SLAM: Map My World

Shane Reynolds

March 20, 2019

Abstract

Suspendisse vitae elit. Aliquam arcu neque, ornare in, ullamcorper quis, commodo eu, libero. Fusce sagittis erat at erat tristique mollis. Maecenas sapien libero, molestie et, lobortis in, sodales eget, dui. Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros. Praesent malesuada, diam id pretium elementum, eros sem dictum tortor, vel consectetuer odio sem sed wisi.

1 Introduction

Consider a robot in an unknown environment, with no known map. The robot takes sensor readings and experiences control actions. Based on these observations and actions, the robot must construct a map and localise itself within the map. In robotics, this scenario is known as the *simultaneous localisation and mapping problem*, or SLAM. In simpler terms, using sensor readings and control data, SLAM concurrently constructs an environment map, and determines the robots location and orientation within that map. This is an important problem since odometric data is subject to small perturbations which are introduced from wheel slippage and sensor noise - often refered to as odometric drift. Mapping allows a robot to revisit previously mapped terrain and reset any localisation error. Also, location and orientation within a given map are normally used as inputs for higher order functions like path planning. The problem is more difficult to solve than localisation with known poses since high dimensionality of map spaces can often lead to computational intractabilityy. This paper explores two approaches to solving SLAM. The first of these solutions is called FastSLAM which employs a combination of Extended Kalman Filters(EKF) and Monte Carlo Localisation (MCL) to solve the problem. The second approach is called GraphSLAM, which solves the problem by optimising a graph structure built by the algorithm. The paper concludes with an application of GraphSLAM in a Gazebo simulation using an off-the-shelf implementation called RTAB-Map in ROS. The GraphSLAM implementation is tested across two different environments providing opportunities for discussion of the algorithm robustness.

2 Background

Robot localisation aims to determine, for some discrete time step t, a distribution of the robot's pose, x_t, given a series of observations, $z_{1:t}$, control actions, $u_{1:t}$, and a map, m. The localisation problem is often expressed, using conditional probability notation, as follows:

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

Mapping of an environment is the problem of determining a distribution over all possible map configurations, m, given a series of observations, $z_{1:t}$, and known robot poses, $x_{1:t}$. The mapping problem is often expressed, using conditional probability notation, as follows:

$$p(m|z_{1:t}, x_{1:t}) \tag{2}$$

Generally, a robot has neither the map, nor known poses, meaning that posteriors for both the map space and the robot's pose need to be determined. Equations (1) and (2) show these distributions are dependent on each other - evaluation of pose posterior requires the map, and evaluation of map posterior requires pose. This is often referred to as the chicken and egg problem. The main implication is that approaches designed to solve equations (1) and (2) cannot be readily applied in their current forms. Further, the problem can no longer be sufficiently expressed using these equations. SLAM, as the name suggests, determines these the map, m, and pose, x_t, simultaneously given sensor observations, $z_{1:t}$, and control actions, $u_{1:t}$. Mathematically, this is expressed as:

$$p(x_t, m|z_{1:t}, u_{1:t}) \tag{3}$$

Equation (3) is the equation for the *online* SLAM problem, which is only concerned with determining the robot's current location. Another variation of the SLAM problem, which is more difficult to solve, is the *full* SLAM problem. The *full* SLAM problem seeks to determine the complete pose history of the robot, $x_{1:t}$. This is often expressed as follows:

$$p(x_{1:t}, m|z_{1:t}, u_{1:t}) \tag{4}$$

Typically, *online* SLAM is used to dynamically localise the robot, whilst *full* SLAM is determine offline and is used to determine where the robot has been. It is not hard to see that the solution to the *online* SLAM problem can be determined by solving the *full* SLAM problem and integrating - the relationship between the two problems can be seen in equation (XXXX):

$$p(x_t, m|z_{1:t}, u_{1:t}) = \int_{x_{t-1}} \int_{x_{t-2}} \cdots \int_{x_1} p(x_{1:t}, m|z_{1:t}, u_{1:t}) dx_1 dx_2 \dots dx_{t-1} \tag{5}$$

One final aspect of the SLAM problem has to do with determining whether or not the robot has previously encountered a feature in the map. The recognition of a map feature is discretely modelled: either the robot has encountered the feature before, or it has not. The literature refers to these recognitions as *correspondences*, and correspondence i is denoted as c_i. Rewriting the *online* SLAM problem shown in equation (3) yields:

$$p(x_t, m, c_t \mid z_{1:t}, u_{1:t}) \tag{6}$$

Similarly, rewriting the *full* SLAM problem shown in equation (4) yields:

$$p(x_{1:t}, m, c_{1:t} \mid z_{1:t}, u_{1:t}) \tag{7}$$

Finally, the *online* and *full* SLAM problems, with correspondences, can be linked similarly to equation (XXXX):

$$p(x_t, m, c_t|z_{1:t}, u_{1:t}) = \int_{x_{t-1}} \int_{x_{t-2}} \cdots \int_{x_1} \sum_{c_1} \sum_{c_2} \cdots \sum_{c_{t-1}} p(x_{1:t}, m|z_{1:t}, u_{1:t}) dx_1 dx_2 \dots dx_{t-1} \tag{8}$$

This paper discusses two approaches for solving the SLAM problem. The first method is Grid Based Fast SLAM, introduced in subsection 2.1; the second method is Graph SLAM introduced in subsection 2.2.

2.1 Grid Based FastSLAM

FastSLAM was first proposed by XXXX. The central idea was an application of Rao-Blackwellized particle filter to estimate the joint posterior (reference to Murphy). To reduce the state space dimensionality, and hence the problem complexity, we will consider the problem with known correspondences, hence the joint posterior we are solving for is the one shown in equation (4). Using the simple Bayes' rule, $p(x_1, x_2) = p(x_2|x_1) \cdot p(x_1)$, equation (4) can be re-expressed as follows:

$$p(x_{1:t}, m|z_{1:t}, u_{1:t}) = p(x_{1:t}|z_{1:t}, u_{1:t}) \cdot p(m|x_{1:t}, z_{1:t}, u_{1:t}) \tag{9}$$

The key assumption made in Murphy's paper was that if the robot's path, $x_{1:t}$, is known then landmark locations are conditionally independent given the poses. This assumption allows us to factorise equation (5) as follows:

$$p(x_{1:t}, m|z_{1:t}, u_{1:t}) = p(x_{1:t}|z_{1:t}, u_{1:t}) \cdot \prod_i p(m_i|x_{1:t}, z_{1:t}, u_{1:t}) \tag{10}$$

Applying Bayes' theorem and then factorisation breaks the problem into two smaller problems: estimation of the robot path $p(x_{1:t}|z_{1:t}, u_{1:t})$; and estimation of the robot map $\prod_i p(m_i|x_{1:t}, z_{1:t}, u_{1:t})$. These smaller problems are known as the *localisation* problem, and *mapping* problem, respectively. Considering the problem in this light hints at an iterative approach to solving SLAM which consists of two steps: calculate the robot position; and then update the map based on that position. Grid based FastSLAM uses a particle filter to solve the localisation problem, such as Monte Carlo Localisation (MCL). The mapping problem is addressed using the *occupancy grid mapping* algorithm, discussed in XXXX. The full Grid Based Fast SLAM algorithm can be seen in section 2.1.2.

2.1.1 Occupancy Grid Mapping

A common technique for map representation is to decompose a continuous environment, m, into a discrete grid representation such that:

$$m = \sum_i m_i \tag{11}$$

Each cell, m_i, is either occupied or unoccupied. This representation, first introduced by XXXX, is referred to as an occupancy grid map.

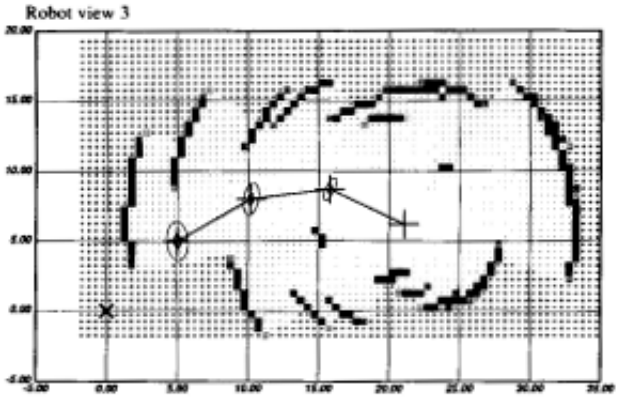

Figure 1: Occupancy grid map taken from XXXX. There are 3 types of dot in the image. Grey dots indicate regions that are yet to be explored; white dots indicate regions that are obstacle free; and black dots indicate obstacles in the environment.

As previously mentioned, the mapping problem requires estimation of the posterior $p(m|x_{1:t}, z_{1:t})$. The main problem with estimating this posterior using a discrete grid cell representation is the high dimensionality. An example of this can be seen by considering a map broken down into 500

discrete cells. If each cell is either occupied or unoccupied, then the total number of different maps that exist are 2^{500}. Estimating $p(m|x_{1:t}, z_{1:t})$ requires determining the probability for each of the 2^{500} maps. This is computationally intractable for a system which needs to operate in real time. Instead of attempting this, occupancy grid algorithms estimate $p(m_i|x_{1:t}, z_{1:t})$ for each grid cell m_i. Considering the occupancy of each grid cell as independent from other grid cells, whilst not strictly true, is a convenient assumption allowing the expression expression of the posterior $p(m|x_{1:t}, z_{1:t})$ as a product of it's marginals:

$$p(m|x_{1:t}, z_{1:t}) = \prod_i p(m_i|x_{1:t}, z_{1:t}) \tag{12}$$

We note that $p(m_i|x_{1:t}, z_{1:t})$ in equation (XXXX) is estimating a fixed binary quantity from a sequence of sensor measurements. According to XXXX beliefs of this type are commonly implemented as *log odds ratios*. We note that the *odds* of a state is defined as the ratio of the probability of the binary event divided by the probability of its negate. Using Bayes' theorem, we express $p(m_i|x_{1:t}, z_{1:t})$ as follows:

$$p(m_i|x_{1:t}, z_{1:t}) = \frac{p(z_t|m_i, z_{1:t-1}, x_{1:t}) \cdot p(m_i|z_{1:t}, x_{1:t})}{p(z_t|z_{1:t-1}, x_{1:t})} \tag{13}$$

Applying the Markov assumption for the first term in the numerator yields:

$$p(m_i|x_{1:t}, z_{1:t}) = \frac{p(z_t|m_i, x_t) \cdot p(m_i|z_{1:t}, x_{1:t})}{p(z_t|z_{1:t-1}, x_{1:t})} \tag{14}$$

We note, applying Bayes' theorem, that:

$$p(z_t|m_i, x_t) = \frac{p(m_i|z_t, x_t) \cdot p(z_t|x_t)}{p(m_i|x_t)} \tag{15}$$

Using equation (XXXX), we can re-express equation (XXXX) as:

$$p(m_i|x_{1:t}, z_{1:t}) = \frac{p(m_i|z_t, x_t) \cdot p(z_t|x_t) \cdot p(m_i|z_{1:t-1}, x_{1:t-1})}{p(m_i|x_t) \cdot p(z_t|z_{1:t-1}, x_{1:t})} \tag{16}$$

A further application of the Markov assumption allows us to express equation (XXXX) as:

$$p(m_i|x_{1:t}, z_{1:t}) = \frac{p(m_i|z_t, x_t) \cdot p(z_t|x_t) \cdot p(m_i|z_{1:t-1}, x_{1:t-1})}{p(m_i) \cdot p(z_t|z_{1:t-1}, x_{1:t})} \tag{17}$$

Similarly, we can follow the same process for the negate of m_i, $\neg m_i$. This yields the following expression:

$$p(\neg m_i|x_{1:t}, z_{1:t}) = \frac{p(\neg m_i|z_t, x_t) \cdot p(z_t|x_t) \cdot p(\neg m_i|z_{1:t-1}, x_{1:t-1})}{p(\neg m_i) \cdot p(z_t|z_{1:t-1}, x_{1:t})} \tag{18}$$

Hence, after some manipulation, the *odds* can be expressed as:

$$\frac{p(m_i|z_{1:t}, u_{1:t})}{p(\neg m_i|z_{1:t}, u_{1:t})} = \frac{p(m_i|z_t, x_t)}{1 - p(m_i|z_t, x_t)} \cdot \frac{p(m_i|z_{1:t-1}, x_{1:t-1})}{1 - p(m_i|z_{1:t-1}, x_{1:t-1})} \cdot \frac{1 - p(m_i)}{p(m_i)} \tag{19}$$

Taking the logarithm of this yields the the *log odds* calculation, l_t, for single grid cell:

$$l_t = \log\left(\frac{p(m_i|z_t, x_t)}{1 - p(m_i|z_t, x_t)}\right) + \log\left(\frac{p(m_i|z_{1:t-1}, x_{1:t-1})}{1 - p(m_i|z_{1:t-1}, x_{1:t-1})}\right) + \log\left(\frac{1 - p(m_i)}{p(m_i)}\right) \tag{20}$$

We note that equation (XXXX) can be expressed recursively since the middle term is just l_{t-1}. Further, the inverted final term is simply the inital belief l_0 Hence, we get:

$$l_t = l_{t-1} + \log\left(\frac{p(m_i|z_t, x_t)}{1 - p(m_i|z_t, x_t)}\right) - l_0 \tag{21}$$

The final term in equation (XXXX) is just the initial belief for the grid cell, and the second to last term is referred to as the *inverse sensor model*. The occupancy grid mapping algorithm takes the current pose x_t and current sensor measurements z_t, along with the array of log odds ratios calculated in a previous iteration for each grid cell. This array is denoted as $\{l_{t-1,i}\}$. The algorithm simply loops through all of the grid cells which comprise an environmental map, updating the log odds ratio for each cell. The pseudo code for the occupancy grid algorithm can be seen below.

Algorithm 1 Occupancy Grid Mapping

procedure OCCUPANCY_GRID_MAPPING($\{l_{t-1,i}\}$, x_t, z_t)
  for all cells m_i **do**
    if m_i in preceptual field of z_t **then**
      $l_{t,i} = l_{t-1,i} + \text{inverse\_sensor\_model}(m_i, x_t, z_t) - l_0$
    else
      $l_{t,i} = l_{t-1,i}$
  return $\{l_{t,i}\}$

2.1.2 Grid Based FastSLAM Algorithm

The grid based FastSLAM algorithm takes as input variable X_{t-1}, which holds the particles from a previous call to the algorithm. Each particle in X_{t-1} consists of a hypothesised robot pose and map. Additionally, odometry u_t, and sensor measurements z_t are also received as inputs. The algorithm commences by initialising empty variable $\hat{X}_t$ to store newly formed particle hypotheses. MCL is used to update the particle odometry from X_{t-1}, forming new pose hypotheses for each particle. Sensor measurements are used to determine the likelihood for each pose hypothesis. Using the updated pose hypotheses, the occupancy grid mapping algorithm is called, forming a map hypothesis for each particle. Once all of the existing particles from X_{t-1} have been updated, then a re-sampling procedure commences, which sees particles selected at random according to their likelihood of existing calculated earlier. The final set of re-sampled particles is returned and the algorithm is completed. These set of particles form the new aggregate hypothesis of the robot's map, and pose within the map - over many iterations of the algorithm, the hypothesis will converge on the true map and robot pose. The full Grid Based FastSLAM algorithm can be seen in Algorithm 2 below.

Algorithm 2 Grid Based FastSLAM

procedure GRIDBASED_FAST_SLAM(X_{t-1}, u_t, z_t)
  $\hat{X}_t = X_t = \emptyset$
  for $k = 1$ to M **do**
    $x_t^{[k]} = \text{sample\_motion\_model}(u_t, x_{t-1}^{[k-1]})$
    $\omega_t^{[k]} = \text{measurement\_model\_map}(z_t, x_t^{[k]}, m_{t-1}^{[k]})$
    $m_t^{[k]} = \text{updated\_occupancy\_grid}(z_t, x_t^{[k]}, m_{t-1}^{[k]})$
    $\hat{X}_t = \hat{X}_t \cup < x_t^{[k]}, m_t^{[k]}, \omega_t^{[k]} >$
  for $k = 1$ to M **do**
    draw i from $\hat{X}_t$ with probability $\propto \omega_t^{[i]}$
    $X_t = X_t \cup < x_t^{[i]}, m_t^{[i]} >$
  return X_t

2.2 GraphSLAM

GraphSLAM, in comparison to GBFS, takes a fundamentally different approach to solving the SLAM problem. The main idea behind GraphSLAM is to build up a graph with nodes derived from robot poses, x_t, and map features, m_t. The graph edges represent a series of constraints. The constraints are used to develop a likelihood cost function - minimising this cost function is analogous to determining pose and map features that maximise the likelihood of measurement observations. Optimisation is achieved using classical optimisation techniques, and is performed off-line once the graph has been constructed. Often is its helpful to think of GraphSLAM as having a front end and a back end, as shown in Figure XXXX. The front end, which receives raw sensor data as input, focuses on graph construction. The back end receives the constructed graph as input, optimises the graph, and outputs map and trajectory estimation.

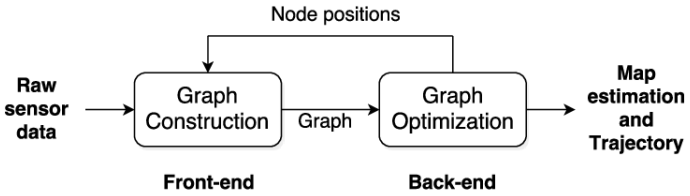

Figure 2: Flow diagram showing how GraphSLAM is decomposed into two distinct tasks. The front end receives sensor input and constructs a graph of pose nodes and map features. The back end optimises the graph to determine the robot trajectory and map posteriors.

2.2.1 GraphSLAM Front End

The front end of GraphSLAM is concerned with constructing the graph and determining the constraints. A robot nagivating some 2D topology typically has a pose comprised of position coordinates x, and y, as well as orientation θ. To make things simple, we will encapsulate these three parameters in a single pose vector $x_t = \begin{bmatrix} x & y & \theta \end{bmatrix}$. To simplify things further often a single parameter is used to represent state variables that contain one or more poses $x_{0:t}$ with the map m. Thrun and Montemerlo define $y_{0:t}$ to be a vector composed of the path $x_{0:t}$ and the map m, whereas y_t is simply composed as a single momentary pose in time, x_t, and the map m:

$$y_{0:t} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_t \\ m \end{bmatrix} \quad \text{and} \quad y_t = \begin{bmatrix} x_t \\ m \end{bmatrix} \tag{22}$$

As the robot moves around the environment, it experiences control actions $u_{1:t}$, takes sensor measurements $z_{1:t}$, and develops correspondences $c_{1:t}$. The posterior of full SLAM seen in equation (4) can be expressed using $y_{0:t}$, which yields the following:

$$p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) \tag{23}$$

Factorising equation XXXX using Bayes rule yields the following expression:

$$p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) = \frac{p(z_t \mid y_{0:t}, z_{1:t-1}, u_{1:t}, c_{1:t})\; p(y_{0:t} \mid z_{1:t-1}, u_{1:t}, c_{1:t})}{p(z_t \mid z_{1:t-1}, u_{1:t}, c_{1:t})} \tag{24}$$

Letting $\eta = p(z_t \mid z_{1:t-1}, u_{1:t}, c_{1:t})^{-1}$, we simplify equation (XXXX) as follows:

$$p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) = \eta\; p(z_t \mid y_{0:t}, z_{1:t-1}, u_{1:t}, c_{1:t})\; p(y_{0:t} \mid z_{1:t-1}, u_{1:t}, c_{1:t}) \tag{25}$$

The *Markov assumption* allows the simplification of the first probability in equation (XXXX):

$$p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) = \eta \; p(z_t \mid y_t, c_t) \; p(y_{0:t} \mid z_{1:t-1}, u_{1:t}, c_{1:t}) \tag{26}$$

Partitioning $y_{0:t}$ into x_t and $y_{0:t-1}$ the second probability term in equation (XXXX) can be expressed as:

$$p(y_{0:t} \mid z_{1:t-1}, u_{1:t}, c_{1:t}) = p(x_t \mid y_{0:t-1}, z_{1:t-1}, u_{1:t}, c_{1:t}) \; p(y_{0:t-1} \mid z_{1:t-1}, u_{1:t}, c_{1:t}) \tag{27}$$

THERE IS AN ADDITIONAL STEP HERE

Substituting equation (XXXX) back into equation (XXXX) yields a recursive expression for the full SLAM posterior:

$$p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) = \eta \; p(z_t \mid y_t, c_t) \; p(x_t \mid x_{t-1}, u_t) \; p(y_{0:t-1} \mid z_{1:t-1}, u_{1:t-1}, c_{1:t-1}) \tag{28}$$

Mathematical induction over t yields the closed form solution where $p(y_0)$ is the prior over the map m and the initial pose x_0. The closed form solution is:

$$p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) = \eta \; p(y_0) \; \prod_t p(x_t \mid x_{t-1}, u_t) p(z_t \mid y_t, c_t) \tag{29}$$

We note that probability $p(x_t \mid x_{t-1}, u_t)$ is the *motion model*, which gives us the probability of our current pose, x_t, given the previous pose and our odometry u_t. The other probability, $p(z_t \mid y_t, c_t)$, is the *sensor model* - this gives us the probability of receiving the sensor measurements given the current position, map, and correspondences. Equation (XXXX) is important because it allows us to think of the full SLAM posterior as a series of independent probabilistic events. Our task now becomes one of optimisation: how should we select $x_{0:t}$ and m so that we maximise the probability $p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t})$. Maximising a set of additive terms is much easier than maximising a set of multiplicative terms, and so we take a log transform which gives us:

$$\log p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) = \log \eta + \log p(y_0) + \sum_t \Big[\log p(x_t | x_{t-1}, u_t) \Big] + \sum_t \Big[\log p(z_t | y_t, c_t) \Big] \tag{30}$$

We assume that the robot motion is distributed normally according to $\mathcal{N}(g(u_t, x_{t-1}), R_t)$, where g is the deterministic motion function, and R_t is the covariance of the motion error. Similarly, measurements z_t are generated according to $\mathcal{N}(h(y_t, c_t), Q_t)$, where h is the measurement function and Q_t is the measurement error covariance. Multivariate normal pdf, where K_1 is used to represent the leading scalar coefficient, allows us to express the *motion model* as:

$$p(x_t | x_{t-1}, u_t) = K_1 \exp \Big\{ -\frac{1}{2} (x_t - g(u_t, x_{t-1}))^T R_t^{-1} (x_t - g(u_t, x_{t-1})) \Big\} \tag{31}$$

Similarly, we can express the *sensor model* as:

$$p(z_t | y_t, c_t) = K_2 \exp \Big\{ -\frac{1}{2} (z_t - h(y_t, c_t))^T Q_t^{-1} (z_t - h(y_t, c_t)) \Big\} \tag{32}$$

Noting that the prior $p(y_0)$ can be factorised into two independent priors $p(x_0)$ and $p(m)$, we typically just replace $p(y_0)$ with $p(x_0)$ since we often have no prior knowledge about the map - $p(m)$ is often subsumed by the normaliser η. With this in mind, we also express the prior $p(x_0)$ with a Gaussian type distribution:

$$p(x_0) = K_3 \exp \Big\{ -\frac{1}{2} x_0^T \Omega_0 x_0 \Big\} \tag{33}$$

with

$$\Omega_0 = \begin{bmatrix} \infty & 0 & 0 \\ 0 & \infty & 0 \\ 0 & 0 & \infty \end{bmatrix} \tag{34}$$

Substituting equations (XX), (XX), and (XX) into equation (XX) and putting constants $\log \eta$, $\log K_1$, $\log K_2$, and $\log K_3$ into C, we can express the negative of equation (XX) as:

$$\begin{aligned} -\log p(y_{0:t}|z_{1:t}, u_{1:t}, c_{1:t}) = C &+ \frac{1}{2} \cdot x_0^T \Omega_0 x_0 \\ &+ \frac{1}{2} \cdot \sum_t (x_t - g(u_t, x_{t-1}))^T R_t^{-1} (x_t - g(u_t, x_{t-1})) \\ &+ \frac{1}{2} \cdot \sum_t (z_t - h(y_t, c_t))^T Q_t^{-1} (z_t - h(y_t, c_t)) \end{aligned} \tag{35}$$

Determining the robot trajectory and map by minimising equation (XX) is the same as maximising equation (XX). In the literature, equation (XX) often has the scalar coefficients removed and is referred to as $J_{GraphSLAM}$. This is expressed as:

$$\begin{aligned} J_{GraphSLAM} = C &+ x_0^T \Omega_0 x_0 \\ &+ \sum_t (x_t - g(u_t, x_{t-1}))^T R_t^{-1} (x_t - g(u_t, x_{t-1})) \\ &+ \sum_t (z_t - h(y_t, c_t))^T Q_t^{-1} (z_t - h(y_t, c_t)) \end{aligned} \tag{36}$$

Another important way to think about equation (XX) is as the sum of constraints derived from the edges of a graph. As the robot moves around the environment, it forms pose nodes periodically. Additionally, it forms map feature nodes from sensor measurements. Edges between two corresponding pose nodes are referred to as motion constraints, and are derived from odometry sensor data - these are represented using notation u_t. Edges between poses and map features are referred to as measurement constraints, and are derived from exteroceptive sensors - these are represented using notation z_t. An example of a graph composed of pose nodes and map feature nodes can be seen in Figure XXXX. The robot moves from pose x_0 to pose x_1, and so on. As the robot moves, sensors detect map features m_1 and m_2. Odometric measurements are captured and used to create edges u_0, u_1, and u_2. Additionally, exteroceptive sensors capture the distance from the robot location to the map features resulting in dashed line edges z_0, z_1, z_2, and z_3.

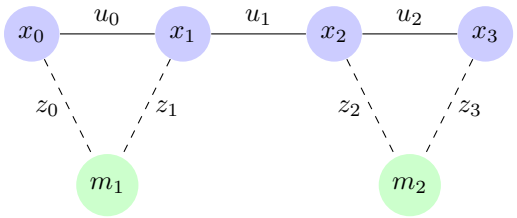

Figure 3: The robot moves from pose x_0 through to pose x_3, identifying map features m_1 and m_2 with exteroceptive sensors - these points are pose nodes and map feature nodes, respectively. Edges are formed between the graph from odometry, and sensor measurements. The edges in the graph are what form the constraints seen in GraphSLAM.

Once a set of constraints have been formed, $J_{GraphSLAM}$ can be found by simply summing the constraints together. Building a graph from pose and measurement nodes, and the subsequent constraint determination is the way that the front end of Graph SLAM is implemented. Once the graph has been determined, it is then passed to the back end for optimisation to determine the robot trajectory and map.

2.2.2 GraphSlam Back End

The back end of GraphSLAM is concerned with finding the robot trajectory, $x_{1:t}$, and map, m, which minimises the cost function, $J_{GraphSLAM}$, derived from the graph constraints obtained in the front end. Often the literature specifies robot trajectory and map using the single parameter, $\boldsymbol{\mu}$. Information from the constraints is expressed in an information matrix, $\boldsymbol{\Omega}$, and information vector, $\boldsymbol{\xi}$. The problem then simply becomes one of solving the following matrix equation:

$$\boldsymbol{\mu} = \boldsymbol{\Omega}^{-1} \cdot \boldsymbol{\xi} \tag{37}$$

Equation (XXXX) can be derived by computing the derivative with respect to each pose and map node variable. Setting each of the resulting expressions equal to zero yields the system of equations. Of course, the cost function can always by minimised by applying a numerical technique such as Stochastic Gradient Descent.

2.2.3 RTAB Map Impementation

RTAB-Map is a graph-based SLAM approach that has been integrated in ROS. The *rtabmap* node, seen in Figure XXXX, takes ROS *tf* input to determine the position of the sensors in relation to the robot base; Odometry from any source such as rotary encoders or IMU; camera input from either RGB-D or stereo cameras; and an optional input from either 2D lidar, point cloud data from 3D lidar.

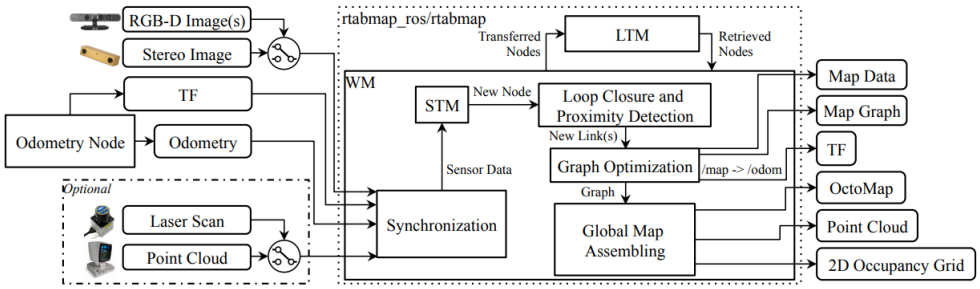

Figure 4: The rtabmap node in ROS receives camera image, and odometry inputs. Additionally, the node receives input from a 2D or 3D lidar. The rtabmap node synchronises these inputs, and creates nodes, and establishes links between these nodes. Upon the detection of a loop closure, rtabmap optimises the graph updating the map and providing correction to odometric data.

After sensor input has been synchronised, the Short-Term Memory (STM) module creates a node which remembers odometry pose, raw data from sensors, and additional information such as visual words for Loop Closure. Nodes are created every few millisecond at a predetermined frequency. As sequential nodes are created constraints are formed linking nodes together. Constraints come in three flavours: Neighbour, Loop Closure, and Proximity. Neighbour constraints are added in the STM between consecutive nodes with odometry transformation. Loop Closure and Proximity constraints are added through loop closure detection or proximity detection, respectively. Graph optimisation propagates computed error to the whole graph each time a new loop closure or proximity constraint is added to the graph - this helps to rectify odometry drift.

2.3 Comparison of Grid Based FastSLAM to GraphSLAM

Sed mattis, erat sit amet gravida malesuada, elit augue egestas diam, tempus scelerisque nunc nisl vitae libero. Sed consequat feugiat massa. Nunc porta, eros in eleifend varius, erat leo rutrum dui, non convallis lectus orci ut nibh. Sed lorem massa, nonummy quis, egestas id, condimentum at, nisl. Maecenas at nibh. Aliquam et augue at nunc pellentesque ullamcorper. Duis nisl nibh, laoreet suscipit, convallis ut, rutrum id, enim. Phasellus odio. Nulla nulla elit, molestie non, scelerisque at, vestibulum eu, nulla. Ut odio nisl, facilisis id, mollis et, scelerisque nec, enim. Aenean sem leo, pellentesque sit amet, scelerisque sit amet, vehicula pellentesque, sapien.

col1	col2
part1	part 2
part1	part 2
part1	part 2
part1	part 2
part1	part 2
part1	part 2

3 Scene and robot configuration

3.1 Robot Model

A robot model with simple geometries was selected for use in the SLAM simulation. The chassis is a rectangular prisim. Two wheels are located either side of the chassis centre forming a differential drive configuration - the robot model can be seen in Figure 5. An RGB-D camera, front mounted on the chassis, provides 3D point cloud data of the environment and is represented by a red cube. Finally, a laser range finder is mounted on top of the robot providing 2D planar depth information.

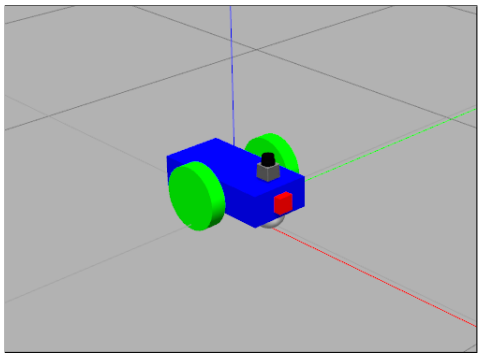

Figure 5: A basic robot model was selected to implement GraphSLAM on using RTAB-Map.

Table 1: A complete list of robot links, including parent links

Link	**Parent Link**
`robot_footprint`	`odom`
`chassis`	`robot_footprint`
`camera`	`chassis`
`hokuyo`	`chassis`
`left_wheel`	`chassis`
`right_wheel`	`chassis`
`camera_link_optical`	`camera`

A full list of the links used to construct the robot model in the `graph_slam_bot.xacro` xml file can be seen in Table 1. Pictorially, the 3D coordinate frame hierachy is shown in Figure 7 on page 12. This shows the coordinate frame attached to each link, in addition to the coordinate fram used for odometry. Note that arrows go from parent frames to child frames.

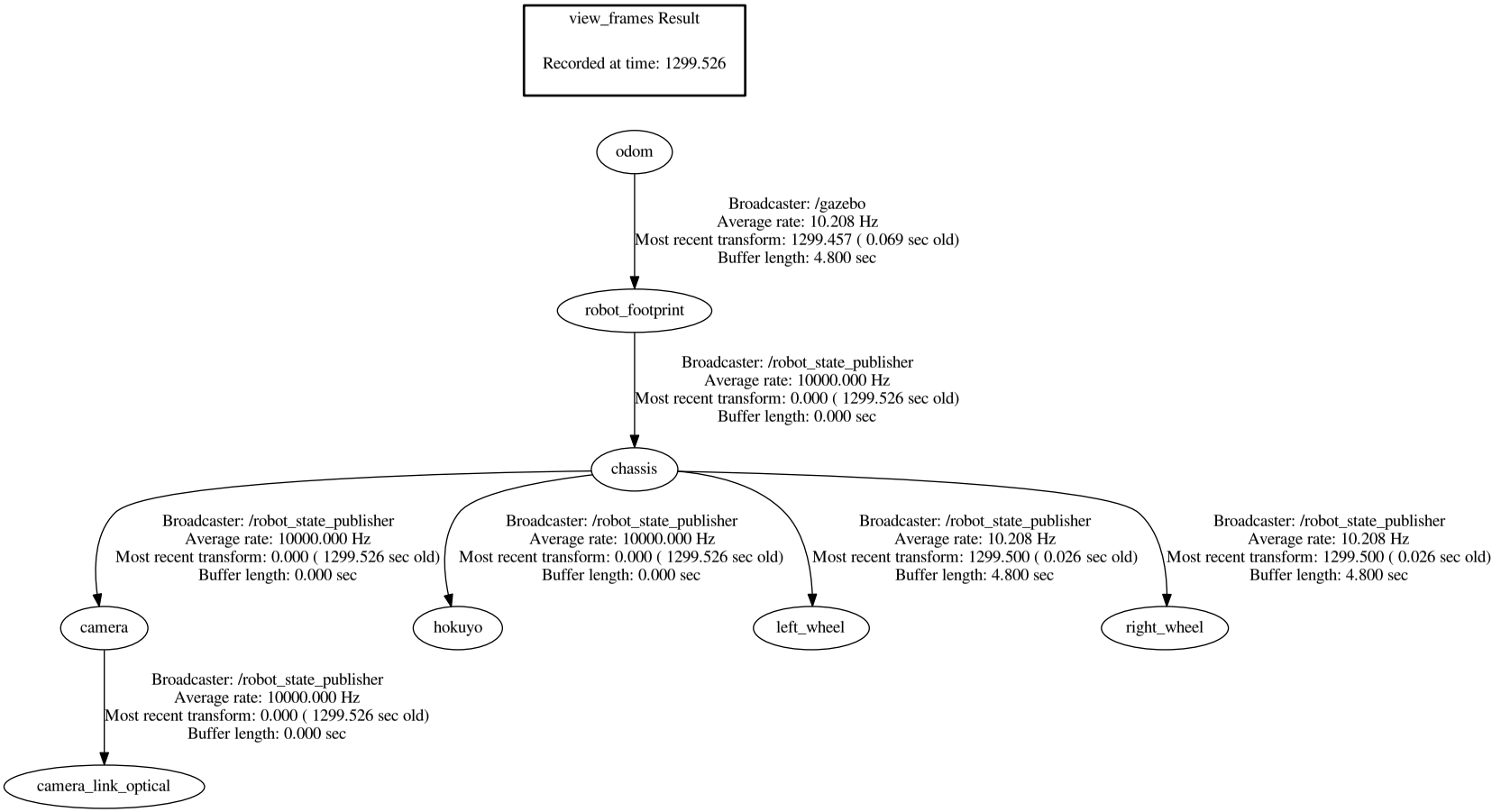

Figure 6: Graphical representation of the coordinate frame hierarchy used for the robot model. Correct specification of the coordinate frames is important since this allows the robot to correctly orient sensor data with respect to the robot chassis.

RTAB-Map is the package used to equip the robot model with GraphSLAM capabilities. The package uses RGB-D or stereo cameras and Lidar, meaning that there is some configuration of the RTAB-Map node to ensure that it is correctly subscribed to the correct topics published from the robot model sensors. A block diagram of the suggested configuration for robot models employing both RGB-D and Lidar can be seen in Figure 7. Notably, there are three topics for subsctiption from the RGB-D camera: RGB Image, Camera Info, and Image Depth. The Laser Scan and Odometry need to be subscribed from the published Lidar and rotary encoders topics, respectively. An RGB-D camera and 2D lidar set-up will see RTAB-Map outputting a 2D occupancy grid, in addition to a 3D map, called a cloud map, which is built using point cloud data. The occupancy grid and cloud map are visualised using Rviz.

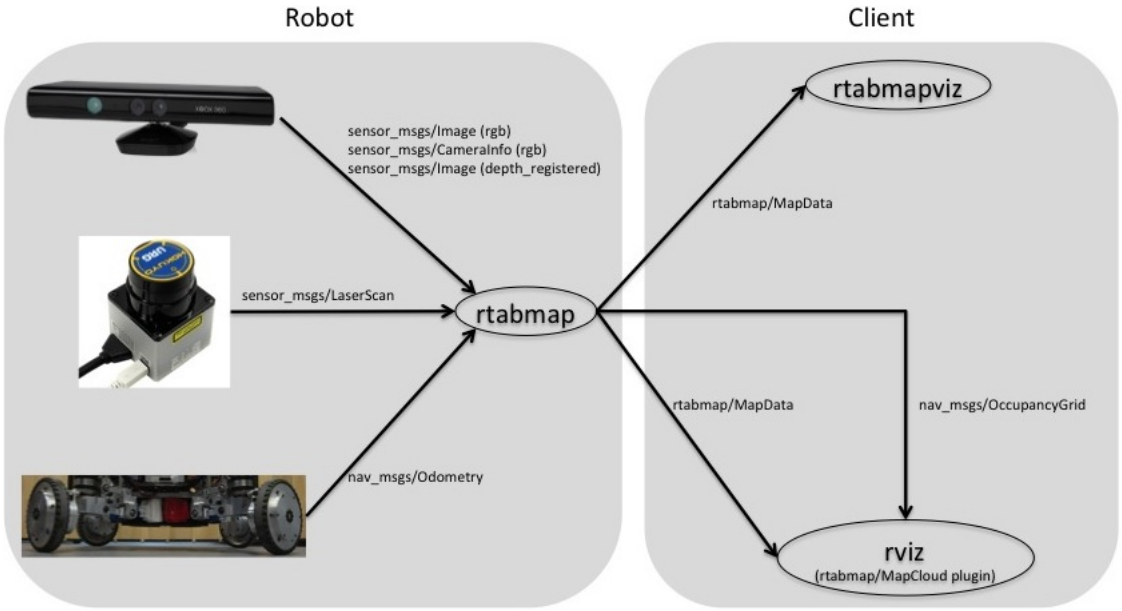

Figure 7: RTAB-Map requires correct subscription to published topics from the RGB-D camera and the robot odometry.

Table 2: Some of the important topics and parameters that need to be properly configured to ensure RTAB-Map will function correctly

Sensor Topic	**Topic Path**
RGBD Camera Image	`/camera_ir/rgb/image_raw`
RGBD Camera Depth	`/camera_ir/depth/image_raw`
RGBD Camera Info	`/camera_ir/rgb/camera_info`
Laser Scan	`/graph_slam_bot/laser/scan`
Odometry	`odom`
RTAB-MAP Parameter	**Value**
`Grid/FromDepth`	`false`
`subscribe_scan`	`true`
`subscribe_depth`	`true`

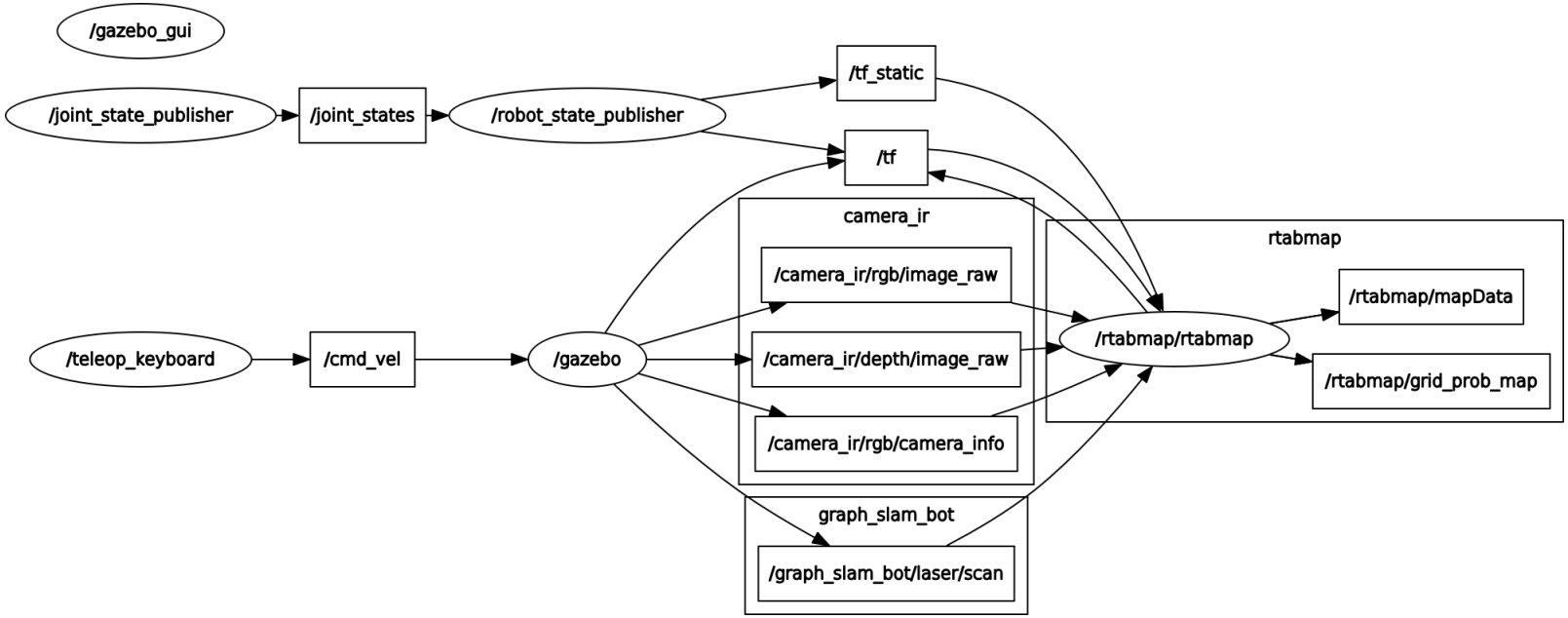

Figure 8: Graphical representation of the robot nodes and topics, with directional arrows showing publish and subscription relationships

3.2 Environment Model

3.2.1 Benchmark Environment

The benchmark model is of a small apartment mock up, featuring a kitchen and lounge room. The model is highly textured and contains many features - a good representation of real world environments. Rtab-Map should perform well in this environment because it is visually rich providing many opportunities for loop closure. The robot spawn point is in the kitchen. The benchmark environment model can be seen in Figure 9.

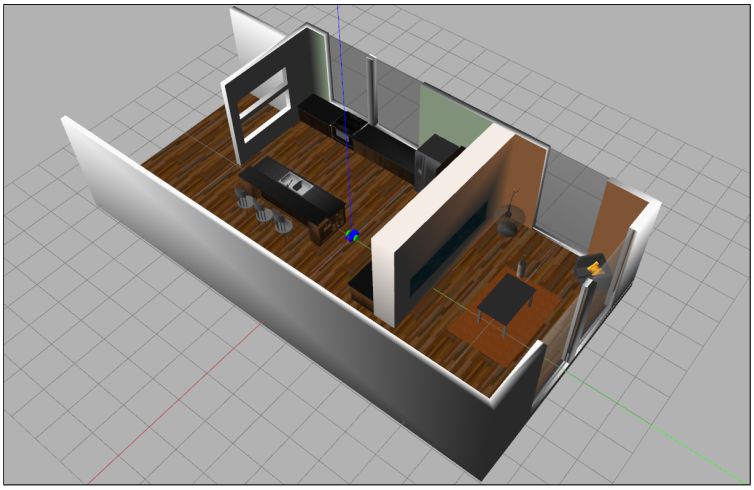

Figure 9: Benchmark environment model is highly textured and feature rich

3.2.2 Alternative Environment

The alternative model was designed to be very simple. The topology is a basic rectangular room. An important design feature is the variation of features in the environment: some areas have many features, and other areas are featureless. This design choice was made to help evaluate Rtab-Map's performance in sections of the environment that are visually bland, or that have repetitive textures. The environment model was developed using the building editor embedded in Gazebo. The alternative environment model can be seen in Figure 10.

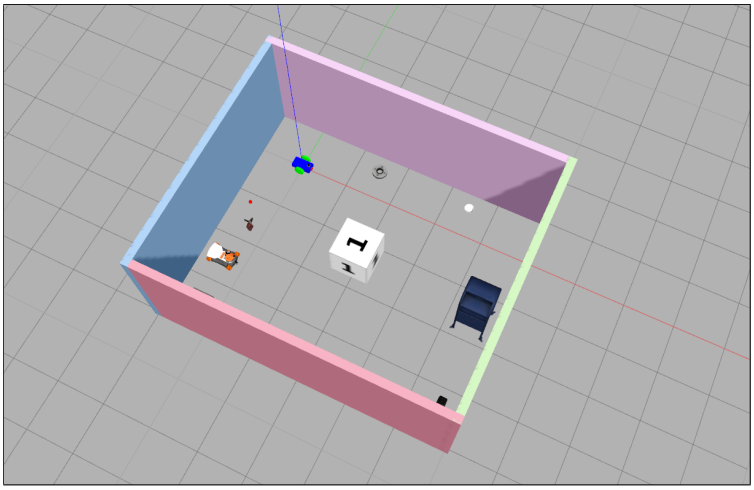

Figure 10: Alternative environment has been designed to contain bland, featureless areas

4 Results

4.1 Benchmark Environment

A teleop node was set up to manoeuvre the robot around the benchmark environment using keyboard input control. Approximately three laps of the kitchen, balcony, and lounge room areas were performed. Figure 11 shows a final iteration of the 2D occupancy grid map. The blue line represents the robot's node graph through the environment which is optimised by GraphSLAM in RTAB-Map. The red lines connecting nodes show loop closure activity. The red and green coordinate axes show the robot's starting location.

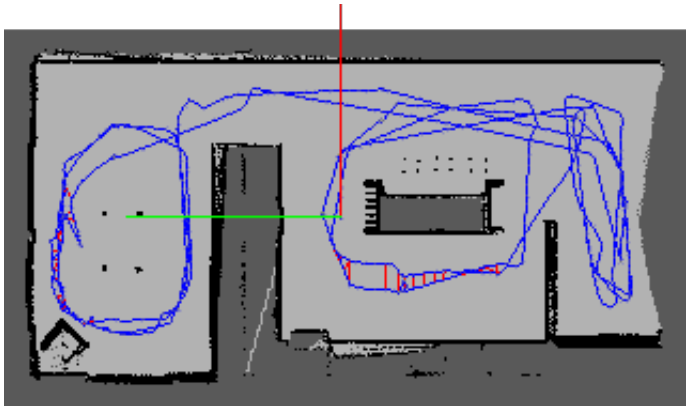

Figure 11: Final 2D occupancy grid map output from RTAB-Map for the benchmark model. The blue line represents links between nodes, and the red lines highlight loop closures.

As the robot traverses the environment RTAB-Map identifies features from images using the SURF detection algorithm which was specified in the mapping node configuration. Figure 12 shows a single image along the robot path. Yellow highlights identify features in the image that SURF has detected. As images are collected along the robot path, features in new images are compared to features from old images. The pink highlights represent instances where two images have features in common. These common features are used to create neighbouring links between images resulting in loop closures. Figure 13 shows the constraint view for the same point in time as Figure 12 - this image indicates where neighbouring links and loop closures were created.

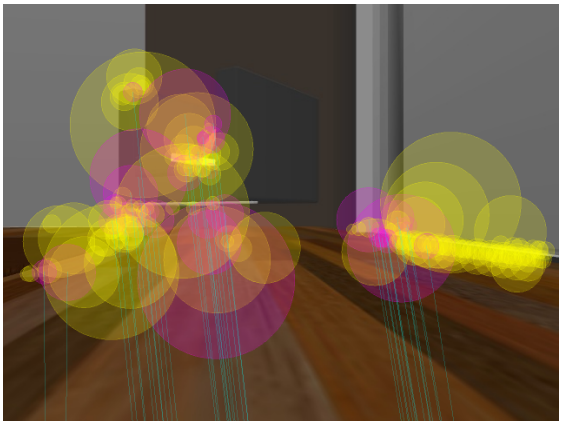

Figure 12: RTAB-Map operating in a feature rich area, as shown by the yellow highlights. The pink highlights represent loop closures.

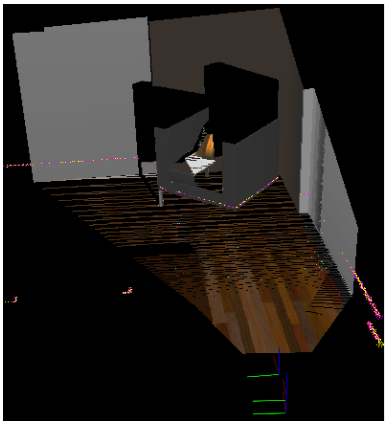

Figure 13: The constraints view of the same image shown in Figure 12. This also shows where neighbouring links and loop closures occur.

RTAB-Map was not able to successfully find features in all areas of the benchmark environment. In particular, there was a stretch of wall with repetitive textures where very few features were found. This was the hallway that connected the lounge, kitchen, and balcony areas. As a result of not being able to detect features, there was no loop closure activity in this region. An example of this can be seen in Figures 14 and 15.

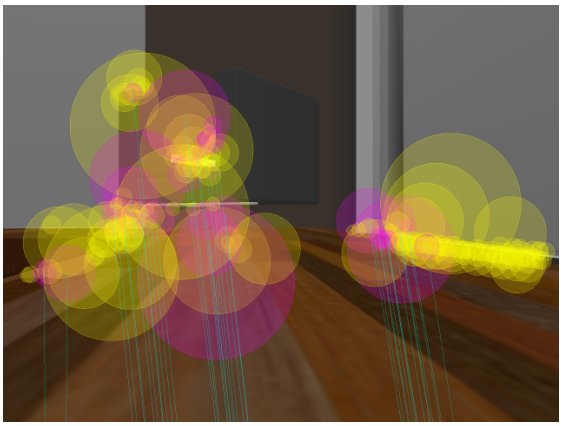

Figure 14: RTAB-Map was unable to detect features along the hallway due it containing bland and repetitive textures

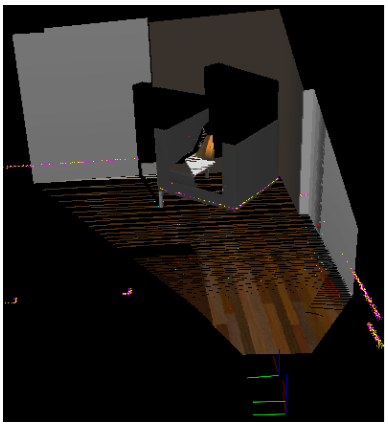

Figure 15: text

Upon completion of the three laps, RTAB-Map had recorded 755 neighbouring links, and 95 loop closures. Throughout this time the robot was able to successfully localise itself in the mapped environment, and make small corrections to the odometry as loop closures took place. Further, RTAB-Map successfully created a 3D map from point cloud data captured by the RGB-D camera. The 3D environment model was of reasonable fidelity, and well aligned with the 2D occupancy grid. Figure 16 shows the created 3D environment model from Rviz.

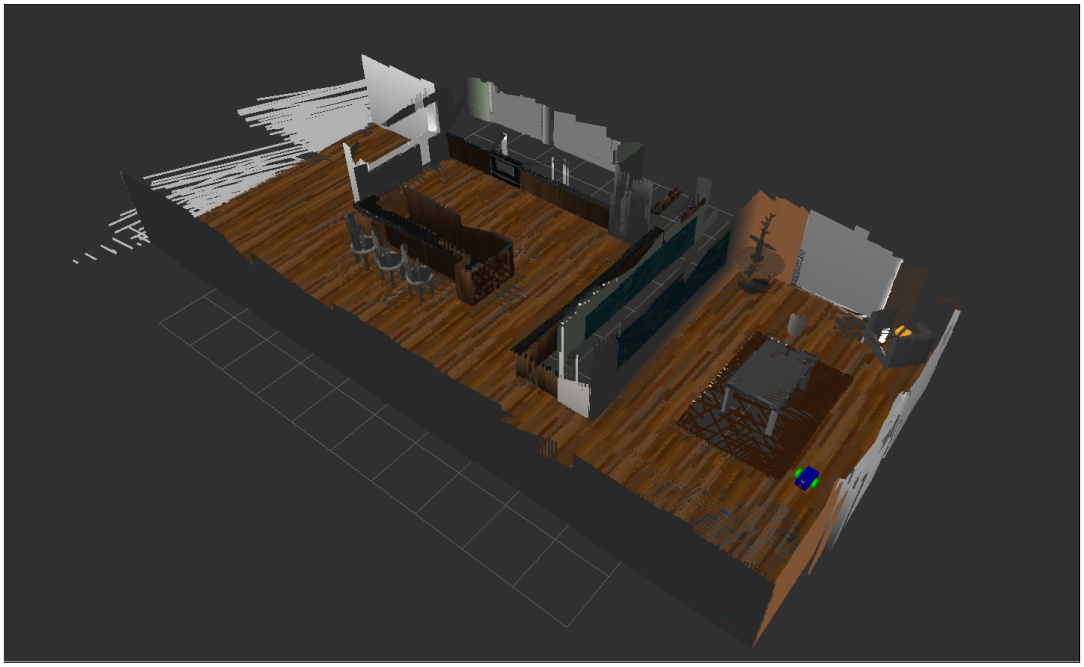

Figure 16: A 3D map of the benchmark environment created by RTAB-Map displayed in Rviz

4.2 Alternative Environment

As with the benchmark model, the robot was manoeuvred through the alternative environment using keyboard commands issued to a teleop node. Three laps of the basic environment were undertaken. The final iteration of the 2D occupancy grid map can be seen in Figure 17. The node graph used by GraphSLAM in RTAB-Map is represented by the blue line, and loop closure activity is shown in red.

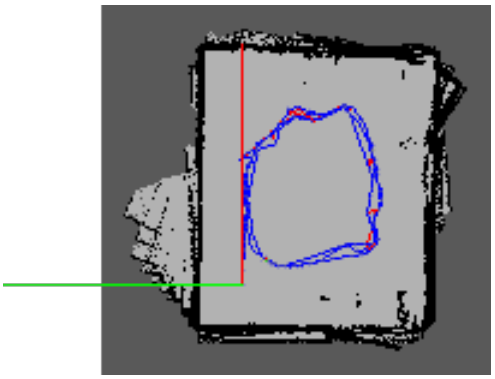

Figure 17: Final 2D occupancy grid map output from RTAB-Map for the alternative model. The blue line represents links between nodes, and the red lines highlight loop closures.

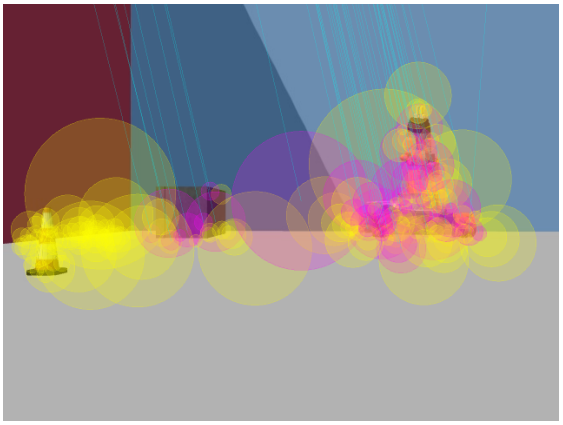

Figure 18: RTAB-Map operating in a feature rich area, as shown by the yellow highlights. The pink highlights represent loop closures.

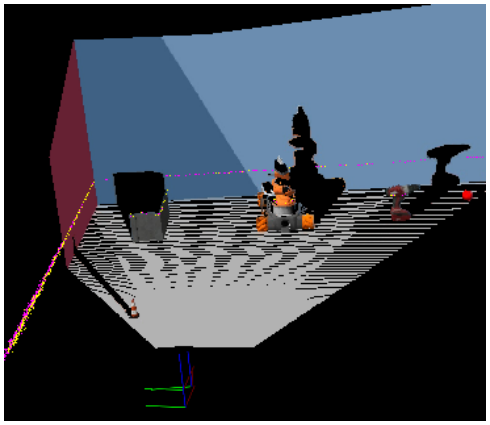

Figure 19: The constraints view of the same image shown in Figure 12. This also shows where neighbouring links and loop closures occur.

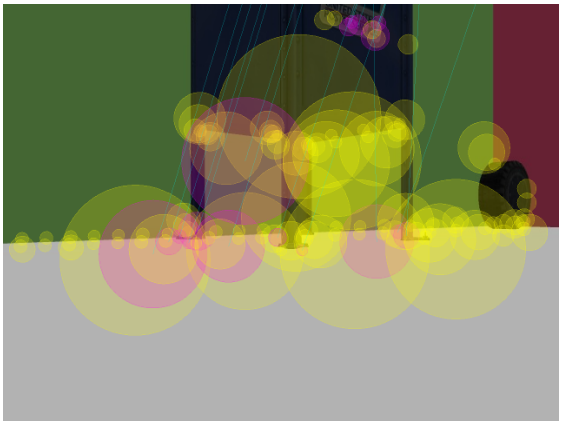

Figure 20: RTAB-Map operating in a feature rich area, as shown by the yellow highlights. The pink highlights represent loop closures.

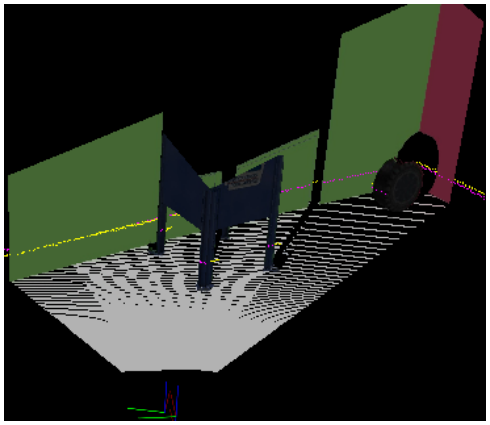

Figure 21: The constraints view of the same image shown in Figure 12. This also shows where neighbouring links and loop closures occur.

Figure 22: text

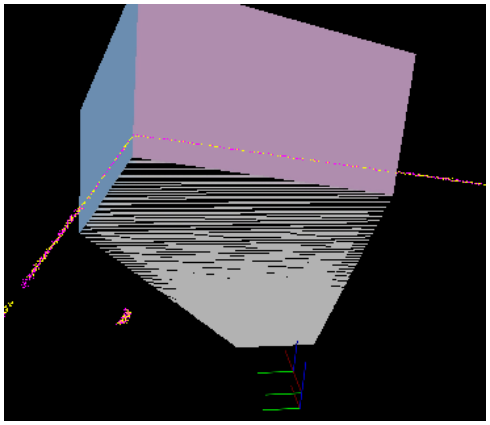

Figure 23: text

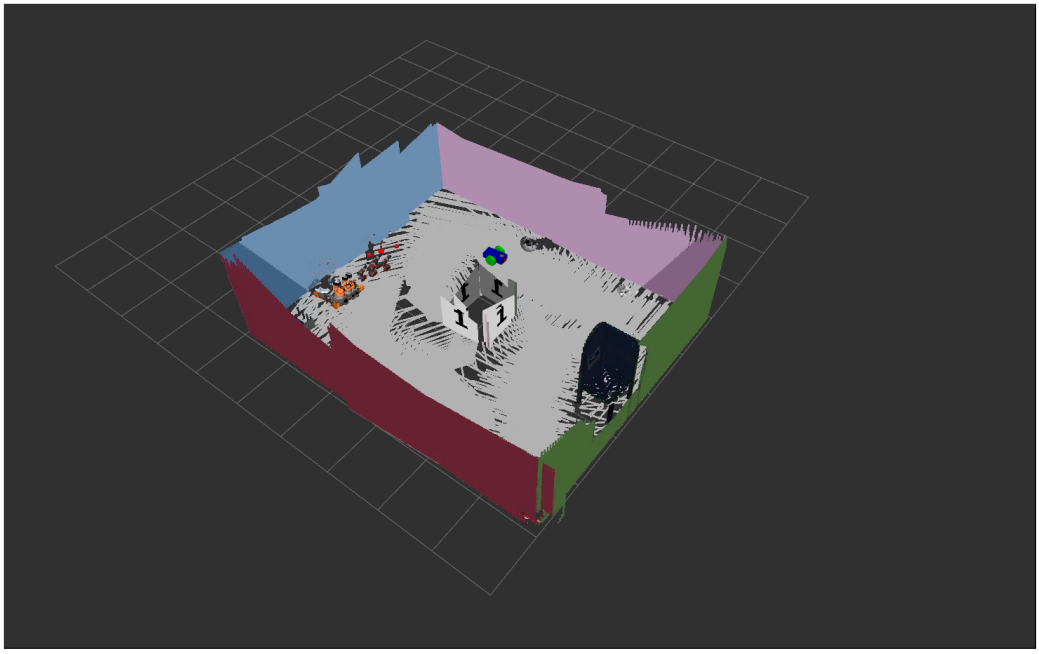

Figure 24: A 3D map of the alternative environment created by RTAB-Map displayed in Rviz

5 Discussion

Discussion - The student explains how the procedure went and methodologies to improve it. The student should compare and contrast the performance of RTAB Mapping in different worlds.

6 Future Work

References