CSE510 : Robotic Algorithms

Final Project

Comparative Analysis of Particle filtering And Kalman Filtering

Submitted by :
Vaibhav Krishna Irugu Guruswamy
Ubit: vaibhavk
Person #: 50060408

Problem Description:

This problem we are trying to solve in this project is related to robotic mapping. Robotic mapping is that branch of one, which deals with the study and application of ability to construct map or floor plan by the autonomous robot and to localize itself in it. Particle filtering and Kalman filtering are extensively used as very efficient techniques to localize and track objects in a very large environment without a lot of complexity.

The Kalman filter operates recursively on streams of noisy input data to produce a statistically optimal estimate of the underlying system state. Particle filters are the sequential (online) analogue of Markov chain Monte Carlo batch methods and are often similar to importance sampling methods.

The project analyses the importance of particle and Kalman filtering, it also enumerates the major differences in using these techniques to localize a robot in an identical environment which is planar and bounded environment.

Background:

1. Recursive Bayesian estimation :

A Bayes filter is an algorithm used in computer science for calculating the probabilities of multiple beliefs to allow a robot to infer its position and orientation. Essentially, Bayes filters allow robots to continuously update their most likely position within a coordinate system, based on the most recently acquired sensor data. This is a recursive algorithm. It consists of two parts: prediction and innovation. If the variables are linear and normally distributed the Bayes filter becomes equal to the Kalman filter.

In a simple example, a robot moving throughout a grid may have several different sensors that provide it with information about its surroundings. The robot may start out with certainty that it is at position (0,0). However, as it moves farther and farther from its original position, the robot has continuously less certainty about its position; using a Bayes filter, a probability can be assigned to the robot's belief about its current

position, and that probability can be continuously updated from additional sensor information.

Recursive estimation deals with the problem of extracting information about parameters, or states, of a dynamical system in real time, given noisy measurements of the system output. Recursive estimation plays a central role in many applications of signal processing, system identification and automatic control.

Our interest in these problems stems from the airborne applications of target tracking, and autonomous aircraft navigation using terrain information. In the Bayesian framework of recursive estimation, both the sought parameters and the observations are considered as stochastic processes. The conceptual solution to the estimation problem is found as a recursive expression for the posterior probability density function of the parameters conditioned on the observed measurements.

2.Kalman Filter - a recursive Bayesian filter for multivariate normal distributions :

The Kalman filter, also known as linear quadratic estimation (LQE), is an algorithm that uses a series of measurements observed over time, containing noise (random variations) and other inaccuracies, and produces estimates of unknown variables that tend to be more precise than those based on a single measurement alone.

The Kalman filter has numerous applications in technology. A common application is for guidance, navigation and control of vehicles, particularly aircraft and spacecraft. Furthermore, the Kalman filter is a widely applied concept in time series analysis used in fields such as signal processing and econometrics.

The Kalman filter uses a system's dynamics model (e.g., physical laws of motion), known control inputs to that system, and multiple sequential measurements (such as from sensors) to form an estimate of the system's varying quantities (its state) that is better than the estimate obtained by using any one measurement alone. The Kalman filter keeps track of the estimated state of the system and the variance or uncertainty of the estimate.

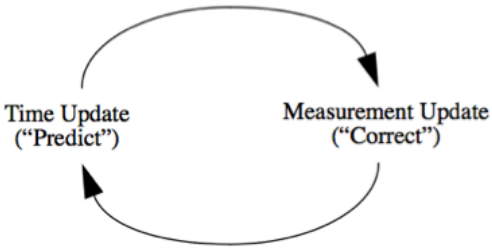
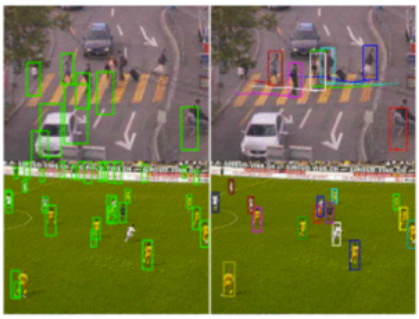


Figure 1-1. The ongoing discrete Kalman filter cycle. The *time update* projects the current state estimate ahead in time. The *measurement update* adjusts the projected estimate by an actual measurement at that time.

3.Particle Filter :

A particle filter, also known as a sequential Monte Carlo method (SMC), is a sophisticated model estimation technique based on simulation. Particle filters are usually used to estimate Bayesian models in which the latent variables are connected in a Markov chain, but typically where the state space of the latent variables is continuous rather than discrete, and not sufficiently restricted to make exact inference tractable (as, for example, in a linear dynamical system, where the state space of the latent variables is restricted to Gaussian distributions and hence exact inference can be done efficiently using a Kalman filter). In the context of HMMs and related models, "filtering" refers to determining the distribution of a latent variable at a specific time, given all observations up to that time; particle filters are so named because they allow for approximate "filtering" (in the sense just given) using a set of "particles" (differently weighted samples of the distribution).

The particle filter aims to estimate the sequence of hidden parameters, xk for k = 0,1,2,3,..., based only on the observed data yk for k = 0,1,2,3,.... All Bayesian estimates of xk follow from the posterior distribution p(xk | y0,y1,...,yk). It is a technique for tracking the state of a dynamic system modeled by a Bayesian network.



Robust Tracking –by –detection using
a detector confidence particle filter

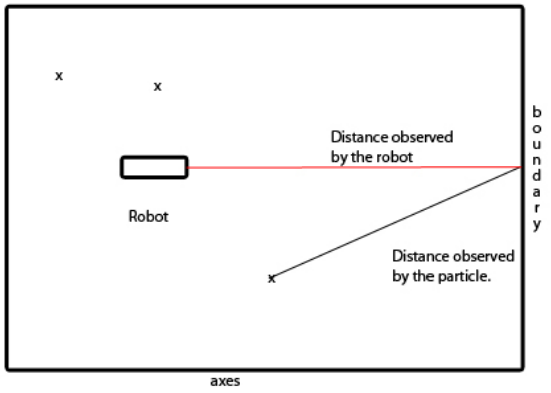
Approach:

Abiding with the initial project progress report which states “we solve the localization problem using these filters. Since the only observation available is a single laser beam, we consider the measurements made with this observation and compare the behaviors and characteristics of both filters using simulation.”, we analyze the ways in which Kalman Filter and Particle Filter localize a robot in a planar environment.

In the problem description it is stated the the only available observation to the robot at any point of time is the single laser beam that measures range between the robot and an obstacle directly in front of it. We start with a 5*5 grid world, which is basically the planar environment where we localize the robot based on the observations we make.

In the case of particle filtering, I have chosen 500 particles to localize the robot’s position with precision. Initially, all the particles are having equal belief and the belief is represented by weight array in the source code. The robot starts at a random position. I have run the algorithm for many different configurations, but at 20 time steps, the particles seemed to be converged enough and localized the position of the robot with considerable precision. The algorithm I have implemented works as follows, for each of the 20 time steps, the robot takes a random action, and moved 0.2 towards the direction in which it takes the action. Now, when the robot moves in a direction, there is also certain process noise involved in this. This process noise is a Gaussian noise with a variance of 0.2. The particles also move 0.2 in the same direction as the robot moves in. The robot makes an observation using it’s laser beam. The observation is also affected by a Gaussian noise with variance of 0.25. The distance

from the robot to the boundary is measured with this observation. A sample diagram is below .



Now, the distances measured by the robot as well as all the particles is sent processed with a Gaussian kernel which is implemented as.

```
      def GaussianKernel(v1, v2, sigma):
            d=math.fabs(v1-v2)
            return ((1/math.sqrt(2*pi*(sigma))))*exp((-
1)*(math.pow(d,2)/(sigma*2)))
```

After each time step, we normalize the beliefs of each of the particles and this process continues for 20 time steps and thus the simulations end.

In the case of Kalman filtering, I have used matrices for positions and estimations just to make the computations a bit easier and less complex. The state of the robot corresponds to the matrix which has [x y x_vel y_vel]. The robot starts at a random position and it makes random moves at each time step. Here the number of time steps is chosen as 15, the algorithm I have implemented, did a fairly good job in localizing the robot at very early stages it self, this is clearly evident from the graphs presented in the last section. Along with the robot position, we also maintain an estimation matrix which estimates the position of the robot at each time step. The estimation matrix isn't affected by any process noise , but the robot movement is affected by a Gaussian noise with variance 0.2. In the measurement update where the corrections are made, we use Kalman gain as described in the code as follows.

```
      inver= np.linalg.inv(np.dot(C,np.dot(P,Ct))+Ez)
            kalg = np.dot(P,np.dot(Ct,inver))
                        est_temp = all_rob_pos_arr[T]-
np.dot(C,all_est_arr[T])
            all_est_arr[T] = all_est_arr[T]+ np.dot(kalg,est_temp)
            init_est=all_est_arr[T]
```

Simulation:

Note: All the simulations are run and results are obtained on a 2.2GHz Intel Core i7 processor, 4GB 1333 MHz DDR3 RAM.

Both Particle filter and Kalman filter simulations are implemented in python, using the ipython tool kit.

Modules used:
Matplotlib – for plotting graphs
Numpy – for numerical calculations
Time – for time comparisons.

Configuration of particle filter simulation :

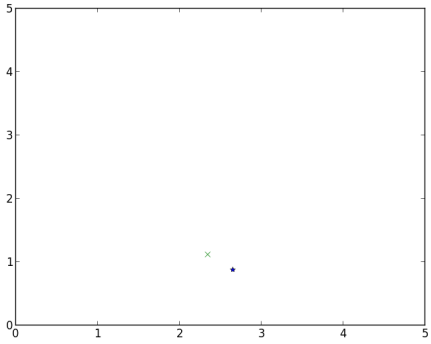
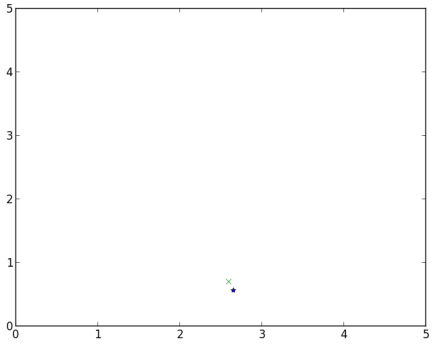
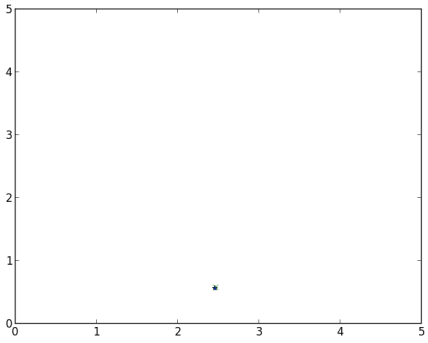
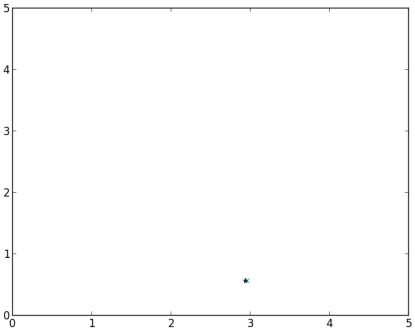
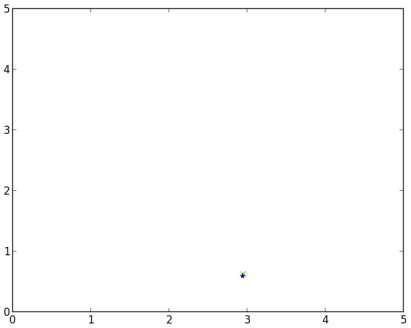
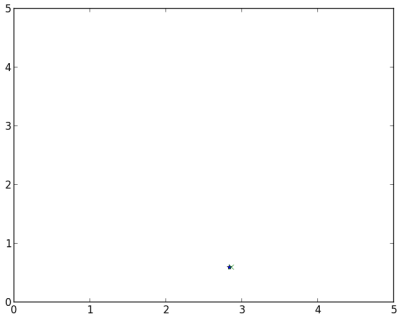
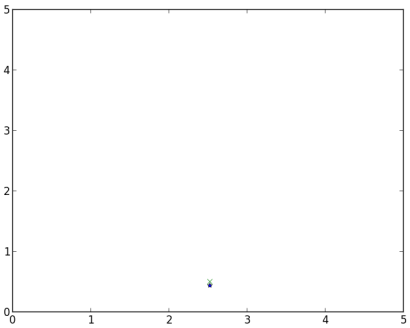
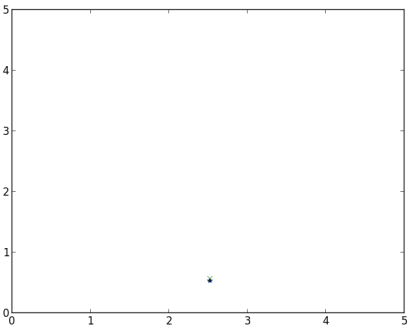
Number of time steps : 20
Number of particles : 500
Gaussian process noise with 0 mean and 0.2 variance.
Gaussian observation noise with 0 mean and 0.25 variance.
Environment : 2D
Boundaries : 5 on X axis and 5 on Y axis.
Process update of 0.2
Robot position is denoted by 'o' and the particles are denoted by 'x'

Configuration for Kalman filter simulation :

Number of time steps : 15
Gaussian process noise with 0 mean and 0.2 variance.
Process update of 0.2
Environment : 2D
Boundaries : 5 on X axis and 5 on Y axis.
Robot position is denoted by '' and the estimate is denoted by 'x'*

Screenshots of python –code executing the algorithms:

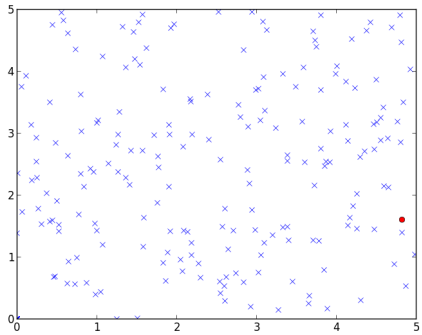
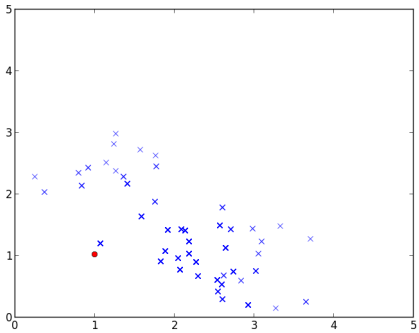
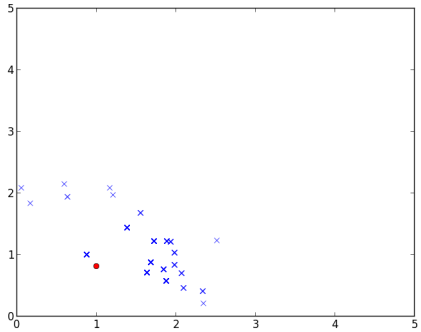
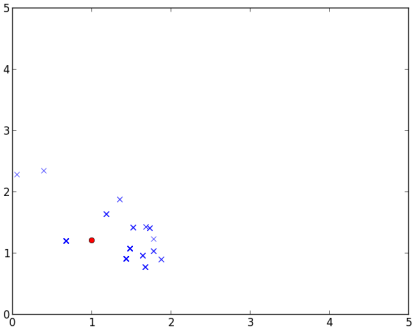
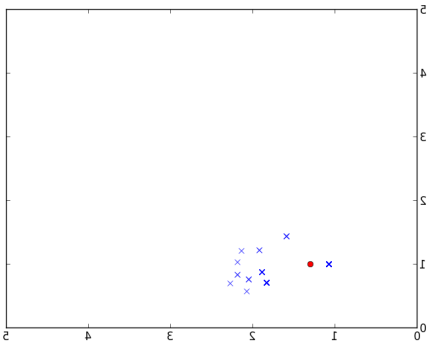
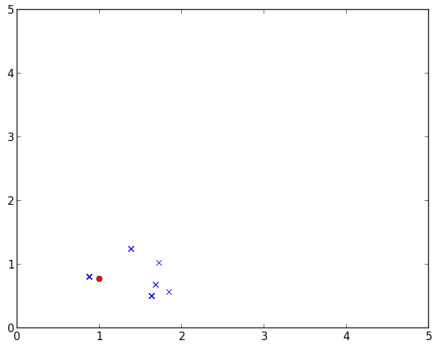
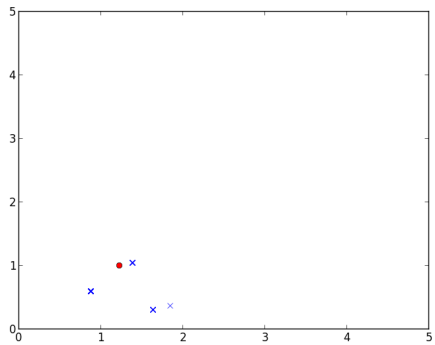
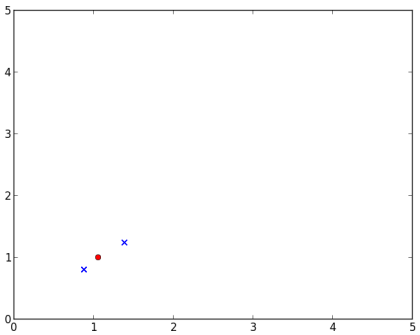
Kalman filter:

Note: all plots are not provided here, plots for each time step can be found in the newkalmanplots directory or allkalanplots.pdf

Screenshots of python –code executing the algorithms:

Particle filter:

Note: all plots are not provided here, plots for each time step can be found in the newparticleplots directory or allparticleplots.pdf

Conclusions and observations :

There are few key observations that have become intuitive after doing this project, if a erroneous result is included in estimating phase of the Kalman filter then it is impossible to recover. Whereas, in particle, since we consider beliefs from different random points its recoverable. A Kalman filter can perform better in a system with non Gaussian noise.

Also, Kalman filter works better in a set up like the given scenario, where the state representation doesn't involve higher order terms. This is clearly evident from the following timestamp observations.

Time taken to execute Kalman filter algorithm : 0.0135369300842 sec
Time taken to execute Particle filter algorithm : 7.53757214546 sec

This can be mainly attributed due to the repeated analysis of different particles performed during the particle filter execution.

References:

1. Lecture notes.
2. Notes based on discussion with TA.
3. http://en.wikipedia.org/wiki/Particle_filter
4. http://en.wikipedia.org/wiki/Kalman_filter
5. Tutorials on Kalman filtering on this channel: http://www.youtube.com/user/TheScienceguy3000?feature=watch