Simulation of Bayes and Kalman Filters

Sarthak Behera (2018CS10384)
Pratik Pranav (2018CS10368)

March 2021

Introduction

Interaction with environment is an important aspect of autonomous systems. State estimation is a crucial component of it wherein the robot takes measurements to ascertain its current state given the measurement and past beliefs. Knowing the current state helps the robot in planning its action for the future. In this assignment, we see two popular filtering techniques :- Bayes Filter and Kalman Filter and their associated properties.

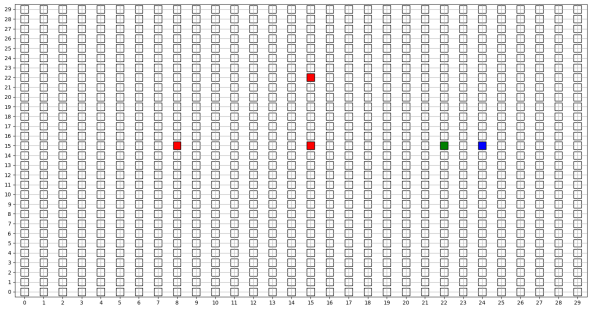

Figure 1: Simulation of Robot Motion

1 Aerial vehicle taking discrete steps

Here we have an aerial vehicle and four sensors whose characterisation is as given in question. We try to estimate various state probabilities in different parts.

A Simulation of robot motion

Robot motion simulation is done using the motion model and associated probabilities as given in the question. The sensor model is also implemented using the chebyshev distance as shown in the question. The observations are stored and displayed as an animation. Figure 1 shows how we have animated the simulation. You can see when sensor are not detecting the they are red while when they are detecting the robot, they are green. Blue square represent the location of robot.

B Filtering the robot's position

Now we estimate the robot's position given the sensor measurements. For this, we use the Bayes Filter to determine various probabilities at each time step and shade the box based on the probability values in grayscale. We notice that as the probability of going *right* and *up* is higher, the robot tends to move towards it and the grayscale shade map also reflects this fact when all sensors return negative measurements for a stretch of time. The accompanying animation helps visualise this phenomenon. Figure 2 is a screenshot of animation at a particular instance. Additionally violet squares shows the estimated location of the robot at a particular instance.

The likelihood distribution over states initially quite distributed over the entire grid. However, with more and more sensor observations they tend to estimate the robot's location more and more accurately. Also, the sensor provided in our case aren't perfect which is also impacting the performance of the filtering algorithm.

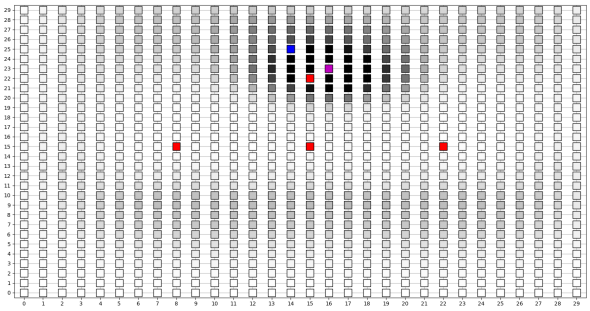

Figure 2: Filtering the Robot Motion

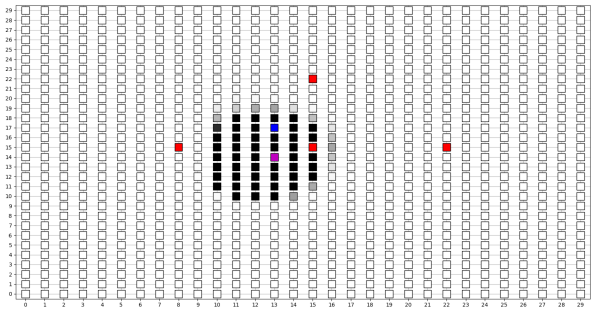

Figure 3: Smoothing given observations till time T

C Smoothing given observations till time T

We perform smoothing to rectify our belief of past states given observations till time T. Smoothing is done through the *forward-backward algorithm*. We make two passes through the set of observations to rectify our probabilities. The forward pass is basically the filtering step done in the previous question. The backward pass is basically the probability of observing the later measurements given that we were at a state at certain time. Now, the smoothed probabilities is simply the multiplication of the above two passes of respective states at certain time. Figure 3 shows a screenshot of the animation developed.

The likelihood distribution over states in case of smoothing is more concentrated towards robots location and worked quite better than filter which is expected as we have all the distribution of future data. However, when sensor doesn't detects anything through out the entire motion the likelihood tends to be concentrated towards right top which is expected due to higher probabilities of those directions in motion model.

D Difference between smoothed and true paths

Figure 4 show the plot for Manhattan distance between actual path and estimated path for part B and C. Smoothing tends to produce more accurate result than filtering which is reminiscent from the fact that error for part B is almost every time higher than that of part C.

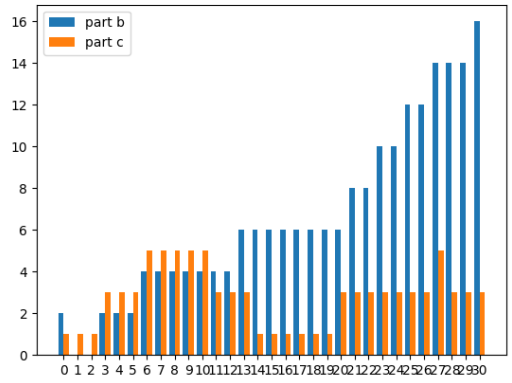

Figure 4: Manhattan Distance between actual path and estimated path in part C and D

For finding the errors, we have used most likely paths for path B and C and calculated the error as Manhattan distance between points on most likely path and robot's location at each time stamp. Most likely path is calculated using *Viterbi Algorithm*.

The likelihood over states for smoothing seems to be more concentrated towards the location of robot while the likelihood over states for filtering sometimes isn't that much accurate. This also causes smoothing to produce more better results than filtering.

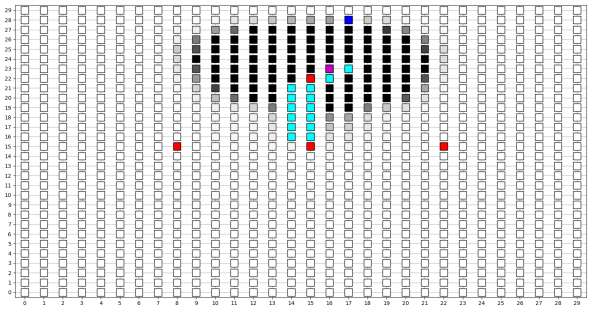

Figure 5: Most likely path

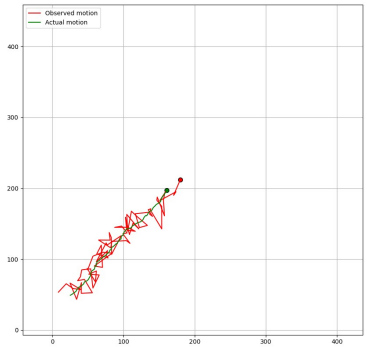

Figure 6: Figure: Motion and sensor model simulation

E Predictive likelihood over future locations

Now we need to predict robot's future location over next T time steps. For prediction part, we used motion model over the current state estimated till that instance. After multiplying, the likelihood over the state space seemed biased towards right top region.

We can see with time the observation tends to be more fuzzier. Hence, observation obtained at the end of T=10 is much better than for T=25. This is on expected terms as the without sensor observation the model predicts just on basis of motion model which just increases the biasness towards directions with better probability.

F Most-likely path given set of observations

To determine the most-likely path, we used the *Viterbi algorithm*. Viterbi algorithm is basically a dynamic programming algorithm for finding most likely sequence of hidden states. Figure 5 shows the screen shot of the animation. The cyan colored squares are squares included in most likely path and violet colored square shows the head of most likely path at that time instance.

2 Airplane motion under radar observations

The airplane is moving in 2D (assuming height is constant) and we get radar observations at each time step. The airplane is controlled by velocity increments at each time step.

A Airplane motion and sensor simulation

The motion model and the sensor model for the airplane is implemented and the animation depicting the trajectory of estimated and actual path is actuated.

B Kalman Filter as the limit of Bayes Filter

Bayes Filter reduces to Kalman Filter when the variables are Gaussian (as in our case) and the transition as well as sensor model is linear. Kalman Filter is more suited to the continuous spaces where keeping probability tables becomes highly non-practical so we impose the condition of probability distribution being normal. To show the difference we have actually done both the bayes and the kalman filter with uncertainty ellipses (uncertainty ellipses in bayes is just adjacent squares such that probability reaches threshold). Basically we are sort of having a exponentially decaying distribution based on manhattan distance. Uncertainty ellipses (in this case circles due to equal uncertainty in x, y) denotes the re-

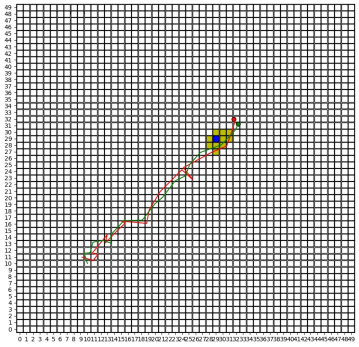

Figure 7: Figure: Bayes Filter with approximate uncertainty ellipse in discrete space

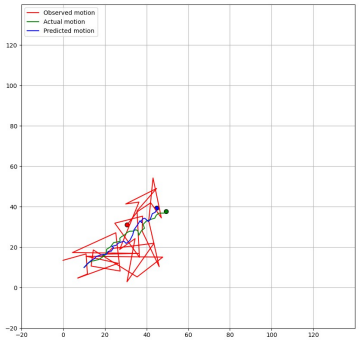

Figure 8: Figure: Kalman Filter

gion where the true position of the airplane is most likely (as shown in next question).

C True, Observed and Estimated trajectories

Here we show the true, observed and the estimated trajectories of the airplane along with uncertainty ellipse for the estimated trajectory. The animation helps in visualising various aspects of the Kalman filter. Note that the ellipse is getting smaller (until it sort of converges) which indicates that we believe the measurements to a certain degree. Rapidly decreasing ellipse sizes indicates our strong belief in measurement while if the ellipse size increases implies we are uncertain about our measurement.

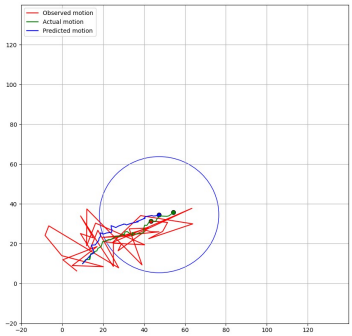

Figure 9: Figure: Uncertainty ellipses

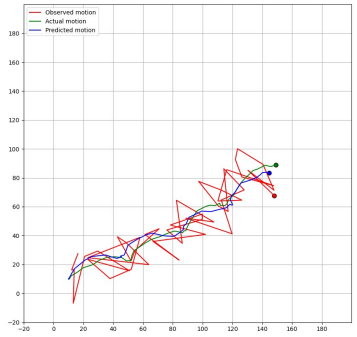

Figure 10: Figure: Sinusoidal control policy

D Sinusoidal control policy

Now we implement the control policy as given in question and as time steps are discrete, we take multiples of a constant for each time step (in code this constant is *Omega*). Plot for the true and estimated trajectory as well as the error between them helps visualise the working of Kalman Filter.

E Varying the uncertainty of sensor model

The covariance matrix of the sensor model affects the Kalman gain factor (or to be precise, it is involved in an

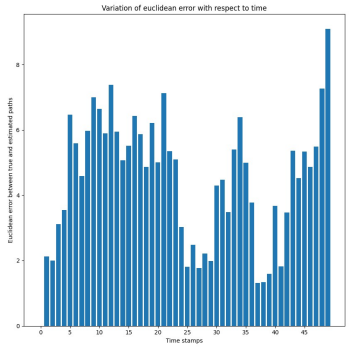

Figure 11: Figure: Euclidean error bar plot

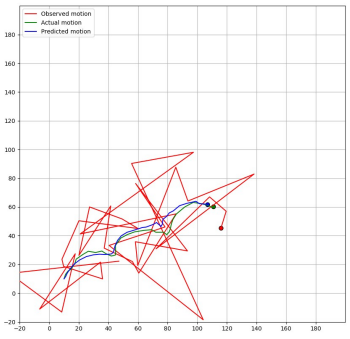

Figure 12: Figure: Sensor uncertainty variance = 500

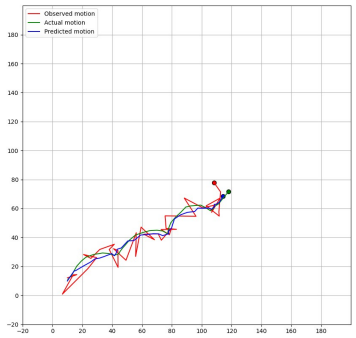

Figure 13: Figure: Sensor uncertainty variance = 20

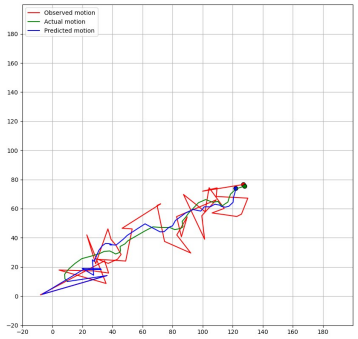

Figure 14: Figure: Initial prior uncertainty s.d = 100

inverse). Kalman gain is the degree of belief, we have on the measurement values so increasing the uncertainty in sensor model leads to decrease in our belief in observed values. Think of observations as rectifying our belief for the state of the airplane. We get some initial belief based on motion model and the observed value pulls us in that direction. More certain we are about our measurement, stronger the pull towards observed value. Thus if uncertainty is high, this pull is weak and we barely change our belief while if uncertainty is low, we move strongly towards this measurement.

F Varying uncertainty in prior belief

Kalman filter is quite robust in the sense that high uncertainty in prior belief doesn't seem to have adverse effects on its working. The uncertainty ellipse rapidly gets smaller (until it sort of converges). Initially the estimated trajectory is very shaky but then it stabilizes as the belief in current state is ascertained.

G Dropping out some observation values

As some observations drop out, we don't do the measurement update step in the Kalman Filter algorithm. Thus the covariance matrix increases without the measurement update and this is evident from the uncertainty ellipses in the animation. When we don't do the update step, the ellipses rapidly grow and then rapidly shrink when measurements come again.

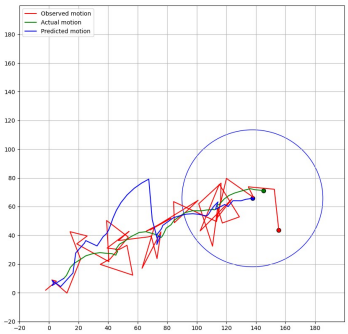

Figure 15: Figure: Uncertainty ellipse after drop period

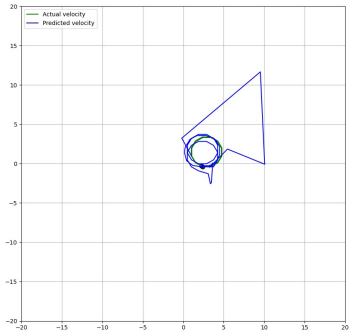

Figure 16: Figure: Tracking velocity after drop period (s.d $= 10$)

H Keeping track of the velocities

Yes, we can track the velocities. But this tracking is made due to correlation between x and v_x and similarly, y and v_y. It was observed that when sensor observations dropped, the uncertainty ellipses of velocity vector stayed same in size which was in contrast to rapid increase in motion uncertainty ellipse (due to absence of measurement update step). This was due the fact that the motion noise is high while velocity noise is very low (noted from R matrix). Thus the uncertainty ellipse of velocity increases but only very slightly and it seems as if it remains the same in size. Now one thing to note is that once the sensor observations come in, the uncertainty ellipse rapidly becomes smaller and the true velocity value is outside the the uncertainty ellipse of estimated velocity. This leads us to conclude that velocity isn't being directly tracked which is in contrast to position vector which is being directly tracked (as sensor measurements give approximate position) and it is also noted that true position remains inside uncertainty ellipse of estimated position for each time instance. Now we note that the estimated velocity actually does converge to the true velocity after continuous time steps of getting the sensor measurements. Thus, velocity is being indirectly tracked through the position itself and once we ascertain the position, the velocity automatically is computed correctly. This strong correlation between position and velocity is due to the fact that velocity is simply the derivative of position and difference between positions in consecutive time steps gives us a fair idea about the velocity. Note the above observations were taken when initial prior standard deviation was 10. If the initial prior was taken to be 0.01 (as in question) then the velocity already converges.

I Data Association from multiple measurements

First, for this question we need to first find a *Data Association* strategy. *Data Association* is a way of matching observations in a new frame to a set of tracked trajectories. It generally involves four steps namely, prediction, gating, data association and updates. In prediction, we predict the next position on the basis of current estimation and control variables. Gating is a method for determining the possible matching observations. For this part, we have assumed each of the observations taken at next instant can be likely transition with relative cost as the distance between predicted point and observations taken. For data association part, we have used *Hungarian Algorithm*. It is a combinatorial optimization problem which solves the assignment problem in polynomial time which eventually gives the required matching for each of the paths. Lastly, update step involves updating the mean and covariance matrix using the observations obtained above

We tested strategy upon several cases when the distance between starting points is large, medium, small and finally, when both have the same starting points. For the first three cases, almost every time out data

association strategy worked perfectly. However, whenever the starting points are same. Sometime, It indeed happens that both the agent might differ in their paths sometime, but they tend to return to their normal path immediately.

Yes, this strategy can indeed be extended to 4-5 agents as Hungarian algorithm generalizes number of agents provided. Also, rest of the steps can also be extended for larger number of agents.