Particle Filter based SLAM

Aswin P Ajayan

Indian Institute of Technology, Bombay

Jun 30, 2020

Overview

What is SLAM

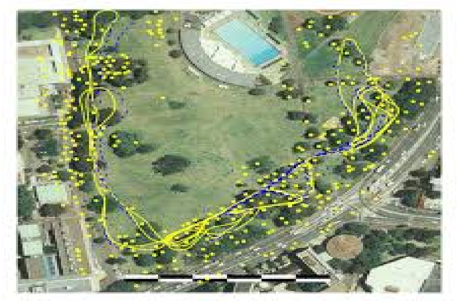
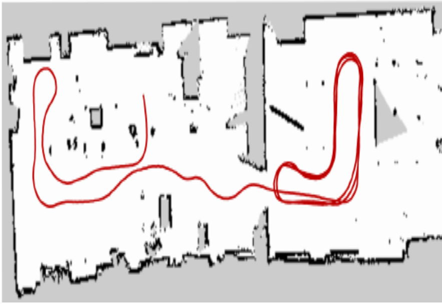
- Computing robot's poses and map environment simultaneously
- Localisation : estimating robots location
- Mapping : building a MAP
- Given
  - $u_{1:T} = \{u_1, u_2, u_3....u_T\}$, the control inputs
  - $z_{1:T} = \{z_1, z_2, z_3, ..., z_T\}$, observations
- Wanted
  - m, map of the environment
  - $x_{0:T} = \{x_0, x_1, x_2, ..., x_T\}$, robot location
  - $p(x_{0:T}, m | u_{0:T}, z_{1:T})$, the SLAM posterior

Objective

- To review the literature available on SLAM. To get acquainted with various techniques available for SLAM and examine some of them in depth. Simulation in reallife constraints using ROS.
  Types of SLAM techniques explored includes
- **Kalman Filter based approaches**
  - Extended Kalman Filter
  - Unscented Kalman Filter
  - Extended Information Filter
- **Particle Filter based approaches**
  - Fast SLAM -Rao Blackwellised Particle Filter
    - Augmented MCL

Types of SLAM

- Full SLAM vs online SLAM
  - Full SLAM $p(x_{0:T}, m | u_{1:t}, z_{1:t})$
  - Full SLAM $p(x_t, m | u_{1:t}, z_{1:t})$
- Feature Based SLAM vs Grid Based

- Active SLAM vs Passive SLAM

Recursive Bayesian Estimation

- **Markov assumption** : next state depends only on the present state

$$p(x_t|x_{0:t}, u_{1:t}) = p(x_t|x_{t-1}, ut)$$

- **Recursive Bayesian Estimation**

$$\begin{aligned} bel(x_t) &= p(x_t|z_{1:t}, u_{1:t}) \\ &= \eta p(z_t|x_t) \int_{x_{t-1}} p(x_t|x_{t-1}, u_t) * bel(x_{t-1}) dx_{t-1} \end{aligned}$$

we can split this into predict and update steps where
Predict Step

$$\overline{bel(x_t)} = \int_{x_{t-1}} p(x_t|x_{t-1}, u_t) * bel(x_{t-1}) dx_{t-1}$$

Update Step

$$bel(x_t) = \eta * p(z_t|x_t) * \overline{bel(x_t)}$$

Bayes filter recursive estimation is realised using kalman filter, Information Filter, particle filter etc.

Kalman Filter Family

Beliefs are represented in parametric form with mean vector μ_t and covariance matrix Σ_t . Closed form expressions are available for belief propagation under certain cases

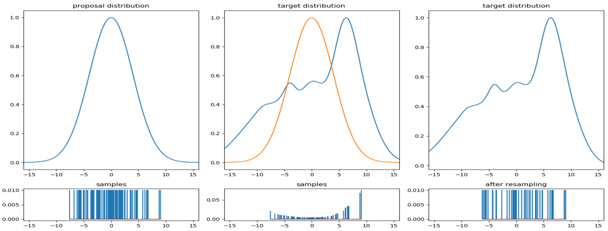
- **Kalman Filter :** Assumes linear models(state transistion and measurement) and gaussian posterior
- **Extended Kalman Filter :** Linearises the model and then applies normal KF
- **Unscented Kalman Filter :** Samples a gaussian posterior from the resulting non linear transformation
- **Information Filters :** Uses canonical representaion of Belief i.e. μ_t^{-1} and Σ_t^{-1}

Particle Filter

- Use particles to represent instead of parametric models.
- Can effectively model mutli modal distributions .
- Can Take care of the non linearities in the model
- Computationally intensive
- Based on importance sampling principal

Particle filter

Importance Sampling - Use to generate samples from an arbitrary distribution



steps

- Draw samples from a proposal distribution
- calculate the importance weight as $w_t^{[j]} = \frac{target(x_t^{[j]})}{proposal(x_t^{[j]})}$
- Resample based on weights

[4] https://github.com/aswinpajayan/seminar-related/scripts/importance sampling.py

Particle filter

- sampling from proposal distribution :

$$x_t^{[j]} \sim \pi(x_t | x_{t-1}, u_{t-1})$$

- Importance weighting :

$$w_t^{[j]} = \frac{target(x_t^{[j]})}{proposal(x_t^{[j]})} = p(z_t | x_t)$$

- Resampling : Draw sample i with probability $w_t^{[j]}$

[1] Source: Probabilistic Robotics, Sebastian Thrun

Rao-Blackwellisation for SLAM - Fast SLAM 1.0

- Particle filters are ideal for low dimensional states i.e. $||x_t||$ is small.
- for SLAM problem, Number of landmarks may be very large. So Particle filter cant be used directly
- Estimate the pose using particle filter and then compute map.

$$\begin{aligned} p(a, b) &= p(b|a).p(a) \\ \sim p(x_{0:t}, m_{1:M}|z_{1:t}, u_{1:t}) &= p(x_{0:t}|z_{1:t}, u_{1:t}).p(m_{1:M}|z_{1:t}, u_{1:t}) \\ &= p(x_{0:t}|z_{1:t}, u_{1:t})\Pi p(m_i|z_{1:t}, u_{1:t}) \end{aligned}$$

Now each particle respresents a path hypothesis and it has an assosicated map with it. Splitting $p(m_{1:M})$ into $\Pi p(m_i)$ reduces the computational complexity further, each of this is calculated by a 2x2 EKF.

RBPF particle structure

Each particle maintains M 2x2 EKF along with the 3 pose variables

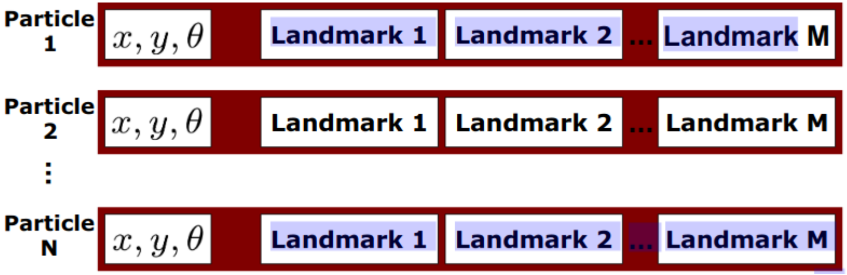
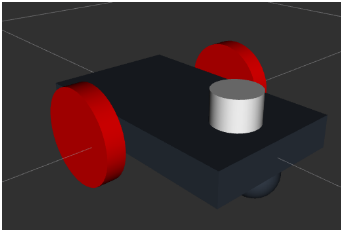


Figure: Particle structure in RBPF-slam

Implementing Particle filter localisation

- **Simulation Platform:** - ROS melodic Ubuntu 18.04
- **Robot:** - Custom robot using URDF
  - motion model libgazebo_ros_diff_drive.so
  - sensor model head_hokuyo_sensor
- **Visualisation:** - matplot lib interactive plots , python 2.7

Creating a custom robot



- *Universal Robot Description Format* provides an easy way to create a custom meshes for a robot. The motion model and sensor model are realised using the gazebo plugins. The robot was created using xacro files: which stands for XML Macros.
- sensor: Laser range finder with standard deviation 0.01 and gaussian model was used

[3] https://www.theconstructsim.com/ros-projects-exploring-ros-using-2-wheeled-robot-part-1/

Creating a motion model

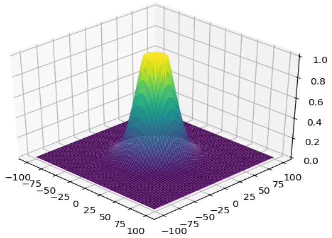
- Particle filter uses a motion model as the proposal distribution and sensor model for inferring importance weights
- A simple motion model was used :

$$x_t = x_{t-1} + v.cos(\theta_{t-1})$$
$$y_t = y_{t-1} + v.sin(\theta_{t-1})$$
$$\theta_t = \theta_{t-1} + w\delta_t$$

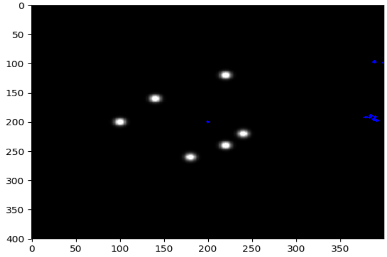
v, w are linear velocity, angular velocity.

Creating a sensor model

- Maximum likelihood field was chosen to be used as the sensor model
- The field was generated using a convolution with a kernel and point landmark map.



field.png



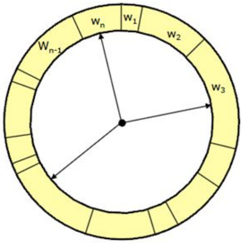
Resampling

Various Resampling approaches were tried out -

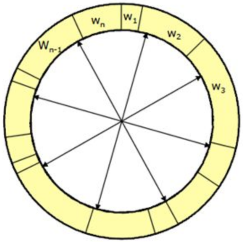
- **Fitness proportion sampling:** Used the default resampler available in numpy
  numpy.random.choice(particles, num=N, p=weights)
- **Roulette wheel selection :** Fitnesss values arranged as CDF around a wheel and a point is chosen at random
- **Stochastic Universal sampling :** Roulette wheel selection will be dominated by a few particles of highest fit. Some particles with high fitness values might be shadowed by the most fit members. This can result is global localisation failure. To avoid this SUS was tried. Though it reduces localisation failure, it can still result in localisation failure

Resampling cont..

Resampling



- Roulette wheel
- Binary search
- O(n log n)



- Stochastic universal sampling
- Low variance
- O(n)

- Stochastic universal sampling
- Systematic resampling
- Linear time complexity
- Easy to implement, low variance

Simulations

- Particle localisation
- Super imposed of ML field

ROS node architecture

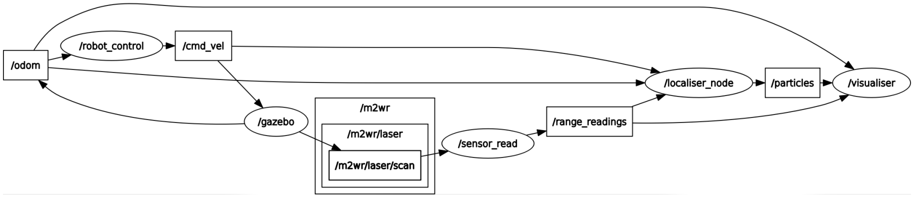


Figure: Ros graph generated using rqt_graph

Physics simulations with gazebo

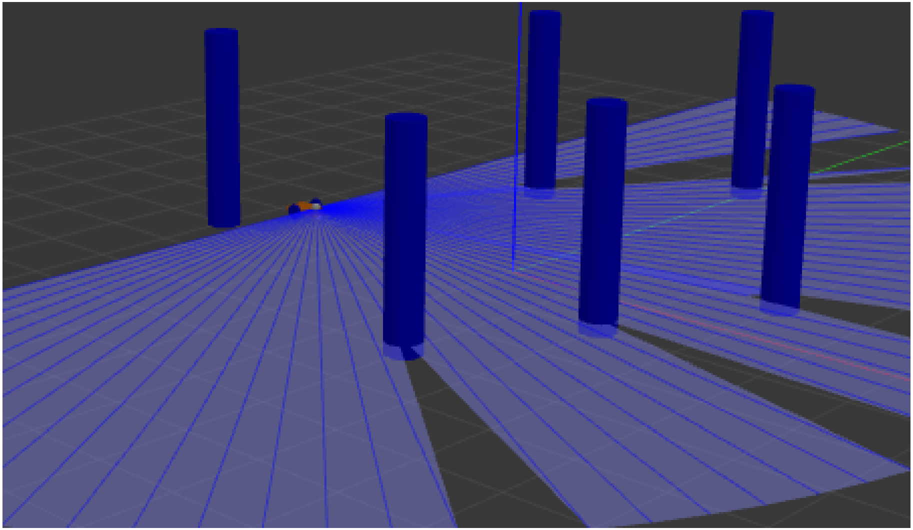


Figure: Robot kinematics and sensor physics is taken care of by gazebo

Suggested Improvements

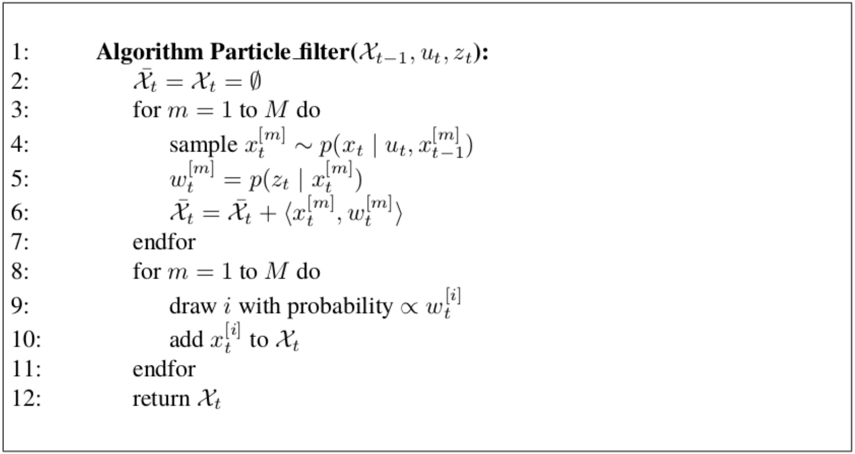
- Increasing the number of particles. Simualtions are quite heavy and the number of particles was limited by laptop hardware
- Better sampling strategy : Current resampling stratgy favours high fit particles to the extent that low fitness particles are often lost. Need a better resampling strategy to improve sampling.
- Augmented MCL to counter globalisation failures. Right now globalisation failure is countered by artificially inflating the sensor noise.
- Accurate kinematic model for the robot

Code Organisation

The entire code can be found at github link in branch touchups

- ROS uses python 2.7. caktin_make build system is used to build custom messages
- robot mesh and model descriptions are present in folder urdf
- launch file is used to initialise various ros nodes - present in launch folder (no_ui_spawn.launch)
- All scripts are found in scripts folder
    - scripts/robot_model.py contains kinematic and sensor models
    - scripts/read_sensor.py contains node to handle hokuyo laser range finder and correspondence algorithm
    - scripts/lv_resample.py contains various resampling algorithms used
    - scripts/my_MCL.py handles robot localisation

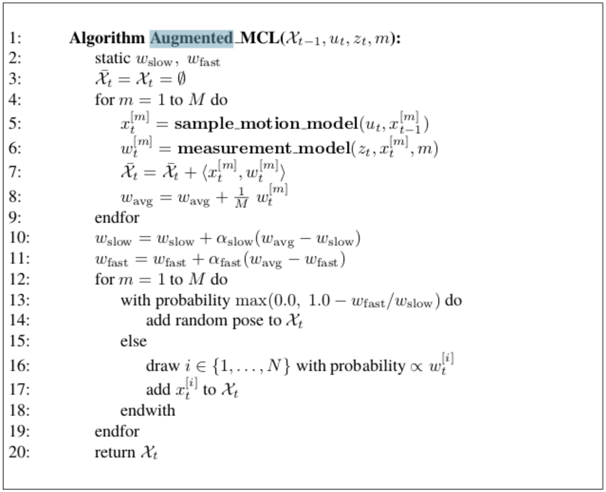
MCL localisation algorithm



```
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 $\mathcal{X}_t$
```

Table 4.3 The particle fi lter algorithm, a variant of the Bayes fi lter based on importance sampling.

Augmented MCL algorithm



```
1:  Algorithm Augmented_MCL($\mathcal{X}_{t-1}, u_t, z_t, m$):
2:      static $w_{\text{slow}}$, $w_{\text{fast}}$
3:      $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
4:      for $m = 1$ to $M$ do
5:          $x_t^{[m]} =$ sample_motion_model($u_t, x_{t-1}^{[m]}$)
6:          $w_t^{[m]} =$ measurement_model($z_t, x_t^{[m]}, m$)
7:          $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
8:          $w_{\text{avg}} = w_{\text{avg}} + \frac{1}{M}\, w_t^{[m]}$
9:      endfor
10:     $w_{\text{slow}} = w_{\text{slow}} + \alpha_{\text{slow}}(w_{\text{avg}} - w_{\text{slow}})$
11:     $w_{\text{fast}} = w_{\text{fast}} + \alpha_{\text{fast}}(w_{\text{avg}} - w_{\text{fast}})$
12:     for $m = 1$ to $M$ do
13:         with probability $\max(0.0,\ 1.0 - w_{\text{fast}}/w_{\text{slow}})$ do
14:             add random pose to $\mathcal{X}_t$
15:         else
16:             draw $i \in \{1, \ldots, N\}$ with probability $\propto w_t^{[i]}$
17:             add $x_t^{[i]}$ to $\mathcal{X}_t$
18:         endwith
19:     endfor
20:     return $\mathcal{X}_t$
```

Table 8.3 An adaptive variant of MCL that adds random samples. The number of random samples is determined by comparing the short-term with the long-term likelihood of sensor measurements.

Future Work

- Perfect Localisation using Augmented MCL and low variance resampling
- Move on to implementing RBPF SLAM in ROS
- Running ROS in client server mode offloading heavy computations to lab machine

References

[1] Probabilistic Robotics by Sebastian Thrun

[2] SLAM Lectures by Prof Cyrill Stachniss

[3] URDF tutorials by construct sim

[4] ROS wiki

[5] Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms By Hugh Durrant-Whyte

Thank You for your Attention