Albert-Ludwigs Universität Freiburg
Institut für Informatik

Arce y de la Borbolla, José 15.03.2021

Master Project in
Autonomous Intelligent Systems

Full Map Posterior SLAM in ROS

Prof. Wolfram Burgard
Supervised by
Dr. Daniel Büscher and Dr. Lukas Luft

Abstract

Current SLAM techniques generally use probabilistic sensor and motion models to cope with the uncertainty in the context of localizing a mobile robot in its environment and generating a map from it. In the context of grid maps, however, usually only the most likely map is computed, leaving aside additional, relevant information in the process.

The work described in this document implements the modeling of the full map posterior distribution using parametric probability density functions, for both the well known reflection map model and the recently introduced decay rate map model. This is done by going into the details of the implementation's streamlining and potential pitfalls, in order to take full advantage of the distribution over all possible maps and have a better localization accuracy overall as well as a measurement of the confidence in the estimated maps, all of this in the context of an extension to the ubiquitous *openslam_ gmapping* libraries and ROS wrappers through very little changes in the original code base.

1 Introduction

Typical implementations of Rao-Blackwellized Particle Filters using a discretized model of the environment in the form of Grid Maps normally make the assumption that each cell in the world is either completely occupied or completely free [7] [6] due to the fact that there is an easy way of computing the probabilistic distribution of the environment given the observations [9] [8].

Depending on the resolution of the discretization, however, the aforementioned assumption might not be entirely true. A given cell could be partially occupied due to an obstacle not aligning with the grid, or being smaller than the cell size for example. In these cases, it makes sense to not just model the state of a cell being in a binary way, that is, either totally free or occupied, but rather as continuum of values in between these two extreme cases [13].

The main objective of this project is to implement the proposed map model in an existing, readily available and popular framework, namely *OpenSLAM Gmapping*, and compare it with the currently used model. Furthermore, we can also take advantage of the additional information encoded in the map to better estimate the particle weights according to explicit derivations of the measurement likelihoods as a way of improving the particle filter even more.

All of the source code created and used for this project, along with the data sets and results, as well as installation and basic usage instructions, can be found in the following repository: https://github.com/joseab10/FMP_gmapping.

2 Developments

Apart from the changes made to the main SLAM program, because of the difficulty in testing, experimenting, evaluating, visualizing and debugging the features of the implemented localization and mapping modifications, several other programs and scripts were developed in conjunction to aid with these tasks. Figure 1 shows some of the most relevant nodes that were created or modified for the purposes of this project, and the message flow between them. The contents of this section are devoted to detailing the scopes and workings of these programs.

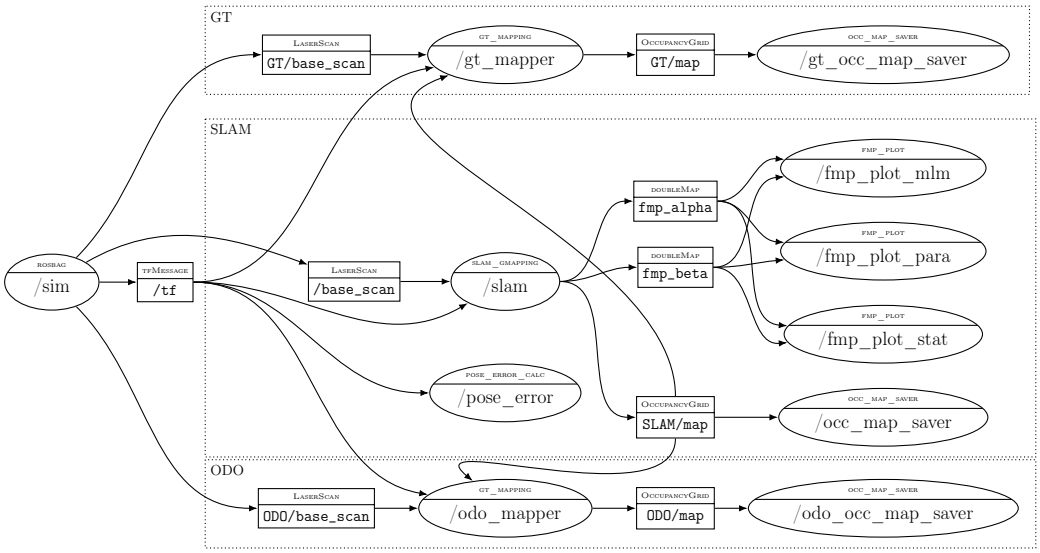


Figure 1: ROS Node Graph.- Graph representing the flow of information between nodes (ellipses) by means of messages (rectangles) for the experiments. The Node Type and Topic are displayed above each of the Node and Message names respectively. Nodes and messages are grouped into namespaces, represented by the dotted rectangles, for convenience. Some nodes and messages used are omitted for clarity.

2.1 SLAM Gmapping

The purpose of this section is to document in depth the development process of the changes done to the SLAM Gmapping source code, starting with the analysis of the existing base, and going through each of the modifications, their motivations and the methodology used to achieve the desired results.

At the beginning of the project, and after consulting with the original authors, it was decided to enhance the existing classes and methods by including additional parameters, control statements and functional features, instead of taking the route of sub-classing for time and practicality reasons. Thus, the following subsections address each of those changes minutely.

2.1.1 Reverse Engineering.

The original code for OpenSLAM Gmapping was developed by Cyrill Stachniss and Giorgio Grisetti back in 2007. It was originally designed to take CARMEN log files as an input to generate a map and corrected pose from the stored sensor measurements. On top of it, a wrapper was developed in order to use these libraries in the context of ROS nodes for robotic applications.

Both the *OpenSlam GMapping* source and it's ROS wrapper class were, at the time of this project, not very clearly documented. Therefore, as the initial step of the project, reverse engineering of these two repositories was performed in order to get an overview of the inner workings of the code, understanding the class structures and relevant fields and properties, and associating the function calls and the sequence of their execution with an abstract template for most SLAM-based algorithms.

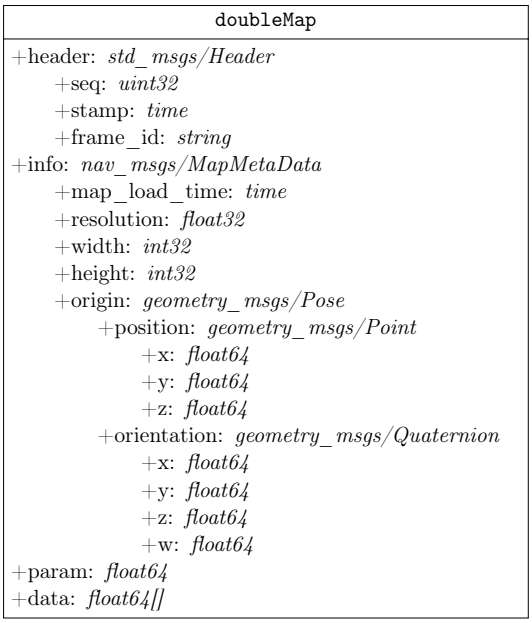
The results of this initial analysis can be found in Apendix A as a set of UML based diagrams. Figure 14a shows the properties and methods of the *SlamGmapping* class that wraps the functionality of the *openslam_gmapping* libraries for its use within ROS. Figure 14b displays some of the relevant namespaces, classes, structures and message types used by GMapping. Figure 15 contains the main classes and their relationships within the *open_slam Gmapping* source. Finally, Figure 16 showcases the order and logic of the function calls whenever a scan message is received by the *gmapping* node. In this last diagram, the abstract steps taken by a generic Particle Filter algorithm are shown inside dashed boxes, to the left of the actual function calls.

2.1.2 Custom ROS Messages

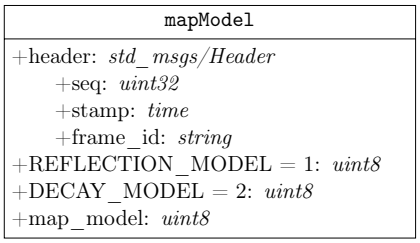
While the base version of ROS includes a plethora of message types to transfer data between different nodes and programs, it was still required to create custom ones for this project.

Since the map models now allow real valued cell states, as well as wanting to be able to publish the Full Posterior maps by means of their distribution parameters, it was decided to create a message type that supported double precision floating point numbers. For that reason, the `doubleMap` message type was created, based on its existing discrete counterpart `nav_msgs/OccupancyGrid`. It basically changes the type of the `data` field from an 8-bit integer to a 64-bit floating point array. It also adds a double precision floating point `param` property for sharing a single real value, usually the homonymous parameter of the prior distribution.

Originally, the map model types was going to be encoded as an enumeration in a header file to be used by the main *GMapping* libraries, as well as being included by some of the miscellaneous nodes that would process the data coming from SLAM. This approach worked fine at the beginning, but as more nodes were being added for extra visualization or data processing, it became cumbersome and error prone to have to manually define the map model as a parameter at the start of each node. Even though this could have been solved by sharing the same parameter through a ROS Launch file, it was decided to better implement the map model types as a message, that would both be used in code for controlling the flow of each program depending on the type of map being used, as well as being set up only once for the SLAM node, and having it publish the model type as a message so that the other nodes that may require this information could subscribe to the topic. Figure 2 shows the structure of the two main message types created for the project.



(a) Double Map Message.



(b) Map Model Message.

Figure 2: Custom Message specifications.

Besides the aforementioned custom messages, boolean messages were also sent directly from the simulator to signal other programs about relevant events. A boolean message published under the topic of `/doLocOnly` was used to signal the pertinent programs that the upcoming odometry and measurement messages are to be used for localization-only computations. This with the purpose of evaluating the SLAM algorithm as a whole (both mapping and localization) by measuring the translational and rotational errors of the poses with a known map, which was generated before this message is sent. Thus, the SLAM node knows to stop updating the map after the signal is received to perform only the localization computations. Similarly, other nodes may start logging to different files, or stop processing data altogether. Another boolean message, this time under the topic called `/endOfSim`, was used to let the interested nodes know when the simulation is over and no more pose or scan messages are going to be published by the simulator. The reason being that, since a lot of experiments are to be run with the purpose of collecting data, it was found as a good solution to signal the nodes to finish processing their remaining data and shutdown as a way to automate the tests. These last two messages are not displayed in Figure 1 so as to not clutter the diagram with less fitting information.

2.1.3 Full Map Posterior

A parameter was added to the original code of *Gmapping* that allows choosing the desired map model between the traditional Reflection maps, and the novel Exponential Decay Rate map model. Since the Decay maps need to store the accumulated traveled distance as well as the number of times a beam has been reflected, a field was added to the cell data structure for this purpose.

Table 1 shows a summary of the formulas most relevant to the implementation of the Full Map posterior functionality in the *OpenSLAM GMapping* node, as taken from [11]. As it can be seen on the table, the posterior distributions for the two map models correspond to a Beta and Gamma distributions, from which their properties can be found in Section 2.6

Properties	**Reflection Model**		**Exponential Decay Rate Model**	
Map Values $m_i =$	Reflection Probability μ_i		Decay Rate λ_i	
Forward Sensor Model $f(r_i, m_i, \delta_i) =$	Binomial Distribution $\mu_i^{\delta_i}(1-\mu_i)^{1-\delta_i}$	(1)	Poisson-Exponential Distribution $\lambda_i^{\delta_i} \mathrm{e}^{-\lambda_i r_i}$	(2)
Prior Distribution $p(m_i) =$	$\mathrm{Beta}(\mu_i; \alpha, \beta)$ (Uninf. with $\alpha = \beta = 1$)	(3)	$\mathrm{Gamma}(\lambda_i; \alpha, \beta)$ (Uninf. with $\alpha = 1, \beta = 0$)	(4)
Map Posterior $\mathrm{BEL}(m_i) =$	$\mathrm{Beta}(H_i + \alpha, M_i + \beta)$	(5)	$\mathrm{Gamma}(H_i + \alpha, R_i + \beta)$	(6)
Measurement Likelihoods $\ell(r_i, m_i) =$	$\frac{(H_i+\alpha)^{\delta_i}(M_i+\beta)^{1-\delta_i}}{H_i+\alpha+M_i+\beta}$	(7)	$\left(\frac{R_i+\beta}{R_i+\beta+r_i}\right)^{H_i+\alpha}\left(\frac{H_i+\alpha}{R_i+\beta+r_i}\right)^{\delta_i}$	(8)
Most Likely Map $m_i^\star =$	$\frac{H_i+\alpha-1}{H_i+\alpha+M_i+\beta-2}$	(9)	$\frac{H_i+\alpha-1}{R_i+\beta}$	(10)

Table 1: Reflection and Exponential Decay Rate Map Models.

The original implementation of *GMapping* included a particle weighting method that looked for a cell's mean hit point closest to the predicted beam endpoint, and plugged it into a zero-mean gaussian distribution with user-defined variance to compute a measurement likelihood. This method, however, depended only on each of the scan beams' endpoints, disregarding their entire traversed path. A more representative approach, as proposed in the original article [11], would be

to take advantage of the explicitly derived formulas for the measurement likelihoods, as can be seen in equations (7) and (8).

The way of computing the particle weight for either map model would then be:

$$w_i = \prod_{r_j} \prod_{k \in \mathcal{I}(r_j, x)} \ell\left(r_k, m_k\right) \tag{11}$$

That is, the particle weight is the product of the measurement likelihoods over all cells traversed by each beam in a scan.

As it can be easily inferred from the likelihood formulas, they will take a value of at most 1 for the Reflection Model and for the cells traveled through in the case of the Decay Model, or a value usually greater than 1 for the beam endpoints in the Decay Maps. Therefore, the resulting product over all traversed cells tends towards 0 for the Reflection model, or either to 0 or exponential growth as either the map resolution or the maximum range of the sensor increases. Thus, as a way of improving the numerical stability of the computations, the particle weights were computed as the sum of the log-likelihoods before normalizing.

$$l_i = \sum_{r_j} \sum_{k \in \mathcal{I}(r_j, x)} \log\left(\ell\left(r_k, m_k\right)\right) \tag{12}$$

For the cases where a likelihood for a single cell was found to be zero, control was broken out of the loop, since the entire likelihood would be zero due to the product. On the other hand, for cases undefined due to a cell not having been visited in the past, along with an undefined prior, i.e. the denominator of the respective likelihood being equal to zero, the cells were ignored for the computation.

Finally, in order to not alter the probability distribution of the particles, the weights are normalized using a *softmax* function so as to add up to a probability of one, while returning them to being normal likelihoods instead of their logarithms.

$$w_i = \frac{\exp\left(l_i\right)}{\sum_j \exp\left(l_j\right)} \tag{13}$$

2.1.4 Computing the beam's travel distance inside a cell.

For the Exponential Decay Rate map model, it is necessary to accumulate the distance each laser beam has traveled inside a cell. For that purpose, the function `computeCellR` was implemented in the `ScanMatcher` class. As it can be seen in Figure 3 and in 2, the function works by:

1. If the Map Model is not the Exponential Decay Rate model, then simply return 0 without going through the computation.

2. Computing the coordinates of the cell center ($\boldsymbol{cc} = (cc_x, cc_y)$) using the function `map2world` built into the map's object.

3. Getting the cell's border by adding the map's resolution `map.`δ to the center coordinates.

Algorithm 1: Computing a Scan's R values.

Input: map m, laser pose $\boldsymbol{lp}$, scan s

```
1 Function computeScanR(m, lp, s) :
2     for b in s do                                   /* For each beam in scan.  */
3         be ← lp + b.z (cos b.θ, sin b.θ)            /* Beam endpoint.          */
4
          /* Discretize beam using Bresenham or similar.                          */
5         line ← gridLine(lp, be)
6         for c in line do                            /* For each cell in the line. */
7             c.R ← c.R+ computeCellR(m,lp, be, c)
```

Verification of the method is simple: the sum of all r_i values for a given beam should add up to the range of the measurement. For that purpose, a simple map was designed and simulated with a sensor statically placed in the center providing a few measurements at select angles. Then r_i values were computed for each cell and asserted that they added up to the original measurement.

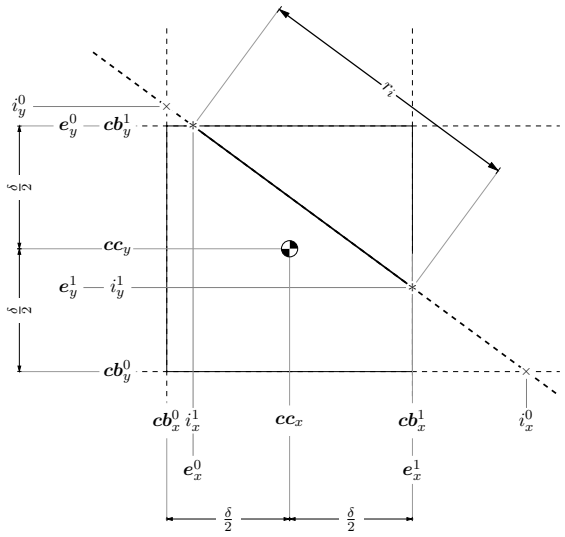


Figure 3: Computing a cell's r_i value.

It is important to note that the accuracy of the computation of the travel distance R for a given cell is also dependent on the algorithm used to discretize the beam's line. For instance, although Bresenham's algorithm [3] might be extremely efficient, it might not return all of the cells crossed by a given line, therefore the validation test mentioned earlier will fail to add up to the actual measurement. In order to solve this issue, a different algorithm can be used such as a line super cover or a modified version of Wu's algorithm [15]. For this reason, Section 2.1.5 deals with

implementing a different way of computing the voxels crossed by a line.

Algorithm 2: Computing a Cell's r_i value.

Input: map m, beam start $\boldsymbol{bs}$, beam end $\boldsymbol{be}$, cell index $\boldsymbol{c}$

Function `computeCellR`(m, $\boldsymbol{bs}$, $\boldsymbol{be}$, $\boldsymbol{c}$) :
  $\boldsymbol{\Delta b} \leftarrow \boldsymbol{be} - \boldsymbol{bs}$ /* Compute beam's delta */
  if m.`world2map`($\boldsymbol{bs}$) $= m$.`world2map`($\boldsymbol{be}$) $= \boldsymbol{c}$ **then**
    return $\|\boldsymbol{\Delta b}\|_2$ /* Beam starts and ends in Cell */
  $\boldsymbol{cb} \leftarrow m$.`map2world`($\boldsymbol{c}$) $\mp \frac{m.\delta}{2}$ /* Compute cell boundaries */
  if $\boldsymbol{cb} \notin [\boldsymbol{bs}, \boldsymbol{be}]$ **then** /* Beam outside of Cell */
    return 0
  if $|\boldsymbol{\Delta b}_x| \simeq 0$ **then** /* Vertical beam */
    if m.`world2map`($\boldsymbol{bs}$) $= \boldsymbol{c}$ **then** /* Beam start */
      return $\left|\boldsymbol{bs}_y - \left(\boldsymbol{cb}_y^0 InBeam?\, \boldsymbol{cb}_y^0 : \boldsymbol{cb}_y^1\right)\right|$
    else if m.`world2map`($\boldsymbol{be}$) $= \boldsymbol{c}$ **then** /* Beam end */
      return $\left|\boldsymbol{be}_y - \left(\boldsymbol{cb}_y^0 InBeam?\, \boldsymbol{cb}_y^0 : \boldsymbol{cb}_y^1\right)\right|$
    else /* Beam passes through cell */
      return $m.\delta$
  else if $|\boldsymbol{\Delta b}_y| \simeq 0$ **then** /* Horizontal beam */
    if m.`world2map`($\boldsymbol{bs}$) $= \boldsymbol{c}$ **then** /* Beam start */
      return $\left|\boldsymbol{bs}_x - \left(\boldsymbol{cb}_x^0 InBeam?\, \boldsymbol{cb}_x^0 : \boldsymbol{cb}_x^1\right)\right|$
    else if m.`world2map`($\boldsymbol{be}$) $= \boldsymbol{c}$ **then** /* Beam end */
      return $\left|\boldsymbol{be}_x - \left(\boldsymbol{cb}_x^0 InBeam?\, \boldsymbol{cb}_x^0 : \boldsymbol{cb}_x^1\right)\right|$
    else /* Beam passes through cell */
      return $m.\delta$
  else
    $m \leftarrow \frac{\boldsymbol{\Delta b}_y}{\boldsymbol{\Delta b}_x}$; $\quad b \leftarrow \boldsymbol{be}_y - m \cdot \boldsymbol{be}_x$ /* Get slope and bias */
    $i_x^0 \leftarrow \frac{\boldsymbol{cb}_x^0 - b}{m}$; $\quad i_x^1 \leftarrow \frac{\boldsymbol{cb}_x^1 - b}{m}$ /* Get int. points */
    if $i_x^0 > i_x^1$ **then** `swap`(i_x^0, i_x^1)
    $i_y^0 \leftarrow m \cdot \boldsymbol{cb}_x^0 + b$; $\quad i_y^1 \leftarrow m \cdot \boldsymbol{cb}_x^1 + b$
    if $i_y^0 > i_y^1$ **then** `swap`(i_y^0, i_y^1)
    $\boldsymbol{e}^0 \leftarrow$`max`($\boldsymbol{cb}^0$, $\boldsymbol{i}^0$) /* Get interval intersection */
    $\boldsymbol{e}^1 \leftarrow$`max`($\boldsymbol{cb}^1$, $\boldsymbol{i}^1$)
    if m.`world2map`($\boldsymbol{bs}$) $= \boldsymbol{c}$ **then** /* Beam start */
      return $\left\|\boldsymbol{bs} - \left(\boldsymbol{bs} \leq \boldsymbol{e}^0 \leq \boldsymbol{be}?\, \boldsymbol{e}^0 : \boldsymbol{e}^1\right)\right\|_2$
    else if m.`world2map`($\boldsymbol{be}$) $= \boldsymbol{c}$ **then** /* Beam end */
      return $\left\|\boldsymbol{be} - \left(\boldsymbol{bs} \leq \boldsymbol{e}^0 \leq \boldsymbol{be}?\, \boldsymbol{e}^0 : \boldsymbol{e}^1\right)\right\|_2$
    else /* Beam passes through cell */
      return $\left\|\boldsymbol{e}^1 - \boldsymbol{e}^0\right\|_2$

2.1.5 Line Supercover

Since the original implementation of GMapping included a version of Bresenham's algorithm as a way of discretizing the measurement beams within the map's grid, not all of the cells crossed by a beam were necessarily taken into account for the map updates. This did not originally represent a major issue, since the particle weighting method only considered the endpoints, and the cells that could have been ignored by the algorithm during map updates, which where crossed by the measurement beams, did not account for big errors in localization.

However, given that part of the project scope was to also implement a particle weighting method based in the explicit measurement likelihood formulas defined in equations 7 and 8, that these method takes into consideration the entirety of the beam instead of just the endpoint, and is sensitive to the map's resolution and the accuracy of the line discretization algorithm, a modified version of Bresenham's algorithm had to be coded.

The algorithm used is called a Supercover of a 2D line [2] based on the implementation of [4]. It can be found in Algorithm 3 and consists of a couple of small modifications to the original Bresenham algorithm:

- It takes the error of the previous iteration into consideration as well as the one from the current step, as well as using the doubled values of the differences in x and y for better accuracy.

- Whereas Bresenham, for each step in the x direction, decides between only two choices, namely keep y as is or incrementing it, the modified version also checks the upper-left and lower-right cells whenever a change in the y axis is performed.

(a) Bresenham's Algorithm. (b) Line Supercover Algorithm.

Figure 4: Line Discretization Algorithm Comparison.

Figure 4 shows the difference in voxel coverage between the two approaches for a given line. While Bresenham's (Figure 4a) selects a single cell per horizontal unit, the Line Supercover (Figure 4b) takes into consideration the preceding left or bottom cells as well whenever there is a change in the vertical direction. Though the original implementation of the Supercover algorithm accounts for the cases where the line exactly intersects the grid crossings (cells shown in gray in the figure), the implemented version ignores these occurrences since the r_i value for such cells would be equal

to 0.

Algorithm 3: Computing a line's Supercover (Modified Bresenham algorithm).

Input: Line start (x_0, y_0), line end (x_1, y_1)

```
1  Function lineSupercover(x_0, y_0, x_1, y_1) :
2      step_x ← 1;  step_y ← 1                          /* Step direction in x and y.  */
3      x ← x_0;  y ← y_0                                /* Line cells.  */
4      δ_x ← x_1 − x_0;  δ_y ← y_1 − y_0
5      dδ_x ← 0;  dδ_y ← 0                              /* Double differences for better precision.  */
6      ε ← 0;  ε_prev ← 0                               /* Current and previous error values.  */
7      mark(x, y)
8      if δ_x < 0 then
9          step_x ← −1;  δ_x ← −δ_x
10     if δ_y < 0 then
11         step_y ← −1;  δ_y ← −δ_y
12     dδ_x ← 2 × δ_x; dδ_y ← 2 × δ_y
13     if dδ_x ≥ dδ_y then                              /* 0 ≤ slope ≤ 1 */
14         ε_prev ← ε ← d_x
15         for i = 0; i < δ_x; i ← i + 1 do
16             x ← x + step_x
17             ε ← ε + dδ_y
18             if ε > dδ_x then                         /* Change in y axis required.  */
19                 y ← y + step_y
20                 ε ← ε − dδ_x
21                 if ε + ε_prev < dδ_x then mark(x, y − step_y)     /* Mark bottom cell.  */
22                 else if ε + ε_prev > dδ_x then mark(x − step_x, y)  /* Mark left cell.  */
23             mark(x, y)
24             ε_prev ← ε
25     else
26         ε_prev ← ε ← d_y
27         for i = 0; i < δ_y; i ← i + 1 do
28             y ← y + step_y
29             ε ← ε + dδ_x
30             if ε > dδ_y then
31                 x ← x + step_x
32                 ε ← ε − dδ_y
33                 if ε + ε_prev < dδ_y then mark(x − step_x, y)
34                 else if ε + ε_prev > dδ_y then mark(x, y − step_y)
35             mark(x, y)
36             ε_prev ← ε
```

Even though this method is, just as Bresenham's, still a particular case of a Discrete Differential Analizer, entirely based in integer arithmetic, given the fact that at any point it might select two voxels instead of one for each horizontal step, its performance will be affected, as well as that of the methods depending on it, i.e.: map updates and particle weighting based on measurement likelihoods. For the same reason, the memory that was originally allocated in a single shot for all line discretization operations was increased.

2.1.6 Average Particle Pose

Originally, the OpenSLAM Gmapping node ROS Wrapper was originally programmed to provide the transformation between the `/map` and `/odom` coordinate frames of the most likely particle, i.e. the particle with the highest weight.

As a way of obtaining a less susceptible to noise and thus more robust estimate of the robot pose, the functionality of computing the average pose over all particles was implemented, while retaining the original functionality by means of an optional parameter.

The average position can be computed as follows for a given set of particles, each defined as $\langle \boldsymbol{x}_i, \boldsymbol{m}_i, w_i \rangle$.

$$\begin{pmatrix} \overline{x} \\ \overline{y} \end{pmatrix} = \begin{pmatrix} \sum_i w_i \cdot x_i \\ \sum_i w_i \cdot y_i \end{pmatrix} \tag{14}$$

where:

- $\boldsymbol{x}_i$ represents the 2D pose $\begin{pmatrix} x & y & \theta \end{pmatrix}^\mathsf{T}$,
- $\boldsymbol{m}_i$ is the particle's map, and
- w_i is the particle's weight

The average orientation, due to it's cyclical nature, however, must be computed using the so called circular average [12]:

$$\overline{\theta} = \arctan\left(\sum_i w_i \cdot \sin\theta_i, \sum_i w_i \cdot \cos\theta_i\right) \tag{15}$$

2.1.7 Overconfidence Uniform Noise

With the intention of preventing the particle weights from becoming too overconfident, and thus eventually leading to particle deprivation and poor localization accuracy, particularly prevalent for the case when using the combination of the Exponential Decay model and Measurement Likelihood particle weights, an optional weighting parameter (w_{oc}) was introduced, along with the functionality to add random uniform noise to the likelihoods of each of the cells traversed by a scan beam.

$$\ell_i = (1 - w_{oc}) \cdot \ell_i(r_i, m_i) + \frac{w_{oc}}{{r_{\max}}^{\delta_j}} \quad , \; w_{oc} \in [0, 1] \tag{16}$$

where:

- $\ell_i(r_i, m_i)$ is the likelihood of having obtained measurement r_i for the current cell m_i,
- $r_{\max}$ is the maximum usable range of the sensor, and
- δ_j takes the value of 1 if beam j of the current scan was a hit, or 0 if it was an out-of-range measurement.

Despite the fact that, at an individual voxel level, the uniform noise does tend to lower the overconfidence of the likelihood estimate, for cases with either a high map resolution, maps spanning a very large area, or extremely long range sensors being used, this method seems to become outweighed by the huge amount of cells per scan ray, lowering its effectiveness. Regardless of this, the functionality was left for possible future uses.

2.1.8 Occupancy Map

In order to visualize the generated map within the ROS environment, it has to be published under the `nav_msgs/OccupancyGrid` message structure, which allows for only three possible states for each cell, namely:

- Unknown
- Free
- Occupied

The original implementation of *GMapping* already implements a conversion for the Reflection Maps, which already encode the probability of a cell being occupied as a probability in the range $[0, 1]$. Thus, the generation of the three-state occupancy grid is achieved simply by thresholding the computed map values.

$$occ_i = \begin{cases} 0 & \frac{h_i}{v_i} < \tau \\ 50 & v_i = 0 \\ 100 & \frac{h_i}{v_i} \geq \tau \end{cases} \tag{17}$$

where:

- occ_i is the occupancy value of cell i, 0 being free, 100 being occupied and 50 having an unknown state,
- h_i is the number of hits the cell has accumulated so far,
- v_i is the number of times the cell has been crossed by a beam, and
- τ is a threshold value.

Computing the occupancy values for the Exponential Decay model requires an additional step before thresholding, since the computed cell values do not encode occupancy directly, but rather the distance traveled by beams inside any given cell.

$$occ_i = 1 - \exp\left(-d \cdot \frac{h_i}{R_i}\right) \tag{18}$$

where:

- d is a cell length constant, which can take the value of the cell height or width (δ), or in our case, the length of the cell's diagonal: $d = \sqrt{2} \cdot \delta$,
- h_i is, as before, the number of hits a cell has accumulated, and
- R_i is the accumulated distance traveled by beams inside of cell i.

Once the map has been converted to encoding reflectivity, it can be passed through a threshold to obtain the occupancy, just as for normal reflection maps.

2.1.9 Localization-only SLAM

For the evaluation of the algorithms, a feature was implemented such that, after receiving a given signal, that is, a custom ROS message described in Section 2.1.2 the particle filter would reinitialize the trajectory tree and all of the particles with the last computed most-likely map, and take the first pose after the signal as their deterministic initial pose.

It would then proceed to disable the map updating procedure so that, in effect, the algorithm performs only the localization part of SLAM. Thus, in order to compare the accuracy of the maps, instead of computing a distance metric between two raster images, we can simply take the geometric distance between the last poses of the localization only phase.

2.2 CARMEN to ROSBag conversion

Most, if not all, of the benchmark datasets used for SLAM can be found in Carnegie Mellon Robot Navigation Toolkit's (CARMEN) log format. In order to use them with GMapping, it is necessary to convert them to a suitable ROSBag format. For that purpose, the `carmen2rosbag.py` script from [5] was used, with a few minor modifications. The script originally published the odometry transform, at the base of the robot associated with each measurement scan. This lead to inaccuracies in both the Localization and Mapping resulting in poor scan matching scores and thus making it unnecessarily slightly more difficult for maps to converge. For this reason, it was set to publish the transform at the pose of the laser sensor instead.

In order to better control the message frequency, a parameter was added to choose to ignore the original time stamps, and use a custom rate instead. This enabled the simulations to have a more consistent feel, while at the same time allowing us to tune the publication rate so as to not loose scans due to buffer overflows on the GMapping side.

Another parameter was added to give the option of publishing an *EndOfSimulation* boolean message at the end of the bag to signal other nodes perform specific actions and shut down.

Finally, to keep the size of the bags more compact, the odometry messages were dropped from the conversion process, since GMapping does not have any use for them, as it relies solely on the TF transformations to estimate the pose of the robot.

2.3 2D Map Simulator

Several popular data sets have been used for benchmarking SLAM algorithms over the years, such as the buildings of the MIT's Computer Science and Artificial Intelligence Lab, the inside of the Intel Research Lab or even the campus and buildings of the Technical Faculty at the University of Freiburg. However, due to the shear amount of data they contain, the scale of the maps to be generated, and the fact that they contain only the noisy recordings directly from the robots, without providing an exact Ground Truth, it was decided to generate our own data from simulations to tackle these issues.

Although ROS comes with its own prepackaged simulator (Gazebo), because GMapping is designed to work with planar Lidar measurements and 2D maps, it was felt that Gazebo would be overkill and require a steeper learning curve to customize it for the project needs. For that purpose, a custom simulator was programmed to be able to generate noisy pose and measurement data for the SLAM algorithm, while keeping track of the Ground Truth trajectory and scans.

The way the simulator works is by taking in settings such as the noise covariance, a map with geometric obstacles and robot commands in the form of JSON files, and generating ROSBag file with the necessary transform, scan, and custom messages. One of the reasons for saving the simulations to files, instead of online transmission of the messages directly to other nodes, is that the computations for the measurements involve raycasting and calculating intersections with the obstacles in the map, thus being relatively computationally expensive, especially if we are trying to perform SLAM at the same time. The other reason being that, given the random nature of the simulated noise and the particle filter, having a fixed set of noisy poses and measurements helps in evaluating the localization algorithms themselves, just as most of the benchmark datasets usually contain data for a single run.

For each movement command, the simulator generates three transform trees, as can be seen in Figure 5, namely a Ground Truth, Noisy Plain Odometry and Noisy SLAM poses respectively. It starts by generating a Ground Truth deterministic movement from the previous pose by following the command's settings. It then adds gaussian noise according to the motion model to generate the Plain Odometry Pose. These two poses are published with a zero-translation and zero-rotation with respect to the global coordinate frame ($/map$), whereas the $/map \mapsto /SLAM/odom$ transform is left unpublished so that the SLAM algorithm makes the pose corrections there.

2.4 Ground Truth Mapper

Although the same full SLAM program could have been used for generating maps with known poses, by reducing the movement and measurement uncertainties down to zero, because of its design as a Monte Carlo method and the overhead of its computation, it was deemed overkill for the purpose of qualitatively comparing the maps generated by the SLAM algorithms. Thus a ground-truth mapping node was developed solely for this purpose.

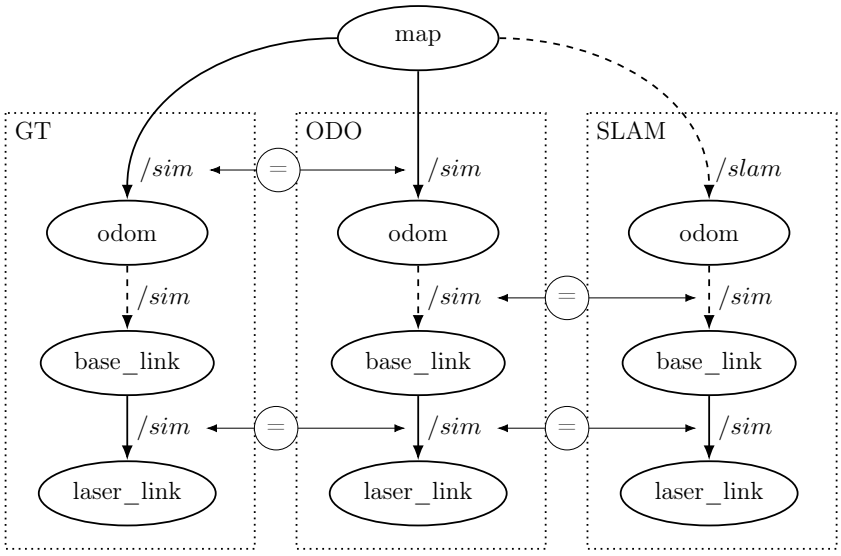


Figure 5: TF Transform Tree.- Transformation Tree used for all the experiments, where *map* is the fixed reference frame from which all others depend. Transformations and frames are grouped according to the data they belong to, namely: Ground Truth (GT); pure, noisy odometry (ODO); and pose corrected via localization and mapping (SLAM). Solid arrows represent static transformations between frames, whereas the dashed ones are dynamic and thus change over time. The labels next to the arrows display the node publishing the corresponding transform, */sim* being the ROSBag simulation and */slam* the GMapping node. The equal signs are used to represent whenever two transforms are equal to each other.

The Ground Truth Mapper, just as its full blown SLAM sibling, subscribes to laser scan messages and outputs an occupancy grid. The main difference being, that it considers the received poses and measurements as being noise-free.

In order to be able to easily compare the maps generated by SLAM and those computed by this program, both maps should share the same conversion from continuous world coordinates to their discretized counterparts in the image space. Thus, this program is designed to publish a map only after the SLAM has done the same. This is achieved by subscribing to the `/map` topic and getting the map's resolution, width and height, and origin directly from the SLAM map since the size and origin of the map change as new poses and measurements become available.

Though the previous approach solves the problem of an ever changing map, it creates a new one by not being able to process scans before the first SLAM map is generated. Furthermore, if the origin of the SLAM map changes, then the indexing of the cells must also change, making it impossible to store the processed scans in an array-like structure. Finally, it was assumed that the map resolution would not change for the entire duration of a given run, since that would have forced to store the entire scan history in its raw form and recompute the endpoints for all of the beams of all of the scans every time a map was to be published.

The way the Ground Truth Mapper solves these issues is by:

- Whenever a scan is received:
  - **If no map has been received**: The beam endpoints are computed and the scan information is stored as a $(\boldsymbol{x}_i, \boldsymbol{e}_i, \boldsymbol{max}_i)$ tuple in a queue, where $\boldsymbol{x}_i$ represents the pose of the laser sensor in world coordinates, $\boldsymbol{e}_i$ is a list of the x and y world coordinates of the endpoint of each beam in the scan and $\boldsymbol{max}_i$ is a boolean list containing *true* if the corresponding beam was a max-range reading, $false$ if it was a hit.
  - **If at least one map has been received**:
    - Store every cell traversed by the beams of the scan in hash tables for hits and visits, indexed by keys composed of tuples with the center of each cell in world coordinates, that defaults to 0 if the key is not found.
    - Compute the discrete map coordinates of the laser pose $\boldsymbol{x}_i$ and each of the beam endpoints $\boldsymbol{e}_i$ by
    $$\boldsymbol{X}_i = \frac{\boldsymbol{x}_i - \boldsymbol{o}_i}{\delta} \tag{19}$$
    where $\boldsymbol{X}_i$ are the coordinates in the discrete map reference frame of the cell corresponding to the $\boldsymbol{x}_i$ point in world coordinates to convert, $\boldsymbol{o}_i$ is the coordinate in the world reference frame of the $(0, 0)$ coordinate in the discrete map reference frame, and δ is the cell size or map resolution.
    - Generate a list of integer coordinates corresponding to the discretization of the line defined by the laser pose cell center and the beam's endpoint cell center using Bresenham's algorithm or similar. Since the ground truth maps are a qualitative measure, a more precise line algorithm is not needed.
    - Convert the discrete coordinates of the line cells back to the world reference frame to get the position center of the cell by using the inverse of equation (19).
    - For each discrete cell traversed by the beam, increment the value stored in either the hits and/or the visits default dictionary under the key corresponding to the center coordinates of the cell.

- Whenever a map is received:
  - If this is the first time we receive a map message, store the map's resolution. Otherwise, assert that the resolution did not change from previous maps.
  - If there were un-processed scans stored in the buffer from before a map was received, process them in the same fashion as mentioned above.
  - Create a map array with all cells defaulting to 50 (unknown state) of the same dimensions and with the same origin as the received map.
  - For each (cell, value) pair of items in the *visits* hash table, compute the occupancy of the cell by Equation (17) and assign it to its corresponding element in the map array.

This method allows for an efficient storage of the map information under the conditions that the grid size might grow or shrink and the cells' indexes might change from map to map as the origin is relocated without having to copy the data between data structures.

2.5 Pose Error Evaluation

To be able to properly compare the accuracy and performance between the different methods of localization and mapping and their parameterizations, an error function is required.

One way of doing this is by comparing the generated maps between several runs of the algorithms and a ground truth map through some sort of distance function [1]. However, this approach comes with it's own caveats, such as selecting a proper metric (like the Hausdorff distance), being dependent on the map size and resolution as well as other factors.

For those reasons, we decided to evaluate the accuracy from the poses themselves, just as it was proposed in [10]. The error function for every movement and measurement step was defined as:

$$\varepsilon_t = \text{trans}\left(\boldsymbol{x}_t, \boldsymbol{x}_t^\star\right) + \alpha \cdot \text{rot}\left(\boldsymbol{x}_t, \boldsymbol{x}_t^\star\right) \tag{20}$$

where ε_t is the pose error at time step t, $\boldsymbol{x}_t$ and $\boldsymbol{x}_t^\star$ are the computed and ground truth poses at time step t respectively, trans$(\cdot)$ and rot$(\cdot)$ are the translational and rotational distance functions, and α is a weighting parameter, also used as a way of making the units of the two error components equal and thus be able to add them together.

For poses defined as $\boldsymbol{x}_t = \begin{pmatrix} \boldsymbol{t}_t^\mathsf{T} & \boldsymbol{q}_t^\mathsf{T} \end{pmatrix}^\mathsf{T}$, where $\boldsymbol{t}_t$ is the displacement of the robot with respect to the world coordinate system and $\boldsymbol{q}_t$ is the rotation of the robot expressed as a quaternion, the translational error between the noisy and the ground-truth pose is easily computed by:

$$\text{trans}\left(\boldsymbol{x}_t, \boldsymbol{x}_t^\star\right) = \left\|\boldsymbol{t}_t^\star - \boldsymbol{t}_t\right\|_2 \tag{21}$$

Since the transforms in ROS have the rotations already expressed as quaternions, and that they provide a robust framework for composing multiple rotations, the relative orientation between the ground truth and the localized pose can be computed simply by:

$$\boldsymbol{q}_t^{\text{rel}} = \boldsymbol{q}_t^\star \cdot \overline{\boldsymbol{q}_t} \tag{22}$$

where the overline above the noisy rotation represents the quaternion conjugate or inverse of said rotation. By definition, a unitary quaternion can be though of as a single angle θ turned around an axis $\boldsymbol{a} = \begin{pmatrix} a_x & a_y & a_z \end{pmatrix}^\mathsf{T}$:

$$\boldsymbol{q} = \begin{pmatrix} q_w \\ q_x \\ q_y \\ q_z \end{pmatrix} = \begin{pmatrix} \cos\frac{\theta}{2} \\ a_x \sin\frac{\theta}{2} \\ a_y \sin\frac{\theta}{2} \\ a_z \sin\frac{\theta}{2} \end{pmatrix} \tag{23}$$

Thus, as our measure of rotational error, we are considering the angle between the two poses by combining (22) and (23).

$$\text{rot}\left(\boldsymbol{x}_t, \boldsymbol{x}_t^\star\right) = 2 \cdot \arccos\left(\left[\boldsymbol{q}_t^\star \cdot \overline{\boldsymbol{q}_t}\right]_w\right) \tag{24}$$

While some authors might prefer to use the squared distances, it was decided not to in order to have a more intuitive understanding of the actual meaning of the errors, especially since both the euclidean distance for the translational error and the arccos function always yield non-negative numbers.

Finally, even though in the end the used metric was the error for last time step only, the accumulated error over the entire trajectory can be computed as the sum of the total errors at each time step t:

$$\varepsilon_{\text{total}} = \sum_{t=1}^{T} \varepsilon_t \tag{25}$$

2.6 Full Map Posterior Plotter

As a means to helping debug the software, as well as providing some visual feedback and qualitative results, a node was developed whose specific purpose is to subscribe to the published α_i and β_i parameter maps and represent useful properties of the Map Posterior in a graphical way.

At the time of this writing, the node supports plotting the following information:

- **Parameters:** Plots the values of α_i and β_i of the posterior distribution for each cell i.
- **Statistics:** Generates heat maps for the mean and variance of the posterior distribution of the map.
- **Most Likely Map:** Shows the unthresholded most likely map (mode) of the posterior distribution.

Given that each of the two map models used in this project corresponds to a specific type of probability distribution, as stated in Section 2.1.3, computing their properties boils down to applying their respective formulas, as shown in Table 2.

Although plotting these maps is as simple as computing each property according to the aforementioned formulas for each voxel, assigning them a color according to a certain function, and finally publishing them under a ROS Topic, a few considerations and special cases had to be made in order for it to work properly during the implementation. For instance, since the support set for Exponential Decay Rate maps is $[0, \infty)$, it was decided to use a logarithmic scale for coloring of the mean, variance and mode images to make them easier to understand. Also, as can be seen in Equation (34), it is theoretically possible for a Beta Distribution to represent a Bi-Modal or even a Uniform distribution. Thus, this cases had to be addressed explicitly. Similarly, when a cell has not been hit by a beam, depending on the prior distribution, the value of said cell could be undefined as the denominator would be equal to zero in the formulas for mean, variance and mode for both map models.

Properties	**Beta**		**Gamma**	
Notation	$\text{Beta}\,(x;\alpha,\beta)$		$\text{Gamma}\,(x;\alpha,\beta)$	
Parameters	shape $\alpha > 0$ shape $\beta > 0$		shape $\alpha > 0$ rate $\beta > 0$	
Support	$x \in [0,1]$		$x \in (0,\infty)$	
PDF	$f(x) = \dfrac{x^{\alpha-1}(1-x)^{\beta-1}}{\mathrm{B}(\alpha,\beta)}$	(26)	$f(x) = \dfrac{\beta^{\alpha}}{\Gamma(\alpha)} x^{\alpha-1}\mathrm{e}^{-\beta x}$	(27)
CDF	$f(x) = \mathrm{I}_x(\alpha,\beta)$	(28)	$f(x) = \dfrac{1}{\Gamma(\alpha)}\gamma(\alpha,\beta x)$	(29)
Mean	$\mathbb{E}[x] = \dfrac{\alpha}{\alpha+\beta}$	(30)	$\mathbb{E}[x] = \dfrac{\alpha}{\beta}$	(31)
Variance	$\mathrm{Var}[x] = \dfrac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$	(32)	$\mathrm{Var}[x] = \dfrac{\alpha}{\beta^2}$	(33)
Mode	$x^{\star} = \begin{cases} \frac{\alpha-1}{\alpha+\beta+2} & \text{for } \alpha,\beta > 1 \\ (0,1) & \text{for } \alpha = \beta = 1 \\ 0,1 & \text{for } \alpha,\beta < 1 \\ 0 & \text{for } \alpha \leq 1, \beta > 1 \\ 1 & \text{for } \alpha > 1, \beta \leq 1 \end{cases}$	(34)	$x^{\star} = \dfrac{\alpha-1}{\beta}$ for $\alpha \geq 1$	(35)

Table 2: Select Mathematical Properties of the Beta and Gamma Distributions.

2.7 Miscellaneous Scripts

Along with the main ROS nodes and programs, a set of helper scripts was coded to aid in functions such as starting the stack of nodes via a ROS Launcher file, a python wrapper of the launch file to redirect the debug logs, scripts for storing, aggregating and plotting data, and others for executing multiple experiments in a processing pool.

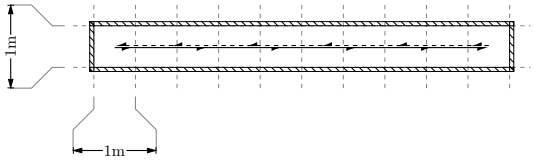
3 Experiments

3.1 Simulations

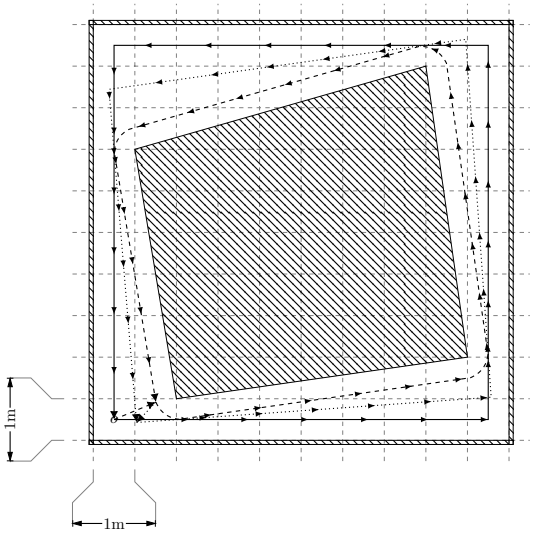
Using the simulator described in Section 2.3, two different maps, shown in Figure 6 were created for the purpose of testing evaluating the different algorithms, as well as some other maps used for specifically debugging some of the added functionalities, such as the improved line discretization algorithm or the computation of the cells' R values.

The second map was designed to be simple, while having two distinct corridors and two similar looking ones to allow some ambiguity, as well as including one loop closure. While all of the

evaluations were performed in this environment, a single corridor map was also created for the initial testings.



(a) 10×1 Corridor Map.



(b) 10×10 Loop Map.

Figure 6: Test Environments.

For all of the experiments, a robot was simulated which followed the odometry model stated in Algorithm 5.5 from [14], with noise parameters:

$$\boldsymbol{\alpha} = \begin{pmatrix} \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 \end{pmatrix} = \begin{pmatrix} 0.01 & 0.05 & 0.01 & 0.05 \end{pmatrix} \tag{36}$$

For the measurements, a laser range sensor was simulated with 180 beams covering an angle in the interval of $\left[-\frac{3}{4}\pi, \frac{3}{4}\pi\right]$, measured from the direction of the robot. For simulating the uncertainty in the measurements, noise sampled from a multivariate, zero-mean gaussian distribution, was added to the ground truth measurements computed through ray tracing. The used covariance matrix was:

$$\boldsymbol{\Sigma} = \begin{pmatrix} \sigma_{rr} & \sigma_{r\theta} \\ \sigma_{\theta r} & \sigma_{\theta\theta} \end{pmatrix} = \begin{pmatrix} 0.005 & 0.0 \\ 0.0 & 0.0 \end{pmatrix} \tag{37}$$

That is, the bearings were completely deterministic, while the ranges had a variance of 5mm

Finally, several simulations were generated, each with a different number of movement and measurement steps and saved to ROS Bag files for repeatability. Tests with a single counter-clockwise mapping loop around the map, following the path parallel to the outer walls were performed with $\{20, 30, 40, 50, 60, 70, 80, 90, 100\}$ steps. In the same way, runs with two loops around the circuit, following both a path parallel to the outer walls and one parallel to the inside walls with rounded corners, were performed with $\{120, 140, 160, 180, 200\}$ steps in total. Lastly, experiments looping around with three paths, namely parallel to the inside/outside walls, and a bisection between the two, were done with $\{240, 270, 300\}$ moves in total. The aforementioned paths can also be seen in Figure 6.

In all of the experiments, a 100 step clockwise loop was executed after the mapping cycle for the localization-only phase of the SLAM for the purpose of evaluating the accuracy of the resulting map. After the mapping phase ends, and at the beginning of the localization loop, the robot is deterministically moved to the starting position as a way of initializing poses of the particle filter.

3.2 SLAM Parameters

For all the experiments, the pose improvement via scan matching was disabled to better compare the efficacy of the proposed methods alone, since the scan matching parameters were highly dependent on the environment, map resolution and other factors and thus required some careful tuning to avoid the estimated poses from jumping around and consequently distorting the computed maps. As for the map's prior belief, the values of the hyperparameters for an uninformative prior were used in all tests, that is: $\alpha = 1$ for either map type and $\beta = 1$ for the Reflection Map and $\beta = 0$ for the Exponential Decay Rate Map. The maps were generated with a resolution of $\delta = 5$cm and an occupancy threshold of 25%. The weight for the uniform distribution to avoid overconfidence was left as 0, since it did not make much difference during initial tests. All other parameters were left using the default values in all cases.

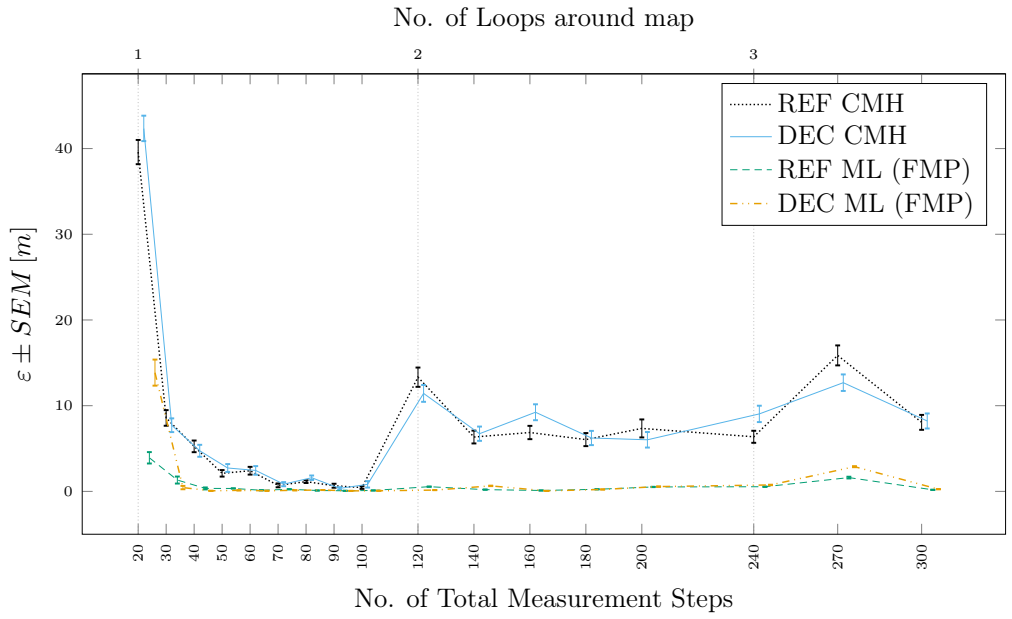
Each combination of map model, particle weighting method and number of measurement steps was executed 1000 times and the pose errors were stored and accumulated during the mapping and localization only phases.

4 Results

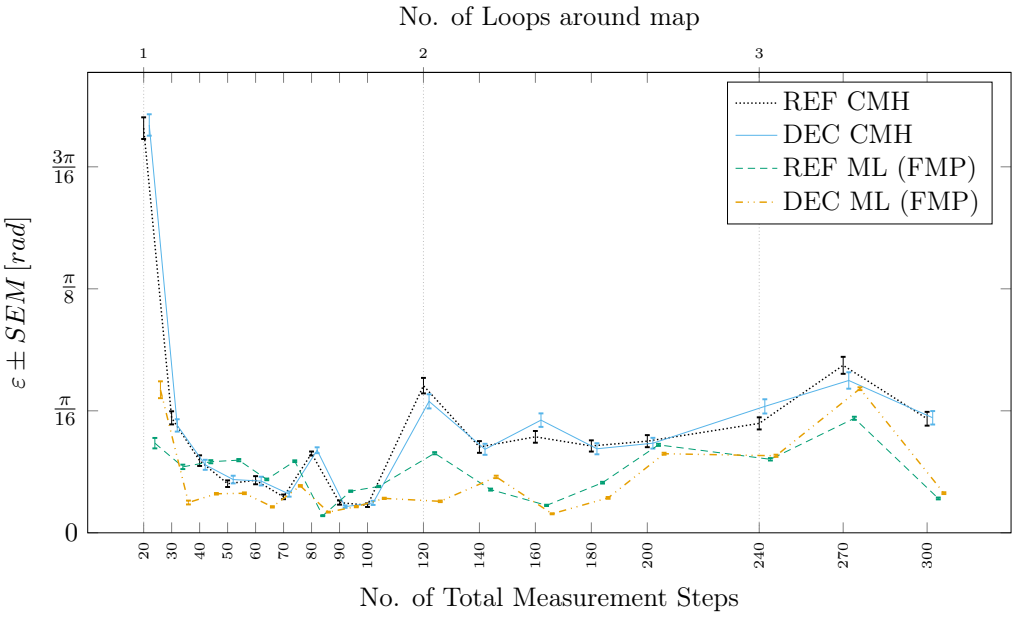
The results of the accumulated mean and standard deviation over 1000 experiments for the translational and rotational errors of the robot pose, with respect to the ground truth position and orientation, after the Localization-Only phase, can be found in Figures 7a and 7b respectively. Each data series represents a combination of one of the map models, i.e. Reflection or Exponential Decay Rate map(labeled REF or DEC respectively), and one particle weighting method, i.e. the so called Closest Mean Hit Likelihood (Original *GMapping* method, labeled CMH) and the proposed Measurement Likelihood metric (labeled ML).

As it can be expected, the curves display a clear descending trend in the errors as the number of mapping steps increases. Interestingly though, both the positional and rotational errors tend to spike upwards at around 120 and 240 measurement steps. This is probably due to the fact that, even though it is greater in total, the number of measurements is in fact a smaller amount per cycle around the map. A difficulty in data association, that is the algorithm being aware that it has reached a previously visited location again, is probably also partially responsible for this results.

Full Map Posterior SLAM in ROS



(a) Translational Error



(b) Rotational Error

Figure 7: Localization Pose Error.- Absolute positional error w.r.t the ground truth pose computed during the Localization-Only phase, after having generated a map using the full SLAM stack. Error bars represent the Standard Error of the Mean (SEM). The four data series represent the combination of Reflection (REF) or Exponential Decay Rate (DEC) map models with the original (CMH) and Measurement Likelihood (ML) particle weighting methods. The curves marked as REF CMH show the results of the original implementation of *gmapping*.

As it can be appreciated from the curves, the methods taking advantage of the Full Posterior Map and the proposed Measurement Likelihoods have not only a lower mean error value in general, but also a lower variability. This can be further seen in Table 3, where the means and standard deviations of each combination of settings is compared via ratio to the original *GMapping* settings, i.e. a Reflection map and particle weights using the Closest Mean Hit likelihood. It is important to note that these mean and standard deviation ratio values make no claims about the actual probability distributions, but are only intended as a way of comparing the expected errors and their variability across the different methods. Values smaller than one represent an accuracy improvement with respect to the original implementation.

Mapping Steps	Reflection		Decay			
	ML		CMH		ML	
	Mean	σ	Mean	σ	Mean	σ
20	0.0990	0.4696	1.0698	1.0485	0.3500	1.0753
30	0.1545	0.4479	0.9007	0.8782	0.0519	0.2144
40	0.0659	0.1691	0.9014	1.0256	0.0127	0.0047
50	0.1629	0.0300	1.2960	1.2011	0.0584	0.0172
60	0.0554	0.0117	1.0148	1.1155	0.0280	0.0276
70	0.3537	0.0321	1.1988	1.0632	0.1841	0.0281
80	0.0921	0.0244	1.4011	1.6175	0.1398	0.0270
90	0.1040	0.0123	0.5289	0.5978	0.0909	0.0171
100	0.2459	0.0212	1.8543	2.4787	0.1689	0.0278
120	0.0413	0.0166	0.8581	0.8743	0.0114	0.0083
140	0.0322	0.0192	1.0613	1.1132	0.1016	0.0369
160	0.0157	0.0083	1.3442	1.1963	0.0085	0.0054
180	0.0406	0.0133	1.0308	1.0885	0.0376	0.0229
200	0.0705	0.0178	0.8194	0.8636	0.0763	0.0194
240	0.0855	0.0364	1.4193	1.3595	0.1179	0.0401
270	0.1015	0.0867	0.7993	0.8275	0.1818	0.0519
300	0.0229	0.0142	1.0198	1.0080	0.0321	0.0169

Table 3: Translational error ratios w.r.t. REF CMH.- Ratio of the means (and standard deviations, respectively) of the absolute translational errors between the tested and the original method. The two map models were tested, namely reflection and exponential decay rate maps, as well as the original particle weighting method (CMH) and the full map posterior one using the explicit measurement likelihoods (ML).

On a more qualitative note, Figure 8 shows a comparison of the occupancy maps obtained through each of the four combinations of map models and particle weighting methods, as computed according to Section 2.1.8. Each of the individual maps can be compared to the Ground Truth map, shown in Figure 9, which was computed using the noise-free poses and measurements from

the simulator via the the Ground Truth Mapper node from Section 2.4. On the other hand, the first column shows the map generated when considering the pose and measurements as noise free, i.e. plain odometry, using the same mapping node, but with the noisy data instead.

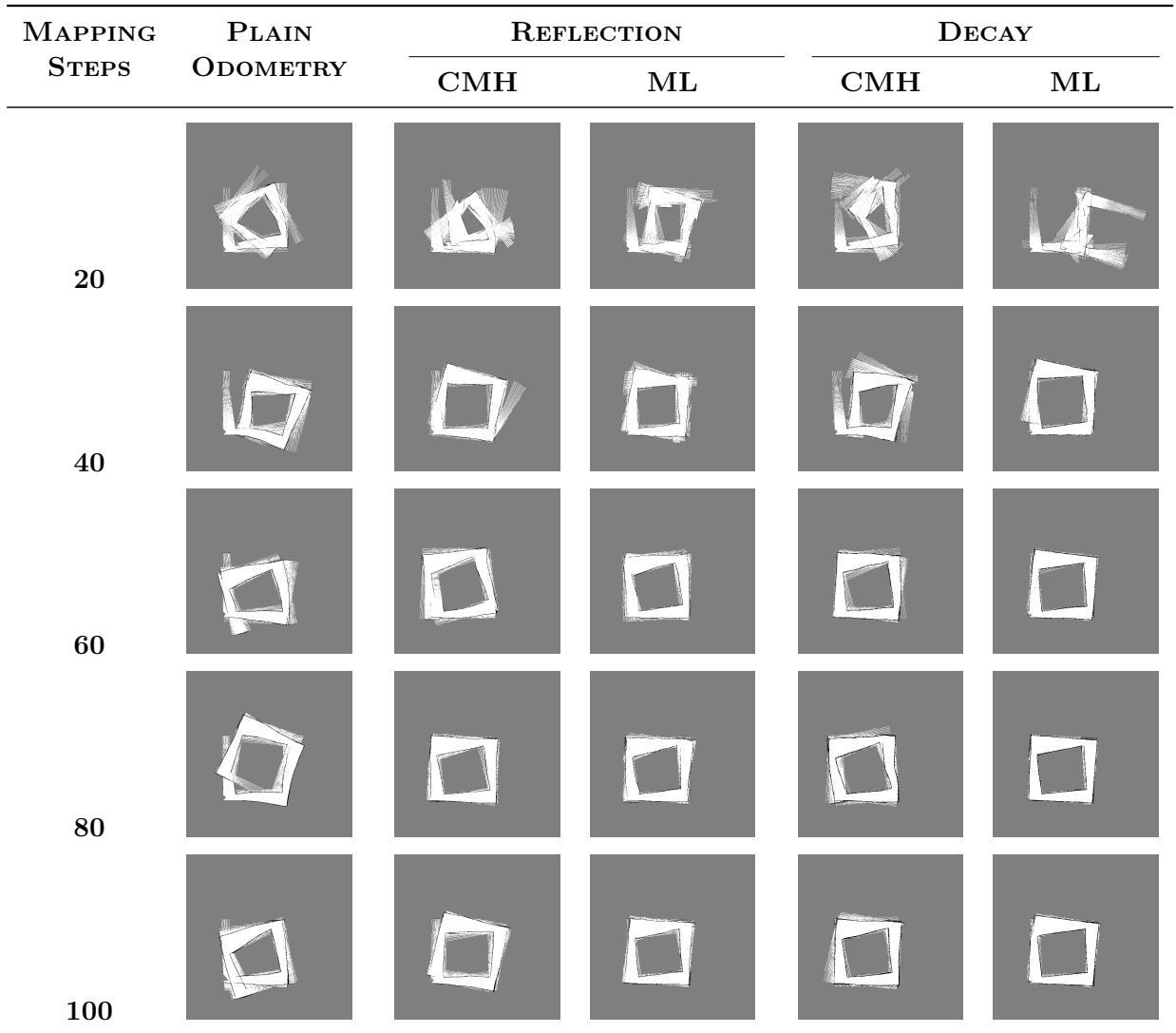


Figure 8: Occupancy Maps.- Generated using the corresponding map model (reflection or exponential decay rate) and particle weighting method (Original[CMH] or full map posterior via Measurement Likelihood [ML]) stated in the column headers, as well as the number of measurements mentioned in the first column of each row.

Full Map Posterior SLAM in ROS

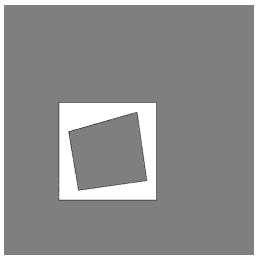
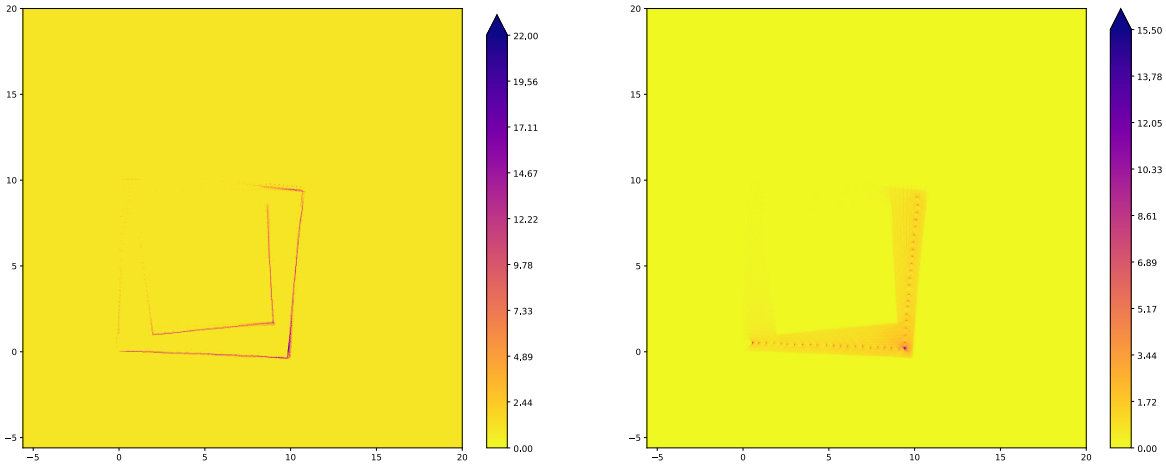


Figure 9: Ground Truth Occupancy Map.

It can be appreciated that the combination of an Exponential Decay Rate Map, together with the Measurement Likelihood particle weighting method, generates a decent enough looking map with as low as 40 measurements in total, which roughly corresponds to making a scan every 1m of travel or so. Not only does the measurement likelihood method improve the efficiency of the Decay maps in terms of number of measurements needed to create a map, but it also helps to create sharper and better defined maps with either map model, as it can also be noticed from the image.

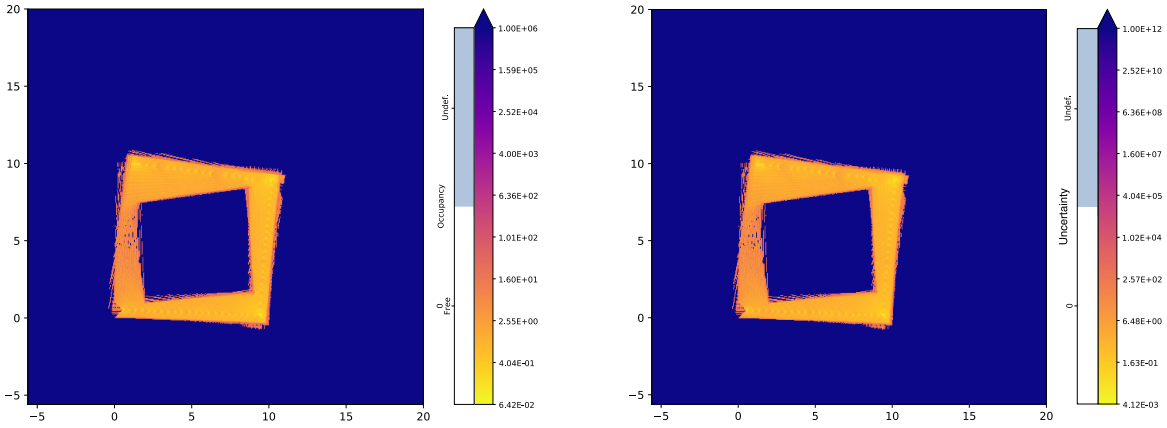
Figures 10, 11 and 12 show an example of the posterior property maps that can be computed using the node from Section 2.6. In this particular case, the generated maps correspond to 100 mapping steps using the Exponential Decay Rate map model and the Measurement Likelihood particle weighting method. As it can be seen, the color values have a logarithmic scale to better appreciate the contrast, as the values tend to go to the extremes, as mentioned in the section detailing its functionality.



(a) Map Posterior α Parameter.

(b) Map Posterior β Parameter.

Figure 10: Posterior Distribution Parameter Maps.



(a) Mean Map. (b) Variance Map.

Figure 11: Map Posterior Moments.

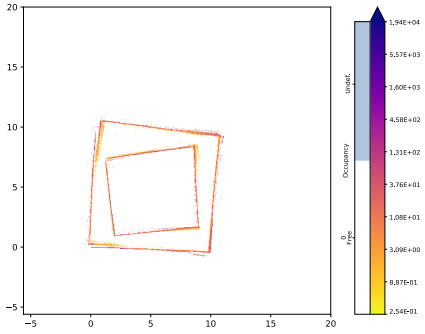


Figure 12: Most Likely Map.

Similar experiments were performed on the converted benchmark datasets, being Building 101 from the University of Freiburg's Technical Faculty the one with the best results for the full posterior map, using the exponential decay rate map model and the measurement likelihood particle weighting method. Due to the mapped space being larger than the simulations, and the sensor having twice as many beams per scan, it was required to lower the map resolution to 10cm, as well as only keep 180 beams and limit the maximum usable range to 10 meters so that the full map posterior algorithm wouldn't become too overconfident and started loosing particle diversity early on. The resulting map of Building 101 using the aforementioned methods and settings can be appreciated in Figure 13.

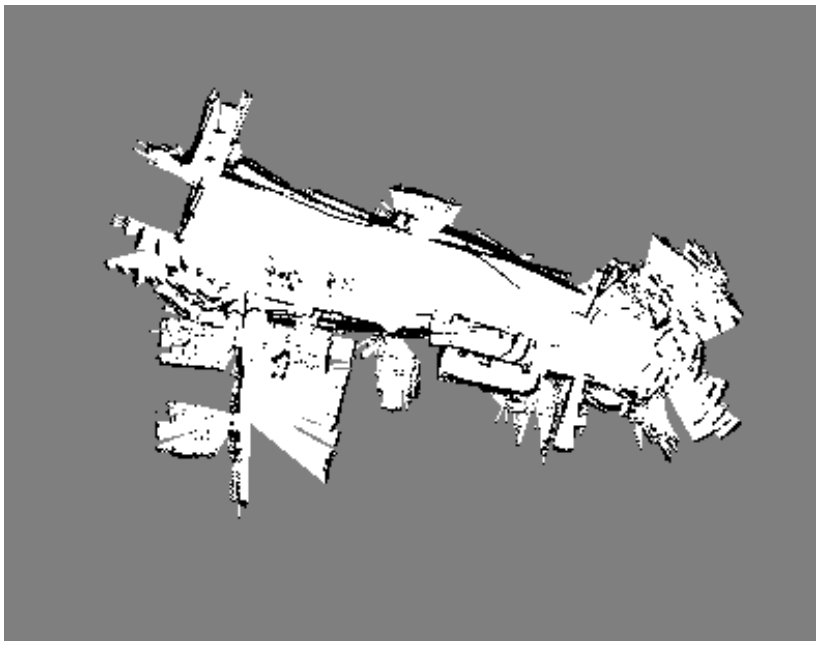


Figure 13: Freiburg's Building 101 Dataset map.- Generated using the proposed full map posterior method with exponential decay rate map model and measurement likelihoods for the particle weights.

5 Discussion

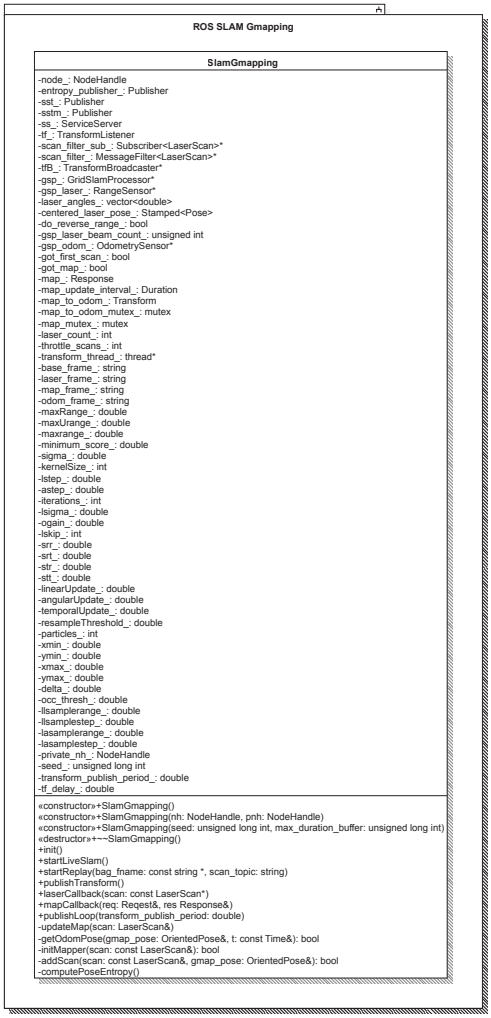
This paper presents a variation of the traditional way of performing simultaneous localization and mapping via particle filters over discretized grid maps, by also supporting the novel exponential decay rate map model, as well as taking advantage of the full map posterior distribution by utilizing proposed measurement likelihoods as the particle weights. As it can be seen in the results section of this document, the proposed method of taking advantage of a map model with more information such as the Exponential Decay Rate, combined with an explicit formulation for the measurement likelihoods, can lead to a faster convergence and smaller error with respect to the estimated poses of the robot. This, however, does not come without some implementation pitfalls. The proposed measurement likelihood method assumes independence of cell states and between all of the measurements, allowing us to evaluate the particle's weight as a product over all cells traversed during a scan. This, however, leads to a very strong dependence on the number of cells as, even for extremely likely cell values, the likelihood of an entire scan tends rapidly towards zero as either the sensor's maximum range or the map's resolution increase. Even further, for the Exponential Decay Rate map model, the values quickly diverge to either zero, or ever larger positive values based on the same causes. This in turn leads to potential numerical stability problems in either map model, and overconfidence and particle deprivation for the Decay Rate maps. Although certain tricks, such as the ones described in this document, help mitigate this problems, it would be interesting to experiment with different ways of evaluating the measurement likelihoods for assigning the weights

to each of the particles in the future.

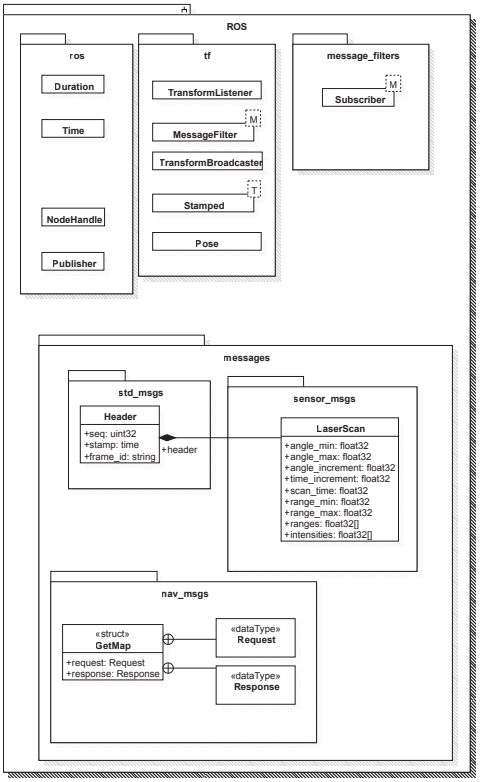
Unfortunately this additional information and accuracy also comes at a cost. The original implementation of *GMapping* embraced the idea of FastSLAM v2.0, particularly in terms of being as efficient and resource saving as possible. Since the newly proposed Measurement Likelihood method takes into account every cell traversed by each beam, instead of only the endpoints as in the original methodology, it can be orders of magnitude more computationally expensive than its previous counterpart. Similarly, to get the best results, a complete line cover algorithm had to be used, instead of the faster and more efficient, yet incomplete, Bresenham's method. This factor needs to be taken into consideration when choosing between the two weighting approaches for real time applications. As future work, it would be good to benchmark the performance of all the methods in terms of time, processing and memory requirements to be able to make a more informed choice regarding the suitability of each algorithm for any given application.

In summary, this document describes a probabilistically sound, albeit slightly more expensive, method that results in an improved accuracy over previous approaches for determining the robot's pose and generating maps of the environment, providing the user with a choice between better position and map estimates by using the modified framework, or the resource efficiency of the original library.

A Reverse Engineering Diagrams



(a) GMapping ROS Wrapper Class



(b) ROS Classes

Figure 14: Class Diagrams.

Full Map Posterior SLAM in ROS

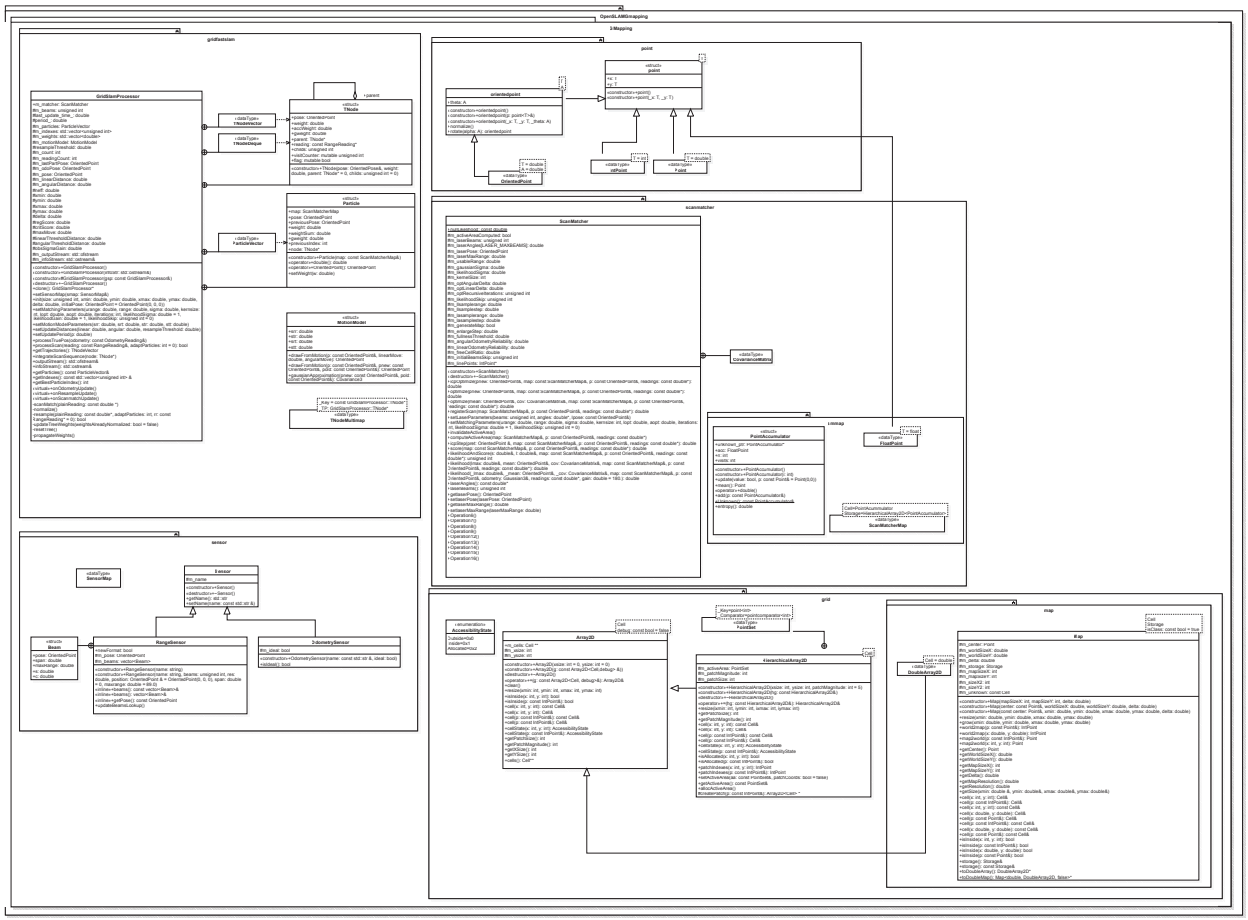


Figure 15: GMapping Classes

Full Map Posterior SLAM in ROS

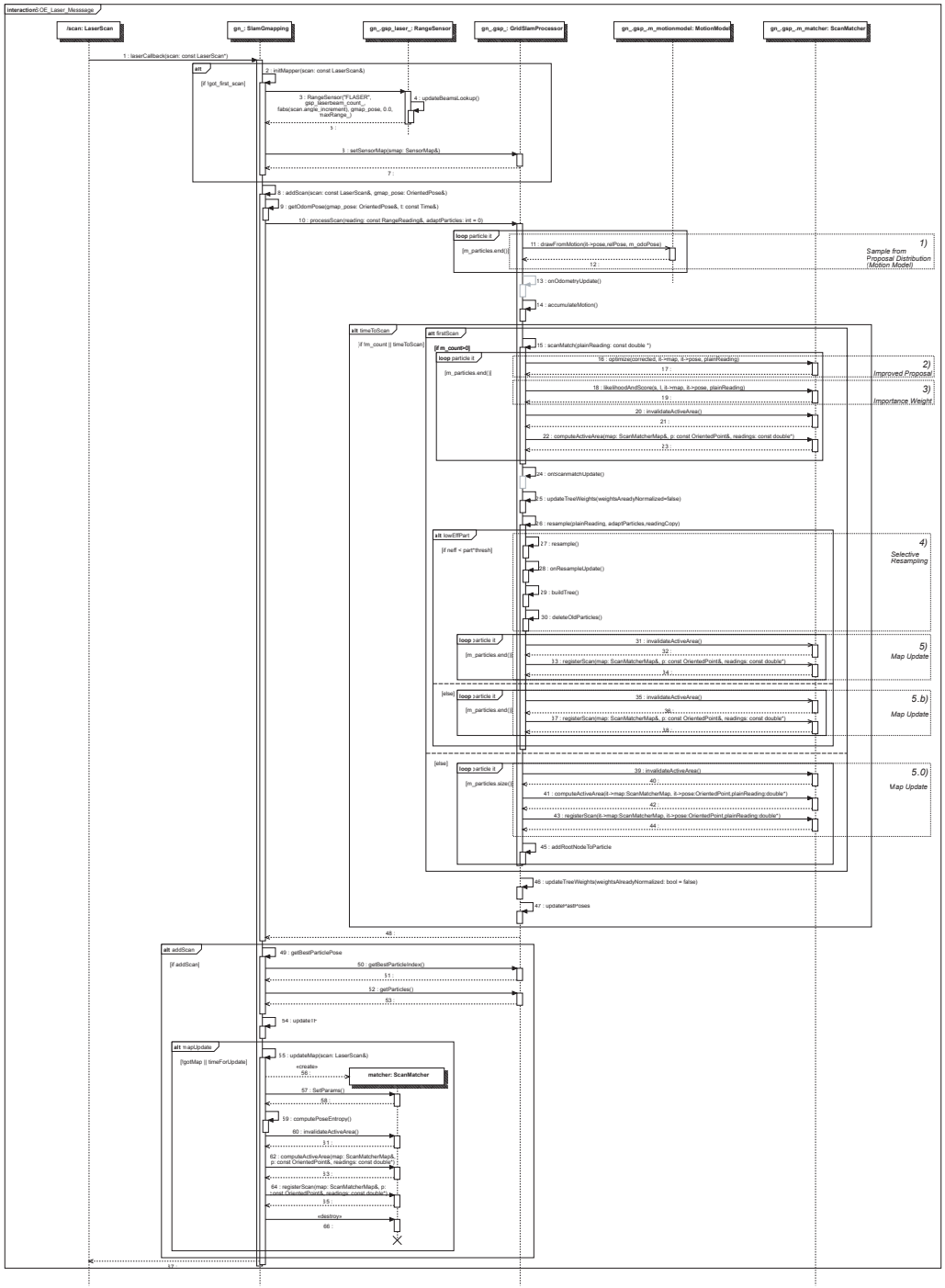


Figure 16: Laser Scan Message Sequence of Events.

References

[1] F. Amigoni, S. Gasparini, and M. Gini. Good experimental methodologies for robotic mapping: A proposal. pages 4176 – 4181, 05 2007.

[2] E. Andres, P. Nehlig, and J. Françon. Supercover of straight lines, planes and triangles. In E. Ahronovitz and C. Fiorio, editors, *Discrete Geometry for Computer Imagery*, pages 243–254, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[3] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965.

[4] E. Dedu. Bresenham-based supercover line algorithm. `http://eugen.dedu.free.fr/projects/bresenham/`, 2001.

[5] J. Deray. Carmen to rosbag converter. `https://github.com/artivis/carmen_publisher`, 2015.

[6] A. Elfes. *Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation*. Carnegie Mellon University, 1989.

[7] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[8] G. Grisetti, C. Stachniss, and W. Burgard. Improving grid-based SLAM with rao- blackwellized particle filters by adaptive proposals and selective resampling. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 2443–2448, Barcelona, Spain, 2005.

[9] G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46, 2007.

[10] R. Kümmerle, B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, and A. Kleiner. On measuring the accuracy of SLAM algorithms. *Journal of Autonomous Robots*, 27(4):387–407, 2009.

[11] L. Luft, A. Schaefer, T. Schubert, and W. Burgard. Closed-form full map posteriors for robot localization with lidar sensors. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep 2017.

[12] J. Rao and A. Sengupta. *Topics in circular statistics*, volume 5. 01 2001.

[13] A. Schaefer*, L. Luft*, and W. Burgard. An analytical lidar sensor model based on ray path information. *IEEE Robotics and Automation Letters (RA-L)*, 2(3):1405–1412, July 2017.

[14] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. MIT Press, Cambridge, Mass., 2005.

[15] X. Wu. An efficient antialiasing technique. *SIGGRAPH Comput. Graph.*, 25(4):143–152, July 1991.