3.
SLAM Solutions

3.1.
Gaussian Filter SLAM Solutions

"Historically, Gaussian Filters constitute the earliest tractable implementations of the Bayes Filter for continuous spaces". It could say that they are also by far the most popular family of techniques to date – despite a number of limitations [1].

Gaussian assumes the idea that beliefs are represented by multivariate normal distributions:

$$p(x) = \det(2\pi\Sigma)^{-\frac{1}{2}} \exp\{-\tfrac{1}{2}(x-\lambda)^T \Sigma^{-1}(x-\lambda)\} \tag{3.1}$$

The distribution over the variable x is characterized by two sets of parameters: the mean λ and the covariance Σ. The mean has the same dimensionality of the state x. The covariance is a symmetric quadratic matrix, positive semi-definite, and its dimension is the dimensionality of the state x squared. Hence, the dimension of the covariance matrix depends quadratically on the dimension of the state vector x.

3.1.1.
Kalman Filter SLAM

"Probably the best studied technique for implementing Bayes filter is the Kalman Filter" [1]. "The Kalman Filter (KF) was developed by R.E. Kalman, whose prominent paper on the subject was published in 1960" [4].

The KF is an algorithm which processes data and estimates variable values. In SLAM context, the variable values to be estimated consist of the robot position

and landmark locations. The data to be processed may be actuator inputs, range sensor readings, motion sensors and digital cameras of the mobile robot. Thus, the KF utilizes all available data to simultaneously estimate robot position and generate a landmark map. In [23] it is explained that KF is a set of mathematical equations that provides an efficient computational (recursive) mean to estimate the estate of a process, in a way that minimizes the mean of the squared error.

"Under certain conditions, the estimates made by a KF are very good; in fact, they are in a sense "optimal" " [4].

The Kalman Filter represents probability distributions at time t by the mean λ_t and the covariance Σ_t. Thus, posterior distributions are Gaussian if the following three properties are fulfilled, in addition to the Markov assumptions of the Bayes filter [1].

1. the next state probability (or motion model), $p(x_t|u_t,x_{t-1})$ in eq. (2.3), must be a linear function in its arguments with added Gaussian noise [1]. This is expressed by the following equation:

$$x_t = A_t x_{t-1} + B_t u_t + \varepsilon_t \tag{3.2}$$

where x_t and x_{t-1} are *state* vectors, and u_t is the control vector at time t, given by

$$x_t = \begin{pmatrix} x_{1,t} \\ x_{2,t} \\ \vdots \\ x_{m,t} \end{pmatrix} \quad \text{and} \quad u_t = \begin{pmatrix} u_{1,t} \\ u_{2,t} \\ \vdots \\ u_{q,t} \end{pmatrix} \tag{3.3}$$

A_t is a square matrix of size m x m, where m is the dimension of the state vector x_t. B_t is of size m x q, where q is the dimension of the control vector u_t. The random variable, ε_t in eq.(3.2), is a Gaussian random vector of size m, that models the uncertainty in the state transition. Its mean is zero and its covariance is denoted by P_t. A state transition of the

form in eq.(3.2) is called a linear Gaussian, "to reflect the fact that it is linear in its arguments with additive Gaussian noise" [1].

The probability $p(x_t|u_t,x_{t-1})$ is obtained by plugging eq.(3.2) into the multivariate normal distribution, eq. (3.1). The mean of the posterior state is given by $A_t x_{t-1}+B_t u_t$ and the covariance by P_t, thus

$$p(x_t \mid u_t, x_{t-1})$$
$$= \det(2\pi P_t)^{-\frac{1}{2}} \exp\{-\tfrac{1}{2}(x_t - A_t x_{t-1} - B_t u_t)^T P_t^{-1}(x_t - A_t x_{t-1} - B_t u_t)\} \quad (3.4)$$

2. the measurement probability (or perception model), $p(z_t|x_t)$ in eq. (2.3), must also be linear in its arguments, with added Gaussian noise [1]:

$$z_t = H_t x_t + \delta_t \quad (3.5)$$

H_t is a matrix of size k x m, where k is the dimension of the measurement vector z_t. The vector δ_t describes the measurement noise with a multivariate Gaussian with zero mean and covariance Q_t. In this way the measurement probability is given by the following multivariate normal distribution [1]:

$$p(z_t \mid x_t) = \det(2\pi Q_t)^{-\frac{1}{2}} \exp\{-\tfrac{1}{2}(z_t - H_t x_t)^T Q_t^{-1}(z_t - H_t x_t)\} \quad (3.6)$$

3. finally, the initial probability $p(x_0)$ must be normally distributed and, denoted by the mean λ_0 and the covariance Σ_0 [1]:

$$p(x_0) = \det(2\pi\Sigma_0)^{-\frac{1}{2}} \exp\{-\tfrac{1}{2}(x_0 - \lambda_0)^T \Sigma_0^{-1}(x_0 - \lambda_0)\} \quad (3.7)$$

These three assumptions are sufficient to ensure that the posterior $p(x_t)$ is always a Gaussian, for any point in time [1].

As described above, the Kalman Filter represents probability distributions at time t by the mean λ_t and the covariance Σ_t. The equations of the Kalman Filter algorithm are depicted in Table 3.1. The inputs of the Kalman Filter is the distribution at time $t-1$, represented by λ_{t-1} and Σ_{t-1}, the control u_t, and the measurement z_t. The output is the distribution at time t, represented by λ_t and Σ_t.

Table 3.1: The Kalman Filter Algorithm [1].

Kalman_filter_algorithm ($\lambda_{t-1}, \Sigma_{t-1}, u_t, z_t$)		
1:	$\overline{\lambda}_t = A_t \lambda_{t-1} + B_t u_t$	(3.8)
2:	$\overline{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + P_t$	(3.9)
3:	$K_t = \overline{\Sigma}_t H_t^T \left(H_t \overline{\Sigma}_t H_t^T + Q_t \right)^{-1}$	(3.10)
4:	$\lambda_t = \overline{\lambda}_t + K_t \left[z_t - H_t \overline{\lambda}_t \right]$	(3.11)
5:	$\Sigma_t = \overline{\Sigma}_t - K_t H_t \overline{\Sigma}_t$	(3.12)
6:	return λ_t, Σ_t	

Let's describe some of the parameters in above equations.

- $\overline{\Sigma}_t$, $\overline{\lambda}_t$ are the predicted covariance and median, representing $\int p(x_t \mid u_t, x_{t-1})\, p(x_{t-1} \mid z^{t-1}, u^{t-1}) dx_{t-1}$ in eq. (2.3), obtained by incorporating the control u_t one step later, but before incorporating the measurement z_t.

- A_t is called the state transition matrix; it describes how one thinks the state will change due to factors not associated with control input. One very nice convention in SLAM is that landmarks will remain stationary. Except for the first row which correspond to changes in robot position, A_t will therefore appear as a diagonal matrix with each diagonal entry containing identity matrices (otherwise A_t would change location of landmarks, which are known to be stationary) [4]. If the robot can have a non-zero velocity at time step t, then the state may change even with no actuator input, and the first column of A_t can account for this. To simplify the analysis, let's assume that the robot comes to a halt after each time step. In this case, the actuator input

fully specifies the most likely new location of the robot. This means that A_t is just the identity matrix, which could be disregarded.

- B_t is a matrix that translates control input into a predicted change in state. Its values will depend on the representation of the control input. It will vary depending on the physical construction of the robot. Because the landmarks will remain stationary the only interesting entries of B_t will be in the first row. Thus, the idea of eq. (3.8) is that the best guess for the new state will be described exactly by the old robot position and how one believes the actuators will change this position.

- P_t is the covariance of the process noise. It accounts for how moving will change the confidence on each individual pair of landmarks as well as robot-landmarks pairs. The entries of P_t will depend on the distance landmarks are away from the robot and other known particularities of the environment and/or state [4].

- Q_t is the covariance matrix for the range sensor noise, it is used to keep track of one's confidence in the range sensor readings.

- H_t transforms one's previous state estimate into a representation used by the sensors. In other words, if the sensors had perceived the world state exactly as predicted by $\overline{\lambda}_t$, they would have returned this information in the form $H_t \overline{\lambda}_t$. Note that the purpose of H_t is very similar to that of B_t.

- Finally, K_t is called the Kalman *gain*. It specifies the degree to which the measurement is incorporated into the new state estimate. The magnitudes of the values in K_t depend on the predicted covariance $\overline{\Sigma}_t$, relative to the combined values of the predicted covariance and sensor uncertainty Q_t.

The Kalman filter is a technique for filtering and prediction in linear systems. However, in most real world SLAM situations there will be some non-linear aspect one might wish to account for. "For example, a robot that moves with constant translational and rotational velocity typically moves on a circular trajectory, which cannot be described by linear next estate transitions" [1]. Thus

the plain Kalman Filters, as discussed above is inapplicable to all but the most trivial robotics problems.

3.1.2. Extended Kalman Filter SLAM

The *Extended Kalman Filter* (EKF) overcomes the linearity assumption [1]. Here, in EKF, the assumption is that the next state probability $p(x_t|u_t, x_{t-1})$, and the measurement probability $p(z_t|x_t)$, are ruled by nonlinear functions f and h, respectively.

$$x_t = f(u_t, x_{t-1}) + \varepsilon \tag{3.13}$$

$$z_t = h(x_t) + \delta \tag{3.14}$$

This model is a generalization of the linear Gaussian model underlying Kalman filters, as stated in eq. (3.2) and eq. (3.5). The function f replaces the matrices A_t and B_t in eq.(3.2) and h replaces H_t in eq. (3.5) [1]. However, the distribution is not longer a Gaussian when it is used nonlinear functions, f and h. In this way, the distribution update does not possess a closed-form solution. Therefore, the EKF calculates an approximation of the true distribution. "Thus, the EKF inherits from the Kalman filter the basic belief representation, but it differs in that this belief is only approximate, not exact as it was the case in linear Kalman Filters" [1].

To manage this approximation EKF utilizes a (first order) *Taylor expansion*. The Taylor expansion constructs a linear approximation to a function f from its value and slope. The slope is given by the following partial derivative [1]:

$$f'(u_t, x_{t-1}) := \frac{\partial f(u_t, x_{t-1})}{\partial x_{t-1}} \tag{3.15}$$

Both the value of f and its slope depend on the argument of f.. Thus f is approximated by its value at λ_{t-1} (and at u_t), and the linear extrapolation is achieved by a term proportional to the gradient of f at λ_{t-1} and u_t [1]:

$$f(u_t, x_{t-1}) \approx f(u_t, \lambda_{t-1}) + \underbrace{f'(u_t, \lambda_{t-1})}_{=: F_t}(x_{t-1} - \lambda_{t-1})$$

$$f(u_t, x_{t-1}) = f(u_t, \lambda_{t-1}) + F_t(x_{t-1} - \lambda_{t-1}) \tag{3.16}$$

Written in form of Gaussians, the next state probability is approximated by:

$$\begin{aligned} & p(x_t \mid u_t, x_{t-1}) \\ & \approx \det(2\pi P_t)^{-\frac{1}{2}} \exp\{-\tfrac{1}{2}[x_t - f(u_t, \lambda_{t-1}) - F_t(x_{t-1} - \lambda_{t-1})]^T \\ & \quad P_t^{-1}[x_t - f(u_t, \lambda_{t-1}) - F_t(x_{t-1} - \lambda_{t-1})]\} \end{aligned} \tag{3.17}$$

The matrix F_t is often called the *Jacobian*. The value of the Jacobian depends on u_t and λ_{t-1}, thus it differs for different time points.

The same linearization is used for the measurement function h. Where, the Taylor expansion is developed around $\overline{\lambda}_t$, the state regarded most likely by the robot at the time when it linearizes h [1]:

$$h(x_t) \approx h(\overline{\lambda}_t) + \underbrace{h'(\overline{\lambda}_t)}_{=: H_t}(x_t - \overline{\lambda}_t)$$

$$h(x_t) = h(\overline{\lambda}_t) + H_t(x_t - \overline{\lambda}_t) \tag{3.18}$$

where $h'(x_t) = \frac{\partial h(x_t)}{\partial x_t}$. Written in form of Gaussian, one gets:

$$p(z_t \mid x_t)$$

$$\approx \det(2\pi Q_t)^{-\frac{1}{2}} \exp\{-\tfrac{1}{2}[z_t - h(\overline{\lambda}_t) - H_t(x_t - \overline{\lambda}_t)]^T$$

$$Q_t^{-1}[z_t - h(\overline{\lambda}_t) - H_t(x_t - \overline{\lambda}_t)]\} \quad (3.19)$$

Table 3.2 depicts the Extended Kalman Filter algorithm.

Table 3.2: The EKF Algorithm [1]

EKF_algorithm ($\lambda_{t-1}, \Sigma_{t-1}, u_t, z_t$)		
1:	$\overline{\lambda}_t = f(u_t, \lambda_{t-1})$	(3.20)
2:	$\overline{\Sigma}_t = F_t \Sigma_{t-1} F_t^T + P_t$	(3.21)
3:	$K_t = \overline{\Sigma}_t H_t^T \left(H_t \overline{\Sigma}_t H_t^T + Q_t\right)^{-1}$	(3.22)
4:	$\lambda_t = \overline{\lambda}_t + K_t\left[z_t - h(\overline{\lambda}_t)\right]$	(3.23)
5:	$\Sigma_t = \overline{\Sigma}_t - K_t H_t \overline{\Sigma}_t$	(3.24)
6:	return λ_t, Σ_t	

In some ways, the EKF is similar to the (linear) Kalman Filter. The difference is that the linear equations in Kalman Filters are replaced by their non-linear generalization in EKFs.

A detailed implementation of the EKF algorithm is shown in Section 4.1.

3.2. Particle Filter SLAM Solutions

3.2.1. Particle Filter Overview

Particle Filters (PF) are alternatives to Gaussian techniques. They do not rely on a fixed functional form of the posterior distribution, such as Gaussians.

Instead, they approximate these posterior distributions by a finite number of values, each harshly corresponding to a region in state space.

"The key idea of the PF is that any posterior distribution $p(x_t)$ can be represented by a set of random state samples drawn from this posterior" [1]. Figure 3.1 shows this idea for a Gaussian; instead of representing the distribution by a parametric form (the mean and covariance that defines the exponential of a normal distribution), PF represents it by a set of samples drawn from this Gaussian. As the number of samples goes to infinity, PF tends to converge uniformly to the correct posterior distribution. Thus this method can represent any arbitrary shape of distribution, making it good for non-Gaussian, multimodal distributions.

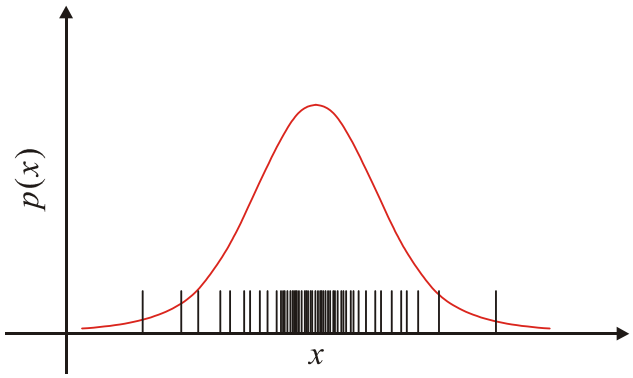


Figure 3.1: Representation of a Gaussian by a set of particles

In PF, the samples are called *particles*, thus the posterior $p(x_t)$ is represented by N weighted particles:

$$\Phi_t := \{\langle x_t^i, w_t^i \rangle / i = 1...N\} \tag{3.25}$$

The correspondence between the Bayes Filters and the approximation made by particles is given by

$$p(x_t) \approx \Phi_t \tag{3.26}$$

In this way, to compute $p(x_t)$ it is necessary to find Φ_t at each time, that is to find all values of x_t^i and w_t^i. As a Bayes Filter algorithm, the PF algorithm constructs the distribution $p(x_t)$ recursively from the distribution $p(x_{t-1})$ one time step earlier. Thus, PF constructs the particle set Φ_t recursively from the set Φ_{t-1}.

In Probabilistic Robotics, the process to generate samples x_t^i is achieved using the prior Φ_{t-1} and the most recent control u_t. The desired weight w_t^i of each particle is given using the most recent measurement z_t. Table 3.3 shows the most basic variant of the PF algorithm[1].

Table 3.3: Particle Filter Algorithm [1]

	Particle Filter_algorithm (Φ_{t-1}, u_t, z_t)
1:	$\overline{\Phi}_t = \Phi_t = 0$
2:	for $i = 1$ to N do
3:	sample $x_t^i \sim p(x_t \mid u_t, x_{t-1}^i)$
4:	$w_t^i = p(z_t \mid x_t^i)$
5:	$\overline{\Phi}_t = \overline{\Phi}_t + \left\langle x_t^i, w_t^i \right\rangle$
6:	end for
7:	for $i = 1$ to N do
8:	draw i with probability $\propto w_t^i$
9:	add x_t^i to Φ_t
10:	end for
11:	return Φ_t

The algorithm first samples by processing each particle x_{t-1}^i in the input particle set Φ_{t-1} as follows:

1. Line 3 of table Table 3.3 generates a estimate x_t^i for time t based on the particle x_{t-1}^i and the control u_t. This step involves sampling for the next state transition $p(x|u_t, x_{t-1})$. Thus the set of particles resulting from iterating line 3 N times represents the distribution $\int p(x_t \mid u_t, x_{t-1}) p(x_{t-1} \mid z^{t-1}, u^{t-1}) dx_{t-1}$ in eq. (2.3).

2. Line 4 computes for each particle x_t^i the corresponding weight (*importance factor*) w_t^i using the measurement z_t. Thus, each w_t^i is the probability of the measurement z_t under the particle x_t^i, in the way of $w_t^i = p(z_t \mid x_t^i)$.

3. Finally lines 7 to 11 implement as the so-called *resampling* or *importance resampling*. This lines draw with replacement N particles from the temporary set $\overline{\Phi}_t$. The probability of drawing each particle is given by its importance factor (weight). The resulting set, Φ_t, is a set of N particles distributed according to the desired $p(x_t)$.

Figure 3.2 shows the Particle Filter algorithm idea. The desired $p(x_t)$ is shown as a red line and the samples x_t^i as blue lines.

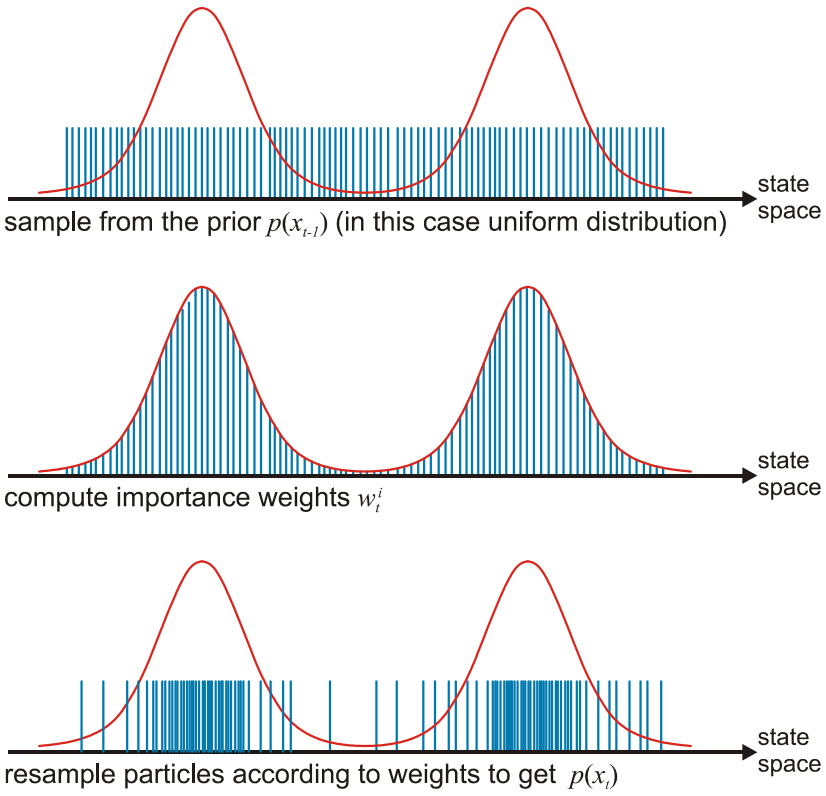
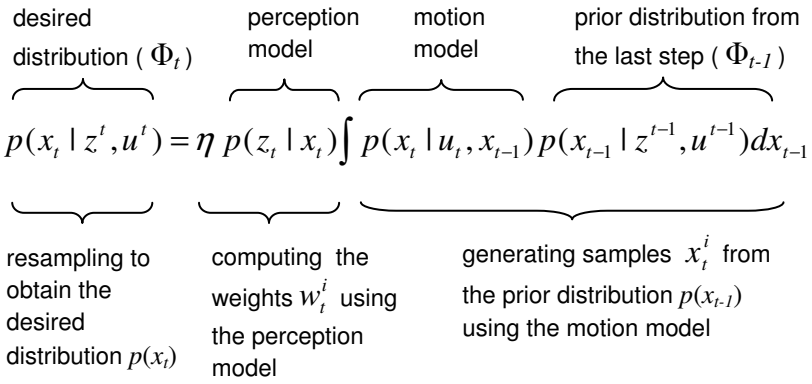


Figure 3.2: Particle Filter idea.

The PF computes the Bayes filter stated in eq. (2.3) from right to left, as shown in the following equation:



3.2.2. Fast SLAM

The Fast SLAM was developed by Montemerlo, Thrun, Koller and Wegbreit [24]. Fast SLAM exploits the condition independence properties of the SLAM model to break up the problem of localizing and mapping into many separate problems.

As it was seen in Section 3.1, the dimension of the covariance matrix Σ_t is the dimensionality of the state x squared. Thus, the number of elements in the covariance matrix depends quadratically on the number of elements in the state vector x. This is because the robot's position uncertainty correlates landmark locations, as Figure 3.3 shows. Supposing an observation (L_1, L_2) made by the robot, then, through another observation for L_1, one ends up to another position for L_2. Now, if the assumption on L_1 is again different, the conclusion on L_2 also is. This is the consequence of not knowing the robot's position precisely. As a result, this lack of knowledge on the robot's position correlates the location of the landmarks.

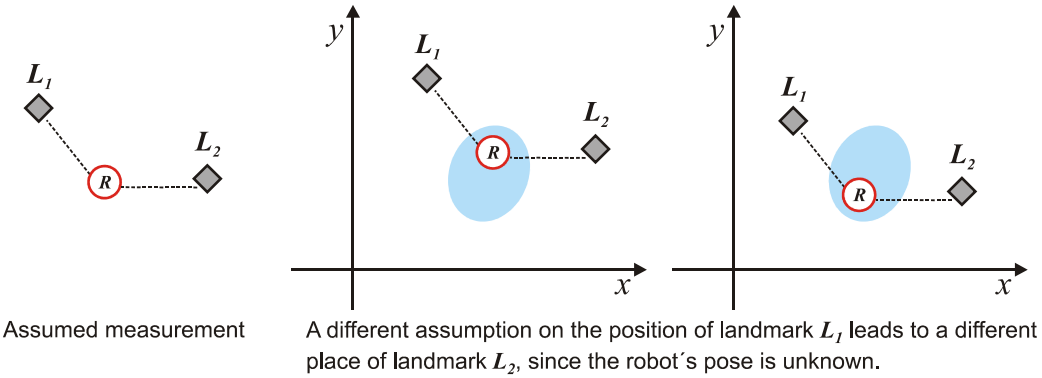


Figure 3.3: Landmark correlation

In this way, if one could know exactly the robot position, there should be no predictable relationship between the landmark observations. An important point in SLAM is that the exact robot pose is not known, but insight of conditional independence of landmarks, given the pose, is enough to motivate FastSLAM, which manages each landmark separately [4] - decoupling into n (number of landmarks) independent estimation problems, one for each landmark. Thus, Fast SLAM decomposes the SLAM problem into a robot position problem, and a set of landmark location estimation problems that are conditioned on the robot position estimated.

Mathematically, decorrelation of the landmark locations leads to a factored representation as following:

$$p(R^t, m \mid z^t, u^t) = p(R^t, L_1, ..., L_n \mid z^t, u^t)$$
$$p(R^t, L_1, ..., L_n \mid z^t, u^t) = p(R^t \mid z^t, u^t) \prod_n p(L_n \mid R^t, z^t, u^t) \qquad (3.28)$$

This factorization is exact and always applicable to the SLAM problem [24]. It decomposes the posterior over robot paths and maps into n+1 recursive estimators: one estimator over robot pats $p(R^t \mid z^t, u^t)$ and n separated landmark pose estimators $p(L_n \mid z^t, u^t, R^t)$ conditioned on each hypothetical path.

FastSLAM keeps track of many possible paths simultaneously (superscript t of the robot pose R^t), as opposed to the traditional Kalman Filter, "which does not even keep track of a single path, but rather updates a single robot pose" [4]. Thus,

Fast SLAM records paths and when the algorithm terminates there will be a record of where the robot has been.

Fast SLAM uses a modified PF for implementing the path estimator, $p(R^t \mid z^t, u^t)$, and the landmark pose estimators $p(L_n \mid z^t, u^t, R^t)$ are realized by Kalman Filters, using separate filters for different landmarks. Because landmark estimation is conditioned on path estimation, each particle in PF has its own local landmark estimations. At time *t*, the *i*-th particle contains

$$s_t^i := \{ w_t^i,\ R^{i,t}, \lambda_{1,t}^i, \Sigma_{1,t}^i,\ \lambda_{2,t}^i, \Sigma_{2,t}^i,\ ...,\ \lambda_{n,t}^i, \Sigma_{n,t}^i \} \tag{3.29}$$

weight — robot's path — landmark *1* — landmark *2* — landmark *n*

where $\lambda_{n,t}^i$ and $\Sigma_{n,t}^i$, are the Gaussian parameters (mean and covariance respectively) related with the landmark position $L_{n,t}^i$. To update the landmark-map for a given path $R^{i,t}$ each observed landmark is processed individually as an EKF measurement update from a known robot pose (unobserved landmarks are unchanged).

Bearing the distribution at time $t-1$ as a set of particles, one gets:

$$\Phi_{t-1} := \{ \langle s_{t-1}^i \rangle / i = 1 ... N \} \tag{3.30}$$

The first version of Fast SLAM algorithm [1] follows the following steps:

1. for each particle extend its path, $R^{i,t-1}$, to generate a new pose using the control u_t and the motion model $p(R_t \mid u_t, R_{t-1})$. Mathematically this means that the distribution at time $t-1$, $p(R^{t-1} \mid z^{t-1}, u^{t-1})$ becomes $p(R^t \mid z^{t-1}, u^t)$. A set of temporal particles is obtained.

2. update the estimation of landmarks, through the EKF

$$p(L_n \mid R^t, z^t, u^t) = \eta\, p(z_t \mid L_n, R_t)\, p(L_n \mid R^{t-1}, z^{t-1}, u^{t-1}) \tag{3.31}$$

and, using the new poses generated in the step 1 and the measurement z_t, compute the new set of $\lambda_{n,t}^{i}$ and $\Sigma_{n,t}^{i}$. This set of means and covariances are added to the temporal particles set.

3. assign the weight for each particle. This is computed using the result of step 1, $p(R^t \mid z^{t-1}, u^t)$, where the measurement z_t was not included, and using the distribution $p(R^t \mid z^t, u^t)$, where z_t is included

$$w_t^i = \frac{p(R^{t,i} \mid z^t, u^t)}{p(R^{t,i} \mid z^{t-1}, u^t)} \tag{3.32}$$

after some considerations and probabilistic transformations, the above equation is given by:

$$w_t^i \approx \eta \det(2\pi U_t^i)^{-\frac{1}{2}} \exp\{-\tfrac{1}{2}(z_t - \hat{z}_t^i)^T U_t^{i\,-1}(z_t - \hat{z}_t^i)\} \tag{3.33}$$

$$U_t^i = H_t^{i\,T} \Sigma_{t-1,n}^i H_t^i + Q \tag{3.34}$$

where z_t is the sensor observation. Note that $\hat{z}_t^i$ is the predicted observation computed using the landmark position estimated in step 2 and the robot poses generated in step 1. H_t^i is the *Jacobian* of the perception model. This set of importance factors (weights) are added to the temporal particles set.

4. finally, resample. Each particle, in the temporal particle set, is drawn (with replacement) with a probability proportional to its importance factor.

A detailed implementation of the Fast Slam 1.0 algorithm is shown in Section 4.2.

3.2.3. DP-SLAM

The Fast SLAM is a good solution to the SLAM problem when there are thousands of landmarks that can be tracked accurately. However tracking landmarks is actually very difficult, particularly in environments with monochromatic areas or repeating patterns.

If the robot is using a LRF, the map generated by the laser has no landmarks; is rather an occupancy grid, as described in Section 2.2.2. "The robot cannot use the range finder to relocate individual points in the grid. However with enough data, the robot might not need to worry about reacquiring landmarks in the first place" [4]. For example, if there was a contoured object in the environment, the robot might align entire occupancy maps by matching up the contour of that object. Distributed Particle SLAM (DP-SLAM) [6] attacks SLAM from this occupancy grid approach, while simultaneously utilizing the conditional independence insight discussed in Fast SLAM.

"One of the steps in the Fast SLAM algorithm was to generate a new pose prediction and a corresponding map prediction" [4]. Thus, the new map prediction was based on the previous pose and the control input u_t. It cannot predict how landmarks will move with respect to the robot anymore because landmarks are not dealt with. However it is possible to make a prediction as to how the occupancy grid will change. Figure 3.4 shows an example of occupancy grid prediction based on a movement of one cell to the right.

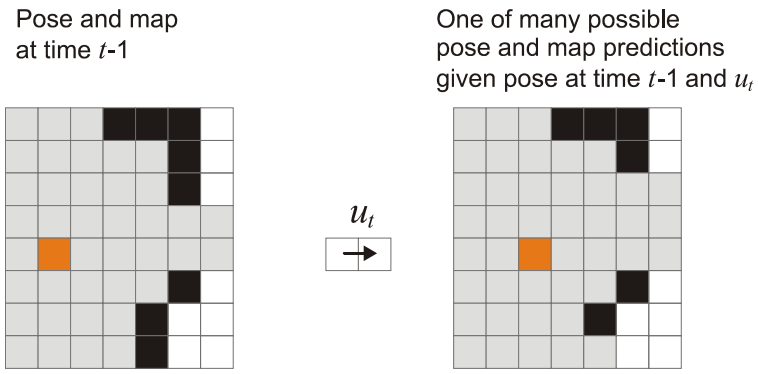


Figure 3.4: Occupancy grid prediction based on a movement of one cell to the right.

Similarly to Fast SLAM, DP-SLAM uses PF, generating new particles (poses) by applying probabilistically generated movement vectors to old poses. Thus DP-SLAM uses the same method to generate samples based on the motion model and the control vector u_t.

However, because the map representation is an occupancy grid, the perception model is quite different consequently, the way to assign weights, and the map update are quite different too.

3.2.3.1. DP-SLAM Map Representation

DP-SLAM uses an occupancy grid mapping representation, specifically stochastic maps, where each square has a sliding scale of various degrees of occupancy, as it was seen in Section 2.2.2.

"The idea of using probabilistic map representation is possibly as old as the topic of robotic mapping itself" [6]. Most of the earliest SLAM methods used probabilistic occupancy grids and were especially useful for sonar sensors susceptible to noisy and/or spurious measurements [6].

However, DP-SLAM concentrates on a model for Laser Range Finder. It has a method for representing uncertainty in the map, which takes into account the distance the laser travels through each grid square.

Earlier approaches, to estimating the total probability of the scan, would trace the scan through the map, giving a weight to the measurement error associated with each potential obstacle by the probability that the scan has actually reached the obstacle [6]. "In an occupancy grid with partial occupancy, each cell is a potential obstacle" [6].

Thus each grid square has the probability of stopping a laser ray, represented by [6]

$$P_c(x_i, \rho_i) = 1 - e^{\frac{-x_i}{\rho_i}} \tag{3.35}$$

where x_i is the distance that the laser ray travels through the square i and ρ_i is called the *opacity* of the square, as shown in Figure 3.5.

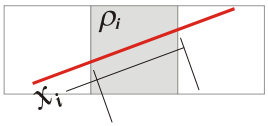


Figure 3.5: Square representation

The probability that an entire laser ray will have been stopped at some point along its trajectory is therefore the cumulative probability that the laser ray is interrupted by squares up to and including the last square, n, it reaches:

$$P(stopped = true) = P_c(\boldsymbol{x}, \boldsymbol{\rho}) = \sum_{i=1}^{n} P_c(x_i, \rho_i) \prod_{j=1}^{i-1} (1 - P_c(x_j, \rho_j)) \qquad (3.36)$$

where x_i is the traveled distance (the laser ray travels through the square) and ρ_i the opacity of the square i, as shown in Figure 3.6.

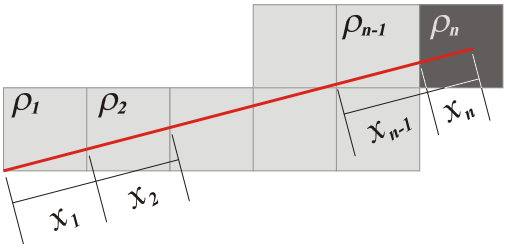


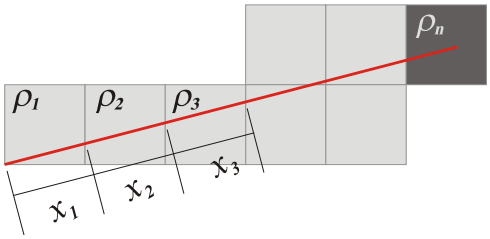
Figure 3.6: Interaction between the laser ray and the square representation

Inside the summation in eq. (3.36), the first term is the probability that the laser ray would be obstructed in the square n. The second term represents the probability that each previous square did not obstruct the laser.

Thus the probability that the laser ray will be interrupted a grid square j is $P(stop=j)$, which is computed as the probability that the laser has reached square $j-1$ and then stopped at j :

$$P(stop = j \mid \boldsymbol{x}, \boldsymbol{\rho}) = P_c(x_j, \rho_j)(1 - P_c(\boldsymbol{x}_{1:j-1}, \boldsymbol{\rho}_{1:j-1})) \tag{3.37}$$

where $\boldsymbol{x}_{1:j-1}$ and $\boldsymbol{\rho}_{1:j-1}$ have interpretations as fragments of the $\boldsymbol{x}$ and $\boldsymbol{\rho}$ vectors. Figure 3.7 illustrates this, where j=3, the vector $\boldsymbol{x} = \{x_1, x_2, ...x_n\}$, and $\boldsymbol{\rho} = \{\rho_1, \rho_2, ...\rho_n\}$.



$$P(stop = 3 \mid \boldsymbol{x}, \boldsymbol{\rho}) = P_c(x_3, \rho_3)(1 - P_c(\boldsymbol{x}_{1:2}, \boldsymbol{\rho}_{1:2})) = (1 - e^{-\frac{x_3}{\rho_3}})[1 - P_c(x_1, \rho_1)][1 - P_c(x_2, \rho_2)]$$

Figure 3.7: Example of application of eq. (3.37) for a given square j=3.

DP-SLAM also defines a vector $\boldsymbol{\delta}$ as a vector of differences, such that δ_i is the distance between the laser distance measurement (the stopping point) and grid square i along the trace of the laser ray. Thus, the conditional probability of the measurement, given that the laser ray that was interrupted in square i, is $P_L(\delta_i | stop=i)$, for which is made the assumption of normally distributed measurement noise. Notice that δ_i terms are only defined if the laser measurement observes a specific stopping point, as shown in Figure 3.8; eq. (3.38) is used to compute this probability.

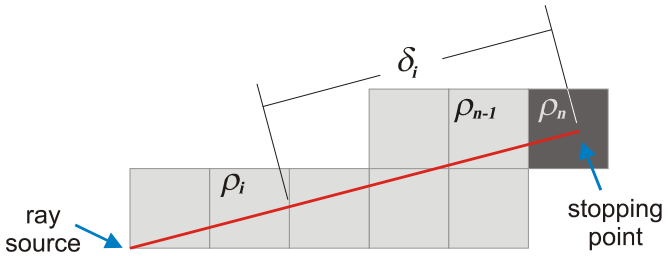


Figure 3.8: Distance between the square i and the stopping point of the laser ray

$$P_L(\delta_i \mid stop = i) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-\delta_i^2}{2\sigma^2}} \tag{3.38}$$

where σ is the standard deviation of the laser measurement.

Clearly in eq. (**3.38**), the lower distance δ_i (which means that square i is closer to the stopping point) the higher the probability. Basically, the idea of this vector of differences $\boldsymbol{\delta}$, is to create a probability distribution as shown in Figure 2.5.

Finally the probability of the laser measurement L, with an observed stopping point, is then the sum, over all grid squares in the range of the laser ray, of the product of the conditional probability of the measurement given that the ray has stopped at that point, and the probability that the ray stopped in each square:

$$P(L, stopped = true) = \sum_{i=1}^{n} P_L(\delta_i \mid stop = i) P(stop = i \mid \boldsymbol{x}, \boldsymbol{\rho}) \tag{3.39}$$

To sum it all in a nutshell, DP-SLAM creates two Gaussians. The first Gaussian computes the probability distribution of stopping the laser ray by all squares along its trajectory (note that measurement z_t must be extended by an arbitrary extra distance). The second Gaussian computes the probability distribution of the measurement z_t (as shown in Figure 2.5) according to the distance between the stopping point and all squares along its trajectory. Finally the weight of this laser ray is obtained by multiplying both distributions. Thus, the weight of an entire scan is the sum of all individual laser ray weight.

To show how this perception model works, let's discuss an example. Suppose that at time $t-1$ two equals particles p_1 and p_2 (supposing that they were selected by resampling and hence they are copies of one particle at time $t-2$), having same map m_1 and m_2 and having the same estimated robot pose R_1 and R_2 as shown in the Figure 3.9 (a).

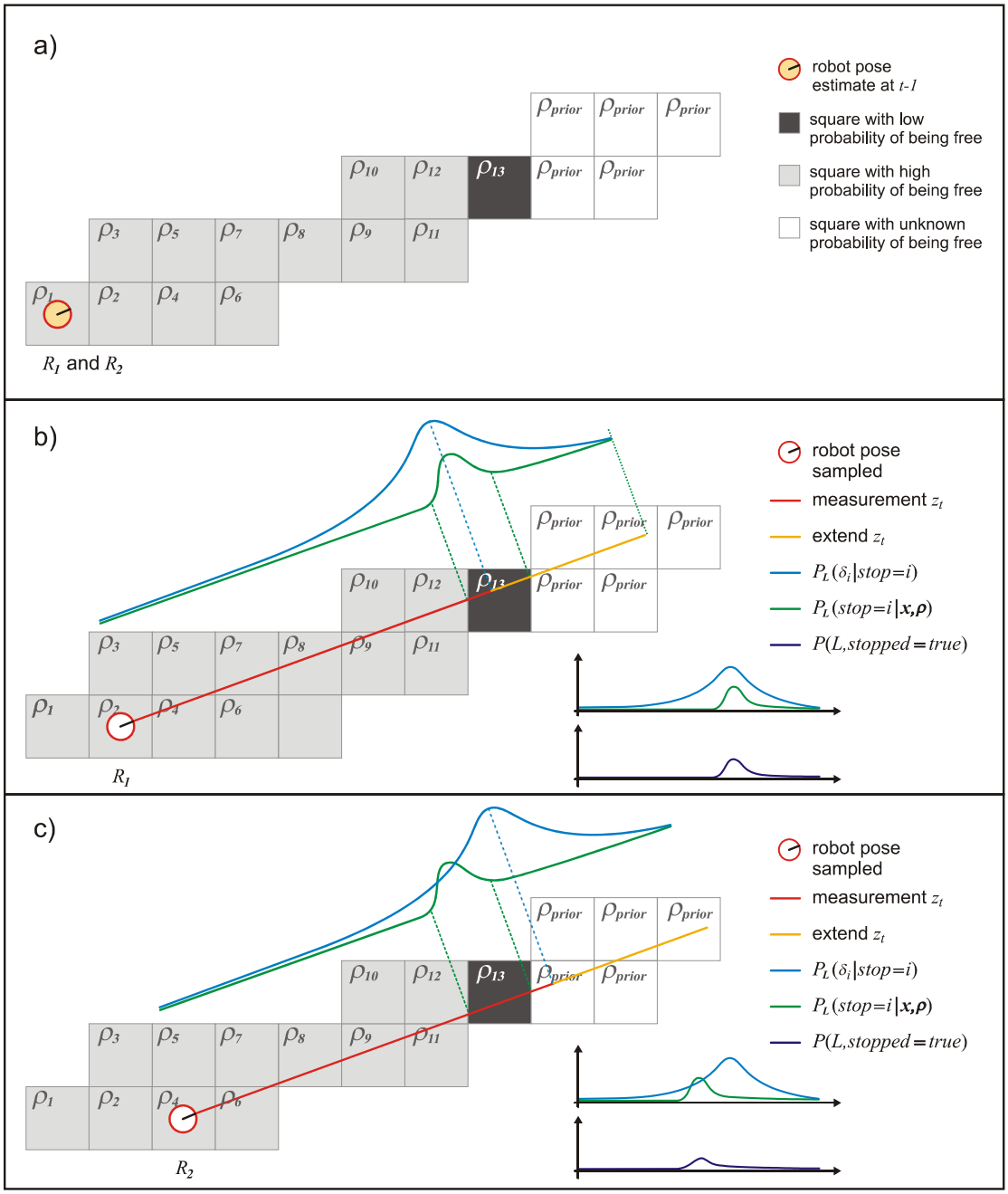


Figure 3.9: Example of computing the probability of a laser ray given two sampled robot poses.

Now suppose that the robot performs a movement u_t and then does a ray measurement z_t. Using the motion model each estimated robot pose R_1 and R_2 has a new different location (predicted), as shown in Figure 3.9 (b) and (c). Weights are acquired by putting the measurement z_t (extending the measurement by an arbitrary extra distance – yellow line in the Figure 3.9 (b) and (c)) into the predicted robot positions R_1 and R_2 and then evaluating using eq. (3.39).

As illustrated in Figure 3.9 (b) and (c); the sampled robot pose R_1 has a higher probability (or weight) than the sampled robot R_2, as shown in the result (dark blue line) of multiplying $P_L(\delta_i \mid stop = i)$ (light blue line) and $P_L(stop = i \mid \boldsymbol{x}, \boldsymbol{\rho})$ (green line). More details about the opacity parameter, ρ, and how the unknown squares (unobserved previously) are treated, can be found in [6].

"DP-SLAM implements a PF over maps and robot poses, using an occupancy grid to represent the map to track the placement of objects in the environment" [6]. Thus, for a PF to properly track this joint distribution, each particle needs to maintain a separated, complete map. During the resampling in this PF, each particle could be resampled, and consequently copied, multiple times. However, because operations must be performed merely copying maps, a direct approach to this method, where a complete map is assigned to each particle, is impractical. "For a number of particles sufficient to achieve precise localization in a reasonably sized environment, this naïve approach would require gigabytes worth of data movement per update" [6].

3.2.3.2. Ancestry Tree

The greater contribution of DP-SLAM is an efficient representation of the map, making map copying more efficient, reducing the memory required to represent large numbers of occupancy grids. DP-SLAM achieves this through a method called: *Distributed Particle Mapping* (DP-Mapping), "which exploits the significant redundancies between the different maps" [6].

A particle from the distribution at time $t-1$ is called a "parent" and its successor (sampled particle) at time t is called "child", while two children with the same parent are "siblings". If a LRF sweeps out an area of size A << M (where M is the area of the total map) and if there are two siblings s_1 and s_2 (each one with a different pose), each sibling will make updates in at most an area of size A to the map it inherits form its parent. Thus the maps for s_1 and s_2 can differ of their parent in at most an area of size A, the remainder area of the map is identical.

Then DP-SLAM proposes recording a list of changes that each particle makes to its parent.

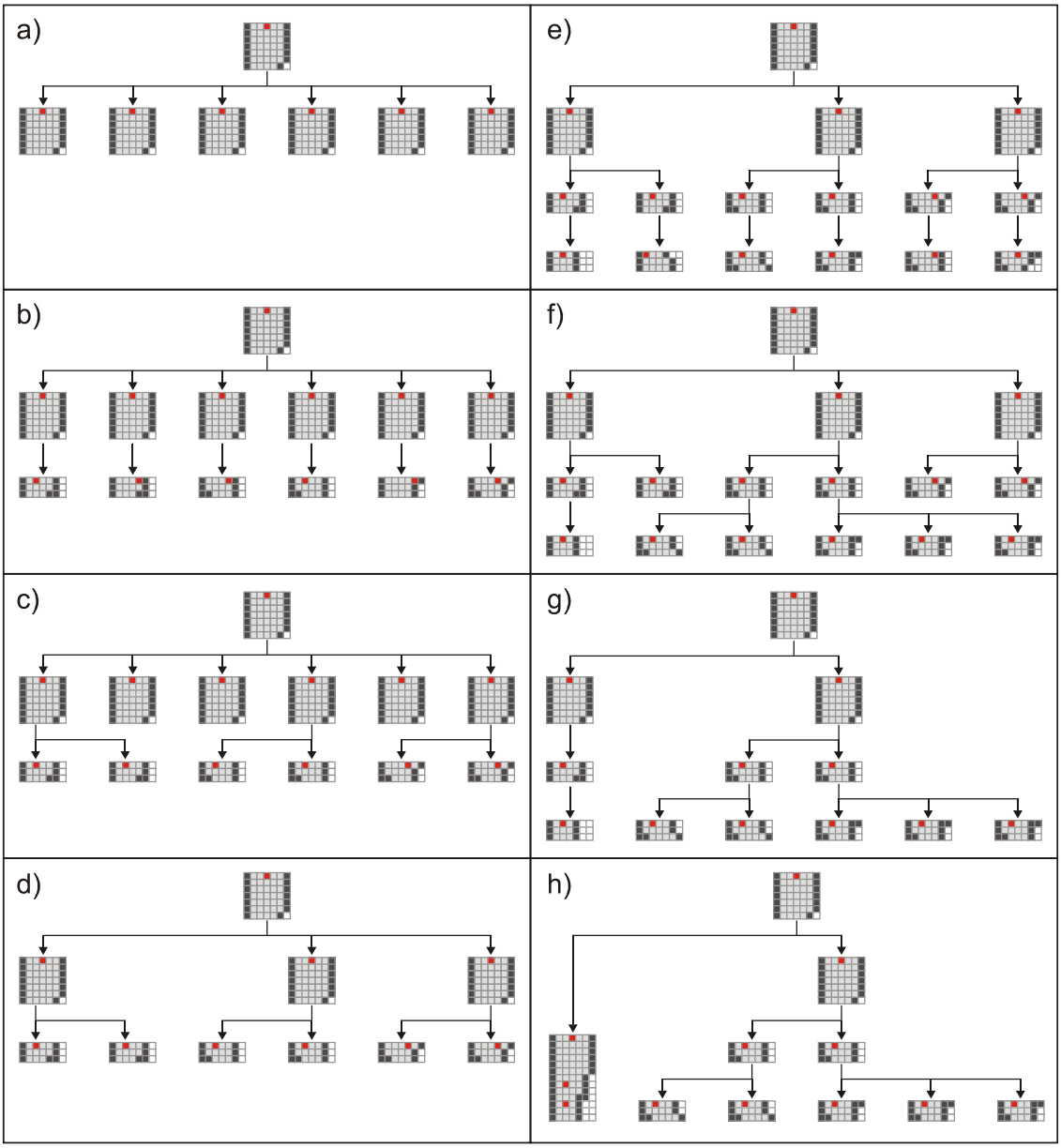


Figure 3.10: Example of particle ancestry tree maintenance

Thus DP-SLAM maintains a so called: *particle ancestry tree*, that does not forget the old particles, because to construct the entire map it is necessary not only one particle but also its ancestors. However this creates a new problem: the height of the particle ancestry tree will be linear with respect to the amount of iterations (each iteration will create a new set of particles). DP-SLAM solves this problem by defining a method of collapsing certain branches of the tree. Any particle that

does not produce any surviving children can simply be removed; this may cause a series of ancestor nodes removal. Additionally, when a particle has only a single child, it is possible to merge this particle child with the particle parent and can be treated as a single particle. This “pruning” technique is explained in Figure 3.10.

Figure 3.10 (a) depicts the beginning of the process. At the top of the figure is a single particle, where the robot’s pose is represented by a red square, and the current map in gray scale. This one particle is resampled many times, to give a number of identical children. The, these new particles are each propagated forward using the motion model. Thus, in Figure 3.10 (b), each particle represents a different pose, and each has a different set of map updates. Then these particles are weighed, based on how well the new updates are in agreement with the existing map, and finally, the particles are randomly resampled proportionately based on these weights, see Figure 3.10 (c).

At this point some particles have greater weight than others, and therefore were resampled more than once. Because the number of particles at each iteration is kept constant, consequently there are other particles which were not resampled. These particles (childless particles), can be removed from the ancestry tree, due to they will have no influence on any future particles, see Figure 3.10 (d).

In Figure 3.10 (e) and (f), the new set of particles are again propagated forward, and then weighed and resampled. However, on the right of Figure 3.10 (f), there is a pair of childless particles which can be removed and when this is done, their common parent will no longer have any children. Thus, this older ancestor can also be removed, as shown in Figure 3.10 (g). Also on the left of Figure 3.10 (f), if the childless ancestor particle is removed, there will be a chain of ancestor particles (on the left of Figure 3.10 (g)), each with one child. Therefore, these nodes can all be merged into a single ancestor particle and consequently collapsing the chain (on the left of Figure 3.10 (h)).

Maintaining the particle ancestry tree in this manner it is guaranteed that the tree will have a branching factor of at least two, and the depth of the tree will be no greater than the number of particles in each generation [6].

3.2.3.3. Hierarchical SLAM

The DP-SLAM provides an accurate and efficient method for building maps. However, there are some trajectories, which cover a sufficient amount of distance before completing a cycle, for which the accuracy of the map can degrade [6]. Small errors are accumulated over several iterations, and although the resulting map may look locally consistent, there is a large total error, which is more evident when the robot closes a large loop. This behavior over large distances is known as “drift”. It is a significant problem faced by essentially all current SLAM algorithms [6].

As a consequence of violated assumptions or as a consequence of particle filtering it is hard to avoid drift. Errors come from three sources: insufficient number of particles, coarse precision, and resampling itself (particle depletion). The consequence of these errors is a gradual but inevitable accumulation resulting from faults to sample, differentiate, or remember a state vector that is sufficiently close to the true state.

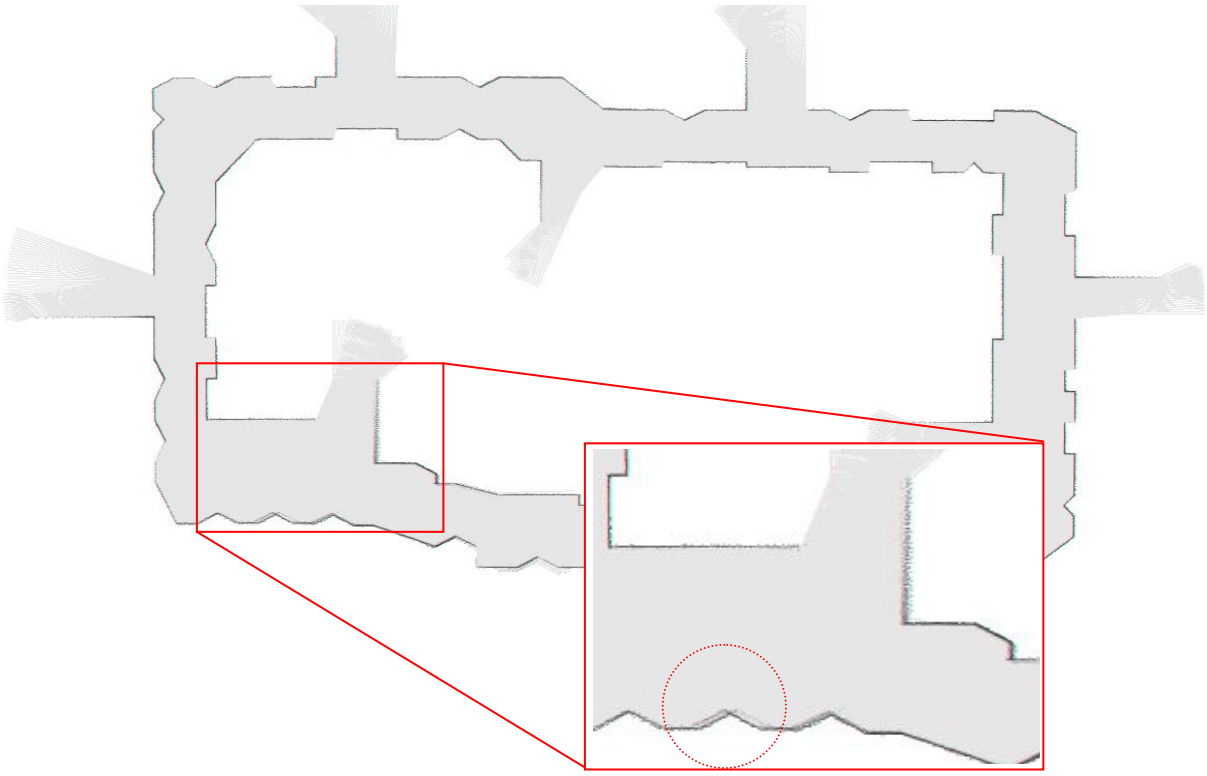


Figure 3.11: Simulated environment (60 x 40 m).

Figure 3.11 shows a loop-closed simulated environment. This map consists of 183 LRF scans. It was build using DP-SLAM (without hierarchical SLAM) with 2800 particles. This loop is large enough that particle diversity is insufficient to correct the small errors that occur.

The reason that a non-hierarchical method cannot manage this data is the extreme longevity of the uncertainty. In a large loop, small ambiguities in the beginning of the map are not resolved for many thousand iterations [6]. Non hierarchical DP-SLAM requires a huge number of particles to maintain this early particle diversity.

3.2.3.4. Hierarchical Algorithm

DP-SLAM uses two levels Hierarchical-SLAM where the lowest level models the physical process (SLAM itself), while the higher level models errors in the lower level.

"Since the total drift over trajectory is assumed to be a summation of many small, largely independent sources of error, it can be well approximated by Gaussian distribution" [6]. Thus DP-SLAM states that the effects of drift on low level maps can be precisely approximated by perturbations on the robot's trajectory endpoints used to construct a low level map.

DP-SLAM uses a standard SLAM algorithm for the low level mapping process. The low level algorithm input is a small portion of the robot's trajectory, along with the associated observations (range scans). This low level SLAM process runs normally, and its output (a trajectory) is treated as distribution over motions (motion model) in the higher level SLAM process, to which additional noise from drift is added. So the output from each of small mapping is the input for a new SLAM process, working at a higher level of time steps.

Because the sampled trajectory is treated as an atomic motion, this defines the placement of the associated observation. "The observation model at the high

level is then just the collection of observations that were made at each step along this trajectory" [6].

The high level SLAM loop for each high level particle is summarized as follows [6]:

1. Sample a high level SLAM state (high level map and robot state).
2. Perturb the sampled robot state by adding random drift.
3. Sample a low level trajectory from the distributions over trajectories returned by the low level SLAM process.
4. Compute a high-level weight by evaluating the trajectory and robot observations against the sampled high level map, starting from the perturbed robot state.
5. Update the high level map based upon the new observations.

Figure 3.12 shows an example of hierarchical SLAM. The entire map is divided into 10 small maps (light green and light blue to distinguish between them). Notice that when the loop is closing, the best path until there (represented by red lines) has a misalignment. This means that there is something wrong in its trajectory. Because it is using hierarchical DP-SLAM there is enough particle diversity and the ambiguities in the beginning of the map can be resolved for, now, 9 iterations (10 small maps).

Thus, resolving ambiguities leads to the map from Figure 3.13. Here the best path (red lines) is different one and therefore the map it carries, a better one, is different too.

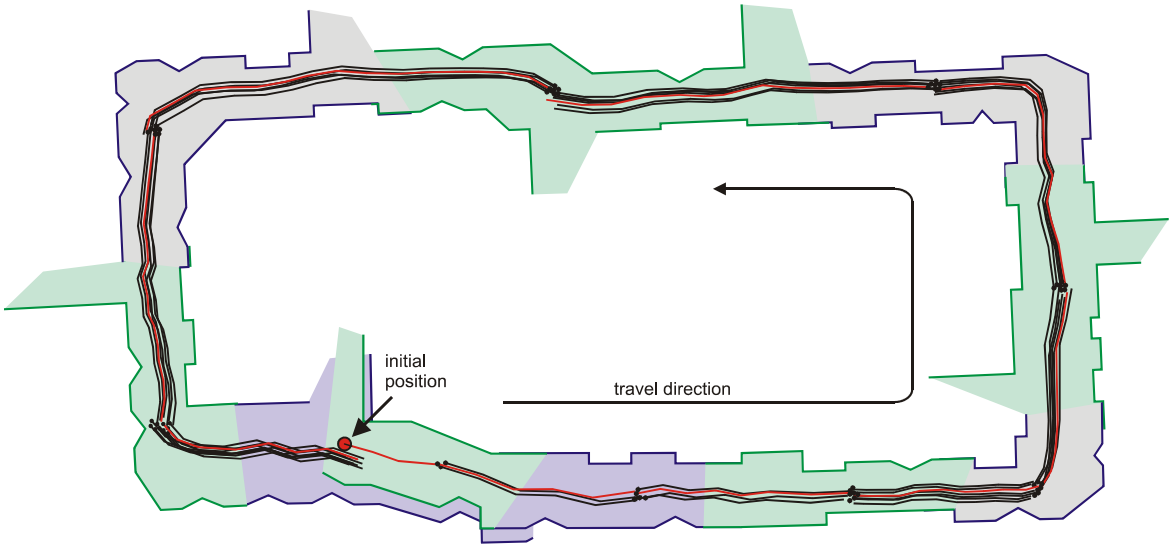


Figure 3.12: Mapping closing a loop. Each black dot is the perturbed endpoints of trajectories.

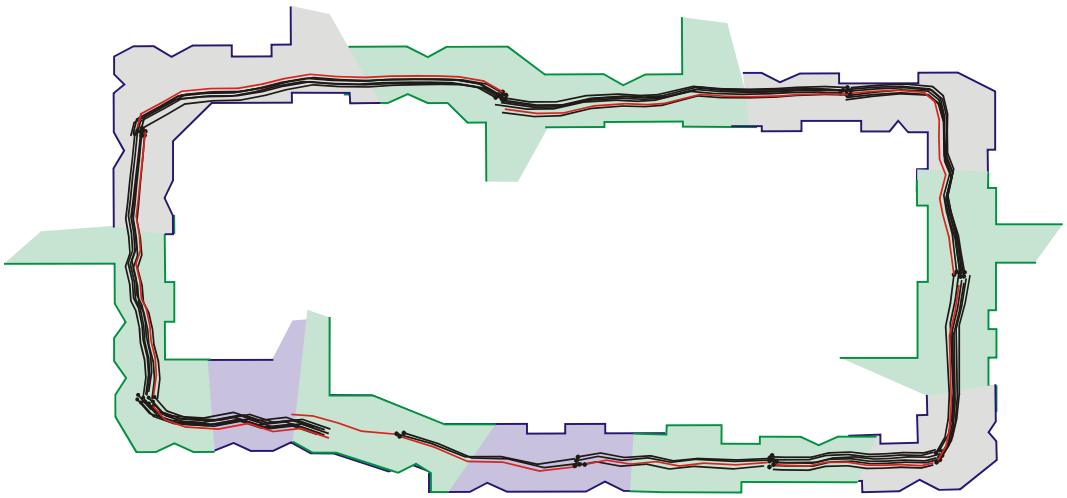


Figure 3.13: Map after ambiguities are resolved.

It is possible to implement the hierarchical SLAM for multiple levels for providing more robustness. This idea of hierarchical SLAM is not restricted to be used solely with DP-SLAM; this method could be effective when used with any other SLAM method [6].

3.3. 3D SLAM Review

Existing SLAM methods produce a two-dimensional cross section of the world, and robot motion is restricted to motion within this plane. However the assumption of a 2D world is unrealistic: wheeled robots traveling across uneven terrain and underwater autonomous vehicles, can all move with six degrees of freedom, three translational and three angular. For robots to operate in this environment, it is not only need to track these three new degrees of motion, but also to maintain a three dimensional representation of the environment.

3D mapping has some advantages compared with 2D [1]:

- 3D maps facilitate navigation. Many robot environments possess significant variation in occupancy in the vertical dimension. Modeling this can greatly reduce the danger of colliding with an obstacle.
- Many robot tasks require three-dimensional information, such as tasks involving the localization and retrieval of objects or people.
- 3D maps carry much information for a potential user of the maps. If one builds a map only for the sake or robot navigation, then the SLAM enforcements would be very few. However, if the map is acquired for later use by a person, 3D information can be absolutely critical.

Some methods exist for three-dimensional motion. They tend to represent the world in terms of a few sparse, pint-sized landmarks. These maps, while useful for localization, and possible for navigation, give very little information about the presence of objects in the world. In [25] a SLAM framework based on 3D landmarks for indoor environment with stereo vision is shown. Reference [26] shows a real-time 3D SLAM is constructed using wide-angle vision.

Carnegie Mellon University´s Mine Mapping project is a notable example of volumetric three dimensional maps, using a series of LRF set at different angles [27]. Using a combined method of both local and global scan matching techniques, a two-dimensional occupancy grid is created. Thus, with the corresponding trajectory from the robot, the remaining three-dimensional data are

filled in to create the volumetric maps. Reference [28] presents an EKF-based 3D SLAM, which uses planar features probabilistically extracted from dense three-dimensional point clouds generated by a rotated 2D LRF. A similar work [29] presents SLAM from visual landmarks and 3D planes, modeling the environment as a set of planar surfaces and lines. These planar surfaces and lines are extracted by fusing data from a camera and a 3D LRF.

Also DP-SLAM [6] proposes a 3D grid map representation. This representation brings two types of challenges, technical and dimensional. The technical problems are mainly issues of sensing. In particular, odometry is unable to detect any motion in the three new degrees of freedom. The dimensional challenges arise from a new dimension added to the problem. The resources needed to deal with SLAM grow exponentially, so that merely extending previous methods is infeasible on any computer architecture.

However, this thesis is focused in indoor structured environments. Assuming a flat terrain, the localization given by the 2D DP-SLAM, can be used to project the corresponding 3D points. Thus a 3D map is constructed, composed by a set of points (a point cloud). This proposed method has similarities with the one presented in [27] where 3D maps are obtained by using the 2D pose information via the geometric projections.

Chapter 5 will show some results by applying DP-SLAM in simulated data. After creating a 2D map, the DP-SLAM algorithm gives the best estimated path. Using this, the corresponding 3D points are projected. The implemented simulator is discussed in detail in the next chapter.