SLAM with Particle Filter for Point Featured Map

Report Author:
John Dahlberg
Project Partner:
Rodrigue Bonnevie

***Abstract*—The great challenge of Simultaneous Localization and Mapping, where the first is dependent on the latter and vice versa, is an important problem to solve for instance in case of robots exploring unknown environments. In this paper, it is implemented with a particle filter in a simulation where a LIDAR equipped robot is performing exploration within a map containing walls and point featured landmarks. While tracking its pose, the landmarks and walls are to be sketched in an estimated map. The implementation shows that the localization and mapping is performing with good, but not sufficient, accuracy in real time.**

I. Introduction

THE problem of *Simultaneous Localization and Mapping*, or SLAM, is much more challenging than localization or mapping problems alone, where either the location or map is known, since they are directly dependent on each other. As the name suggests, SLAM provides the surrounding map that has been discovered so far as well as the location of an agent, typically a robot, within the map. The domains of implementations are many. These includes robots indoor, outdoor, underwater and aerial systems.[1].

There are several different approaches of SLAM, such as with Extended Kalman Filter (EKF), with Occupancy Grid or with Particle Filter to give a few examples. A version of the latter is developed and covered in this paper. There are some great strengths with Particle Filters in contrast to for instance EKF. One example is that multimodal hypothesis representation is possible which is very convinient for global localization in an environment with a great deals of symmetries, such as rooms. A drawback, nevertheless, is the computational demand of many particles. Another is that the particle representation only is an approximation of a probability distribution unlike EKF, and the true robot pose may be lost in the representation over time. In this implementation, though, quite few particles are needed for tracking the robot and the extent of run time is short.

The defined map for testing out the SLAM solution for a robot is depicted in Figure 1. On top of the blue walls 13 uniquely identifiable point featured landmarks are placed. The walls and landmarks are detectable by a LIDAR on the robot.

Two goals of this implementation are defined, in addition to provide a working concept of a Particle Filter SLAM solution. The first is to map and localize the robot with an accuracy that ensures the robot estimate to be kept within the walls of the map, as well as that the true robot pose is kept within the estimated map. The other goal is the algorithm can run in real-time.

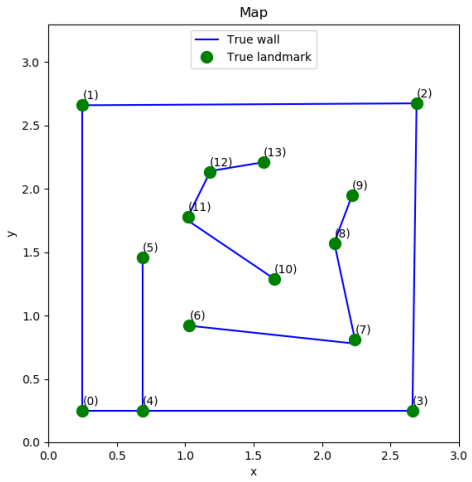


Fig. 1. The map point featured map used for the implementation.

II. Contribution

The implementation has proposed a SLAM solution with a particle filter for a point featured map. It has shown to successfully be working with a reasonable but not sufficient result given the defined map as well as the measurement and odometry simulation. With appropriately tuned parameters, this if fulfilled in the sense of

- Accuracy and
- Computational performance.

Considering accuracy, the difference between the true trajectory and map respectively the estimated ones are mostly but not always kept to be sufficiently small so that the estimated trajectory is not interfering with the true wall, and similarly that the true trajectory does not intersect with the estimated walls. The accuracy will be further specified in Section VII. On the latter point i.e. computational performance, the implementation is able to, at least for reasonable capacity of CPUs in most laptops, run in real time. This is necessary for a real-world application.

III. Outline

In next Section IV related work will discussed including different approaches of SLAM with Particle filter and other SLAM solutions. Thereafter in Section V the method will be described into details in the main algorithm Section, V-A, and

the use of a occupancy grid in Section V-B. Next the implementation is explained in Section VI providing an illustrating figure and video of the implementation. After that in Section VII the main results of the simulation are stated. Finally, in Section VIII, a summary is given of the paper together with some discussion of conclutions.

IV. Related work

Solutions of SLAM has been very successful last decades with a great deal of different approaches. It is now considered as a solved problem.[1]. The earliest approach of which has had an important impact is the SLAM with the Extended Kalman Filter, or EKF. The algorithm is thoroughly described by Thrun et al. [2]. The maps in SLAM with EKF are based on point featured landmarks as in Figure 1. The two algorithms of EKF SLAM provided by Thrun et al. are both managing known respectively unknown correspondence, that is, with and without assumption of unique identification of measurements from the landmarks. The problem of data association is further covered by Whyte et al.[3] On top of that, SLAM withouth predetermined landmarks is covered by Eliazar et al.[4] Since the EKF is based on Gaussian distributed noise, it scales well with dimensionality. Nevertheless it cannot represent multimodal distributions. Another common approach of SLAM is Particle Filter as used in this paper. Particle samples can handle multimodal representations, but scales very badly with number of dimensions. This is due to the curse of dimensionality, that is, number of particles needed scales exponentially with number of dimensions, and is for that reason not very efficient. Further on, particle samples are only describing an approximation of the distribution. A solution of this is using Rao–Blackwellized particle filters to marginalize the distribution. See algorithm by Särkkä [5]. A trick to marginalize out a subset and represent it with gaussian distribution distribution rather than a joint distribution has been developed by Doucet and Andrieu [6] which is based on the Rao–Blackwellization method that Gelfand and Smith introduced [7]. Grisetti et al for instance use the Rao–Blackwellized particle filters for SLAM. [8]. Another approach for SLAM is with occupancy grid maps. An algorithm of this is provided and discussed by Thrun et al. [9]. A problem with these is that occupancy grid contains a large set of cells, and each particle has its own set meaning that the size of particles is very restricted. A solution of this is to use efficient data structures, such as Quadtrees or Octtrees. Representing occupancy grids with Quadtrees has been investigated by Li and Ruichek.[10]

V. Method

This section will describe the algorithm in details. The algorithm is based on measurements from the true pose of the robot. These are LIDAR distance and bearing observations that are reflected by the walls as well as the landmarks that can be seen, i.e. not blocked by walls.

A. Particle Filter SLAM algorithm

The algorithm of the SLAM with particle filter implementation is roughly summarized through the flowchart in Figure 2 to provide an overview. Now, the algorithm starts with

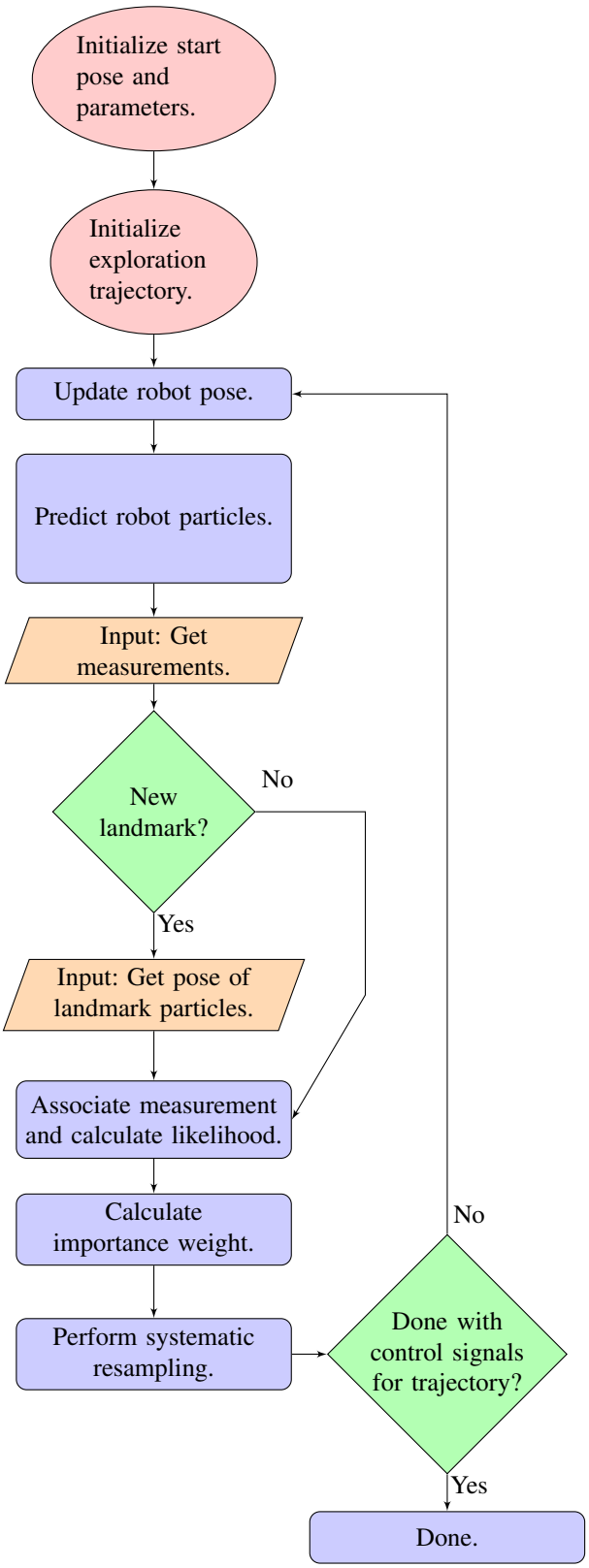


Fig. 2. Flowchart overview of the SLAM with Particle Filter algorithm.

1) Initialization of parameters. These includes the process

noise co-variance matrix

$$R = 1.3 \begin{pmatrix} 1 \times 10^{-2} & 0 & 0 \\ 0 & 1 \times 10^{-2} & 0 \\ 0 & 0 & 5 \times 10^{-2} \end{pmatrix}, \quad (1)$$

the measurement noise co-variance matrix

$$Q = 5 \begin{pmatrix} 1 \times 10^{-2} & 0 \\ 0 & 1 \times 10^{-2} \end{pmatrix}, \quad (2)$$

the map resampling noise

$$Q_w = \begin{pmatrix} 1 \times 10^{-2} & 0 \\ 0 & 1 \times 10^{-2} \end{pmatrix}, \quad (3)$$

and the threshold $\lambda_\psi = 1 \times 10^{-8}$ for ignoring measurements outlier. Finally the start pose of the robot $\boldsymbol{x_0}$ is initialized, in this case it is set to $\begin{pmatrix} 0.5 & 0.5 & \frac{\pi}{2} \end{pmatrix}^T$. The robot particles

$$\begin{aligned} \boldsymbol{S_t} &= \left\{ \boldsymbol{s_t^1}, \boldsymbol{s_t^2} \cdots \boldsymbol{s_t^M} \right\} \\ &= \left\{ \begin{pmatrix} s^1_{x_t} \\ s^1_{y_t} \\ s^1_{\theta_t} \end{pmatrix}, \begin{pmatrix} s^2_{x_t} \\ s^2_{y_t} \\ s^2_{\theta_t} \end{pmatrix}, \cdots, \begin{pmatrix} s^M_{x_t} \\ s^M_{y_t} \\ s^M_{\theta_t} \end{pmatrix} \right\} \end{aligned} \quad (4)$$

at any time t are all initialized with the same pose as the robot, i.e. $s_0^i \equiv \boldsymbol{x_0} \forall i \in [0, M]$. Note however that no map is yet defined. Only an arbitrary coordinate system that the initialized robot pose is set with respect to. Since the initial robot pose is completely certain in the frame as it is a pure definition, all particles are justified to be initialized with this pose. Of that reason, the localization from here i a tracking problem. The use of these parameters will gradually be covered in this section.

2) The trajectory through the map is initialized for exploration. It should give coverage of all parts of the map for complete mapping, see Figure 3. The trajectory is made by the robot start pose $\boldsymbol{x_0}$ and a set of control signals $\{v_t, \omega_t\}$, $\forall t$, corresponding to linear and angular velocity at any time. In a real-world robotics problem, SLAM is performed to discover the map, consequently the trajectory cannot be defined in advance. Nevertheless exploration algorithms is not a focus in this paper and not included in the implementation, why the trajectory is assumed to be known.

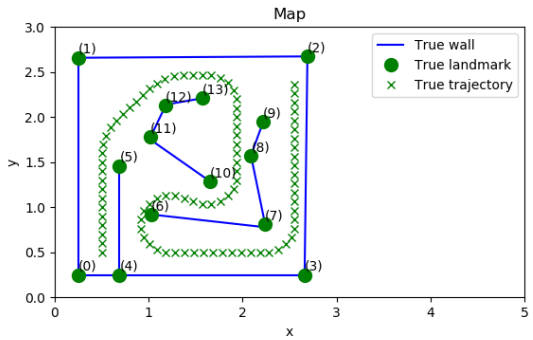


Fig. 3. The predefined trajectory of the map. It allows coverage of the complete map.

3) The robot pose $\boldsymbol{x_t} = \begin{pmatrix} x_t & y_t & \theta_t \end{pmatrix}^T, \forall t$ is updated based on predetermined control signals that specifies linear and angular velocity, v_t and ω_t, respectively. The updated states are given by

$$\boldsymbol{x_{t+1}} = \boldsymbol{x_t} + \boldsymbol{u_t}\Delta_t = \begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix} + \begin{pmatrix} v_t cos(\theta_t)\Delta_t \\ v_t sin(\theta_t)\Delta_t \\ \omega_t \Delta_t \end{pmatrix} \quad (5)$$

where $\Delta_t \equiv 0.1$s is timestep between two iterations.

4) The robot particles $\boldsymbol{S_t}$ used for representing the approximate probability distribution of the robot are predicted almost in the same fashion as the robot motion update, i.e., equation (5). The difference is that some disturbances is added to simulate imperfect odometry, specifically,

$$\boldsymbol{S_{t+1}} = \boldsymbol{S_t} + \boldsymbol{u_t}\Delta_t + \boldsymbol{n_t}, \quad (6)$$

where

$$\begin{aligned} \boldsymbol{n_t} &= \begin{pmatrix} r \sim \mathcal{N}(b_1, R_{1,1}) \\ r \sim \mathcal{N}(b_2, R_{2,2}) \\ r \sim \mathcal{N}(b_3, R_{3,3}) \end{pmatrix} \\ &= \begin{pmatrix} r \sim \mathcal{N}(0, 1.3 \times 10^{-2}) \\ r \sim \mathcal{N}(1 \times 10^{-3}, 1.3 \times 10^{-2}) \\ r \sim \mathcal{N}(1 \times 10^{-2}, 1.3 \times 5 \times 10^{-2}) \end{pmatrix} \end{aligned} \quad (7)$$

is random Gaussian noise with biases b_1, b_2, b_3 and variance corresponding to process noise R. The choice of the noise free motion model and prediction disturbances will be further explained in Section VI.

5) The set of measurements inputs

$$\begin{aligned} \boldsymbol{M_t} &= \left\{ \boldsymbol{m_t^1}, \boldsymbol{m_t^2} \cdots \boldsymbol{m_t^{n_{cs}}} \right\} \\ &= \left\{ \begin{pmatrix} r_t^1 \\ \alpha_t^1 \end{pmatrix}, \begin{pmatrix} r_t^2 \\ \alpha_t^2 \end{pmatrix}, \cdots, \begin{pmatrix} r_t^{n_{cs}} \\ \alpha_t^{n_{cs}} \end{pmatrix} \right\} \end{aligned} \quad (8)$$

are now simulated and received, where r_t^i and α_t^i is the distance and bearing, respectively, to landmark $i \in [0, n_{cs}]$ and $n_{cs} \in [0, n]$ is the number of currently seen landmarks of all the existing n landmarks. Measurements of landmarks are provided with unique identification of the landmarks, with the motivation of keeping it simple and less computationally heavy, as opposed to associating landmarks with unknown correspondences by calculating maximum likelihoods. The generated measurements

$$\boldsymbol{m_t^i} = \bar{\boldsymbol{m}}_t^{\boldsymbol{i}} + \boldsymbol{n_t^m} = \begin{pmatrix} \bar{r}_t^i \\ \bar{\alpha}_t^i \end{pmatrix} + \begin{pmatrix} r \sim \mathcal{N}(0, \sigma) \\ r \sim \mathcal{N}(0, \sigma) \end{pmatrix} \quad (9)$$

are stored, where $\bar{\boldsymbol{m}}_t^{\boldsymbol{i}}$ is the true distance and bearing between the robot and any landmark i and $\boldsymbol{n_t^m}$ is added zero mean measurement noise with variance $\sigma \equiv 2 \times 10^{-2}$.

6) The set of landmark particles

$$\boldsymbol{W_t} = \left\{ \boldsymbol{w_t^{i,j}} \right\} \quad (10)$$

is updated, where $i \in [0, M]$ is the particle index, $j \in [0, n_s]$ is the landmark index and $n_s \in [0, n]$ is the number of landmarks that has been seen at least once.

If a landmark j is detected and has not been identified before, its corresponding landmark particle

$$\begin{aligned}\boldsymbol{w_t^{i,j}} &= \boldsymbol{x_{t(2\times1)}^{i}} + \boldsymbol{\bar{m}_t^j} + \boldsymbol{n_t^m} \\ &= \begin{pmatrix} x_t \\ y_t \end{pmatrix} + \begin{pmatrix} r_t^i \\ \alpha_t^i \end{pmatrix} + \begin{pmatrix} r \sim \mathcal{N}(0, Q_{1,1}) \\ r \sim \mathcal{N}(0, Q_{2,2}) \end{pmatrix}\end{aligned} \tag{11}$$

is added to $\boldsymbol{W_t}$. Note that the sum of the two last terms above in equation (10), i.e. received measurement and noise, is modeled with measurements noise variance Q. Equation (9) however corresponds to a simulation of the actual measurements with variance σ. In the real-world case this would depend on the perception equipment of the robot and is unknown.

7) The likelihoods $\psi_{i,j}$ of each particle with index $i \in [0, M]$ and measurement with index $j \in [0, n_s]$ are calculated in conjunction with a binary set of outlier o_t. It is described in accordance with the pseudo code seen in Algorithm 7, even if was done without loops but elemetwise matrix operations in the implementation.

Algorithm 1 getLikelihoods($\boldsymbol{S_t}$, $\boldsymbol{M_t}$, $\boldsymbol{W_t}$, λ_ψ, Q)

$o_t \leftarrow \emptyset$
$\psi_t \leftarrow \emptyset$
for all $\boldsymbol{m_t^j} \in \boldsymbol{M_t}$ **do**
  $\psi_t^i \leftarrow \emptyset$
  $\hat{\boldsymbol{z}}_{\boldsymbol{t}}^{\boldsymbol{j}} \leftarrow \begin{pmatrix} \sqrt{(w_x^j - x_t^i)^2 + (w_y^{i,j} - y_t^i)^2} \\ atan2(w_y^{i,j} - y_t^i, w_x^{i,j} - x_t^i) - \theta_t \end{pmatrix}$
  for all $\boldsymbol{s_t^i} \in \boldsymbol{S_t}$ **do**
    $\boldsymbol{\nu^{i,j}} \leftarrow \left(\boldsymbol{m_t^j} - \hat{\boldsymbol{z}}_{\boldsymbol{t}}^{\boldsymbol{j}}\right)$
    $\psi_t^{i,j} \leftarrow \frac{1}{2\pi det(Q)^{\frac{1}{2}}} exp\left(-\frac{1}{2}\boldsymbol{\nu^{i,j}}^T Q^{-1} \boldsymbol{\nu^{i,j}}\right)$
    $\psi_t^i \leftarrow \psi_t^i \cup \left\{\psi_t^{i,j}\right\}$
  end for
  if *mean of* $\psi_t^i \leq \lambda_\psi$ **then**
    $o_t^j \leftarrow 1$
  else
    $o_t^j \leftarrow 0$
  end if
  $o_t \leftarrow o_t \cup \left\{o_t^j\right\}$
  $\psi_t \leftarrow \psi_t \cup \left\{\psi_t^i\right\}$
end for
return ψ_t, o_t

8) Thereafter the importance weights

$$w_t^i = \frac{\prod_{j=1}^{n} \Psi_t^{i,j}}{\eta} \; \forall j \mid o_t^j = 1 \tag{12}$$

are calculated, where the i is the particle index, j is the landmark index and

$$\eta = \sum_{i=0}^{M} w_t^i \tag{13}$$

is a normalization constant.

9) To keep the relevant particles, resampling is performed. In order to minimize sample variance, i.e. the mean and standard deviation of the particle representation is not the same as for the random distribution[11], a systematic resampling algorithm is used, presented in Algorithm Low_variance_sampler, Table 4.4 in *Probalistic robotics* by Thrun et al. [11]. In addition to the Low_variance_sampler algorithm, except the robot particle, its corresponding landmark particle is too replaced by the landmark particle of the resampled robot particle. On top of that, Gaussian map resample noise

$$\boldsymbol{n_t^w} = \begin{pmatrix} r \sim \mathcal{N}(0, Q_{w_{1,1}}) \\ r \sim \mathcal{N}(0, Q_{w_{2,2}}) \end{pmatrix} \tag{14}$$

is added to each landmark. This is to allow landmark particles to converge to better representations. To further reduce sample variance, the injection of map resample noise is suspended for landmark particles corresponding to landmarks that are not seen.

10) Finally, if there are still control signals in the queue of trajectory elements, the next iterations continues from step 3), else the algorithm is done.

B. Occupancy Grid

In this implementation, a binary occupancy grid is utilized with the main purpose of making the mapping more extensive. In addition to the detected and mapped landmarks, the occupancy grid contributes with estimation of where the walls of the map are situated. Note that for this implementation, the occupancy grid is not used for localization of the map.

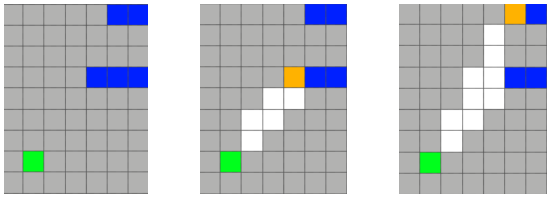
The grid is covering a 3×3 m^2 region around the map with 100×100 cells. All are initially marked as unknown, meaning that no area has been discovered by the LIDAR of the robot. The occupancy grid is based on wall reflected LIDAR measurements $\boldsymbol{M_t^W} = \{\boldsymbol{m_t^i}\}$ where

$$\begin{aligned}\boldsymbol{m_t^i} = \boldsymbol{\bar{m}_t^i} + \boldsymbol{n_t^m} = \begin{pmatrix} \bar{r}_t^i \\ \bar{\alpha}_t^i \end{pmatrix} + \begin{pmatrix} r \sim \mathcal{N}(0, \sigma) \\ r \sim \mathcal{N}(0, \sigma) \end{pmatrix} \\ \forall \bar{\alpha}_t^i \in \left\{\frac{2\pi}{360} i, \; \forall i \in [0, 360]\right\}\end{aligned} \tag{15}$$

in all directions with an assumed resolution of 1°with distances and bearing to walls rather than landmarks. For each measurement $\boldsymbol{m_t^i}$, all grid-cells in a line from the center of mass point of the robot particle cluster S_t to distance $\bar{r}_t^i$ away in the direction $\bar{\alpha}_t^i$ are marked as free space. The occupancy grid-cell furthers away are marked as occupied. To avoid that during a sequence, i.e. one turn of the LIDAR, a cell marked as occupied is overwritten and marked as free (which may happen due to numerical issues of grid-cells representating straight lines), marked occupied cells are kept track of during a sequence. These are not allowed to be overwritten until next sequence i.e. set of $\boldsymbol{M_t^W}$. The issue is illustrated in Figure 4.

VI. Implementation

The executable program is supposed to simulate a robot exploring (with a predetermined trajectory) in the map containg walls and a set of landmarks, recall Figure 3. The landmarks are not defined in a real-world application but are detectable in direct sight (not through walls) but attached at the top of walls



(a) Instance 1 (b) Instance 2 (c) Instance 3

Fig. 4. In the first instace in Figure 4(a), the robot (green) is placed next to some walls (blue). Thereafter in Figure 4(b) a measurement is received and the grid cells within the ray are marked as free space (white) and occupied (orange). Finally in Figure 4(c), another measurement overwrites the earlier marked occupied space. This is avoided by keeping track of the cells marked as occupied during a sequence.

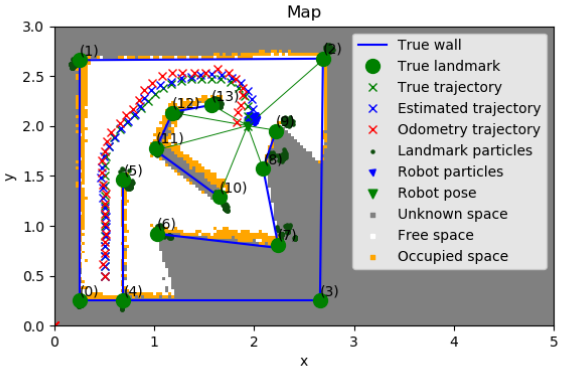


Fig. 5. Visualization of a time step during the implementation.

and can therefore be detected on both sides of the wall as in the simulation. All measurements are simulated by calculating the distance and angle (relative the robot pose) with artificial added noise in accordance with equation (9), (15).

In a real-world application, the odometry would be predicted with the control signals. However, due to systematic errors and white noise in the models corresponding to factors like unpredictable variation of friction, differences of wheel radii etc, the true trajectory will differ from the modeled one by the control signal. Since one cannot anticipate these disturbances, the particles should be predicted only with the control signals, that is, the ideal motion model. Nevertheless the opposite is carried out in this implementation; the true trajectory is perfectly in accordance with the control signals and the predicted motion model of particles are simulated with disturbances. This choice is based on the fact that the trajectory is predetermined and is to be fully controlled in the simulation. Using an ideal odometry model is justified by disturbing the motion prediction of the particles instead.

Now, the interface plots of the implementation is presented in Figure 5, complemented with a Video of the implementation. As seen in the figure and video, what is visualized is the true robot pose and trajectory, the odometry trajectory i.e. only based on prediction without regarding measurements, the estimated trajectory and landmark particles and finally the occupancy grid. The estimated robot pose and landmark posision is defined as the center of mass of the robot and landmark particle set, respectively. The mapped wall are sketched out ass the orange occupied space in the map whereas the white space is representing what has been explored by the LIDAR of the robot. Without performing any measurements or specifying CPU capacity, tested on a few standard personal laptops, the execution of the code is slightly slow but reasonable fast to run in real-time (check video for better understanding of speed performance).

VII. Experimental results

Using $M = 100$ particles, a simulated measurement noise variance $\sigma = 2 \times 10^{-2}$ and an odometry disturbance defined in equation (7), the result of the complete trajectory run is visualized in Figure 6, complemented with the difference of distance between estimated and true robot pose during the implementation run illustrated in Figure 7. The maximum difference of distance in this simulation can in the Figure be read out as 25 cm, whereas the final error is 15 cm. It can be noted that the error is steadily increasing with time (steps) as expected, because of the accumulating motion model uncertainty, except for two instances; around time step 40 and 80. What happens around the first instance i.e. time step 40, is that a few landmarks are recognized and seen again, specifically landmark (5), (6) and (7) seen from robot position $(x_{40}, y_{40}) \approx (1.8, 1.1)^T$. At the latter instance around time step 80, landmark (2) is re-detected from robot position $(x_{80}, y_{80}) \approx (2.5, 1.8)^T$, followed by revisiting landmark (1), (10), (11), (12), (13). The landmarks that has been seen earlier has in general better precision and accuracy since the uncertainty of the robot pose estimate increases with time due to motion. As the position of the early detected landmarks are suspended in noise injection when not seen, they still are quite accurate and complete the measurement information when re-detected later, why the error is as expected decreasing at these two places.

Running the code, the time taken can be noted to be $\sim$ 1 minute, depending on CPU capacity, which may not be outstandingly fast, nevertheless allow real-time application for a small robot transporting several meters. This was one of the goals with the project that can reasonably be said to successfully be fulfilled. The other goal was that the (distance) error between the estimated and true robot pose and map attributes is sufficiently small in the sense that neither the true robot pose collides with the estimated map walls, nor that the estimated robot pose interferes with the true walls of the map, with respect to the same frame. This is close but yet not adequately fulfilled in simply considering the visual result of Figure 6. In the same Figure it can also be concluded that some landmarks (take a look at landmark (4) and (6)) are not only a bit off but have too low uncertainty meaning they are drifted away. Other than that, the representations of robot pose and landmark positions with the particles are in general quite accurately descriptive, as well as the occupancy estimations of the walls. The result seems to be similar for different random seed initialization, meaning different configurations

of deterministic random sequence generations for the code.

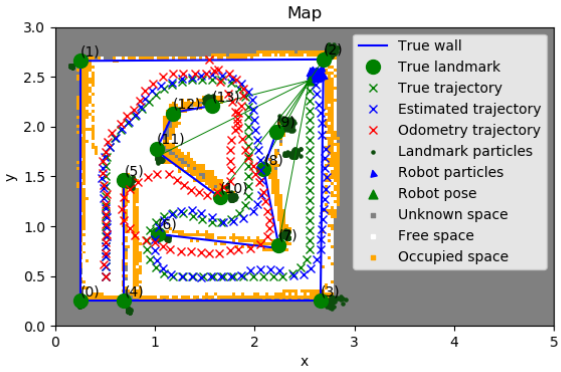


Fig. 6. Visualization of the final step of the implementation.

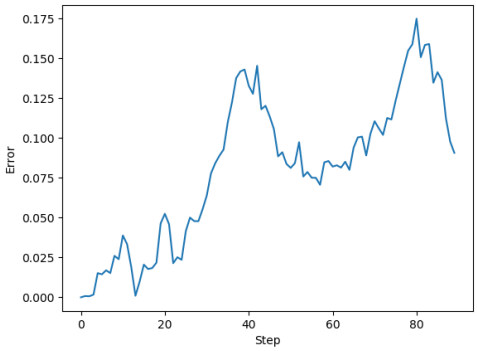


Fig. 7. The distance error between the true and estimated robot pose during the complete run of the trajectory in the implementation.

VIII. Summary and Conclusions

Given the methods and configurations covered in this paper, the implemented simulation (Figure 6 and Video) ended up in a computationally acceptable performance for real time application as one of the goals was determined to be investigated. The proposed SLAM solution with Particle Filter works in the way that it successfully maps reasonable estimates of landmarks and detected walls with good robot pose tracking. However, the goal of no allowed interference between true robot pose and map with estimated robot pose and map was not entirely fulfilled as seen in Figure 6.

What helped keeping the good accuracy and precision of the early detected landmarks, contributing better recovery of robot pose estimate when re-detecting these, was ensuring low sample variance by using systematic resampling and suspend injection of map noise to landmarks when not seen. Unfortunately, the estimated robot pose drifted away too much from the true one while only keeping track of newly detected landmark, to the extent that the second goal was not fulfilled. What surely could have helped is to increase number of particles as 100 is quite low order of magnitude. Nevertheless, this would increase the computational demands a lot and instead perhaps eliminate the chance of running in real-time. Another aspect is the use of the occupancy grid. In this implementation, there is only one instance of occupancy grid for visual purposes of sketching the walls of the map and what has been explored. Instead of a point featured map, each robot particle can have its own occupancy grid of which the likelihoods and of that particle weights are based on. This would give much more rich data and calculations. Though, this would radically arise computational demands especially for many particles. What however would help is using efficient data structures for occupancy grids such as Quadtrees that was mention in Section IV and covered by Li and Ruichek.[10]. Another attribute of the occupancy grid that is not implemented in this project is defining a configuration space by inflating the occupancy grid map. This would be done by marking all occupancy cells with a robot radius distance margin from any wall as occupied, as the center of the robot is strictly not able to get closer to the walls. Considering the trajectory was predetermined in this implementation, this was not needed (more that for visual purposes how much the robot exceeds the minimum distance to any wall). It should however be included in a real-world application of SLAM where exploration is performed.

References

[1] H. Durrant-Whyte and T. Bailey., "Simultaneous localization and mapping: part i." *IEEE robotics & automation magazine*, vol. 13.2, p. 99, 2006.

[2] B. W. Thrun, S. and D. Fox, *Probalistic robotics*. London, England: MIT Press, 2005, pp. 312–332.

[3] H. Durrant-Whyte and T. Bailey., "Data association in stochastic mapping using the joint compatibility test." *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION*, vol. 17, pp. 890–897, 2001.

[4] A. Eliazar and R. Parr., "Dp-slam: Fast, robust simultaneous localization and mapping without predetermined landmarks." *IJCAI*, vol. 3, 2003.

[5] S. Särkkä, *Bayesian Filtering and Smoothing*. London, England: Cambridge University Press, 2013, p. 151.

[6] C. Andrieu and A. Doucet, "Particle filtering for partially observed gaussian state space models." *Journal of the Royal Statistical Society*, vol. 64, no. 4, pp. 827–836, 2002.

[7] A. E. Gelfand and A. F. M. Smith., "Sampling-based approaches to calculating marginal densities." *Journal of the American Statistical Association*, vol. 85, no. 410, pp. 398–409, 1990.

[8] G. Grisetti, G. D. Tipaldi, C. Stachniss, W. Burgard, and D. Nardi, "Fast and accurate slam with rao–blackwellized particle filters," *Robotics and Autonomous Systems*, vol. 55, no. 1, pp. 30 – 38, 2007, simultaneous Localisation and Map Building. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S092188900600145X

[9] B. W. Thrun, S. and D. Fox, *Probalistic robotics*. London, England: MIT Press, 2005, p. 478.

[10] Y. Li and Y. Ruichek, "Building variable resolution occupancy grid map from stereoscopic system — a quadtree based approach," in *2013 IEEE Intelligent Vehicles Symposium (IV)*, 2013, pp. 744–749.

[11] B. W. Thrun, S. and D. Fox, *Probalistic robotics*. London, England: MIT Press, 2005, pp. 108–110.