SISTEMAS ROBÓTICOS AUTÓNOMOS

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores de Faculdade de Engenharia da Universidade do Porto

A. Paulo G. M. Moreira
SISTEMAS ROBÓTICOS AUTÓNOMOS
(course programme)

1 - Introduction, development environments for robotic applications: simulators and ROS - *Robotic Operating Systems*
2 – Different types of locomotion and traction
3 – Kinematics, dynamics and control of mobile robots
4 – Perception and sensors
5 – Localization for mobile robots
6 – Trajectory planning
7 – Task scheduling
8 – SLAM - Simultaneous Localization and Mapping
The probabilistic localization is a method which calculates / approximates the probability distribution of the robot localization at each instant.

Its ingredients are:

- The initial probability distribution $p(l)_{t=0}$
- The probabilistic model of each sensor (not necessarily linear and Gaussian)
- Data acquired by each sensor
- Probabilistic model of robot motion (not necessarily Linear and Gaussian)
- A map of the environment
PROBABILISTIC LOCALIZATION

Conditional probability:

\[ p(A \land B) = p(A | B)p(B) \]

and

\[ p(A \land B) = p(B | A)p(A) \]

hence the Bayes formula:

\[ p(A | B) = \frac{p(B | A)p(A)}{p(B)} \]
PROBABILISTIC LOCALIZATION

What is the probability that the robot is in position $l$ given the measure $s$?
**PROBABILISTIC LOCALIZATION**

<table>
<thead>
<tr>
<th>Initial Condition</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>The robot is placed somewhere in the environment but it is not told its location. Therefore, $p(l)$ is a uniform distribution</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Perception Phase</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>The robot queries its sensors and finds out it is next to a door but it does not know which one $p(l) = p(s)l \cdot p(l)/p(s)$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Action Phase</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>The robot moves one meter forward. To account for inherent noise in robot motion the new belief is smoother. Saying it in other words, the new belief is computed through convolution with the encoder probability: $p(l) = p(l)^* p(l)_{Enc}$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Perception Phase</th>
<th></th>
</tr>
</thead>
</table>
| The robot queries its sensors and again it finds itself next to a door and update its probability distribution $p(l) = \alpha \cdot p(s \mid l) \cdot p(l)$  
Factor $\alpha$ here represents the $1/p(s)$ |  |
PROBABILISTIC LOCALIZATION

Action/Prediction phase:
- Uses proprioceptive sensors, such as encoders and gyroscopes. In other words robot sensors that measure position / movement without depending on anything external to the robot.

Perception/Actualization phase:
- Uses exteroceptive sensors such as sonar, compasses and cameras. In other words robot sensors that measure the position / movement but rely on something external to the robot.
GRID PROBABILISTIC LOCALIZATION

Markov with grid map:

$P(l(x, y, \theta) | s_1, \ldots, s_n)$

(cap. 5 pag. 233)
# PROBABILISTIC LOCALIZATION

<table>
<thead>
<tr>
<th>Markov Localization</th>
<th>Kalman Filter based Localization</th>
</tr>
</thead>
<tbody>
<tr>
<td>• The configuration space is divided into many cells. The configuration space of a</td>
<td>• The probability distribution of both the robot configuration and the sensor model is assumed to be continuous and Gaussian!</td>
</tr>
<tr>
<td>robot moving on a plane is 3D dimensional (x,y,θ). Each cell contains the probability</td>
<td></td>
</tr>
<tr>
<td>of the robot to be in that cell.</td>
<td></td>
</tr>
<tr>
<td>• The probability distribution of the sensors model is also discrete.</td>
<td>• Since a Gaussian distribution only described through mean value μ and</td>
</tr>
<tr>
<td>• During Action and Perception, all the cells are updated</td>
<td>variance σ², we need only to update μ and σ²! Therefore the computational cost is very low!</td>
</tr>
<tr>
<td>• PROS:</td>
<td></td>
</tr>
<tr>
<td>1. any probability distribution and any statistical error model for the sensor can</td>
<td>• PROS</td>
</tr>
<tr>
<td>be considered</td>
<td>At every Action and Perception update we need to update only μ, σ, therefore we need 4 equations: 2 during the Action and 2 during the Perception phase.</td>
</tr>
<tr>
<td>2. the robot can start from any unknown position and can recover from</td>
<td></td>
</tr>
<tr>
<td>ambiguous situations</td>
<td>• As we need to update only 2 quantities instead of many cells, the computational cost is very low.</td>
</tr>
<tr>
<td>3. can be used for topological localization</td>
<td></td>
</tr>
<tr>
<td>• CONS:</td>
<td>• CONS:</td>
</tr>
<tr>
<td>Every cell must be updated during Action and Perception. In some cases the</td>
<td>Only Gaussian distributions are considered and therefore if the robot probability</td>
</tr>
<tr>
<td>computation can become too heavy for real-time operations. For example, a robot</td>
<td>distribution or the sensor model cannot be approximated by a normal distribution the</td>
</tr>
<tr>
<td>moving on a plane is described through (x,y,θ). If we have a 20x100 m² environment</td>
<td>Kalman filter cannot be adopted or will give poor results. It may also not converge!</td>
</tr>
<tr>
<td>and the size of each cell is 0.1 m in x-y and 1 deg in θ, then the configuration</td>
<td>Furthermore it is not possible to recover from ambiguous situations or situations where the robot is completely lost!</td>
</tr>
<tr>
<td>space would sum up to 100x20x100x360 = 72x10⁶ cells which need to be updated at each</td>
<td></td>
</tr>
<tr>
<td>time!!!</td>
<td></td>
</tr>
</tbody>
</table>
MONTE CARLO LOCALIZATION

Particles Filter:

- The probability density function of the robot's pose is represented by a set of particles and not a function.

- It is a recursive algorithm based on Bayes rule, which propagates (Action / Prediction phase) and updates (Perception / Update phase) these same particles.

- The accuracy of the result depends on the number of particles (it must take careful with the computational weight)
MONTE CARLO LOCALIZATION

Particles Filter / MCL - Monte Carlo Localization, basic version:

1: Algorithm MCL($X_{t-1}, u_t, z_t, m$):
2:     $\bar{X}_t = X_t = \emptyset$
3:     for $m = 1$ to $M$ do
4:         $x_t^m = \text{sample\_motion\_model}(u_t, x_t^{m-1})$
5:         $w_t^m = \text{measurement\_model}(z_t, x_t^{m}, m)$
6:         $\hat{X}_t = \hat{X}_t + \langle x_t^m, w_t^m \rangle$
7:     endfor
8:     for $m = 1$ to $M$ do
9:         draw $i$ with probability $\propto w_t^i$
10:        add $x_t^i$ to $X_t$
11:     endfor
12: return $X_t$
MONTE CARLO LOCALIZATION

Particles Filter / MCL - Monte Carlo Localization, basic version:

Line 1: the algorithm receives the previous set of particles, input signals to the motion model, the measurements provided by sensors and a map.

Line 2: the current set of particles is empty.

Lines 3 and 7: MOTION AND INCORPORATION OF MEASURES
each particle, using the motion model, is moved to a new location and then, using the sensors model, is assigned a probability / weight for that location.

Lines 8 and 11: RESAMPLING
generating M random numbers, the new particles are chosen proportional to its weight / probability.
MONTE CARLO LOCALIZATION

Extracting the function of the probability density from the particles:

(b) assuming a Gaussian and calculating the mean and standard deviation.

(c) calculating a histogram

(d) using a Gaussian kernel, each particle is assumed to be the center of a Gaussian distribution. The choice of the standard deviation smoothes the resultant curve.
MONTE CARLO LOCALIZATION

Influence of particle number:

(a) 25 samples

(b) 250 samples
MONTE CARLO LOCALIZATION

Problems associated with resampling

The resampling made by the previous way gives rise that a stopped robot without sensors have increasingly sure of its localization, i.e. the particles tend to concentrate.

A solution is to stop to make resampling and integrating measures when the robot is stopped and when it is moving to have a resampling method on the basis of the variance of the different particle's weights.

Another solution is to use a low variance sampling.
MONTE CARLO LOCALIZATION

Low variance resampling

1: Algorithm $\text{Low\_variance\_sampler}(\mathcal{X}_t, \mathcal{W}_t)$:
2: $\hat{\mathcal{X}}_t = \emptyset$
3: $r = \text{rand}(0; M^{-1})$
4: $c = w_t^{[1]}$
5: $i = 1$
6: for $m = 1$ to $M$ do
7: $u = r + (m - 1) \cdot M^{-1}$
8: while $u > c$
9: $i = i + 1$
10: $c = c + w_t^{[i]}$
11: endwhile
12: add $x_t^{[i]}$ to $\hat{\mathcal{X}}_t$
13: endfor
14: return $\hat{\mathcal{X}}_t$
MONTE CARLO LOCALIZATION

Low variance resampling

Being the particles aligned with a width proportional to its weight, starting from a random point, runs up all particles with a constant step equal to $1 / M$ and selects the corresponding particle.

Advantages:
- runs throughout the space particle in a more systematic way.
- if all particles have the same weight, the resulting particle set is the same.
MONTE CARLO LOCALIZATION

One dimension example

a) randomly distributed particles
b) observes a door: the particles near that door are over weight by the sensor model.
c) new distribution of the particles and application of motion model. It has three zones with a higher particle densit.
d) observes a door: the particles near that door are over weight by the sensor model.
c) new distribution of the particles and application of motion model. Now the zone with higher particle density coincides with the actual localization of the robot.
MONTE CARLO LOCALIZATION

Failure recover

In the form presented the particulate filter does not recover from a kidnapping of the robot or a bad estimative of the localization because after a while, all the particles will be in one area near real position of the robot.

To avoid this you should always place some particles randomly along the navigation space of the robot. The two questions that arise are: how many and how to distribute them.

One option could be a fixed number of random particles but it yields better results if we change the number on the average probability of current measures that can be approximated by the average particle weights:

$$p(z_t \mid z_{t-1}, u_t, m) \approx \frac{1}{M} \sum_{m=1}^{M} w_t^{[m]}$$

A low value indicates that the robot can be lost (or badly localized) and it is better to release more particles randomly. Due to sensor noise or a large dispersion of particles, we should filter this value with a fast filter and another slow filter, acting in function of these two results.
MONTE CARLO LOCALIZATION

Failure recover

1: Algorithm Augmented_MCL($X_{t-1}$, $u_t$, $z_t$, $m$):
2:     static $w_{slow}$, $w_{fast}$
3:     $\tilde{X}_t = X_t = \emptyset$
4:     for $m = 1$ to $M$ do
5:         $x_t^{[m]} = \text{sample motion model}(u_t, x_t^{[m-1]})$
6:         $w_t^{[m]} = \text{measurement model}(z_t, x_t^{[m]}, m)$
7:         $\tilde{X}_t = \tilde{X}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
8:         $w_{avg} = w_{avg} + \frac{1}{M} w_t^{[m]}$
9:     endfor
10:    $w_{slow} = w_{slow} + \alpha_{slow}(w_{avg} - w_{slow})$
11:    $w_{fast} = w_{fast} + \alpha_{fast}(w_{avg} - w_{fast})$
12:    for $m = 1$ to $M$ do
13:        with probability $\max(0.0, 1.0 - w_{fast}/w_{slow})$ do
14:            add random pose to $X_t$
15:        else
16:            draw $i$ with probability $\propto w_t^{[i]}$
17:            add $x_t^{[i]}$ to $X_t$
18:        endwith
19:    endfor
20:    return $X_t$
MONTE CARLO LOCALIZATION

Failure recover

Illustrative example
(Robocup Sony Aibo dogs):
MONTE CARLO LOCALIZATION

Additional improvements:

If we have an almost perfect sensor with a probabilistic model very close to the deterministic model, the particulate filter may not work because for the minimum deviation from the robot localization estimative almost all particles will be at zero weight.

To solve this problem we can artificially increase the sensor noise model, including in the model other error sources such as the approaches used in the algorithm.

Another possibility will be a part of the particles being placed in a localization related to the observations model, ie in places where the observations fit the map:

\[
x_t^{[m]} \sim p(z_t \mid x_t)
\]

and the weight of the particles is calculated by the robot motion model:

\[
w_t^{[m]} = \int p(x_t^{[m]} \mid u_t, x_{t-1}) \text{bel}(x_{t-1}) \, dx_{t-1}
\]
MONTE CARLO LOCALIZATION

Dynamic environments:

In very dynamic environments, for example with many people around the robot, we have additional difficulties for the particles filter.

One solution is to estimate, beyond the location of the robot, the position and speed of people and other unforeseen obstacles. This approach is not easy because it greatly increases the size of the state estimative, which is time variant.

Alternatively you can perform a preliminary processing of the measured data, eliminating unexpected sensor's measures arising from objects not present on the map (outliers).
References:

Figures from:
