Simultaneous Localization and Mapping(SLAM)

Elerson Rubens da S. Santos

December 2, 2016

1 Introduction

With the development of computers, manipulators and sensing devices, today we are able to build fully capable robots. These robots can be useful in search and rescue operations, allowing the exploration of in inhospitable environments, where it could be dangerous for human rescue teams.

Considering robotics exploration, SLAM (*Simultaneous Localization and Mapping problem*) can be considered one of major problems to be solved. The problem aims to map an unknown environment given only odometry, which represents the robot movement, and sensing data, which usually comes from a laser or a sonar. Moreover, this problem deals with the chicken and egg dilemma, to build a map a localization is required and to have a localization a map is required.

This document studies Fast SLAM[], a probabilistic solution for the SLAM problem. Fast SLAM is a particles filter that, in overview, works as follows. Each particle represents a pose(position and orientation) in the environment and has its own map. The update of the particles pose is done using a motion model, which considers the robot mobile dynamics and the odometry. The particles are sampled considering the likelihood of the ranger measurements and the map. The map is only updated for the new sampled particles. This process occurs in a loop as long the robot is running.

This document is divided as follows. Section 2 presents the motion model, which describes how the particle poses are updated. Following the sensor likelihood and the map update model. Next, the results and a conclusion are presented.

2 SLAM Particle Filter

The SLAM problem seeks the posterior $p(x_{1:t}, m|z_{1:t}, u_{1:t})$, where $x_{1:t}$ represents the path along the environment, m is the map, $z_{1:t}$ are the measurements of a laser (or other sensor), $u_{1:t}$ is the control. One simple formulation to solve this problem is by using a particle filter, estimating the above probability using $p(x_t) \propto p(x_t|x_{t-1}, u_t)bel(x_{t-1})$, $p(z|m, x_t)$ and $p(m|x_t, z_t)$, which represents the

motion model estimation, measurement model estimation and map estimation, respectively.

A generic particle filter designed to do SLAM is presented in algorithm 5. To each particle, a new pose is sampled given the dynamics of the robot and the odometry displacement. Next, a measurement model is performed using the current map for the particle, the sampled pose and the sensor measurements. This gives the weights for the particles. Following, the map is updated using the sensor measurement, and the pose. The last step is to sample the particles given the weight obtained in the scan matching procedure. In next sections, the motion model estimation, measurement model estimation and map estimation are presented.

Algorithm 1 SLAM - Generic Algorithm

1: **procedure** SLAM(χ_{t-1}, μ, z) ▷
2: $\chi_t \leftarrow \emptyset$
3: **for** $k = 1$ **to** M **do**
4: $x_t^{[k]} \leftarrow$ sample_motion_model(μ_t, x_{t-1})
5: $w_t^{[k]} \leftarrow$ measurement_model_map($z_t, x_t^{[k]}, m_{t-1}^{[k]}$)
6: $m_t^{[k]} \leftarrow$ updated_occupancy_grid ($z_t, x_t^{[k]}, m_{t-1}^{[k]}$)
7: **for** $k = 1$ **to** M **do**
8: draw i with probability $\propto w_t^{[i]}$
9: add $x_t^{[i]}, m_t^{[i]}$ to χ_t
10: **return** b ▷

2.1 Motion Model

The motion of a non holonomic robot from pose x_{t-1} to x_t can be described by rotation δ_{rot1}, followed by a translation δ_{trans} and other rotation δ_{rot2}. The second rotation is need to account noise in translation and rotation.

The motion model that describes this movements, considers that each of that movements may suffer from noise, and this noise can be probabilistically modeled. This is presented in the algorithm 2. First δ_{rot1} is estimated as the difference of the angle of the movement and the current robot heading. Next the translation δ_{trans} is computed, following the remaining rotation between poses δ_{rot2}. New values of rotations($\overline{\delta}_{rot1}$,$\overline{\delta}_{rot2}$) and translation $\overline{\delta}_{trans}$ are sampled using a Gaussian distribution with 0 mean. Finally, using the new sampled rotations and translation, the new poses are calculated.

Algorithm 2 sample_motion_model

procedure SAMPLE_MOTION_MODEL($(x_t, y_t, \theta_t), (x_{t-1}, y_{t-1}, \theta_{t-1}), \boldsymbol{\alpha}$) ▷

$\delta_{rot1} \leftarrow atan2(y_t - y_{t-1}, x_t - x_{t-1}) - \theta_t$

$\delta_{trans} \leftarrow \sqrt{(y_t - y_{t-1})^2 + (x_t - x_{t-1})^2}$

$\delta_{rot2} \leftarrow \theta_{t-1} - \theta_t - \delta_{rot1}$

$\overline{\delta}_{rot1} \leftarrow \delta_{rot1} - \mathcal{N}(0, \alpha_1|\delta_{rot1}| + \alpha_2\delta_{trans})$

$\overline{\delta}_{trans} \leftarrow \delta_{rot1} - \mathcal{N}(0, \alpha_3\delta_{trans} + \alpha_4(|\delta_{rot1}| + |\delta_{rot2}|)$

$\overline{\delta}_{rot2} \leftarrow \delta_{rot1} - \mathcal{N}(0, \alpha_1|\delta_{rot1}| + \alpha_2\delta_{trans})$

$x \leftarrow x_{t-1} + \overline{\delta}_{trans} cos(\theta + \overline{\delta}_{rot1})$

$y \leftarrow y_{t-1} + \overline{\delta}_{trans} sin(\theta + \overline{\delta}_{rot1})$

$\theta \leftarrow \theta + \overline{\delta}_{rot1} + \overline{\delta}_{rot2}$

return (x, y, θ) ▷

2.2 Measurement Model

The measurement model is responsible for determine the probability of a measurement $z_t^{[k]}$ given a pose x_t and the map m, which is can be represented as $p(z_t^{[k]}|x_t, m)$, considering that a laser has k measurement beams.

This probability can be determined by matching the scan with the map in a given pose. Here it used a simple model that counts the number of measurements falling in obstacles lookup table. This lookup table is constructed considering the position where the laser detects an obstacle. A new measurement represents a good fit when it falls in in most of the lookup table filled positions. Therefore the model is computed as follows:

$$p(z_t^{[k]}|x_t, m) = exp((-m/M) * \gamma) \tag{1}$$

, where m is the number of measurements that fell in a filled position in the lookup table, M is the number of sensor measurements(constant), and γ is constant.

The lookup table is updated together with the map, considering the most recent obstacles that were detected.

2.3 Occupancy Grid

A occupancy grid represents a map as a grid of positions, representing $p(m|z_{1:t}, x_{1:t}) = \prod p(\boldsymbol{m_i}|z_{1:t}, x_{1:t})$, where m_i is the grid cell with index i. With this factorization, we can compute the occupancy grid for each cell. Also, the algorithm uses *log odds* to represent the occupancy, this is a common representation to avoid numerical instabilities. Therefore, the representation of a grid $l_{t,i}$ in time t and position i is:

$$l_{t,i} = \frac{log(p(m_i|z_{1:t}, x_{1:t}))}{1 - p(m_i|z_{1:t}, x_{1:t})}$$

To get the probability for a given position, we can use the following:

$$p(\boldsymbol{m_i}|z_{1:t}, x_{1:t}) = 1 - \frac{1}{1 + exp(l_{t,i})}$$

The algorithm 3 shows the computation of the occupancy grid given the current measurements $z_{t,sensed}$ and pose x_t. The algorithms tries to update all grid cells that are in the perception area of the robot. When a grid is in that perception area, the inverse model (Algorithm 4), which computes $p(\boldsymbol{m_i}|z_{1:t}, x_{1:t})$ is called.

Algorithm 3 Occupancy Grid Mapping

```
1: procedure OCCUPANCY_GRID_MAPPING(map, x_t, z_t)
2:     for all cells map do
3:         if m_i is in the perception field of z_t then
4:             m_i ← m_i + inverse_sensor_model(m_i, x_t, z_t) − l_0
5:     return b                                                    ▷
```

The inverse model algorithm first gets the center of mass of the grid, and using the pose $x_t = (x, y, \theta)$, computes the distance r from the cell to the robot, the angle ϕ between the robot and the cell, which is used to determine the beam k that is passing through the cell.

Algorithm 4 Inverse Sensor Model

```
1: procedure INVERSE_MODEL(m, x_t, z_t)
2:     Let x_i, y_i be the center of mass of m_i
3:     r = √((x_i − x)^2 + (y_i − y)^2)
4:     φ = atan2(y_i − y, x_i − x) − θ
5:     k = argmin_j |φ − θ_{j,sens}|
6:     if r > min(z_max, z_t^k + α/2) or |φ − θ_{k,sens}| > β/2) then
7:         return l_0
8:     if z_t^k < z_max and |r − z_max| < α/2 then
9:         return l_occ
10:    if r < z_t^k then
11:        return l_free
```

The algorithm will return the values l_0, which is the value for unknown grid cell, when distance r between the grid cell and the robot is higher than minimum from the real measurement z_t^k and z_{max}(the maximal distance sensed by the sensor). Also, the algorithm return l_0 when the difference between angles ϕ and $\theta_{k,sens}$ is higher then the constant $\beta/2$, this account the laser width.

It returns occupied if the laser measurement at position k is less than z_{max}, which means that the laser hit some object. To return that a position is free, the cell distance has to be less than laser measurement, meaning that the laser passed through it.

2.4 Improved SLAM algorithm - Gmapping

Here it is shown Gmapping, the implemented algorithm, which gives better results that the one presented in the SLAM generic algorithm.

The algorithm adds a step while computing the a new particle. This considers that the most expensive part in the algorithm is the update of the occupancy grid, which forbids the addition of many particles in the filter.

In this sense, the Gmapping algorithm adds new particles only in the perception/motion models of the algorithm (perception particles). The best particle from this part is used to update the occupancy grid(map particles).

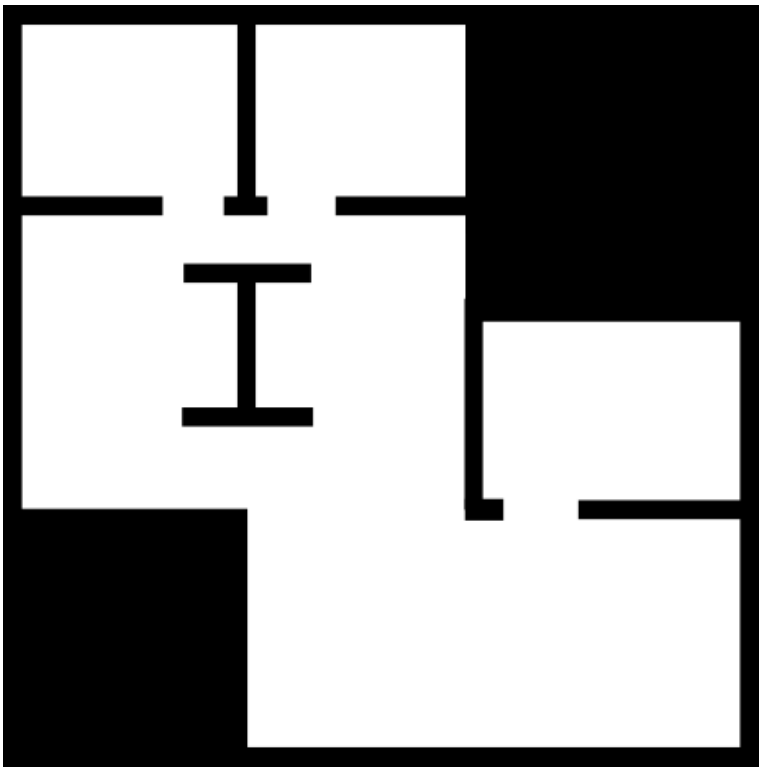
Algorithm 5 Gmapping

procedure SLAM(χ_{t-1}, μ, z) ▷
  $\chi_t \leftarrow \emptyset$
  for $k = 1$ **to** M **do**
    $Best\_particle \leftarrow \{\emptyset, \emptyset\}$
    for $p = 1$ **to** P **do**
      $xp_t^{[p]} \leftarrow$ sample_motion_model(μ_t, x_{t-1})
      $wp_t^{[p]} \leftarrow$ measurement_model_map($z_t, x_t^{[k]}, m_{t-1}^{[k]}$)
      $Best\_particle \leftarrow SelectBestParticle(xp_t^{[p]}, wp_t^{[p]}, Best\_particle)$
    $x_t^{[k]} \leftarrow Best\_particle[1]$
    $w_t^{[k]} \leftarrow Best\_particle[2]$
    $m_t^{[k]} \leftarrow$ updated_occupancy_grid $(z_t, x_t^{[k]}, m_{t-1}^{[k]})$
  for $k = 1$ **to** M **do**
    draw i with probability $\propto w_t^{[i]}$
    add $x_t^{[i]}, m_t^{[i]}$ to χ_t
  return b ▷

3 Experiments

The experiments were executed using ROS and Stage. ROS is a widely used robotics framework, providing code implementations, simulators, and other facilities. Stage is one of the 2d simulators present in ROS, allowing accurate simulations of actuators, sensors, map objects. The SLAM algorithm was implemented using R language.

Figure 1: Simulated Environment



To run the experiments the map presented in Figure 1 was artificially created, and all data coming from the robot(lasers, odometry, ground truth) were collected in 1 execution of the stage simulator. The expected map from the SLAM algorithm is presented in Figure 2. To generate this figure, all parameters from the SLAM algorithm were set to consider the ground truth position from the collected data.

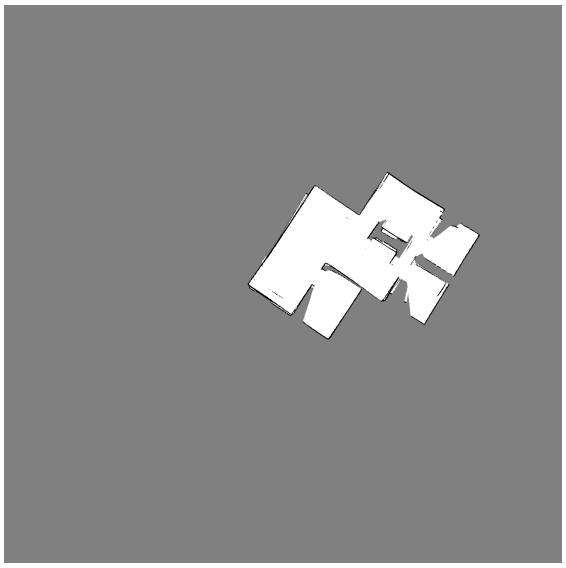
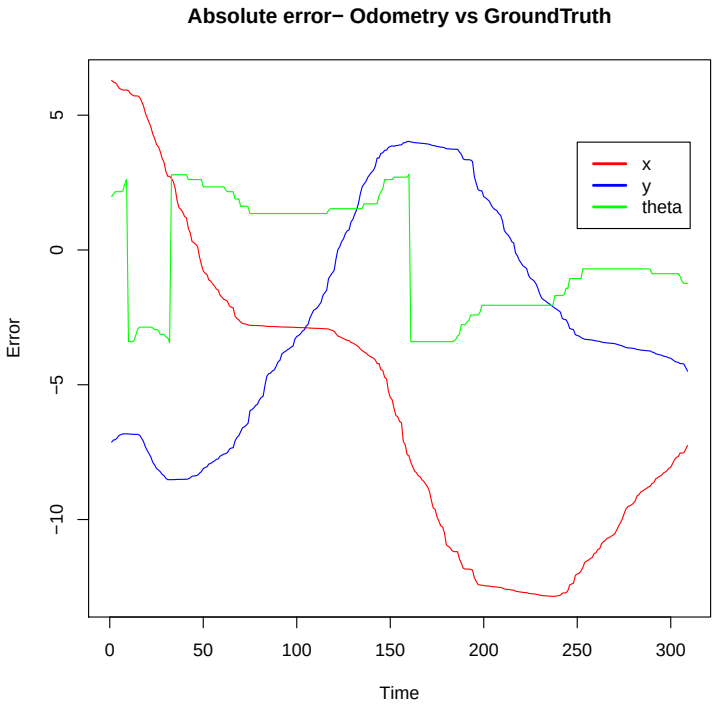


Figure 2: Expected SLAM output

Figure 3 shows the absolute difference between the odometry and the ground truth. This shows that there is noise in the odometry data, which makes a SLAM algorithm needed to compute a map.

Figure 3: Absolute Error

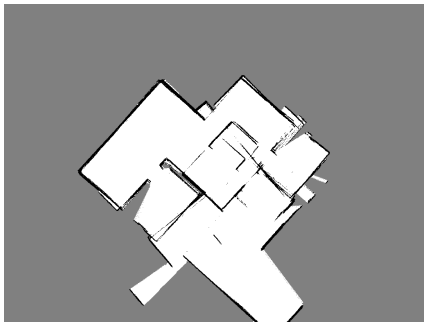


Three sets of experiments were executed, considering 10, 20, 40 map particles each. In each set of experiments, the number of perception particles were 50, 100, 200. Figures 4,5,6 show this results. In the experiments containing 10 map particles and 50 perception particles it is possible to see a huge difference to the expected map. For the others, visually, the experiments containing 200 perception particles presents the best results. However, it is possible to see an error on top of the map, this error can be fixed using new step in the SLAM algorithm, a loop closure, allowing the SLAM to fit the best way possible new data from the sensors and the positions already mapped.

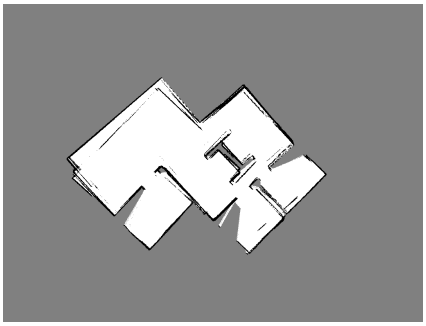
Table 1 shows the average execution time for all experiments. As expected, the execution time changes more when increasing map particles, which uses the most computation resources in the algorithm. It also shows that the implemented algorithm can run in realtime, as all map was created in under 10 seconds.

Table 1: Execution time

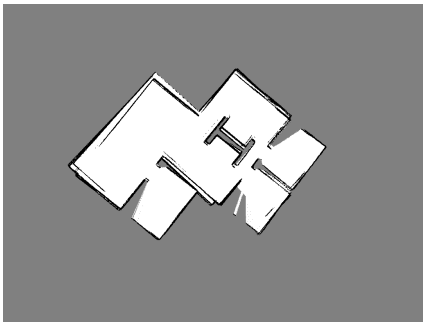
Map particles	Perception particles	Execution Time(seconds)
10	50	2.43
10	100	2.67
10	200	3.22
20	50	3.11
20	100	3.62
20	200	4.30
40	50	4.76
40	100	5.53
40	200	7.36



(a) 50 perception particles

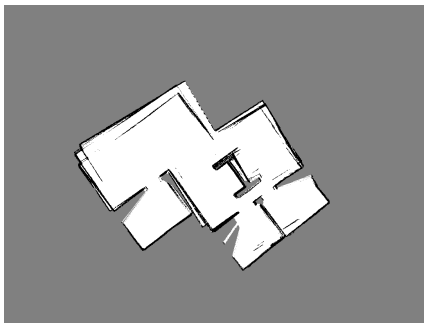


(b) 100 perception particles

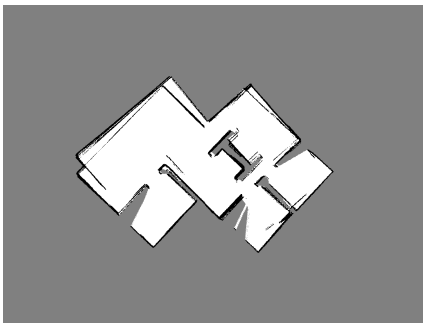


(c) 200 perception particles

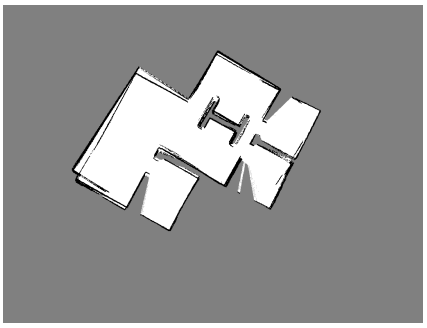
Figure 4: plots of 10 map particles



(a) 50 perception particles

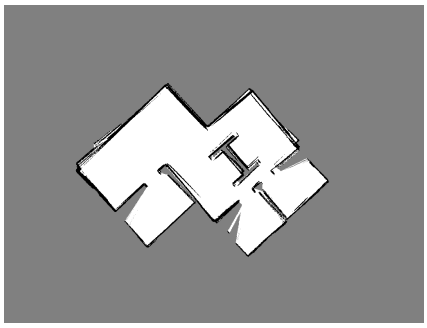


(b) 100 perception particles

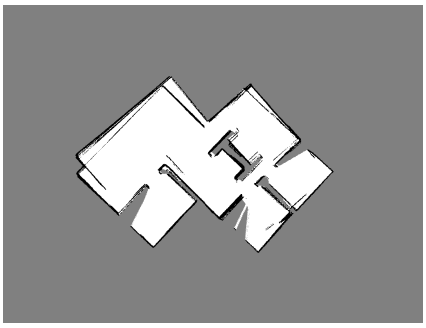


(c) 200 perception particles

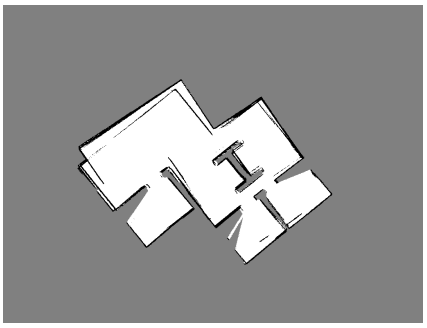
Figure 5: plots of 20 map particles



(a) 50 perception particles



(b) 100 perception particles



(c) 200 perception particles

Figure 6: plots of 40 map particles

4 Conclusion

This document presented a SLAM algorithm, which was implemented in R, and tested using simulation data. The experiments showed that the implementation worked well, as the map obtained by the SLAM algorithm and the ground truth were similar.

One of the possible improvements in the current implementation is the use of loop closure algorithm, allowing the map to close better. Other important improvement would be a better algorithm for the scan matching, which cold allow the reduction of some particles in the algorithm and, therefore, improve the performance of the algorithm. Also, a quantitative metric could be used to compare results.

Bibliography

Sebastian Thrun, Wolfram Burgard, and Dieter Fox. 2005. Probabilistic Robotics (Intelligent Robotics and Autonomous Agents). The MIT Press.

G. Grisetti and C. Stachniss and W. Burgard. "Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters".IEEE Transactions on Robotics, feb, 2007.