Implementation of Target Tracking Algorithms

Yu Liu, Omair Khalid, Wajahat Akhtar

November 15, 2017

1 Introduction

Multivariate state estimation problems are of great interest in innumerable practical applications. In such kinds of problems, the available measured data is used together with prior knowledge about the physical phenomena and the measuring devices, in order to sequentially produce estimates of the desired dynamic variables.In this report, we specifically apply ways to solve target-tracking problems under the Bayes framework. We demonstrate that under the assumption that if the multivariate state follows a Gaussian distribution and both motion and measurement model are linear, target-tracking tasks (single and multi-targets) can be easily addressed by the Kalman filtering framework and its extension.

2 Single-object Bayesian filtering

2.1 Bayes Theorem - where it all starts

Bayes' Theorem describes the probability of an event, based on prior knowledge of conditions that might be related to the event, and forms the basis for the Bayesian Filter. Bayes' Theorem is particularly useful in the case of tracking objects, since the prior information of the object's location may greatly help us better predict its location e.g. in real world scenarios, it is logical to assume from before (a priori) that the object is not be too far away from its previous position.

The Bayes' Theorem is formulated as follows:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{1}$$

$$P(A) : PriorProbability$$

$$P(B|A) : LikelihoodProbability$$

$$P(B) : MarginalProbability$$

2.2 Bayes Filter

In target tracking application, Bayes Filter adapts the Bayes Rule into estimating, sequentially, the states of the target based on a sequence of (partial) observations from a sensor. As the sensor measurements are not perfect and contain noise, we cannot be sure of the exact location or state of a target. That is why we employ Probability Density Functions that help us quantify our uncertainty. We must also pay attention to the fact that the state vector can have more than one elements and thus we will be dealing with a multivariate probability density function.

If we want to make use of the Bayes' Theorem, one of things we need is $P(A)$, the Prior Probability. Here, the Prior Probability will represent our knowledge of where we think our target could be, while taking into account the uncertainty associated with that guess. We can imagine that our target has reached the current state from the previous state following some "rules" that we know from before. For example, for a car traveling on a road, we can say that at the next time step the car maintained the same velocity and thus covered the corresponding distance. In the Bayes Filter, these "rules" are considered as the Dynamical Model that the car has to follow to achieve the next state, and the Dynamical Model helps us predict where our target could be.

$$P_{k|k-1}(x) = \int_0^{k-1} f_{k|k-1}(x,y) P_{k-1}(y) dy \tag{2}$$

$P_{k|k-1}(x)$: Probablity that the target is at x coordinate when it transitioned from time step = k-1 to time step = k

$f_{k|k-1}(x,y)$: The model that dictates how the target progpogated over time. Here, we employ a Markov kernel.

$P_{k-1}(y)$: Probability that the target was at y coordinate at time step = k-1

The equation above is called the Chapman-Kolmogorov equation. With the help of the Dynamical Model as represened by the Chapman-Kolmogorov equation, we propagate our target, dictated by the Markov Kernel, and estimate the Prior Probability, $P_{k|k-1}(x)$, of our target being at x coordinate at time step k.

The Observational Model is required to relate the state of the target to the observations of the target, while taking into account the uncertainty associated with it. Here, according to the Bayes' Theorem, the Observation Model helps us find the Likelihood Probability represented by:

$$g_k(z|x) : LikelihoodProbability \tag{3}$$

In the end, our Bayes Filter takes on the following form:

$$P_k(x|z) = \frac{P_{k|k-1}(x) g_k(z|x)}{\int_0^{k-1} P_{k|k-1}(y) g_k(z|y) dy} \tag{4}$$

The denominator term in the above equation corresponds to the Marginal Probability, $P(B)$, in the Bayes Filter. This is just a normalization term which helps us account for all the propagation of that the target underwent since its birth.

3 Kalman Filter

Kalman Filter is an exact implementation of the Bayes' Filter under the following conditions:

- Dynamical Model is linear, with Gaussian Process Noise
- Observation Model is linear, with Gaussian Observation Noise

The state vector is assumed to be a vector of random variables following the Multivariate Gaussian Probability Density Function.

3.1 Dynamical model

In the Dynamical Model of the Kalman Filter, we define the the Markov Kernel as being linear, following the following form:

$$f_{k|k-1}(x, y) = \mathcal{N}(x; Fy, Q) \tag{5}$$

F : Linear State Equation

Q : Co-variance of the Process Noise

$\mathcal{N}(x; mean, Covariance)$: Multivariate Gaussian Probabiliy Density represented by a mean and covariance

Note that in 5, the mean of the Gaussian Distribution is the product of the linear model F and the previous state. We know that the state vector at the previous time step is also Multivariate Gaussian and looks as follows:

$$P_{k-1}(y) = \mathcal{N}(x; m, P) \tag{6}$$

Finally, we can compute the Prior Probability using the Chapman-Kolomogorov equation (2). The final result looks as follows:

$$f_{k|k-1}(x, y) = \int \mathcal{N}(x; Fy, Q)\mathcal{N}(x; m, P)dy$$

$$f_{k|k-1}(x, y) = \mathcal{N}(x; Fm, FPF^T + Q) \tag{7}$$

3.2 Observation Model

The Observation Model also takes the form of a Gaussian multivariate distribution to account for the uncertainty of sensor measurements. The Likelihood Model can be written as:

$$g(z|x) = \mathcal{N}(z; Hx, R) \tag{8}$$

H : Observation matrix - maps the (partial) measurements onto the state space

R : Co-variance of the Observation Noise

3.3 Kalman Filter Equation

$$f_{k|k-1}(x, y)g(z|x) = \mathcal{N}(x; Fm, FPF^T + Q)\mathcal{N}(z; Hx, R)$$
$$f_{k|k-1}(x, y)g(z|x) = \mathcal{N}(x; \bar{m}, \bar{P}) \tag{9}$$

$$\bar{m} : m + K(z - Hm) \tag{10}$$

$$\bar{P} : (I - KH)P \tag{11}$$

$$K = PH^T(HPH^T + R)^{-1} \tag{12}$$

K is called the Kalman Gain in the equation above. The denominator is skipped in the notation because it only acts as the normalization factor, so our results, without the denominator, is correct only up to a scale.

3.4 Implementation Details

3.4.1 Simulation

In order to inspect the behaviour of the Kalman Filter, we have to simulate the behaviour of the target. For this purpose, we define a random initialization state of the target and propagate it for a defined number of steps using the same Dynamic Model we will use for the Kalman filter.

Furthermore, we also generate the measurements of the state for each time step using the state vector of the target at each time step. We do this by first taking the state vector of the current time step, transform it to the measurement space before using it as the mean of the Gaussian Distribution from which we will extract the measurement. The covariance used is defined in the Initialization step.

3.4.2 Initialization

In this step, we define all the required constants and constant matrices. They include co-variance of the Process Noise Q, the co-variance of Observation Noise R, the State Transition matrix F and H matrix, which is the mapping of the state from state space to the measurement space.

3.4.3 Prediction

The steps of Prediction and Update take place in a loop, where each iteration gives us the position where the Kalman Filter believes the object is.

Firstly, we implement (2) by multiplying F with the state vector in the previous state, and implement $FPF^T + Q$. Now, we have the mean and the covariance of the probablity distribution, according to (7).

Next, we implement (10), (11) and (12) to get the new Mean and Covariance describing the state of the target.

3.4.4 Update

In the Update step, we just make the new Mean and Covariance of the state of the target equal to the one that will be used as the previous state in the next iteration.

3.5 Results

Below are two examples of the implemntation of the Kalman Filter. The blue crosses correspond to the groud truth of the target state, and the red plus signs correspond to the tracking of the Kalman Filter.

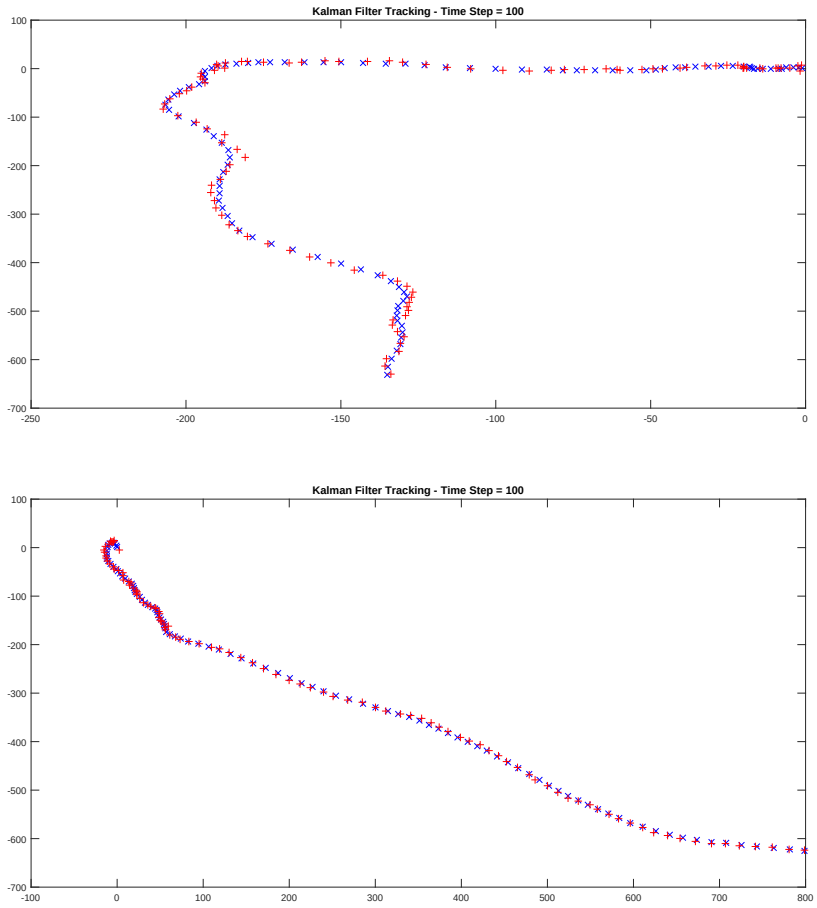

Figure 1: Kalman Filter with Time Step k = 100

4 Extended Kalman Filter

In the Extended Kalman Filter, the "extension" lies in the fact that it now allows us to incorporate a non-linear State Space to Measurement Space transformation model i.e. a non-linear Observation Model, and a nonlinear State Transition/Dynamical Model. One example of a non-linear Observation Model would be estimating the state of the target with the help of radar measurements. As it may happen, the target state would be defined in Cartesian Coordinates and the Measurements would be defined in the Polar Coordinates:

$$State\ Space: \begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{bmatrix}$$

$$Observation\ Space: \begin{bmatrix} r \\ \phi \end{bmatrix}$$

The EKF works by transforming the nonlinear models at each time step into linearized systems of equations. To tackle the non-linearity of the Observation Model, we employ what is called a Jacobian matrix J, a matrix of Partial Derivatives of the variables of measurement space w.r.t. the variables of the State Space.

$$J = \begin{bmatrix} \frac{\partial r}{\partial x} & \frac{\partial r}{\partial \dot{x}} & \frac{\partial r}{\partial y} & \frac{\partial r}{\partial \dot{y}} \\ \frac{\partial \phi}{\partial x} & \frac{\partial \phi}{\partial \dot{x}} & \frac{\partial \phi}{\partial y} & \frac{\partial \phi}{\partial \dot{y}} \end{bmatrix}$$

As in many cases where you approximate a nonlinear system with a linear model, there are cases where the EKF will not perform well. If you have a bad initial guess of the underlying system's state, then you could get garbage out. In contrast to the standard Kalman filter for linear systems, the EKF is not proven to be optimal in any sense; it's merely an extension of the linear-system technique to a wider class of problems.

4.1 Implementation Details

4.1.1 Simulation

In order to inspect the behaviour of the ExtendedKalman Filter, we have to simulate the behaviour of the target. For this purpose, we define a random initialization state of the target and propagate it for a defined number of steps using the same Dynamic Model we will use for the Prediction step in the Extended Kalman filter. For the sake of experimentation, we are using a Linear Prediction model but a non-linear Observation Model.

Furthermore, we also generate the measurements of the state for each time step using the state vector of the target at each time step. We do this by first taking the state vector of the current time step, transform it to the measurement space using the relationship between the Measurement and State Space: $range = \sqrt{x^2 + y^2}$ and $bearing = atan2(x, y)$. We then use the *range* and *bearing* as the mean vector of the Gaussian Distribution from which we will extract the measurement. The covariance used is defined in the Initialization step.

4.1.2 Initialization

In this step, we define all the required constants and constant matrices. They include co-variance of the Process Noise Q, the co-variance of Observation Noise R, the State Transition matrix F and H matrix, which is the mapping of the state from state space to the measurement space.

4.1.3 Prediction

The steps of Prediction and Update take place in a single loop, where each iteration gives us the position where the Extended Kalman Filter believes the target is.

The steps are the same as that in tke Kalman Filter. Firstly, we implement (2) by multiplying F with the state vector in the previous state, and implement $FPF^T + Q$. Now, we have the mean and the covariance of the probablity distribution, according to (7).

4.1.4 Update

We replace the H in the equations (11) and (12) with J, which renders the following equations:

$$f_{k|k-1}(x, y)g(z|x) = \mathcal{N}(x; Fm, FPF^T + Q)\mathcal{N}(z; Hx, R)$$
$$f_{k|k-1}(x, y)g(z|x) = \mathcal{N}(x; \bar{m}, \bar{P}) \tag{13}$$

$$\bar{m} : m + K(z - Hm) \tag{14}$$

$$\bar{P} : (I - KJ)P \tag{15}$$

$$K = PJ^T(JPJ^T + R)^{-1} \tag{16}$$

H is a matrix that helps relate the measurements to the state space. In the Update step, we implement (14), (15) and (16) to get the new Mean and Covariance describing the state of the target. We then make the new Mean and Covariance of the state of the target equal to the one that will be used as the previous state in the next iteration.

4.2 Results

Below is one of the experiments of Extended Kalman Filter.

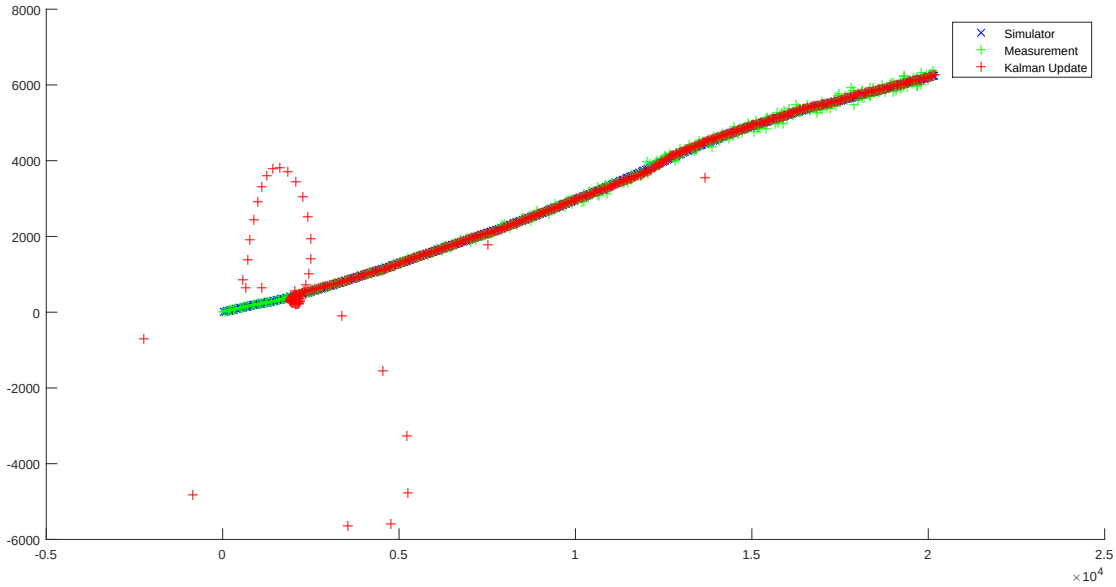

Figure 2: Standard Deviation of Process Noise, Observation Noise: 1.0, 1.0

5 Gaussian mixture (Gaussian sum) filter

5.1 Theory

A Gaussian mixture model is commonly employed to fit unknown multivariate Gaussian distributions and can accurately represent distributed subpopulations within an overall population. In the context of multi-senor fusion, one may model several observations from a host of sensors using Gaussian mixture model, where ideally the number of mixture components are equivalent to the number of targets at each frame in a video sequence, and their means predict object locations.

Extended from Kalman filter as covered in the earlier section, a Gaussian mixture model essentially estimates the parameters of the individual normal distribution components by computing individual Kalman prediction and update. A Gaussian mixture model is parameterized by the mixture component weights and the component means and variances/covariances. The sum of all mixture component weights is one (each component weight is normalized). A mixture component weight typically is the product of that previous weight and the likelihood (the probability of observed data given the model parameters).

5.2 Implementation Details

In our implementation, Gaussian mixture is stored in a j-length cell of structures, where j stands for the number of Gaussian components. Each structure contains key fields to represent a Gaussian component such as weight, mean and covariance. Each Gaussian component's mean are randomly initialized, while its weight is set to $\frac{1}{j}$ to keep the sum as one. All Gaussian components have the same initial covariance, since it is assume that all Gaussian components are equally likely to track or not track a target with the same uncertainty.

The Markov kernel and measurement matrix are linear and represented by matrices the same way as depicted in the earlier section. Furthermore, motion and measurement uncertaints, as well as the simulator tested in this case are the same as the one used in the above section.

Every time step each Gaussian component's mean and covariance is independently updated via Kalman prediction and update step (each follows the equation 9-12). Each component's weight is scaled by the likelihood (i.e., g(z|x)) via MatLAB's mvnpdf function. This function estimates the probability of obtaining an observation at a given state, where the state corresponds to the Gaussian component's mean updated by Kalman filter. Once all weights of all Gaussian components are updated, each is normalized by the sum of all the weights to maintain a sum of 1.

5.3 Results and analysis

Figure 3 and 4 display single-object Gaussian mixture filtering with varied number of Gaussian components. It can be observed in all cases that the filter quickly converges within a few time steps as scattered Gaussian components segregate near to the target's location. Since each Gaussian component is filtered independently by a Kalman filter, these results again verify how quickly Kalman filter converges under linear motion, measurement and Gaussian distribution. Please note that no Gaussian component is lost throughout the entire tracking process; all Gaussian components simply have estimated means that coincided with the target's location. Weight of each Gaussian component changes drastically once the program starts but eventually converges to $\frac{1}{j}$.

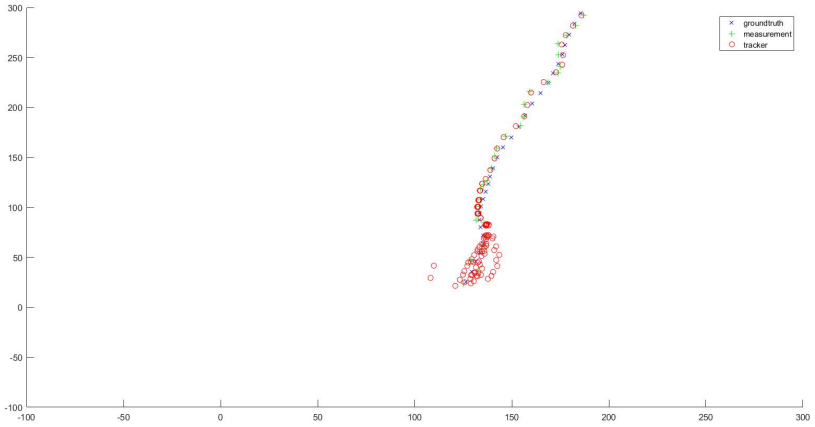

Figure 3: Single target tracking, 10 Gaussian components; time step k = 30

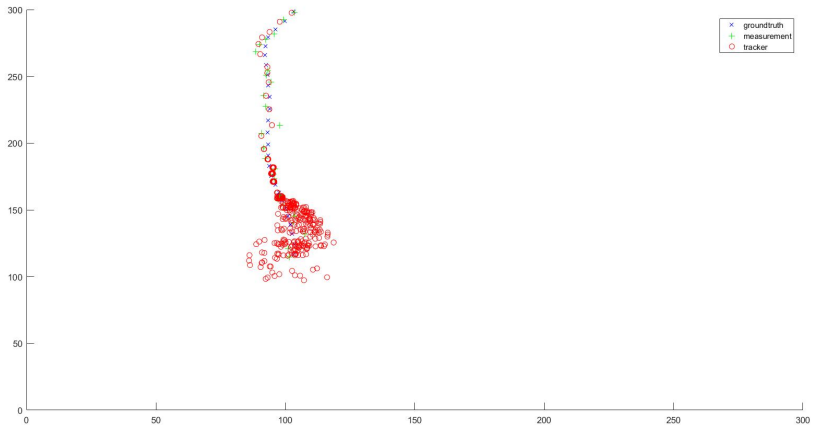

Figure 4: Single target tracking, 40 Gaussian components; time step k = 30

The above tests initialize Gaussian components to be near the target. Figure 5 and 6 display single-object Gaussian mixture filtering with the same number of Gaussian components (20), but with widen initial means. It can be seen that generally it takes more iterations to converge when Gaussian components are initially far away from the observation, but the number of extra iteration needed is insignificant in the case of single target tracking (since in every time step there is only one observation). Such phenomenon may not be the same for multi-target tracking, as all Gaussian components tend to converge to the nearest series of observations and may eventually lose track of some targets in the surveillance region.

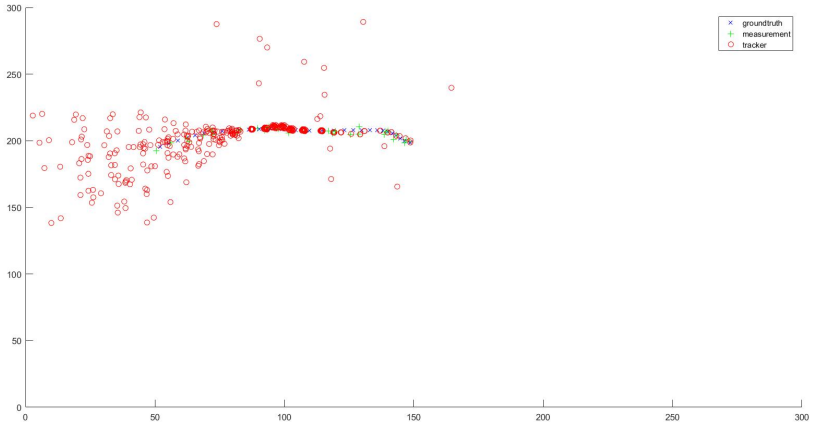

Figure 5: Single target tracking, 20 Gaussian components, initial means far from target; time step k = 30

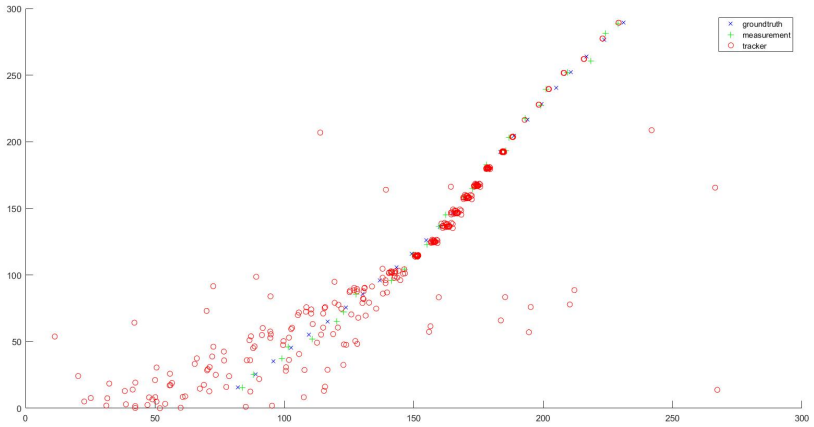

Figure 6: Single target tracking, 20 Gaussian components, initial means far from target; time step k = 30

It is worth mentioning that a Gaussian mixture model such as the one described in this section can perform a multi-target tracking where ideally as the filter converges, each Gaussian component would track a single target (Gaussian's mean corresponds to the target's location). However in this section we do not demonstrate multi-target tracking but will cover more details in section Probability Hypothesis Density filter .

6 Particle Filter

The Particle Filter Method is a Monte Carlo technique for the solution of the state estimation problem. The particle filter is also known as the bootstrap filter, condensation algorithm,interacting particle approximations and survival of the fittest.The key idea is to represent the required posterior density function by a set of random samples (particles) with associated weights, and to compute the estimates based on these samples and weights. As the number of samples becomes very large, this Monte Carlo characterization becomes an equivalent representation of the posterior probability function, and the solution approaches the optimal Bayesian estimate.

The most widely known Bayesian filter method is the Kalman filter. However, the application of the Kalman filter is limited to linear models with additive Gaussian noises. Extensions of the Kalman filter were developed in the past for less restrictive cases by using linearization techniques. Similarly, Monte Carlo methods have been developed in order to represent the posterior density in terms of random samples and associated weights. Such Monte Carlo methods, usually denoted as particle filters among other designations found in the literature, do not require the restrictive hypotheses of the Kalman filter. Hence, particle filters are applied to non-linear models with non-Gaussian errors.

6.1 Implementation of Particle Filter

The aim of this section was to Implement Particle Filter or Montecarlo Localization (MCL) algorithm to localize single object in a specified region for different time step k. In order to implement Particle Filter three main steps were followed.First step was to predict the position of the object using particles.Which is followed by weighting step where particle are assigned weights by probability distribution over the space.Once weighting is done re-sampling is applied in order to avoid replication of particles resulting in localization of the object at different time step, We perform re-sampling only when the particles are moving if they are still or not moving re-sampling is no more valid/applied acting as a draw back of re-sampling. A simulator is created for comparison which act as the ground truth for the real model containing noise. Below here is a detailed technical explanation of steps we performed to implement Particle filter with parameters initialization and pseudo code.

6.1.1 Parameter Initialization

Parameter initialization is an important step in particle filter which need to be performed carefully as it can be computational expensive. Below here is the list of parameters which need to be set before starting the main step of particle filter.

- Frames per second was set to 1 and was denoted as delta_t .
- We take noise into account both for observations and measurement which is represented by sigma. As in practical, it cannot be neglected it can be varied depending

on the type of sensor we are using.

- Co-variance matrix could be initialized with any value as it gets updated.
- Linear Sate Transition model denoted by matrix F was used with sigma Q(process noise) and sigma R(measurement noise).The size of the matrix remain fixed same as motion model.
- H matrix is created to convert state of the model to measurement space as we need to measure state only not velocities as our motion model contains state and velocity of both x and y.
- Generate random particles around the region(e.g 800x800) the more the number of particles the better it will be for the model to predict the state , but it can be computational expensive.The initial state of the object must lie within in the space where particles are generated.

6.1.2 Simulator Design

In order to inspect the behaviour of the Particle Filter, we have to simulate the behaviour of the target. For this purpose, we define a random initialization state of the target for time step k using the same prediction model for the particle filter with process noise(sigma Q) as given in equation 17 below.

$$\tilde{x}_k^{(i)} \sim f_{k|k-1} * (x_k | x_{k-1}^{(i)}) \tag{17}$$

Furthermore, we also generate the measurements of the state for each time step using the state vector of the target at each time step with noise (sigma R) which is shown in equation 18 below.

$$\tilde{w}_k^{(i)} = g_k(z_k | \tilde{x}_k^{(i)}) \tag{18}$$

We do this by first taking the state vector of the current time step, transform it to the measurement space by multiplying with H matrix before using it as the mean of the Gaussian Distribution from which we will extract the measurement with noise sigma R. The co-variance is used as its defined in the Initialization step.

6.1.3 Sampling

Prediction

The main aim of the prediction step is to predict the position of the object using probability density function,The more the particles fall into a region the higher the probability of the region ,So The more samples we use, the better the approximation.So in that case of object prediction the more the number of particles at a region the higher the probability that the object is located there.In prediction step we define given change in position x, y with time step k for samples $i = N$, with some sigma(deviation) value,Sigma value

is used to compensate the noise as when the object moves to a certain position from one position there are more chances that it doesn't reach to its destination value due to some noise.In order to perform prediction we perform the following step using equation 17.

- We have used certain number of particles/sample in order to approximate/predict position of the object , these particles are in generated around the object randomly by using probability density function.
- We add the change in x,y and noise to the particles to predict the position of object after certain time step k , with sigma representing the variation from mean because when the particles moves from one place to other some noise also get added to them which deviates them from reaching to the goal.
- We store the predicted position of the particles which is nothing else but the prediction of the object position.once prediction is completed it is carried out by another step called weighting.

Weighting

Weighting is an important step in Particle filter,Where we evaluate the importance weight for each sample $i = N$ at time step k, which is represented by using equation 18 given above.While assigning the weight to the samples we compute total sum of weight by simple concatenation which is stored in a variable called w_sum to get the sum of all weights.We use total sum of weights for the normalization of each particle using equation 19 given below.

$$\tilde{w}_k^{(i)} = \frac{\tilde{w}_k^{(i)}}{\sum_{j=1}^{N} \tilde{w}_k^{(j)}} \tag{19}$$

The equation above shows that the weight of each particle is divided by the total weighted sum of all the particles resulting in a normalize weight for each particle.

6.1.4 Re-sampling

Re-sampling is a method to look for the replication of particles , if any of the particle is repeating or away from some threshold we don't consider it. Re-sampling is performed on N particles with $x_k^{(i)}$ $i = N$. In re-sampling we calculate mean u and co-variance cov of the particle set.Where, one draws (with replacement) a new set of N particles from the discrete approximation to the filtering distribution $g_k(z_k|\tilde{x}_k^{(i)})$ (or to the full posterior distribution. if one is interested in computing the full posterior) provided by the weighted particles using equation 20 below.

$$\tilde{w}_k^{(i)} = N(x_k^{(i)}, u, cov) \tag{20}$$

Re-sampling is performed whenever the effective sample weight $\tilde{w}_k^{(i)}$ drops below a certain threshold. Note that since re-sampling is done with replacement, a particle with

a large weight is likely to be drawn multiple times and conversely particles with very small weights are not likely to be drawn at all. Also note that the weights of the new particles will all be equal to 1/N. Thus, re-sampling effectively deals with the degeneracy problem by getting rid of the particles with very small weights. What actually is done is we check whether the size of u is greater then the value of c ,where u is the pointer to the weight of the particles and c is initialized with the weight of first particle, $\tilde{w}_k^{(i)}$ is a random number generated by discrete approximation and u is calculated by adding particle number and index of the particle we are dealing with the number of the particle, Which increments in every loop by value of step k.We compare u with c,if the size of u is greater then c we keep increasing the index of the particle and also update c to be the sum of the previous weight to the next weight and move to the step 1.Mean u and co-variance cov are calculated using the equation 21 , 22 below.

$$u = \sum_{i=1}^{N} \tilde{w}_k^{(i)} * x_k^{(i)} \tag{21}$$

$$cov = \sum_{i=1}^{N} \tilde{w}_k^{(i)} * (x_k^{(i)} - u), (x_k^{(i)} - u) \tag{22}$$

6.2 Results and Analysis of Particle Filer

In this section we discuss the results achieved using particle filter with advantages and drawbacks.The results were generated by tweaking parameters which are important in changing the behaviour of the filter.In order to make filter work parameters should be set properly which could be seen in the results shown below.

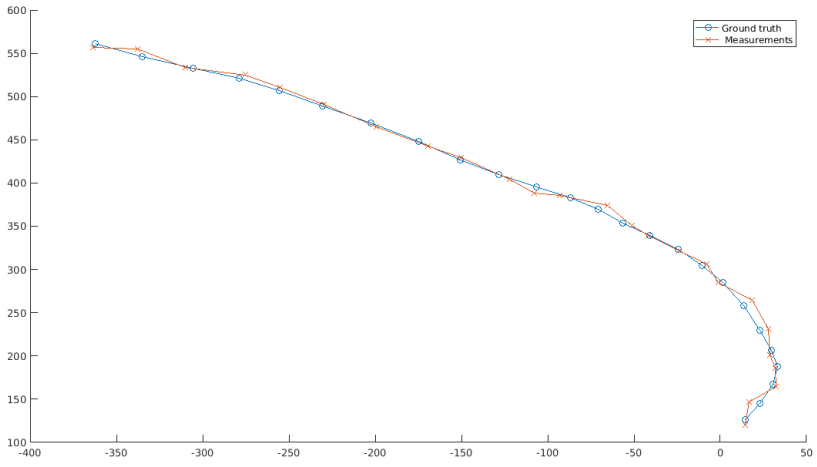

Figure 7: Simulator Particle Filter with Time Step k = 25

Figure 7 above shows the output of the simulator , with ground truth and measurements for an object for time step k = 25 . Below here are the results for tracking single object using fix number of time step k = 25 with change in number of particle from 10-80.

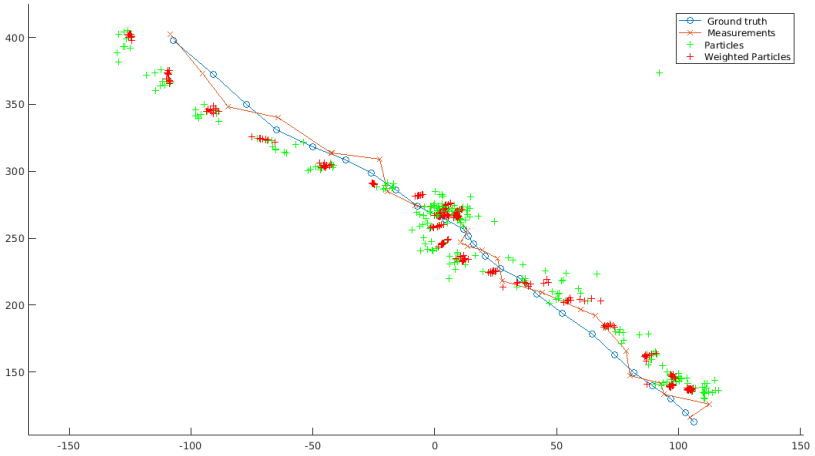

Figure 8: Particle Filter Response with Particles = 10 , Time Step k = 25

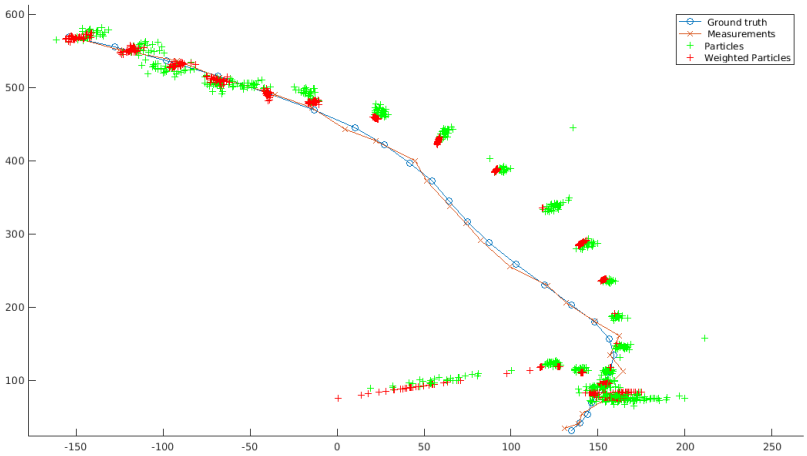

Figure 9: Particle Filter Response with Particles = 30 , Time Step k = 25

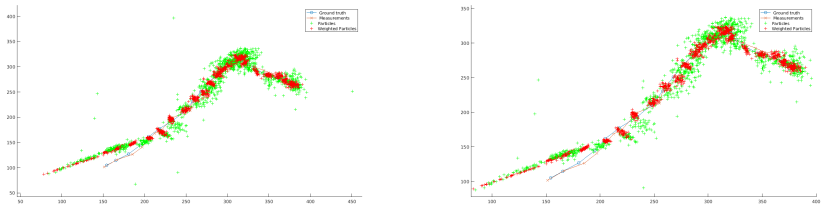

Figure 10: P.F Response with Particles = 50 , Time Step k = 25 : Left = real, Right = Zoom in

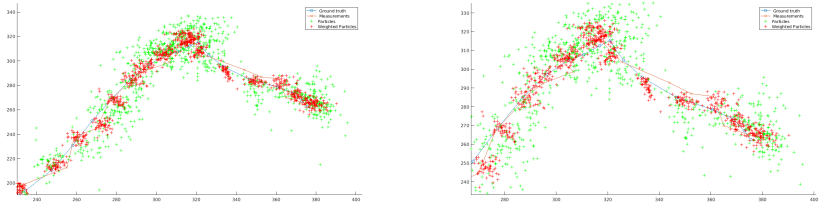

Figure 11: P.F Response with Particles = 50 , Time Step k = 25 : Left = Zoom in, Right = Max Zoom

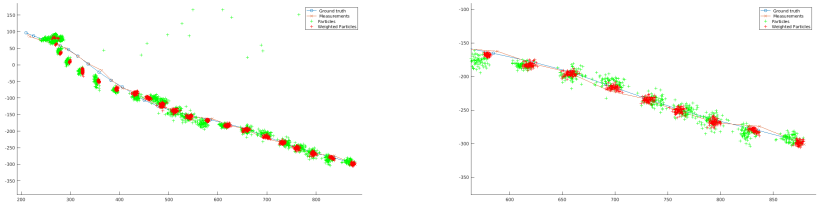

Figure 12: P.F Response with Particles = 80 , Time Step k = 25 : Left = real, Right = Zoom in

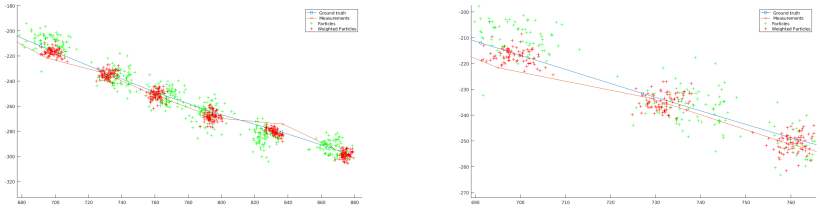

Figure 13: P.F Response with Particles = 80 , Time Step k = 25 : Left = Zoom in, Right = Max Zoom

Figure 9,10,11,12 and 13, below shows the output of the single object tracking using different number of Particles that is 10,30,50 and 80 with ground truth and measurements for an object for time step k=25.We found out that if less number of particles are used with small time step like 25 the convergence of particles is very rare with the original observations which proves the explanation defined above that the more the number of particles the better the tracking result of the object is with the ground truth model.In second part we try to change the time step from 25 to higher numbers while keeping the number of particles fixed to 120, as we already know that the more the particles are the better is the prediction,but by increasing number of time step we found out that the more the number of steps we use the better particles prediction results in with high convergence to simulator measurements as could be seen in figure 14 and 15 below.

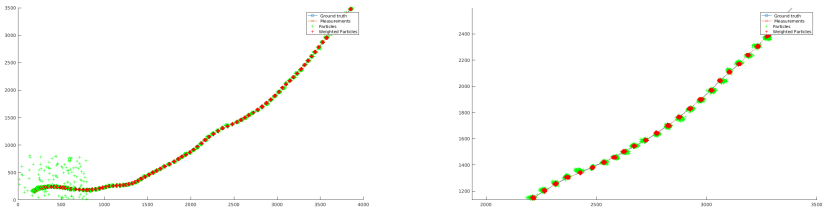

Figure 14: Particle Filter Response with Particles = 120 , Time Step k = 50

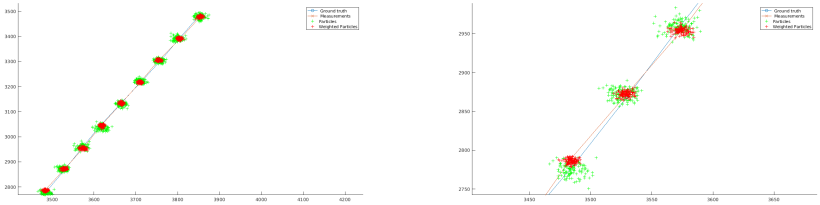

Figure 15: Particle Filter Response with Particles = 120 , Time Step k = 50

7 The Probability Hypothesis Density filter

The PHD filter is an approximation developed to alleviate the computational intractability in the multi-target Bayes filter. Instead of propagating the multi-target posterior density in time, the PHD filter propagates the posterior intensity, a first- order statistical moment of the posterior multi-target state. This strategy is reminiscent of the constant gain Kalman filter, which propagates the first moment (the mean) of the single- target state. In a multi-target environment, not only do the states of the targets vary with time, but the number of targets also changes due to targets appearing and disappearing (dictated by surviving rate, P_s). Often, not all of the existing targets are detected by the sensor (dictated by detection rate, P_D). Moreover, the sensor also receives a set of spurious measurements (clutter, dictated by lambda) not originating from any target. As

a result, the observation set at each time step is a collection of indistinguishable partial observations, only some of which are generated by targets. The objective of multi-target tracking is to jointly estimate, at each time step, the number of targets and their states from a sequence of noisy and cluttered observation sets.

7.1 Theory

PHD filter is an analogous solution to the multi-target Gaussian sum filter. It shows that if the initial prior intensity is a Gaussian mixture, the posterior intensity at any subsequent time step is also a Gaussian mixture. Moreover, closed form recursions for the weights, means, and covariances of the constituent Gaussian components are derived. The resulting filter propagates the Gaussian mixture posterior intensity in time as measurements arrive in the same spirit as the Gaussian sum filter. The fundamental difference is that the Gaussian sum filter propagates a probability density using the Bayes recursion, whereas the Gaussian mixture PHD filter propagates an intensity using the PHD recursion. An added advantage of the Gaussian mixture representation is that it allows state estimates to be extracted from the posterior intensity in a much more efficient and reliable manner than clustering in the particle-based approach. In general, the number of Gaussian components in the posterior intensity increases with time due to data association between each Gaussian component and measurement. However, this problem can be effectively mitigated by keeping only the dominant Gaussian components at each instance.

7.2 Implementation Details

7.2.1 Simulation

In the Simulation, we try to create a ground truth scenario where targets are moving through the state space and may survive or die with some probability. The propagation of target from one time step to another follows the same prediction model as will be used in the Prediction in Steps 1 and 2. What's different in the implementation of PHD filter as compared to the filters discussed in the previous sections is the inclusion of Probability of Survival P_s at each time step, which dictates if the target has survived or died. In this way, the propagation of the target is subject to whether it survived or not.

As we are propagating the target, we use the predicted mean of the target at each iteration as the mean for a Gaussian distribution (Covariance set as in Initialization) from which we extract the sensor measurements. However, here again, the measurement gathered from each target at each time step may be misdetected, as dictated by P_D. Furthermore, we may also get false alarms (produced by Poisson Distribution), which are included in the set of measurements gathered at each time step.

7.2.2 Step 1: Prediction of Birth Targets

In this step, we deal with the birth of the new Gaussians, meaning that we assign a weight, mean and covariance to each new Gaussian tracker created. In our implementation, we

set the number of newly-born Gaussian trackers created to be a constant (2 times the number of targets). All these Gaussians are assigned the weight = 0.5, meaning that, at birth, they have a 50% chance of accurately matching a target. The mean values are randomly generated from a Gaussian distribution. The covariances are also set to be the same for each of these Gaussians, for the sake of simplicity.

7.2.3 Step 2: Prediction of Existing Targets

The prediction of the existing targets follows equation (7). However, the prediction of the weights of all the existing Gaussians follow:

$$w_{k|k-1} = p_{S,k} w_{k-1} \tag{23}$$

$$p_{S,k} : Probability\ of\ Survival$$

$$w_{k-1} : Weight\ at\ the\ previous\ iteration$$

7.2.4 Step 3: Construction of PHD Update Components

Using equations (11) and (12), we calculate K_k and $P_{k|k} = \bar{P}$ for each Gaussian tracker. The term $\eta_{k|k-1}$, which is a mapping of the propagated mean of the Gaussian onto the measurement space will be used, along with K_k and $P_{k|k}$, in the Update step in order to calculate the the new mean, new weight and new covariance for the respective Gaussians.

$$\eta_{k|k-1} = H_k m_{k|k-1} \tag{24}$$

$$H_k : Mapping\ from\ state\ to\ measurement\ space$$

$$m_{k|k-1} : Propagated\ mean\ of\ the\ Gaussian$$

7.2.5 Step 4: Update

In the update step, we first deal with the case where we received no measurement from the sensor. To tackle this scenario, we update the weight, mean and covariance of each Gaussian as follows:

$$w_k = (1 - P_D) w_{k||k-1} \tag{25}$$

$$m_k = m_{k|k-1} \tag{26}$$

$$P_k = P_{k|k-1} \tag{27}$$

After this step, we have the updated Gaussians in the extreme case where no measurement is obtained. Next, we will perform check how well each existing Gaussian matches each measurement, and create an updated set of Gaussians per measurement. In this end we will be left with N Gaussians, where N = No. of Gaussians x No. of Measurements. The equations used are as follows:

$$w_k = P_D w_{k||k-1} \mathcal{N}(z; \eta_{k|k-1}, S_k) \tag{28}$$

$$m_k = m_{k||k-1} + K_k(z; \eta_{k|k-1}) \tag{29}$$

$$P_k = P_{k|k} \tag{30}$$

At the end of this step we have the number of Gaussians totalling to N added with the original number of Gaussians (sum of both extremities, where either all targets are observed or none is observed).

7.2.6 Step 5: Merging and Pruning

In this step, the Gaussians with very little weight are removed and the ones close together are merged. We have three parameters here i.e. T, the truncation threshold (0.00001), U, the merging threshold (4), and J_{max}, the maximum allowable number of Gaussians.

First, we remove those Gaussians from the list whose weight is below T. Next, for each Gaussian, we make a set $\mathcal{L}$ of all other Gaussians whose distance from that Gaussians is less than U, where the Distance is as follows:

$$(m_k^{(i)} - m_k^{(j)})^T (P_k^{(i)})^{-1} (m_k^{(i)} - m_k^{(j)}) \tag{31}$$

$$m_k^{(i)} : Mean\ of\ the\ Gaussian\ under\ consideration$$

$$m_k^{(i)} : Mean\ of\ the\ iterated\ Gaussian(s)$$

$$P_k^{(i} : Covariance\ of\ the\ Gaussian\ under\ consideration$$

Next, we proceed to merging all the elements in $\mathcal{L}$. All the elements in $\mathcal{L}$ are accounted for to form a single Gaussian, using the formulae below:

$$\bar{w}_k^l = \sum_{i \in \mathcal{L}} w_k \tag{32}$$

$$\bar{m}_k^l = \frac{1}{\bar{w}_k} \sum_{i \in \mathcal{L}} w_k x_k \tag{33}$$

$$\bar{P}_k^l = \frac{1}{\bar{w}_k} \sum_{i \in \mathcal{L}} w_k (P_k + (\bar{m}_k^l - m_k)(\bar{m}_k^l - m_k)^T) \tag{34}$$

$$w_k : weight\ of\ the\ Gaussian\ in\ list\ \mathcal{L}$$

$$x_k : mean\ of\ the\ Gaussian\ in\ list\ \mathcal{L}$$

$$P_k : covariance\ of\ the\ Gaussian\ in\ list\ \mathcal{L}$$

$$Superscript\ l\ corresponds\ to\ the\ resulting\ Combined\ Gaussian$$

After the Merging step as described above, the last filtering is performed by only keeping only J_{max} number of Gaussian in our list to propagate to the next time step.

7.3 Results and analysis

The following figures display how our PHD filter tracks moving targets in a noiseless environment (no false alarm). More specifically, figure 16 shows the result where after the pruning step, all remaining Gaussian components are propagated to the next iteration. On the other hand, figure 17 shows the result when only N highest-weight Gaussian components are propagated, where j corresponds to the number of moving targets that still survive.

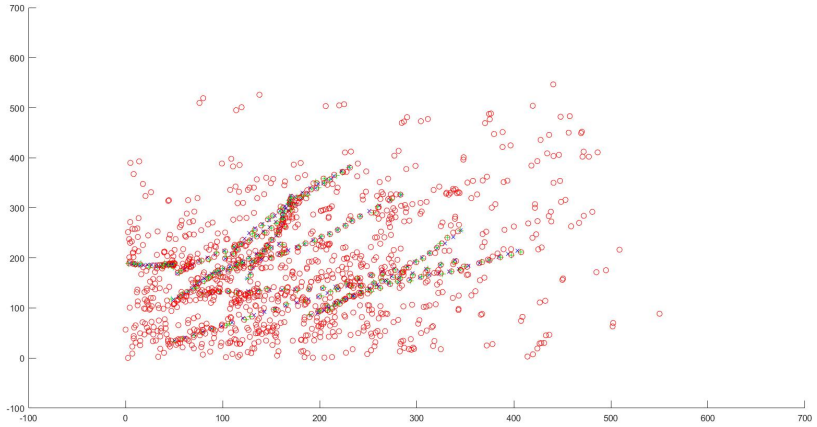

Figure 16: 10 targets, time step k = 60, lambda = 0, $P_D = 0.98$, $P_S = 0.99$; J_{max} not set

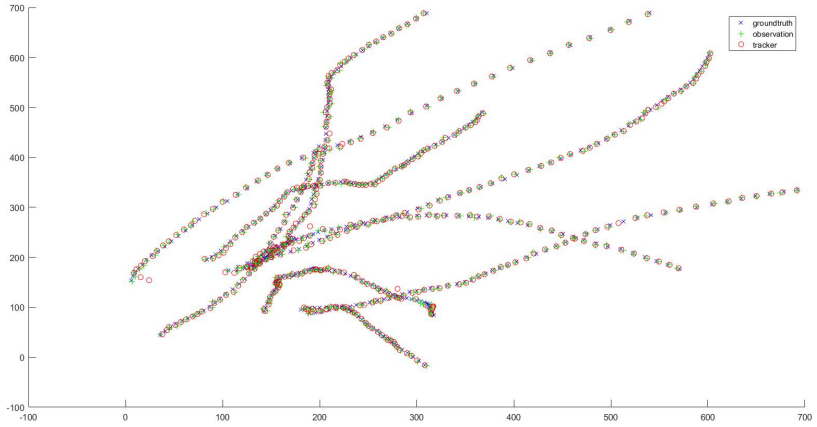

Figure 17: 10 targets, time step k = 60, lambda = 0, $P_D = 0.98$, $P_S = 0.99$; J_{max} = No. of survived target

From figure 16, all Gaussians being plotted have weights over the truncation threshold. The cause of the Gaussians that do not track moving targets, as we investigated, is actually due to the parameter we set for Pd and the birth Gaussian injected iteratively. These Gaussians are contributed by the extreme case when no measurement is obtained to update Gaussian weight using likelihood, hence Gaussian weights are simply updated by 1 – Pd. Even with Pd set as high as 0.98, $1 - P_D$ still is a relatively high scaling factor compared to the truncation parameter. The product of this scaling factor and birth Gaussian's initial weights (0.5) are significantly larger than the truncation threshold. As a result, Gaussian components associated with no detection would remain after pruning for a couple more iterations (eventually their weight would be scaled below the truncation and be filtered in the pruning step).

When only limiting the number of Gaussian to propagate (in other words, only the ones with highest weights survive for the next iteration), we see that the survived Gaussian still successfully track moving targets in a noiseless environment. This demonstrates the fact that after the pruning step, Gaussians with the highest weights correspond to targets' location. PHD's robustness under noiseless environment remains even when we increased to 50 moving targets.

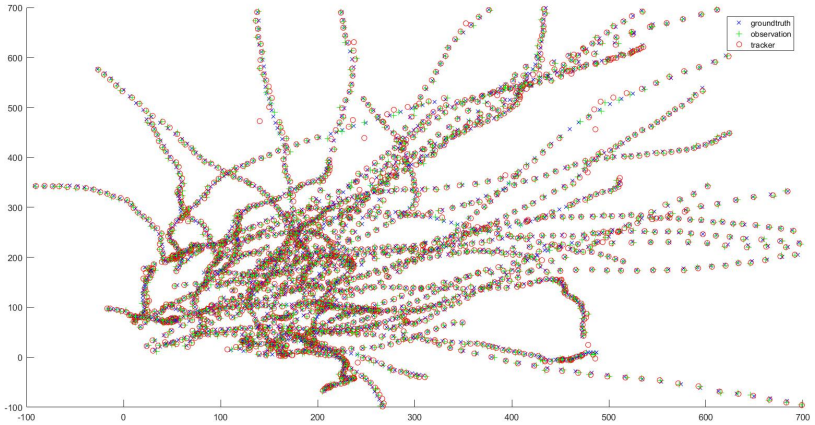

Figure 18: 50 targets, time step k = 60, lambda = 0, $P_D = 0.98$, $P_S = 0.99$; J_{max} = No. of survived target

In the above tests, 2N of Gaussian are given birth in every iteration, which all demonstrate fine tracking. By reducing birth Gaussian from 2N to N, we observe that PHD start to miss tracking a number of moving targets. Missed targets have some chances to be re-tracked thanks to birth Gaussian as shown below (gap between tracker's record for some targets). We conclude that the more birth Gaussian injected, the higher chance moving targets get re-tracked if lost. Moreover, when zero birth Gaussian is injected, we found that the PHD filter soon lose track of all moving targets even if the motion is slow and steady.

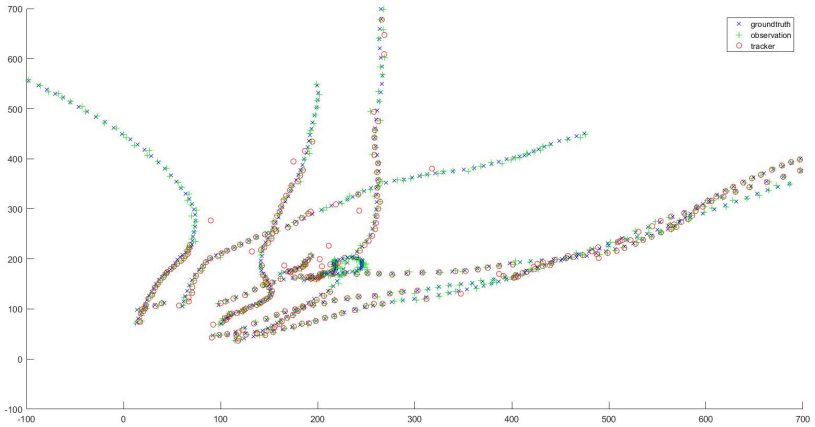

Figure 19: 10 targets, time step k = 60, lambda = 0, $P_D = 0.98$, $P_S = 0.99$; J_{max} = No. of survived target; No. of birth-Gaussians reduced

In a noisy environment, tracking performance of PHD degrades under higher false alarm rate (higher lambda). In figure x (lambda = 3), we see that with moderate false alarm, the PHD is capable of tracking most moving targets. One moving target is lost at an early stage right after an intersection of three moving targets, which may be the cause of that. In figure x (lambda = 20), more targets are lost at an early stage, and we can observe some attempt of re-tracking by the filter, which all end up failing. In figure x we apply the same false alarm rate, but allow more Gaussian to propagate to the next iteration. By propagating more potential trackers to the filter, all targets are successfully tracked, at a cost of higher false positives.

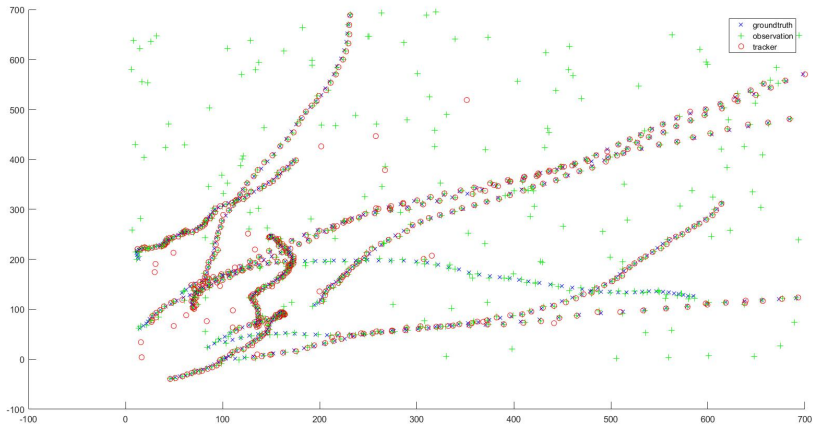

Figure 20: 10 targets, time step k = 60, lambda = 3, $P_D = 0.98$, $P_S = 0.99$; J_{max} = No. of survived target

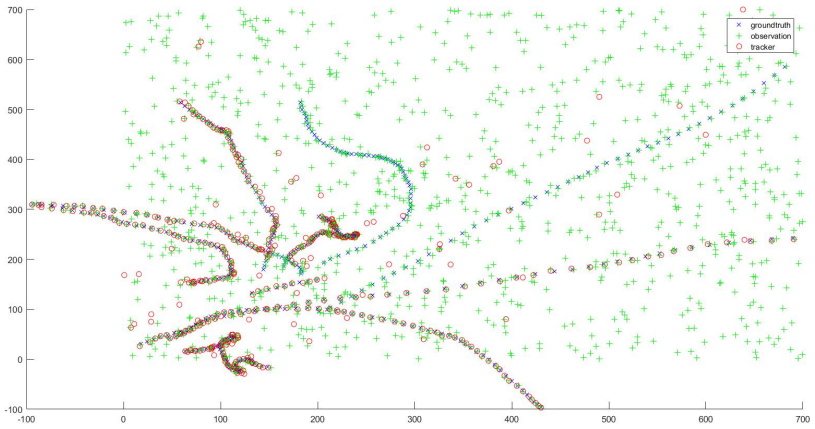

Figure 21: 10 targets, time step k = 60, lambda = 10, $P_D = 0.98$, $P_S = 0.99$; J_{max} = No. of survived target

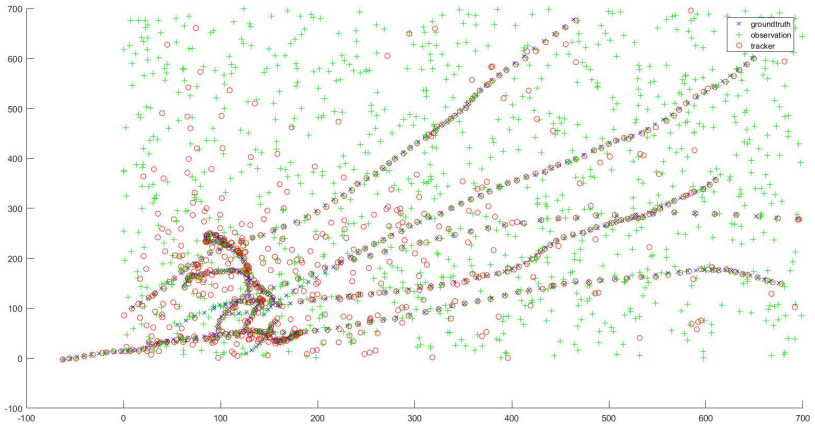

Figure 22: 10 targets, time step k = 60, lambda = 20, $P_D = 0.98$, $P_S = 0.99$; J_{max} = No. of survived target; No. of Gaussians to propagate increased

8 Conclusion

In this report, we demonstrate a multitude of filters to solve target-tracking problems. We began with the simplest tracking task - single target under ideal object detection and survival. Kalman filter proves to quickly converge to track the target's location. Next, by relaxing the ideal tracking scenarios, we also demonstrate that extensions of Kalman equations can be adapted into extended Kalman, Monte Carlo framework and Gaussian mixture model to solve more complex tracking problems. Lastly we introduced PHD

filter which incorporates the concept of birth-Gaussian, non-ideal object detection and survival rate as well as false alarms into multi-target tracking problems. We showed that even with high clutter and non-ideal detection / target survival (which are commonly encountered in real life applications), PHD filter is still capable of tracking multiple targets with good accuracy. Most importantly, it has great likelihood to re-track lost targets given sufficient Gaussians are introduced.