Universitat de Girona

COMPUTER VISION AND ROBOTICS RESEARCH INSTITUTE (VICOROB)

Probabilistic Robotics

Lab Report

Particle Filter Algorithm

No.	Full Name	Student ID
1	Moses Chuka Ebere	u1985468
2	Joseph Oloruntoba Adeola	u1988552

Instructor: Eduardo Ochoa

UDG, DECEMBER 2022

1 Objective

The aim of this lab is to implement the particle filter algorithm to localize a two-dimensional robot (turtlebot) in a given map.

2 Introduction

The particle filter or Montecarlo Localization algorithm is a recursive state estimation algorithm whose key idea is to localize a robot by representing the posterior belief $bel(x_t)$ by a set of random particle samples drawn from a Gaussian distribution.

3 Implementation

We begin by introducing a set of N samples and weight. Each sample (particle) represent a possible state the robot can be. The weight is a probability that represents the relative importance of each of the N samples and the sum of all weights equal to one. High weight samples are thought to be closer to the true state sequence than low weight samples. As a recursive state estimation algorithm, the three major steps for implementing the particle filter algorithm are: prediction, weighting (update), and resampling. These steps are discussed in the following subsections.

3.1 Prediction

In the prediction step, the new state of each particle (x_k, y_k, θ_k) is obtained by adding the odometry reading $(\Delta x_k, \Delta y_k, \Delta\theta_k)$ to the previous state of the particle $(x_{k-1}, y_{k-1}, \theta_{k-1})$. To achieve this, zero mean Gaussian noise is first added to the odometry reading to model the uncertainty in odometry measurement. The measurement is then transformed from the robot frame to the world frame using the transformation matrix $({}^{W}T_R)$ before adding it to the previous particle state. This step is demonstrated below

$$\begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} = \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \theta_{k-1} \end{bmatrix} + \left\{ \begin{matrix} \begin{bmatrix} \cos\theta_{k-1} & -\sin\theta_{k-1} \\ \sin\theta_{k-1} & \cos\theta_{k-1} \end{bmatrix} \begin{bmatrix} \Delta x_k + noise_x \\ \Delta y_k + noise_y \end{bmatrix} \\ \Delta\theta_k + noise_\theta \end{matrix} \right\}$$

where $noise_x, noise_y$ and $noise_\theta$ are zero mean gaussian noise added to the odometry's x, y and θ reading.

3.2 Weighting

The robot's range and bearing sensors provide measurements that are used to implement the weighting step, also known as the weight update. As the robot scans the room, the lines returned from the $split\_and\_merge()$ function are contrasted with the lines in a predefined map of the room. First, all the lines measured by the robot are converted to polar coordinates $(range, bearing)$ by using the $get\_polar\_line()$ function. Similarly, all the predefined lines in the room's map are also converted to polar coordinates for each particle (the particle is passed as the second argument of the $get\_polar\_line()$ function). The weight function, which is Gaussian, is then used to estimate the likelihood (probability) of each measured line with all expected line (lines from the map). This step is carried out independently for range and bearing. The $range\_weight$ is then multiplied with the $angle\_weight$ and the result is stored in an array.

$$w = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right]$$

Where x the measured value (range or angle), µ the expected value (extracted from given map lines) and σ the uncertainty of the measurement.
Next, the optional part of the lab is implemented by comparing the length of the lines observed with the length of all the lines in the room's map. If the length of any observed line is greater than the length of all the lines in the room's map, the weight at that point will be set to zero. Finally, for each measured line, the maximum element in each row of the *weight* array is chosen and stored in *max_weight*. Next, all the elements of *max_weight* are multiplied together, and the value obtained is stored as *w_new*. The particle's weight is then updated by multiplying its previous weight w_{k-1} with the newly computed w_{new}.

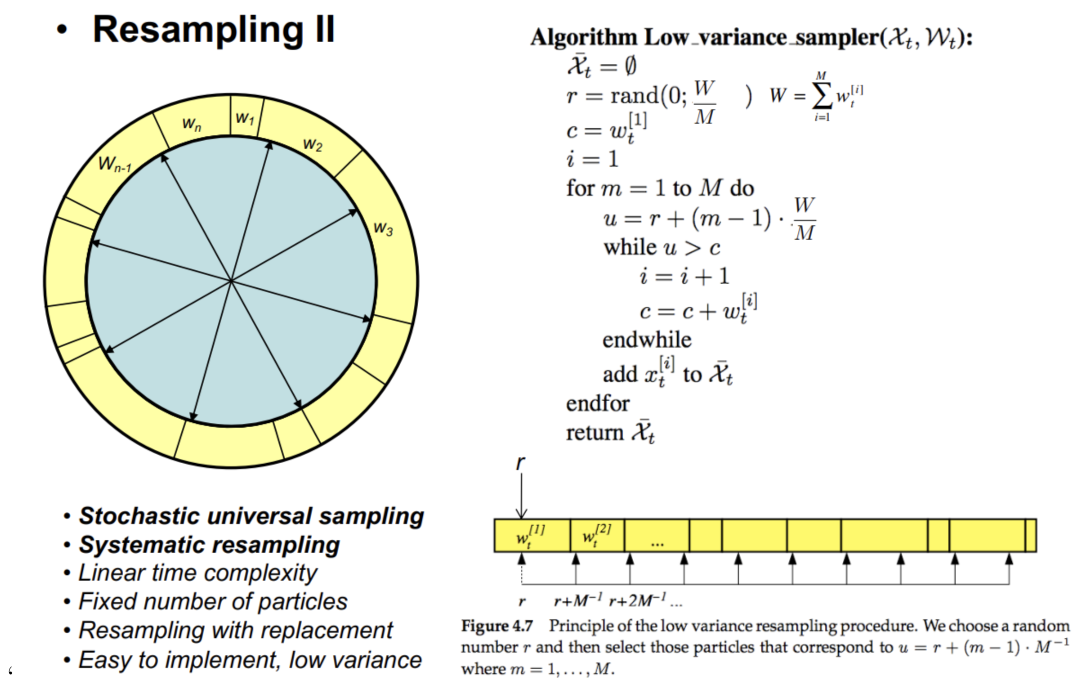
$$w_k^i = w_{k-1}^i \times w_{new}$$

3.3 Resampling

Resampling is the process that converts a set of N weighted particles into a new set of N weighted particles that are typically uniformly distributed. In the node file provided, the resampling step is always performed whenever the particle weights indicate some form of particle degeneracy. Particle degeneracy is measured by evaluating

$$N_{eff} = \frac{1}{\sum_i^n w^{[i]}}$$

Whenever N_{eff} is less than a given threshold, the systematic resampling algorithm will be implemented.
In systematic resampling, new particles are chosen at random from a set of weighted particles, with replacement. The likelihood of selecting a particle is related to its weight, while the number of particles remains constant. Higher weight particles are more likely to be reproduced, whereas lower weight particles are more likely to be deleted. (That is, certain particles are chosen several times while others are not). After resampling, the weight of all the particles is set to $1/N_s$, where N_s is the number of particles. The Psuedocode in figure.1 below explains the systematic resampling algorithm



- **Resampling II**

- ***Stochastic universal sampling***
- ***Systematic resampling***
- *Linear time complexity*
- *Fixed number of particles*
- *Resampling with replacement*
- *Easy to implement, low variance*

Algorithm Low_variance_sampler($\mathcal{X}_t, \mathcal{W}_t$):

```
    $\bar{\mathcal{X}}_t = \emptyset$
    $r = \text{rand}(0; \frac{W}{M})$   $W = \sum_{i=1}^{M} w_t^{[i]}$
    $c = w_t^{[1]}$
    $i = 1$
    for $m = 1$ to $M$ do
        $u = r + (m-1) \cdot \frac{W}{M}$
        while $u > c$
            $i = i + 1$
            $c = c + w_t^{[i]}$
        endwhile
        add $x_t^{[i]}$ to $\bar{\mathcal{X}}_t$
    endfor
    return $\bar{\mathcal{X}}_t$
```

Figure 4.7 Principle of the low variance resampling procedure. We choose a random number r and then select those particles that correspond to $u = r + (m-1) \cdot M^{-1}$ where $m = 1, \ldots, M$.

Figure 1: Systematic resampling algorithm (Source: Lecture slides)

4 Discussion

One issue that came up while writing the code for the particle filter algorithm was the particle degeneracy problem, where one particle takes practically all the weight, and other particles' contribution to localization become insignificant. For us, this issue arose due to incorrect implementation of the weighting step. We had a hard time comprehending the weighting step before we successfully implemented it.