product can be computed more quickly. With reference to the "spring force" analaogy above, links can be adjusted or replaced to accommodate for the links to be removed, maintaining the "force balance". However this analytical method is rarely applied and again, numerical methods are more efficient and so frequently used. They are able to converge in a fraction of the time.

2.4 RtabMAP

RTABMAP (RTB) is a competition winning SLAM algorithm by Matthieu Labbe, that uses vision sensors to collect data and then localize the robot and map the environment. The main strength of RTB is its application loop closure, which determines if the robot has seen the location before. As the robot collects new images, so it creates a database that all new images are progressively compared to. As such the database increases in size as the map grows, causing the process of comparison to take longer. Complexity increases linearly. RTB uses clever memory management methods to create an algorithm that works long term and over large mapping spaces. It is able to accomplish its goal in real time as the loop closure can be accomplished faster than the new camera images can be obtained. There are also small differences between the front end and back end of traditional graph slam in RTB. The FE / BE arrangement is illustrated in the following figure:

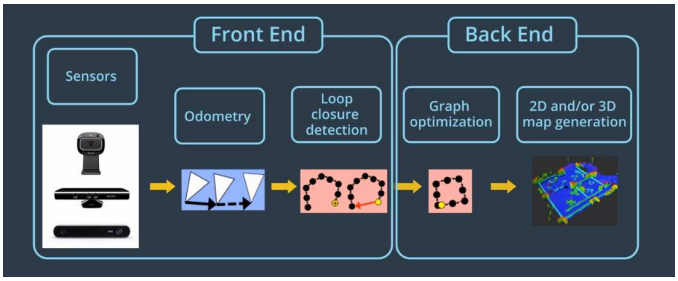

Fig. 7. Front end and back end of RTABMAP.

The main focus of the front end is in sensors, odometry and loop closure detection. Unlike most 2D graph slam applications, RTB does not use landmarks. Only the constraints produced by odometry and loop closure. The odometry can come from wheel encoders, lidar, IMU, visual odometry, the last of which is accomplished by identifying and tracking specific 2d features, such as those found in SURF (Speeded-Up Robust Features). Though the approach is powerful and can use just one monocular sensor, a weakness of appearance-only based mapping is that it contains no distance data. In order to add this 'metric' data, it is necessary to add an RGB-Depth (RBGD) or stereo camera. With this equipment, RTB can compute the geometric constraint between the images and loop closure. It is also possible to add a laser range finder, which will refine the geometric constraint and so return a more accurate localization. The FE also uses "Bag-of-words" in graph management, adding nodes and loop closure detections. All of this information is optimized, also by the front end, as well as assembling an occupancy grid.

2.4.1 Loop Closure

Loop closure is important for graph optimization and ensuring that duplicates of already-seen locations are not added. Within many SLAM applications, matches are found between visited locations at two levels - local and global. This approach is likely to fail as the map size becomes larger. RTB is different in that it only uses global loop closure and memory management techniques to ensure real time performance.

2.4.2 Visual Bag Of Words

RTABMAP uses a method called "Visual Bag of Words" (BOW) to accomplish loop closure. The above figure provides an overview of the process. The aim is to identify features that have been seen before. Its input is the camera image, and each of those images is of course composed of features. Such a feature could be a patch of wall or brick with a complex pattern, or a corner of a window or door. RTB uses as default the SURF (Speed Up Robust Features) method for extracting features. It extracts precise combinations of pixels that make up these features, termed a *descriptor*, each of which is unique. SURF breaks these areas up into smaller areas, and pixel intensities in regions

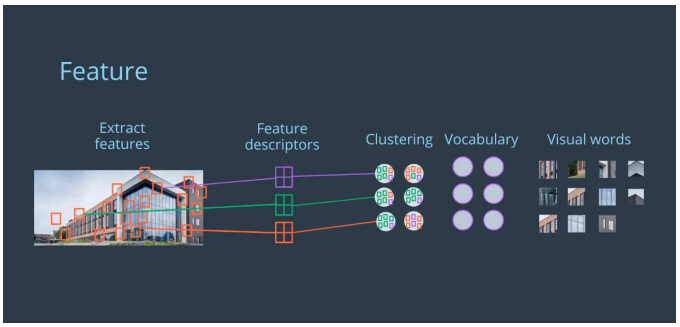

Fig. 8. Visual Bag Of Words.

of regularly spaced sample points are calculated and compared. When these differences are found, they can be used to categorize the sub-regions of the image. A disadvantage of this method is that comparing feature descriptors directly takes a lot of time, and RTABMAP requires real time operation. For this reason a 'vocabulary' is used for faster comparison. A vocabulary is created by clustering similar features together, known as *synonyms*. The collection of these clusters represents the vocabulary, in a process known as quantization. Each feature is in this way linked to a word. It is now known as a "visual word", and a fully quantized set of features can now be called a "bag of words". In order to make image retrieval faster and more efficient, a link is made between each word and the image it was found in. A 'matching score' is assigned to all images containing the same words. This score is known as an *inverted index*. So if a particular word is observed in a new image, its score will increase. The more matching features with a previously seen image, the higher the score, and this is assessed by a Bayesian Filter. The filter provides the hypothesis that an image has been seen before, and when a certain threshold H is reached, a loop closure is detected.

2.4.3 RTABMap Memory Management

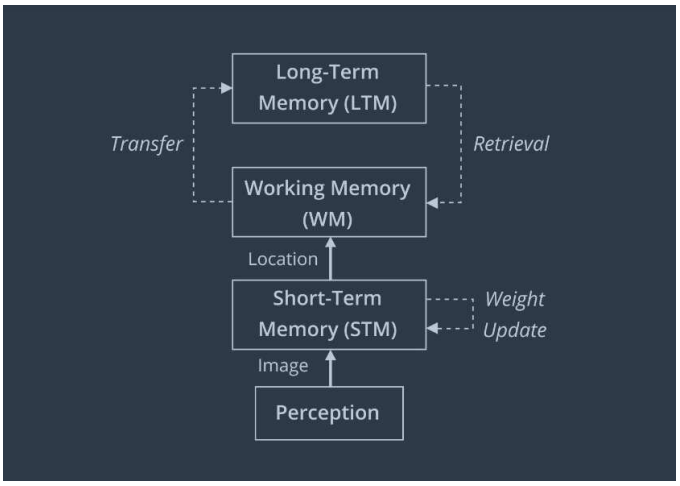

Fig. 9. RTABMap Memory Management.

As mentioned above, the process of loop closure requires a large number of previously seen locations and the associated vocabulary to be stored in memory. For this reason RTB employs a memory management approach which places a limit on the number of locations that may be considered as candidates for loop closure detection. The main goal for this memory management is to maintain the most recent and frequently observed locations stored in **working memory (WM)**, and offload all others into **Long Term Memory (LTM)**. The process is as follows:

1) A new image is received from the camera. A new node is created in the **Short Term Memory (STM)**. The STM has a fixed size, denoted by **S**.s
2) A bag of words is created for this new image / node.
3) Nodes stored in the STM are not considered for loop closure detection. This is because they will be extremely similar to each other, having come from the same area.
4) A weighted analysis and the age of each node is considered, in order to determine which nodes are passed to the WM, and so can now be considered for loop closure detection. *Loop closure detection thus occurs in the WM*. The weighted analysis takes into account how long the robot spent in a given area. It does this by comparing two consecutive nodes, and if they are similar, the first node's weight is increased by one. No new node is created for the second image. When the size S of the STM is reached, the oldest added node is transferred to the WM.
5) A time limit that governs how long it is acceptable for the loop closure process to take place. If the process takes in excess of this limit, some nodes of the graph are transferred to the LTM. *In this way, the size of the WM can be kept almost constant*.
6) Nodes in the LTM cannot be considered for loop closure and graph optimization. If a closure is detected, the neighbouring locations of an old node can be transferred back to the WM. This process is called *retrieval*.

The figures below illustrate some of the ways that rtabmap represents loop closures.

2.4.4 RTABMap Optimization and Output

Graph Optimization. As described above, the graph errors must be minimized. RTABMap supports three types of optimization - Tree Based Network Optimizer (TORO), General Graph Optimization (G2O) and GTSAM (Smoothing and Mapping). Following the acceptance of a loop closure hypothesis, and new constraint is added to the graph and RTB will use the selected optimization method to eliminate the errors in the graph. Each of the named optimizations works with node poses and link transformations as constraints. Errors in the odometry can be propagated to all links, correcting the map. This is in contrast to other GraphSLAM methods which also use landmarks in the process of graph optimization. The eventual output of RTB is one of either a 2D Occupancy Grid Map, a 3D Occupancy Grid Map or a 3D Point Cloud. RTB has a constant time complexity due to the management of how many nodes may be considered for loop closure.

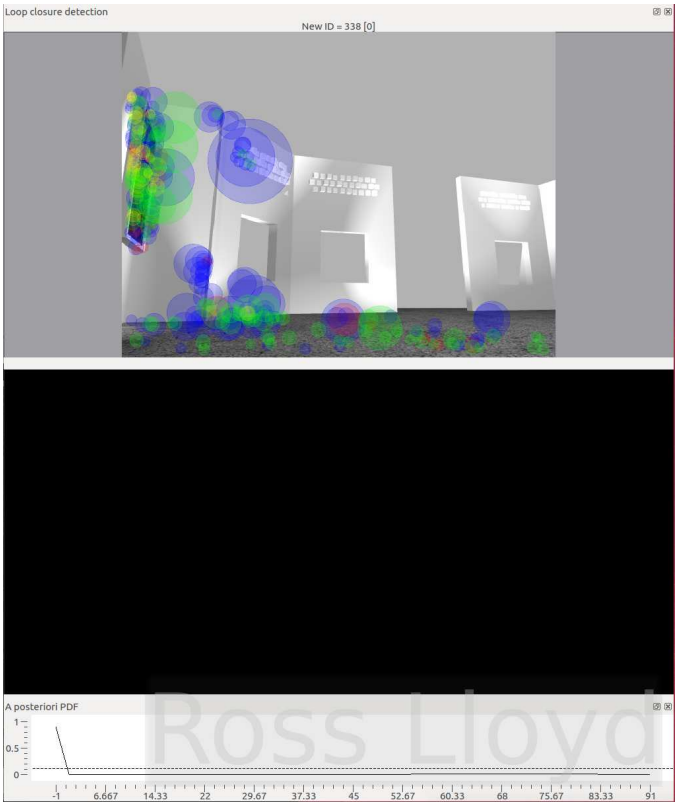

Fig. 10. Images moved to LTM marked as blue circles.

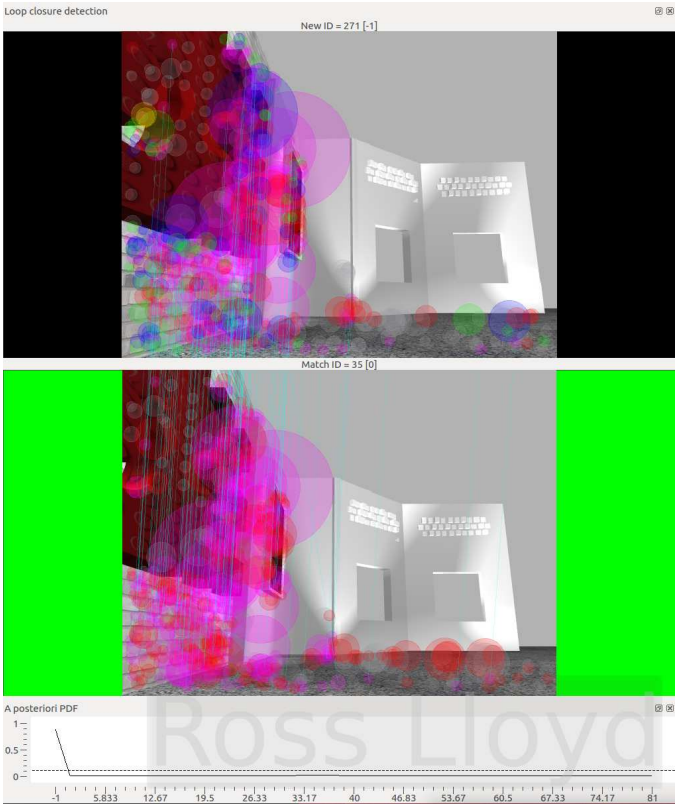

Fig. 12. Loop closure detected. Note very high pdf value.

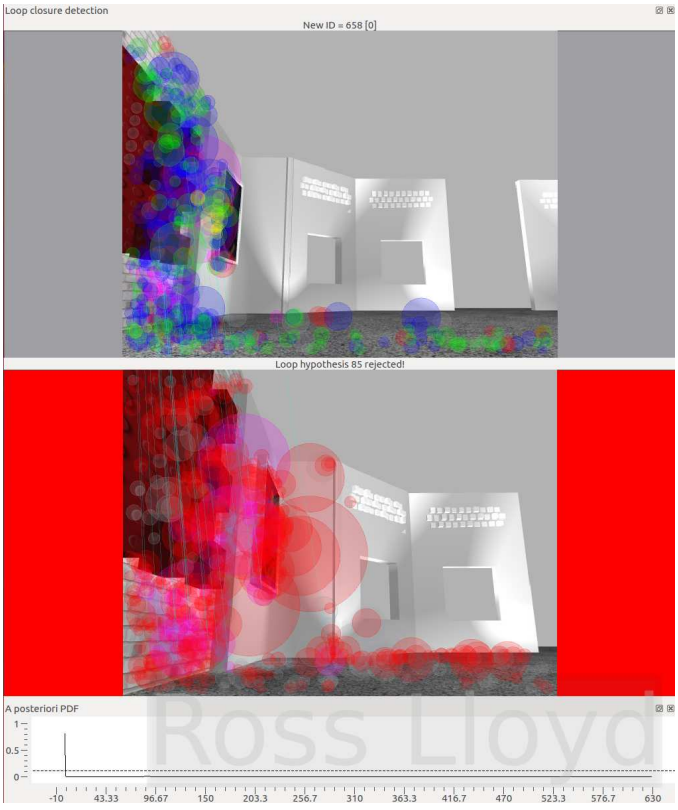

Fig. 11. Rejected Loop Closure shown in red.

Fig. 13. 3D Loop closure representation.

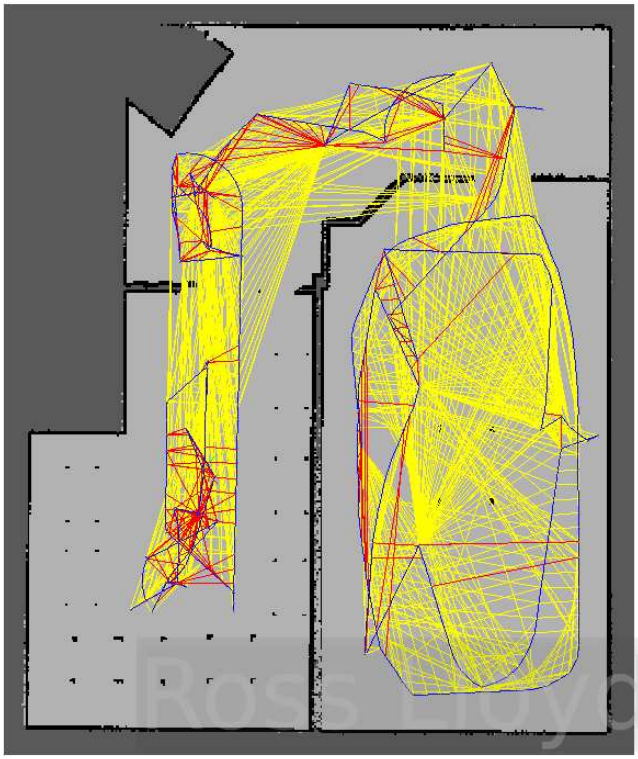

Fig. 14. Graph view showing constraints of global (red) and local (yellow) loop closures.

3 Simulations

3.1 Gazebo World

3.1.1 World design

For the section of the project requiring creation of an original map, a completely new layout was employed rather than repurposing an existing gazebo map, in order to gain experience working with the gazebo building editor. In order to provide the mapping algorithm with a rich set of features to work with, three main areas, with differing textures and custom CAD created / textured STL models, was created (see map pictures below). In room one, the Udacity logo and wall text were added to test alignment of the images, as shown by their readability and accurately reproduced geometry. Pictures and signs featuring Udacity staff in picture frames was also introduced, which serve to add colour but also will show misalignment clearly. Other details were introduced to give the "bag of words" plenty of features to work with. Plenty of space for the robot to move around was provided, so that making multiple passes over the same areas was relatively easy. It also made it less likely that objects would enter the minimum range of the robot's RGBD sensor. Both in sim and in reality, this can lead to incorrect loop closures which survive the optimization process. Room two, a lobby area, again features custom STL models, though in this case some of the walls are left more featureless to test if a lack of features would cause the algorithm to fail. This space also features the classic PR2 robot as a 'receptionist', whose desk is constructed of repurposed existing gazebo models. The wall behind it is given a brick texture and a wall feature, as well as some tesselated wall tiles again to test the algorithm's ability to deal with repeating patterns. In the final room are some more complicated geometry items- robot models taken from the gazebo model files. Additionally a number of tables are used as this frequently presents a problem for rtabmap, where legs can frequently become repeated due to the similar geometry. Lastly, a picture of a Udacity staff member and a Miura-Ori origami fold (from the original Udacity Robotics ND Term 1) is also included. Lighting is also added in all rooms. Three rooms present enough of a challenge to the loop closure algorithm without creating extremely large databases, which in any case were in the region of 1GB at level 2 decimation.

Fig. 15. Sixbot World - Plan View.

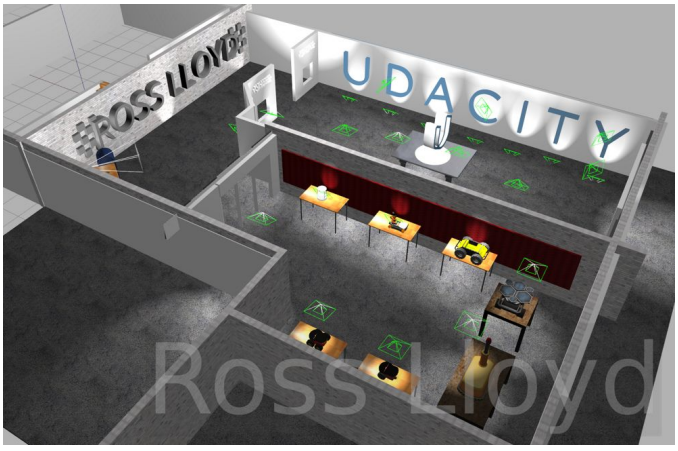

Fig. 16. Sixbot World - Iso View1.

Fig. 17. Sixbot World - Iso View2.

Fig. 20. STL models 2.

Fig. 18. Sixbot World - Iso View3.

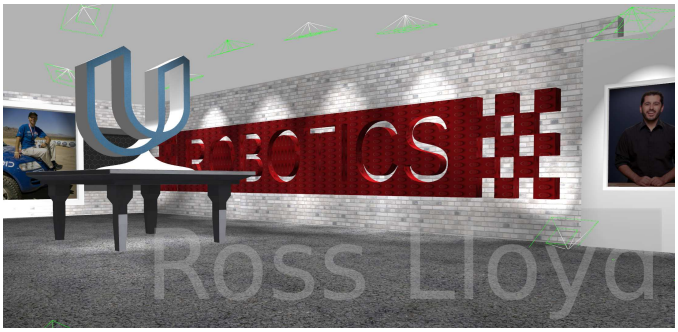

Fig. 19. STL models 1.

3.2 Robot Model

3.2.1 Model design

The type of robot created is of 6 wheel differential drive design. It features the original hokuyo laser scanner and stl mesh file, mounted on the front of the robot and low down. As planar lasers only see in one 2D plane, the low mounting point ensures that the main body intersects more accurately with obstacles detected at that height. A Kinect360 sensor is used. The STL model and textures have been taken from the ROS Turtlebot package, and a tower created to give the sensor some elevation and clearance above the planar laser. Note the laser is not fouled by this tower as the scan zone is 180 degrees forwards. Additionally, an upward angle of 15 degrees is included by virtue of a small wedge. This is to prevent large areas of the RGBD image being occupied by the featureless floor area in front of the robot and to ensure that features at higher elevations are also captured. Note that the vertical field of view of the sensor is sufficient that floor details are still captured, just not to the exclusion of more important environmental features. For detailed layout, please see the Orthographic Drawing below. The body and tower were created in a CAD package, and the wheel models downloaded from a popular CAD resource (*GrabCAD*). Then textures were created and added in *Blender* in order to decorate and better light the model in gazebo / RVIZ.

3.2.2 Packages Used

kinect_camera_controller turtlebot kinect v1 package was used to simulate the kinect 1 sensor. Some of the positioning parameters were altered and a CAD model was found online to replace the low quality one in the model (no performance impact was observed with using the more complex model). Publishes topics

- depth/image_raw
- rgb/image_raw
- camera info topics

The overall package structure is based on common ROS robot package structures such as turtlebot, and follows those outlined in the ROS wiki.

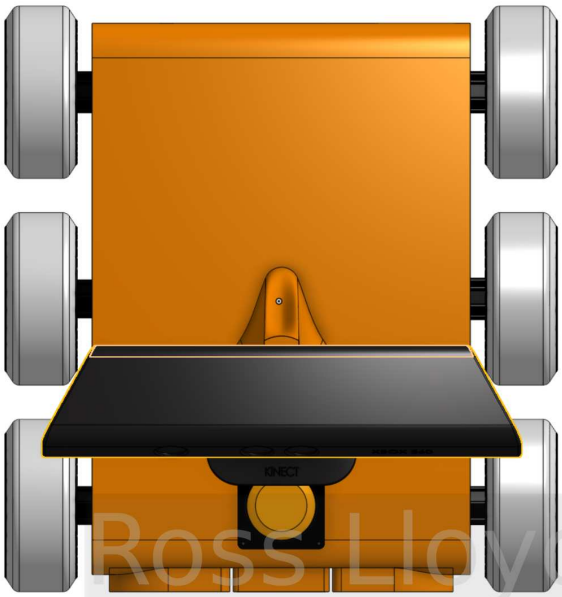

Fig. 21. CAD Plan view of sixbot.

Fig. 23. CAD Front view of sixbot.

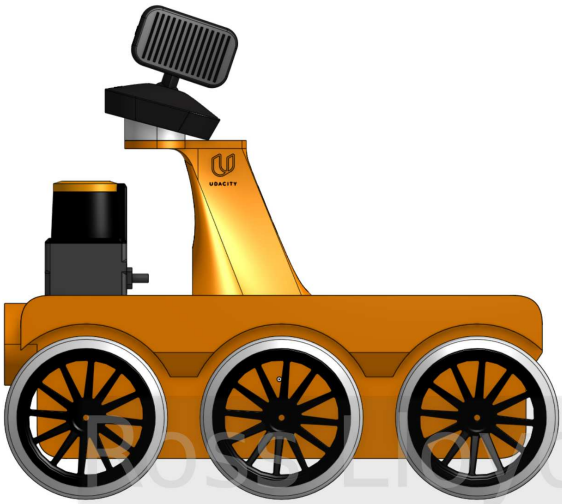

Fig. 22. CAD Side view of sixbot.

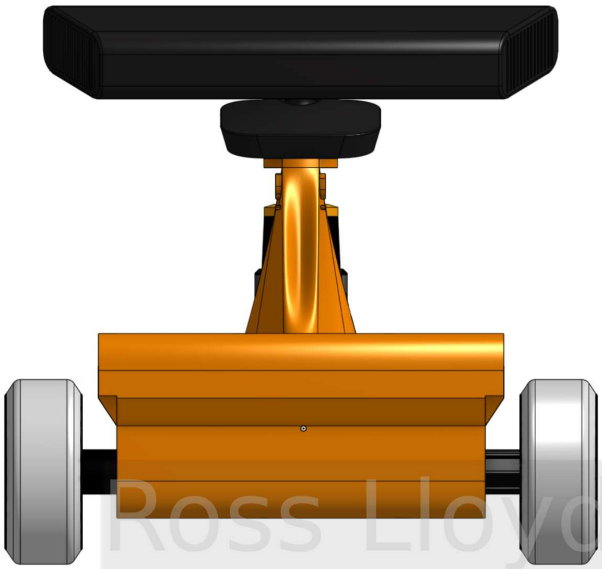

Fig. 24. CAD Rear view of sixbot.

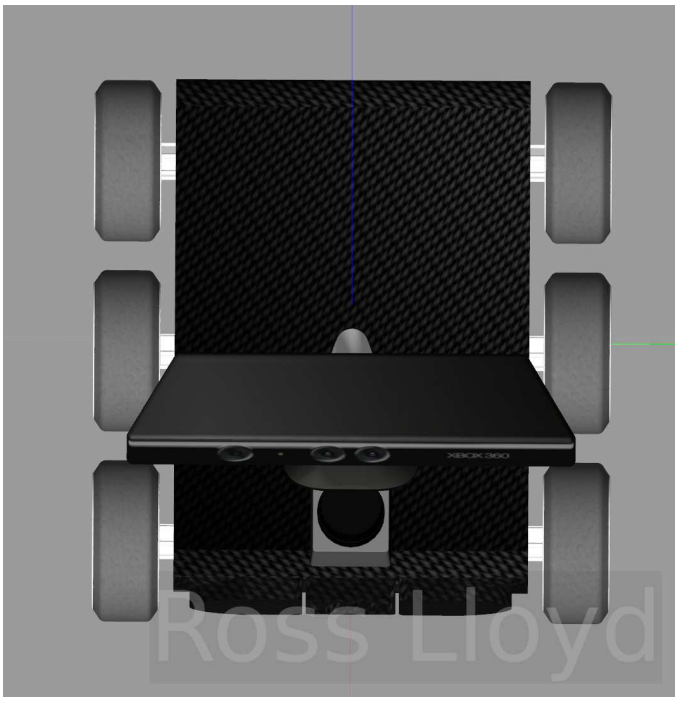

Fig. 25. Textured Gazebo Plan view of sixbot.

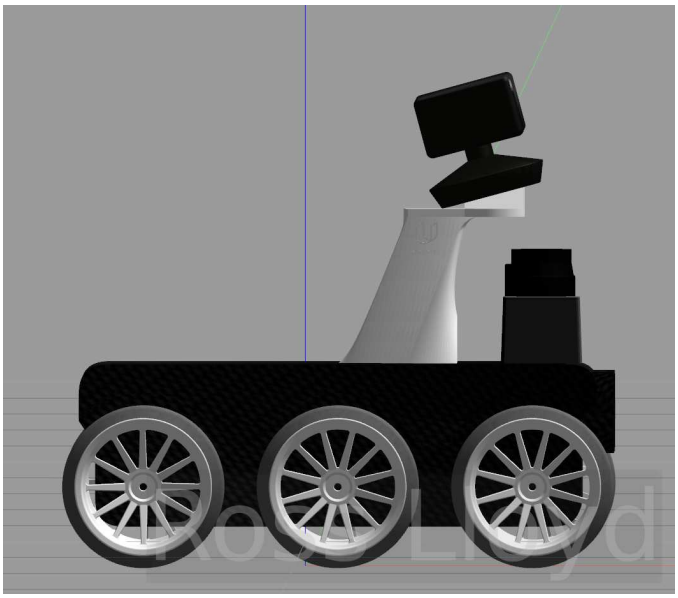

Fig. 26. Textured Gazebo side view of sixbot.

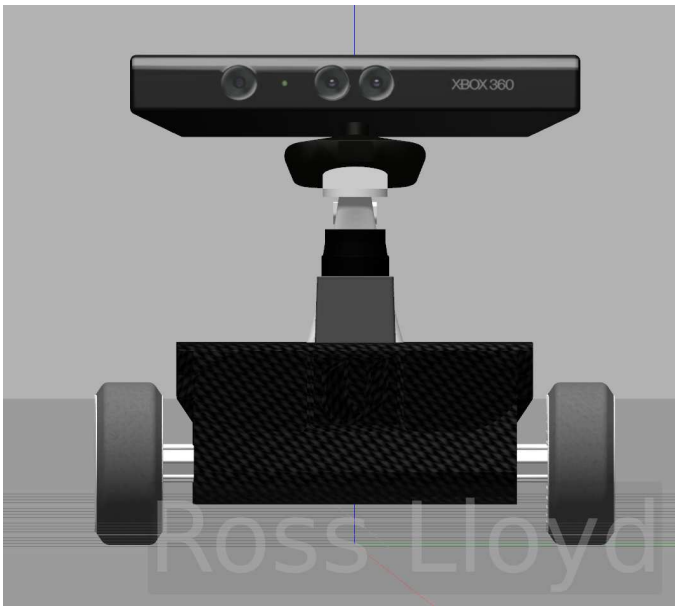

Fig. 27. Textured Gazebo front view of sixbot.

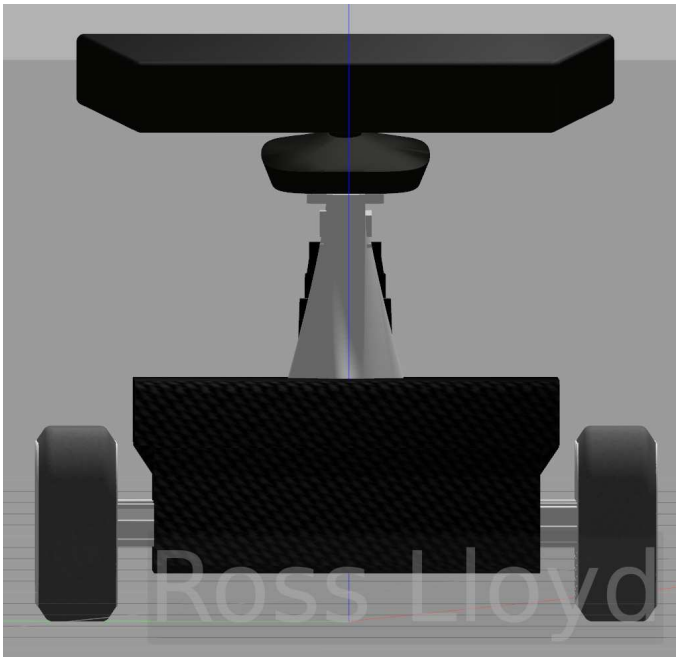

Fig. 28. Textured Gazebo rear view of sixbot.

3.2.3 Model Packages

diffDrivePlugin6W - this is a gazebo library that can drive a 6 wheel differential drive robot in gazebo. **gazebo_ros_head_hokuyo_controller** was used to simulate the hokuyo laser scanner.

3.2.4 Launch Files

Two separate launch files were created to bring the mapping project up. It was observed that bringing everything up at once could lead to ”node not found” errors and so the launch files are: *sixbot_world.launch* - launch of the world+robot, which instantiates the robot description and the custom world, along with the laser scanner nodelet

manager and rviz. *rtabmap.launch* launches the mapping node.

3.2.5 Parameters

Rtabmap parameters were as follows:

- **Loop Closure Detection**: Kp/DetectorStrategy - 0=SURF
- **Maximum visual words per image**: Kp/MaxFeatures - 400
- **Number of extracted SURF features**: SURF/HessianThreshold - 150
- **Loop Closure Constraint**: Reg/Strategy - 1 (ICP)
- **Min visual inliers to accept loop closure**: Vis/MinInliers - 10

3.2.6 URDF

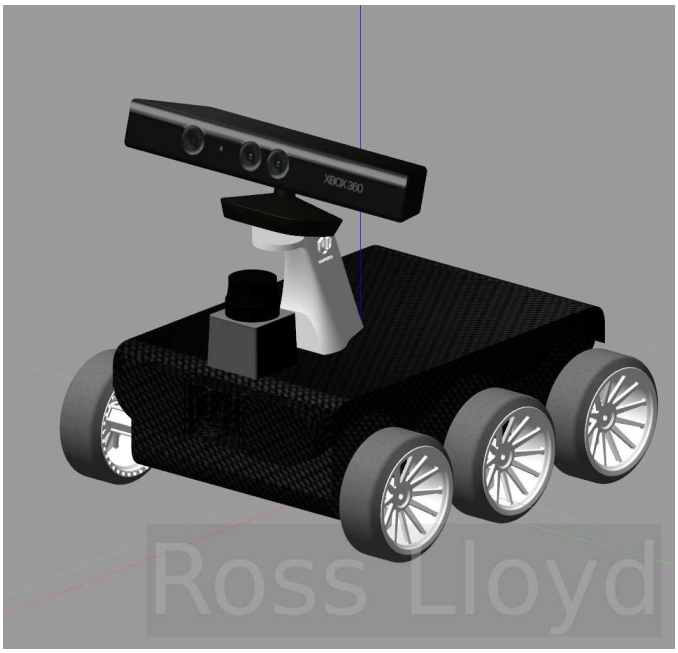

Fig. 29. Textured Gazebo iso view of sixbot.

Fig. 30. Rendered view 1 of sixbot.

Fig. 31. Rendered view 2 of sixbot.

3.3 Real World Robot

3.3.1 Robot Design / Hardware

In order to apply rtabmap to a real world mapping task, a wheeled robot platform was designed from scratch. It is a two wheel differential drive robot plus a free wheeling caster, with the capacity to carry sensors and either a laptop or TX2. The components are as follows:

- **Base** - the lower platform was built around a 200kg dolly platform which originally came with 4 casters. In the final design only one was retained, which was moved to a frontal position by means of re-drilling. Drill holes for hardware were carried out as per the CAD file so that positioning of sensors would reflect the URDF as closely as possible. Note that in the CAD models below, the caster is represented by a large dome. This is because the model was also to be used in ROS, which models casters as simple objects with zero friction. Roboclaw board and Motor CAD models were obtained from the manufacturer websites.
- **Motor housing** - the housing for the motors and the motor control board was designed in CAD around the base in order to 3D print using PLA. It features vents for cooling, routing for cables and mounting point through holes for various size bolts. A PC fan is used to provide cooling for the motor control board as it can draw up to 18A to drive the motors. There is an on/off switch which controls main power to the control board. 12V is supplied from the buck converter to power the fan.
- **Support Shelves** - wooden shelf platforms supported by CAD designed and 3D printed uprights, with through holes for securing with bolts. Holes were drilled as per the CAD file so that positioning of sensors would reflect the URDF as closely as possible.
- **Locating hardware** - for the Kinect Sensor and RPlidar, brackets were created in CAD that would ensure the sensors were accurately placed to match the URDF. These slide onto and are affixed with bolts to the wooden shelves. See pictures below for detail
- **Motors** - 2 x GoBilda 720 oz/in 105rpm DC gear motors were sourced for the main drive. The motors were a good combination of cost, torque and

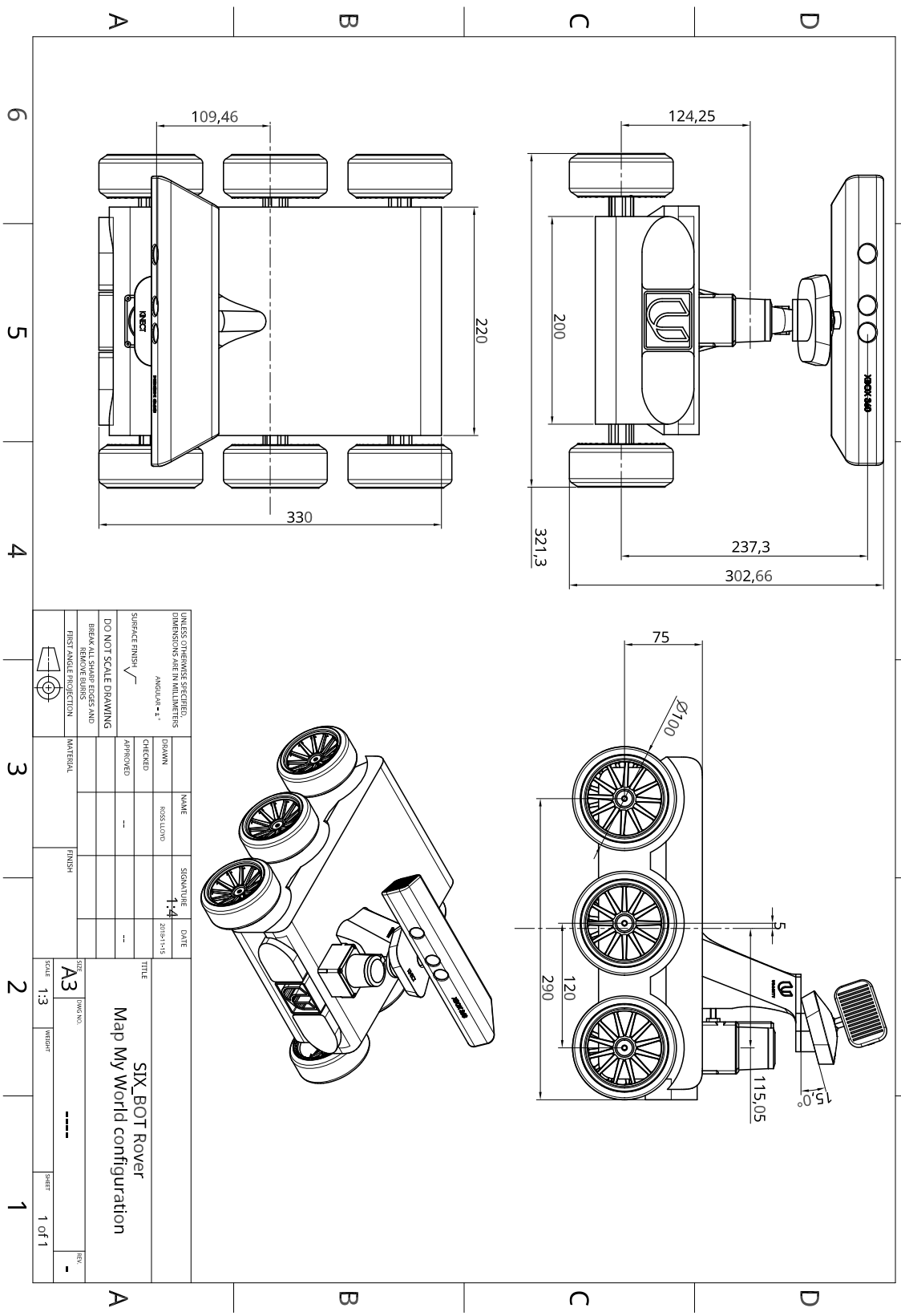

Fig. 32. Orthographic drawing of sixbot, used in the ROS simulations.

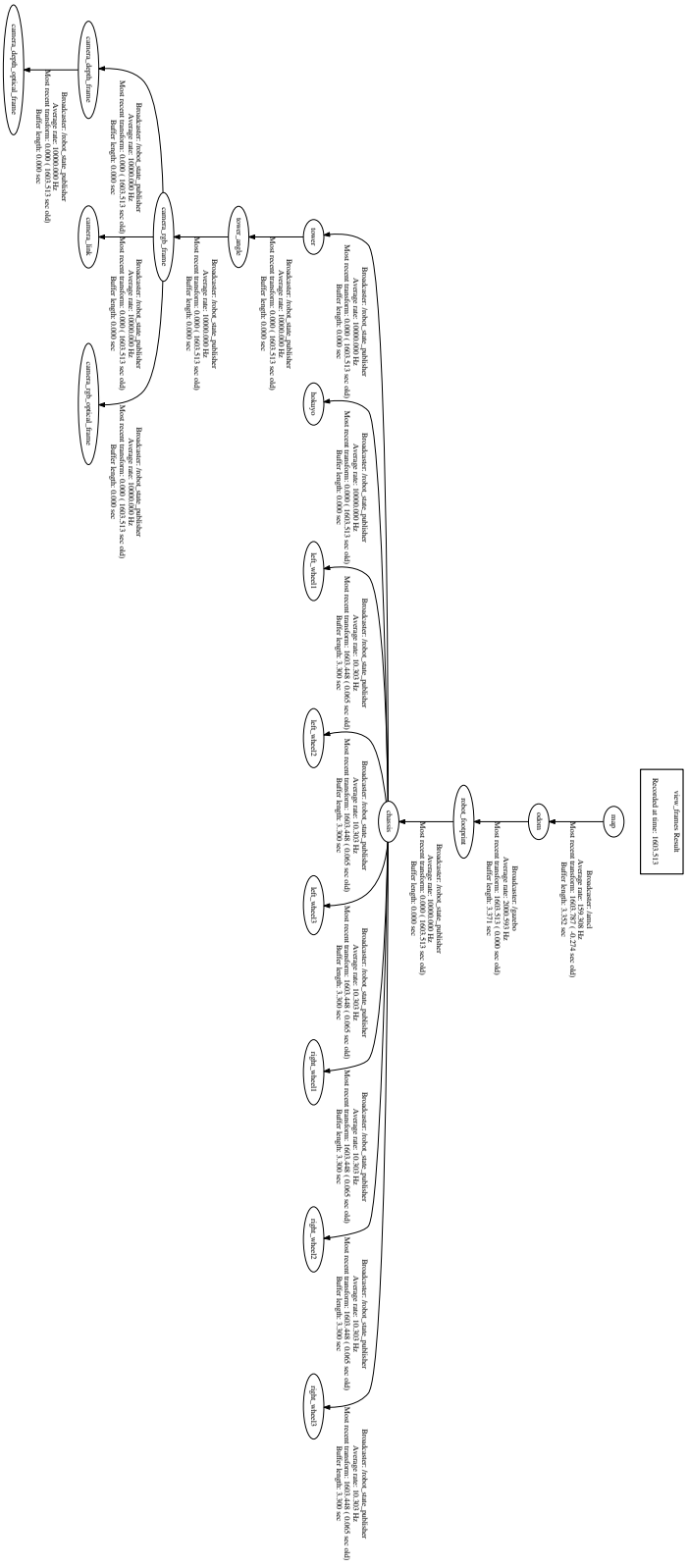

Fig. 33. sixbot tf tree.

quadrature encoders, allowing for precise odometry and differential control.

- **Motor Control Board** - 1 x Roboclaw 2x15A board. This is connected to the TX2 (localization mode) or laptop (mapping mode) via USB3. It also processes the encoder data from the two motors.
- **Wheels** - CAD designed tyres, wheels and couplings. Tyres and wheel hubs were 3D printed with flexible filament and PLA respectively, couplings were machined from aluminium and secured to the motor shafts with allen bolts.
- **Battery Power** - 1x 4S Lipo, 4500 MAh. Though sufficient for localization and free movement, it was necessary to supplement with a tether and power brick when using the laptop as the battery drained too quickly when running the needed boost converter to run the laptop. Provides 16V straight to the motors and control board. The battery is supported by a CAD designed / 3D printed battery compartment.
- **Buck Converter** - provides 12V (from 16V) to power the kinect 2 and TX2.
- **Lidar** - in the final configuration, an affordable 360 degree RPLidar A1 was used as this provided adequate accuracy for use with RTABMap. It draws roughly 0.5 amp at 5V via the USB port of the TX2 or laptop. It sits atop a riser designed to prevent undue pressure being placed on the connector cable. [Note - originally a DIY 360 degree lidar was built, designed around a Lidar Lite V3 laser rangefinder. Though it was functional, the data it produced was not sufficiently accurate or reliable enough for mapping. Shots are included below for completeness.]
- **Computing** - In the mapping configuration, it was necessary to use a powerful laptop to get sufficient frame rate from the *kinect2_bridge* node to create accurate maps. The TX2 was not able to provide the needed power in this case, but performs extremely well when rtabmap is in localization mode. The latter configuration does away with the tether.

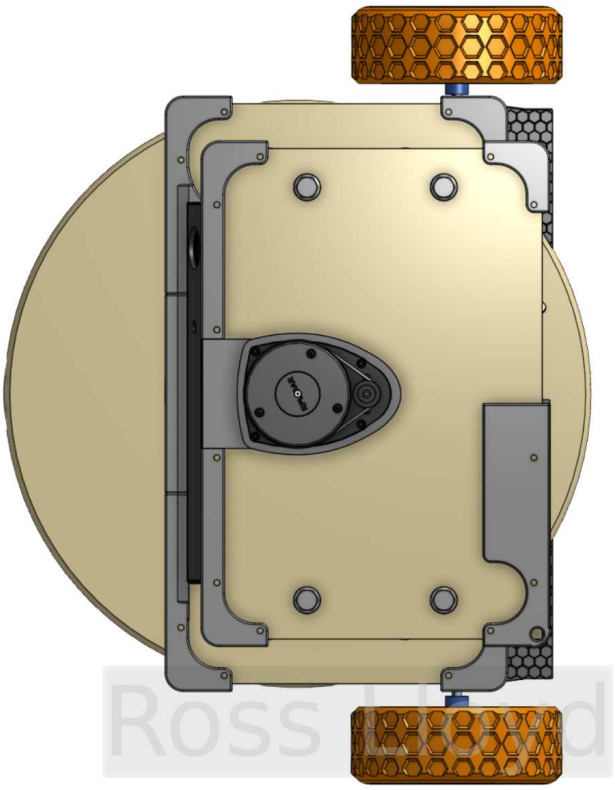

Fig. 35. CAD Plan view of gbot.

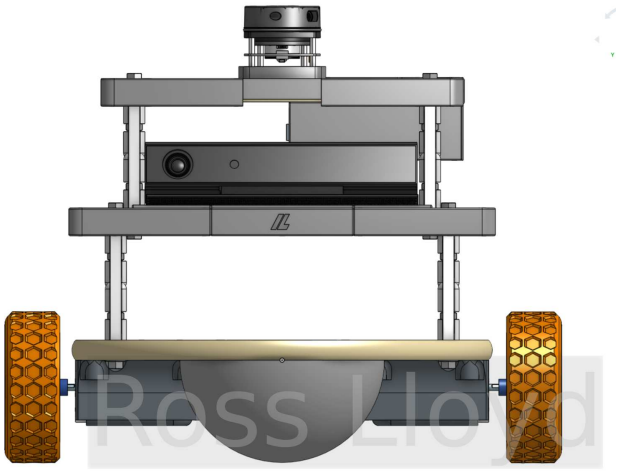

Fig. 34. CAD front view of gbot.

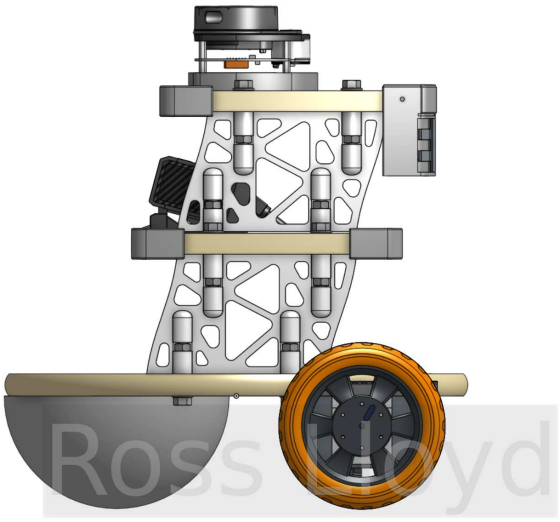

Fig. 36. CAD side view of gbot.

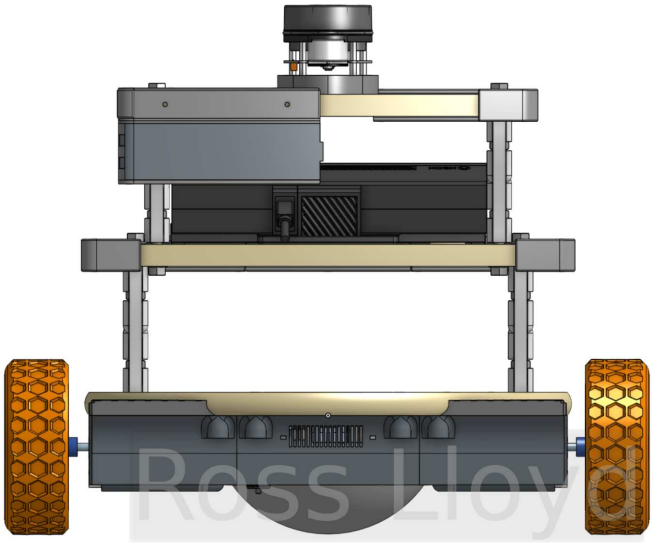

Fig. 37. CAD Rear view of gbot.

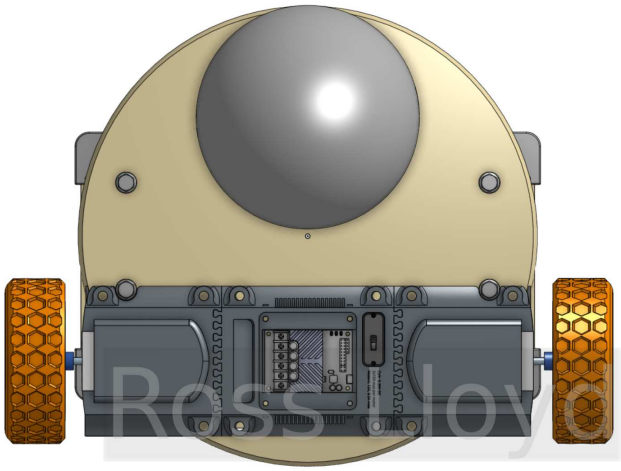

Fig. 38. Underside CAD view of gbot.

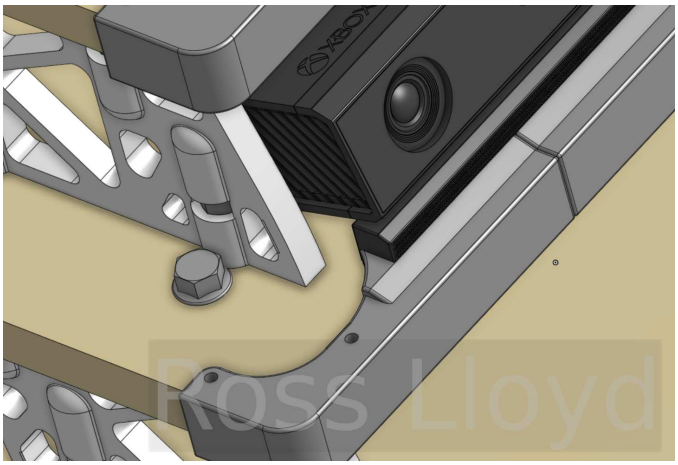

Fig. 39. Locator feature for Kinect 2 on gbot.

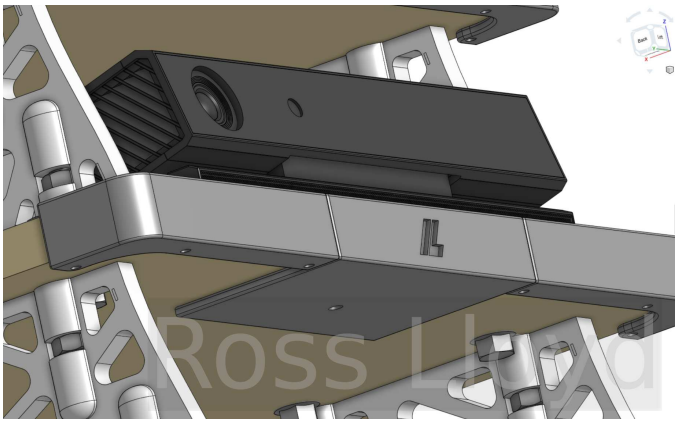

Fig. 40. Securing feature for Kinect 2 on gbot.

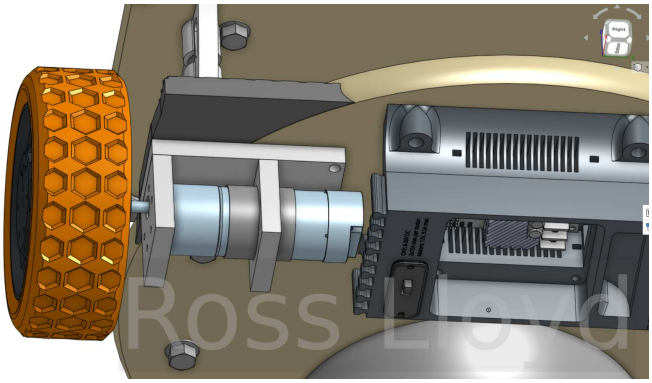

Fig. 41. Method of securing motors on gbot.

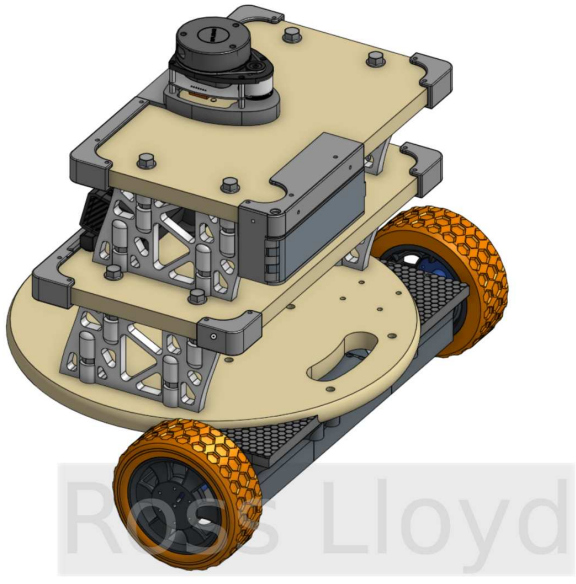

Fig. 42. Isometric view of gbot.

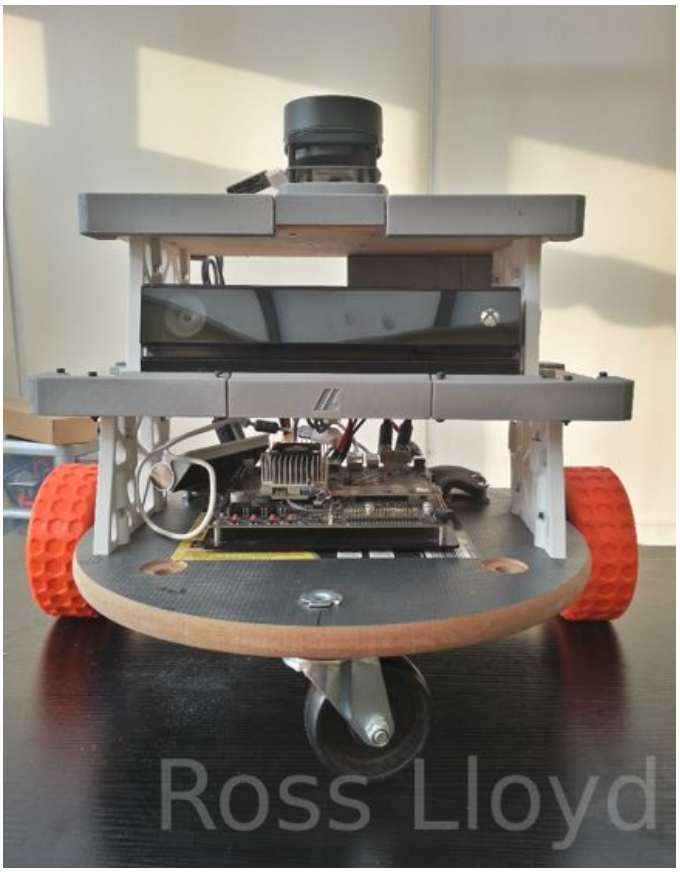

Fig. 43. Front view of gbot also showing Kinect2.

Fig. 44. Side view of gbot.

Fig. 45. Rear view of gbot also showing battery compartment.

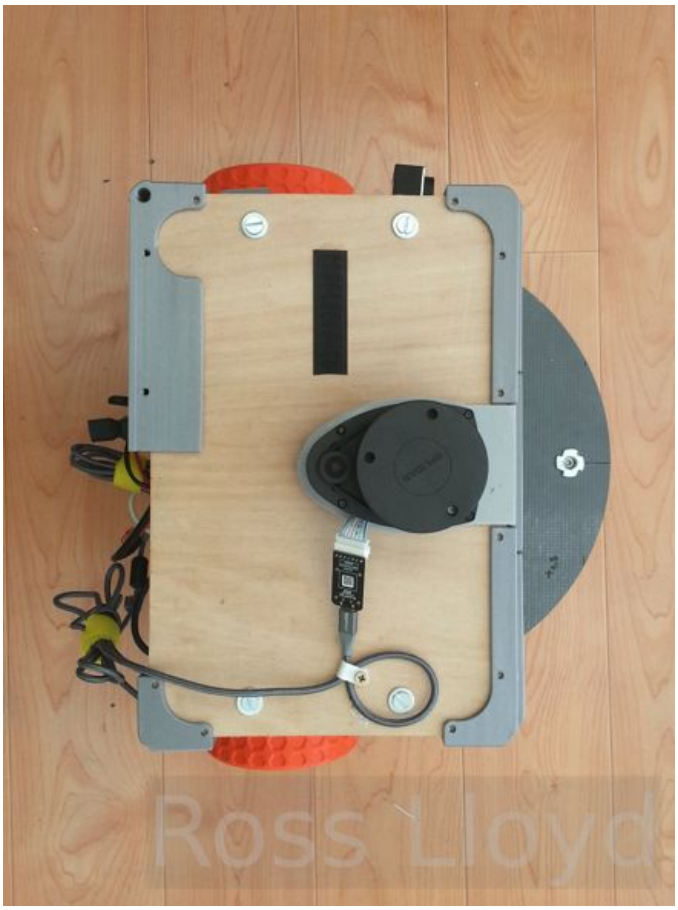

Fig. 46. Plan view of gbot.

Fig. 47. 3D printed Motor Housing.

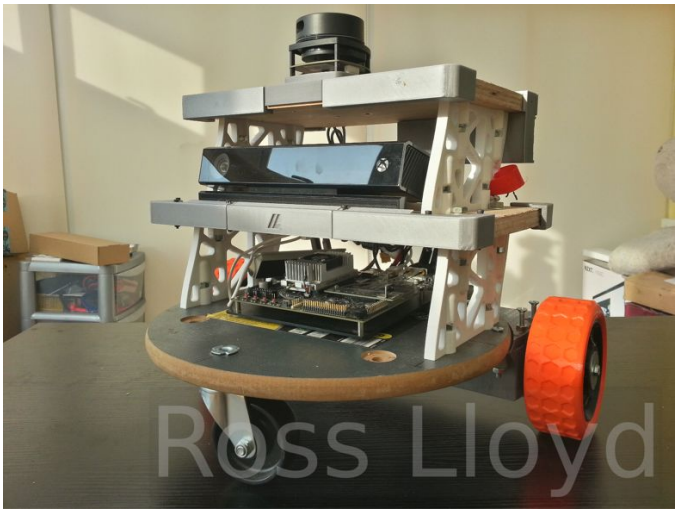

Fig. 48. Profile view of gbot.

Fig. 49. RPLidar A1 Laser Scanner.

Fig. 50. Gbot 3d printed / machined wheel 1.

Fig. 51. Gbot 3d printed / machined wheel 2.

Fig. 52. Positioning of power brick for laptop does not foul sensor.

Fig. 53. Gbot mapping configuration from front during mapping (hallway).

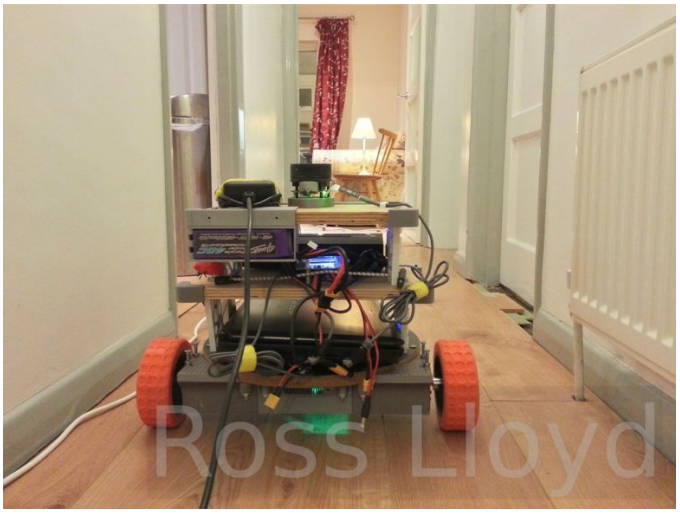

Fig. 54. Gbot mapping configuration from rear during mapping (hallway).

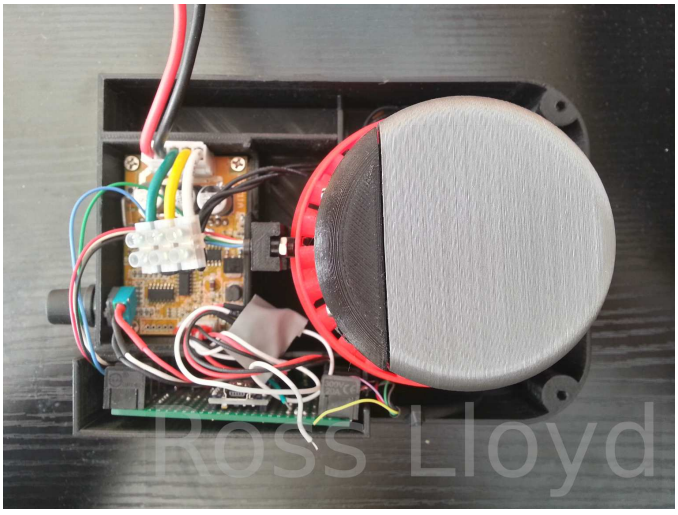

Fig. 55. Unused DIY lidar plan view.

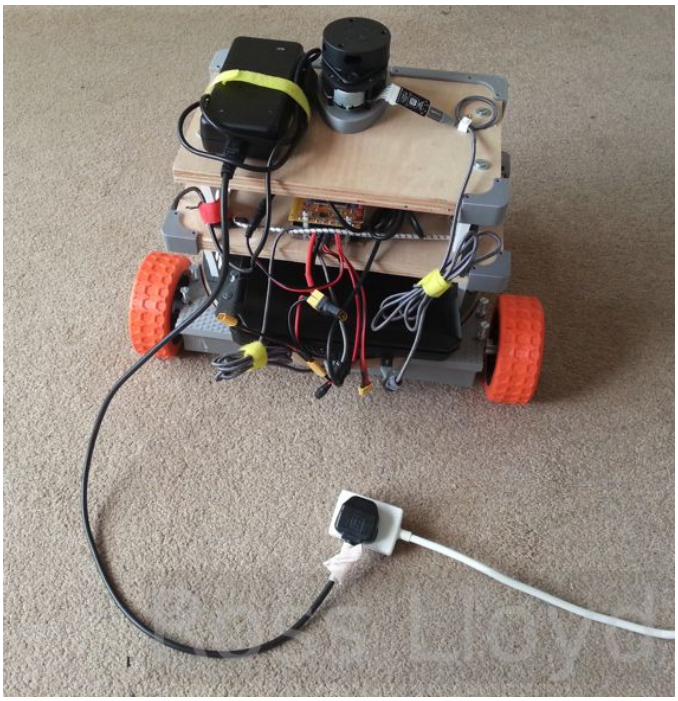

Fig. 56. Gbot power tether compromise during mapping.

Fig. 57. Unused DIY lidar front view.

Fig. 58. Unused DIY lidar profile view.

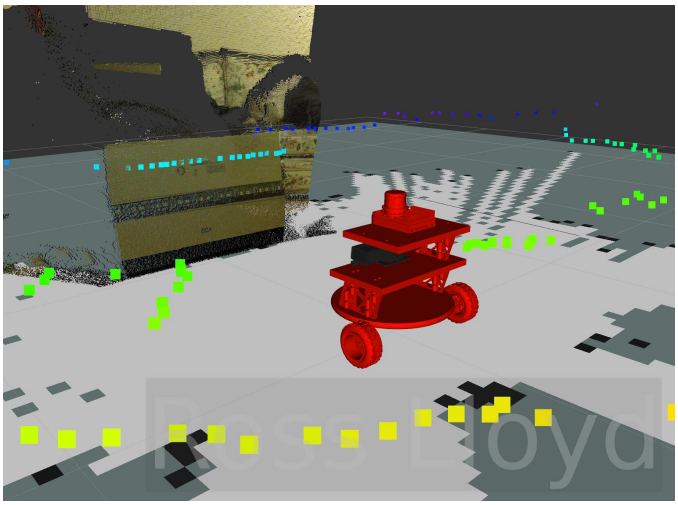

Fig. 59. Unused DIY lidar acquiring data. Unfortunately the data was insufficiently accurate for use with this project.

3.3.2 Packages Used

For the real robot, a number of additional packages were needed to allow for motion control and driving sensors. These were built on the robot (laptop or TX2) and their data sent via wifi to the client computer, which runs rtabmap. The nodes on the robot are:

roboclaw_node. This is the community-created freeware driver recommended by basicmicro. It subscribes to the **/udacity_bot_teleop_joystick/cmd_vel** topic and publishes the **/roboclaw/odom** and **/tf** topics, as well as some status data.

kinect2_bridge is responsible for processing and publishing the rgbd data, in this case the quarter hd (qhd) cloud. The ros throttling node is also applied, as well as a node that syncs the rgb and depth clouds together into an rgbd_image topic, which ensures there are no problems with mismatched timestamps between the rgb and depth images.

- kinect2/qhd_/data_throttled_image/
- kinect2/qhd_/image_depth_rect/
- kinect2/qhd_/camera_info/

are combined into

- rtabmap/rgbd_image

by the rgbd_sync node. Rtabmap subscribes to the compressed transport type of the data.

rplidar_ros node - outputs the **/scan** topic, which is angle and range data from the 360 degree rplidar A1 2D planar laser scanner. The usual robot state publisher and joint state publisher nodes are instantiated, as well as a static transform publisher to reorient the data from the kinect2. The launch file also runs the robot description, accessing the urdf file.

The robot is controlled with a slightly modified keyboard teleop file that publishes on **/udacity_bot_teleop_joystick/cmd_vel** which solved an issue with using the node between other packages. **rtabmap** - RVIZ subscribes to and displays the following topics:

- /odom
- /rtabmap/mapData
- /scan
- /map
- /rtabmap/mapGraph

The overall package structure is based on common ROS robot package structures such as turtlebot, and follows those outlined in the ROS wiki.

3.3.3 URDF

Please see github document for full details of the urdf. Points to note are that the iai_kinect2 package is included, to support their implementation of the **kinect2_bridge** node. Note that the stl file used in the model is not the iai model, but one found via grabcad online and the iai package modified to suit its use. No **.gazebo** file is needed as this is a real robot and gazebo will not be used.

3.3.4 Meshes

As detailed in the hardware section, the robot is an original design made mostly in CAD (onshape) and then 3d printed, machined from aluminium or made from wood. The meshes in this folder are conversions from STEP to binary stl, which is what the urdf requires. The CAD model for the kinect 2 and the rplidar come from online repositories (rplidar and grabcad).

3.3.5 Parameters

Server: In order to run ROS over the network, it was necessary to add the following lines to the /bashrc file of the client machine

- export ROS_IP=192.168.1.94
- export ROS_MASTER_URI=http://192.168.1.99:11311

and on the onboard robot machine:

- export ROS_IP=192.168.1.99
- export ROS_MASTER_URI=http://192.168.1.99:11311

A **chrony** server runs on the client machine, which the onboard robot computer syncs with in order to ensure topics

are published and processed with matching timestamps. *RTABMap*: The settings of interest that were found to create good quality maps were as follows:

- **Loop Closure Detection**: Kp/DetectorStrategy - 0=SURF
- **Maximum visual words per image**: Kp/MaxFeatures - 1000
- **Number of extracted SURF features**: SURF/HessianThreshold - 700
- **Loop Closure Constraint**: Reg/Strategy - 1 (ICP)
- **Min visual inliers to accept loop closure**: Vis/MinInliers - 10
- **Factor affecting ICP correction** - Icp/MaxCorrespondenceDistance - 0.1

As the real environment possesses a greater density of features than simulated ones, the max words, SURF threshold and minimum inliers was increased over those used in the gazebo worlds.

4 Results

4.0.1 Kitchen World Results

The robot and configured package produced a good quality map of the kitchen world, as evidenced by the total of 46 global loop closures and good visual fidelity of the map. The extents of the mapped area are slightly limited by the height of the robot's RGBD sensor, which is not able to see onto horizontal areas such as the counter top or the cushion of the armchair. There is also some slight occlusion of features such as the stand of the table in the sitting room area. It was possible to achieve this map by taking two contra-rotating paths around each room, and adding in-place rotations with the robot.

Fig. 60. 3D map of kitchen world from above.

Fig. 61. Kitchen map view 2.

There is good reproduction of repetitive features such as the chair legs, and evidence of good loop closures in the form of well aligned wall edges.

Fig. 62. Kitchen map view 3.

Though the RGBD sensor is angled up, there is enough of the floor captured in frame to reproduce the floor pattern and the rug.

Fig. 63. Constraints view list showing number of loop closures. 46 global LC's are observed.

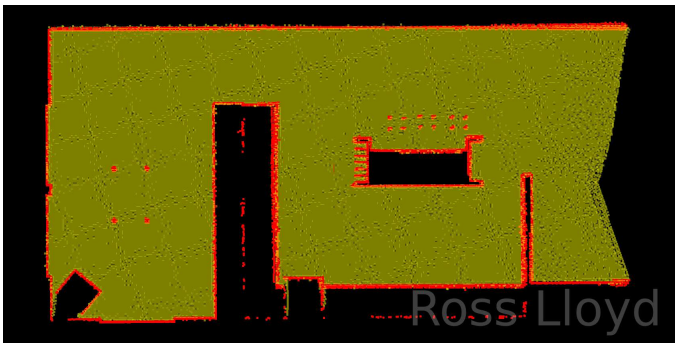

Fig. 64. Kitchen map occupancy grid.

The occupancy grid is accurate and well aligned, with well defined edges.

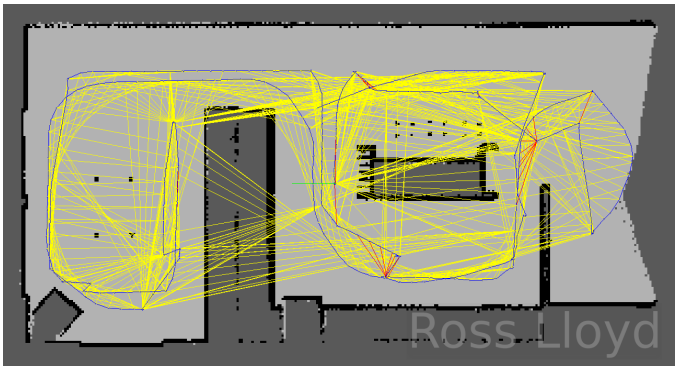

Fig. 65. Kitchen map constraints view.

This view shows the local and global loop closures that rtabmap made as the robot explored the environment. The global closures have occurred at the spots where in-place rotations took place. The floor plan is reproduced well with no noise in the main navigable areas.

Fig. 66. Kitchen map 3d loop closure example.

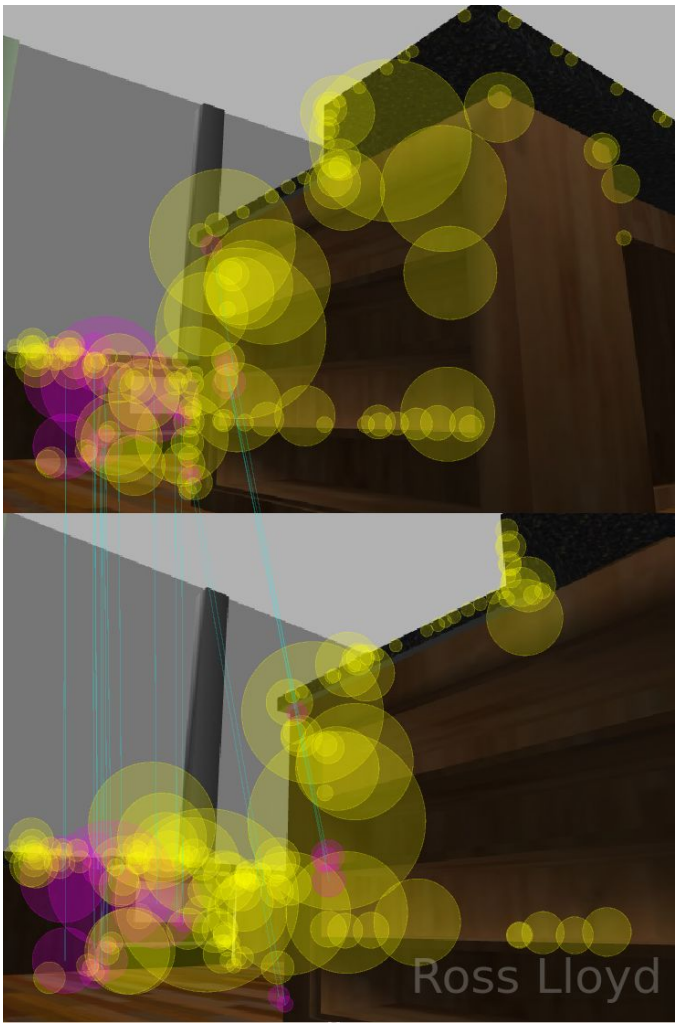

Fig. 67. Kitchen map 2d loop closure example.

In this image, the pink circles show where matching features have been identified by RTABMap, leading to the loop closure which can be observed in the top right of the constraints image. Before it is the RGBD image representation of this loop closure.

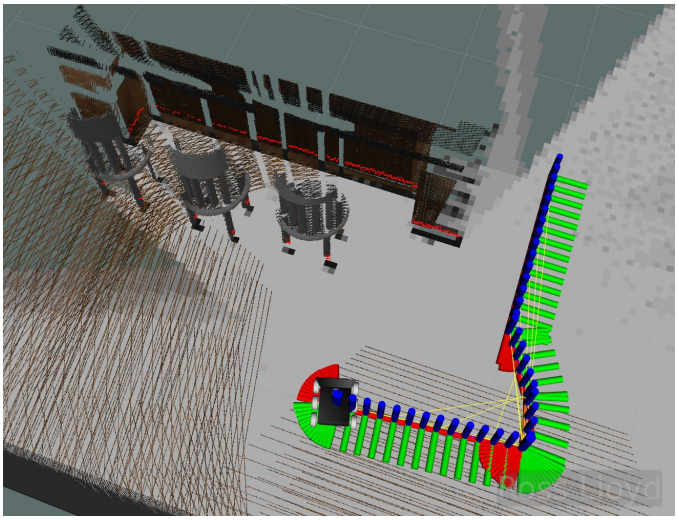

Fig. 68. Kitchen during mapping.

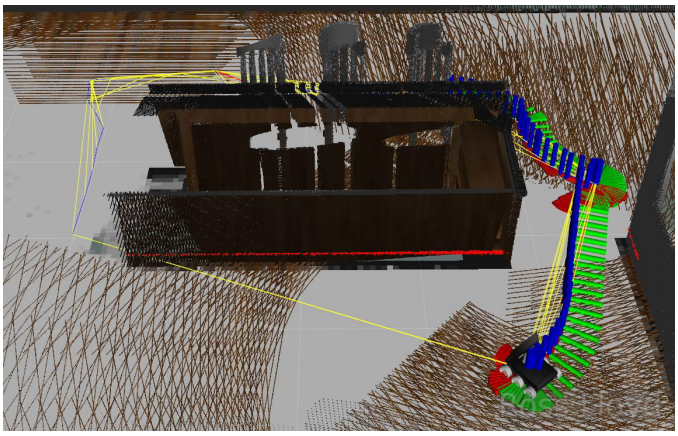

Fig. 69. Kitchen during mapping 2.

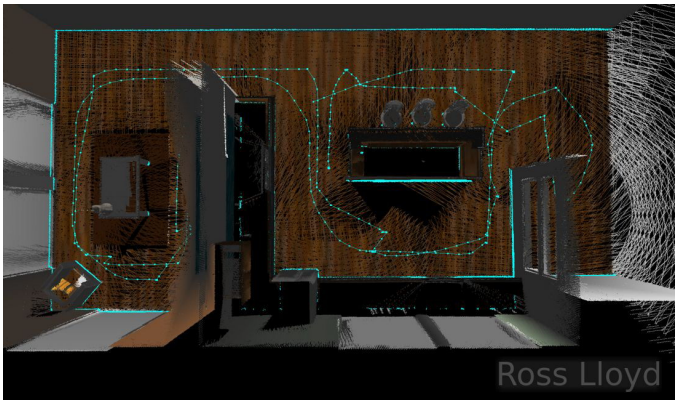

Fig. 70. Kitchen map showing odometry.

4.0.2 *Sixbot World Results*

Rtabmap produced an excellent quality map of the sixbot world. It was mapped with one or two passes per room and in-place rotations. 529 global loop closures were made.

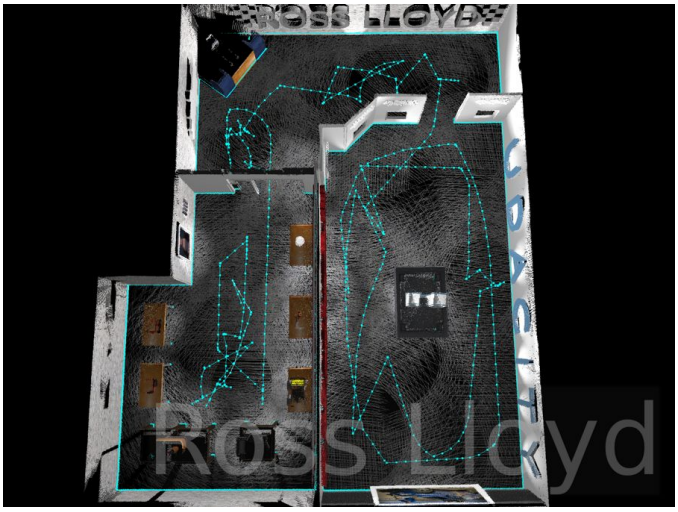

Fig. 71. 3D map of sixbot world from above, showing odometry.

The image above shows the relatively few passes needed and the accurate alignment of walls and features.

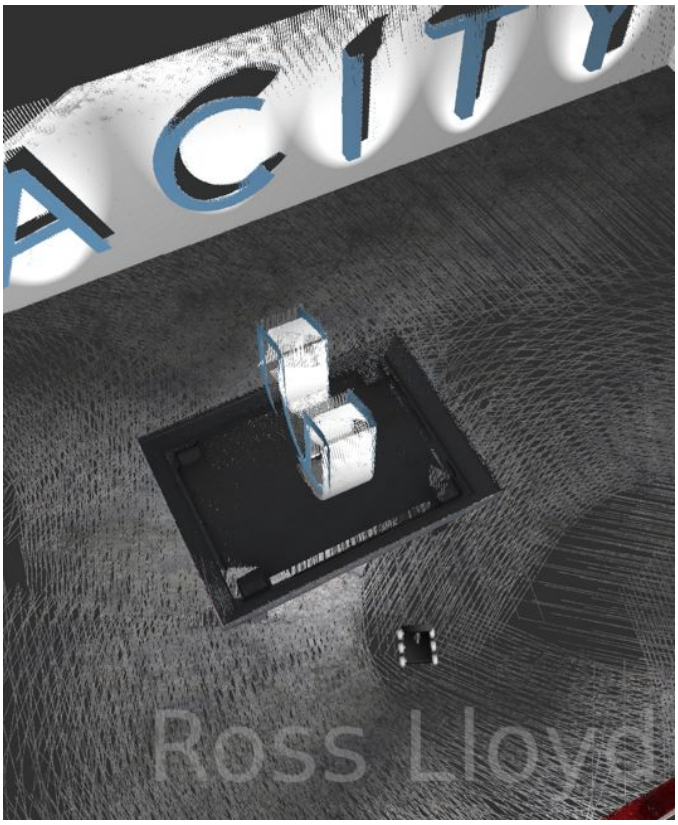

Fig. 72. Sixbot world map view detail - UDACITY signs.

It was difficult to capture all of the Udacity 'U' due to the height and angle of the rgbd sensor on the robot, which lead to some surfaces not rendering. The wall feature has rendered extremely well with a little shadowing due to the body of each letter blocking the virtual infra red rays and camera view.

Fig. 73. Sixbot world map detail - flat squares as pixel.

In the above picture, the hole-filling capacity of changing the pixel type to flat squares can be seen. When using the small point pixel, the wall behind can be seen. This does not affect the robot's ability to localize however as it works from the rgbd images.

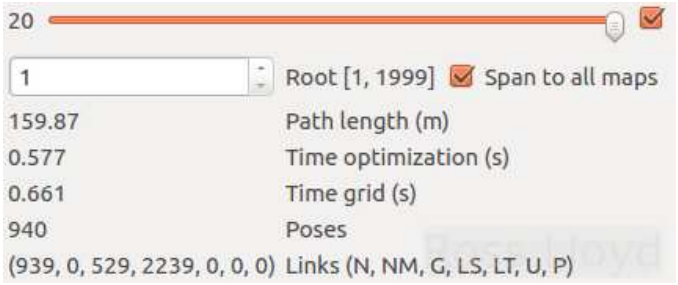

Fig. 74. Constraints view list showing number of loop closures. 529 global LC's are observed.

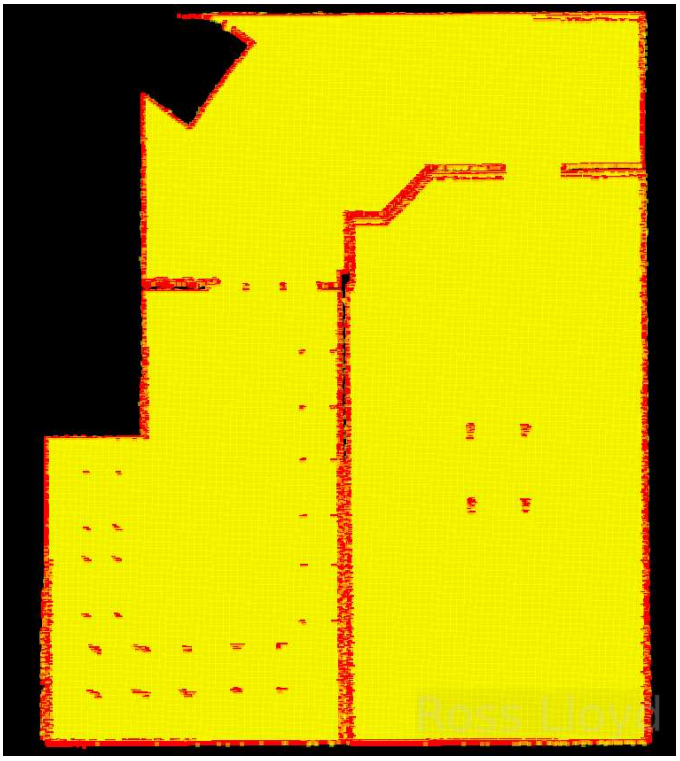

Fig. 75. Sixbot world occupancy grid.

The occupancy grid is neat and accurate with no noise in the unoccupied areas. Regions occluded by furniture are black. The table legs have been clearly captured and individuated.

Fig. 76. occlusion of detail due to obstacles.

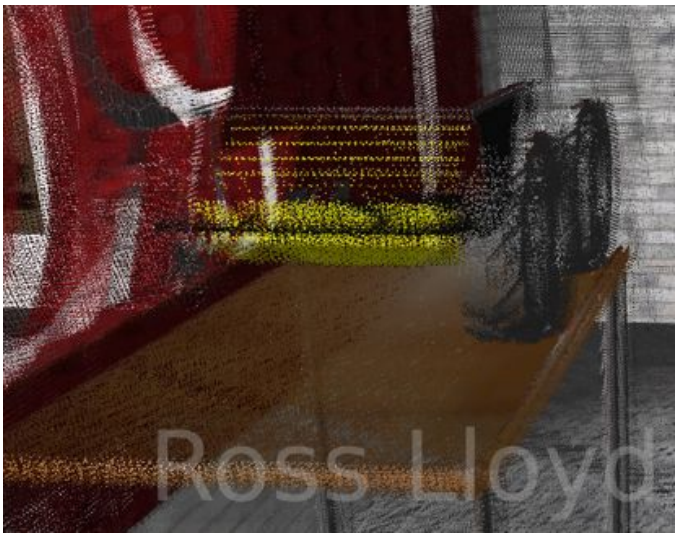

Fig. 77. Slight misalignment of robot detail.

The above shows a misalignment of a robot detail. This is likely due to a loop closure detection with the required number of inliers, but which was not perfectly aligned with the matching image and not caught by the optimizer.

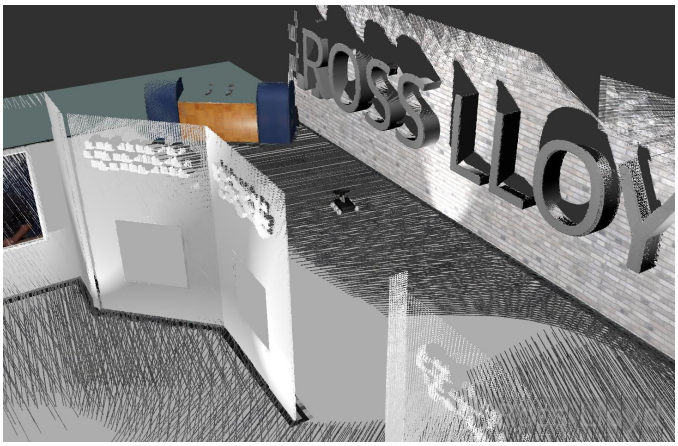

Fig. 78. Sixbot world during mapping 1.

The above picture shows the robot mapping room 2. The 2d map is also switched on and is shown clearly conforming to the wall layout. There is good reproduction of the 'Ross Lloyd' name plaque and the small wall features inside room1. Due to the upward angle of the rgbd sensor, the floor covers in a more sparse way. This is acceptable as it is less feature rich than other areas.

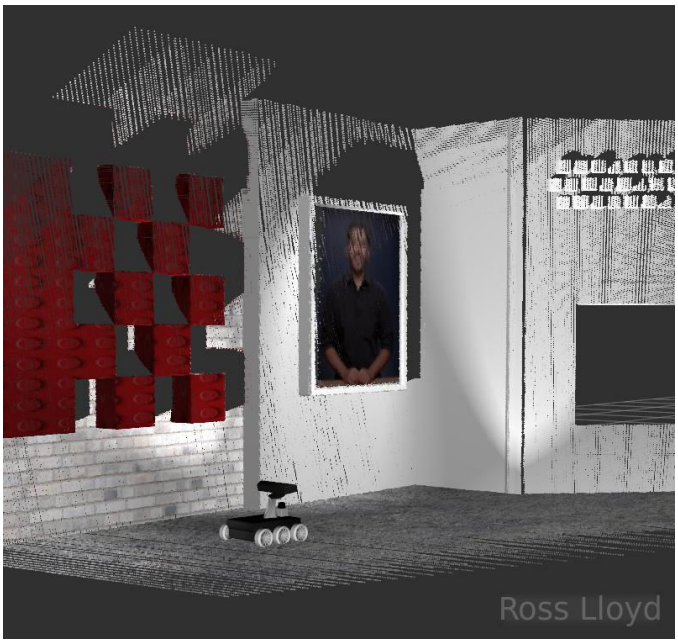

Fig. 79. Sixbot world during mapping 2.

Good reproduction of the wall pictures and 'robotics' plaque.

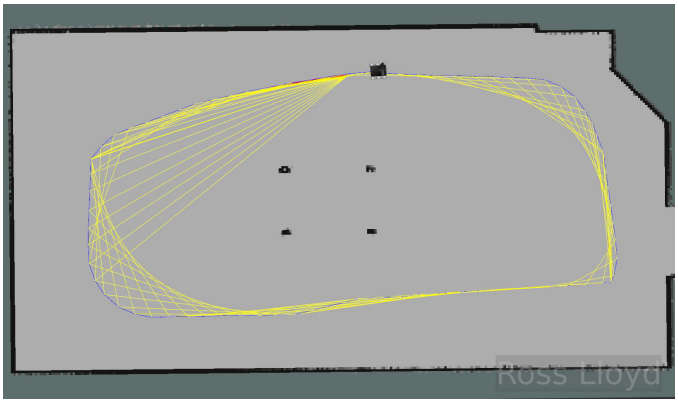

Fig. 80. 2D map being constructed during mapping.

The above shows a number of local loop closures and a global closure illustrated by the displayed constraints.

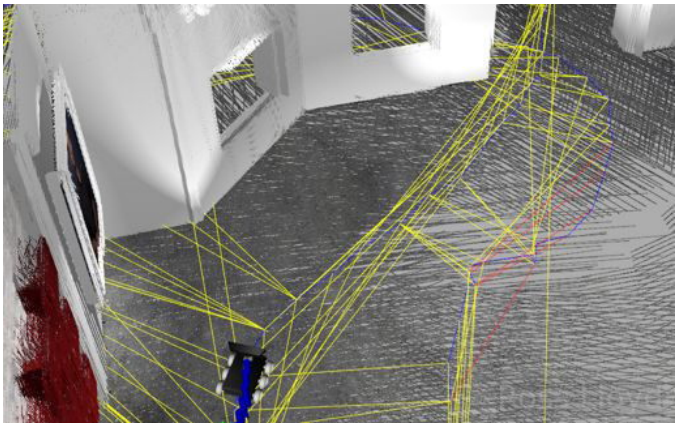

Fig. 81. Global Loop closure during mapping 1 (red constraint edge).

Above: More edge constraints depicting loop closures.

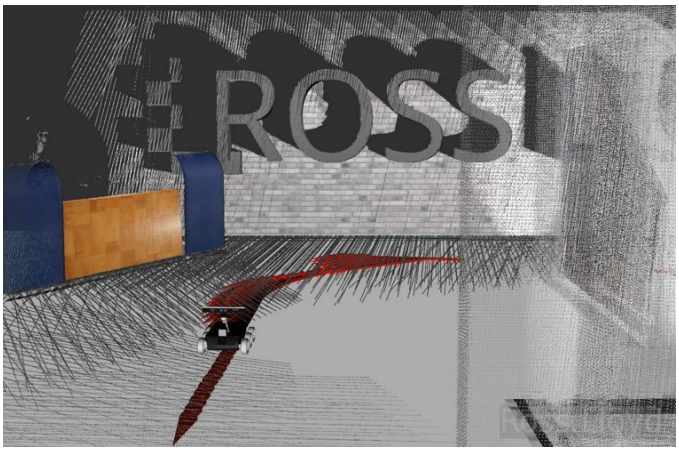

Fig. 82. Odometry display during mapping.

Above: Robot movement illustrated by displayed odometry (arrow head display)

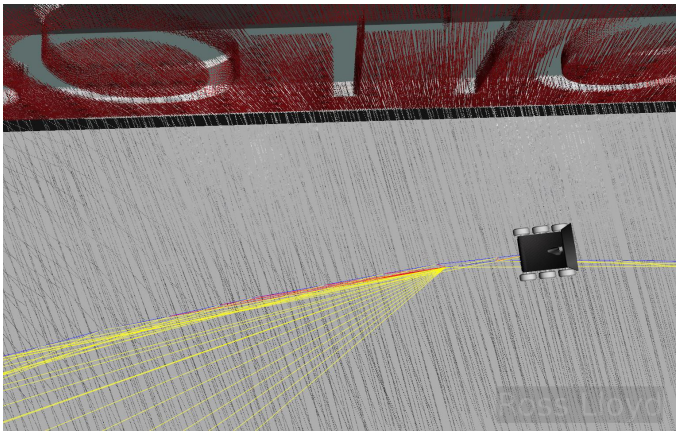

Fig. 83. During mapping, displaying graph view. Red shows a loop closure.

Above: More edge constraints depicting loop closures. Reproduction of the robotics sign is showing good accuracy.

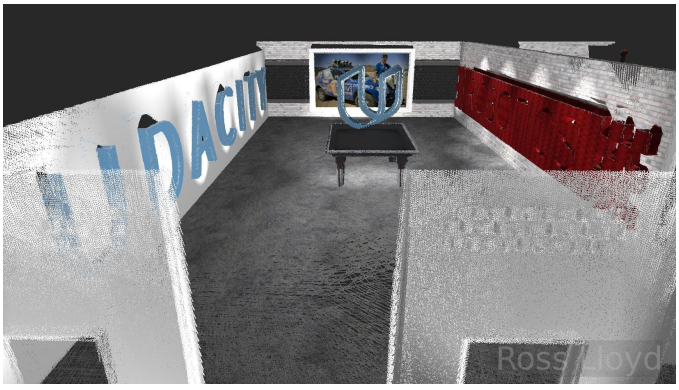

Fig. 84. View of Room 1 map.

Above: Picture of room 1 following mapping.

Fig. 85. Mapping result of ”robotics” sign.

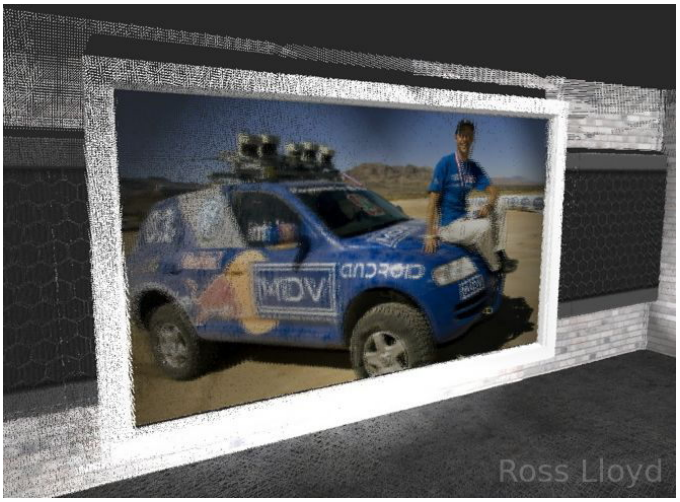

Fig. 86. Mapping result of Stanley picture.

Above: ’Stanley’ picture shows some slight blurring but is otherwise a good reproduction. The area around the picture is also well represented.

Fig. 87. Mapping result of room 2.

Above: Mapping result of room 2. The ross lloyd wall plaque shows good reproduction, though some of the wall tiles on the rightappear to have been occluded.

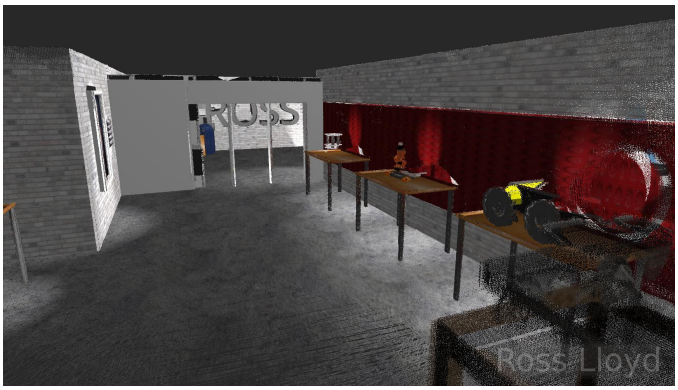

Fig. 88. Mapping result of room 3.

Above: The robots have not been fully reproduced due to the height and angle of the rgbd sensor Below: This principle is better illustrated by viewing from behind the sensor, where no gaps are present.

Fig. 89. Third person view from behind camera.

Fig. 90. Mapping result of robots in room 3.

Fig. 91. Mapping result of objects in room 3.

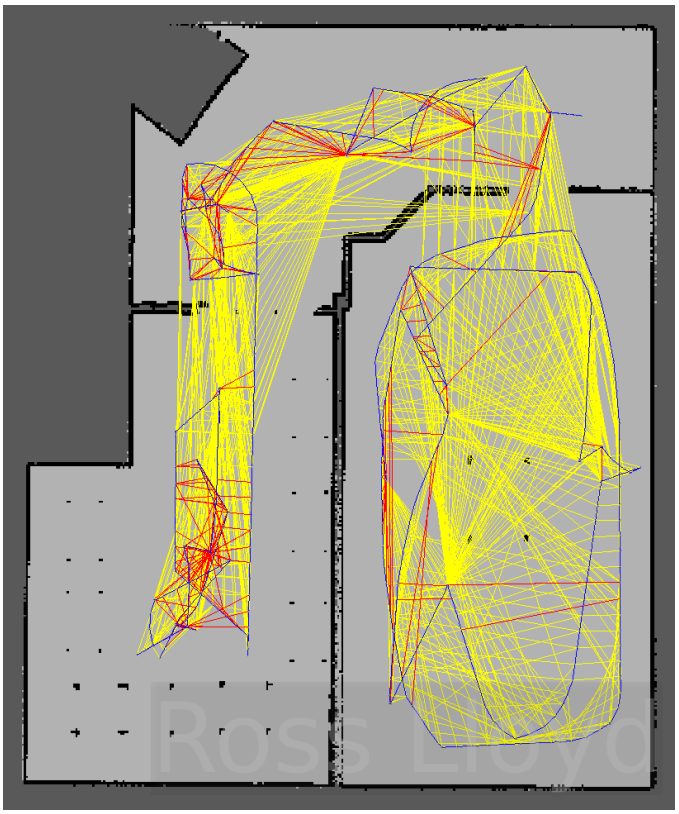

Fig. 93. Map Graph View.

Fig. 92. Loop Closure being added.

4.0.3 *Real World Mapping Results*

Reasonable results were obtained for the real robot, though the quality was not as high as for the virtual worlds (see discussion for reasons). It was noted that greater inaccuracy was introduced when using depth clouds at distances greater than 2m despite the cemra being calibrated out to this distance. For this reason the final maps were generated with a depth cloud range of 0.6m minimum to 1.8m maximum. This lead to greatly improved accuracy and less subtly repeated or ”smeared” images, albeit with higher elevated areas of walls and the ceilings not being visible in the generated map (these can of course be reintroduced by changing the parameter in *rtabmap databaseviewer*). Greatest areas of difficulty appeared to be chair legs, which showed the most tendency to smear / duplicate. The hallway is of notably worse quality due to the inability of the robot to accumulate a large number of images that covered the areas towards the ceiling. This is because as the robot rotated in place (frequently an effective way of capturing a large amount of images and loop closures) it could only ”see” a metre or so up the walls. This was further impacted by the fairly short maximum depth cloud limit of 1.7m (see discussion). Some vertical misalignment was also observed where the robot moved over room thresholds. A lot of noise is observed in the laser scans and occupancy grid maps, which is due to inaccuracy in odometry. See discussion for possible fixes. Additionally the Kinect2 and RPlidar A1 had a tendency to perceive reflections from glass and metal surfaces which further added to the noise. Some areas were not accessible to the robot, which accounts for the incomplete maps in some areas. In localization tests, the robot

was able to localize itself after rotation or moving a short distance. It thus performs adequately in the ″kidnapped robot″ scenario.

Fig. 94. Constraints view showing 296 global loop closures.

Fig. 95. Mapping result: Bookcase.

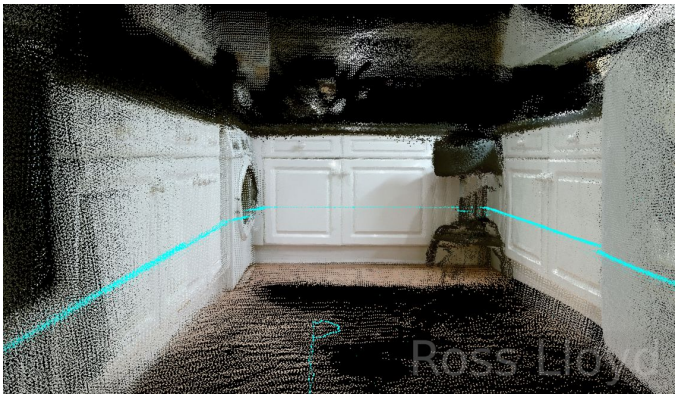

Fig. 96. Mapping result: Kitchen.

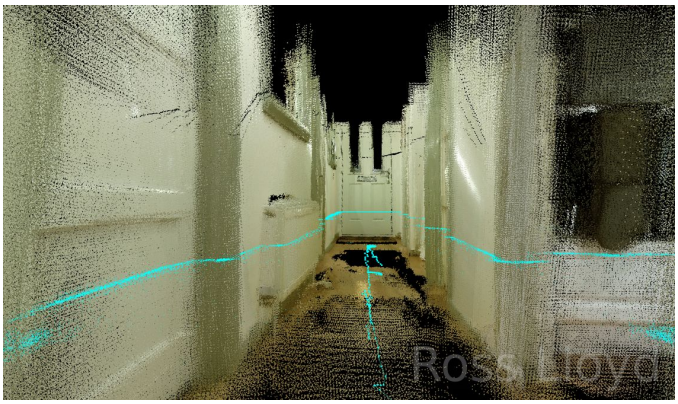

Fig. 97. Mapping result: Hallway.

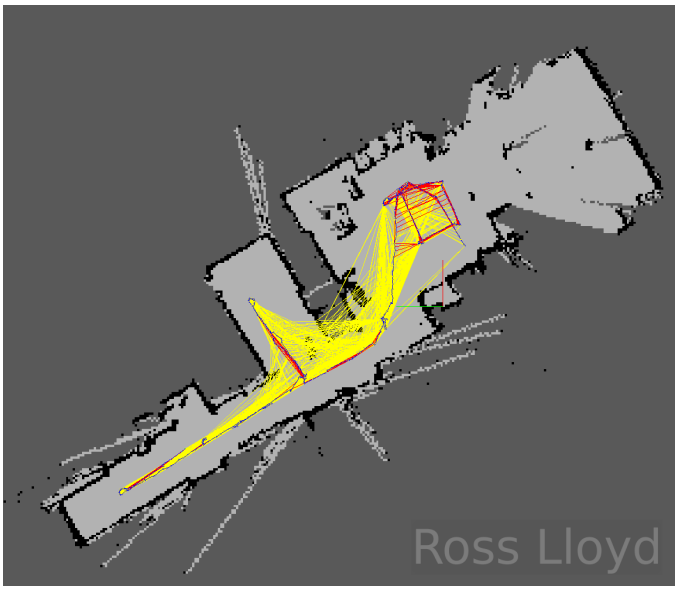

Fig. 98. Mapping result: Graph view.

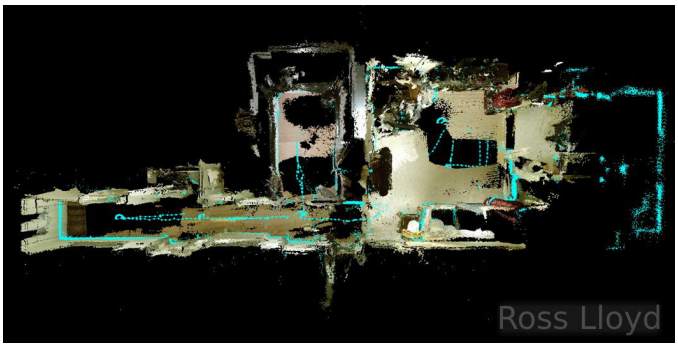

Fig. 99. Mapping result - odometry.

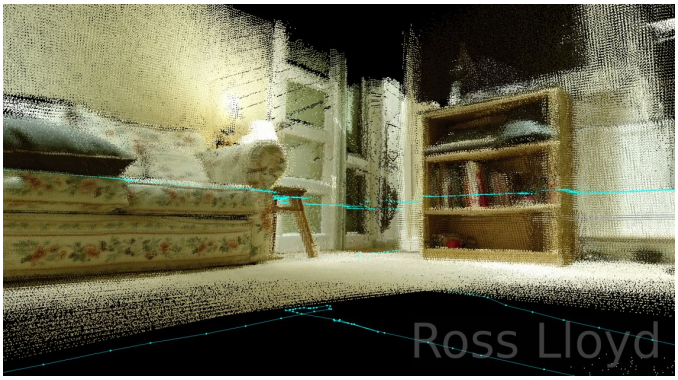

Fig. 100. Mapping result detail - sofa.

Fig. 101. Real world - during mapping 1.

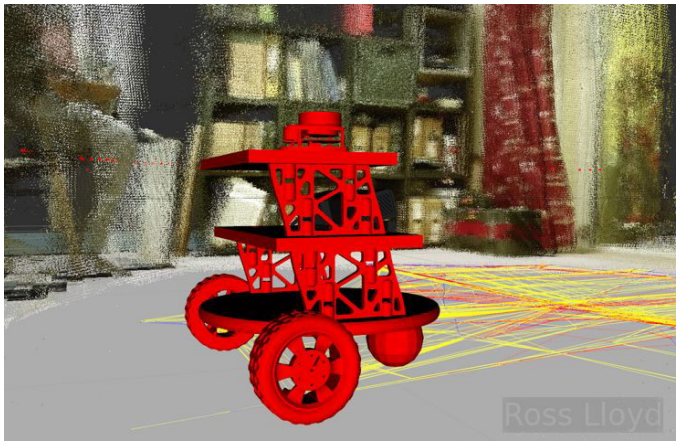

Fig. 102. Real world - during mapping 2.

Fig. 103. 2D Map of house areas covered by session.

Fig. 104. Occupancy grid areas covered by session. Note noise.

Fig. 105. Mapping result - plan view.

5 Discussion

5.0.1 Gazebo Worlds

The mapping for the two gazebo worlds was very successful as was predicted, due to the ”perfect” nature of the environments. Floors are perfectly flat and model geometry is fairly simple. Light levels are consistent and there are no real world difficulties to contend with such as the RGB image becoming washed out or experiencing differing brightness levels. Timing issues between the robot and client are absent as everything runs in sim-time, unlike on the real robot where a time server must be used to ensure that the timestamps of subscribed topics match up. It is also possible to set sensor distances very high as they do not suffer from the real sensor’s drop off in accuracy over distance. This meant that the higher sections of the map could be captured by the up-angled rgbd sensor. For the kitchen world the database generated was much smaller, however processing speed was poor due to the complexity of geometry of the kitchen world. Though this did not impact mapping quality, the process was slow. The custom world did not have this problem, though the robot speed was limited to 0.4 due to the previously mentioned physics anomaly in gazebo. There were adequate features for SURF / SIFT to generate a useable bag-of-words and vocabulary. A higher sensor placement would help capturing some of the occluded items, though the trade off would be a more vulnerable and / or physically less balanced robot.

5.0.2 Real World Mapping

Mapping the real space was distinctly more challenging than mapping the virtual gazebo worlds. The effects could be seen in the quality of the map, where though loop closures occurred, the quality of the closure was sometimes

poor and a lot of image ”ghosting” could be seen. Changing inlier parameters for loop closure detection or type of feature extraction helped up to a point, but were not able to fix the issues completely. The problems were traced to inaccuracy in odometry, which showed large amounts of drift even under non-mapping conditions.

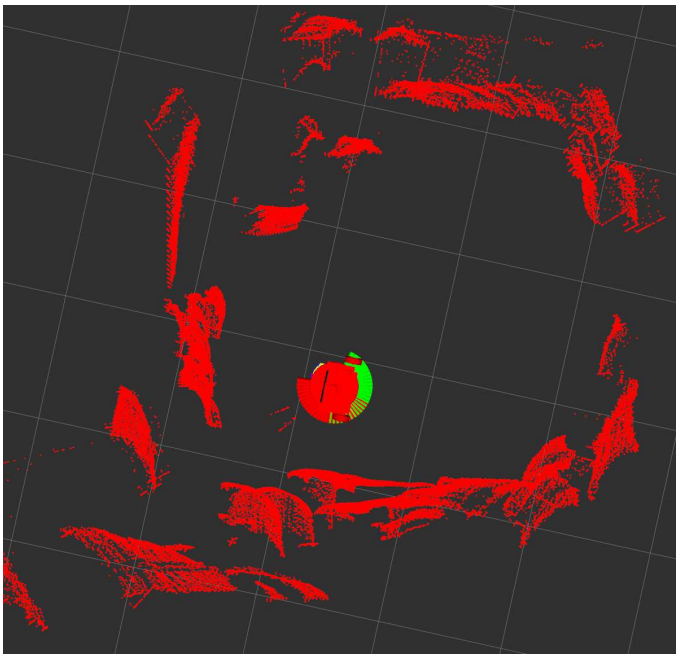

Fig. 106. Rotational odometry drift.

This could possibly be improved by experimenting with the parameters for the **roboclaw_node** which controls the motion control card and calculates odometry. It is also possible that due to the 3D printed design of the motor housing and hand drilled base of the robot that wheel tracking is not perfectly aligned. Though rtabmap is capable of dealing with some drift in odometry, the large amounts of drift illustrated in the laserscans likely negatively influenced the accuracy of the map. Experimenting with using visual odometry only could be an additional experiment to be carried out at another time.

It was also found that the TX2 was not capable of returning the kinect2 depth clouds at a published rate adequate to form good maps, impacted further by the inability of the TX2 to use the OpenCL performance improvement algorithms used in the **kinect2_bridge** node. Ultimately a GPU equipped laptop was used instead which utilised CUDA for the depth cloud processing and OpenCL for the depth registration. At this point the quality of the maps increased hugely as can be seen by comparing the image below with those from the real world mapping results images.

Improvements could be realised by adding an Inertial Measurement Unit to the robot, as the carpeted areas and edging presented subtle variations in pitch orientation that affected the image matching in some cases. RTABMap has a large scope of application and especially in virtual worlds can cope with highly varied environments. With both virtual and real environments, sensor limitations reduce the size of space that can be mapped, and RGBD sensors can struggle with outdoor lighting conditions. However this can be overcome using stereoscopic cameras, which are supported by rtabmap also. Though a newer sensor, it is possible that the infra-red system that the kinect 2 employs, as well as its higher resolution, may perform worse than the older kinect 1 which uses a structured light approach, and a lower resolution. This would have reduced the needed computational overheads and may have allowed use of the TX2. Though a kinect 1 sensor was available, it had been heavily modified to work with a PC and was not considered reliable enough. Sourcing an original kinect 1 for windows would solve this problem.

The appearance based element of RTABMap may fail in feature-poor environments and especially struggles in long hallways, unless a laser scanner and odometry is present. However the addition of these sensors makes the program extremely robust.

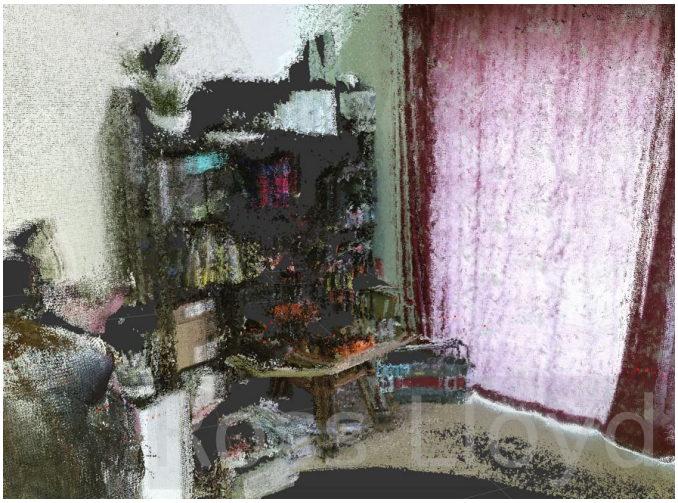

Fig. 107. Map quality using TX2.

6 Conclusion / Future work

RTABMap was demonstrated to be a robust SLAM solution that is effective in both simulated and real environments. It produced good quality maps that allowed subsequently for accurate and reasonably rapid. robot localization. For future work, the autonomous navigability of the maps generated by the program will be tested on the real robot. The addition of an IMU, as well as experimentation with other rgbd sensors, will be carried out in order to assess any improvements to mapping performance. SLAM algorithms have use in any space where a robotic solution is to be applied in a real environment, and as such could see use in rescue applications for mapping out disaster areas prior to entry by first responders. If combined with object identification and dynamic mapping, it could be applied to stock monitoring tasks in shops. Another application is in designing accessibility options for public buildings, where the robot can be used to map a space prior to route planning, as well as identifying immediately through use of wheeled robots where a wheelchair might struggle to gain access.

References

Probabilistic Robotics - Thrun