Particle Filter: Vishwajeet Karmarkar

1)

In the dataset there are two control signals given to the robot, linear velocity along the robot's x-axis and angular velocity about its z axis (anti-clockwise being positive). Given the available data the robot is assumed to be a point mass moving with the mentioned velocities. Thus the motion model takes these velocities as its input.

The motion model is built on the Markov assumption. Under this assumption the likelihood of the future state depends on the present state and not past states [1]. The model assumes that the robot traces circular paths for every control step, with the centre of the circle (Instantaneous centre of circle, ICC) changing as the robot transitions in space [2]. If there is no angular velocity, then the robot moves in a circle centred at infinity [2].

Motion Model:

Step 1: Finding ICC location.

Once a control signal is received, for time dT, it is assumed that the values of V and W do not change. Thus the instantaneous ICC is located at a radius, $R = V/\omega$ for the time period dT.

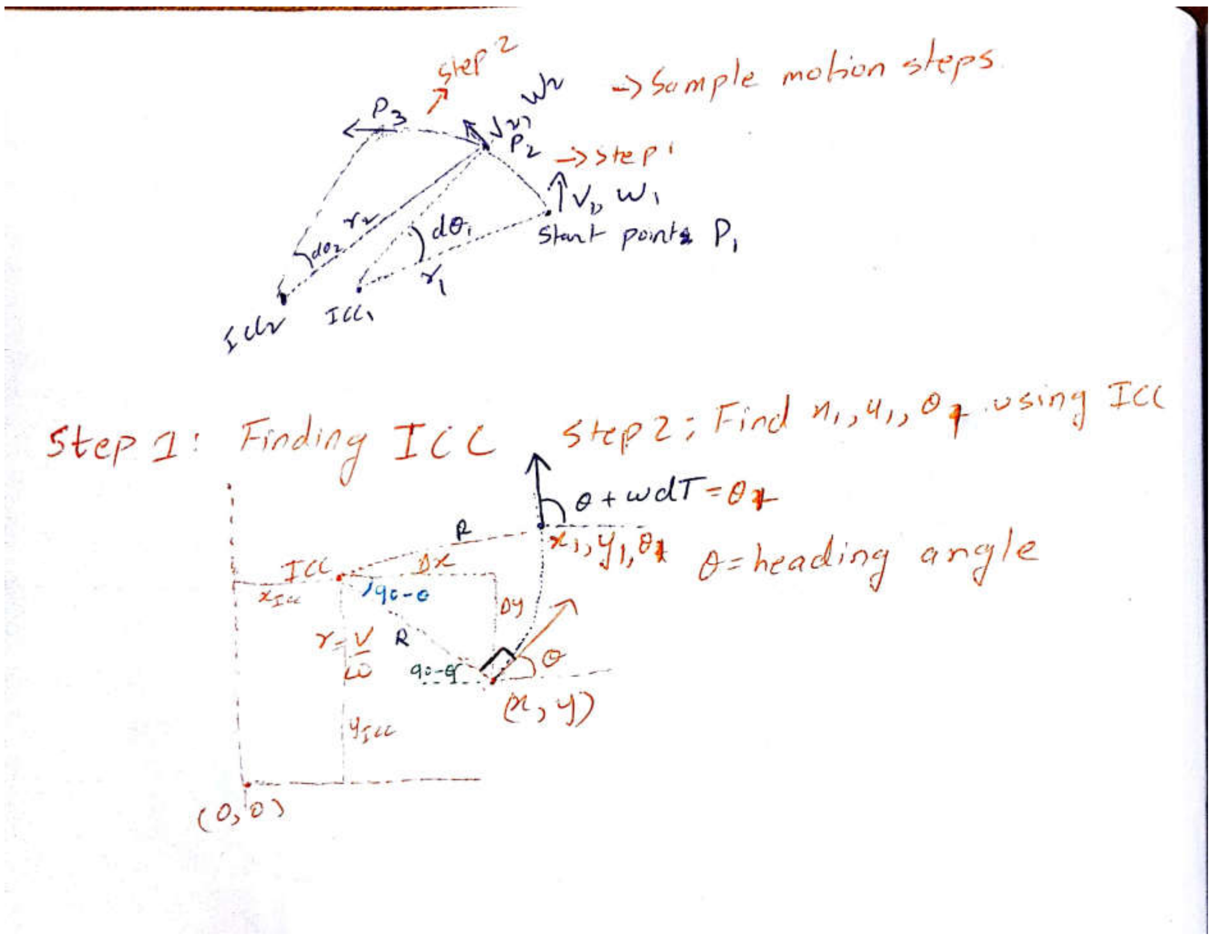


Figure 1: Plot showing proposed motion model derivation, with robot at (x, y, θ)(prior) going to $(x1, y1, \theta 1)$

From the figure drawn above, it can be seen that

$$x_{Icc} = x - \Delta x, y_{Icc} = y + \Delta y.$$

Applying simple geometry as above, it can be seen that

$$\Delta x = R \times sin(\theta), \Delta y = R \times cos(\theta)$$

$$\because x_{Icc} = x - R \times sin(\theta), y_{Icc} = y + R \times cos(\theta)$$

The robot now moves in a circle centred at ICC with the same radius R for time dT. Its new heading thus is $\theta 1 = \theta + dt \times \omega$.

The new position can again be calculated from the diagram as

$$X1 = X_{lcc} + R \times sin(\theta + dt \times \omega)\,, Y1 = Y_{lcc} - R \times cos(\theta + dt \times \omega).$$

Thus for every control signal for a short time delta T, the robot moves sequentially in small arcs. To check for the validity of the model a small test can be run. If the robot receives constant linear and angular acceleration of 1 m/sec and 1 rad/sec indefinitely, it should keep tracing a circle with a radius of 1. On testing this input to the model we get the following output.

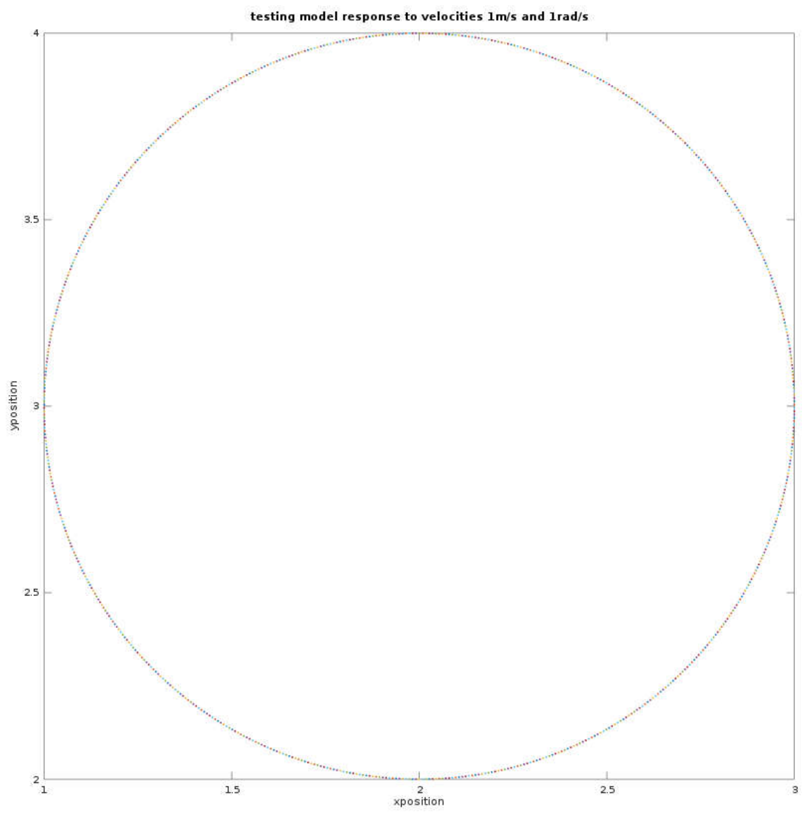


Figure 2

The model traces a circle with a radius of 1, thus it behaves as expected.
In reality the actual velocity will be different due to a number of real world constraints such as noise, wheel misalignment and so on. Assuming these noise values are Gaussian in nature, before each control step a small Gaussian noise is added to the control signal.

$$\therefore V = V_{control} + V_{noise}$$
$$\therefore \omega = \omega_{control} + \omega_{noise}$$

Another assumption that the heading of the robot will only increase by $\omega \times dT$, will also not hold and hence a correction factor (also Gaussian) is added to the final orientation [2].

The model is not linear in its input. A linear model should satisfy the format: $y = a \times x + c$, where a and c are constants. However the model above uses sines and cosines which are in nature non-linear.

2)

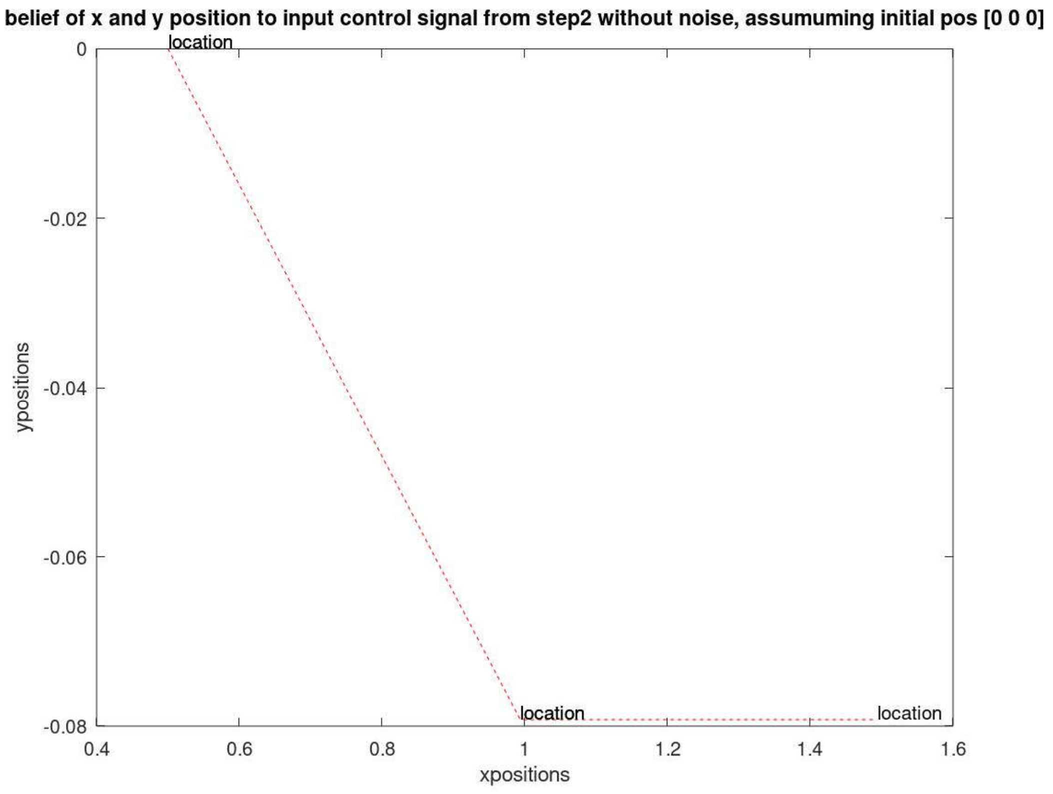


Figure 3

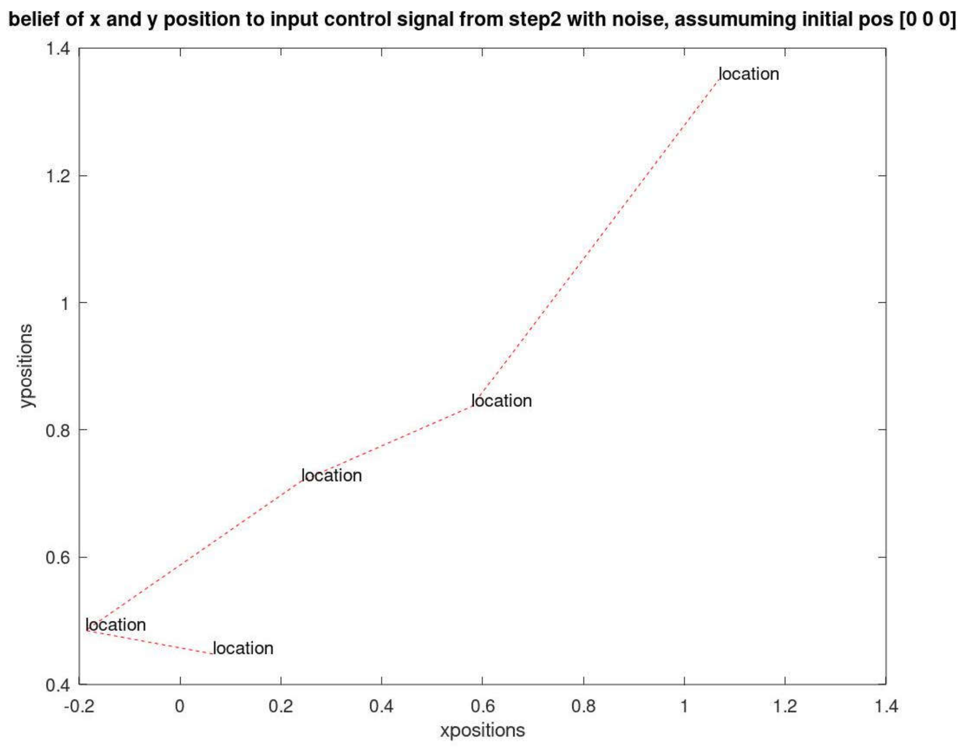


Figure 4

The above two plots show the response of the motion model to given control commands. The first figure assumes an ideal case where no noise is incorporated, while the second incorporates a noise to simulate real world behaviour. It can be easily seen using simple hand calculations that the no-noise position is accurate, however once noise is introduced even small control steps can have huge deviation.

3)

The plot below shows the actual state, plotted using the ground-truth data and the path traced by the motion model. As is seen it is very clear that real world parameters throw the motion model tracking off at a very early stage. This can be seen even with hand calculations, at the initial stage when only a linear velocity is applied.

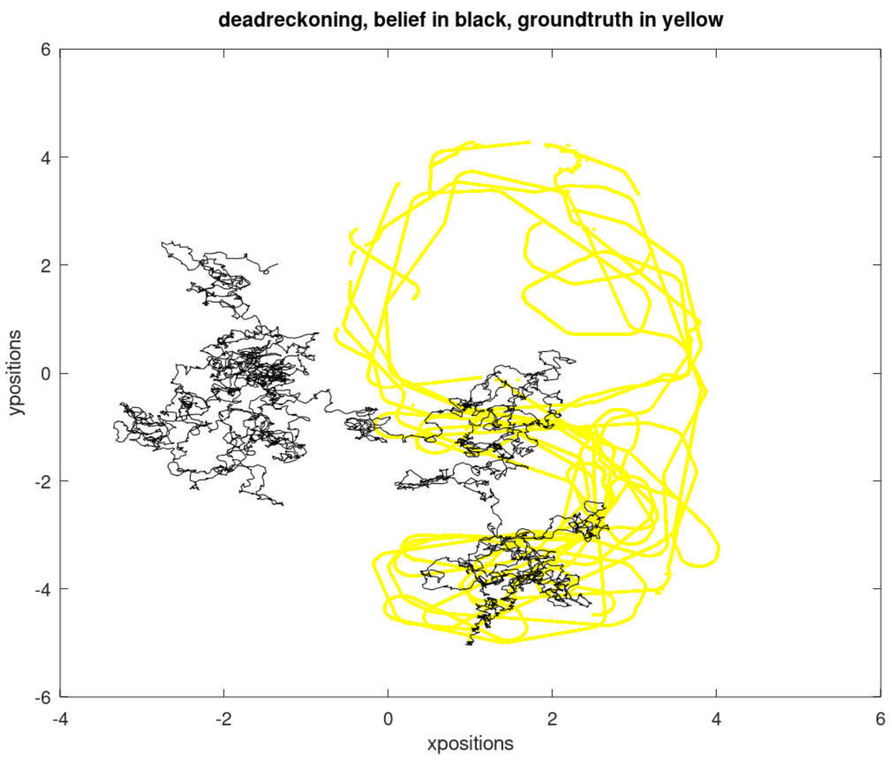


Figure 5

4)
The particle filter comes under the general category of Bayes Filters. The Bayes filter is a recursive filter which estimates the posterior belief of a robot's state from the prior belief, control information and sensor data[2]. In general all Bayes filters have two main steps:

1. Control update or prediction step: In this step the filter calculates $\overline{bel}$, or the robots predicted belief.
2. Measurement update or correction step: In this step the filter takes into account sensor measurement and corrects the predicted belief, to give the final posterior belief

Particle filter is a non-parametric type filter. It does not have a fixed functional form [2]. Instead it picks a number of estimates from the posterior, each of which is called a 'particle'. These particles keep propagating over time, tracking the posterior belief.

Particle filter operation:
In a generalized fashion, the particle filter works slightly different than the steps mentioned in the general Bayes Filter. There is no estimation of $\overline{bel}$. Instead the algorithm calculates the posterior belief bel from the motion model, and then samples state from that posterior belief. The number of samples is determined by the amount of particles chosen at the start.
The PF algorithm:

```
1. The first step is deciding the number of particles(N)
2. Create a distribution of particles around the known initial state
   For j= 1 : T
           For i = 1 to N
               1. Feed the $i^{th}$ particle to the motion model. The motion model outputs posterior belief
```

2. *Feed the i^th bel to the measurement model. Measurement model calculates sensor output for i^th bel state. With a known sensor measurement, the probability of sensor reading, to the $\overline{bel}$ state is calculated*

$$P(N) = p(z_t \mid x_t^N)$$

3. *Normalizing all values, p(N), the normalized weight of each particle is calculated*

$$w_t^N = {P(N)}\Big/{\sum P(N)}$$

```
Re-sampling step:
    For k=1:N
        1. Pick a particle from the total list of particles. Particles with higher weights have
           better chance of getting picked up. This step is repeated for the N samples.
    End
In the last step the actual particle position can be assumed to be at the mean of all particles/
```

$$Bel = \frac{\sum_1^N bel(N)}{N}$$

End

The above algorithm recursively calculates the posterior belief.
Since the particle filter is non-parametric, it is easier to model systems which are complicated or non-Gaussian in behaviour. This also applies for systems with high number of dimensions, in terms of state variables and sensor dimensions [2].
Assumptions:

1) The particle filter assumes a Markovian world, where no knowledge of the past states is needed, and only the prior state is sufficient to determine the future state. [2]
2) Another assumption that has been done in this project is that sensor values are independent to each other, in both their working and noise parameters.

$$P(Sense_1 \cap sense_2) = P(Sense_1) \times P(Sense_2)$$
$$P(Sense_1 | Sense_2) = P(Sense_1) \quad P(Sense_2 | Sense_1) = P(Sense_2$$

5)

The measurement model makes use of known features of the robots working space. In our data there are 15 landmarks, whose state (position) is known with considerable accuracy. These landmarks are identified with distinct features (signatures), which are barcodes with a unique number. The feature list contains the following

$$f(z) = \{f_1, f_2, \dots f_n\} = \begin{matrix} x_1 & x_2 & & x_n \\ y_1, & y_2, & \dots & y_n \\ s_1 & s_2 & & s_n \end{matrix}$$

At each time step when the robot's sensor identifies a landmark with a known signature S, it calculates the range R and heading theta for that signature.

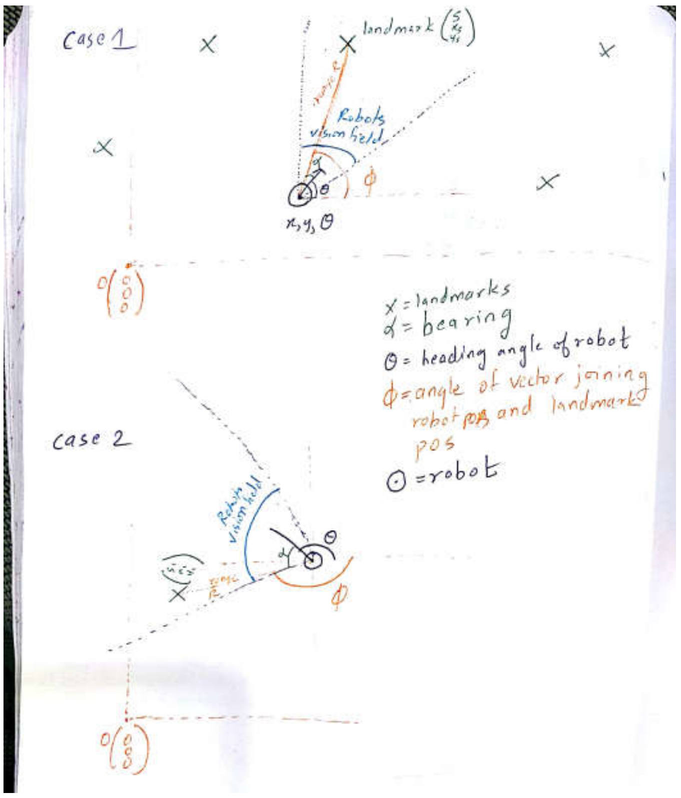


Figure 6: Shows robots field of vision, robot heading angle and landmark bearing angle wrt the robot

Measurement model:
For each particle the model takes as input the robots prior belief as input, and based on the landmark signature it identifies, calculates the range and heading to that landmark. Please refer to the figure above.

The range R is found using the distance rule.

$$R = \sqrt[2]{((x_{landmark} - x_{robot})^2 + (y_{landmark} - y_{robot})^2)}$$

To find the bearing of the landmark with respect to the robots heading, it is important to note that the robot follows the right hand rule. Thus for the robot's z axis, anti-clockwise rotation is positive. For the bearing calculation, we convert the landmark coordinates from the ground frame to the robots reference frame. We do this by:

$$x_{landmark_{robotframe}} = diff_x = x_{landmark} - x$$
$$y_{landmark_{robotframe}} = diff_y = y_{landmark} - y$$

To find the angle of the line joining the robot co-ordinates with the landmark coordinates, we can use the atan2 function. Atan2 gives the value of the robot in a 0: π , and 0: $-\pi$ radians range. Thus if a line is in the 3rd quadrant, the value will lie in $\frac{-\pi}{2}$ to -π, for the 1st quadrant between 0 to $\frac{\pi}{2}$ and so on. Thus to find the bearing w.r.t positive rotation of the robot, we change the angle values as follows. Refer the figure for visual illustration.

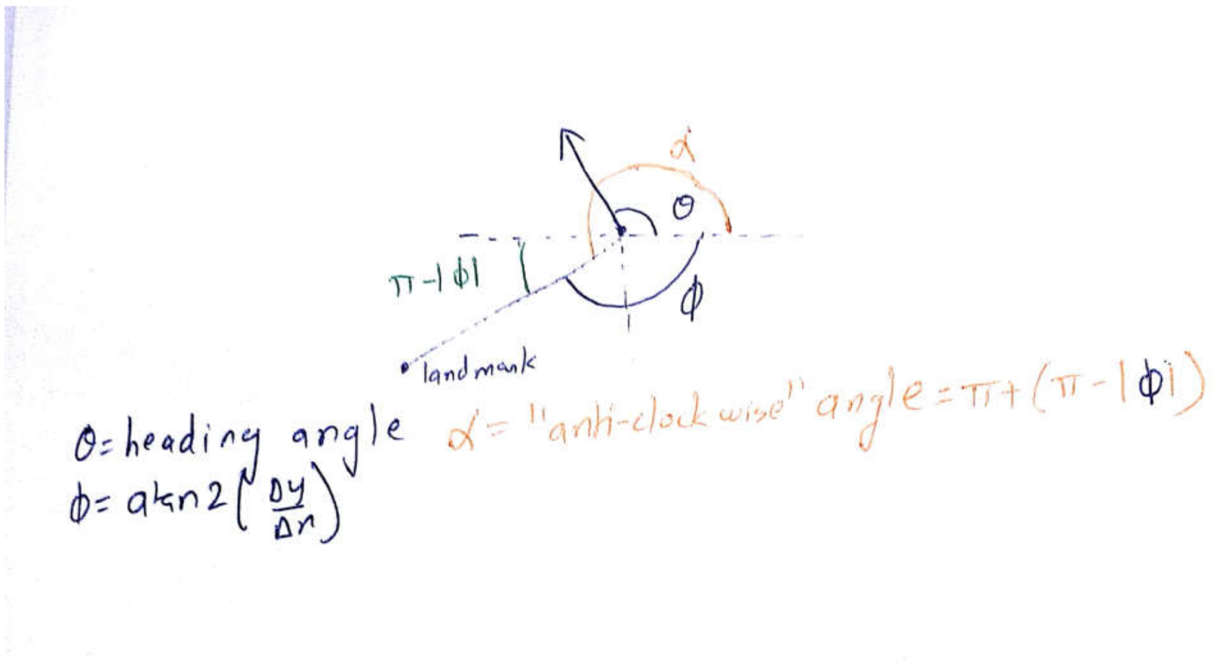


Figure 7: finding the 'anti-clockwise' angle

If $(\theta > 0)$

$$\theta_{anti-clockwise} = \theta$$

else

$$\theta_{anti-\ clockwise} = \pi + (\pi - abs(\theta))$$

endif

Once we have the 'anti-clockwise' angles we can calculate the bearing angle as follows

$$bearing\ = heading\_angle_{anticlockwise} - \theta_{anticlockwise}$$

After finding these sensor readings for each particle, we effectively found

$$sensor\ readings \mid predicted\ state$$

At that particular time step, since we know what the actual sensor readings are, and what the sensor readings are given the predicted state, we have enough information to calculate:

$$P(z_t \mid x_t^m)$$

This probability is calculated by using

$$P = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

Here x is the value from the measurement model, μ is the actual sensor reading and σ is the variance of measurement.I also make an assumption at this point. Because we have multiple sensor readings, we assume that the probabilities of those sensor readings, given a particular state are independent of each other. Thus using conditional independence rule from probabilities we can claim:

$$P(z_{total} \mid x_t^m) = P\left(z_{sensor_1} \middle| x_t^m\right) \times P\left(z_{sensor_2} \middle| x_t^m\right) \times P\left(z_{sensor_n} \middle| x_t^m\right)$$

$$\therefore\ P(z_{total} \mid x_t^m) = P\left(z_{sensor_{range}} \middle| x_t^m\right) \times P\left(z_{sensor_{heading}} \middle| x_t^m\right)$$

Once we have $P(z_{total} \mid x_t^m)$, for all particles N, we can normalize their values to get the weights of each particle. These normalized values are then fed to the re-sampling part of the code, where the particles are selected, with the likeliness of getting picked higher for particles with a larger weight.

Few points on the measurement model:

1) Since there is a time difference between control signals and the sensor measurement, sensor measurement at a time closest to the control signal time, but lesser than that time is taken(nearest past measurement).
2) While calculating the probability

$$P\left(z_{sensor_{range}}\right) and\ P\left(z_{sensor_{heading}}\right)$$

The variances for each sensor are taken proportional (using constants c_1 and c_2) to the actual sensor reading. This is done because the landmarks keep varying and hence the variance values must track which landmark is being measured.

$$\sigma_{range} = c_1 \times range;\ \sigma_{heading} = c_2 \times heading$$

6)

The following are the range and bearing estimates to the given positions and landmarks

landmark = 6	range = 8.573130	heading = -1.584756
landmark = 13	range = 4.129145	heading = -1.196759
landmark = 17	range = 5.216302	heading = -3.075999

8)

Since there are no sensor readings for positions to control signal from point 2, there would be no measurement update step. Hence filter is not being implemented for point 2.

For point 3 the dead-reckoned plot vs the particle filter output at various time steps is shown below. The particle filter is using 1000 particles in the case shown below.

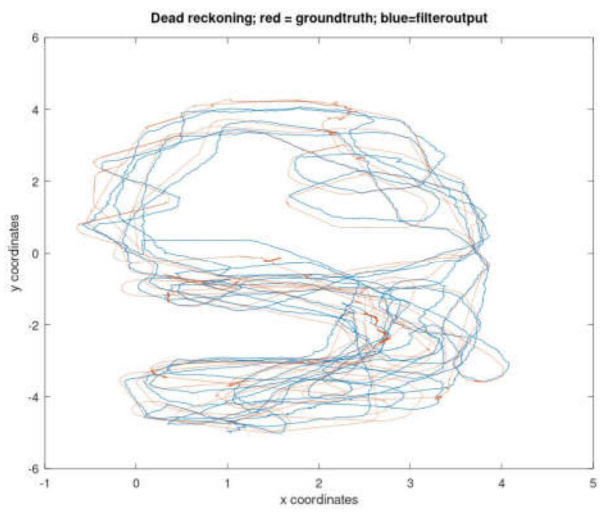


Figure 8

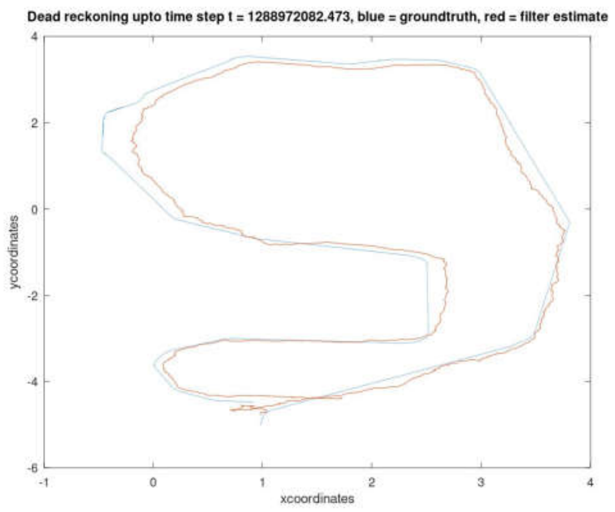


Figure 9

As can be seen from Figure 5, without the filter the robot starts deviating very rapidly from its actual path. Figure 8 shows that the filter, for the same noise parameters leads to a massive improvement in the robots localization. It is clearly seen that the recursive incorporation of the sensor data into the robots state belief, led to good level of tracking even with a basic motion model. Figure 9, which tracks the robot until T= 1288972082.473 gives a clearer representation of the tracking, due to the absence of overlapping lines. Filters with a high number of particles and with a high range of distribution over the initial estimate, lead to better performance. This is shown in the next point. If the actual motion, compared to the model is very noisy(due to poor modelling, conditions etc) then the particle filter's performance degrades even if the sensor values are accurate.

9)

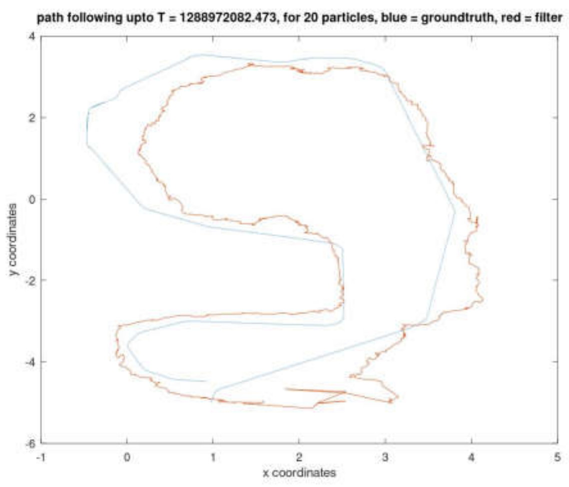
The first trend that strikes is that increasing the number of particles leads to better tracking. Figures 10 to 13 show the improvement in tracking against increasing magnitude of particles.

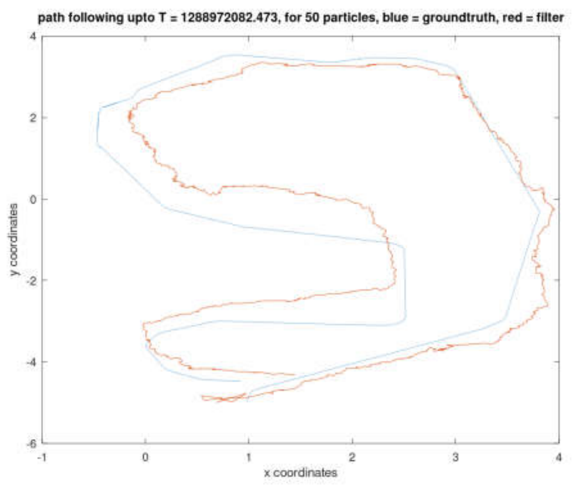
With higher number of particles there is a wider estimate of true state's being captured. There is also a lower chance that the weights of all particles converge, leading to a poor filter output. If this happens, and the particles converge to a narrow zone, the importance of measurement update step reduces.

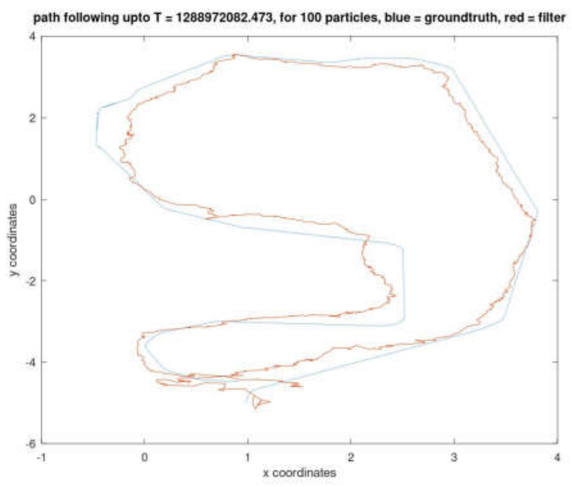
In the worst case scenario, the particles get narrowed down to such a small area that:

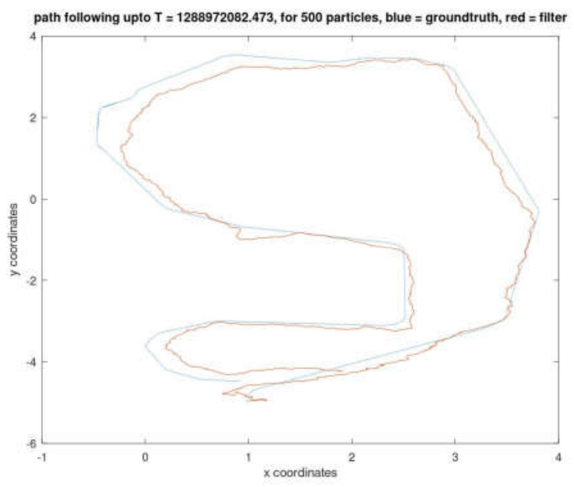
$$p(z_t \,|x_t^1) \approx p(z_t \,|x_t^2) \approx \; p(z_t \,|x_t^3) \; \approx p(z_t \,|x_t^N)$$

If this happens all particles will have similar weights, and the filter will stop tracking over time. This observation also holds true for another trend:



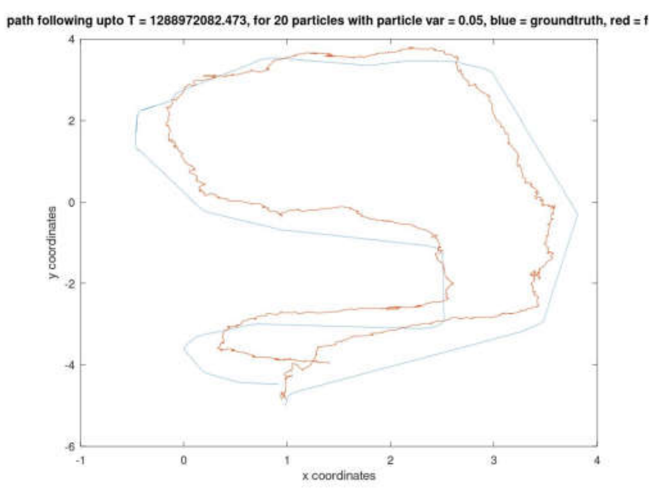






Figures 10,11,12,13: Figures show a positive relationship between growth of particles and filter tracking

If the initial estimate of particles is too narrow (low variance), the filter tracking performance is poor. On the other hand if the variance is too high, that too leads to a dip in performance. This can be seen in Figures 14, 15. Figure 14 shows the estimate with a very low variance of 0.05 of the initial estimate, while Figure 15



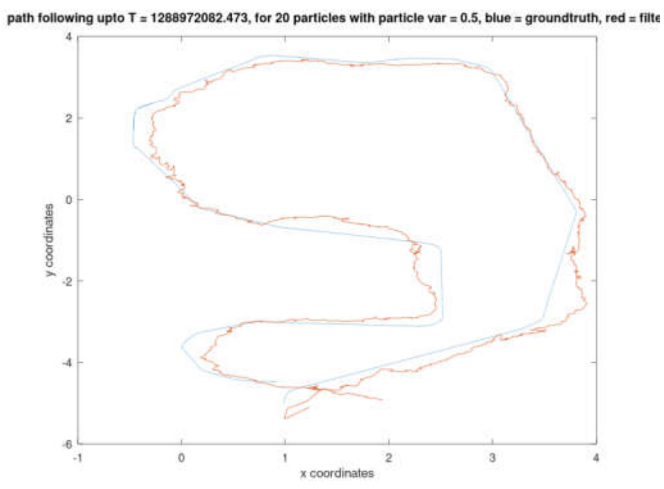


Figure 14,15

shows the performance when variance of the initial estimate is increased to 0.5. Both filters are run for 20 particles. Another interesting trend can be seen when these two figures are compared to Figure 9, which also runs a 20 particles system, but for an initial estimate variance of 2. Thus it might be concluded that optimum performance is found in a band of variance, and values above and below the band lead to degradation of the filter performance. (in our case 0.05(performance) < 2(performance) < 0.5(performance)

Figure 16 shows a simulated case when the sensor is made to miss 200 initial sensor readings. The plot shows that the filter does not manage to recover to track the state accurately in the mentioned time period. This shows that given a motion model, accuracy of the sensors and ability of the sensor model to estimate missed readings has a high impact on the filter performance.

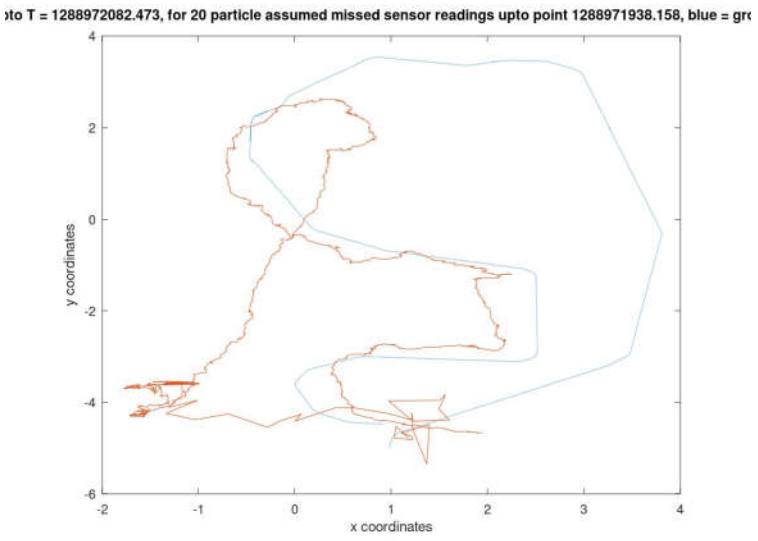


Figure 16

Limits: Increasing number of particles leads to a non linear increase in computational time. For 1000 particles it takes exponentially more time to compute, as compared to 500. Thus the risk reward ratio (in terms of computational time) goes down with an increase in particle size.

References:

[1] Particle Based Self-Localization using visual landmarks: Wardah Inam, Hamilton Institute

[2] Probabilistic Robotics: Thurn, Burgard,Fox

[3] A particle filter tutorial for mobile-robot localization: Ioannis Rekleitis, McGill University