APRIL 1, 2019

HOMEWORK 3

AUE 8930: AUTONOMOUS DRIVING TECHNOLOGIES

NAYAN DESHMUKH

Table of Contents

List of Figures

Problem 1

Question 1

Process of Kalman Filter

Kalman filtering process works in two steps – Prediction step and Correction step. These steps are explained below.

Nomenclature:

y_k^p : Predicted state of the system
y_k : State of the system at any given time-step
y_{k-1} : State of the system at previous time-step
u_k : Input to the system/Control signal
z_k : Measured value
P_k^p : Predicted error covariance
P_k : Error covariance at any given time-step
P_{k-1} : Error covariance at previous time-step
K : Kalman gain
Q : Covariance of process noise
R : Covariance of observation noise
A : System state matrix
B : Input matrix
C : Output matrix

Step 1: Prediction

1) The state of the system at previous time-step is used to predict the state of the system at current time-step. The predicted state is obtained by the following expression,

$$\mathbf{y_k^p = A*y_{k-1} + B*u_k}$$

2) Similar to the prediction of state of the system, the error covariance for current state of the system is calculated using the value of error covariance at the previous time-step.

$$\mathbf{P_k^p = A*P_{k-1}*A^T + Q}$$

Step 2: Correction

1) In the first step of correction, Kalman gain is calculated based on the error covariances and output matrix. The Kalman gain is then used to update/correct the system state. Kalman gain is given by:

$$\mathbf{K = P_k^p*C^T*(C*P_k^p*C^T + R)^{-1}}$$

2) The Kalman filter updates the state of the system using the measured value. It basically compares the measured state and the predicted state of the system and corrects the current based on trustworthiness of the two values.

$$\mathbf{y_k = y_k^p + K*(z_k - C*y_k^p)}$$

3) Using the Kalman gain, the current error covariance is also corrected so as to use it in the next time-step.

$$\mathbf{P_k = (I - K*C)*P_k^p}$$

Condition for application of Kalman filter:

- Kalman filter is used when the system process is linear and when it contains Gaussian noise.

Process of Extended Kalman filter

The process of extended Kalman filter is almost same as the Kalman filter. The extended Kalman filtering process also has Prediction step and Correction step similar to the Kalman filter. But here one more step is added before prediction step, the non-linear process is locally linearized before passing it to the prediction step.

The fundamental difference between the KF and EKF is that the later is used when the process is nonlinear. EKF does not guarantee optimality but gives the best state estimation for a non-linear process.

Nomenclature:

x_t: State of the system at any timestep t
$g(u_t,x_{t-1})$: Non-linear state function of the system
μ_t: Linearized state of the system at any timestep t
$\sum_t$: Error covariance of the system at any timestep t
G_t: Jacobian matrix that creates tangent plane to the previous timestep state of the system
H_t: Jacobian matrix that creates tangent plane to the current state of the system
K_t: Kalman gain
I: Identity matrix
Q_t: Process noise covariance matrix
R_t: Measurement noise covariance matrix
z_t: Measured values

Step 1: Local Linearization

The Extended Kalman filter is used when the process is non-linear. The non-linear function of system state is first linearized locally so that the Kalman filter algorithm can then be used for it. The linearization of the state function is done by performing Taylor expansion. First two terms of the Taylor expansion are approximated as the state function and higher order terms are neglected. Subsequently we obtain the following expression for state of the system at any time-step t.

$$\left.\begin{aligned} x_t &= g(u_t, x_{t-1}) \\ z_t &= h(x_t) \end{aligned}\right\} \text{Non-linear expressions}$$

$$\left.\begin{aligned} g(u_t, x_{t-1}) &\simeq g(u_t, \mu_{t-1}) + \frac{\partial g(\mu_t, \mu_{t-1})}{\partial x_{t-1}}(x_{t-1} - \mu_{t-1}) \\ g(u_t, x_{t-1}) &\simeq g(\mu_t, \mu_{t-1}) + G_t(x_{t-1} - \mu_{t-1}) \\ h(x_t) &\simeq h(\bar{\mu}_t) + \frac{\partial h(\bar{\mu}_t)}{\partial x_t}(x_t - \bar{\mu}_t) \\ h(x_t) &\simeq h(\bar{\mu}_t) + H_t(x_t - \bar{\mu}_t) \end{aligned}\right\} \begin{gathered}\text{Expansion using} \\ \text{Taylor series}\end{gathered}$$

Jacobians of the matrices here (G_t and H_t) do the linear transformation of the function by creating a tangent plane to each point in the 3D space so as to locally linearize it.

Step 2: Prediction

1) The state of the system at previous time-step is used to predict the state of the system at current time-step. In case of the extended Kalman filter, this function is a combination of previous state of the system and current input to the system. Based on these two factors, the predicted state is obtained by the following expression,

$$\bar{\mu}_t = g(u_t, \mu_{t-1})$$

2) Similar to the prediction of state of the system, the error covariance for current state of the system is calculated using the value of error covariance at the previous time-step. The Jacobian matrix G_t, system noise covariance (Q) and error covariance at previous step give the current error covariance.

$$\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + Q_t$$

Step 3: Correction

1) In the first step of correction, Kalman gain is calculated based on the error covariances, Jacobian matrix (H_t), and measurement noise covariance matrix (R_t). The Kalman gain is then used to update/correct the system state. Kalman gain is given by:

$$K_t = \bar{\Sigma}_t H_t^T * (H_t \bar{\Sigma}_t H_t^T + R_t)^{-1}$$

2) The Kalman filter updates the state of the system using the measured value. It basically compares the measured state and the predicted state of the system and corrects the current based on trustworthiness of the two values.

$$\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$$

3) Using the Kalman gain, the current error covariance is also corrected so as to use it in the next time-step.

$$\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$$

Condition for application of Extended Kalman filter:

- Slightly non-linear system process model having Gaussian noise distribution.

Process of Unscented Kalman filter

The Unscented Kalman filter uses a deterministic technique known as the unscented transformation to pick a minimal set of sample points, called sigma points, around the mean. These sigma points are then propagated through non-linear functions, from which new mean and covariance estimates are then formed.

Initially, the sigma points are defined and propagated through non-linear functions and this process is called Unscented Transformation (UT).

1) Define Sigma points:

$$\chi_0 = \bar{x} \qquad i = 0$$
$$\chi_i = \bar{x} + \left(\sqrt{(L+\lambda)P_x}\right)_i \qquad i = 1, \ldots, L$$
$$\chi_i = \bar{x} - \left(\sqrt{(L+\lambda)P_x}\right)_{i-L} \qquad i = L+1, \ldots, 2L$$

   x' is assumed to be the mean of x and P_x, its covariance. The dimension of x is L. A set of 2L+1 weighted samples or sigma points are chosen as follows:
   where,
   $\lambda = \alpha^2(L + k) - L$ is a scaling parameter
   The constant α determines the spread of the sigma points around mean x'
   The constant k is secondary scaling parameter

2) Propagation of Sigma points:
   Each sigma point is propagated through non-linear function.

$$y_i = f(\chi_i) \quad i = 0, \ldots, 2L$$

   The estimated mean and covariance of y are then calculated.

Step 1: Prediction

Calculate sigma points for the process.

$$x^a_{k-1} = \left[\hat{x}^a_{k-1} \quad \hat{x}^a_{k-1} + \gamma\sqrt{P^a_{k-1}} \quad \hat{x}^a_{k-1} \mp \gamma\sqrt{P^a_{k-1}}\right]$$

Based on the previous state of the system and input to the system, predict the current state of the system at timestep t. Also, find the error covariance of the system as shown below.

$$\chi^x_{k|k-1} = F\left[\chi^x_{k-1}, u_{k-1}, \chi^v_{k-1}\right]$$
$$\hat{x}^-_k = \sum_{i=0}^{2L} W_i^{(m)} \chi^x_{i,k|k-1}$$
$$P^-_k = \sum_{i=0}^{2L} W_i^{(c)} \left[\chi^x_{i,k|k-1} - \hat{x}^-_k\right]\left[\chi^x_{i,k|k-1} - \hat{x}^-_k\right]^T$$
$$y_{k|k-1} = H\left[\chi^x_{k|k-1}, \chi^n_{k-1}\right]$$
$$\hat{y}^-_k = \sum_{i=0}^{2L} W_i^{(m)} y_{i,k|k-1}$$

Step 2: Correction

Correction step for the Unscented Kalman filter is exactly the same as any other Kalman filter. First, the Kalman gain is computed. The system sensor measurements are recorded. The current state of the system is found using the predicted state of the system, predicted value of the system, measured value of the system and Kalman gain. The Kalman gain decides which value to trust more, system predicted or measured and accordingly makes alteration to the predicted state estimate to give actual state of the system. Also, the error covariance is found. The current system state and error covariance are then fed into the prediction step again as system state and error covariance of the previous timestep to obtain the system state at next timestep.

$$P_{\hat{y}_k \hat{y}_k} = \sum_{i=0}^{2L} W_i^{(c)} [\mathcal{Y}_{i,k|k-1} - \hat{y}_k^-][\mathcal{Y}_{i,k|k-1} - \hat{y}_k^-]^T$$

$$P_{\hat{x}_k \hat{y}_k} = \sum_{i=0}^{2L} W_i^{(c)} [\mathcal{X}_{i,k|k-1} - \hat{x}_k^-][\mathcal{Y}_{i,k|k-1} - \hat{y}_k^-]^T$$

$$K_k = P_{x_k y_k} P_{\hat{y}_k \hat{y}_k}^{-1}$$

$$\hat{x}_k = \hat{x}_k^- + K_k (z_k - \hat{y}_k^-)$$

$$P_k = P_k^- - K_k P_{\tilde{y}_k \tilde{y}_k} K_k^T$$

Condition for application of Unscented Kalman filter:

- Gives balanced performance for linear as well as non-linear process models.

Process of Particle filter

Particle filter is used in a situation where we want to know the state of a system and measurements available have a specific relationship with the state of the system. Particle filter can be used for linear as well as non-linear system processes. It is a technique for implementing recursive Bayesian filter by Monte-Carlo sampling.

Particle filtering works in three steps – prediction step, update step and resample step.

Step 1: Prediction step

Initialization of particle filter results in generation of N random particles. The particle filter samples the particle from the proposal distribution. Then, it takes each particle and predicts its next state based on the motion model and control input.

Step 2: Compute the importance weight and update the system state

Sensor measurements are recorded. These measurements are then compared with the particle's state estimate. Depending on the difference between the state estimate and measured value, the importance weight is calculated. More the deviation in the values of estimate and sensor measurement, less the importance weight.

Step 3: Resampling

After computing importance weight for each particle, the ones with insignificant weights are discarded. New set of particles is chosen so that each particle survives in proportion to its weight and the procedure is again repeated from step 1 till the system receives observations or sensor measurements.

Condition for application of Particle filter

- Performs good with both linear as well as non-linear process model containing Gaussian or non-Gaussian noise.

Question 2

The Problem of SLAM:

Simultaneous Localization and Mapping (SLAM) is a computational problem of constructing and updating a map of unknown environment while simultaneously keeping a track of robot's location within it. The robot is able to estimate its location and positions of landmarks in an unknown environment given the robot controls and observations of the nearby features or landmarks over discrete timesteps.

SLAM is a chicken-egg problem, meaning, a map is needed for localizing a robot while a good pose estimate for the robot is needed to build a map. SLAM is regarded as one of the hardest problems in robotics. This is because both – the robot pose and map are unknown at initial timestep. Hence, errors are induced in the system thus, reducing the reliability of safe operations of robot in the real world.

Necessity of SLAM in autonomous driving:

Localization is one of the important aspects of autonomous vehicles. A vehicle able to identify its own location even in any environment is an important functionality in self-driving vehicles. Currently, technologies such as Global Navigation Satellite System (GNSS) and RTK-GNSS are thought of as a solution to the localization problem. But, these technologies have limitations like signals being affected due to atmospheric conditions, blocking of signals due to infrastructure, problems of performance in dense urban areas, etc. Due to these limitations, its application for localization of autonomous vehicles seems insufficient.

Another approach to localization is to take advantage of road infrastructure in order to guide the vehicle in the lane. But even this approach seems insufficient as it constrains the lateral position of the vehicle, also this system will only work where the road markings are identifiable, thus rendering it useless for when there are no road markings, or when they are not visible.

One of the latest technologies to solve the problem of localization for autonomous vehicles is SLAM. SLAM computes the location of vehicle in any environment, be it in global or local frame, thus, the vehicle is enabled to perform any perception or planification tasks. SLAM can predict the evolution of the other obstacles on the road and choose which maneuver is the most appropriate taking into account the vehicle's current location and predicting its motion in coming seconds. The SLAM framework is compatible to any sensor or estimation technique that suits the prerequisite of estimating both the localization of the vehicle and the map at the same time. The map is very important for autonomous driving since it offers a first level of perception that is needed in order to make appropriate driving decisions. Thus, SLAM technology is considered as one of the important aspects of autonomous driving [1].

Question 3

Working of EKF SLAM

The EKF SLAM works on the principle of Extended Kalman filter technique. To determine the location of vehicle using EKF slam, two inputs are required – one, map of the environment and second, sequence of sensor measurements. Given these inputs, EKF SLAM can estimate vehicle's position at any timestep. The EKF SLAM can thus be used for position tracking, global localization, vehicle in an unknown environment problem, etc.

The EKF SLAM works in five steps – local linearization step, initialization, prediction step, recording of measured values of environment and correction step.

Step 1: Local Linearization

The non-linear function of system state is first linearized locally so that the Kalman filter algorithm can then be used for it. The linearization of the state function is done by performing Taylor expansion. First two terms of the Taylor expansion are approximated as the state function and higher order terms are neglected. Subsequently we obtain the following expression for state of the system at any time-step t.

$$\left.\begin{array}{l} x_t = g(u_t, x_{t-1}) \\ Z_t = h(x_t) \end{array}\right\} \text{Non-linear expressions}$$

$$\left.\begin{array}{l} g(u_t, x_{t-1}) \simeq g(u_t, \mu_{t-1}) + \dfrac{\partial g(\mu_t, \mu_{t-1})}{\partial x_{t-1}} (x_{t-1} - \mu_{t-1}) \\ g(u_t, x_{t-1}) \simeq g(\mu_t, \mu_{t-1}) + G_t(x_{t-1} - \mu_{t-1}) \\ h(x_t) \simeq h(\bar{\mu}_t) + \dfrac{\partial h(\bar{\mu}_t)}{\partial x_t} (x_t - \bar{\mu}_t) \\ h(x_t) \simeq h(\bar{\mu}_t) + H_t(x_t - \bar{\mu}_t) \end{array}\right\} \begin{array}{l}\text{Expansion using} \\ \text{Taylor series}\end{array}$$

Jacobians of the matrices here (G_t and H_t) do the linear transformation of the function by creating a tangent plane to each point in the 3D space so as to locally linearize it.

Step 2: Initialization

The system needs to have some initializing parameters to compute the state of the system at subsequent timesteps. Initially, robot starts in its own reference frame, all landmarks are unknown. If there are N landmarks on the map, to initiate the EKF SLAM process, the state vector of mean estimates is assumed to be a vector of zeros. Also, the error covariance matrix is required to initiate the process. All the elements of covariance matrix are set zero apart from the diagonal elements that correspond to the landmarks. Ideally, for unknown landmarks, error covariance of infinite is set since we have no clue about them. Therefore, the state vector of mean estimates and covariance matrix required to initialize look like this,

$$\mu_0 = (0, 0, 0, 0 \ldots\ldots 0)^T$$

$$\Sigma_0 = \begin{bmatrix} 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \infty & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \infty \end{bmatrix}$$

Step 3: Prediction

Again, the prediction step in EKF SLAM is similar to EKF, first the mean estimate of system state is computed based on the state vector of the system at previous timestep and the motion input to the system.

$$\bar{\mu}_t = g(u_t, \mu_{t-1})$$

u_t in this equation is the motion input to the system and μ_{t-1} is the mean state vector of the system at previous timestep.

Subsequently, error covariance for the current timestep is predicted based on the previous timestep and linearizing Jacobian matrix (G_t). The system noise covariance (R_t) is accounted for in this step.

$$\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + Q_t$$

Step 4: Measured values of the environment recorded

In this step, the data available from the system sensors is recorded. The data is usually in parameters that can be measured like – distance, angle, temperature, etc. This observation is recorded as z_t^i, where i is the number of observation at any time-step t.

Step 5: Correction

First the Kalman gain is calculated based on the predicted error covariance and Jacobian matrix of observation (H_t). The measurement noise covariance it accounted in this step,

$$K_t = \bar{\Sigma}_t H_t^T * (H_t \bar{\Sigma}_t H_t^T + R_t)^{-1}$$

After obtaining the value for Kalman gain, the state vector of the system is updated. Depending on its trust on predicted value and measured value, the EKF makes changes to the state vector of the system. This new state vector for the current timestep gives the data of current pose of the vehicle as well as the location of obstacles or object from the vehicle's current location in global coordinates.

$$\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$$

Subsequently, the error covariance of the current timestep is calculated.

$$\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$$

This state vector for system at current timestep is again fed into the prediction step to find mean estimate of state vector at next timestep. Similarly, the error covariance computed at the end of first timestep is fed into the prediction step to predict the error covariance at next timestep.

Applicable Conditions:

- Can be used when the system process is only slightly non-linear.
- The system noise has Gaussian distribution.

Working of FastSLAM

FastSLAM uses particle filter for map generation and localization of the robot. In FastSLAM, the key aspect is that the pose of the robot at every timestep is known, hence, the mapping becomes easy. In this technique, the particle filter is used to represent the trajectories of the robot and a map is built for every individual sample or landmark. Thus, particle filter represents non-linear process model and non-Gaussian pose distribution for the robot pose but the FastSLAM linearizes the observation model since low dimension EKFs are used.

FastSLAM uses Rao-Blackwellized method to map the environment and to localize itself. Rao-Blackwellized is a probabilistic method which states that for a given probability of a, probability of b can be easily computed by the following way,

$$p(a, b) = p(b|a)\,p(a)$$

For our case of localization and mapping,

a = Trajectory of robot

b = Map

for given p(a), we can find p(b) using p(a)

$$p(\underset{\substack{\uparrow\\ \text{poses}}}{x_{0:t}}, \underset{\substack{\uparrow\\ \text{map}}}{m_{1:M}} | \underset{\substack{\uparrow\\ \text{observations}}}{z_{1:t}}, u_{1:t}) = \underset{\substack{\uparrow\\ \text{path posterior}}}{p(x_{0:t}|z_{1:t}, u_{1:t})}\; \underset{\substack{\uparrow\\ \text{map posterior}}}{p(m_{1:M}|x_{0:t}, z_{1:t})}$$

$$= \underset{\substack{\uparrow\\ \text{Particle filter using}\\ \text{Monte-Carlo Localization}}}{p(x_{0:t}|z_{1:t}, u_{1:t})} \prod_{i=1}^{M} \underset{\substack{\uparrow\\ \text{2 Dimensional EKFs}}}{p(m_i|x_{0:t}, z_{1:t})}$$

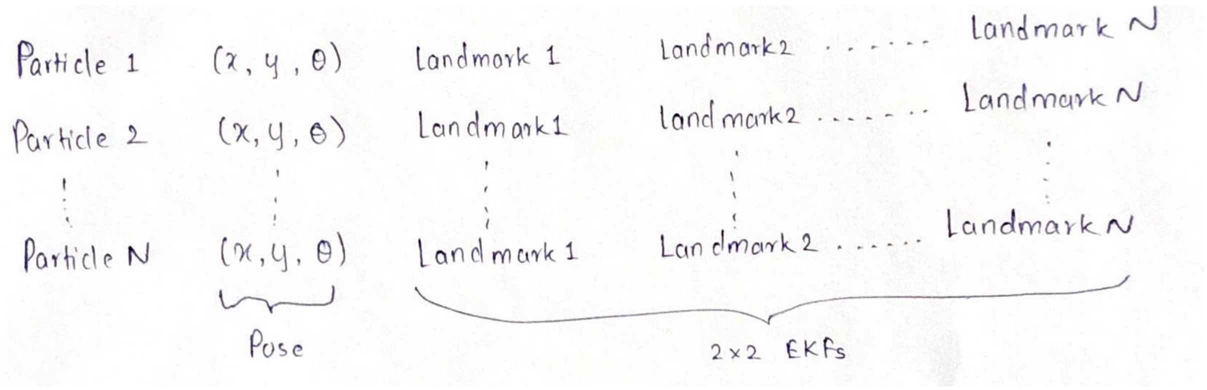
Here we can see that pose of the robot is determined using Particle filter that uses Monte-Carlo localization, while the landmarks are determined by two dimensional EKFs.

Step 1: Initialization

Before going for the path hypothesis, it is necessary to initialize the FastSLAM by setting the initial pose condition. Initially, the pose of the robot is set to [x,y,ϴ]=[0,0,0]. The pose of robot at each subsequent timestep is already known.

Step 2: Extend path posterior by sampling a new pose for each sample

If N particles are used for FastSlam. Each particle of the total N particles has its own robot pose and observed landmarks at every timestep. After every timestep, each particle registers their hypothesis of the path taken by the robot and its current distance from the observed landmarks.



Step 3: Take Observations

Observations are recorded with the help of the system sensors.

Step 4: Compute particle weight

Depending on the difference between the predicted robot state and the measured state, importance weights are given to the particles. The lesser the difference, higher the weight.

Step 5: Update belief of observed landmarks (EKF update rule)

In this step, the extended Kalman filter updates the belief of each individual landmark to initialize it for the next timestep.

Step 6: Resample

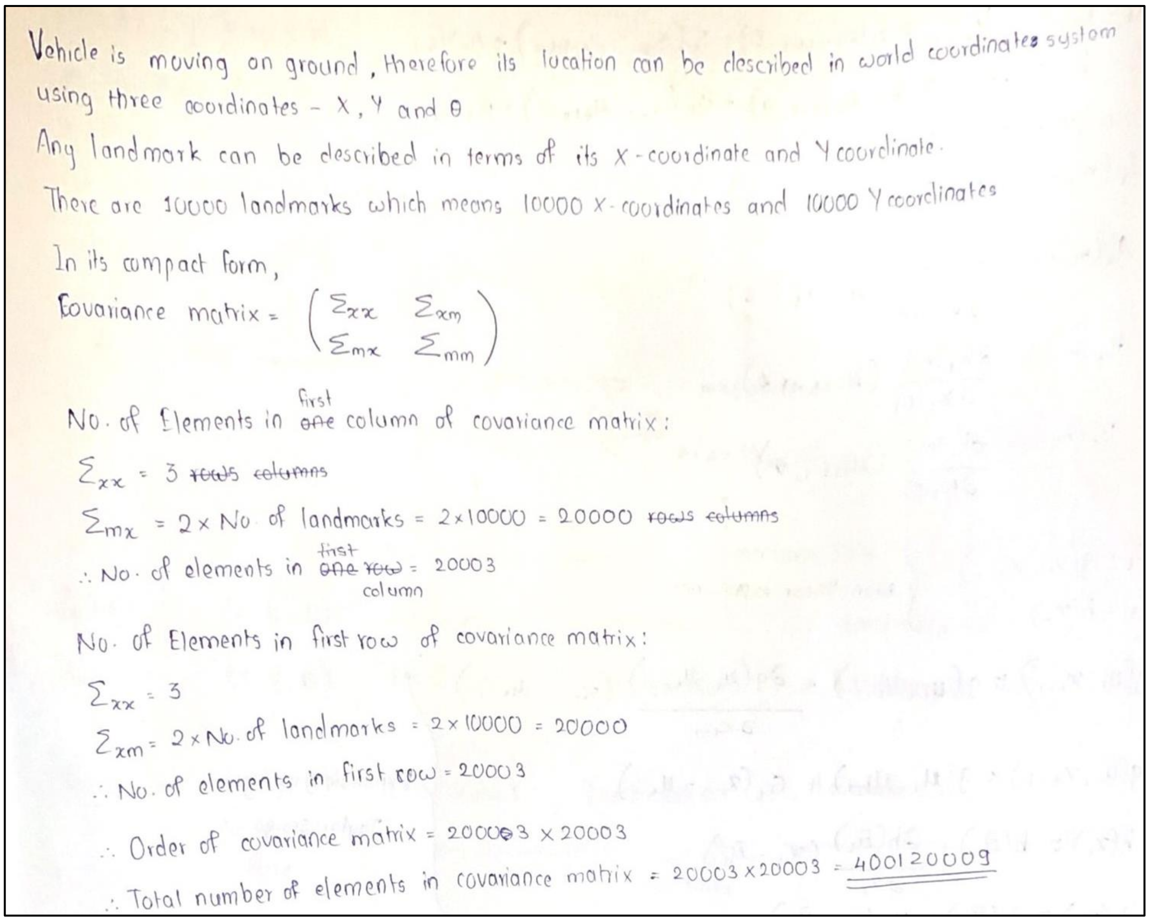
Based on the update belief and computed weights, the particles are redistributed. Now these resampled particles are considered as initialization for next timestep.

Applicable Conditions:

- Huge map with many landmarks.
- Non-linear process model and non-Gaussian pose distribution for robot pose estimate.

Question 4

Elements in covariance matrix of EKF SLAM



Vehicle is moving on ground, therefore its location can be described in world coordinates system using three coordinates - X, Y and θ

Any landmark can be described in terms of its X-coordinate and Y coordinate.

There are 10000 landmarks which means 10000 X-coordinates and 10000 Y coordinates

In its compact form,

Covariance matrix = $\begin{pmatrix} \Sigma_{xx} & \Sigma_{xm} \\ \Sigma_{mx} & \Sigma_{mm} \end{pmatrix}$

No. of Elements in ~~one~~ first column of covariance matrix:

Σ_{xx} = 3 ~~rows~~ ~~columns~~

Σ_{mx} = 2 × No of landmarks = 2×10000 = 20000 ~~rows~~ ~~columns~~

∴ No. of elements in ~~one~~ ~~row~~ first column = 20003

No. of Elements in first row of covariance matrix:

Σ_{xx} = 3

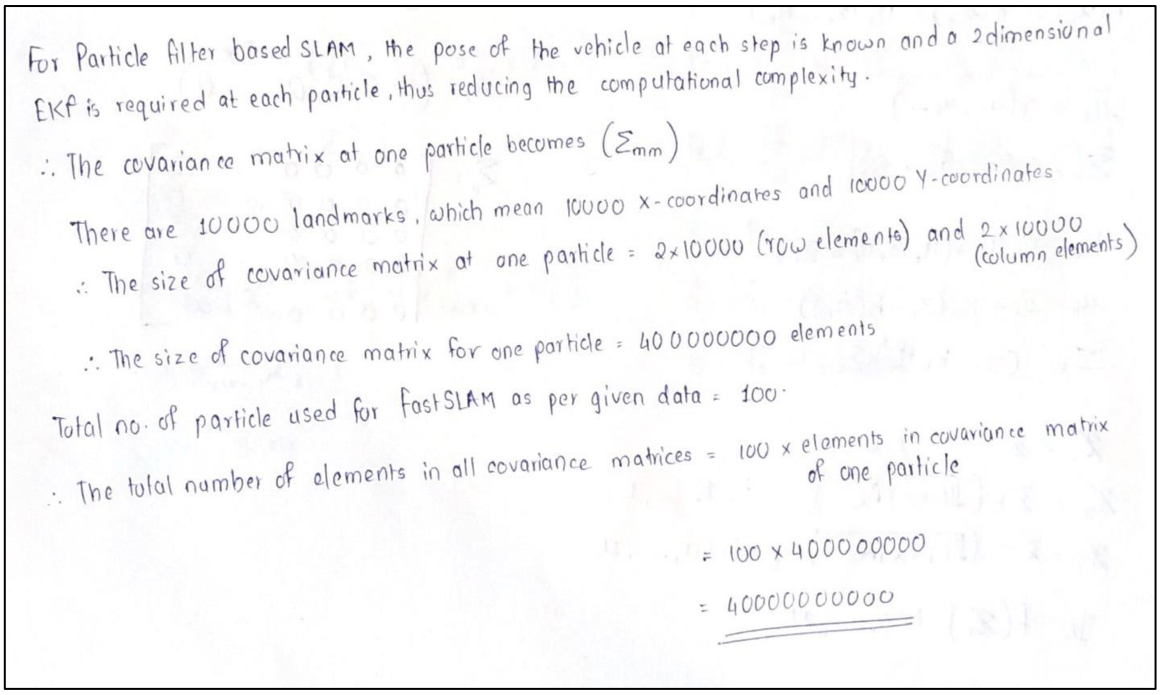
Σ_{xm} = 2 × No. of landmarks = 2 × 10000 = 20000

∴ No. of elements in first row = 20003

∴ Order of covariance matrix = 20003 × 20003

∴ Total number of elements in covariance matrix = 20003×20003 = 400120009

Total number of elements in all covariance matrices when using Fast SLAM



For Particle filter based SLAM, the pose of the vehicle at each step is known and a 2dimensional EKF is required at each particle, thus reducing the computational complexity.

∴ The covariance matrix at one particle becomes (Σ_{mm})

There are 10000 landmarks, which mean 10000 X-coordinates and 10000 Y-coordinates

∴ The size of covariance matrix at one particle = 2×10000 (row elements) and 2×10000 (column elements)

∴ The size of covariance matrix for one particle = 400000000 elements

Total no. of particle used for FastSLAM as per given data = 100.

∴ The total number of elements in all covariance matrices = 100 × elements in covariance matrix of one particle

= 100 × 400000000

= 40000000000

Question 5

C-obstacle space:

The C-obstacle space is a region in the workspace of a robot where its configurations/movements are not possible. In other words, obstacle in c-space is just such a region in c-space whose set of points are not legal configurations of robot.

C-obstacle space of a square obstacle for a circle vehicle using Minkowski sum:

Minkowski sum: $A \oplus B = \{a + b \mid a \in A, b \in B\}$

Square Obstacle: Side length = 4 meters

Circle Vehicle: Diameter = 2 meters

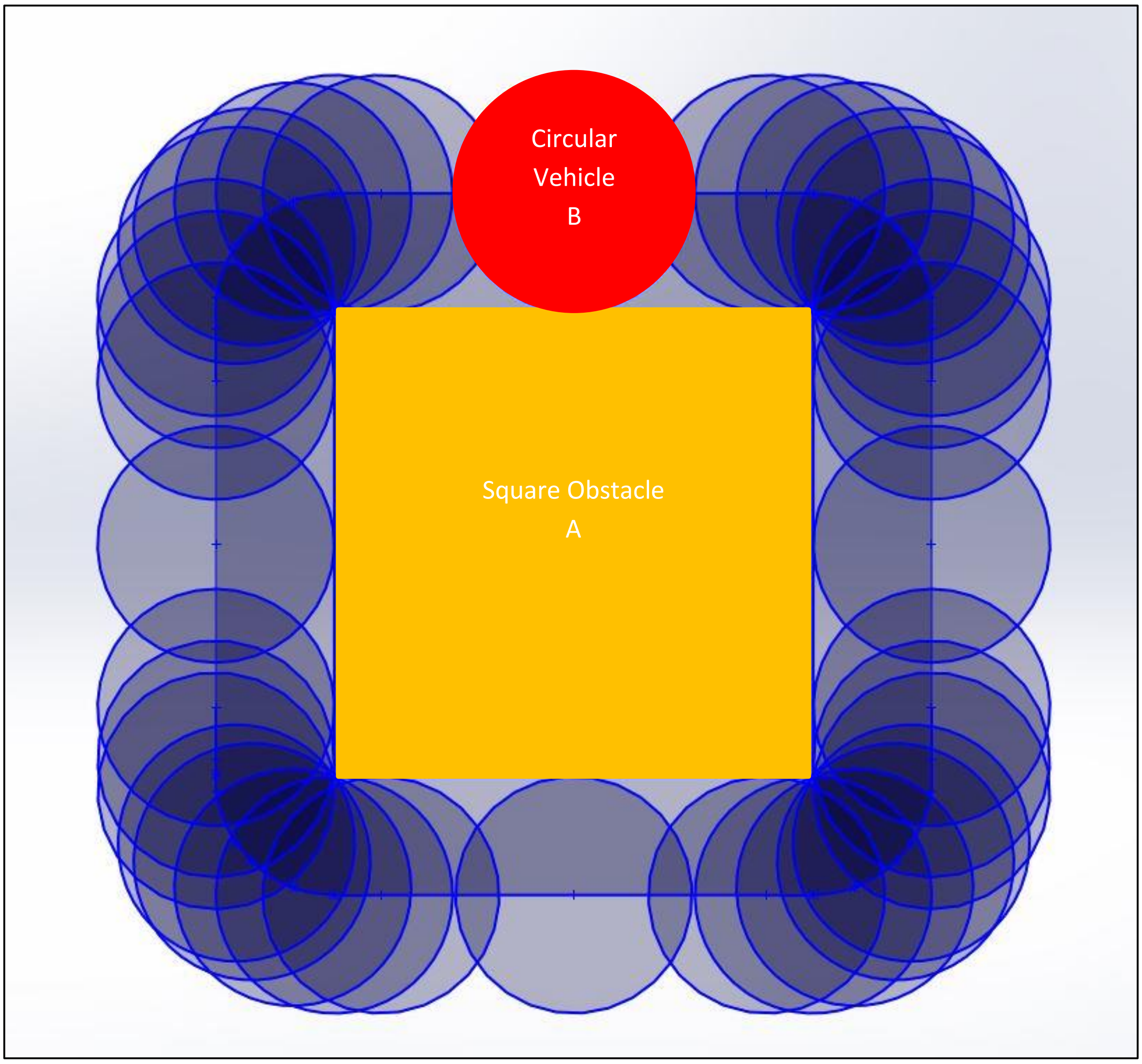


Figure 1: Configuration of circle vehicle around square obstacle

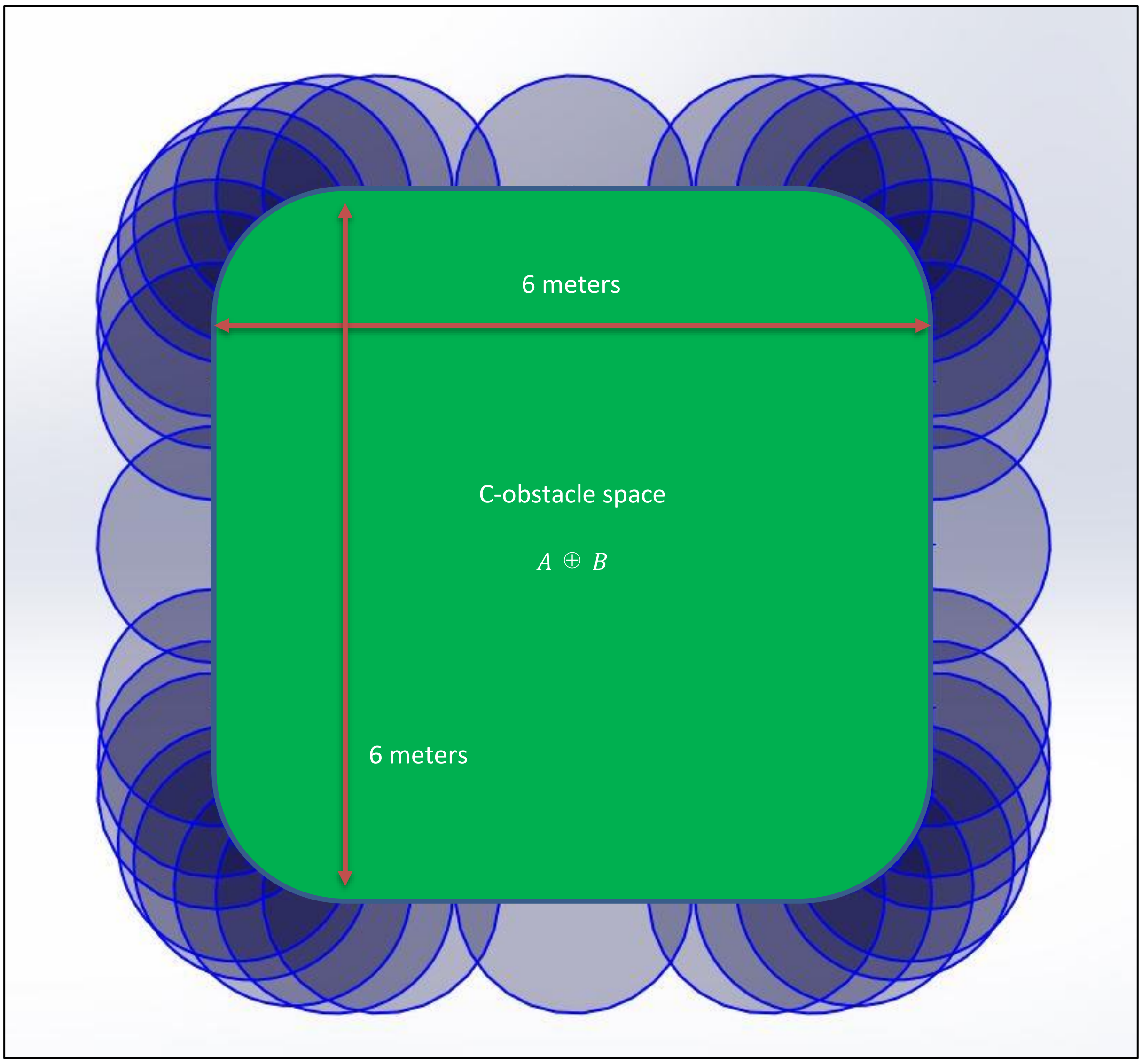


Figure 2: C-Obstacle Space

Question 6

Voronoi Diagram using L1 metric

L1 metric:

(x,y): |x| + |y| = constant

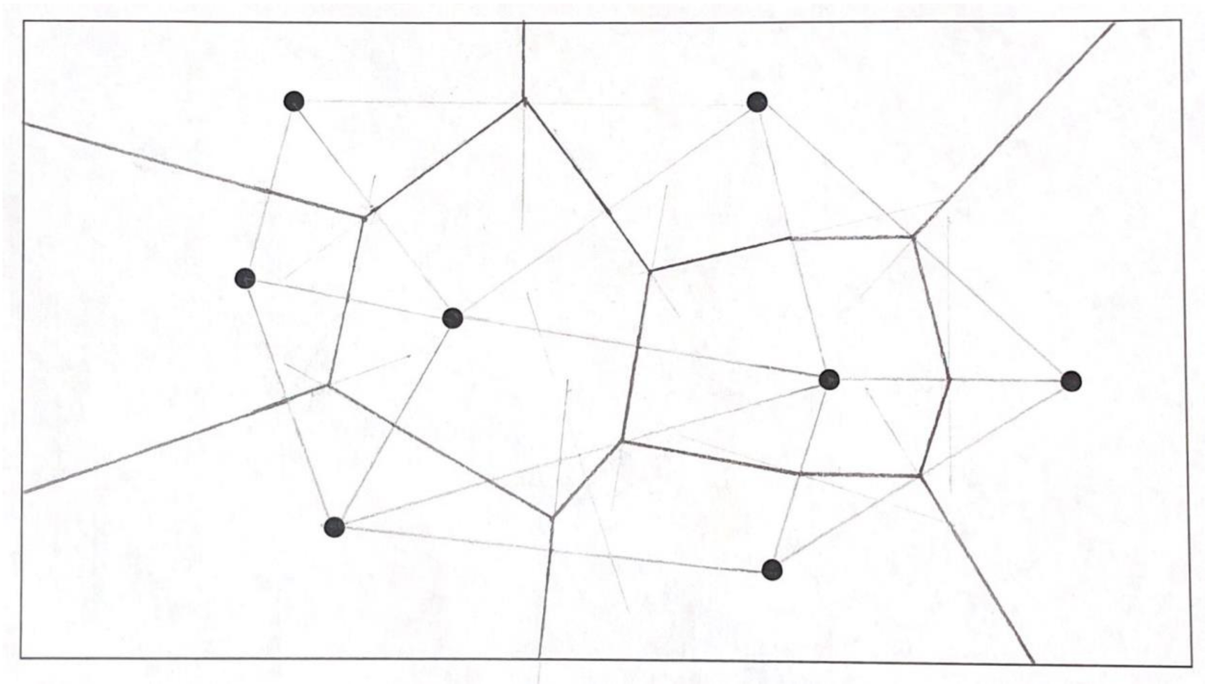


Figure 3: Voronoi diagram

Procedure followed to draw the Voronoi diagram

1. All the given points were joined to form triangles.
2. Mid-point of each line segment was marked, and perpendicular bisectors were drawn for each line segment.
3. Then the bold lines shown in figure (3) were drawn connecting the perpendicular bisectors to show the possible motion that can be planned around the dots given.

Problem 2

The MATLAB code for this problem is attached in the appendix of the report. The results of the EKF SLAM are discussed here.

Results

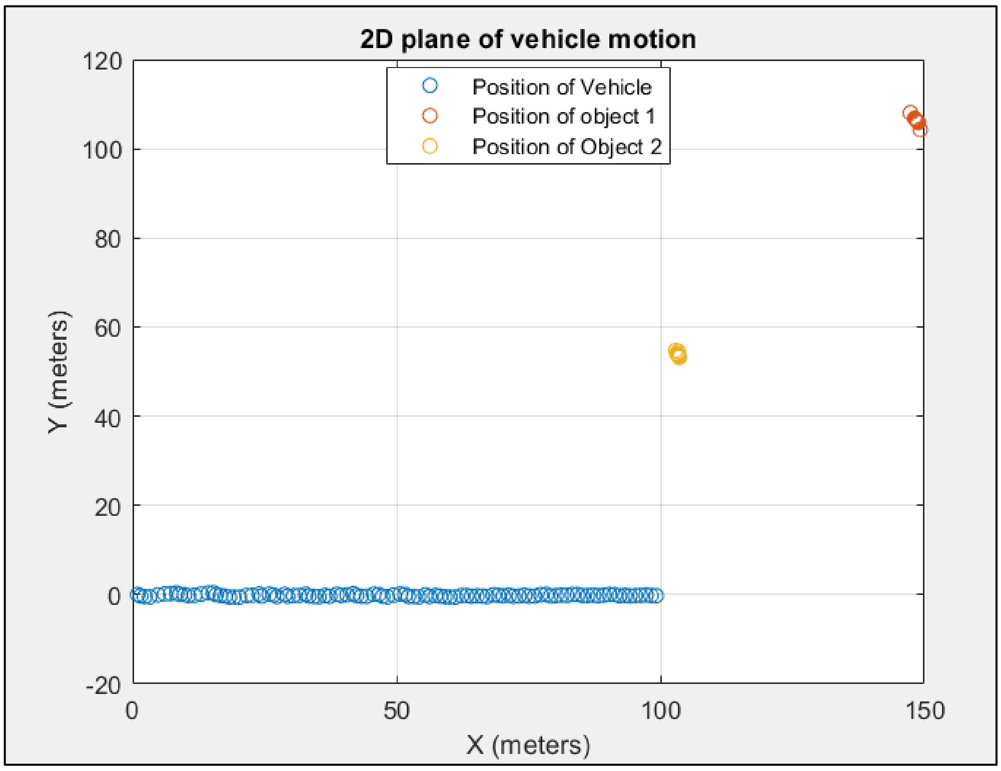
To initialize the EKF SLAM, the initial mean state vector was given. But depending on the elements of the error covariance matrix the results of the SLAM change a bit. An observation can be made that our trust in location of the two objects in the 2D plane alters the result of the EKF SLAM.

Scenario 1: Not very sure about location of objects in 2D plane

In this condition, we know only a rough estimate of where the objects are located in 2D plane and not the exact location of it. Hence, the error covariance of the objects is set 100. This is how the error covariance matrix looks like,

$$\Sigma = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 100 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 100 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 100 \end{bmatrix}$$

For this initial condition of covariance matrix, following result is obtained,



Interpretation

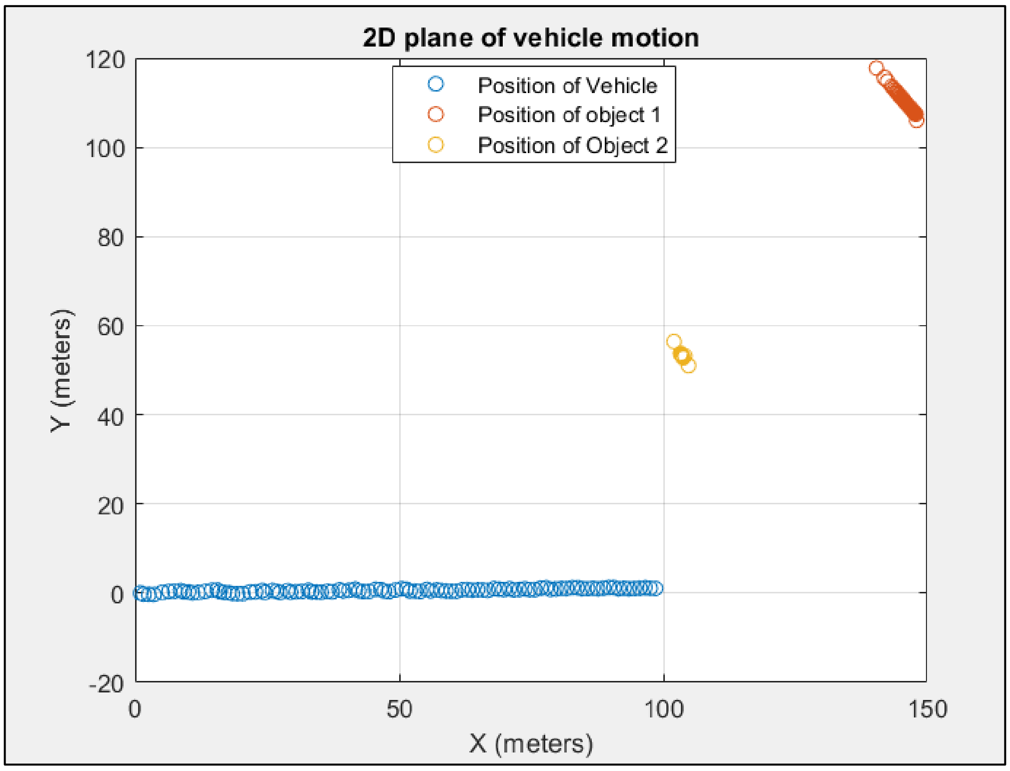
From this graph we can see that since we are not quite sure about the location of the two objects, but we have a rough estimate of where they are located, the spread of estimated location of the object determined by EKF SLAM is small.

Scenario 2: Robot in completely unknown territory without any knowledge of the location of objects

In this condition, we know nothing about where the objects are located in 2D plane. Hence, the error covariance of the objects is set 1000000. Ideally, since we don't know the location of object, the covariance of object must be set to infinite but then computation of SLAM problem becomes impossible. Hence, I have given covariance of objects a huge number in covariance matrix instead of infinite. This is how the error covariance matrix looks like,

$$\Sigma = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1000000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1000000 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1000000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1000000 \end{bmatrix}$$

For this initial condition of covariance matrix, following result is obtained,



Interpretation

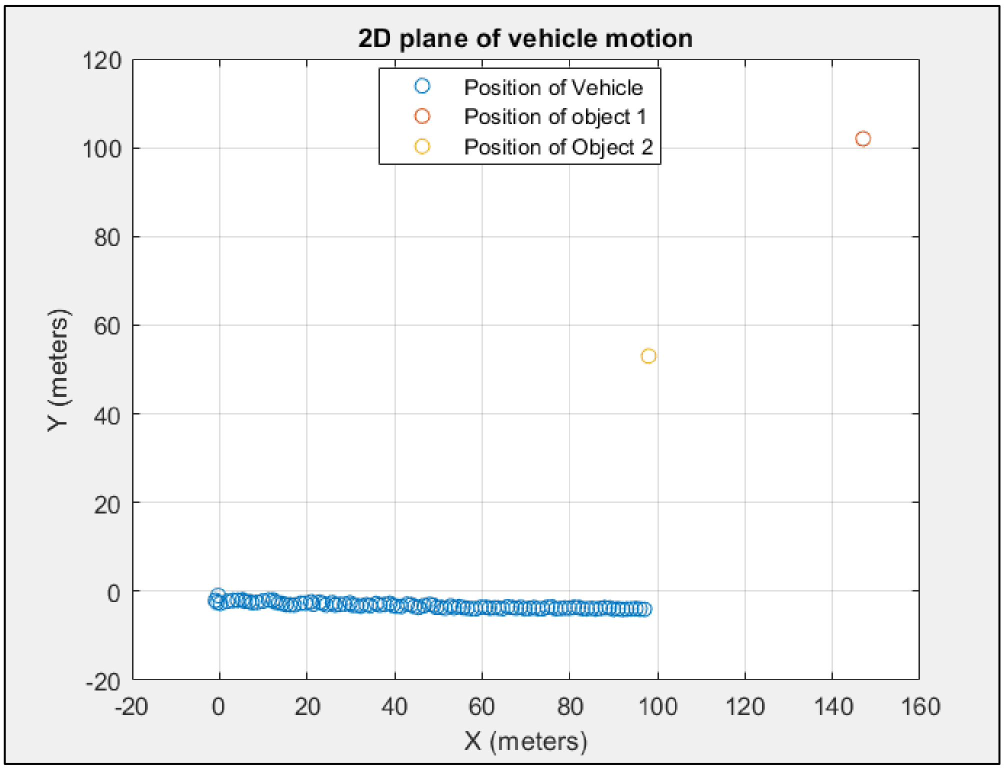
Here, we have absolutely no clue where the objects are located in the 2D plane and hence, the EKF SLAM gives a wide range of position estimates of the objects over sampling time.

Scenario 3: We know perfectly where the objects are in the 2D plane of motion of robot

Since we perfectly know the location of objects in this scenario, the error covariance for objects is set to 0 and hence, the covariance matrix looks something like this,

$$\Sigma = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

For this initial condition of covariance matrix, following result is obtained,



Interpretation

In this scenario, we exactly know where the objects are located on the 2D plane of robot motion and hence, EKF SLAM gives constant location of the two objects for any sampling time.

Reference

1) Bresson, Guillaume, et al. "Simultaneous Localization and Mapping: A Survey of Current Trends in Autonomous Driving." *IEEE Transactions on Intelligent Vehicles* 2.3 (2017): 194-220. *CrossRef.* Web.

Appendix

Matlab Code

```
clear

clc



% Load given sensor measurements for location of objects in plane of motion
% of vehicle
load 's1.mat'
load 's2.mat'
```

Given parameters

```
v=1; % Velocity of vehicle
w=0; % Radial veocity of vehicle
dt=1; % Sampling time
```

Initialization condition of EKF SLAM

```
u=[0;0;0;147;102;98;53]; % Initial mean state estimate vector for system
E=[0,0,0,0,0,0,0;
   0,0,0,0,0,0,0;
   0,0,0,0,0,0,0;
   0,0,0,10,0,0,0;
   0,0,0,0,10,0,0;
   0,0,0,0,0,10,0;
   0,0,0,0,0,0,10]; % Initial error covariance matrix

uo=[0;0;0;147;102;98;53]; % Initial mean state vector for system
Eo=[0,0,0,0,0,0,0;
   0,0,0,0,0,0,0;
   0,0,0,0,0,0,0;
   0,0,0,100,0,0,0;
   0,0,0,0,100,0,0;
   0,0,0,0,0,100,0;
   0,0,0,0,0,0,100]; % Initial error covariance matrix
```

EKF SLAM

```
I=eye(7); % Identity matrix
R=[0.1,0,0,0,0,0,0;
    0,0.1,0,0,0,0,0;
    0,0,0.1,0,0,0,0;
    0,0,0,0,0,0,0;
    0,0,0,0,0,0,0;
    0,0,0,0,0,0,0;
    0,0,0,0,0,0,0]; % Motion noise covariance matrix
Q=[0.1,0;
   0,0.01]; % Sensing noise covariance matrix
V=[v;
   0;
   0]; % Velocity vector
J=[0,0,0;
   0,0,0;
   0,0,0]; % Jacobian matrix for locally linearizing the system
Fx=[1,0,0,0,0,0,0;
    0,1,0,0,0,0,0;
    0,0,1,0,0,0,0]; % Fx is a matrix to map any variable vector or matrix to higher
dimensional space matrix
Fx1=[1,0,0,0,0,0,0;
     0,1,0,0,0,0,0;
     0,0,1,0,0,0,0;
     0,0,0,1,0,0,0;
     0,0,0,0,1,0,0]; % Fx1 is a matrix to map any variable vector or matrix to
higher dimensional space matrix
Fx2=[1,0,0,0,0,0,0;
      0,1,0,0,0,0,0;
      0,0,1,0,0,0,0;
      0,0,0,0,0,1,0;
      0,0,0,0,0,0,1]; % Fx2 is a matrix to map any variable vector or matrix to
higher dimensional space matrix

X1=zeros(100,1);
Y1=zeros(100,1);
M1=zeros(100,1);
N1=zeros(100,1);
M2=zeros(100,1);
N2=zeros(100,1);

for i = 1:100
% Prediction Step
u1=u+Fx'*V; % mean state estimate of the system for current timestep based on mean
state vector of previous timestep
G=I+Fx'*J*Fx; % Jacobian matrix to locally linearize the non-linear function
E1=G*E*G'+R; % Error covariance prediction for current timestep based on error
covariance of previous timestep

delx1=u1(4,1)-u1(1,1); % Prediction of the distance of object 1 in x-direction
dely1=u1(5,1)-u1(2,1); % Prediction of the distance of object 1 in y-direction
del1=[delx1;
      dely1]; % Prediction matrix for distance of object 1
q1=del1'*del1;
z_hat1=[sqrt(q1);
       atan((dely1/delx1))]; % Prediction matrix for location of object 1
h1=(1/q1)*[-sqrt(q1)*delx1, -sqrt(q1)*dely1, 0, sqrt(q1)*delx1, sqrt(q1)*dely1;
         dely1, -delx1, 0, -dely1, delx1]; % Jacobian matrix of predicted location
matrix
z1=[s1(i,1);
    s1(i,2)]; % Sensor measurements for object 1
H1=h1*Fx1; % Mapping Jacobian matrix to high dimensional space matrix

% Correction Step
K1=E1*H1'*((H1*E1*H1'+Q)^(-1)); % Kalman gain
u=u1+K1*(z1-z_hat1); % Update mean state vector of the system
E=(I-K1*H1)*E1; % Update error covariance matrix of the system
```

```
% Store values of the mean state vector
X1(i,1)=u(1,1);
Y1(i,1)=u(2,1);
M1(i,1)=u(4,1);
N1(i,1)=u(5,1);
end

for i = 1:100
% Prediction Step
u2=uo+Fx'*V; % mean state estimate of the system for current timestep based on mean
state vector of previous timestep
G=I+Fx'*J*Fx; % Jacobian matrix to locally linearize the non-linear function
E2=G*Eo*G'+R; % Error covariance prediction for current timestep based on error
covariance of previous timestep

delx2=u2(6,1)-u2(1,1); % Prediction of the distance of object 2 in x-direction
dely2=u2(7,1)-u2(2,1); % Prediction of the distance of object 2 in y-direction
del2=[delx2;
      dely2]; % Prediction matrix for distance of object 2
q2=del2'*del2;
z_hat2=[sqrt(q2);
       atan((dely2/delx2))]; % Prediction matrix for location of object 2
h2=(1/q2)*[-sqrt(q2)*delx2, -sqrt(q2)*dely2, 0, sqrt(q2)*delx2, sqrt(q2)*dely2;
         dely2, -delx2, 0, -dely2, delx2]; % Jacobian matrix of predicted location
matrix
z2=[s2(i,1);
    s2(i,2)]; % Sensor measurements for object 2
H2=h2*Fx2; % Mapping Jacobian matrix to high dimensional space matrix

% Correction Step
K2=E2*H2'*((H2*E2*H2'+Q)^(-1)); % Kalman gain
uo=u2+K2*(z2-z_hat2); % Update mean state vector of the system
Eo=(I-K2*H2)*E2; % Update error covariance matrix of the system

% Store values of the mean state vector
M2(i,1)=uo(6,1);
N2(i,1)=uo(7,1);
end
```

Plot locations of robot and the two objects

```
figure(1)

plot(X1,Y1,'o')
grid on
hold on
plot(M1,N1,'o')
plot(M2,N2,'o')
hold off
xlabel('X (meters)')
ylabel('Y (meters)')
title('2D plane of vehicle motion')
legend('Position of Vehicle','Position of object 1','Position of Object 2')
```