Robotic SLAM **Aswin P Ajayan : 183079032**

Contents

1 Introduction

Terms

- State estimation - find out the pose
- Localisation - pose w.r.to landmark or map
- Mapping
- navigation and motion planning - a star, wave front dijkstra

1.1 What is SLAM

Computing robot's poses and the map of the environment at the same time.
Localisation : estimating robots location
Mapping : building a MAP

Given

- Robots control inputs

$$u_{1:T} = \{u_1, u_2, u_3....u_T\}$$

- Observations

$$z_{1:T} = \{z_1, z_2, z_3, ..., z_T\}$$

Wanted

- Map of the environment

$$m$$

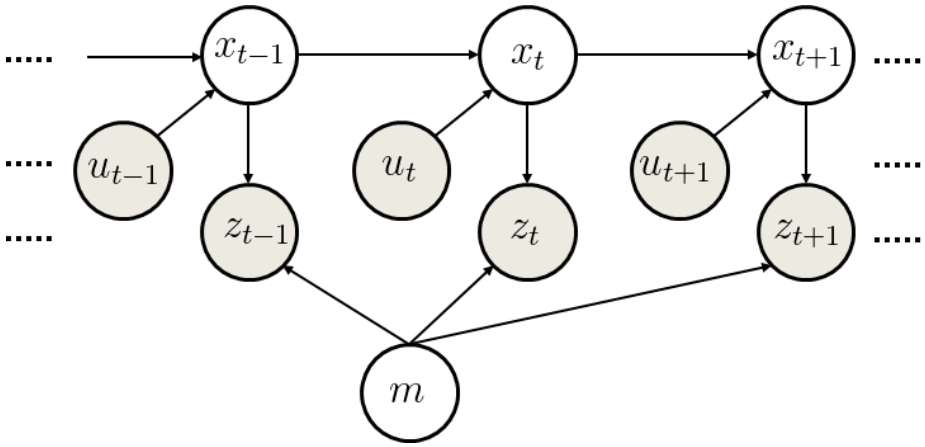
- path of the Robot

$$x_{0:T} = \{x_0, x_1, x_2, ..., x_T\}$$

Using the robots control inputs we can predict the position of the robot. From the observations $z_{1:T}$, we can calculate the position of the robot. Both the steps have some error associcated with it . Lets call the first one the model noise and second one the sensor noise. So we have to associate a probability with both of them. The error accumulates over time(even if the error in individual measurements is really small)

So in the probalistic terms our problem minimises to

$$p(x_{0:T}, m|z_{1:T}, u_{1:T})$$



1.2 Full Slam vs online SLAM

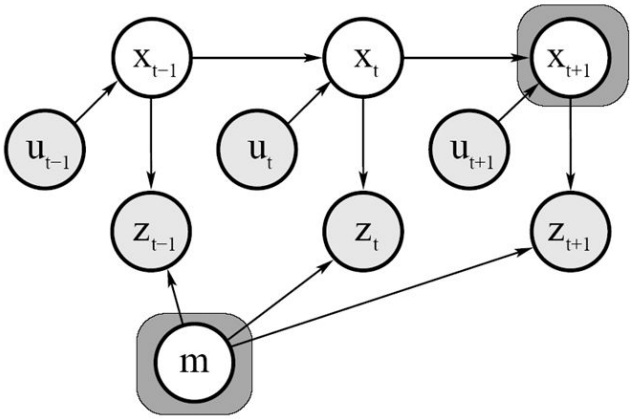
- Full SLAM estimates the entire path

$$p(x_{0:T}, m | z_{1:T}, u_{1:T})$$

- Online SLAM estimates only the most recent pose

$$p(x_t, m | z_{1:T}, u_{1:T})$$

Graphical Model of Online SLAM:



$$p(x_t, m \mid z_{1:t}, u_{1:t}) = \int \int \ldots \int p(x_{1:t}, m \mid z_{1:t}, u_{1:t})\, dx_1 dx_2 \ldots dx_{t-1}$$

10

1.3 Types of SLAM

occupancy maps created from lidars, sonars etc. - volumetric SLAM feature based approach - store features and localise based on that volumetric SLAM maybe better for navigation applications . Topological representations vs geometric representations. Static vs dynamic features. Active - robot decides the path so as to build a map vs passive slam - may follow a fixed path i.e. path not optimised for mapping/ exploration

2 Bayes Filter

2.1 State Estimation

Goal $p(x|z,u)$
Recursive Bayes Filter

$$\begin{aligned} bel(x_t) &= p(x_t|z_{1:t}, u_{1:t}) \\ &= \eta p(z_t|x_t, z_{1:t-1}, u_{1:t}) * p(x_t|z_{1:t-1}, u1:t) \\ &= \eta p(z_t|x_t) * p(x_t|z_{1:t-1}, u1:t) \\ &= \eta p(z_t|x_t) \int_{x_{t-1}} p(x_t|x_{t-1}, z_{1:t-1}, u_{1:t}) * p(x_{t-1}|z_{1:t-1}u_{1:t-1}) dx_{t-1} \\ &= \eta p(z_t|x_t) \int_{x_{t-1}} p(x_t|x_{t-1}, u_t) * bel(x_{t-1}) dx_{t-1} \end{aligned}$$

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 gives a framework for recursive state estimation using the above equations. The actual realisation may be kalman filtering , EKF or particle filter (Linear non linear motion models) (distributions) *Kalman Filter* - Gaussians , requires linear or linearised model *Particle filter* - Non-parametric , Arbitrary models

2.2 Probability motion models

$p(x_t|u_t, x_{t-1})$ we can model this in two ways

- odometry models - measurement of velocity (tends to be more accurate)
- velocity models - we know the input commands, but no measurement of vel

2.3 Model for laser scanners

scan z consisits of k beams $z_t \epsilon \Re^k$ i.e. $z_t = \{z_t^1, z_t^2, ...z_t^k\}$,
Assuming beams are independednt, then

$$p(z_t|x_t, m) = \prod_{i=1}^{k} p(z_t^i|x_t, m)$$

- Beam endpoint model (likelihood calculated as gaussian blur on occupancy map)
- Ray cast model (occlusion, sensor accuray, satuaration , random)
- model for range bearing sensors $z_t^i = (r_t^i, \phi_t^i)^T$

$$r_t^i = ||m - x|| + gaussian$$

$$\phi_t^i = \angle(m - x) - \theta + gaussian$$

3 Kalman filter Equations

$$\begin{aligned}
\bar{\mu_t} &= g(u_t, \mu_{t-1}) \\
\bar{\Sigma_t} &= G_t \Sigma_{t-1} G_t^T + R_t \\
K_t &= \bar{\Sigma_t} H_t^T (H_t \bar{\Sigma_t} H_t^T + Q_t)^{-1} \\
\mu_t &= \bar{\mu_t} + K_t(z_t - h(\bar{\mu_t})) \\
\Sigma_t &= (I - K_t H_t)\bar{\Sigma_t}
\end{aligned}$$

3.1 Summary

- Diverge for large non linearities
- Can deal only single modes
- Successful in medium scale scenes with good data associations
- Approximations exists to reduce the computational complexity

commonly used Datasets:

- Victoria Park Data sets - Trees are the landmark -Data assosciation -girth and height
- Tennis Court Dataset - for mapping precescion

4 Extended Kalman Filter vs Unscented Kalman Filter

EKF works by linearising the state transfer equations thus making sure that all the conditional and marginal distributions will remain gaussian. Gaussians are closed space. **UKF** uses the nonlinear state transmission equation, then tries to sample a gaussian distribution from the non-gaussian resulted from the non-linear operation.

4.1 Strategy for chosing sampling points and weights for UKF

$$
\begin{aligned}
X^{[0]} &= \mu \\
X^{[i]} &= \mu + (\sqrt{(n+\lambda)\Sigma})_i \; for \; i = 1, ..., n \\
X^{[i]} &= \mu - (\sqrt{(n+\lambda)\Sigma})_i \; for \; i = n+1, ..., 2n \\
w_m^{[0]} &= \frac{\lambda}{n+\lambda} \\
w_c^{[0]} &= w_m^{[0]} + (a - \alpha^2 + \beta) \\
w_m^{[i]} = w_c^{[i]} &= \frac{1}{2(n+\lambda)} \; for \; i = 1, ..., 2n
\end{aligned}
$$

where $\alpha \, \epsilon \, (0, 1]$; $k >= 0$; $\lambda = \alpha^2(n+k) - n$;
too small value of K will lead to UKF EKF . Too large - diverge
$\sqrt{\Sigma} = VD^{1/2}V^{-1}$, we are sampling the gaussian along the eigen vectors of the covariance matrix.

$$\bar{\mu}_t = \sum_{1=0}^{2n} w^{[i]} g(X^{[i]})$$

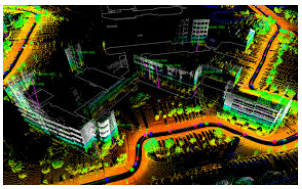
$$\bar{\Sigma}_t = \sum_{1=0}^{2n} w^{[i]} (g(X^{[i]}) - \bar{\mu}_t)(g(X^{[i]}) - \bar{\mu}_t)^T + R_t$$

5 Grid Maps

SLAM problems can make one of the types of maps

- **Feature Based Maps:** Maps is repsented by a few lanndmarks. Advantage of this type of maps is the space time complexity can be smaller compared to Volumetric Maps. On the downside, we have to implement a feature detector - i.e. observations are not directly used in this method.
- **Volumetric Maps:** Volumetric Maps on the other hand uses all observations and represent the map using a 3D(lidar point cloud) or 2D(range

Figure 1: Feature map overlayed on victoria park



(a) 3D grid map from lidar point cloud

(b) Occupancy Grid Map

Figure 2: Volumetric Maps

sensor) grid. Occupancy is often shown with a black pixel and free areas are white. Unseen areas are marked with grey. Feature based maps are used in kalman filter implementations, Particle filter based approaches use Grid mapping and sometimes feature based maps.

5.1 Representation - Occupancy Grid Map

The **Occupancy grid map** partitions the space into finitely many grid cells. $m = \{m_i\} \ni$ every m_i forms a binary random variable . Size of grid cell is a free parameter which decides the trade of between computational complexity and approximation Error.

- $p(m_i) \to 1 \Rightarrow$ high confidence that grid cell is occupied
- $p(m_i) \to 0 \Rightarrow$ high confidence that grid cell is free
- $p(m_i) \to 0.5 \Rightarrow$ no information about occupancy of grid cell

Problem statement: calculate the posterior $p(m|z_{1:t}, x_{1:t})$

Assumptions:

- A grid is either fully occupied or fully free.
- Conditional independence of grid cells i.e. $p(m) = \prod p(m_i)$

5.2 Algorithm

Log odds is defined as

$$l_o = log \frac{p(m_i)}{1 - p(m_i)}$$

$$p(m_i) = 1 - \frac{1}{1 + exp(l_o)}$$

We are hunting for the posterior $p(m_i|z_{1:t}, x_{1:t})$. If we express this posterior in odds form , we have

$$o_{t,i} = \frac{p(m_i|z_{1:t}, x1:t)}{p(\neg m_i|z_{1:t}, x1:t)}$$

$$= \frac{p(m_i|z_t, x_t)}{p(\neg m_i|z_t, x_t)} \cdot \frac{p(m_i|z_{1:t-1}, x_{1:t-1})}{p(\neg m_i|z_{1:t-1}, x_{1:t-1})} \cdot \frac{p(\neg m_i)}{p(m_i)}$$

In log odds form

$$l_{t,i} = inverse\_sensor\_model + l_{t-1,i} - l_o$$

In each step we will update only the grids which are in the perception range of the sensor at that particular time instant

5.3 Inverse Sensor Model

inverse_sensor_model: is the implementation of inverse measurement model $p(m_i|z_t, x_t)$ in log odds form.

5.4 Scan Matching

"The assumption that poses is known is fundamentally flawed". If we use the pose from raw odometry data the grid map which we recieve is not generally usable.

"**Scan-matching** tries to incrementally align two scans or a map to scan without revisiting the past/map. Some common methods used for scan matching - Iterative Closest point match, RANSAC for outlier rejection . A standard cost function is given below

$$x_t^* = \arg\max_{x_t} \{p(z_t|x_t, m_{t-1})p(x_t|u_{t-1}, x_{t-1}^*)\}$$

5.5 Summary

- Occupancy Grid maps are used in conjuction with Particle filter based SLAM, feature based maps are used in kalman filter based SLAM.
- Occupancy Grid map by itself is not a SLAM algorithm. It assumes known poses.
- Scan matching is used to build usable maps .

6 Fast SLAM

6.1 Review of Particle filters

- Non-parametric recursive Bayes filter
- uses samples to represent belief distribution

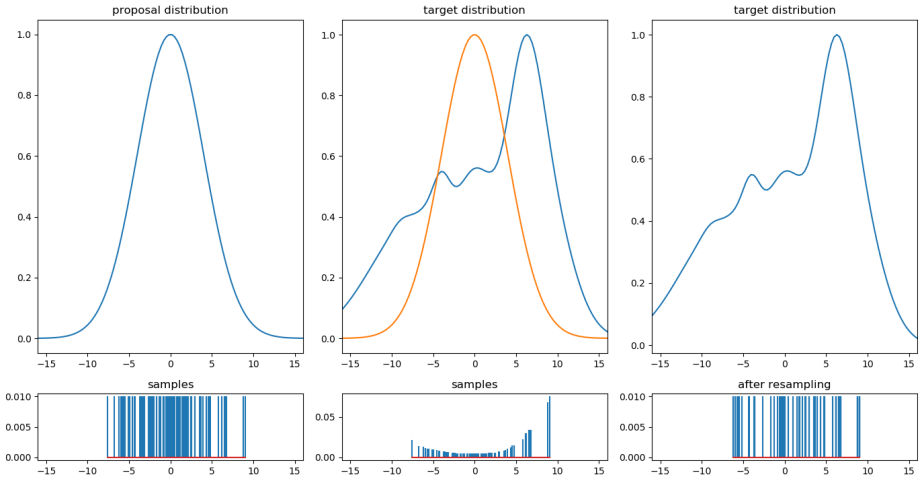


Figure 3: Generating samples from arbitrary distribution using importance sampling

- as opposed to kalman filter family it can model multi modal distributions.
- Uses importance sampling principle to draw samples from arbitrary distributions
  - 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 $w_t^{[j]}$

6.2 Rao-Blackwellization for SLAM

SLAM problem is represented as finding the posterior

$$p(x_{1:t}, m_{1,x}, m_{1,y}, \ldots, m_{M,x}, m_{M,y})$$

Particle filters are ideal for low dimensional problems . Hence cant be directly used to solve the SLAM problem(dimension 2N + 3). One approach is to estimate the pose using particle filter and then computing map . Mathematically

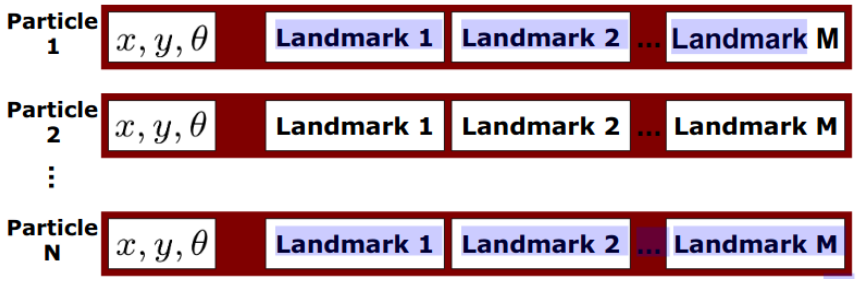


Figure 4: Particle structure in RBPF-slam

this factorisation can be expressed as

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

Effectively we have split it into a path posterior and a map posterior. 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.

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

6.3 Algorithm

from slides by Prof Syrill Stachniss
FastSLAM1.0_known_correspondence(z_t, c_t, u_t, X_{t-1}):
*f*or k = 1 to N do — loop through N particles

Let $\langle x_{t-1}^{[k]}, \langle \mu_{1,t-1}^{[k]}, \Sigma_{1,t-1}^{[k]} \rangle \dots \langle \mu_{M,t-1}^{[k]}, \Sigma_{M,t-1}^{[k]} \rangle \rangle$ be a particle in X_{t-1}

$x_t^{[k]} \sim p(x_t|x_{t-1}, u_t)$ — sample pose

$j = c_t$ — observed feature with correspondence
if feature j never seen before:
$\mu_{j,t}^{[k]} = h^{-1}(z_t, x_t^{[k]})$ — initialize mean
$H = h'(\mu_{j,t}^{[k]}, x_t^{[k]})$ — calculate Jacobian
$\Sigma_{j,t}^{[k]} = H^{-1} Q_t (H^{-1})^T$ — initialize covariance
$w^{[}k] = p_0$ — default importance weight
else

$\hat{z}^{[k]} = h(\mu^{[k]}_{j,t-1}, x^{[k]}_t)$ measurement prediction
$H = h^{'}(\mu^{[k]}_{j,t-1}, x^{[k]}_t)$ calculate Jacobian
$Q = H\Sigma_{j,t-1}H^T + Q_t$ measurement Covariance
$K = \Sigma^{[k]}_{j,t-1}H^T Q^{-1}$ calculate Kalman gain
$\mu^{[k]}_{j,t} = \mu^{[k]}_{j,t-1} + K(z_t - \hat{z}^{[k]})$ update mean
$\Sigma^{[k]}_{j,t} = (I - KH)\Sigma^{[k]}_{j,t-1}$ update covariance
$w^{[k]} = |2\pi Q|^{-\frac{1}{2}} exp\{-\frac{1}{2}(z_t - \hat{z}^{[k]})^T Q^{-1}(z_t - \hat{z}^{[k]})\}$

endif
for all unobserved features $j^{'}$ do:
$\langle \mu^{[k]}_{j,t}, \Sigma^{[k]}_{j,t} \rangle = \langle \mu^{[k]}_{j,t-1}, \Sigma^{[k]}_{j,t-1} \rangle$ leave unchanged
end for
$X_t = resample \langle x^{[k]}_{t-1}, \langle \mu^{[k]}_{1,t-1}, \Sigma^{[k]}_{1,t-1} \rangle \ldots, w^{[k]} \rangle$

7 Implementation in ROS (Particle Filter Localisation)

Robotic Operating system provides a framework for simulating and testing robotic algorithms and real world interaction. Platform used for simulation

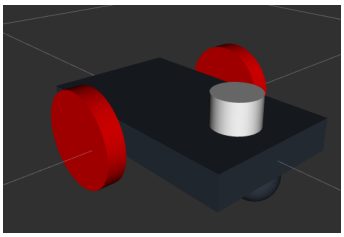
- ROS melodic
- Ubuntu 18.04 (intel i3 3130 with radeon graphics)
- ros gazebo for simulating physics
- python 2.7

Out of the box, ROS still uses python2.7.

7.1 Creating a custom robot

Various robots available in default in ROS have two major issues. Platform portability, and unneseccary sensors which can bog down the system performance. To avoid this issues, a simple custom robot was made using UDRF - Universal Robot Description Framework. A good tutorial on making a small 2 wheeled robot is available at the constructsim website

- *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



7.2 Sensor Model

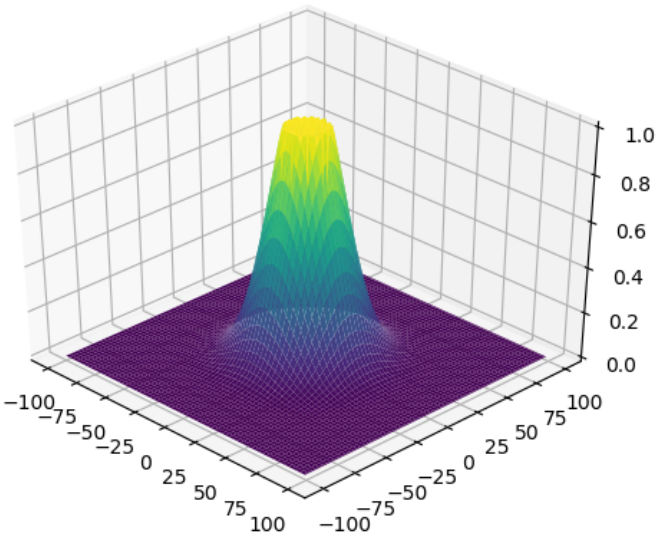
Maximum Likelihood field was chosen as the sensor model. Field was precomputed. Field was computed as a convolution between kernel and point landmarks represented a impulses.
World was created with cylinders as landmarks. Each cylinder had a radius of 0.2m. So a kernel was created taking into account the shape and the radius

```
def get_kernel(k_size, radius, std):
    """function to generate kernel to produce
    discretised likelihood grid we can convolve
    this kernel with landmarks(ground_truth /
    observed to produce discretised likelihood field)

    :k_size : kernel size (-k_size:k_size) (scaled)
    :radius : radius of landmark
    :std    : standard deviation of meaurement model
    returns: flat top gaussian kernel  """

    cov = np.diag([std ** 2, std ** 2])
    mean = np.array([0, 0], dtype=np.float)
    x_axis = np.arange(-k_size, k_size)
    y_axis = np.arange(-k_size, k_size)
    x_values, y_values = np.meshgrid(x_axis, y_axis)
    grid = np.empty(x_values.shape + (2,))
    grid[:, :, 0] = x_values
    grid[:, :, 1] = y_values
    cov_inv = np.linalg.inv(cov)
    z_norm = np.einsum('...k,kl,...l->...', grid - mean, cov_inv, grid - mean)
    lim = radius ** 2 / (2 * std * std)
    z_norm[z_norm < lim] = 0
    z_norm[z_norm > lim] = z_norm[z_norm > lim] - lim
    kernel = np.exp(-0.5*z_norm)
    kernel = kernel / np.amax(kernel)
    return kernel
```

*einstein summation can speed up the process. Speed is not really much of a concern as kernel calculation is a one time job.



Kernel is convolved with point landmark map to get the ML field

7.3 Resampling approaches

Various Resampling approaches were tried out -

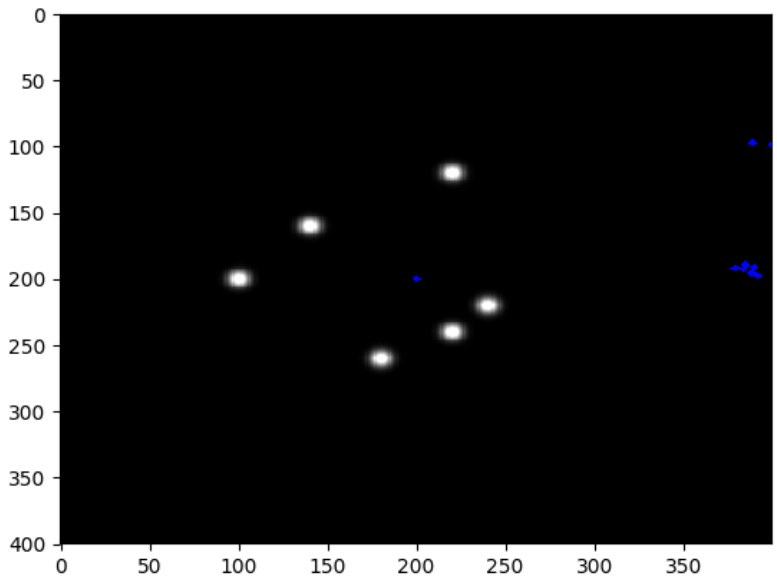
- **Fitness proportion sampling:** Used the default resampler available in numpy
  numpy.random.choice(particles, num=N, p=weights)

```
indeces = np.random.choice(indeces, NUM, 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

```
def lv_sampler(weights):
    """
    function to perform low variance sampling
```

field.png




```
    taken from S thruns Book
    :weights: importance weights of corresponding particles
    :returns: indeces
    """
    indeces = []
    NUM = np.float(len(weights))
    r = np.random.uniform(0, 1 / NUM)
    c = weights[0]
    i = 0
    for m in np.arange(NUM):
        u = r + (m - 1) * (1 / NUM)
        while(u > c):
            i += 1
            c += weights[i]
        indeces.append(i)
    return np.array(indeces, dtype='int')
```

7.4 simulation results

Simulation results obtained where just satisfactory. Robot was initialised with some random samples and was able to localise under motion noise and a rela-

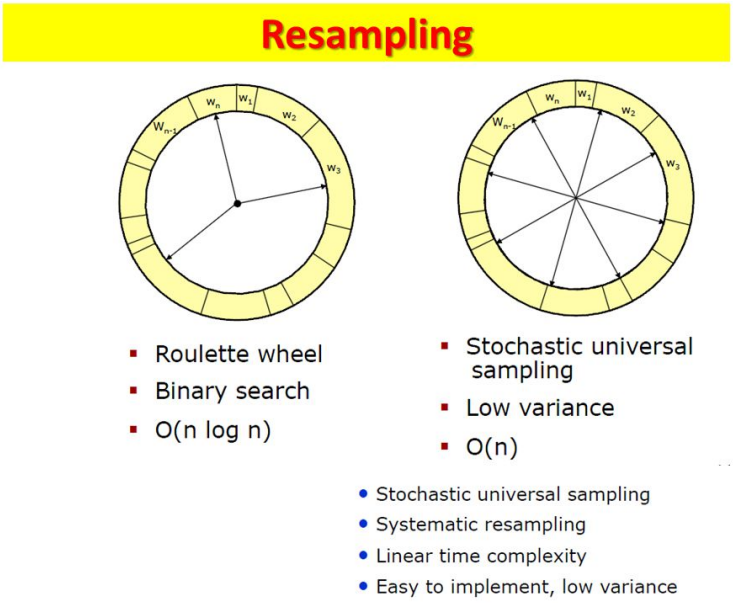


Figure 5: Particle structure in RBPF-slam

tively bad motion model.

- Particle localisation
- Super imposed of ML field

The main point of concern is that the fittest particle is dominating. This might lead to global localisation failure. A condition in which all the particles corresponds to a single wrong state. This is handled to a certain extend by artificially adding noise to the particles, but better approaches are available in literature

7.5 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

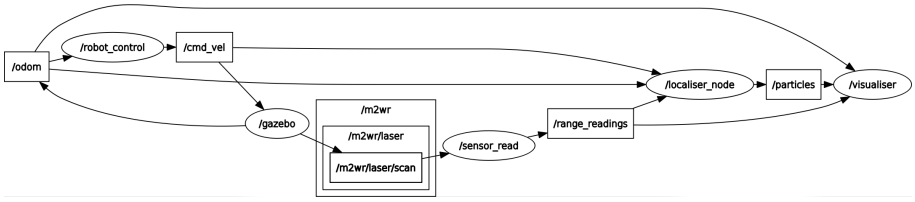


Figure 6: Ros graph generated using rqt_graph

7.6 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

8 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

9 References and Links

- Probabilistic Robotics by Sebastian Thrun
- SLAM Lectures by Prof Cyrill Stachniss
- URDF tutorials by construct sim

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.

1

- ROS wiki
- Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms By Hugh Durrant-Whyte
- Webpage from Prof. Syrill Stachniss can be found here

- A collection of various robotics algorithms pythonrobotics
- github repository `https://github.com/aswinpajayan/seminar-related.git`

10 Appendix

10.1 PF algorithm

10.2 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: $x_t^{[m]} = \textbf{sample\_motion\_model}(u_t, x_{t-1}^{[m]})$
6: $w_t^{[m]} = \textbf{measurement\_model}(z_t, x_t^{[m]}, m)$
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.

1