# Monte Carlo Localization Using Particle Filter

Computer Vision Lab Exercise 03

## Goal

The goal of this exercise is to implement a Monte Carlo Localization (MCL) algorithm to localize a one-dimensional mobile robot being moved, with constant velocity, in a hallway. You will program all the code in **Python**.

## MCL Problem

The MCL is an implementation of the Markovian Localization problem where the involved pdfs (probability density functions) are represented through samples (particles) and the Bayes Filter is implemented through the Particle Filter. Markov Localization addresses the problem of state estimation from sensor data. MCL is a probabilistic algorithm and instead of maintaining a single hypothesis as to where in the world a robot might be, MCL maintains a probability distribution over the space of all such hypotheses. The probabilistic representation allows it to weight these different hypotheses in a mathematically sound way.

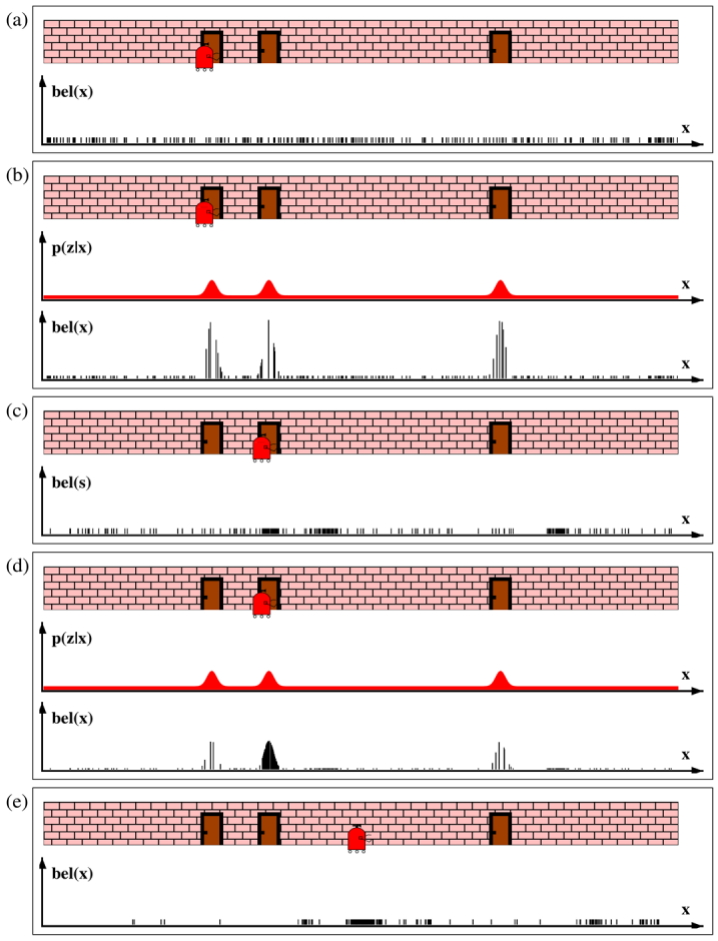


Figure 1: MCL of a mobile robot moving in a hallway.

In Figure 1, the initial global uncertainty is achieved through a set of pose particles drawn at random and uniformly over the entire pose space, as show in Figure 1(a). As the robot senses the door, line 5 (see Figure 2) of in the algorithm MCL assigns importance factors to each particle. The resulting particle set is shown in Figure 1(b). The height of each particle in this figure shows its importance weight. It is important to notice that this set of particles is identical to the one in Figure 1(a). The only thing modified by the measurement update are the importance weights.

Then Figure 1(c) shows the particle set after re-sampling (line 8-11 in the algorithm MCL) and after incorporating the robot motion (line 4). This leads to a new particle set with uniform importance weights, but with an increased number of particles near the three likely places. The new measurement assigns non-uniform importance weights to the particle set, as shown in Figure 1(d). At this point, most of the cumulative probability mass is centered on the second door, which is also the most likely location. Further motion leads to another re-sampling step, and a step in which a new particle set is generated according to the motion model (Figure 1(e)). The particle sets approximate the correct posterior, as would be calculated by an exact Bayes filter.

## Algorithm

Figure 2 shows a general Particle Filter and Figure 3 the pseudocode of the MCL Algorithm as an instance of the Particle Filter.

1: **Algorithm Particle_filter**($\mathcal{X}_{t-1}, u_t, z_t$)**:**
2: $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
3: for $m = 1$ to $M$ do
4: sample $x_t^{[m]} \sim p(x_t \mid u_t, x_{t-1}^{[m]})$
5: $w_t^{[m]} = p(z_t \mid x_t^{[m]})$
6: $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
7: endfor
8: for $m = 1$ to $M$ do
9: draw $i$ with probability $\propto w_t^{[i]}$
10: add $x_t^{[i]}$ to $\mathcal{X}_t$
11: endfor
12: return $\mathcal{X}_t$

Figure 2: Particle filter algorithm.

1: **Algorithm MCL**($\mathcal{X}_{t-1}, u_t, z_t, m$)**:**
2: $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
3: for $m = 1$ to $M$ do
4: $x_t^{[m]} =$ **sample_motion_model**$(u_t, x_{t-1}^{[m]})$
5: $w_t^{[m]} =$ **measurement_model**$(z_t, x_t^{[m]}, m)$
6: $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
7: endfor
8: for $m = 1$ to $M$ do
9: draw $i$ with probability $\propto w_t^{[i]}$
10: add $x_t^{[i]}$ to $\mathcal{X}_t$
11: endfor
12: return $\mathcal{X}_t$

Figure 3: MCL algorithm.

## Work to do

You must complete the `.py` file containing the algorithm. These are the steps you must follow:

1. Make sure that the Python environment has been successfully installed. A recommended environment should have `Python 3.7.3`, with the following libraries, `numpy (1.17.4)`, `matplotlib (3.1.2)`, and `scipy (1.4.1)`.
2. Read the `.py` file first and try to understand how it works. Even though it is incomplete, it is possible to run. It is necessary to have the `animation.mat` file in the same folder.
3. Program the `getMeasurementModel()` function to compute the probability of observing a door given the location.
4. Program the `moveParticles()` function to simulate the motion of the robot, given the odometry observation. Since it is not a perfect motion, include the noise as well.
5. Program the `measureParticles()` function to calculate the weight of each particle.
6. Program the `resampleParticles()` function either using Resampling Wheel or Low Variance Sampling.
7. After finishing the lab exercise, upload **ONLY ONE** file (no `.zip` or multiple versions), and please name the file as: `particle_filter_{YOUR-LEGI-NUMBER}.py`, for example, `particle_filter_20-987-654.py`.

**Note 1**: **Report is NOT required!** If the code is running like the reference video it should be enough. Only write a report which should be a single `.pdf` file with no more than **four A4 pages** if you believe your solution is not working properly. In case you decide to write a report, please clearly describe the problem encountered and explain the details.

**Note 2**: Please do not modify the code before the comment

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%    COMPLETE THESE FUNCTIONS    %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

**Note 3**: You only need to concentrate in these four functions within the `Simulator` class: `getMeasurementModel()`, `moveParticles()`, `measureParticles()`, and `resampleParticles()`.

**Note 4:** Your **CANNOT** use any built-in function for sampling.

## Learning Materials

The lab can be solved with few lines of code. However, understanding is the key. Since no report is required, most of the time will be expending on learning. In order to complement the lecture (which does not actually include MCL), it is highly recommended to see some of the great videos from Sebastian Thrun in the Udacity course: *Artificial Intelligence for Robotics*.

**Videos**: Link of the Udacity course can be found next. In particular for the Lesson 1, it is also recommended to complete the Quiz questions (which are in Python), it will give you some insight in how to complete the lab, the code is quite similar. Lesson 4 (Kalman Filters) is **NOT** required for this exercise and can be skipped.

> Lesson 1 [Localization] and Lesson 8 [Particle Filters] (and some of them quiz questions).

**Books**: This lab exercise is based on the *Probabilistic Robotics* book [ETH Library]. Available digital version [pdf]. If you want to complement the learning even further, the recommended chapter are: Chapter 1., Chapter 2. (Bayes Filter included), Chapter 8. (until 8.3.3). You could also see Chapter 4.2 in the book for re-sampling techniques, like Low Variance Resampling (Table 4.4 in the book).