

Probabilistic Robotics Course

Particle Localization

G. Grisetti, B. Della Corte

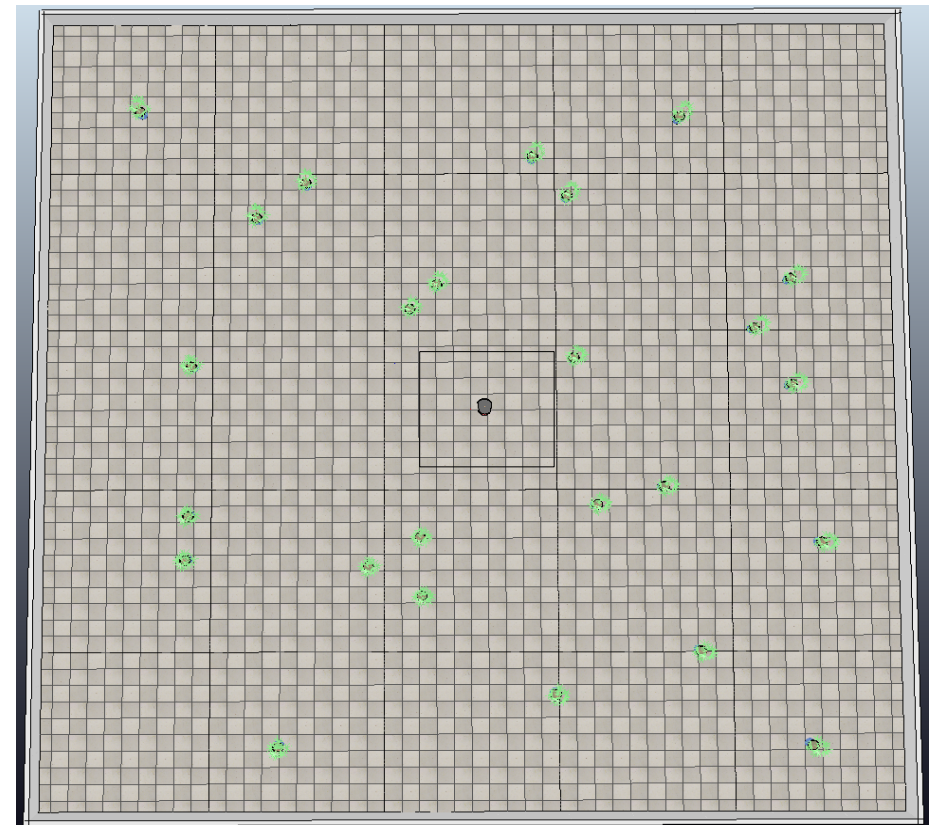
`{grisetti,dellacorte}@dis.uniroma1.it`

Dept of Computer Control and Management Engineering
Sapienza University of Rome

Scenario

Orazio moves on a 2D plane

- Is controlled by translational and rotational velocities
- Senses a set of uniquely distinguishable landmarks through a “2D landmark sensors”
- The location of the landmarks in the world is known



Approaching the problem

We want to develop a Particle Filter based algorithm to track the position of Orazio as it moves

The inputs of our algorithms will be

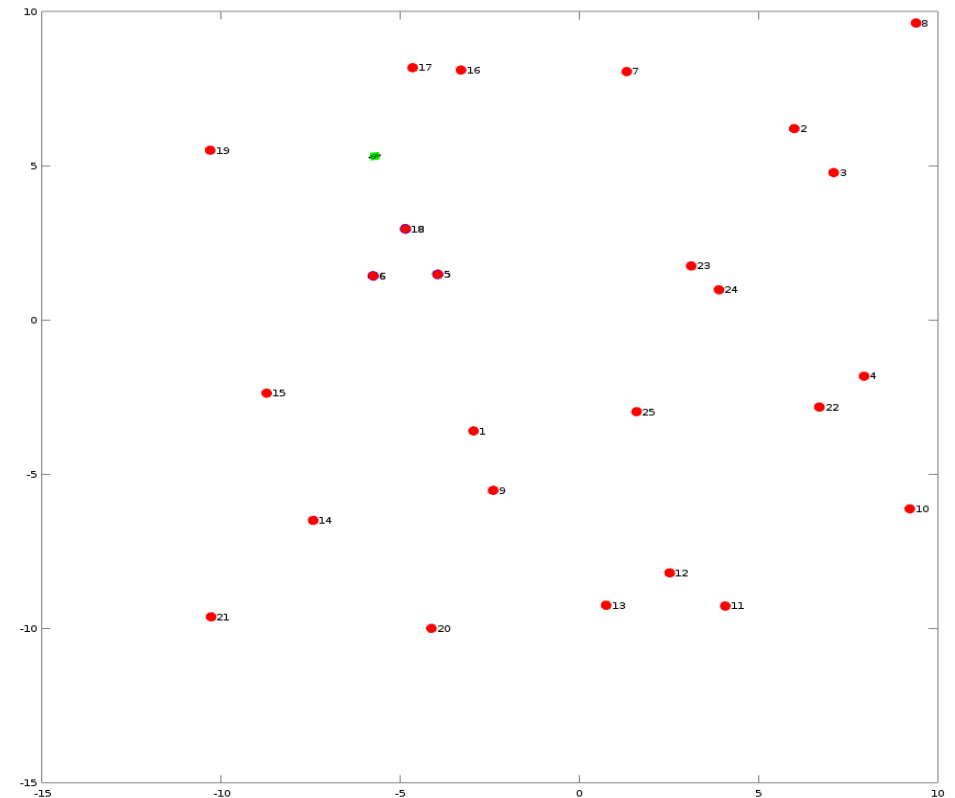
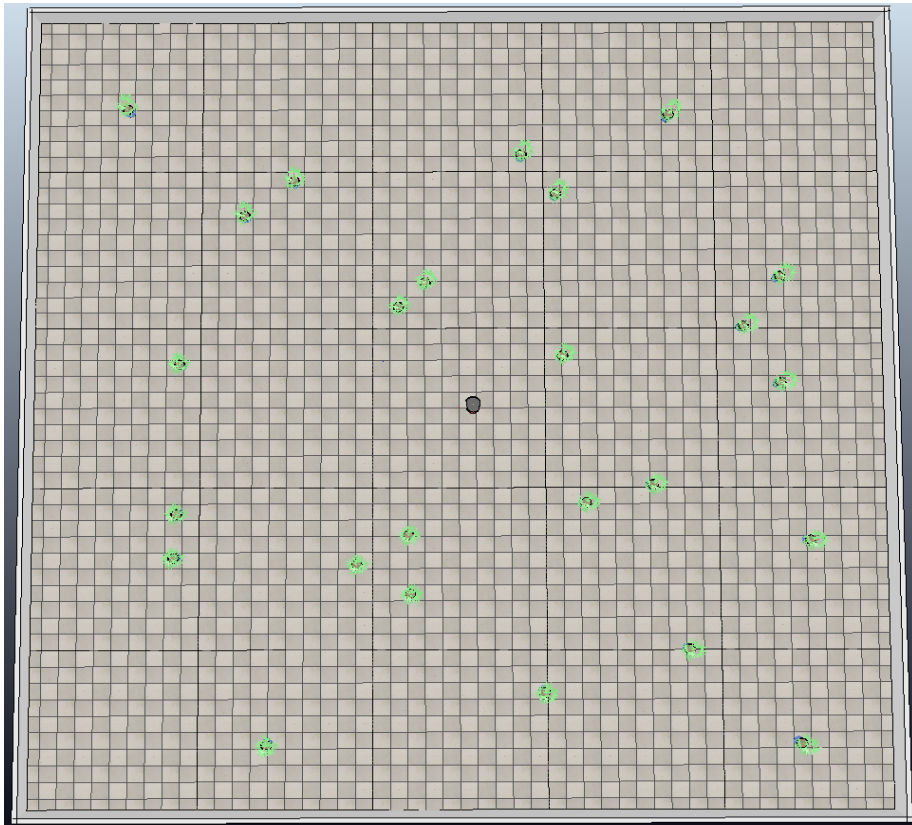
- velocity measurements
- landmark measurements

The prior knowledge about the map is represented by the location of each landmark in the world

Prior

The map is represented as a set of landmark coordinates

$$\mathbf{l}^{[i]} = \begin{pmatrix} x^{[i]} \\ y^{[i]} \end{pmatrix} \in \mathbb{R}^2$$



Domains

Define

- state space

$$\mathbf{X}_t = [\mathbf{R}_t | \mathbf{t}_t] \in SE(2)$$

- space of controls (inputs)

$$\mathbf{u}_t = \begin{pmatrix} \Delta_t v_t \\ \Delta_t \omega_t \end{pmatrix} = \begin{pmatrix} u_t^1 \\ u_t^2 \end{pmatrix} \in \mathbb{R}^2$$

Instead of considering rotational and translational velocities, we consider the integrated motion in the interval as input

- space of observations (measurements)

$$\mathbf{z}_t^{[i]} = \begin{pmatrix} x_t^{[i]} \\ y_t^{[i]} \end{pmatrix} \in \mathbb{R}^2$$

Domains

Find an Euclidean parameterization of non-Euclidean spaces

$$\mathbf{X}_t = [\mathbf{R}_t | \mathbf{t}_t] \in SE(2) \rightarrow \mathbf{x}_t = \begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix} \in \mathbb{R}^3$$

- state space

$$\mathbf{u}_t = \begin{pmatrix} u_t^1 \\ u_t^2 \end{pmatrix} \in \mathbb{R}^2$$

poses are not Euclidean, we map them to 3D vectors

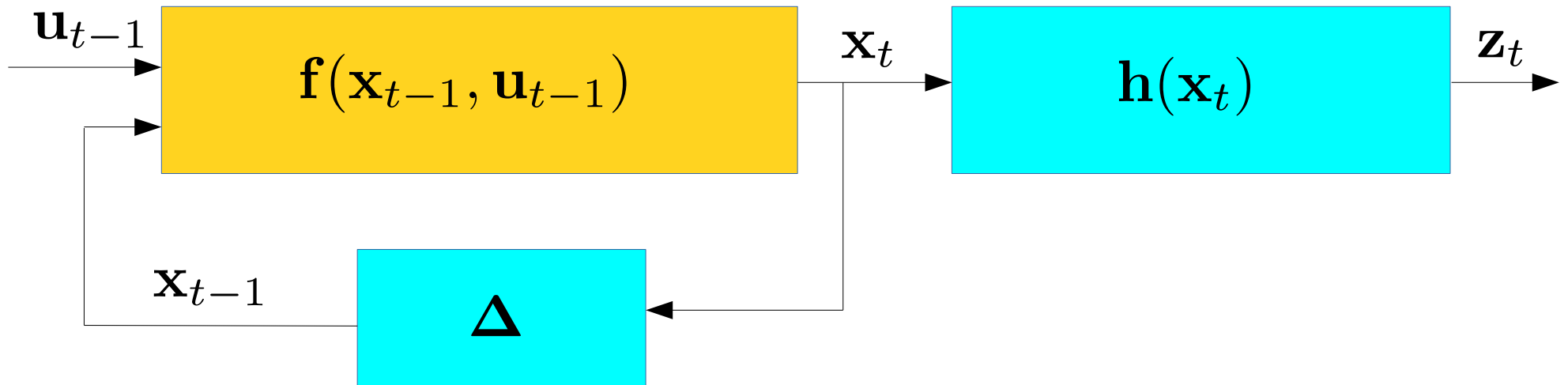
- space of controls (inputs)

$$\mathbf{z}_t = \begin{pmatrix} x_t^{[i]} \\ y_t^{[i]} \end{pmatrix} \in \mathbb{R}^2$$

measurement and control, in this problem are already Euclidean

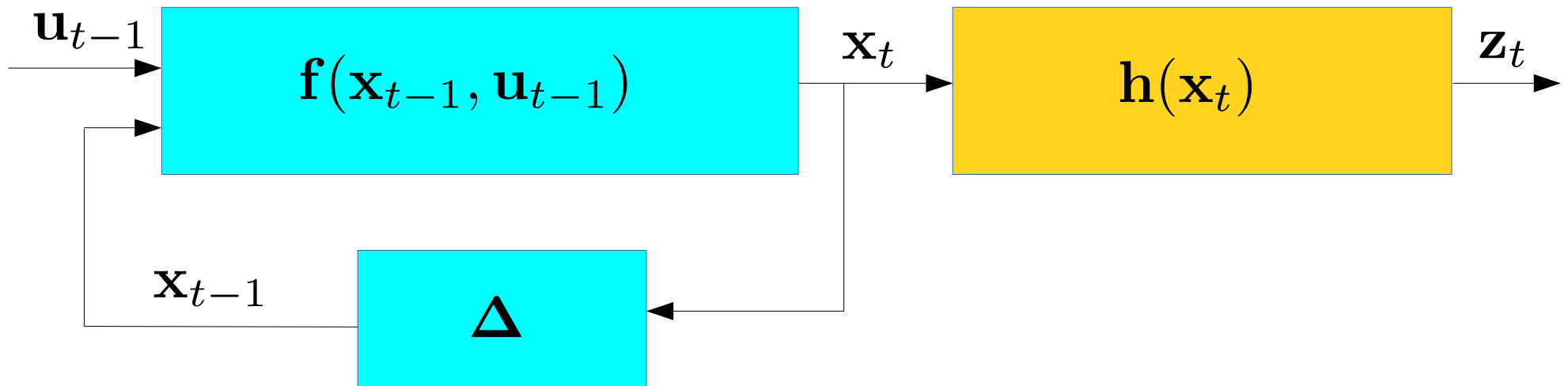
- space of observations (measurements)

Transition Function



$$\mathbf{x}_t = \mathbf{f}(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) = \begin{pmatrix} x_{t-1} + u_{t-1}^1 \cdot \cos(\theta_{t-1}) \\ y_{t-1} + u_{t-1}^1 \cdot \sin(\theta_{t-1}) \\ \theta_{t-1} + u_{t-1}^2 \end{pmatrix}$$

Measurement Function



We have [i]
measurement
functions,
one per
landmark

$\mathbf{z}_t^{[i]}$

$$\begin{aligned} &= \mathbf{h}^{[i]}(\mathbf{x}_t) \\ &= \mathbf{R}_t^T (\mathbf{l}^{[i]} - \mathbf{t}_t) \end{aligned}$$

relative position of
the i^{th} landmark
w.r.t the robot at
time t

$$= \begin{pmatrix} \cos \theta_t (x^{[i]} - x_t) + \sin \theta_t (y^{[i]} - y_t) \\ -\sin \theta_t (x^{[i]} - x_t) + \cos \theta_t (y^{[i]} - y_t) \end{pmatrix}$$

$$\mathbf{R}_t = \begin{pmatrix} \cos \theta_t & -\sin \theta_t \\ \sin \theta_t & \cos \theta_t \end{pmatrix}$$

rotation matrix of theta

Measurement Function

At each point in time, our robot will sense only a subset of K landmarks in the map

The measurement is thus consisting of a stack of measurements

$$\mathbf{z}_t = \begin{pmatrix} \mathbf{z}^{[i_1]} \\ \mathbf{z}^{[i_2]} \\ \dots \\ \mathbf{z}^{[i_K]} \end{pmatrix} = \mathbf{h}(\mathbf{x}_t) = \begin{pmatrix} \mathbf{h}^{[i_1]}(\mathbf{x}_t) \\ \mathbf{h}^{[i_2]}(\mathbf{x}_t) \\ \dots \\ \mathbf{h}^{[i_K]}(\mathbf{x}_t) \end{pmatrix}$$

index of the landmark
generating the measurement

Control Noise

We assume the velocity measurements are effected by a uniform noise resulting from the sum of two aspects

- a constant noise
- a velocity dependent term whose amplitude grows with the speed
- translational and rotational noise are assumed independent

$$\mathbf{n}_{u,t} \sim \begin{pmatrix} a_1 ||u_t^1|| U(-0.5, 0.5) \\ a_2 ||u_t^2|| U(-0.5, 0.5) \end{pmatrix}$$

Measurement Noise and Observation model

We assume it is zero mean, Gaussian and constant

$$\mathbf{n}_z \sim \mathcal{N} \left(\mathbf{n}_z; \mathbf{0}, \begin{pmatrix} \sigma_z^2 & 0 \\ 0 & \sigma_z^2 \end{pmatrix} \right)$$

With this noise the observation model becomes

$$p(\mathbf{z}_t^{[i]} \mid \mathbf{x}_t) \propto \exp \left(-(\mathbf{h}^{[i]}(\mathbf{x}_t) - \mathbf{z}_t^{[i]})^T \Sigma_z^{-1} (\mathbf{h}^{[i]}(\mathbf{x}_t) - \mathbf{z}_t^{[i]}) \right)$$

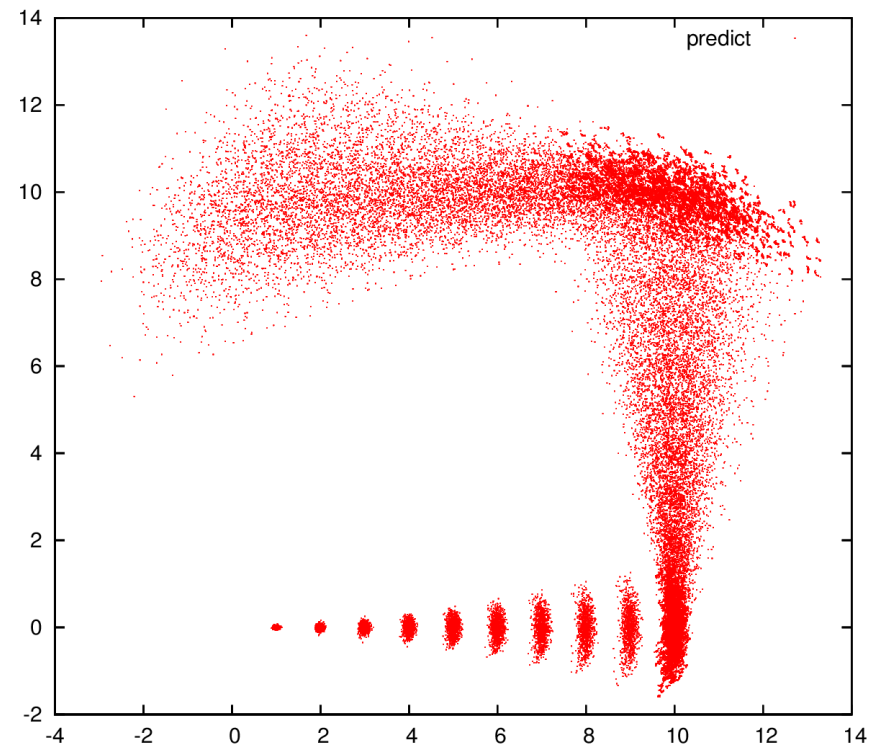
$$p(\mathbf{z}_t \mid \mathbf{x}_t) = \prod_i p(\mathbf{z}_t^{[i]} \mid \mathbf{x}_t);$$

Predict

When a new control u_{t-1} becomes available, we compute the location by

- generating an independent noise sample for each particle
- computing the location of the new sample through the transition function, evaluated at the particle, at the control and at the noise sample

$$\mathbf{n}_u^{(i)} \sim p(\mathbf{n}_u)$$
$$\mathbf{x}_{t|t-1}^{(i)} = \mathbf{f}(\mathbf{x}_{t-1|t-1}^{(i)}, \mathbf{u}_{t-1} + \mathbf{n}_u^{(i)})$$
$$w_{t|t-1}^{(i)} = w_{t-1|t-1}^{(i)}$$



evolution of the particles when the robot moves on a square

Prediction: code

```
1 function samples = prediction(samples, transition)
2
3     dim_particles = size(samples,2);
4
5     u = transition.v;
6     %it returns u = [ux, uy, utheta]. simply not consider uy
7     u_x = u(1);
8     u_theta = u(3);
9
10    a1 = abs(u_x);
11    a2 = abs(u_theta);
12    %apply transition to every particle
13    for i=1:dim_particles
14        % sample noise from uniform between
15        % [-0.5; 0.5]
16        noise_x = (rand() - 0.5)*a1;
17        noise_theta = (rand() - 0.5)*a2;
18
19        samples(:,i) = %TODO;
20    end
21 endfunction
```

Apply motion model to each particle

$$\mathbf{x}_{t|t-1}^{(i)} = \mathbf{f}(\mathbf{x}_{t-1|t-1}^{(i)}, \mathbf{u}_{t-1} + \mathbf{n}_u^{(i)})$$

Update

To compute the update we need to address the data association

We solve it in a greedy manner, for each particle

- given a particle, we evaluate the likelihood of all combinations landmarks/measurements

sample

$\mathbf{x}_t^{(i)}$

negative log likelihood of
measurement m w.r.t
landmark n seen by particle i

$$l_{mn}^{(i)} = (\mathbf{h}^{[n]}(\mathbf{x}_t^{(i)}) - \mathbf{z}^{[m]})^T \Sigma_z^{-1} (\mathbf{h}^{[n]}(\mathbf{x}_t^{(i)}) - \mathbf{z}_t^{[m]})$$

$$l_m^{(i)} = \min_m l_{mn}^{(i)}$$

negative log likelihood is the
one of the “closest” landmark

$$p(\mathbf{z}_t | \mathbf{x}_t^{(i)}) \propto \exp \left(-k^{(i)} l_{\text{miss}} - \sum_m l_m^{(i)} \right)$$

Sample likelihood

number of dropped measurements

Update(1): code

```
1 function weights = update(samples, weights, landmarks, observations
2 )
3 % init useful stuff
4 [...]
5
6 sigma_z_noise = 10;
7 Sigma_z_noise = [sigma_z_noise^2 0;
8                 0 sigma_z_noise^2];
9 Omega_z_noise = inv(Sigma_z_noise);
10
11 l_miss = 30; %this is a threshold
12
13 for i=1:num_particles
14     %init association matrix
15     l_mn = zeros(num_landmarks_seen, num_landmarks);
16     particle = samples(:,i); %current particle
17
18     for n=1:num_landmarks
19         landmark = landmarks(n); %current landmark
20         land_x = landmark.x_pose;
21         land_y = landmark.y_pose;
22         [curr_h, -] = measurement_function(particle, [land_x; land_y]);
23
24         for m=1:num_landmarks_seen
25             %current measurement
26             measurement = observations.observation(m);
27             delta = %TODO;
28             l_mn(m,n) = %TODO;
29         endfor
30     endfor
```

*We have access to
measurement.x_pose and .y_pose*

Update

Being able to evaluate the likelihood of each predicted sample we can perform the conditioning

$$w_{t|t}^{(i)} = w_{t|t-1}^{(i)} p(\mathbf{z}_t \mid \mathbf{x}_{t|t-1}^{(i)})$$

$$\mathbf{x}_{t|t}^{(i)} = \mathbf{x}_{t|t-1}^{(i)}$$

After conditioning we NEED to resample

this will be the
distribution
after resampling

$$\{j^{(i)}\} = \text{uniformSample}(\{w_{t|t}^{(i)}\})$$

$$\tilde{\mathbf{x}}_{t|t}^{(i)} = \mathbf{x}_{t|t}^{(j^{(i)})}$$

$$\tilde{w}_{t|t}^{(i)} = \frac{1}{\#\text{samples}}$$

Update(2): code

```
1  for i=1:num_particles
2      %still in the particle cycle
3      [...]
4      %once the association matrix of
5      %the current particle is filled
6
7      negative_log_likelihood = min(l_mn ');
8      dropped_measurements = 0;
9      for k=1:num_landmarks_seen
10         if(negative_log_likelihood(k) > l_miss)
11             negative_log_likelihood(k) = 0;
12             dropped_measurements++;
13         endif;
14     endfor
15
16     sum_of_negative_log_likelihood = sum(
17         negative_log_likelihood);
18     weights(i) *= %TODO;
19 endfor
20 endfunction
```

We can update the weights as:

$$p(\mathbf{z}_t \mid \mathbf{x}_t^{(i)}) \propto \exp \left(-k^{(i)} l_{\text{miss}} - \sum_m l_m^{(i)} \right)$$
$$w_{t|t}^{(i)} = w_{t|t-1}^{(i)} p(\mathbf{z}_t \mid \mathbf{x}_{t|t-1}^{(i)})$$

UniformSample: code

```
1 function sampled_indices = uniformSample(weights ,  
    num_desired_samples)  
2  
3 dim_weights = size(weights,1);  
4 %normalize the weights (if not normalized)  
5 normalizer = 1./sum(weights);  
6 %resize the indices  
7 step = 1./num_desired_samples;  
8  
9 y0 = rand()*step; %sample between 0 and 1/num_desired_sample;  
10 yi = y0; %value of the sample in the y space  
11 cumulative = 0; %this is our running cumulative distribution  
12  
13 for weight_index=1:dim_weights  
14     cumulative += normalizer*weights(weight_index); %update cumulative  
15     % fill with current_weight_index  
16     % until the cumulative does not become larger than yi  
17     while(cumulative > yi)  
18         sampled_indices(end+1,1) = weight_index;  
19         yi += step;  
20     endwhile  
21  
22 endfor  
23  
24 endfunction
```

Resampling: code

```
1 function [new_samples, new_weights] = resample(samples, weights,
2         dim_samples)
3
4     indices = uniformSample(weights', dim_samples);
5
6     new_samples = samples;
7     new_weights = ones(1, dim_samples) / dim_samples;
8
9     for i=1:dim_samples
10         new_samples(:, i) = %TODO;
11     endfor
12 endfunction
```

Call uniformSample

$$\{j^{(i)}\} = \text{uniformSample}(\{w_{t|t}^{(i)}\})$$

$$\tilde{\mathbf{x}}_{t|t}^{(i)} = \mathbf{x}_{t|t}^{(j^{(i)})}$$

Reset weights

$$\tilde{w}_{t|t}^{(i)} = \frac{1}{\text{\#samples}}$$