Embodied AI

Assignment-1

Harshil Vagadia
2019CS10356
Himanshu Gaurav Singh
2019CS10358

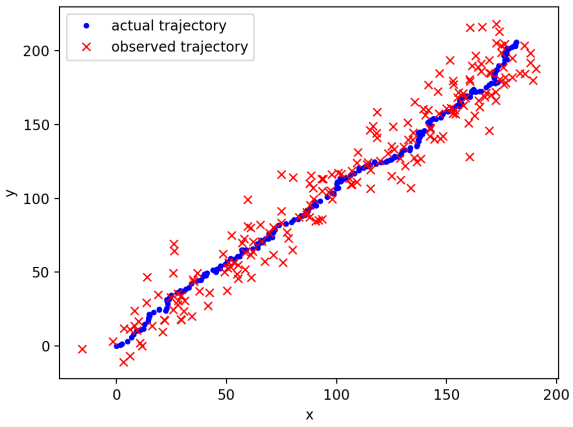
February 28, 2022

Contents

1 Kalman Filter

1.a Motion Model

The motion model is implemented in the Airplane model class in the code. Refer toe the code for the details. The observation model is implemented as specific functions in the code. The model is simulated with the initial position $(0, 0)$ and v_x, v_y as 1. A gaussian with standard deviation is 0.01 is used as prior for all four state parameters. The control value is kept to $[0, 0]$. The following is the plot.



Actual trajectory and Observed trajectory

1.b Kalman Filter

The model for estimation is as follows:

1. **State**

$$X_t = \begin{bmatrix} x_t \\ y_t \\ \dot{x_t} \\ \dot{y_t} \end{bmatrix} \tag{1}$$

x_t, y_t are the coordinates of the airplane and $\dot{x_t}, \dot{y_t}$ are velocities in the x and y direction respectively.

2. **Action(Control) model**

$$U_t = \begin{bmatrix} \delta\dot{x_t} \\ \delta\dot{y_t} \end{bmatrix} \tag{2}$$

$$X_{t+1} = A_t * X_t + B_t * U_t + \epsilon_r,$$

$$A_t = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad B_t = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \epsilon_r \sim N(0, R)$$

$\delta\dot{x_t}, \delta\dot{y_t}$ represents increments provided to the velocity.

3. **Observation model**

$$Z_t = \begin{bmatrix} \delta x_t^{'} \\ \delta y_t^{'} \end{bmatrix} \tag{3}$$

$$Z_t = C_t * X_t + \epsilon_q$$

$$C_t = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \quad \epsilon_q \sim N(0, Q)$$

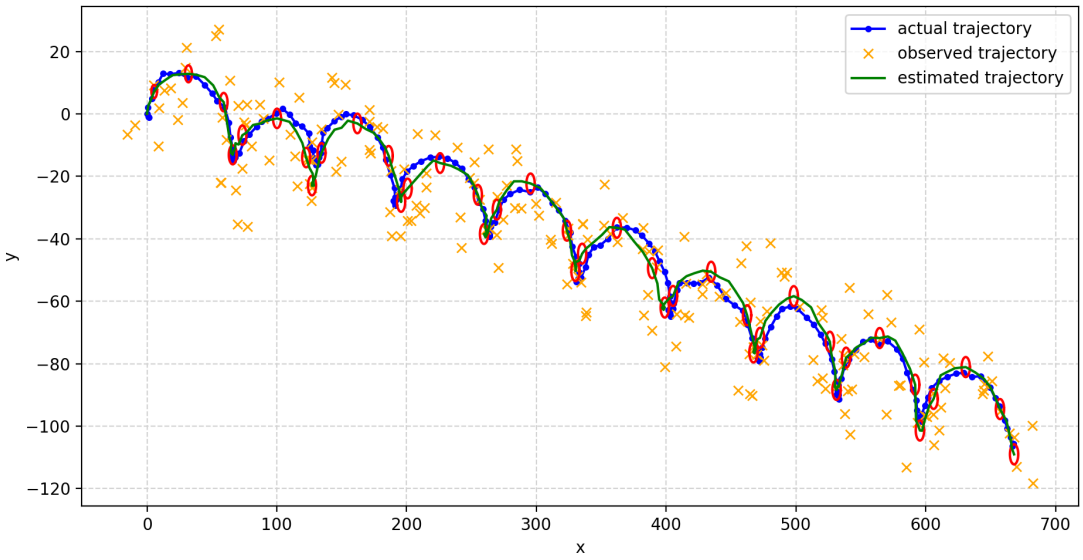
This refers to the position observation received from the sensors.

1.c Simulation results

The following control input value is taken for simulation.

$$U_t = \begin{bmatrix} \cos(t*\pi/10) \\ \sin(t*\pi/10) \end{bmatrix} \tag{4}$$

Rest of the values are same as suggested in the problem statement. The plot is as follows.

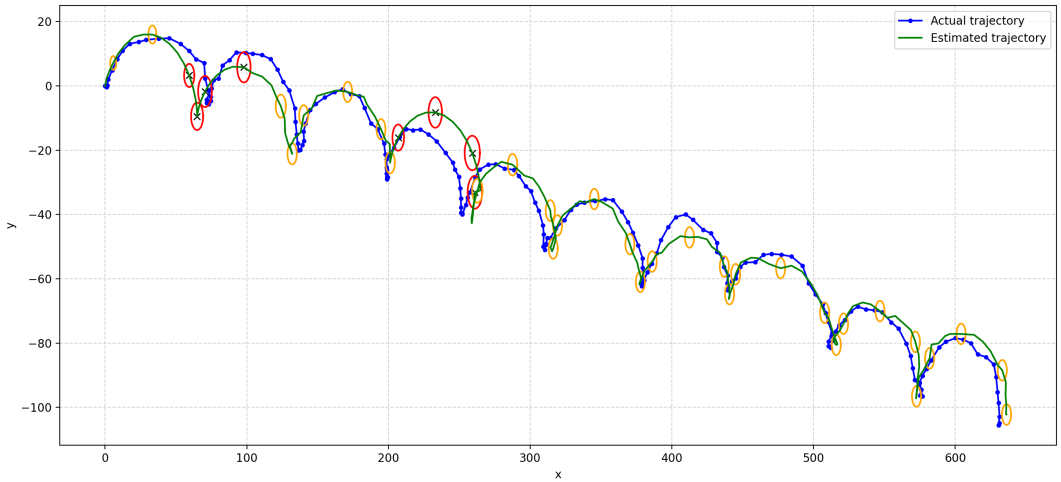


The red ellipses are the uncertainty ellipses marked at every fifth timestep for visual clarity

Observe that despite significant variance in the observations, the model is able to "filter" out the state quite efficiently.

1.d Loss of observation

On removing the sensor for timesteps $t = 10- > 30$ and $t = 60- > 80$, we get the following results. The implementation is done by breaking down the kalman filter algorithm into action step and observation step and thereafter applying the observation step only at desired timesteps.



The red ellipses are the uncertainty ellipses marked at points with no observations

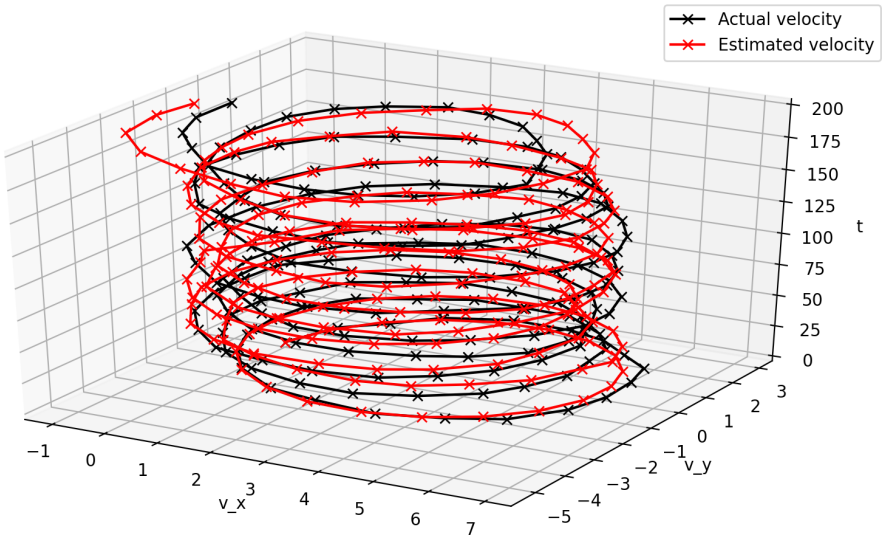
Observations

1. The accuracy of the estimation decreases in absence of observation data.

2. The standard deviations are larger for time steps at which observation data is unavailable. This is reasonable since we do not perform the correction step in the Kalman Filter algorithm at those timesteps. Thus, the decrease in variance caused due to it is not done.

1.e Velocity estimation

The estimated and actual velocities are as follows:



Observations

1. Observe that we are able to estimate the velocity reasonable well despite the fact that the sensor inputs only provide position measurements.

2. This is because $\dot{x_t}, \dot{y_t}$ are correlated to x_t, y_t. through the action model A_t. During the measurement incorporation half(second half) of the Kalman Filter, we update μ_t to $\bar{\mu_t} + K_t(z_t - C_t\bar{\mu_t})$. The rows of the kalman gain matrix K_t corresponding to the velocity parameters are not all zero, suggesting that the correlation of the velocity and position is being accounted.

1.f Data association

We found out this problem quite interesting and relevant. As an example, in a RADAR setting where, with several airplanes in the airspace and each of the updates appearing as a blip on the screen, it is important to associate each of the blip to the previous trajectories.

1.f.1 Approach

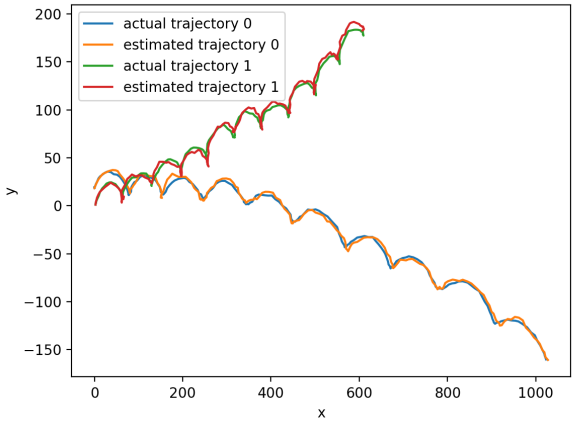
We have followed a greedy approach at each timestep. Suppose there are n airplanes with estimates (μ_i, Σ_i) after incorporating action at any timestep. Suppose the observations at the same timestep are Z_i. Observe that these observations are shuffled randomly and we need to assign each of them to an estimate. The objective is:

$$\texttt{argmax}_p\{\Pi_{i=1}^n \Pr(Z_{p(i)}|X_i)\}$$
where $X_i \sim N(\mu_i, \Sigma_i)$ and p is a permuation of $\{1, 2, ..n\}$

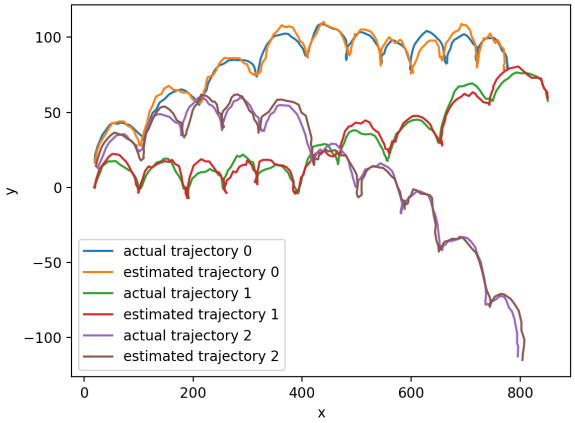
Maximising $\Pi_{i=1}^n \Pr(Z_{p(i)}|X_i)$ can be reframed as maximising the sum of log-likelihood corresponding to each estimate.
We can compute optimal p by iterating through each permutation and computing the sum of log-likelihoods. This has $O(n.n!)$ time complexity but works well for small number of objects to track.
However, this maximisation objective can be easily composed into a **maximum weight bipartite matching problem** that can be solved using the **Hungarian algorithm** using $O(n^3)$ time. In our implementation n is at most 4 or 5 hence the brute force method works well.

1.f.2 Results

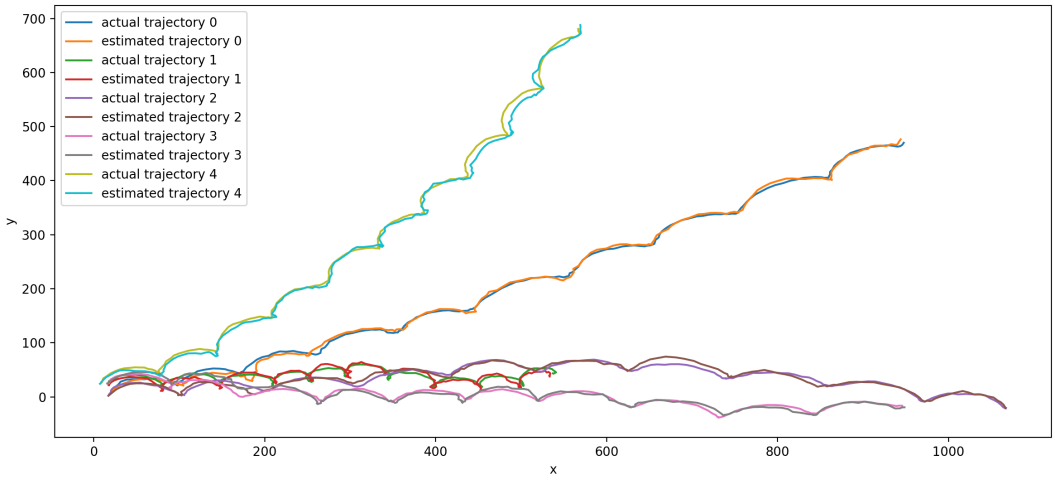
On implementing the algorithm, we get the following results. The initial positions of each of the plane is chosen uniformly randomly from the area $(0, 0)$ to $(25, 25)$ and the initial veolcity in both x and y directions are initialised uniformly randomly from $(0, 1)$. For $n = 2$ airplanes,



For $n = 3$ airplanes,



For $n = 5$ airplanes,



Observe that with the given data association strategy, we are able to provide reasonably accurate state estimations for n atleast upto 5 airplanes.

2 Incorporating Landmark Observations

2.a Model

Observe that the distance from a landmark is a non linear observation in terms of the state. Formally we can write,

$$Z'_t = h_t(X_t) + \epsilon \text{ where } h_t(X_t) = ||X_t - landmark||$$

This can be incorporated using the **Extended Kalman Filter** algorithm discussed in class. We linearize the above equation around the mean of the current estimated state.

$$h_t(X_t(=(x,y))) = \sqrt{(x-l_0)^2 + (y-l_1)^2}$$

$$H = \left[\partial H_1/\partial x, \partial H_1/\partial y, 0, 0\right]$$

$$\partial H_1/\partial x = (x-l_0)/h_t, \partial H_1/\partial y = (y-l_1)/h_t$$

$$Z_t \approx h_t(\bar{\mu_t}) + H_t(X_t - \bar{\mu_t})$$

The third and fourth entries in H are zero since the distance will be partially independent of the velocity.

2.b Implementation

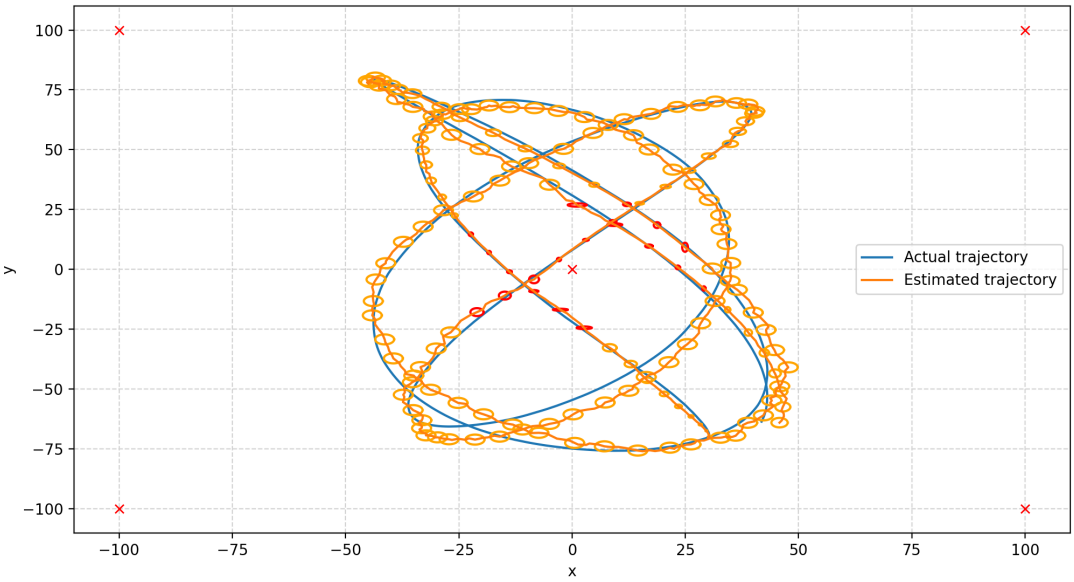
The `Airplane Model` class in the code is extended to include EKF.

2.c Simulation Results

The airplane model is simulated with landmarks with most parameters as mentioned in the problem statement. However, for better clarity and visualisation of the non linear feature, the initial state of the plane is tweaked to the following parametes:

$$(x_0, y_0) = (30, -70), (\dot{x_0}, \dot{y_0}) = (0.1, 0.1)$$
$$U_t = \{-10^{-3}x_t, -5 * 10^{-4}y_t\}$$

U_t has been chosen in this fashion in order to ensure that it comes within the range of landmarks several times in its trajectory. The results are as follows:

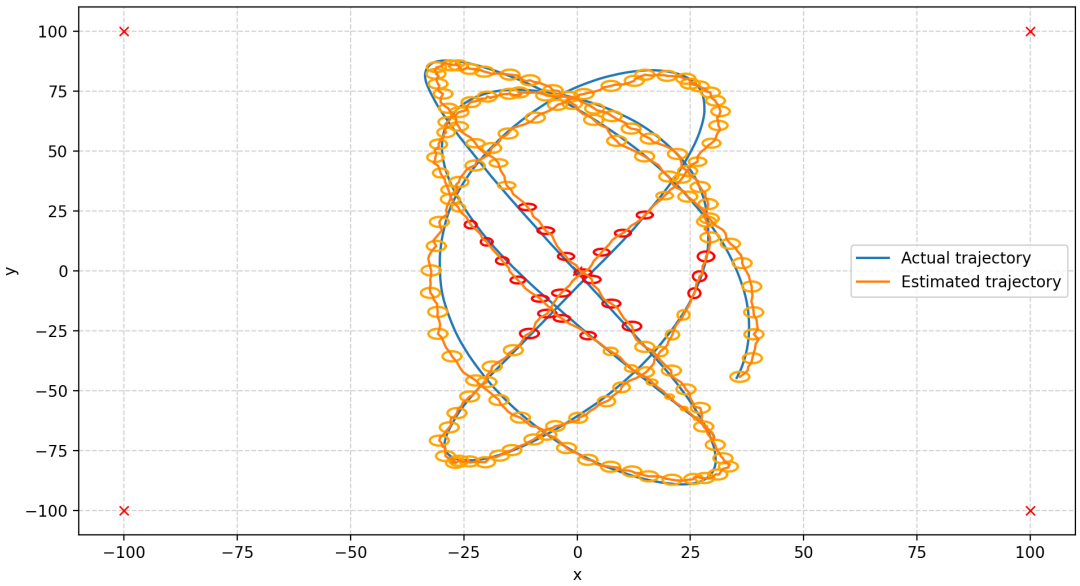


The orange ellipses show standard deviations for points outside landmark ranges
The red ellipses show standard deviations for points within landmark ranges

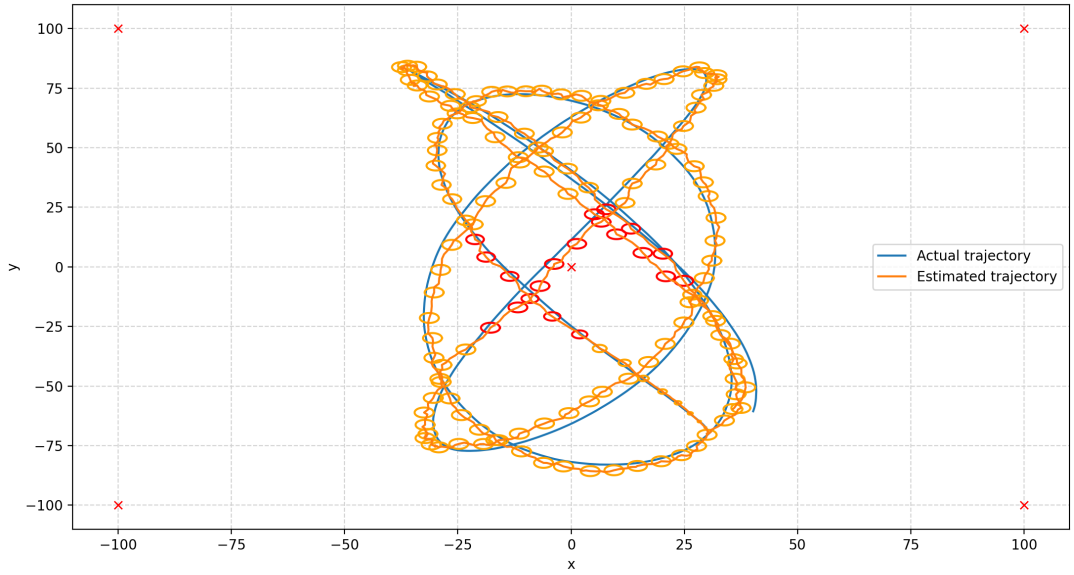
Observe that near the landmarks, the estimated and actual trajectories are closer together and the red ellipses are smaller than the orange ones. This is reasonable since in regions close to landmarks, we recieve extra information that can be used to further correct our estimation of the actual position which improve accuracy and reduces the variance of the estimate.

2.d Increasing landmark data variance

On increasing the standard deviation from 1 to 5. We get the following plot.



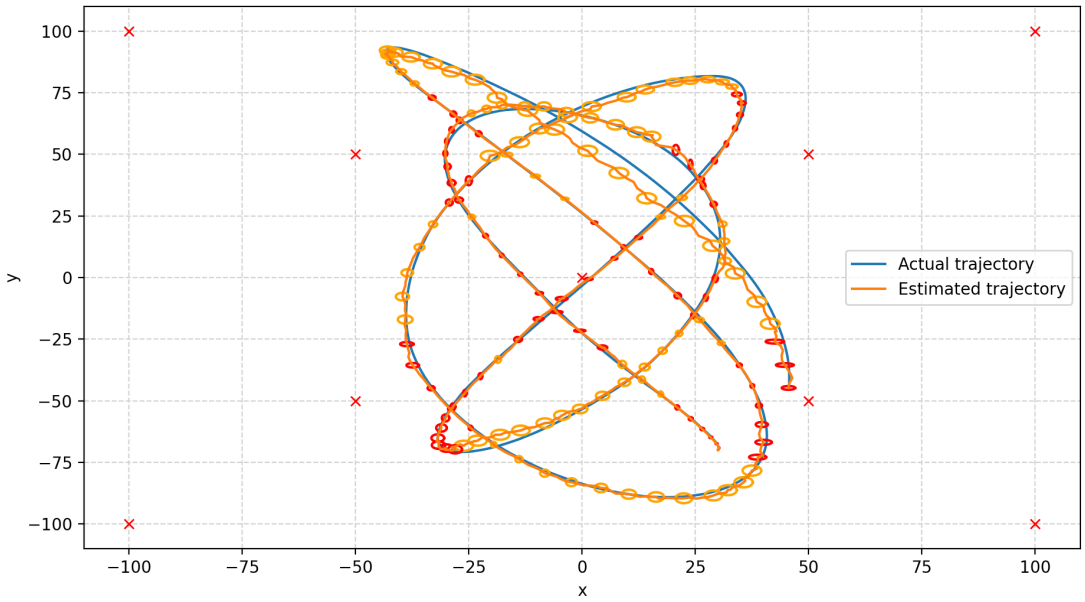
On increasing it to 20, we get



Observe that the as the standard deviation of the landmark measurement rises, the estimated trajectory deviates more than the actual. Also, the standard deviation of the estimates also increases. This is reasonable since greater uncertainty in measurement is likely to be reflected in greater uncertainty in the prediction.

2.e Increasing landmarks

Apart from the original, I planted landmarks at $(50, 50), (-50, 50), (-50, -50), (50, -50)$. With the same initial state as above, the results are as follows.



Observe that the estimated trajectory now more correctly predicts the actual. Also, the number of red ellipses has increased and all(both red and orange) the ellipses have become smaller in size. This is reasonable since with more instances of extra information available, the accuracy will increase which will propagate to less error for states in which extra information is not available.

3 Robot Localisation

3.a Simulation and Estimation

The robot moves around the grid by choosing randomly a feasible direction to move in. Localisation is done by modelling system as hidden markov model. The state evolves with time without any external control (robot moves randomly) and noisy sensor readings are available.

$X_t = (x, y)$ and $E_t = (e_1, e_2, e_3, e_4)$ where x, y is the position of robot in grid and $e_i \in \{Far, Close\}$ for each direction North, East, South and West respectively

The transition probabilities are defined as

$$P(X_{t+1}|X_t) = 1/N(X_t) \text{ if } |X_{t+1} - X_t| = 1 \text{ else } 0 \text{ where } N(X_t) \text{ is the number of valid neighbours of } X_t.$$
$$P(E_t|X_t) = \Pi_{i=1}^{4} P(e_i|x_i) \text{ where } x_i \text{ is the distance to the wall in } i^{th} \text{ direction.}$$
$$P(e_i|x_i) = 1 - (x_i - 1)/(R_{max} - 1) \text{ if } x_i < R_{max} \text{ else } 0$$

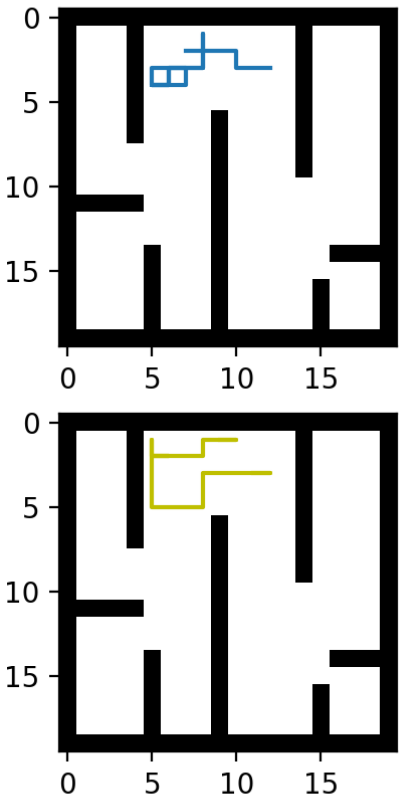
Shown below is simulation of the robot and its estimated position. Red dot denotes the actual position of the robot. Yellow dot denotes the estimated position ($\hat{X}_t$) of the robot via the markov model. Red dot will not be visible when estimated and true position overlap.

$$\hat{X}_t = argmax P(X_t|E_{1..t})$$

Simulation

As seen from the simulation, the estimated position is mostly somewhere around the actual position. The estimator is able to figure out the general area of the robot.

The figure below shows actual path and most likely path of the robot given all the sensor observations. Most likely path was calculated by Vitterbi algorithm. Most likely path is in the same region as the actual path. The estimated position sometimes strays far away from the actual position, but the most likely path always remains close to the actual path. This is due to the fact most likely path has all the observation available while estimated position has observations only upto that time instant available.



Blue Line: Actual Path, Yellow Line : Most Likely Path

Given below is the actual path, most likely path and all the sensor observations for the above figure and simulation.

Path: [(3, 12), (3, 11), (3, 10), (2, 10), (2, 9), (2, 8), (3, 8), (3, 7), (4, 7), (4, 6), (4, 5), (4, 6), (4, 5), (3, 5), (3, 6), (3, 7), (3, 6), (4, 6), (3, 6), (3, 7), (3, 8), (2, 8), (1, 8), (2, 8), (2, 7)]
Most Likely Path: [(3, 11), (3, 12), (3, 11), (3, 10), (3, 9), (3, 8), (4, 8), (5, 8), (5, 7), (5, 6), (5, 5), (4, 5), (3, 5), (2, 5), (2, 6), (2, 5), (1, 5), (2, 5), (2, 6), (2, 7), (2, 8), (1, 8), (1, 9), (1, 10), (1, 9)]
Observations: [(0, 0, 0, 0), (1, 1, 0, 0), (1, 1, 0, 0), (1, 1, 0, 0), (1, 0, 1, 0), (0, 0, 0, 0), (0, 0, 0, 1), (0, 0, 0, 0), (0, 0, 0, 0), (0, 0, 0, 1), (0, 0, 0, 1), (0, 0, 0, 1), (0, 0, 0, 1), (1, 0, 0, 1), (1, 0, 0, 0), (0, 0, 0, 1), (1, 0, 0, 1), (1, 0, 0, 1), (0, 0, 0, 0), (1, 0, 0, 0), (0, 0, 0, 0), (1, 0, 0, 0), (1, 0, 0, 0), (1, 0, 0, 0)]

3.b Belief Estimation

The simulation below shows the belief of the markov model after each observation. Darker squares indicates higher probability of robot being in that square. Red dot shows the actual position of the robot. Initially the belief is spread over multiple regions according to the observations. This regions are nearly symmetric. As more data is acquired, the belief localises to the region of the actual robot.
Note that it is difficult to get the exact position of the robot as the sensors only tells if the walls are far/close and not the exact distance to the wall. Thus the belief is generally spread over a region of few squares.

Simulation

3.c Error Measurement

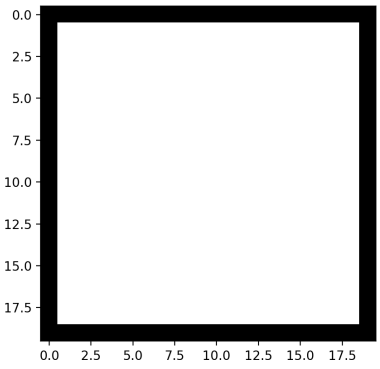
Manhattan distance is used to define the error between two points. In a single run, the error between actual and estimated position is summed for all time stamps. This is defined as Position Estimation Error. The error between two paths is defined as the sum of error between corresponding points in the paths. Most Likely Path Error is defined as error between actual path and most likely path. The error was computed for 50 runs and the mean and standard deviation were noted.

Position Estimation Error Mean: 94.64
Position Estimation Error Std Dev: 75.17781913434254
Most Likely Path Error Mean: 98.76
Most Likely Path Error Std Dev: 115.46384892321608

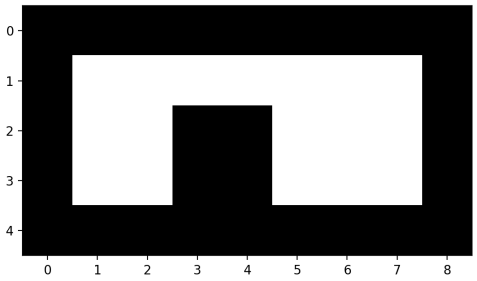
3.d Changing R_{max}

Setting $R_{max} = 1$ will result in no relevant observations. All the measurements will definitely by Far, and hence there is no scope of any localisation.
Setting $R_{max} = 10$ increases the range of sensors. The effect of this depends on the structure of layout. In sparse/large layout where walls are far away, increasing the range will bring in new measurements thus helping in better estimation. For closed/small layout, the walls are close enough. Increasing the range decreases the quality of measurements of close walls (the gradient of probability decreases). Hence the estimation will be worse.

Here is a comparison between between the effects of $R_{max} = 5$ and $R_{max} = 10$ for two layouts, open and closed.



(a) Open Layout



(b) Closed Layout

1. Open Layout, $R_{max} = 5$
   Position Estimation Error Mean: 88.28
   Most Likely Path Error Mean: 97.22
2. Open Layout, $R_{max} = 10$
   Position Estimation Error Mean: 64.8
   Most Likely Path Error Mean: 58.6
3. Closed Layout, $R_{max} = 5$
   Position Estimation Error Mean: 25.04
   Most Likely Path Error Mean: 15.2
4. Closed Layout, $R_{max} = 10$
   Position Estimation Error Mean: 47.18
   Most Likely Path Error Mean: 57.06

As seen from the above data, increasing R_{max} on open layout decreases error, while it increases on closed layout.

3.e Changing Layout

As seen from belief estimation, the belief is initially spread over multiple region. The asymmetry in layout will force the belief in one region or another eventually. However, if the regions were completely symmetric, it is impossible to determine exactly in which region the robot is. Below is an example of a symmetric layout, and the belief cannot figure out exactly in which square the robot is.

Hard Localisation Simulation

Similarly, having an extremely asymmetric layout with more walls will result in very quick localisation. Below is an example of such a layout were localisation is very easy.

Easy Localisation Simulation