4.
Detailed Implementation

4.1.
EKF SLAM

The Extended Kalman Filter (EKF) fuses all available information about the system's state to compute a state estimate. This is accomplished through a recursive three-stage cycle consisting of *prediction*, *observation* and *update* steps.

1. Prediction

This step involves computing $\overline{\lambda_t}$ and $\overline{\Sigma}_t$.

As seen in eq. (3.20), $\overline{\lambda_t}$ depends on u_t and λ_{t-1}. Using a naïve example, the control u_t could be one of two types: the first specifies the translational and rotational velocities, and the second specifies odometry information (such as distance traveled, angle turned). That is:

$$u_t = \begin{bmatrix} v_t \\ \omega_t \end{bmatrix} \quad \text{or} \quad u_t = \begin{bmatrix} d \\ \varphi \end{bmatrix} \tag{4.1}$$

Thus, the uncertainty of the control u_t is given by its covariance matrices:

$$U = \begin{bmatrix} \sigma_v & 0 \\ 0 & \sigma_\omega \end{bmatrix} \quad \text{or} \quad U = \begin{bmatrix} \sigma_d & 0 \\ 0 & \sigma_\varphi \end{bmatrix} \tag{4.2}$$

Using a landmark map representation, λ_{t-1} is a vector containing the robot and landmark positions at one time step earlier. That is:

$$\lambda_{t-1} = [\underbrace{Rx_{t-1} \quad Ry_{t-1} \quad R\theta_{t-1}}_{\text{Robot position}} \quad \underbrace{L_1x_{t-1} \quad L_1y_{t-1}}_{\text{Landmark 1 position}} \quad \ldots \quad \underbrace{L_nx_{t-1} \quad L_ny_{t-1}}_{\text{Landmark n position}}]^T \tag{4.3}$$

Figure 4.1 shows the robot position at the previous time step, *t–1*, and the predicted robot position made by the odometry information in a Cartesian plane. The predicted state vector $\overline{\lambda}_t$ is given by

$$f(u_t, \lambda_{t-1}) = \begin{bmatrix} \overline{R}x \\ \overline{R}y \\ \overline{R}\theta \end{bmatrix} = \begin{bmatrix} Rx_{t-1} + d\cos(R\theta_{t-1} + \varphi) \\ Ry_{t-1} + d\sin(R\theta_{t-1} + \varphi) \\ R\theta_{t-1} + \varphi \end{bmatrix} \tag{4.4}$$

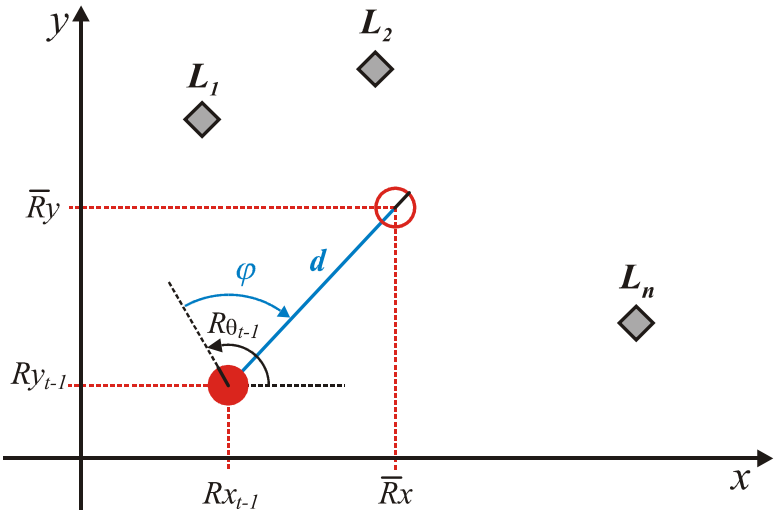

Figure 4.1: Robot position (red fill circle) at time *t–1* , and predicted robot position (white filled circle).

Notice however, that it is not necessary to predict landmark positions. Thus, the predicted state vector $\overline{\lambda}_t$ is now

$$\overline{\lambda}_t = \left[\overline{R}x \;\; \overline{R}y \;\; \overline{R}\theta \;\; \overline{L}_1x \;\; \overline{L}_1y \;\; \overline{L}_2x \;\; \overline{L}_2y \;\; \ldots \;\; \overline{L}_nx \;\; \overline{L}_ny\right]^T$$

where

$$\left[\overline{L}_1x \;\; \overline{L}_1y \;\; \overline{L}_2x \;\; \overline{L}_2y \;\; \ldots \;\; \overline{L}_nx \;\; \overline{L}_ny\right]^T = \left[L_1x_{t-1} \;\; L_1y_{t-1} \;\; L_2x_{t-1} \;\; L_2y_{t-1} \;\; \ldots \;\; L_nx_{t-1} \;\; L_ny_{t-1}\right]^T$$

In order to compute the predicted covariance $\overline{\Sigma}_t$, as in eq. (3.21), F_t is given by:

$$F_t = \frac{\partial f(u_t, \lambda_{t-1})}{\partial_{\lambda_{t-1}}} = \begin{bmatrix} 1 & 0 & -d\sin(R\theta_{t-1} + \varphi) \\ 0 & 1 & d\cos(R\theta_{t-1} + \varphi) \\ 0 & 0 & 1 \end{bmatrix} \tag{4.5}$$

Notice that F_t is a matrix of size m x m, where m is the dimension of the state vector (3+2n), being n the number of landmarks. However since the environment is stationary, again, movement of landmarks does not need to be predicted, thus the remaining elements in F_t are zero.

P_t is given by eq. (4.6) where U was described in eq. (4.2) and Jf_{u_t} is given by:

$$Jf_{u_t} = \frac{\partial f(u_t, \lambda_{t-1})}{\partial_{u_t}} = \begin{bmatrix} \cos(R\theta_{t-1} + \varphi) & -d\sin(R\theta_{t-1} + \varphi) \\ \sin(R\theta_{t-1} + \varphi) & d\cos(R\theta_{t-1} + \varphi) \\ 0 & 1 \end{bmatrix} \tag{4.6}$$

$$P_t = Jf_{u_t}\, U\, Jf_{u_t}^T \tag{4.7}$$

In the same way, Jf_{u_t} is of size m x 2; however the elements related to landmarks are zero. Thus the mean $\overline{\lambda}_t$ is the predicted state vector of size (3+2n) x 1 and $\overline{\Sigma}_t$ the predicted covariance matrix of size (3+2n) x (3+2n).

2. Observation

The observation step computes the innovation vector and the innovation covariance, that is $\left[z_t - h(\overline{\lambda}_t)\right]$ and $\left(H_t\, \overline{\Sigma}_t\, H_t^T + Q\right)$ in the eq. (3.23) and eq. (3.22), respectively.

The perception function, $h(\overline{\lambda}_t)$, that represents the predicted landmark position seen from the predicted robot position is shown in Figure 4.2.

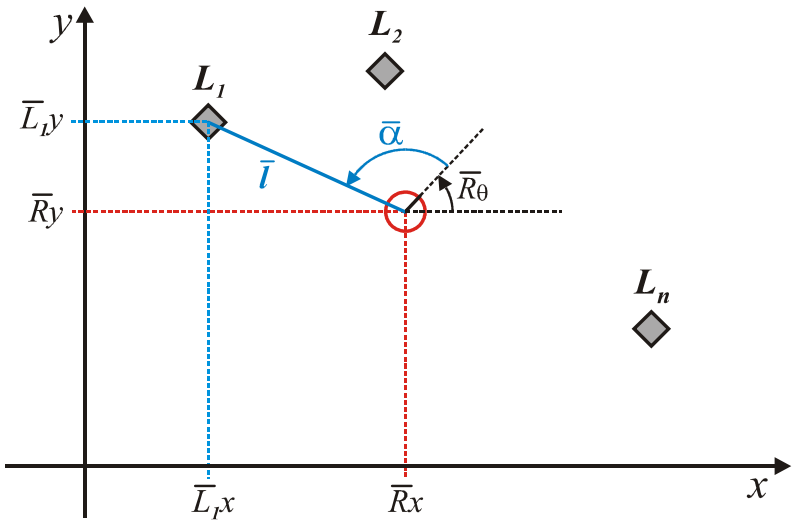

Figure 4.2: Predicted landmark position seen from the predicted robot position

In the same Cartesian plane, the predicted distance and angle ($\bar{l}$ and $\bar{\alpha}$) from the predicted robot position to the landmark L_1, is given by eq. (4.8):

$$h(\bar{\lambda}) = \begin{bmatrix} \bar{l} \\ \bar{\alpha} \end{bmatrix} = \begin{bmatrix} \sqrt{(\bar{L}_1 x - \bar{R}x)^2 + (\bar{L}_1 y - \bar{R}y)^2} \\ \arctan(\bar{L}_1 y - \bar{R}y / \bar{L}_1 x - \bar{R}x) - \bar{R}\theta \end{bmatrix} \tag{4.8}$$

Note that $h(\bar{\lambda}_t)$ shown in eq. (4.8) is written for the Landmark 1, in the same way it must be computed for the n landmarks. Thus, the size of the vectors $h(\bar{\lambda}_t)$ and the z_t is $2n$.

To compute the matrix H_t, the Jacobian of h at $\bar{\lambda}_t$ is given by

$$H_t = \frac{\partial h(\bar{\lambda})}{\partial_{\bar{\lambda}}} = \begin{bmatrix} \frac{\partial}{\partial \bar{R}x}\bar{l} & \frac{\partial}{\partial \bar{R}y}\bar{l} & \frac{\partial}{\partial \bar{R}\theta}\bar{l} & \frac{\partial}{\partial \bar{L}_1 x}\bar{l} & \frac{\partial}{\partial \bar{L}_1 y}\bar{l} \\ \frac{\partial}{\partial \bar{R}x}\bar{\alpha} & \frac{\partial}{\partial \bar{R}y}\bar{\alpha} & \frac{\partial}{\partial \bar{R}\theta}\bar{\alpha} & \frac{\partial}{\partial \bar{L}_1 x}\bar{\alpha} & \frac{\partial}{\partial \bar{L}_1 y}\bar{\alpha} \end{bmatrix} \tag{4.9}$$

$$H_t = \begin{bmatrix} \frac{-d_x}{\sqrt{d_x^2 + d_y^2}} & \frac{-d_y}{\sqrt{d_x^2 + d_y^2}} & \mathbf{0} & \frac{d_x}{\sqrt{d_x^2 + d_y^2}} & \frac{d_y}{\sqrt{d_x^2 + d_y^2}} \\ \frac{d_y}{d_x^2 + d_y^2} & \frac{-d_x}{d_x^2 + d_y^2} & -1 & \frac{-d_y}{d_x^2 + d_y^2} & \frac{d_x}{d_x^2 + d_y^2} \end{bmatrix}$$

where

$$d_x = \bar{L}_1 x - \bar{R}x \quad \text{and} \quad dy = \bar{L}_1 y - \bar{R}y$$

Notice however, that eq. (4.9) shows H_t using only the Landmark 1, thus the true size of H_t is 2 x (3+2n).

$$H_t = \frac{\partial h(\bar{\lambda})}{\partial_{\bar{\lambda}}} = \begin{bmatrix} \frac{\partial}{\partial \bar{R}} \bar{l} & \frac{\partial}{\partial L_1} \bar{l} & \dots & \frac{\partial}{\partial L_n} \bar{l} \\ \frac{\partial}{\partial \bar{R}} \bar{\alpha} & \frac{\partial}{\partial L_1} \bar{\alpha} & \dots & \frac{\partial}{\partial L_n} \bar{\alpha} \end{bmatrix} \tag{4.10}$$

and Q_t represents the covariance matrix for the sensor noise:

$$Q = \begin{bmatrix} \sigma_l & 0 \\ 0 & \sigma_\alpha \end{bmatrix} \tag{4.11}$$

3. Update

Finally, this last step computes the desired λ_t and Σ_t, using eq. (3.22), eq. (3.23) and eq. (3.24).

It is important to point out that in the beginning, where the robot starts the mapping process, it may not see all landmarks, or not even one. Therefore, as the robot moves through the environment, the state vector λ_t grows. Thus, when a new landmark is acquired, the state vector is augmented. Figure 4.3 shows how the robot sees a new landmark (p) at distance l and at angle α; eq. (4.12) and eq. (4.13) show how this is modeled.

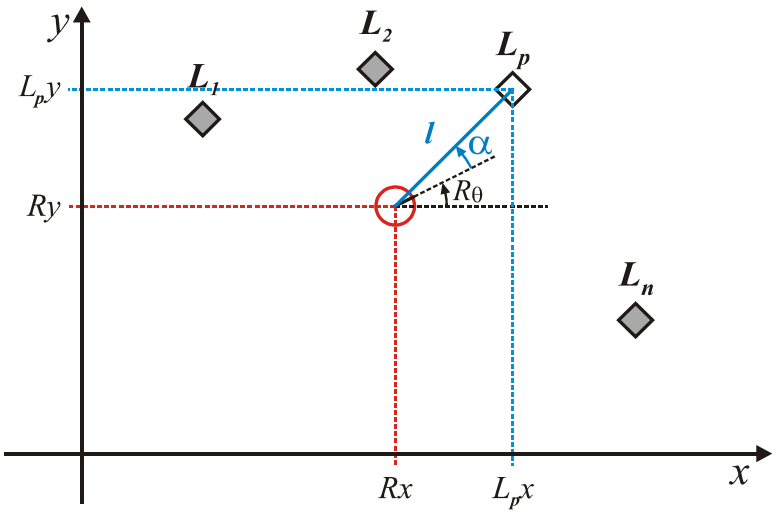

Figure 4.3: New landmark L_p, is added to the state vector.

$$z_p = \begin{bmatrix} L_p x \\ L_p y \end{bmatrix} = \begin{bmatrix} Rx + l\cos(R\theta + \alpha) \\ Ry + l\sin(R\theta + \alpha) \end{bmatrix} \tag{4.12}$$

$$\lambda_t := \begin{bmatrix} \lambda_t \\ L_p \end{bmatrix} \tag{4.13}$$

As it was seen in Section 3.2.2, the covariance matrix Σ_t, represents the relationships landmarks-robot and landmarks-landmarks (because the robot's pose uncertainty correlates landmark positions), these relationships are shown in eq. (4.14).

$$\Sigma_t = \begin{bmatrix} C_{RR} \ (3\text{x}3) & C_{RL_1} \ (3\text{x}2) & \cdots & C_{RL_n} \ (3\text{x}2) \\ C_{RL_1}^T \ (2\text{x}3) & C_{L_1L_1} \ (2\text{x}2) & \cdots & C_{L_1L_n} \ (2\text{x}2) \\ \vdots & \vdots & \ddots & \\ C_{RL_n}^T \ (2\text{x}3) & C_{L_1L_n}^T \ (2\text{x}2) & & C_{L_nL_n} \ (2\text{x}2) \end{bmatrix} \tag{4.14}$$

In the same way the covariance matrix Σ_t grows in dimension to include the new landmark. To do this, the covariance matrices, C_{LpLp} (L_p to L_p), C_{RLp} (robot to L_p) and C_{L1Lp} … C_{LnLp} (all old landmarks L_1… L_n to L_p) are computed by eq. (4.15), eq. (4.16) and eq. (4.17), respectively.

$$C_{LpLp} = \underset{Rx,Ry,R\theta}{Jz_p}\, C_{RR}\, \underset{Rx,Ry,R\theta}{Jz_p}^{T} + \underset{l,\alpha}{Jz_p}\, Q_t\, \underset{l,\alpha}{Jz_p}^{T} \tag{4.15}$$

$$C_{RLp} = \underset{Rx,Ry,R\theta}{Jzp}\, C_{RR} \tag{4.16}$$

$$C_{L_1Lp} \dots C_{LnLp} = \underset{Rx,Ry,R\theta}{Jz_p} \begin{bmatrix} C_{RL_1} & \dots & C_{RL_n} \end{bmatrix} \tag{4.17}$$

where Jz_p are the Jacobians:

$$\underset{Rx,Ry,R\theta}{Jz_p} = \begin{bmatrix} \frac{\partial}{\partial Rx} L_p x & \frac{\partial}{\partial Ry} L_p x & \frac{\partial}{\partial R\theta} L_p x \\ \frac{\partial}{\partial Rx} L_p y & \frac{\partial}{\partial Ry} L_p y & \frac{\partial}{\partial R\theta} L_p y \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\boldsymbol{l}\cos(R\theta+\alpha) \\ 0 & 1 & \boldsymbol{l}\sin(R\theta+\alpha) \end{bmatrix} \tag{4.18}$$

$$\underset{l,\alpha}{Jz_p} = \begin{bmatrix} \frac{\partial}{\partial \boldsymbol{l}} L_p x & \frac{\partial}{\partial \alpha} L_p x \\ \frac{\partial}{\partial \boldsymbol{l}} L_p y & \frac{\partial}{\partial \alpha} L_p y \end{bmatrix} = \begin{bmatrix} \cos(R\theta+\alpha) & -\boldsymbol{l}\sin(R\theta+\alpha) \\ \sin(R\theta+\alpha) & \boldsymbol{l}\cos(R\theta+\alpha) \end{bmatrix} \tag{4.19}$$

Finally, the augmented covariance matrix looks like eq. (4.20), representing now: *n:=n+1* landmarks.

$$\sum_t = \begin{bmatrix} C_{RR} \ (3x3) & C_{RL_1} \ (3x2) & \cdots & C_{RL_n} \ (3x2) & C_{RL_p} \ (3x2) \\ C_{RL_1}^T \ (2x3) & C_{L_1L_1} \ (2x2) & \cdots & C_{L_1L_n} \ (2x2) & C_{L_1L_p} \ (2x2) \\ \vdots & \vdots & \ddots & & \vdots \\ C_{RL_n}^T \ (2x3) & C_{L_1L_n}^T \ (2x2) & & C_{L_nL_n} \ (2x2) & C_{L_nL_p} \ (2x2) \\ C_{RL_p}^T \ (2x3) & C_{L_1L_p}^T \ (2x2) & \cdots & C_{L_nL_p}^T \ (2x2) & C_{L_pL_p} \ (2x2) \end{bmatrix} \quad (4.20)$$

4.2. FastSLAM

The Fast Slam algorithm receives as input the previous distribution $p(x_{t-1})$, as a particle set $\Phi_{t-1} := \{\langle s_{t-1}^i \rangle / i = 1...N\}$, the odometry (or control) information u_t, and the landmark measurements z_t. From the idea exposed in Section 2.2.4, the FastSLAM algorithm uses the following steps.

1. Each particle *i* representing the robot position at time *t*−1, R_{t-1}^i, from the set Φ_{t-1}, is moved following the control vector u_t and the motion model.

$$u_t = \begin{bmatrix} v_t \\ \omega_t \end{bmatrix} \quad \text{or} \quad u_t = \begin{bmatrix} d_t \\ \varphi_t \end{bmatrix} \quad (4.21)$$

Table 4.1 shows an algorithm [1] for sampling from the motion model $p(R_t \mid u_t, R_{t-1})$ to generate a random pose R_t^i. Lines 1 and 2 perturb the commanded control parameters with noise, drawn from the error parameters α_1 to α_6 (a detailed explanation of these error parameters can be found in [1]). The noise values are then used to generate the sample's new pose in lines 4 through 7.

Table 4.1: Sample Motion Model Algorithm [1]

Sample Motion Model_algorithm (R_{t-1}, u_t)	
1:	$\hat{v} = v_t + sample(\alpha_1 \mid v_t \mid + \alpha_1 \mid \omega_t \mid)$
2:	$\hat{\omega} = \omega_t + sample(\alpha_3 \mid v_t \mid + \alpha_4 \mid \omega_t \mid)$
3:	$\hat{\gamma} = sample(\alpha_5 \mid v_t \mid + \alpha_6 \mid \omega_t \mid)$
4:	$Rx_t = Rx_{t-1} - \frac{\hat{v}}{\hat{\omega}}\sin(R\theta_{t-1}) + \frac{\hat{v}}{\hat{\omega}}\sin(R\theta_{t-1} + \hat{\omega}\Delta t)$
5	$Ry_t = Ry_{t-1} + \frac{\hat{v}}{\hat{\omega}}\cos(R\theta_{t-1}) - \frac{\hat{v}}{\hat{\omega}}\cos(R\theta_{t-1} + \hat{\omega}\Delta t)$
6:	$R\theta_t = R\theta_{t-1} + \hat{\omega}\Delta t + \hat{\gamma}\Delta t$
7:	*return* $R_t=(Rx_t, Ry_t, R\theta_t)$

Figure 4.4 illustrates the outcome of this sampling routine. A temporal particle set $\hat{\Phi} = \{R_t^1, R_t^2, ...R_t^p\}$ containing all the samples is created, where p is the number of samples (particles).

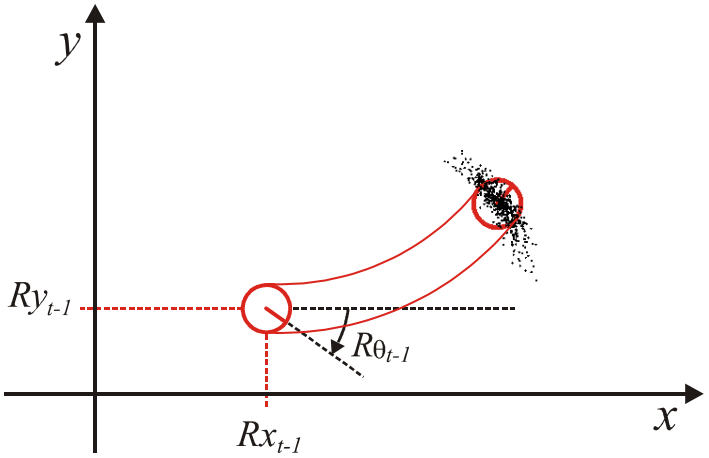

Figure 4.4: Sampling. Each black dot represents a possible robot position.

2. Update the estimation of landmarks, using the EKF, the set $\hat{\Phi}$ of robot poses samples created in step 1, and the landmark measurements z_t.

   Because the j^{th} landmark positions ($L_j^i x_{t-1}$, $L_j^i y_{t-1}$) is known at time $t-1$ as well as the poses ($R^i x_t$, $R^i y_t$) and rotations ($R^i \theta_t$) of each particle

(obtained in step 1), it is possible to compute the distance and angle between them, as shown in Figure 4.5, where $\hat{r}_j^i$ and $\hat{\phi}_j^i$ are the distance and angle between particle *i* and landmark *j*.

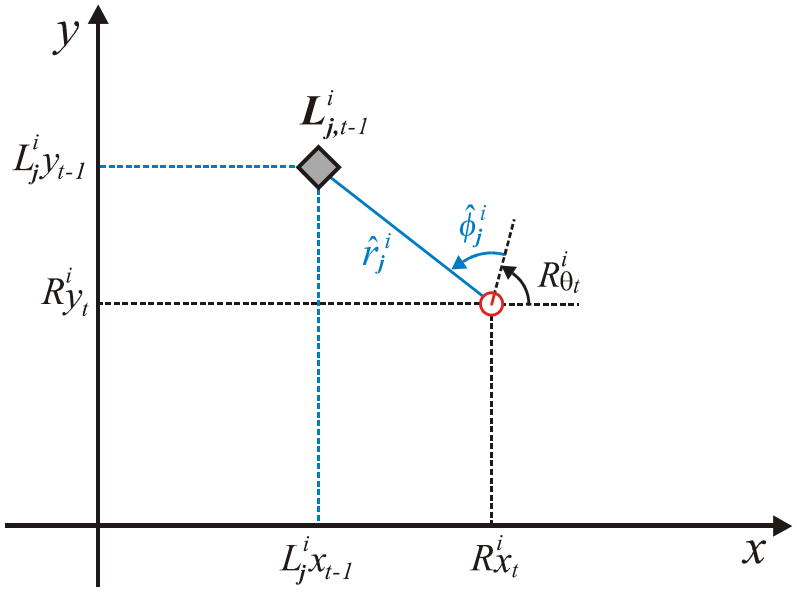

Figure 4.5: Distance and rotation between particle *i* and landmark *j*.

Thus, function *h* used by the EKF is given by:

$$h_j^i = \begin{bmatrix} \hat{r}_j^i \\ \hat{\phi}_j^i \end{bmatrix} = \begin{bmatrix} \sqrt{(L_j^i x_{t-1} - R^i x_t)^2 + (L_j^i y_{t-1} - R^i y_t)^2} \\ \arctan((L_j^i y_{t-1} - R^i y_t)/(L_j^i x_{t-1} - R^i x_t)) - R^i \theta_t \end{bmatrix} \tag{4.22}$$

and the Jacobian of h_j^i, named H_j^i, is

$$H_j^i = \frac{\partial h_j^i}{\partial_{L_{j,t-1}}} = \begin{bmatrix} \frac{\partial}{\partial_{L_j x_{t-1}}} \hat{r}_j^i & \frac{\partial}{\partial_{L_j y_{t-1}}} \hat{r}_j^i \\ \frac{\partial}{\partial_{L_j x_{t-1}}} \hat{\phi}_j^i & \frac{\partial}{\partial_{L_j y_{t-1}}} \hat{\phi}_j^i \end{bmatrix} = \begin{bmatrix} \frac{a}{\sqrt{a^2+b^2}} & \frac{b}{\sqrt{a^2+b^2}} \\ \frac{-b}{a^2+b^2} & \frac{a}{a^2+b^2} \end{bmatrix} \tag{4.23}$$

where

$$a = L_j^i x_{t-1} - R^i x_t \qquad b = L_j^i y_{t-1} - R^i y_t$$

In addition, the covariance of the landmark positions $\Sigma^{i}_{j,t-1}$ is known. Thus, the landmark updates are computed as follows:

$$S^{i}_{j} = H^{i}_{j}\,\Sigma^{i}_{j,t-1}\,H^{i\,T}_{j} + Q \tag{4.24}$$

$$W^{i}_{j} = \Sigma^{i}_{j,t-1} H^{i\,T}_{j}\,S^{i\,-1}_{j} \tag{4.25}$$

$$Z = z_{j,t} - h^{i}_{j} \tag{4.26}$$

$$\lambda^{i}_{j,t} = \lambda^{i}_{j,t-1} + W^{i}_{2}\,Z \tag{4.27}$$

$$\Sigma^{i}_{j,t} = \Sigma^{i}_{j,t-1} - W^{i}_{j}\,S^{i}_{j}\,W^{i\,T}_{j} \tag{4.28}$$

where $z_{j,t}$ is the j^{th} landmark sensor observation in z_t and Q represents the covariance matrix for the sensor noise, that is:

$$Q = \begin{bmatrix} \sigma_{d} & 0 \\ 0 & \sigma_{\phi} \end{bmatrix} \tag{4.29}$$

The following figure shows the updated landmark j for the particle i, namely $L^{i}_{j,t}$, represented by the blue arrow.

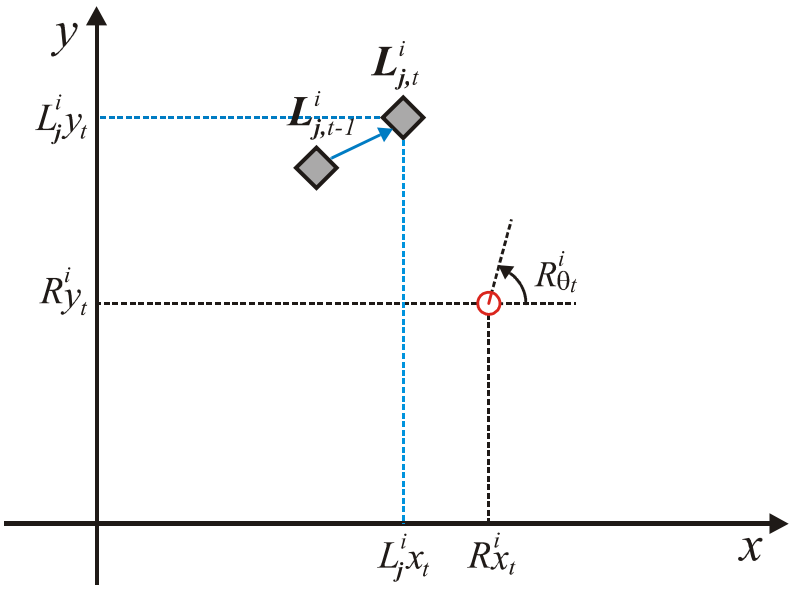

Figure 4.6: Updated landmark j for the particle i.

Note that this update is made for landmark *j* and particle *i*, thus it must be repeated *n* x *p* times, being *n* the number of observed landmarks at time *t* (the unobserved landmarks are not updated) and *p* the number of particles.

This set of median and covariance are added to the temporal particles set, as follows:

$$\begin{aligned}\hat{\Phi} = \{ & R_t^1, \lambda_{1,t}^1, \Sigma_{1,t}^1, ..., \lambda_{n,t}^1, \Sigma_{n,t}^1, \\ & R_t^2, \lambda_{1,t}^2, \Sigma_{1,t}^2, ..., \lambda_{n,t}^2, \Sigma_{n,t}^2, \\ & R_t^p, \lambda_{1,t}^p, \Sigma_{1,t}^p, ..., \lambda_{n,t}^p, \Sigma_{n,t}^p \}\end{aligned} \tag{4.30}$$

3. Assign the weight for each particle. Using the measurement vector z_t, each particle of the temporal particle set $\hat{\Phi}$ is evaluated.

   Because the j^{th} updated landmark positions ($L_j^i x_t$, $L_j^i y_t$) and the poses ($R^i x_t$, $R^i y_t$) and rotations ($R^i \theta_t$) of each particle is known (obtained in the 1), it is possible, once more, to compute the distance and angle between them, as shown in Figure 4.7, where r_j^i and ϕ_j^i are the distance and angle between particle *i* and the updated landmark *j*.

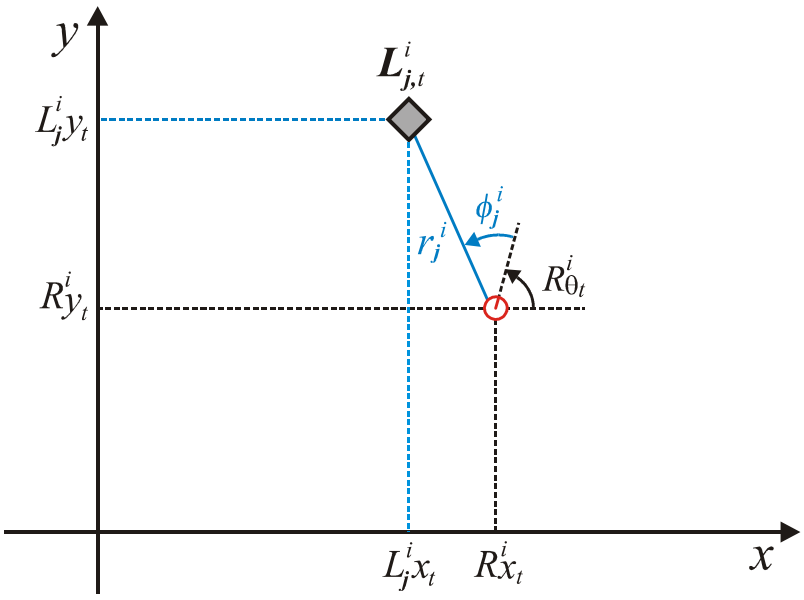

Figure 4.7: Distance and rotation between particle *i* and updated landmark *j*.

Now, the function h_j^i is

$$h_j^i = \begin{bmatrix} r_j^i \\ \phi_j^i \end{bmatrix} = \begin{bmatrix} \sqrt{(L_j^i x_t - R^i x_t)^2 + (L_j^i y_t - R^i y_t)^2} \\ \arctan((L_j^i y_t - R^i y_t)/(L_j^i x_t - R^i x_t)) - R^i \theta_t \end{bmatrix} \tag{4.31}$$

and the Jacobian of h_j^i, defined as H_j^i, becomes

$$H_j^i = \frac{\partial h_j^i}{\partial_{L_{j,t}}} = \begin{bmatrix} \frac{a}{\sqrt{a^2+b^2}} & \frac{b}{\sqrt{a^2+b^2}} \\ \frac{-b}{a^2+b^2} & \frac{a}{a^2+b^2} \end{bmatrix} \tag{4.32}$$

where

$$a = L_j^i x_t - R^i x_t \qquad b = L_j^i y_t - R^i y_t$$

The updated covariance of the landmark positions $\Sigma_{j,t}^i$ is also known. Thus, the importance factors (weights) can be computed, as follows:

Table 4.2: Compute weights algorithm;

Compute weights_algorithm ($\hat{\Phi}$, z_t)
1: for i =1 to p do
2: $w_t^i = 1$
3: for j=1 to n do
4 $U_j^i = H_j^i \Sigma_{j,t}^i H_j^{iT} + Q$
5: $w_t^i := w_t^i * \det(2\pi U_j^i)^{-\frac{1}{2}} \exp\{-\frac{1}{2}(z_{j,t} - h_j^i)^T U_j^{i\,-1}(z_{j,t} - h_j^i)\}$
6: end for
7: end for

where, once again, $z_{j,t}$ is the j^{th} landmark sensor observation. This set of importance factors is added to the temporal particles set:

$$\begin{aligned}\hat{\Phi} = \{ & w_t^1, R_t^1, \lambda_{1,t}^1, \Sigma_{1,t}^1, ..., \lambda_{n,t}^1, \Sigma_{n,t}^1, \\ & w_t^2, R_t^2, \lambda_{1,t}^2, \Sigma_{1,t}^2, ..., \lambda_{n,t}^2, \Sigma_{n,t}^2, \\ & w_t^p, R_t^p, \lambda_{1,t}^p, \Sigma_{1,t}^p, ..., \lambda_{n,t}^p, \Sigma_{n,t}^p \}\end{aligned} \tag{4.33}$$

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

   Prior to resampling, the weights of particles should be normalized such that they sum up to one. Then, compute the cumulative probability density function (*cdf*) which defines intervals that reflect each particle "share" of the total probability, as illustrated Figure 4.8. Draw p times one particle by generating p independent uniformly distributed random numbers r in the interval [0,1]; obviously the "heaviest" particles are more likely to be drawn.

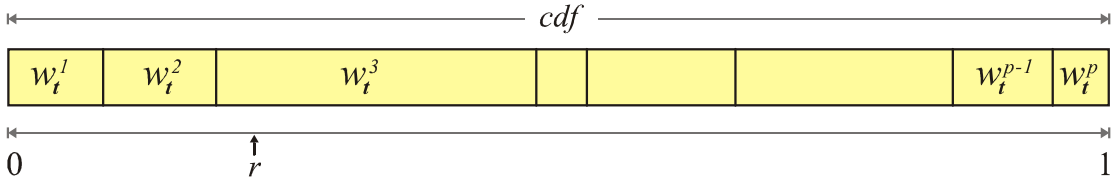

Figure 4.8: Cumulative probability distribution and a random number r.

The output of resampling step is a new set Φ_t that possesses many duplicates, since particles are drawn with replacement. This new set is distributed according to the desired posterior $p(x_t)$.

5. If the measurement vector z_t, introduces a new landmark never seen before, this needs to be included into the set of particles, as follows.

The new observed landmark in z_t consist of its distance r_q and its angle ϕ_q referenced to the unknown true robot position, as shown in Figure 4.9 and eq. (4.34).

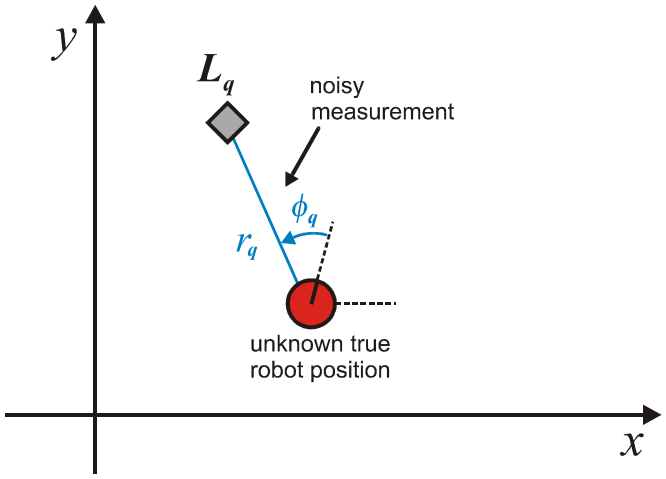

Figure 4.9: New observed landmark $\boldsymbol{L_q}$.

$$L_q = \begin{bmatrix} r_q \\ \phi_q \end{bmatrix} \tag{4.34}$$

The new landmark L_q must be included within each particle in the particle set through its mean and covariance. Thus, for each particle, function h_q^i and its Jacobian H_q^i are given by:

$$h_q^i = \begin{bmatrix} x_q^i \\ y_q^i \end{bmatrix} = \begin{bmatrix} R^i x_t + r_q \cos(R^i \theta_t + \phi_q) \\ R^i y_t + r_q \sin(R^i \theta_t + \phi_q) \end{bmatrix} \tag{4.35}$$

$$H_q^i = \frac{\partial h_q^i}{\partial_{L_q}} = \begin{bmatrix} \cos(R^i \theta_t + \phi_q) & -r_n \sin(R^i \theta_t + \phi_q) \\ \sin(R^i \theta_t + \phi_q) & r_n \cos(R^i \theta_t + \phi_q) \end{bmatrix} \tag{4.36}$$

The mean is $\lambda_{q,t}^i = h_j^i$ and the covariance, $\Sigma_{q,t}^i$, is computed using the covariance matrix for the sensor noise Q:

$$\Sigma_{q,t}^{i} = H_q^i Q H_q^{i\,T} \tag{4.37}$$

Now the particle set Φ_t has a new landmark incorporated, making the number of total landmarks become n:=*n+1*.

4.3. Simulator

To test the grid mapping algorithm, it is necessary to acquire data from a LRF mounted on a robot that moves along a structured environment. It is also possible to evaluate the algorithm from simulations of both the environment and the LRF readings, as long as noise is introduced in the process.

The simulation of the robot perception based on LRF provides several advantages. First of all, it is cheaper than experimenting with a real device. The simulator gives the opportunity to concentrate more on the intelligence algorithms such as localization and mapping (SLAM). Developing the simulator is easier than building a new robot or perception system [30], due to the testability of many configurations.

The simulator used in this work is implemented on Matlab®, explaining as follows.

4.3.1. 3D Environment Simulation

The element used in the simulator to represent the structured environment is the plane. All structured environment elements are approximated using planes; straight walls, curved walls, doors, windows, floors, ceilings, etc, can be modeled using a group of rectangles. Figure 4.10 shows an environment constructed in such a way.

Each rectangle in this simulator is defined by four points, from which their equation obtained. Thus taking, three points (a,b,c) of the four given points, the equation of the plane that contains this rectangle is given by:

$$n_x(x-a_x)+n_y(y-a_y)+n_z(z-a_z)=0 \tag{4.38}$$

where

$$\begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} = (\boldsymbol{b}-\boldsymbol{a})\times(\boldsymbol{c}-\boldsymbol{a}) = \left(\begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} - \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}\right) \times \left(\begin{bmatrix} c_x \\ c_y \\ c_z \end{bmatrix} - \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}\right) \tag{4.39}$$

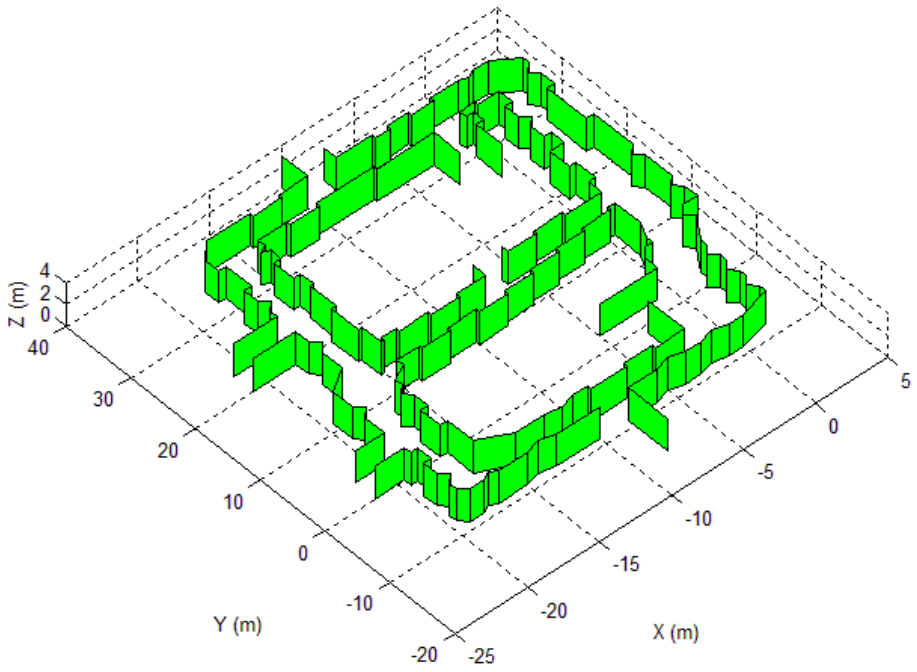

Figure 4.10: Simulated structured environment using rectangles.

4.3.2. LRF Simulation

The working principle of the LRF, also known as a LIDAR, is shown in Figure 4.11. In the presented simulation, laser ray data are assumed 1° apart, from

-90° to 90°. Note that these values are adjustable parameters in the simulator, which can vary depending on the simulated LRF model.

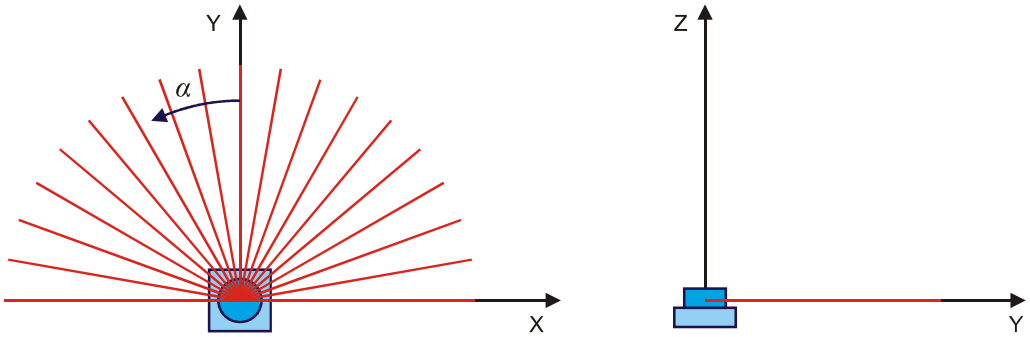

Figure 4.11: Laser ray from a simulated LRF.

Other adjustable parameters from the simulator are shown in Table 4.3.

Table 4.3: Adjustable parameters on simulated LRF.

Parameter	Units
Field of view	± rad
Operating range	m
Angular resolution	rad
Statistical Error	± mm, ±%, stdv (mm)

To acquire 3D data, the simulated LRF can be routed along its axis *X*, from 0° to 90° as shown in Figure 4.12, emulating a rotational support for the 2D LRF.

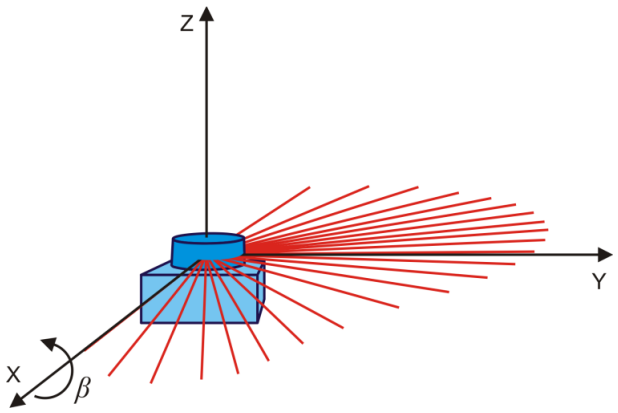

Figure 4.12: Simulated LRF rotation to acquire 3D data

Each measured laser ray is modeled as a vector in the polar system (r, α, β), with the origin as the point where the rays come from, where r represents the measured range, the rotation in the Z axis is given by α, and the rotation in X is given by β.

Simulated measurements are obtained by merging the simulated LRF routines with the models of the simulated environment. The point that defines the position of the LRF in the environment is the same point where the rays come from. Thus, the position and rotation of the LRF (in the environment and therefore in a global coordinate system) is given by four variables: x, y, z and θ, where θ is the rotation on an axis parallel to the Z axis of the global coordinate system.

With the LRF's rays and its position and rotation in the global coordinate system, the equation for each ray vector is constructed. Using the two points ($\boldsymbol{a}$ and $\boldsymbol{b}$) that define a ray vector, the line equation in 3D is given by

$$\frac{(x-a_x)}{c_x}=\frac{(y-a_y)}{c_y}=\frac{(z-a_z)}{c_z} \tag{4.40}$$

where

$$\begin{bmatrix} c_x \\ c_y \\ c_z \end{bmatrix} = (\boldsymbol{b}-\boldsymbol{a}) = \left(\begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} - \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} \right) \tag{4.41}$$

Finally, virtual data is acquired equating the line equation (of each ray vector) and the plane equation (of each plane), solving it to find the intersection points, and then computing the distances *r*, between these points and the LRF.

Figure 4.13 and Figure 4.14 show a scan where the position of the virtual LRF sensor is $x = 2.0$, $y = 12.3$, $z = 0.4$ and $\theta = 30°$ while the angle α ranges from -90° to 90° (181 rays), with a constant angle $\beta = 10°$.

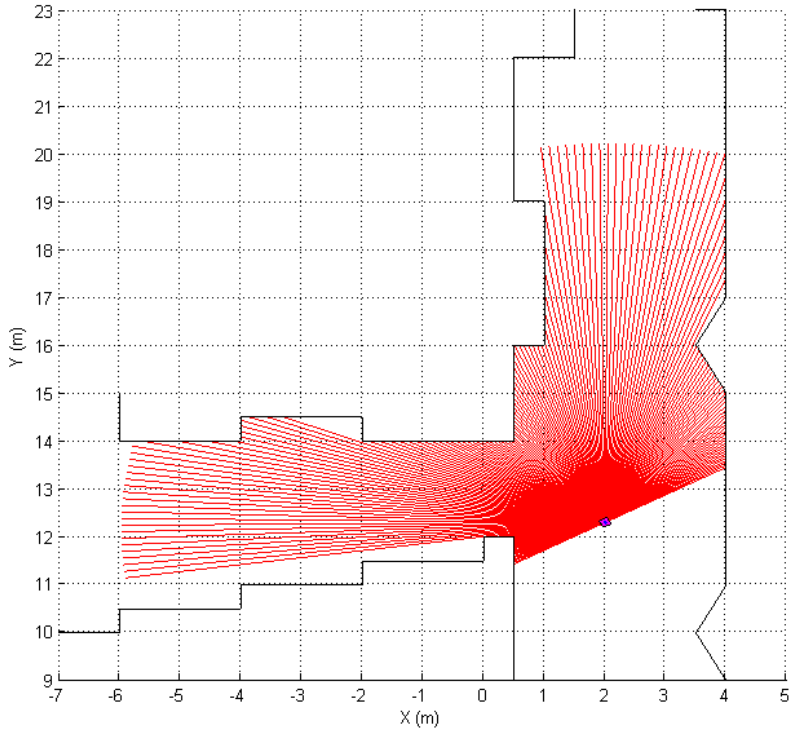

Figure 4.13: Virtual LRF readings in a simulated environment (top view).

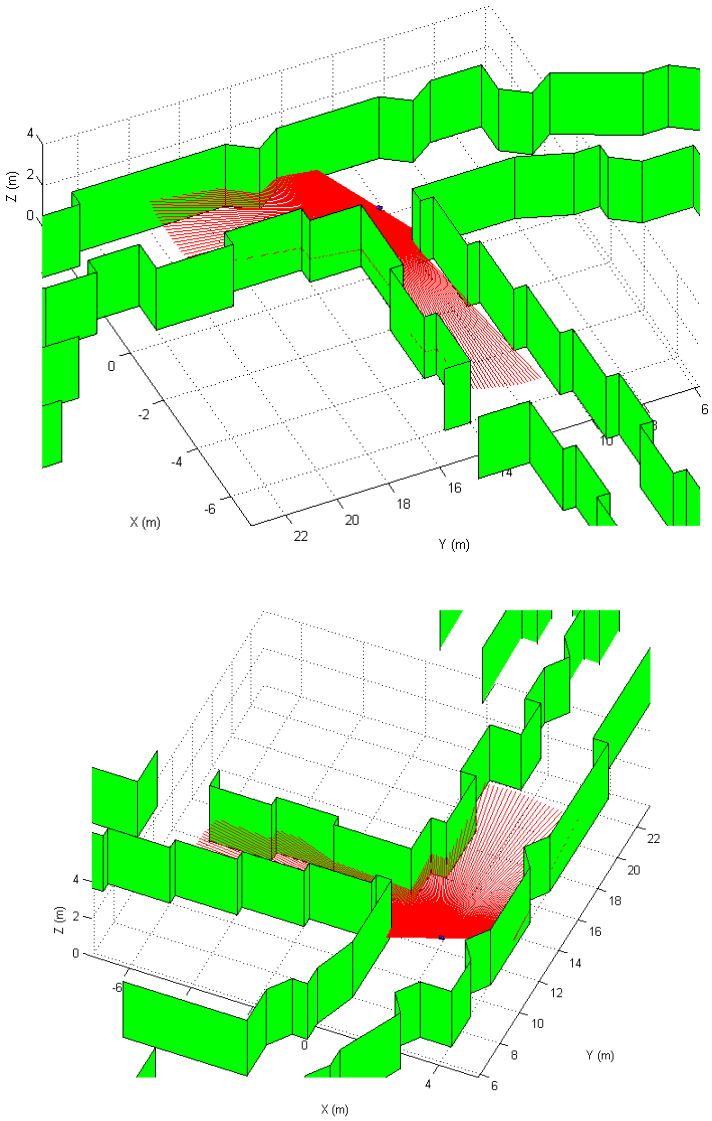

Figure 4.14: Virtual LRF readings in a simulated environment, from two different points of view.

4.3.3. Error introduction in virtual data

The LRF suffers from two types of errors: systematic and statistical. Systematic error can be reduced to acceptable limits by a good calibration or replacing the faulty sensor. However statistical errors are always present in the measurements. Thus for LRFs, each manufacturer gives in general the following three ways to represent the statistical error:

- **By standard deviation**. The standard deviations (stdv) of the measurements are given. For example the LRF SICK, model LMS200, has a stdv of 5mm, and the model LMS291 has a stdv of 10mm.

- **By a percentage of the measurement or by a fixed number**. For instance, the LRF SICK, model LMS291-S05 has an error of +/- 10mm.

- **By a fixed number until a certain range, and beyond that by a percentage**. For instance, the LRF HOKUYO, model URG-04LX, has an error of +/- 10mm in the range from 0 to 1m and, beyond that, 1% of the measurement. The model URG-04LX-UG01 has an error of +/- 30mm in the range from 0 to 1m and beyond that, 3% of the measurement.

These ways of expressing the error can be simulated and introduced in the distances *r,* virtually measured from the presented simulator.

The above presented simulator description just focused in its basic functionalities, however there are innumerable improvements that can be made, especially regarding the performance in time response. Solving equations for many planes/rectangles and lines is computationally expensive. Reference [31] a simulated 3D laser measurement system is presented, based on LMS SICK 200 mounted on a rotational platform. This simulator uses multi core processing power to generate 3D data. In [32] a simulator for airborne altimetric LIDAR is presented. This simulator is conceived having three components: terrain component, sensor component and platform component.

4.4. Scan Matching

One of the objectives of this work is to map an environment without using odometry information. Thus, the unique available information is taken from the scans made by the Laser Range Finder (LRF). To estimate the robot movement, the alignment of two consecutive scans needs to be performed.

It is important to notice that the scan matching searches for the displacement between two consecutive scans, but this displacement is not necessarily the robot

displacement, since LRF may be mounted away from the robot center. To calculate the robot movement it is necessary to know where the LRF is situated on the robot, specifically where the LRF is located with respect to the coordinate system of the robot**.**

As described in Section 2.3.3, the scan matching method used in this work is the Normal Distribution Transform (NDT). This method was selected basically because it has a featureless representation of the environment, consequently it doesn't need search for correspondences in the scans.

NDT uses Newton´s algorithm to minimize the function $\boldsymbol{f} = -score$. In most systems, the initial estimate of the solution is given by odometry information. But large error in the initial position estimate can contribute to the non-convergence of the algorithm.

Therefore, to generate a robust algorithm, in this work a Genetic Algorithm (GA) is used for optimization, from the Differential Evolution routines described in Section 2.4.5.

4.4.1. Differential Evolution Optimization for NDT

First, let's rewrite eq. (2.15)

$$score(\boldsymbol{p}) = \sum_i \exp\left(-\frac{(x_i' - q_i)^{\mathrm{T}} \Sigma_i^{-1} (x_i' - q_i)}{2}\right) \tag{4.42}$$

The outline of this optimization, given two scans (the *first* and the *second* one), is as follows:

1. Build the NDT of the *first* scan, as described in Section 2.3.3.
2. Use the *first* scan to evaluate the distribution of its points using eq. (4.42). This means that the vector of parameters must be $\boldsymbol{p} = [0\ 0\ 0]^T$ and the $score(\boldsymbol{p})$ will give a value that is the maximum possible. This is,

because the same scan is used to build the NDT and to evaluate the score. Let's call this maximum value $\boldsymbol{A}$.

3. Filter the *second* scan, in order to eliminate regions with high density readings (this will be explained in Section 4.4.3).

4. Initialize the ED algorithm, giving it the filtered *second* scan and the function $\boldsymbol{A}-score(\boldsymbol{p})$ as the fitness function. That is, ED should minimize this function, estimating the vector $\boldsymbol{p}$ and evaluating the filtered *second* scan into the NDT of the *first* scan.

The use of ED is robust and useful to estimate the vector $\boldsymbol{p}$, which in this 2D case is composed by the translation (Δx, Δy) and rotation ($\Delta\theta$) between scans. This tree displacement values are the variables to be codified; thus, one chromosome is composed as:

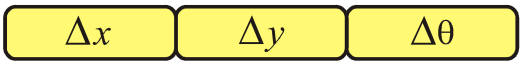

Figure 4.15: Chromosome composition

and are represented by real numbers. The fitness function, as exposed above, is given by:

$$Fitness = \boldsymbol{A} - \sum_i \exp\left(-\frac{(\boldsymbol{x}_i' - \boldsymbol{q}_i)^T \Sigma_i^{-1} (\boldsymbol{x}_i' - \boldsymbol{q}_i)}{2}\right) \tag{4.43}$$

- **The Stopping Criterion**

The stopping criterion used in this optimization is given by two values. The first is the Fitness value; thus, the optimizations stops if the fitness values is less than 1.0 (this value was acquired empirically). The second stopping criterion is the number of generations, established in 50 generations; although this criterion value is varied for analysis purposes in Section 5.1.1.

- **The Search Space**

Looking up in real data, references [8] and [33], most robot displacements are in its face direction; lateral movements are minimal because they are due to sliding. Another behavior in robots movements is that backward displacements are few and short. Thus, the search space for the three variables in the estimated vector p is:

Table 4.4: Search space for vector $\boldsymbol{p}$

Variable	Min value	Max value
Δx	-0.25 m	0.25 m
Δy	-0.50 m	1.00 m
$\Delta\theta$	-20.00 °	20.00 °

In this way, the translation and rotation between two scans is limited by this search space. These values were empirically defined such that they guarantee enough similar information in two consecutive scans in order to mach them.

This search space and the LRF's scanning frequency (which will be discussed later) therefore might limit the maximum lineal and angular velocity of the robot. However, the LRF's scans are sufficiently fast for most achievable robot speeds for non-airborne systems. Therefore, the search space plays a minor role in limiting the robot speed.

- **DE Parameters**

The following parameters used in DE for scan matching optimization were empirically acquired.

Table 4.5: DE Optimization Parameters.

Parameter	Value
Population size (*NP*)	100
Number of generations	50
Scaling Factor (*F*)	1.00
Crossover constant (*CR*)	0.95

Some consideration should be taken into account before using this ED optimization, as explained next.

4.4.2. Parameters and Considerations

a. Environment considerations

The environment to be mapped needs variety. E.g. long corridors without doors or any wall-shape variations will lead to poor scan matching performance, the ED will result in an estimated vector $\boldsymbol{p} = [0\ 0\ 0]$. How long these corridors can be without compromising scan matching depends on the LRF's maximum range, e.g., if the LRF's range is 8m, then the length of such corridor should be less than this value. But note that there are LRFs such as SICKs that have more than 50 meters in range.

In the same way, the width of these corridors should be less than LRF's range, to ensure that the robot will pickup data from both sides (left and right) of the sensor position.

b. LRF Considerations

Perhaps the most known LRF in Robotic is the SICK. SICK has a set of LRFs for many applications, and the most popular being the family of

LMS-200. Thus, for example, the SICK LMS-200-3016 model has the following main features [34]:

Table 4.6: SICK LMS-200-3016 features.

Field of view	180 °	
Scanning Frequency	75 Hz	
Operating range	0 m … 80 m	
Angular resolution	0.25°, 0.5°, 1.0°	
Systematic Error	+/- 15mm	
Statistical Error	+/- 5mm	

The scanning frequency of the above SICK is 75Hz, meaning 13.33 ms per scan. Thus with such data, it is possible to calculate the maximum linear and angular velocity of the LRF in order to guarantee that the movement of the robot does not affect the LRF's readings.

The following simplified equations could help to estimate these maximum speed as a function of the LRF's scanning frequency.

$$v_{\max} = \varepsilon_t * f_s \tag{4.44}$$

$$\omega_{\max} = \varepsilon_r * f_s \tag{4.45}$$

where f_s is the scanning frequency of the LRF, and the parameters ε_t and ε_r are the maximum error introduced by the robot's movement into the scan readings. For example, if the desired error must be at most 5mm in translation, and the scanning frequency of the LRF is 75 Hz, then the maximum speed of the LRF is $\boldsymbol{v}_{max} = 0.375$ m/s, while for a desired error of up to 0.25° in rotation, then $\boldsymbol{w}_{max} = 0.327$ rad/s.

Note that this maximum values in translation are referred to LRF speeds, which are not necessarily the same as the robot speed,

depending on where the LRF is mounted on the robot and if the robot is making a turn.

4.4.3. Scan Filtering

The value of eq. (4.42) is influenced by the density of the readings. Regions with higher reading density are produced when the robot is close to a wall, thus eq. (4.42) results in higher values in this regions, see Figure 4.16. This situation is not desirable, as seen in [35]. The ED optimization could give us mismatched scans, as Figure 4.17, since such oversampling in one small region could negatively affect the matching of regions further away.

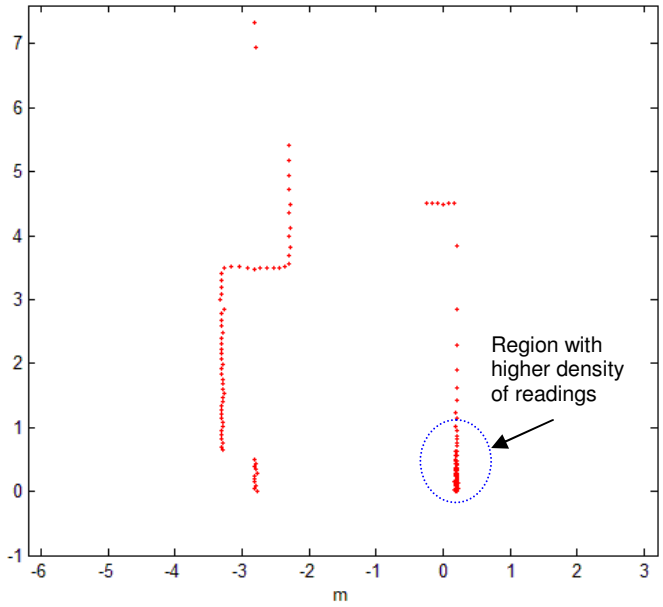

Figure 4.16: Density regions produced by a robot situated close to the right wall.

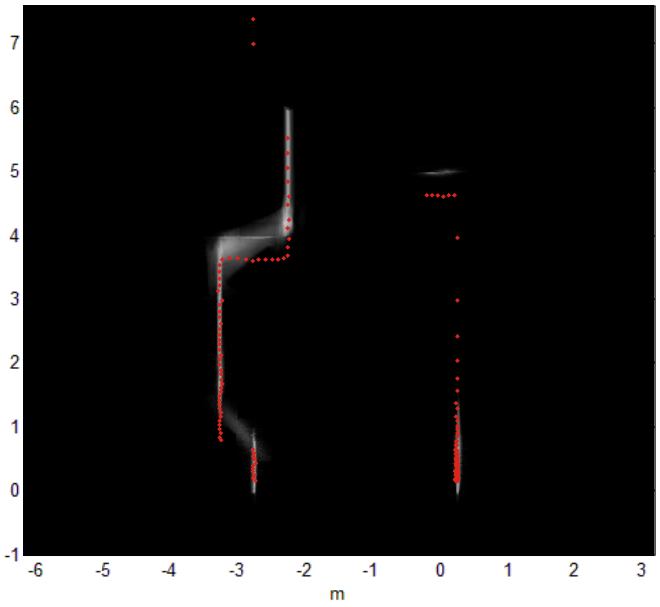

Figure 4.17: Mismatched scans, showing the NDT of a *first scan* (grayscale) and a *second scan* (red dots). The right-bottom wall produces a high number of readings, bringing down the second scan and compromising the match.

To overcome this situation this work uses the approach used by [35], replacing small clouds of close points by their center of gravity. This has the effect of smoothing the distribution of points over the scan. It also greatly reduces the number of scan points, without loosing too much information.

The idea behind this filter is to move a circular window with fixed radius over the scan and to replace the readings inside the window with their center of gravity. The radius of the window defines the minimum distance between the points in the filtered scan. This radius has to be defined experimentally. Low values for this parameter do not solve the influence of the readings density, while high values may render the resulting scan too sparse. In this work, this parameter is set to 10 cm.

Figure 4.18 shows the same scan points of Figure 4.16, before filtering (left) and after filtering (right). Reducing the number of scan points also improves the speed of the ED optimization.

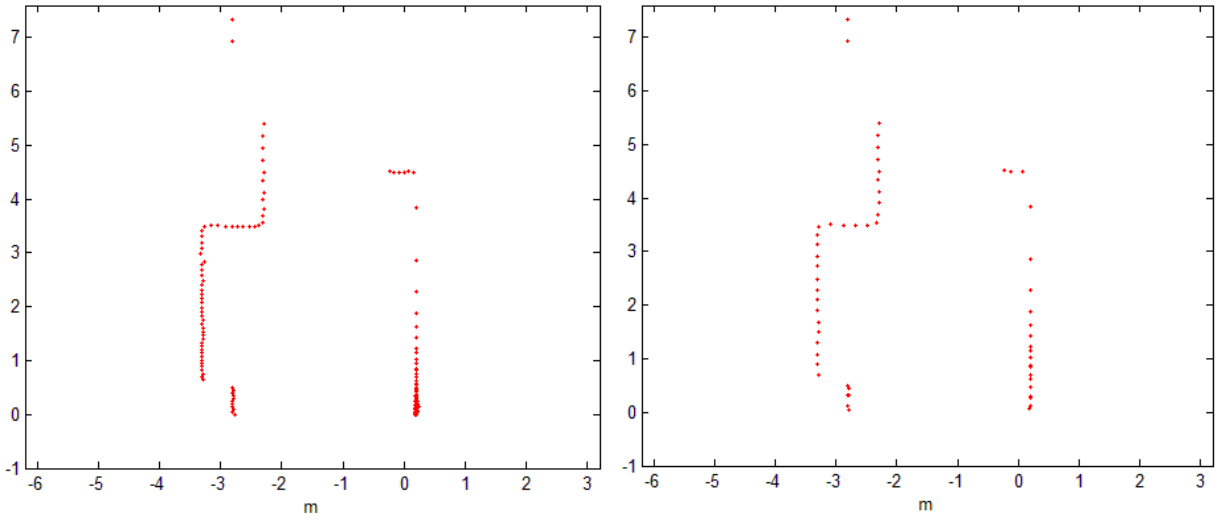

Figure 4.18: Scan filtering. Original scan with 171 points (left) and filtered scan with 59 points (right).

4.5. DP-SLAM

The DP-SLAM algorithm is presented in this work in the form of flowcharts. The detailed implementation of DP-SLAM is too extensive to be presented here. However the code used in this work is free, available to download from [8]. This code was modified in this work to include scan matching as a motion model, as explained later.

As explained in Section 3.2.3, DP-SLAM uses a hierarchical algorithm. The relationships between the low and high levels are shown in Figure 4.19. Here, the input data is composed by the odometry and the range scans. Thus, each position, estimated by the odometry reading, has attached its range scan. A piece of data is used as input to low level SLAM. The output is the best estimated trajectory attached with its corresponding range scans.

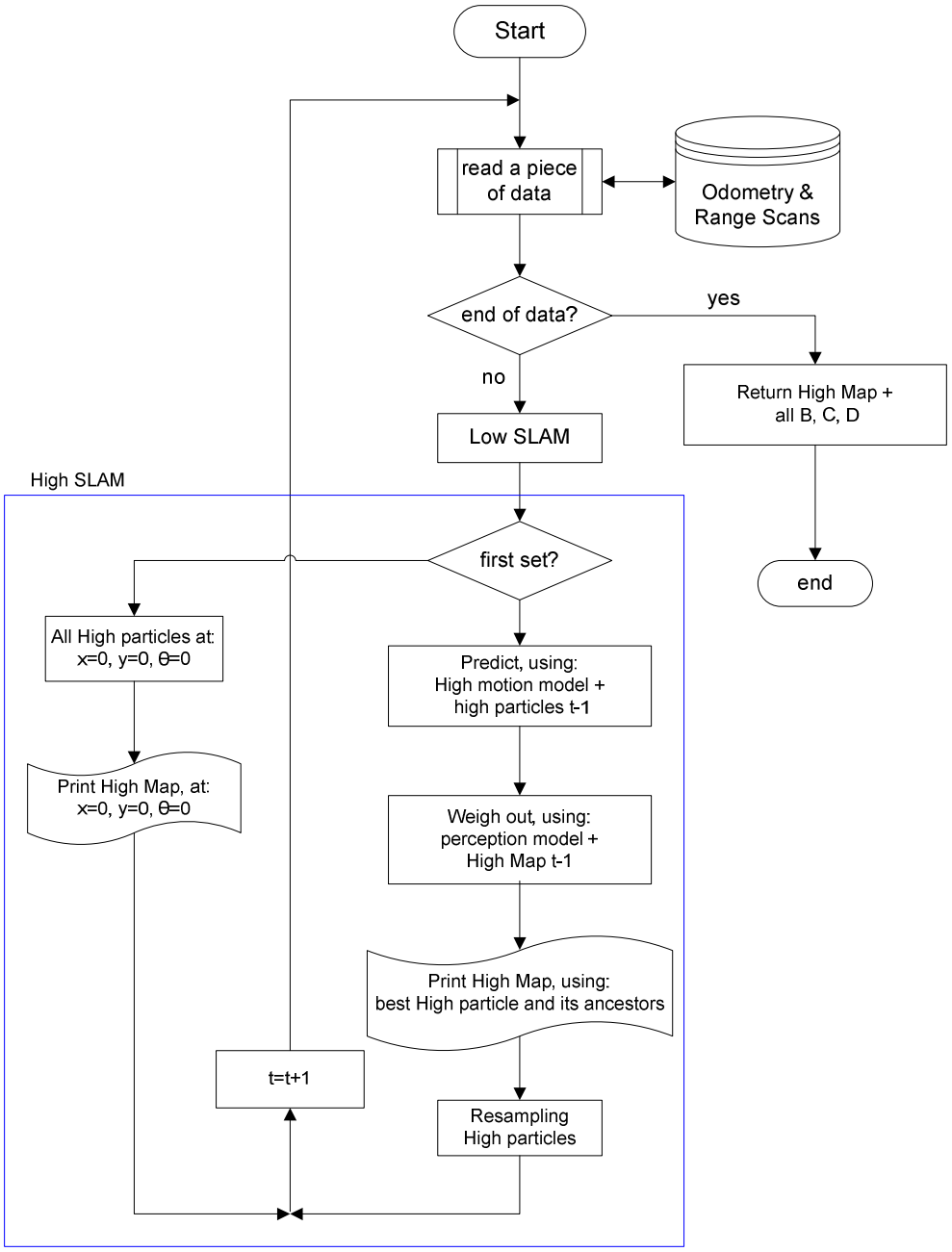

Figure 4.19: DP-SLAM flow chart

Figure 4.20 shows the low SLAM flowchart. Notice that, in the beginning, all los particles are located in an initial position, and using the firs data scan a low-map is printed. This initial position and low-map is used by the subsequent *t+1* iteration.

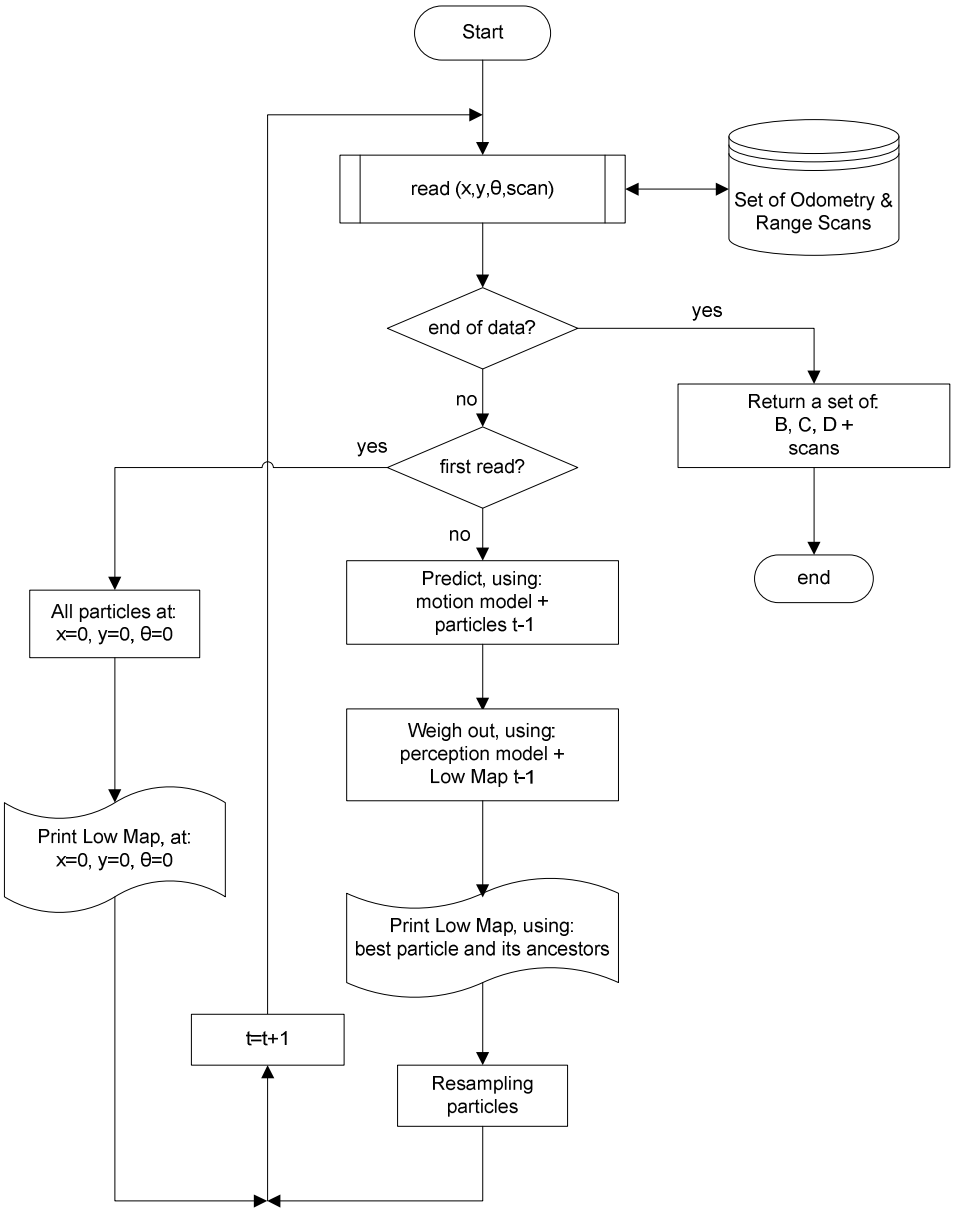

Figure 4.20: Low (level) DP-SLAM flow chart

As show in Section 3.2.3, DP-SALAM uses a particle filter to track the robot position. Thus the motion model is used to predict (i.e. generate samples) and the perception model is used for particles weighing. A low-map is printed each iteration, using the best particle and its ancestors. Note that the best particle at iteration *t*, may not be a child of the best particle at iteration *t−1*.

The output of the Low SLAM (LSLAM) is used by the High SLAM (HSLAM), as shown in the Figure 4.19. Note that HSLAM is slightly similar to LSLAM, but instead of sampling robot positions, HSLAM samples robot

trajectories. Also both low an high, use the same perception model, but their laser variances are different. The LSLAM is the basic SLAM algorithm, working unperturbed while the HSLAM is working with slightly different data and as such, requires a different laser noise model. With a "rigid" trajectory passed up from the LSLAM, there is less room for minor perturbations, and certain amount of assumed drift. All of this is included in the high level laser variance, which needs to be correspondingly larger [6]. Empirical results show that using a standard deviation of 7cm at the higher level works well [6].

Finally the High SLAM output is the best High Map and robot path.

4.5.1. Motion Model

The motion model proposed by Eliazar [6], is shown in eq. (4.46).

$$\begin{bmatrix} R^i x_t \\ R^i y_t \\ R^i \theta_t \end{bmatrix} = \begin{bmatrix} R^i x_{t-1} + D\cos(R^i \theta_{t-1} + B/2) + C\cos(R^i \theta_{t-1} + (B+\pi)/2) \\ R^i y_{t-1} + D\sin(R^i \theta_{t-1} + B/2) + C\sin(R^i \theta_{t-1} + (B+\pi)/2) \\ R^i \theta_{t-1} + B \end{bmatrix} \quad (4.46)$$

The new position $R_t=[Rx_t\ ,\ Ry_t\ ,\ R\theta_t]$ depends on the last robot position R_{t-1} and the parameters B, C and D. The value $(R\theta_{t-1} + B/2)$ is called the major axis movement, and both B and D are expected to be distributed normally distributed with respect to the reported odometry values b and d (amount of rotational and translational movement, respectively). But the mean of each B and D will scale linearly with both b and d, while the variance will scale with b^2 and d^2. C is an extra lateral translation term, which is present to model shift in the orthogonal direction of the major axis. This axis, called minor axis, is at angle $(R\theta_{t-1} + (B+\pi)/2)$. In this view, B, C and D are all conditionally Gaussian given b and d:

$$B \sim \mathcal{N}(d\mu_{Bd} + b\mu_{Bb}, d^2\sigma^2_{Bd} + b^2\sigma^2_{Bb})$$
$$C \sim \mathcal{N}(d\mu_{Cd} + b\mu_{Cb}, d^2\sigma^2_{Cd} + b^2\sigma^2_{Cb})$$
$$D \sim \mathcal{N}(d\mu_{Dd} + b\mu_{Db}, d^2\sigma^2_{Dd} + b^2\sigma^2_{Db})$$

where μ_{Ax} is the coefficient for the contribution of the odometry term x to the mean of the distribution over A. DP-SLAM uses an automatic parameter estimator to obtain these μ_{Ax}.

Note that the above implementations assume that odometry readings are available. This work, on the other hand, does not make use of odometry information. This is obtained instead from scan matching. In this way, the "scan odometry" consist of tree displacements: Δx, Δy, $\Delta\theta$, displacements that are referenced to the current robot position. Because scan matching is used, no extra lateral displacement needs to be considered. In the implemented approach, two consecutive scans are taken from two different points in the environment, as shown in Figure 4.21.

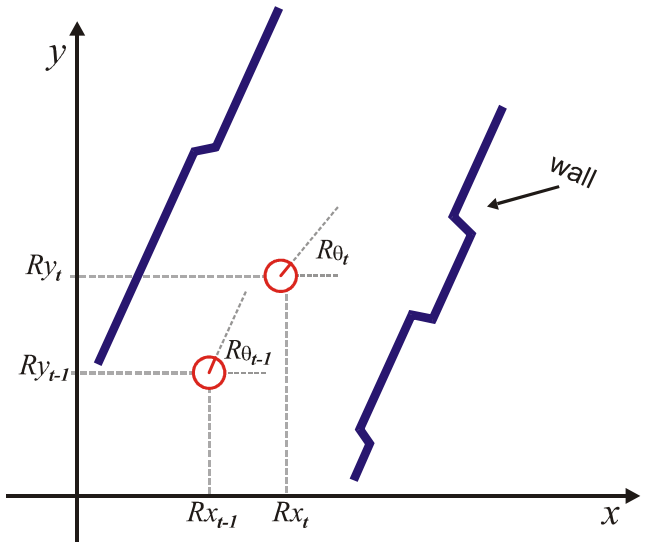

Figure 4.21: Two consecutive robot positions in an environment.

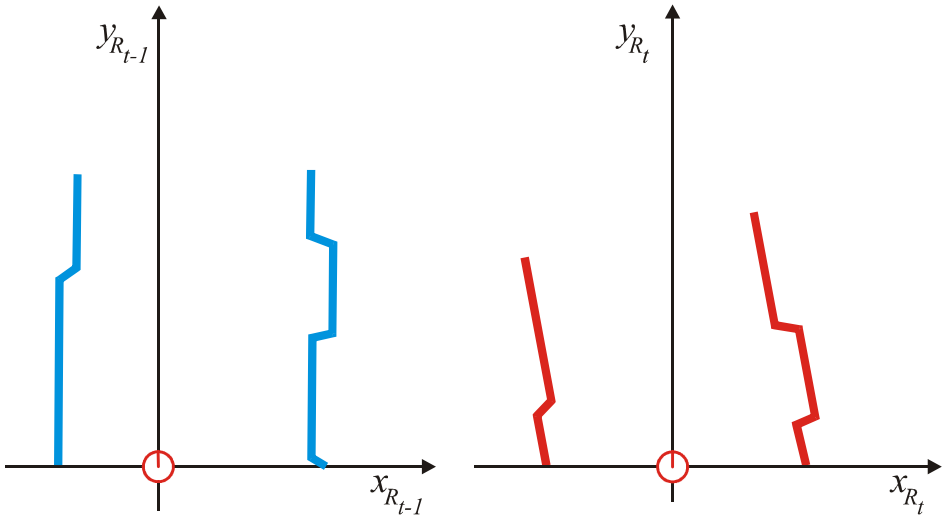

Figure 4.22: Environment seen from the current robot position (left) and second robot position (right).

Thus, the scan matching process searches for an alignment of both scans, by rotating and translating the second scan onto the first scan coordinate system, to obtain the actual robot displacements Δx, Δy, $\Delta\theta$. The aligned scans are shown in Figure 4.23.

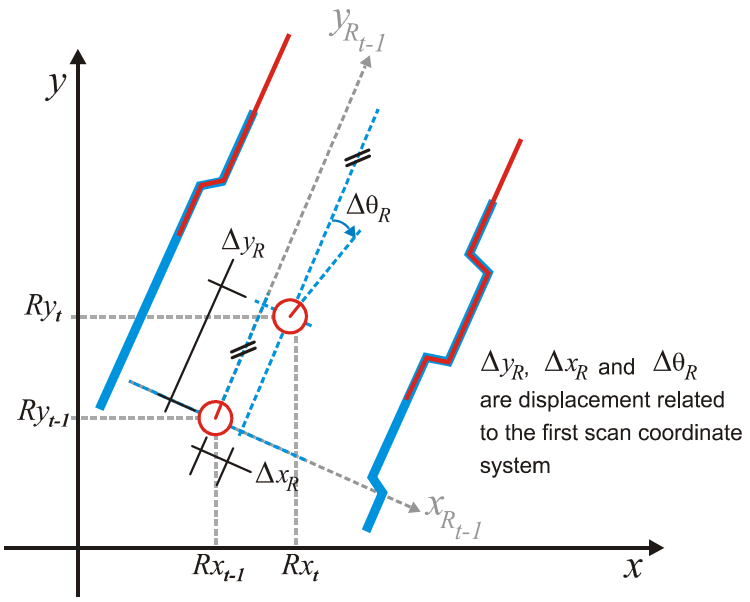

Figure 4.23: Aligned scans in a global coordinate system, displacements Δy_R, Δx_R and $\Delta\theta_R$ are related to the first scan coordinate system.

Assuming a perfect scan matching, eq. (4.47) gives the new robot position by

$$\begin{bmatrix} R^i x_t \\ R^i y_t \\ R^i \theta_t \end{bmatrix} = \begin{bmatrix} R^i x_{t-1} + d\cos(\alpha) \\ R^i y_{t-1} + d\sin(\alpha) \\ R^i \theta_{t-1} + \Delta\theta \end{bmatrix} \tag{4.47}$$

where:

$$d = \sqrt{(\Delta x)^2 + (\Delta y)^2} \quad \text{and} \quad \alpha = R^i\theta_{t-1} - \pi/2 + \text{atan2}(\Delta y, \Delta x)$$

However, because the scan matching process isn't perfect, it will give us an approximation of Δx, Δy, and $\Delta\theta$. It is expected that these approximations are distributed according to a distribution shape. The shape of this distribution must be acquired empirically by comparing the approximated displacement, after convergence of the scan matching, with actual position (estimated or simulated), as explained in Chapter 5. Table 4.7 shows an algorithm to sample from this scan matching motion model, $(R_t \mid u_t, R_{t-1})$, to generate a random poses R_t^i. Lines 1 through 3 perturb the "scan odometry" parameters with noise, drawn from the error parameters v_x, v_y and v_θ (they will be explained in Chapter 5). The noise values are then used to generate the new sample pose in lines 4 through 8. of the algorithm.

Table 4.7: Sample scan matching motion model algorithm, where atan2(Δy, Δx) is defined as the generalization of the arc-tangent function of $\Delta y/\Delta x$ over [0, 2π].

Sample SM Motion Model algorithm (R_{t-1}, u_t)	
1:	$\Delta\hat{x} = \Delta x + sample(v_x)$
2:	$\Delta\hat{y} = \Delta y + sample(v_y)$
3:	$\Delta\hat{\theta} = \Delta\theta + sample(v_\theta)$
4:	$d = \sqrt{(\Delta\hat{x})^2 + (\Delta\hat{y})^2}$
5:	$\alpha = R\theta_{t-1} - \pi/2 + \text{atan2}(\Delta\hat{y}, \Delta\hat{x})$
6:	$Rx_t = Rx_{t-1} + d\cos(\alpha)$
7:	$Ry_t = Ry_{t-1} + d\sin(\alpha)$
8:	$R\theta_t = R\theta_{t-1} + \Delta\hat{\theta}$
9:	*return* $R_t = (Rx_t, Ry_t, R\theta_t)$

Finally, the LSLAM output in Figure 4.20, using the proposed motion model, is a set of changes between positions $\Delta\hat{x}$, $\Delta\hat{y}$, $\Delta\hat{z}$ and its corresponding range scans.

4.5.2. High Motion Model

As seen in Section 3.2.3.4, DP-SLAM states that the effects of drift on low level maps can be accurately approximated by perturbations to the endpoints of the robot trajectory used to construct a low level map.

By sampling drift only at endpoints, it will fail to sample some of the internal structure that is possible in drifts, e.g., it will fail to distinguish between a linear drift and a spiral drift pattern with the same endpoints. However, the existence of significant, complicated drift patterns within a map segment would violate the assumption of moderate accuracy and local consistency within the low level mapper [6].

The "motion" model in high SLAM is assumed to be Gaussian, and evenly distributed about the lateral axes. The specific values for these variances are highly mutable, affected by the specific SLAM algorithm used at the low level, and the amount of resources used, as well as elements from the robot or the environment [6].

Figure 4.24 shows how the high motion model works. As shown in Figure 4.19 the first set of data received from the low SLAM is simply printed in the high map at zero position. This is shown in the Figure 4.24, the first set of data is a set of variables $\Delta\hat{x}, \Delta\hat{y}, \Delta\hat{z}$, (represented by a red line) along with its scans (represented by the green region of the figure).

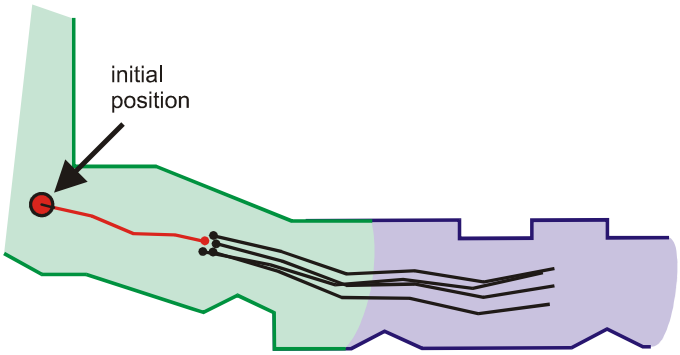

Figure 4.24: High Motion Model

The second set of data received from the low SLAM (black line) is not connected with the endpoint of the first set (red dot). Instead, it is linked with many samples (black dots), generated by applying the high motion model at the first endpoint (red dot). Notice, however, that Figure 4.24 shows the second set of scans (piece of map in blue color) only for one sample (the best sample); thus it is understood that high SLAM keeps a different map for each sample. Note that the motion model generates samples not only disturbing the endpoint position, but also the endpoint rotation.

In the next chapter the presented algorithms are evaluated, both using simulated data and real experimental data taken from the literature.