Monte Carlo Methods in Mobile Robotics: The Particle Filter

M. Abdul Galil

***Abstract*—In mobile robotics, the localization problem, defined as the ability of the robot to localize itself with respect to the environment without any external references other than a map of the environment, is an essential step towards any other high level mobile robotics applications such as: Mapping, Autonomous Navigation .. etc. This report tackles one of the recent, yet very famous, approaches for solving this problem, namely the Particle Filter. Particle filters are based on Monte Carlo Methods of statistics. A demonstration of the particle filter localization is attached with the report using MATLAB software for simulating a hypothetical mobile robot navigating a hypothetical indoor environment. Furthermore, the report discusses the potential improvements that were suggested in the literature to enhance the performance of the algorithm and to decrease its computational cost.**

***Keywords*—Monte Carlo, Mobile Robotics, Localization, Particle Filter, MATLAB.**

I. Introduction

The need for robots inside indoor environments to perform various tasks is increasing, robots can handle a lot of our tedious work, making our life easier. However, the problem with indoor environments, such as houses, company buildings and so on, pose a challenge for robots. This challenge is that GPS signals are attenuated and often weakened inside buildings, leading to an erroneous position estimate. This complicates the problem of localization inside indoor environments and thus hinders the ability of the robots to perform various tasks such as doing laundry, cooking and cleaning houses or companies. Indoor environments might also include collapsed buildings, where there might be survivors waiting to be rescued. Search and rescue missions are best performed by robots, specially under the dangerous conditions of a collapsing bu idling. However, this is not feasible if the only source of information about position is GPS because it is useless under such conditions. This problem was addressed in research for many years. Many scientists poured their efforts into the process of finding a solution. Finally a feasible solution was proposed. By using recursive state estimation, through a framework of a **Bayes Filter**, the location of the robot can be tracked successfully with respect to the initial position, allowing the robot to keep track of its position while navigating indoor environments without the need for an external reference such as GPS, only a map of the environment is needed. The following subsections provide an concise introduction to recursive estimation, Bayes filter framework and the inevitable associated **Uncertainty** in the process.

A. *The inherent uncertainty*

Indoors environment are inherently unpredictable; cases where people moving around or changing positions of obstacles are very likely, making the deterministic approach for mobile robot localization unfeasible because it is very likely to fail if any of the above conditions happened. Elements contributing to uncertainty are many, below is a a list of them with a brief description of each.

1. **Environment**: Physics might be deterministic in simple situations where an isolated phenomenon is observed, but when there are too many mutually interacting phenomena involved, determinism become meaningless because we can not be 100 percent sure about what is happening.

2. **Sensors**: All sensors have inherent uncertainty in the measurements they provide, this noise arises from two factors. The first being the limitations imposed by the physical phenomenon upon which a sensor is based. The second being the noise that is present in all natural processes.

3. **Robots**: The actuators of the robots such as motors or pneumatic systems are not precise, they don't produce the exact actuation even when the same control signal is sent.

4. **Models**: The models developed to provide an estimate of the way a robot's state evolves involve many many assumptions which definitely affects the accuracy, thus the resulting models are a crude description of the real process.

5. **Computation**: Even the computations involved under the assumed-to-be-accurate models of the physical processes are not easy and require a huge amount of computation power to be fully implemented. This is not allowed because most robotic systems are real-time systems, meaning that they need to respond in a real-time fashion. Thus these algorithms are often approximated to achieve real-time response.

This uncertainty must be taken into account as robotics is facing new challenges such as indoor localization. This lead to the birth of the robotics variant of the probabilistic approach; Probabilistic Robotics.

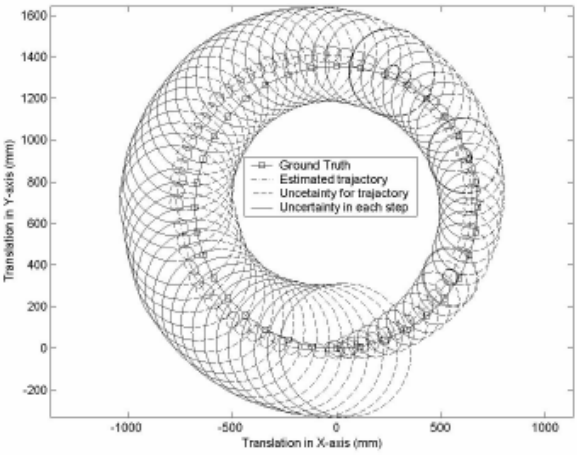
B. *Probabilistic Robotics*

Determinism in a highly unstructured and noisy world is no longer a valid assumption. We simply have to accept the fact that we can not know the exact position or state of a robot at a given time with zero uncertainty except in very few controlled cases. We can at best have estimates of the

state accompanied with an associated probability of being the correct state for each estimate. Furthermore, we have to start incorporating this fact into our algorithms in order to make robots capable to safely operate under such conditions.

To illustrate the importance of this, let's do a thought experiment. A robot is navigating an environment of which a map is available to the robot. The robot is navigating the environment without any external reference on position, by repeatedly checking the map for the probable obstacles that might face it during its traverse and planning its motion according to the feasible paths. The robot keeps track of its position by using odometry. Everything should be okay, as long as the robot doesn't lose track of its position, other wise it won't be able to avoid obstacles nor perform the motion planning. However, various sources of error exist (such as wheel slipping in the case of using an encoder for estimating odometry). This leads to an increasing uncertainty in the estimate of the position of the robot. Consider fig[1]:

Fig. 1. Uncertainty in Robot Localization



The growing circles represent the uncertainty on the robot's current position, and as illustrated by fig[1], as the robot continues moving, this uncertainty grows steadily unless a mean of limiting this uncertainty is followed. This uncertainty if not accounted for will lead to undesirable consequences such as the even where the robot collides with an object believing that it is away.

II. Recursive Estimation

Recursive estimation is the process of fully estimating the current state of a system given the previous states. In order to fully understand this procedure, concepts such as *state*, *Markov Assumption* and *Bayes Filter* must be introduced.

A. State of a System:

In the context of the localization problem, it's sufficient to think of the *state* of the system as the collection of all aspects of the robot and its environment. This includes the position of the robot, the locations of the objects in the environment .. etc. States often develop with time, such as the position of the robot. However, some states are static, such as the positions of the objects in the environment.
A state is known to be *complete* if estimating the future state of the system can be done using only the current state of the system. This implies that all the previous states doesn't directly affect the way in which the system is going to evolve in the future. If the state is chosen such it satisfies this condition, the system is described as a *Markov Chain*.
Picking the state of the system to be complete is impractical and most of the time is impossible. In the case of mobile robot localization, a complete state doesn't just include the position of the robot. It has to include the types of the materials of the objects of the environment, the state of the robot's sensors, the illumination of the room (in case of using a camera sensor), the thoughts going inside the minds of the people who are in the environment (since they pose potential moving objects). It's clear that this choice of state is by all means impractical. At best, the current state-of-art algorithms pick the state of the combined system of the robot and the environment to consist of the pose of the robot (position and orientation), a map of the environment and maybe some parameters associated with the sensors of the mobile robot. Even though this is not a complete state of the system, it is often okay to assume the completeness of this state and get away with good results. This assumption is commonly known as *Markov Assumption*.
In mobile robotics, states are continuous, most of the time. Some mobile robotics systems might include discrete states too, this type of systems is referred to as hybrid systems. The states in mobile robots are often changing, and develop according to a set of differential equations in a stochastic manner.

B. Interaction between a Robot and Environment:

There are two ways in which an interaction between the robot and its environment can take place. These two are Actuation and Perception. The robot can affect its environment using actuators, and it uses its sensors to perceive the environment, in other words to gain knowledge about it. The first is done by means of **Control** and the second by means of **Measurement**. Often, Control tends to increase the uncertainty around the state and Measurement tends to decrease this uncertainty. It is by combining measurement with the control that a robot is able to move and interact with its environment without substantially increasing the uncertainty around its state.

C. Belief Distributions:

Instead of representing the state of the robot in an n-dimensional state space by a deterministic single point, this state is better thought of as an (n+1)-dimensional space in which the additional dimension is the probability (***Belief***) distribution associated with each state. This clearly requires big changes in our natural intuition about the problem. Belief Distributions are formally defined as: posterior probabilities

over the state variables conditioned on the available data, denoted as *bel(x_t)* for a hypothetical state x_t and is represented mathematically by:

$$bel(x_t) = p(x_t|z_{1:t}, u_{1:t})$$

In the previous equation, $z_{1:t}$ and $u_{1:t}$ represent all the controls and measurements since the beginning of time till time t. Another important concept is the prediction probability distribution which represents a temporary estimate of the state of the system given all previous controls and all measurements except for the last measurement. It can be thought of as a prediction step in the recursive estimation algorithm that precedes incorporating the measurement into the prediction which is called the update step. It is through this step that the system is propagated in time. This propagation is stochastic in nature, so it increases uncertainty by default. The update step is therefore mandatory to limit the growth of this uncertainty as the system evolves in time.

D. *Bayes Filter*

The Bayes filter is the most basic algorithm for calculating belief distributions. To calculate the belief at time t, the algorithm takes as input the previous belief *bel*(t-1), the control action at time t $\rightarrow u_t$ and the measurement at time t $\rightarrow z_t$. This clearly says that Bayes Filter algorithm is a recursive algorithm. The basic algorithm is depicted in the following figure:

1: **Algorithm Bayes_filter(**$bel(x_{t-1}), u_t, z_t$**):**
2: for all x_t do
3: $\overline{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1})\ bel(x_{t-1})\ dx$
4: $bel(x_t) = \eta\ p(z_t \mid x_t)\ \overline{bel}(x_t)$
5: endfor
6: return $bel(x_t)$

Line 3 in the previous algorithm is the so called prediction step where the system is stochasticly propagated into the future one time step, where as line 4 is the update step where the measurement is incorporated in the belief to calculate a better, and hopefully less susceptible to noise, estimate of the state.

III. Motion Model

In this section, motion models which are models that are used to propagate the stochastic state of the system into the future, are presented. These models are also called the state transition probability distribution functions. Robot kinematics is the back bone of motion models. However, in the context of probabilistic robotics, Robot kinematics is approached in a non-deterministic manner to account for the uncertainty as presented in the previous sections. For the purposes of this report, we will confine our discussion to the 2D case where the state of the robot constitutes of the 2D pose (x,y) and the orientation (θ) and the input in this case is assumed to be the angular and the rotation velocity. When augmented, the state vector and input vectors then expressed as:

$$x_t = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}, and \quad \mathbf{u}_t = \begin{pmatrix} v_t \\ \omega_t \end{pmatrix}$$

Normally, motion models return a probability density function for an arbitrary state x_t. However, for the purposes of particle filters, sampling from these probability density functions is sufficient. This will become clearer when we get to discussing the particle itself in a later section. But for now lets represent an algorithm representing a state transition probability density function. This algorithm is suitable for particle filters and is called sampling motion model. The algorithm is represented in the folowing figure

1: **Algorithm sample_motion_model_velocity(**u_t, x_{t-1}**):**
2: $\hat{v} = v + \mathbf{sample}(\alpha_1|v| + \alpha_2|\omega|)$
3: $\hat{\omega} = \omega + \mathbf{sample}(\alpha_3|v| + \alpha_4|\omega|)$
4: $\hat{\gamma} = \mathbf{sample}(\alpha_5|v| + \alpha_6|\omega|)$
5: $x' = x - \frac{\hat{v}}{\hat{\omega}} \sin\theta + \frac{\hat{v}}{\hat{\omega}}\ sin(\theta + \hat{\omega}\Delta t)$
6: $y' = y + \frac{\hat{v}}{\hat{\omega}} \cos\theta - \frac{\hat{v}}{\hat{\omega}}\ cos(\theta + \hat{\omega}\Delta t)$
7: $\theta' = \theta + \hat{\omega}\Delta t + \hat{\gamma}\Delta t$
8: *return* $x_t = (x', y', \theta')^T$

The parameters α_1 to α_6 are noise characteristics of the system. And the keyword *sample* represents an algorithm that generates a random sample from a zero-centered distribution that has a variance equal to the input to this algorithm.
This sampling model samples states from the state space according to a distribution determined by the α parameter set. A full derivation of the model can be found in [1], here we'll briefly mention the intuition behind it. In the ideal case, inputs are decoupled in the sense that the linear velocity input is independent of the angular velocity input. However, in practise, this is not the case. Often there's a noise that is imposed on the linear velocity input, and this noise is proportional to the magnitudes of both angular and linear velocity inputs. Same can be said about the angular velocity input. This noise is not deterministic, it doesn't have a constant value. However, this noise often is following a distribution of some kind. Tuning the α parameters set and the sample algorithm can successfully provide a mean of approximating the distribution which this noise is following. This is the approach followed in producing a suitable motion model of a certain robot. These parameters are generated using big data sets from experiments with the robot.

IV. Measurement Model

Sensors are the only way a robot is able to perceive the environment. Without sensors, robots are blind, deaf and numb. But, unlike our highly accurate perceiving systems, current state-of-the-art sensors still suffer from noise. This section is concerned with developing models that represent the uncertainty that arise from the inherent noise on the measurements of the sensors. This measurement model gives us an indication about how much information is contained in measurements. The output of the algorithm described in the following lines

is a probability density function indicating the likelihood of a measurement, given the state of the robot and a map of the environment. The model developed here models the noise on one type of sensors which are called beam range finder models. Such sensors measure the distance between the robot and the nearest obstacle in given directions. Examples on this family of sensors are Laser range finders, Laser scanners, Ultrasound sensors, etc ..

We start out by determining the types of noise that might affect the measurement of this type of sensors. These can be summarized into 4 types of noise. Note that in the following discussion, we will adopt the notation $z_t to$ represent the readings returned by the sensor and $z_t^(*)$ to denote the actual distance that should be measured by the sensor. Given the preceding, the types of noise are:

Gaussian Noise This noise arises from the imperfections in our sensor. All sensors have this noise. Assuming that this is the only noise present in the measurement process, the probability density function of measurements will be a gaussian distribution centered around z_t^* with variance σ_{hit}^2. Mathematically, this distribution is given by the following equation:

$$p_{hit}(z_t|x_t, m) = \frac{1}{\sqrt{2\pi\sigma_{hit}^2}} e^{\frac{-1}{2}\frac{(z_t - z_t^*)^2}{\sigma_{hit}^2}}$$

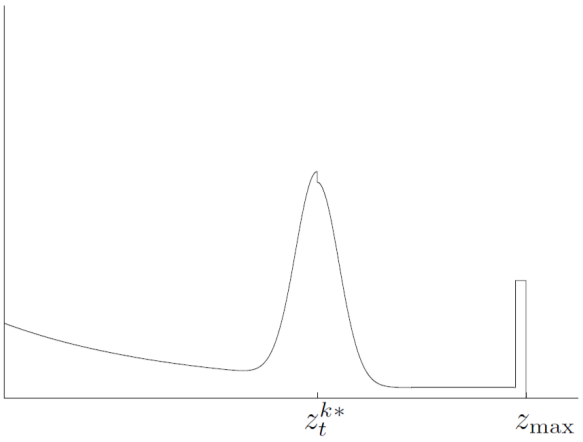
Unexpected Objects This is mainly the objects that are not listed in the map, moving objects or people. They cause the sensor to measure a distance smaller than what it should measure at a given location in the map. The probability for this event to happen follows an exponential distribution with a parameter λ_{short}. This distribution is given in the following equation:

$$p_{short}(z_t|x_t, m) = \frac{1}{\lambda_{short}} e^{\frac{z_t}{\lambda_{short}}} \; for \; 0 < z_t < z_t^*$$

Failures Sometimes the sensor completely fails at returning a measurement, because it reached its maximum range. In this case, the sensor will return its maximum range. This is represented by assigning a probability of 1 at the maximum range.

Random Noise Noise that is just random, and it is represented by a uniform distribution.

The actual model of the uncertainty associated with the measurement is a combination of these with tunable weights that sum up to one. These weights, in addition to the parameters $\lambda_{short} and \sigma_{hit}$ are typically found from experimental data. The sensors are used to generate a lot of data sets to produce a distribution, a fitting optimization problem is then solved for the set of weights and the parameters that best describes the experimental data. Below a figure of a typical shape of the resulting measurement model:



"Pseudo-density" of a typical mixture distribution $p(z_t^k \mid x_t, m)$.

V. Brief Introduction to Monte Carlo Methods

Monte Carlo methods, also known as Monte Carlo analysis, are processes that use random samples to statistically evaluate mathematical functions. In other words, Monte Carlo methods are mainly used in computing to solve problems such as organizing traffic flow, modelling the evolution of stars and predicting fluctuations in the stock market. Consequently, these methods, as many other methods, might produce some sort of errors or might also fail. However, one could reduce the percentage of error by extending the range of random samples. Different methods and applications of Monte Carlo will be discussed in the following lines.

Monte Carlo integration is a pure mathematical method used to evaluate numerical integration using random numbers. However, this method can only deal with definite integrals in a sense that Monte Carlo is extremely useful when dealing with higher-dimensional integrals.

Monte Carlo is widely used in simulation and optimization to solve various kind of problems. That is, a function of a vector that has a large number of dimensions can be minimized or maximized using the Monte Carlo method. In addition, Monte Carlo method is a very powerful tool used to function a computer chess or solve a travelling sales man problem.

Monte Carlo is also used to study the complex quantum system. For instance, a solution for a quantum many-body problem can be provided using quantum Monte Carlo. Not only that, this method, somehow, allows a direct treatment and description of complex many-body effect that is encoded in the wave functions.

Monte Carlo methods are essentially important in computational physics, physical chemistry and related applied fields. In addition, this method is also used in modelling galaxy evolution and microwave radiation. The basis of modern weather forecasting is also formed using Monte Carlo methods.

VI. MONTE CARLO LOCALIZATION

After discussing the motion and measurement models, it's time to discuss the framework in which they are combined together to produce a state estimation algorithm that localizes the robot inside an environment. This algorithm is the **Particle Filter**.

A. Particle Filter vs Kalman Filter

Particle filter is a non-parametric filter, unlike its cousin the Kalman Filter which is a parametric filter. They both are derived from the Bayes Filter, but each has different assumptions to make about the motion model. The Kalman filter assumes that the only noise present on the actuation (control) is gaussian noise, i.e. it parametrizes the noise on the motion model. Additionally, Kalman filter assumes a linear motion model, it can't deal with non-linear dynamics in its basic form, though it is possible through the Extended version of filter, known as Extended Kalman Filter, to include non-linear motion models. However, the assumption about the gaussian noise is still retained.

Particle filters on the other hand are non-parametric, they don't assume the noise on the motion to be of a particular distribution, and the most basic form of the filter deals directly with non-linear models.

B. Basic Algorithm:

Instead of representing the posterior (the state belief) by a parametric distribution (like the Kalman Filter), the algorithm represents the posterior by a set of random state samples, called "particles", hence the name. These are hypotheses about the states that are also samples of the probability density function representing the state belief. At first glance this might seem to be a bad idea, since samples are approximations of the real thing, but when considered again, this approach has a lot of advantages that the parametric approach doesn't have. Particularly, the non-parametric nature of this approach allows it to represent multi-modal distributions and other distributions that often doesn't have a closed form. This allows for a more robust representation of the state belief. The basic algorithm is represented in the following figure:

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

Here, χ_t is a the set of particles (samples) at time t, u_t is the input vector at time t and z_t is the measurement after executing the control at time t. The algorithm is recursive in nature since it is derived from the recursive Bayes filter algorithm, so it takes as input the state belief at the previous time step, namely χ_{t-1}.

Initially, the new particle set is empty. Another temporary particle set is generated first. This is done by applying the motion model that was already presented to each particle to produce a posterior $\bar{\chi}_t$ set representing the state belief before incorporating the measurements. Then, given the measurement, each particle in $\bar{\chi}_t$ is assigned a weight indicating the likelihood of the measurement given the state of that particular particle. The result is a set of particles accompanied by a set of weights associated with each particle. Then, the new particle set χ_t is filled by drawing with replacement from $\bar{\chi}_t$. This drawing procedure follows a discrete probability distribution. In other words, the probability for a particular particle to be drawn is equal to the weight associated with it.

This basically constitutes for a resampling step in which the particles with high weights are duplicated, and the ones with small weights die. This re-sampling step is then repeated for each time step.

An implementatino of the resampler is given in the following figure:

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

The new particle is then constrcuted from the particles with the highest weights, and thus when the filter works fine, this set represents the state belief of the system. The Particle Filter algorithm, same as Bayes filter, requires an initial posterior $p(x_0$, which defines the type of problem that the filter is going to solve. If we assume a uniform posterior on the whole state space, it means that the robot doesn't know its position intially, and thus it means that the problem to be solved is a global localizarion problem. However, if the robot has an initial estimate of its initial position, the problem is simplified to a tracking problem.

C. Comments

There are a few arguments against Particle filters, they can be summarized in the following:

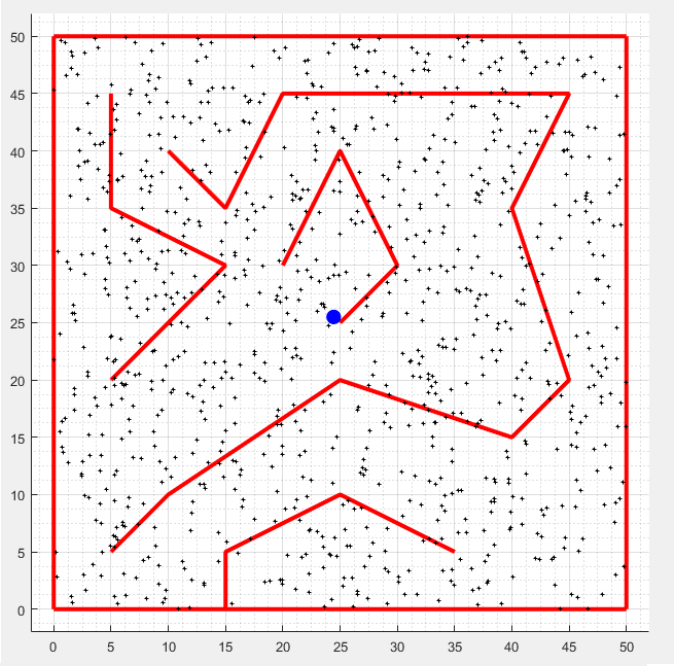
1) **Computational Cost**: Generally, in order to get robust and accurate estimates of the state, a big number

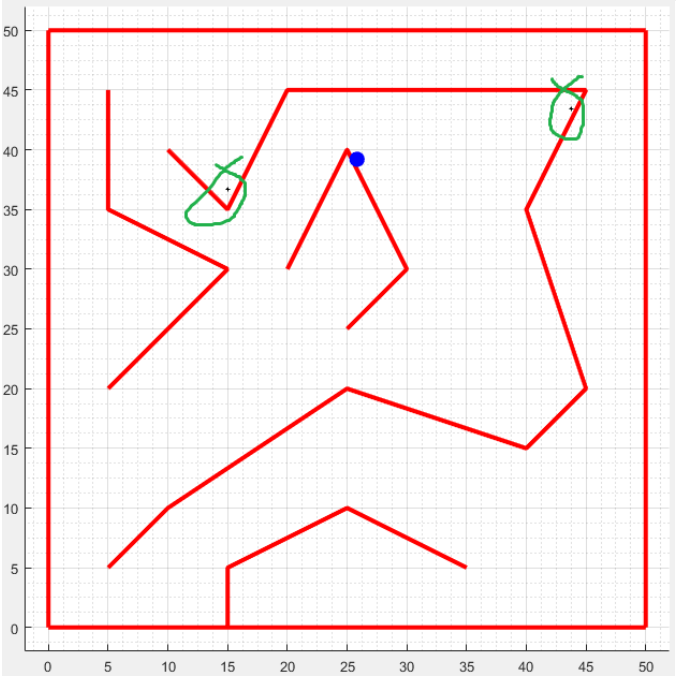
of particles must be used. But, a big number of particles means a huge computational burden. This is the first of the flaws of particle filters. Though this filter can handle a lot of situations that other filters can not handle, this comes at a cost of the computational power required to perform the associated computations.

2) **Approximation**: The underlying procedure of the particle filter is based on approximating the distributions by randomly drawn samples from them. This is an important assumption that simplified the associated computations, but it shouldn't be correct at all times. Generally, a big number of particles is required for this assumption to be retained while also retaining a good performance of the filter.
3) **Particle Deprivation**: As mentioned, as time progresses, particles with low weights eventually die, leaving only those with high weights which corresponds to high probability of being the correct state. That means that, if the filter converges to a solution that is not the correct state, there won't be a possible way of recovering from this because all the particles will be the ones that correspond to the false positive. A solution to this problem can be conditioning the resampling step on the variance of the weights of the particles such that re-sampling is performed only when the variance is big, so as to prevent the chances of the algorithm converging to a particular state erroneously.

VII. Demonstration

We aimed at demonstrating the Particle filter in action, by providing a simulation environment for a mobile robotics situation where a robot is navigating an environment and trying to localize itself. In order to do that, we harnessed the powers of the software MATLAB. Below is a figure showing the environment developed:
The black dots in the second figure correspond to the particles (the state belief samples). The blue dot is the mean of the particle set. The red lines are part of the map of the environment, and here they represent walls.
Initially, as it is obvious, the particles are randomly distributed accross the entire state space, this assumption is then refined over time by incorporating measurements into the state belief. After re-sampling step, the particles become more concentrated around the areas in the map in which it is likely to get incorporated set of measurements. This can be seen in the third figure
The implemented verstion however of the filter suffers from the typical disadvantages of the Particle filter, it is computationally cost. This is due to a lot od reasons, one of them being the use of a computationally costy algorithm known as the Ray-Casting algorithm to simulate sensor readings. Another, reason is the size of the particles required for an accurate localization. Furthermore, it seems that under the current implementation, a huge number of particles (¿ 5000) must be used to generate accurate localization accuracy. This needs further investigation to confirm the necessity of this bug number. However, the current implementation is extremely slow if the number of particles is bigger than five hundred.





References

[1] Thrun, Sebastian, Wolfram Burgard, and Dieter Fox. Probabilistic Robotics. Cambridge, Mass.: MIT Press, 2005. Print.

[2] Fernandez-Madrigal, Juan-Antonio and Jose Luis Blanco Claraco. Simultaneous Localization And Mapping For Mobile Robots. Hershey, Pa.: Information Science Reference, 2013. Print.

[3] Siegwart, Roland, Illah Reza Nourbakhsh, and Davide Scaramuzza. Introduction To Autonomous Mobile Robots. Cambridge, Mass.: MIT Press, 2011. Print.

[4] Nuchter, Andreas. 3D Robotic Mapping. Berlin: Springer-Verlag, 2009. Print.