Chapter 4

Tracking and Identifying of Multiple Targets

John Hoffman[1], Christian Ketelsen[2], Michael Kouritzin[3], Alfonso Limon[4], Yuriy Mileyko[5], Kerianne Yewchuk[3]

Report prepared by Kerianne Yewchuk (kyewchuk@ualberta.ca), Christian Ketelsen (cketelsen@wsu.edu), Alfonso Limon (alfonso.limon@cgu.edu) and Yuriy Mileyko (ym4@njit.edu)

4.1 Introduction

There are many statistical methods of tracking single and multiple targets; this manuscript will focus on the state estimation problem. Ideally, a generalization of the recursive Bayes non-linear filter would track and resolve the state(s) of single or multiple targets, but that is currently computationally intractable. The Probability Hypothesis Density (PHD) makes the tracking problem computationally feasible by propagating only the first-order multi-target statistical moments by using a particle filter implementation for the PHD. The problem then becomes one of estimating the targets' state based on the output of the PHD when using a particle filter implementation.

This paper describes one heuristic method for obtaining a state estimator from the PHD. The approach used in this paper, based on the Expectation-Maximization (EM) algorithm, views the PHD distribution as a mixture distribution, and the particles as an i.i.d. sampling from the mixture distribution. Using this, a maximum likelihood estimator for the parameters of the distribution can be generated. The EM seems to work fairly well, particularly when targets are well spaced.

4.2 Problem Description

The problem is one of tracking and identifying a finite set of multiple targets by means of data collected from a set of multiple sensors. The motion of each target is modelled as a discrete time, continuous space Markov process; it is also assumed that targets move independently. The exact number of targets

[1]Lockheed Martin
[2]Washington State University
[3]University of Alberta
[4]Claremont Graduate University
[5]New Jersey Institute of Technology

is unknown and may change with time depending on the corresponding birth/death model. At each time step k, the observations $Z_1^k, \ldots, Z_M^k$ are gathered from the sensor suite. These observations are affected by clutter; the amount of clutter is modelled using a Poisson distribution.

The PHD is a computationally efficient means of solving the tracking problem; however, it does not provide an estimate of the targets' states. Our approach for solving this part of the problem is described in section 4.4.

4.3 Target Tracking and the PHD

Consider a single-sensor, single-target problem. Let x_k be the target state variable at time step k, and z_k the observation at time step k. Assume that x_k is a Markov process with initial distribution $p_0(x_0)$ and transition equation $p_{k+1|k}(x_{k+1}|x_k)$. Also assume that the observations z_k are conditionally independent given the process x_k and of marginal distribution $p(z_k|x_k)$. In this case, the recursive equations for the posterior distribution can be written as

$$\begin{aligned} p_{k+1|k}(x_{k+1}|Z^k) &= \int p_{k+1|k}(x_{k+1}|x_k)p_{k|k}(x_k|Z^k)dx_k \\ p_{k+1|k+1}(x_{k+1}|Z^{k+1}) &= \frac{p(z_{k+1}|x_{k+1})p_{k+1|k}(x_{k+1}|Z^k)}{\int p(z_{k+1}|x_{k+1})p_{k+1|k}(x_{k+1}|Z^k)dx_{k+1}}, \end{aligned}$$

where[6] $Z^k = \{z_1, \ldots, z_k\}$. The target state estimators can be given by

$$\hat{x}^{MAP}_{k+1|k+1} = \arg\sup_x p_{k+1|k+1}(x|Z^{k+1}), \hat{x}^{EAP}_{k+1|k+1} = \int x p_{k+1|k+1}(x|Z^{k+1})dx,$$

where $\hat{x}^{MAP}_{k+1|k+1}$ and $\hat{x}^{EAP}_{k+1|k+1}$ are the Bayes-optimal maximum *a posteriori* and expected *a posteriori* estimators, respectively.

The generalization of the above equations for the multiple-sensor, multiple-target system is not quite obvious. The target state is not just one random vector anymore. The number of targets, as well as their positions, velocities, identities, etc., are all unknown and should be treated as random variables. To deal with this, the tool of finite-set statistics (FISST) can be used. It is based on the following ideas [1]:

- Introduce a notion of a single "global sensor" that encompasses the whole sensor suite.
- Introduce a notion of a "global target" with multi-target state $X = \{x_1, \ldots, x_n\}$.
- Regard the set of observations, $Z = \{z_1, \ldots, z_m\}$, as a "global measurement" of the "global target."
- Use the *multi-sensor/multi-target measurement model*, which is a randomly varying set $\Sigma = T(X) \cup C(X)$, to model multi-target multi-sensor data. Here $T(X)$ denotes targets' data and $C(X)$ denotes clutter.

[6] Conditioning on Z^k is by definition the same as conditioning on $\mathcal{Y}_k = \sigma\{Z^k\}$.

- Use the *multi-target motion model*, which is a random set $\Gamma_{k+1} = \Phi_k(X_x, V_k)$, to model the motion of multi-target systems.

These ideas make it possible to reformulate multi-sensor, multi-target problems as single-sensor, single-target problems. The process of reformulation relies heavily on the notion of the belief-mass function, which is a generalization of the probability-mass function (see [2]). The FISST multi-sensor multi-target differential and integral calculus introduces the *set integral* and the *set derivative*, which are used to state the problem in rigorous mathematical terms. Omitting all the details (in [1] and [2]), the resulting recursive equations are

$$\begin{aligned} p_{k+1|k}(X_{k+1}|Z^{(k)}) &= \int p_{k+1|k}(X_{k+1}|X_k)p_{k|k}(X_k|Z^{(k)})\delta X_k \\ p_{k+1|k+1}(X_{k+1}|Z^{(k+1)}) &= \frac{p(Z_{k+1}|X_{k+1})p_{k+1|k}(X_{k+1}|Z^{(k)})}{\int p(Z_{k+1}|Y)p_{k+1|k}(Y|Z^{(k)})\delta Y}, \end{aligned}$$

where X_k is a multi-target state, $f_{k|k}(X_k|Z^{(k)})$ is a multi-target posterior density, and $f(Z|Y)$ is a multi-sensor, multi-target likelihood function.

Note that the integrals in the above formulae are set integrals, the computation of which is very expensive. Actually, even in single-target problems, the recursive equations are too complicated to calculate explicitly. Therefore, some approximation should be found. It turns out that, when signal-to-noise ratio (SNR) is high enough, the first-order moment, $\hat{f}_{k|k} = \int x f_{k|k}(z|Z^k)dx$, is a good approximation for single-target problems. Unfortunately, it is not possible to use this straightforward generalization for multiple target problems, because the integral $\int X f_{k|k}(X|Z^{(k)})\delta X$ cannot be defined. The problem is resolved by using some function h that maps the state-set X into a vector space. Then, the first-order moment is computed indirectly as

$$E[h(\Gamma_k)] = \int h(X) f_{k|k}(X|Z^{(k)})\delta X.$$

One of the possible choices for the function h is $h(\Gamma_k) = \delta_{\Gamma_k}$, where $\delta_{\Gamma_k}(x) = \sum_{w\in\Gamma_k} \delta_w(x)$. This makes the first-order moment, denoted by $D_{k|k}(x|Z^{(k)})$, the *probability hypothesis density* (PHD). It has the property that $\int_S D_{k|k}(x|Z^{(k)})dx$ is the expected number of targets contained in the region S (for further information about the PHD see [2]). If the SNR and the signal to clutter ratio are high enough, and the multi-target system has a zero covariance, then the PHD is a good approximation for the unnormalized multi-target posterior density, and an explicit recursive equation can be derived for $D_{k|k}(x|Z^{(k)})$ (see [1]). In general, the time update equation depends on the motion model, which normally includes birth and death of targets.

$$D_{k+1|k}(y|Z^{(k)}) = \int (d_{k+1}(x) f_{k+1|k}(y|x) + b_{k+1|k}(y|x)) D_{k|k}(x|Z^{(k)})dx$$

where $d_{k+1}(x)$ is the probability that a target with a state x at time-step k will disappear at time-step $k+1$, and $b_{k+1|k}(y|x)$ is the PHD of the multi-target density $\bar{b}_{k+1|k}(X|x)$ that describes the likelihood that a target with a state x at time-step k will generate a set X of new targets at time step $k+1$.

Assuming that there are no births and deaths, the above equation reduces to

$$D_{k+1|k}(y|Z^{(k)}) = \int f_{k+1|k}(y|x) D_{k|k}(x|Z^{(k)})dx. \tag{4.1}$$

The approximate formula for the observation update is

$$D_{k+1|k+1}(x|Z^{(k+1)}) \simeq \sum_{z\in Z_{k+1}} \frac{p_D f(z|x) D_{k+1|k}(x|Z^{(k)})}{\lambda_{k+1} c_{k+1} + p_D \int f(z|x) D_{k+1|k}(x|Z^{(k)}) dx} + \\ +(1-p_D) D_{k+1|k}(x|Z^{(k)}). \tag{4.2}$$

Thus, an approximation of the distribution of the whole set of targets at each time step can be computed quite efficiently. The question then arises as to how the state of each separate target can be estimated. A new approach to this problem was proposed and is described in the next section along with other existing approaches.

4.4 State Estimation and the EM Algorithm

Using the results of the previous section, it is assumed that at time step k the function $D_{k|k}(x|Z^{(k)})$ is given. Integrating it over the whole space, the expected number of targets can be computed. The question then becomes: how can this information be used to find the state of each target?

One of the existing approaches is based on peak detection. The idea is very simple. When the expected number of targets is M and the function $D_{k|k}(x|Z^{(k)})$ has M local maxima, it is reasonable to assume that targets' states are located at the points where those maxima are achieved (see Figure 4.1). There are some difficulties when targets are located very close to each other; for example, the number of local maxima may be smaller than the expected number of targets (see Figure 4.2).

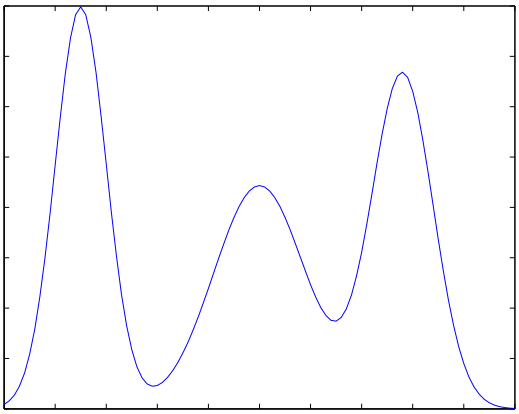

Figure 4.1: Expected number of targets is 3, so the targets are assumed to be located under each peak.

Another approach is based on clustering algorithms, which are ubiquitous in the pattern recognition community. Since the PHD, $D_{k|k}(x|Z^{(k)})$, is represented as a set of particles (see section 4.5.2), it is reasonable to expect that particles are clustered around each target. Applying a clustering algorithm, the loci for the targets can be computed. One of the most popular clustering algorithms is Nearest Neighbour Clustering. It usually gives good results, but the time needed to construct the loci is quadratic in the number of particles. Also, non-intuitive clusters are constructed in some cases.

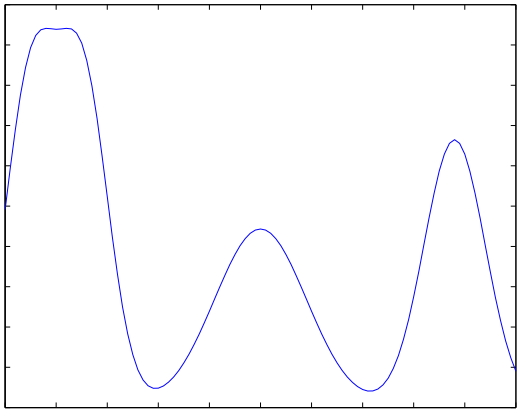

Figure 4.2: Expected number of targets is 4, so there is no natural way to determine targets' states using just the location of the relative maxima.

The new approach is based on the following idea. Suppose that the PHD, $D_{k|k}(x|Z^{(k)})$, is a mixture of Gaussian densities with unknown means and covariance matrices, i.e.,

$$D_{k|k}(x|Z^{(k)}) = \sum_{i=1}^{M} \alpha_i N_i(x|\mu_i, \Sigma_i) = p(x|\Theta),$$

where M is the expected number of targets, $\Theta = \{\alpha_i, \mu_i, \Sigma_i\}_{i=1}^{M}$, and α_i satisfy $\sum_{i=1}^{M} \alpha_i = 1$. The likelihood of these parameters given data $\mathcal{X}$ is

$$\mathcal{L}(\Theta|\mathcal{X}) = \prod_{i=1}^{N} p(x_i|\Theta).$$

It is reasonable to expect that maximization of the likelihood, using the set of particles representing the PHD as the data $\mathcal{X}$, should give the values of the parameters, $\Theta^* = \arg\max_{\Theta} \mathcal{L}(\Theta|\mathcal{X})$, which will provide a good approximation for $D_{k|k}(x|Z^{(k)})$. Then the means, μ_i^*, should be good estimators for the states of targets.

The difficult part of this process is the maximization of the likelihood. The usual algorithms for non-linear optimization may not be efficient in this case, since the number of parameters is quite large. Therefore, the need for more robust techniques arises. One such technique is the Expectation-Maximization (EM) algorithm.

4.4.1 Expectation-Maximization Algorithm

Suppose the data $\mathcal{X}$ for maximum-likelihood estimation is incomplete or has missing values[7]. Assume that there exists a complete data set $\mathcal{Z} = (\mathcal{X}, \mathcal{Y})$, where $\mathcal{Y}$ is missing information. Let

$$p(z|\Theta) = p(x, y|\Theta) = p(y|x, \Theta)p(x|\Theta)$$

[7]Sometimes it is convenient to assume that there are some missing (or hidden) parameters to simplify the expression for the likelihood function.

be a joint density function. Then a new likelihood function is $\mathcal{L}(\Theta|\mathcal{Z}) = \mathcal{L}(\Theta|\mathcal{X},\mathcal{Y}) = p(\mathcal{X},\mathcal{Y}|\Theta)$. Note that it is a random variable, since $\mathcal{Y}$ is supposed to be unknown and random. So, it can be assumed that $\mathcal{L}(\Theta|\mathcal{X},\mathcal{Y}) = h_{\mathcal{X},\Theta}(\mathcal{Y})$ for some function $h_{\mathcal{X},\Theta}$, where $\mathcal{X}$ and Θ are constant and $\mathcal{Y}$ is a random variable.

The EM algorithm consists of two steps.

1. Compute the expected value of the log-likelihood, i.e.

$$Q(\Theta,\Theta^{(i-1)}) = E[\log p(\mathcal{X},\mathcal{Y}|\Theta)|\mathcal{X},\Theta^{(i-1)}],$$

   where $\Theta^{(i-1)}$ are current parameter estimates, $\mathcal{X}$ is the observed data, and $\mathcal{Y}$ is the random variable with respect to which the expected value is computed. Note that Θ is a normal variable, so $Q(\Theta,\Theta^{(i-1)})$ is a normal function of Θ.

2. Maximize the expectation computed in the first step and find

$$\Theta^{(i)} = \arg\max_{\Theta} Q(\Theta,\Theta^{(i-1)}).$$

These two steps are repeated until some stopping criterion is satisfied. It can be demonstrated that each additional iteration increases the log-likelihood; therefore, the algorithm is guaranteed to converge to a local maximum of the log-likelihood function.

4.4.2 Mixture-Density Parameter Estimation

To return to the state estimation problem, recall that the PHD is supposed to be a mixture of Gaussian distributions with unknown parameters. The log-likelihood function is in this case

$$\log(\mathcal{L}(\Theta|\mathcal{X})) = \sum_{i=1}^{N}\log(p(x_i|\Theta)) = \sum_{i=1}^{N}\log\left(\sum_{j=1}^{M}\alpha_j N(x_i|\mu_j,\Sigma_j)\right).$$

Suppose that the data $\mathcal{X}$ is incomplete and that there exists an unobserved datum $\mathcal{Y} = \{y_i\}_{i=1}^{N}$. Let $y_i \in \{1,\dots,M\}$ and $y_i = k$ if the i^{th} sample, x_i, was generated by the k^{th} Gaussian component, i.e. the component with the mean μ_k and the covariance matrix Σ_k. Then the expression for the log-likelihood can be rewritten as

$$\log(\mathcal{L}(\Theta|\mathcal{X},\mathcal{Y})) = \log(P(\mathcal{X},\mathcal{Y}|\Theta)) = \sum_{i=1}^{N}\log(P(x_i|y_i)P(y)) = \sum_{i=1}^{N}\log(\alpha_{y_i}N(x_i|\mu_{y_i},\Sigma_{y_i})).$$

Obviously, the problem is that values of $\mathcal{Y}$ are unknown. Assuming that y_i are random variables, it is possible to apply the EM algorithm. Let $\Theta^g = (\alpha_1^g,\dots,\alpha_M^g,\mu_1^g,\dots,\mu_M^g,\Sigma_1^g,\dots,\Sigma_M^g)$ be some guess.

Omitting a few details (which can be found in [3]) the expressions for Θ^{g+1} are

$$\begin{aligned}
\alpha_l^{g+1} &= \frac{1}{N}\sum_{i=1}^{N} p(l|x_i,\Theta^g) \\
\mu_l^{g+1} &= \frac{\sum_{i=1}^{N} x_i p(l|x_i,\Theta^g)}{\sum_{i=1}^{N} p(l|x_i,\Theta^g)} \\
\Sigma_l^{g+1} &= \frac{\sum_{i=1}^{N} p(l|x_i,\Theta^g)(x_i-\mu_l^{g+1})(x_i-\mu_l^{g+1})^T}{\sum_{i=1}^{N} p(l|x_i,\Theta^g)},
\end{aligned}$$

where

$$p(l|x_i,\Theta^g) = \frac{\alpha_l^g N(x_i|\mu_l^g,\Sigma_l^g)}{\sum_{k=1}^{M}\alpha_k^g N(x_i|\mu_k^g,\Sigma_k^g)}.$$

Only some distributions allow derivation of analytic expressions of Θ^{g+1}, and the Gaussian distribution is one of these. In general, some numerical procedure should be used to maximize the expectation $Q(\Theta,\Theta^g)$. Using the above formulae and some reasonable stopping criterion, the estimates for the targets' states, μ_l^*, $l=1,\ldots,M$, can be found efficiently at each time step.

4.5 Numerical Simulation

To test the state estimation method based on the Gaussian mixture assumption and the EM algorithm, several numerical tests were done. The model used for the tests was a racing car model, and the PHD method was implemented using a simple particle filter.

4.5.1 Model

Suppose cars are racing on a circle of length L with velocities in the range $[-b,-a]\cup[a,b]$, where the sign defines a direction of motion. The initial positions and velocities of the cars are distributed uniformly on intervals $[0,L)$ and $[-b,-a]\cup[a,b]$, respectively. The motion of the cars is described by

$$\begin{aligned}
x_{k+1} &= x_k + v_k, \\
v_{k+1} &= (v_k + N(0,\sigma_v^2))(2B(p)-1),
\end{aligned}$$

where x_k is a position of a car at time step k, v_k is the velocity of a car at time step k, $N(0,\sigma_v^2)$ denotes Gaussian noise with a variance σ_v^2, and $B(p)$ is a binomially distributed random variable (at each time step, it is 1 with probability p and 0 with probability $1-p$). Since the track is a circle, when a car crosses one of the boundaries (i.e., its position becomes greater than L or less than 0) it continues its motion from another boundary. Also note that this motion model is highly non-linear, since velocities randomly change not only in magnitudes, but also in directions.

The observation part of the model is defined as follows: The probability of observing a target (car) is P_D, the amount of clutter has Poisson distribution with a mean λ, and the clutter itself is distributed uniformly on the interval $[0,L)$. The observed data is

$$z_k = x_k + N(0,\sigma_{ob}^2),$$

where $N(0, \sigma_{ob}^2)$ is Gaussian noise with a variance σ_{ob}^2. To simplify computations, there is also a no births, no deaths assumption.

The computations were done using the following values for the parameters. The length of the track, $L = 100$; velocities are confined to the range $[-3, -2] \cup [2, 3]$, so $a = 2, b = 3$; maximum number of targets equals 6; variance of Gaussian noise for velocities, $\sigma_v^2 = 0.1$; probability of retaining the same direction, $p = 0.98$; probability of detecting a target, $P_D = 0.9$; variance of Gaussian noise for observations, $\sigma_{ob}^2 = 3$; mean of Poisson distribution for the amount of clutter, $\lambda = 10$.

4.5.2 Interactive Particle Filter

Note that equations (4.1) and (4.2) contain an integral, whose computation requires a lot of effort. To deal with this problem, a sequential Monte Carlo method can be used. One of these methods leads to the Interactive Particle Filter. At each time step k, the PHD is represented as

$$D_{k|k}(x|Z^{(k)}) = \frac{M_{k|k}}{N} \sum_{i=1}^{N} \delta_{\xi_{k|k}^i}(x),$$

where $\xi_{k|k}^i$ are simulated random samples, also named particles. Initially, $\xi_{0|0}^i$ are i.i.d random samples uniformly distributed in $[0, L) \times ([-b, -a] \cup [a, b])$. The number of particles, N, has to be sufficiently large, so for the presented model $N = 5000$ was used. The initial number of targets can be regarded as a random variable T such that $k = 1, \ldots, M$, where M is the maximum number of targets. Thus, the initial mass of the system can be computed as the expected value of T, i.e. $M_{0|0} = E[T]$. For the presented model $M_{0|0} = 3.5$.

The motion update is done by setting $M_{k+1|k} = M_{k|k}$ and moving each $\xi_{k|k}^i$ according to the motion of the targets, i.e. $\xi_{k+1|k}^i = \xi_{k|k}^i + v_k$. The observation update is more complicated. First, the following integral should be computed

$$I(z) = \int f(z|x) D_{k+1|k}(x|Z^{(k)}) dx = \frac{M_{k+1|k}}{N} \sum_{i=1}^{N} f(z|\xi_{k+1|k}^i).$$

Then, the total mass is

$$M_{k+1|k+1} = \int D_{k+1|k+1}(x|Z^{(k+1)}) dx = \sum_{z \in Z_{k+1}} \frac{P_D I(z)}{\lambda_{k+1} c_{k+1} + P_D I(z)} + (1 - P_D) M_{k+1|k}.$$

The next step is computing weights

$$\omega_{k+1|k+1}^i = \sum_{z \in Z_{k+1}} \frac{P_D f(z|\xi_{k+1|k}^i)}{\lambda_{k+1} c_{k+1} + P_D I(z)} + (1 - P_D) M_{k+1|k}.$$

These weights are used to resample the particles using the following rule

$$P(\xi_{k+1|k+1}^i = \xi_{k+1|k}^j) = \frac{\omega_{k+1|k+1}^j}{\sum_{l=1}^{N} \omega_{k+1|k+1}^l}.$$

Therefore, particles with smaller weights should eventually disappear. The particle filter as well as the EM algorithm was implemented using the Matlab software package. The code for the main routines is given in the Appendix, and results of the computations are presented below.

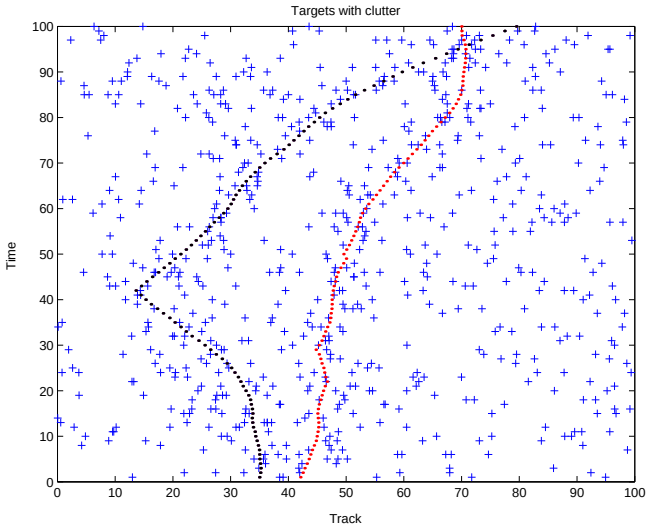

Figure 4.3: Motion of two targets.

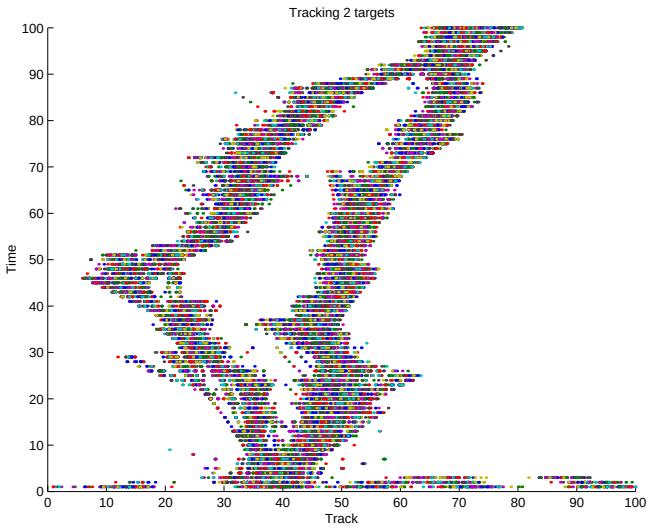

Figure 4.4: Tracking of two targets.

4.5.3 Results

First, the work of the Interactive Particle Filter is presented. In Figure 4.3, observations are shown; trajectories of targets are indicated by dots, while clutter is marked by crosses. The output of the particle filter is illustrated in Figure 4.4. Note how particles eventually cluster around the targets. Taking into account a relatively small number of particles, the PHD gives a good result. Another proof of the efficiency of the PHD is the graph of the total mass of the system, shown in Figure 4.5. Notice that the estimation of the number of targets is quite accurate, since, although the graph oscillates wildly, the average is always close to 2, which is the actual number of targets.

The EM algorithm turns out to be very sensitive to the output of the PHD. In general, it works well when trajectories of targets do not intersect and do not cross the boundaries. This case is shown in Figure 4.6. The difficulties arise when the trajectories intersect or cross the boundaries, the reason being a 'bad' output of the PHD.

The first case is illustrated in Figure 4.7. Note how the inaccurate estimation occurs at the point of intersection. The output of the particle filter for this case is shown in Figure 4.8, and in Figure 4.9, the approximation of the PHD as an unnormalized density function is presented. While the former figure shows no possible obstruction to state estimation, the later figure clarifies the poor performance of the EM algorithm. As the time draws close to the time of intersection, the density function has only one local maximum, a situation which makes identification of two targets very difficult for the EM algorithm.

Figures 4.10, 4.11, and 4.12 illustrate the targets' states estimation, the output of the particle filter, and the unnormalized density function, respectively, for trajectories which cross the boundaries. Notice that the density function has three local maxima that cause the EM algorithm to fail. In all cases, when the particles are not clustered well around the targets, the estimation is not accurate.

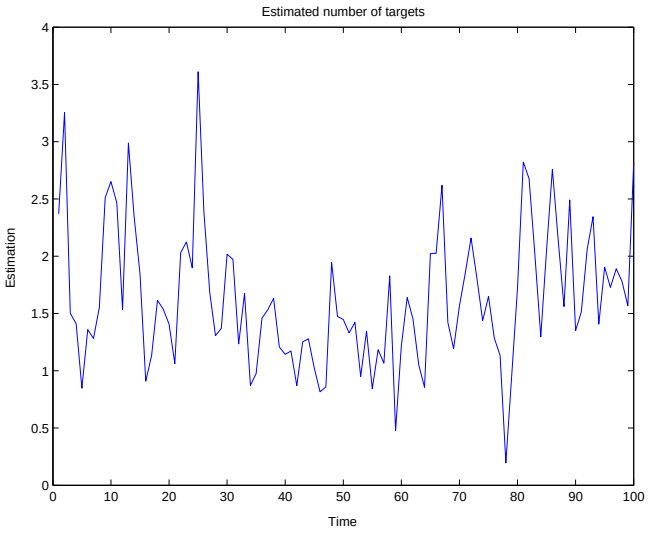

Figure 4.5: Total mass of the system with two targets.

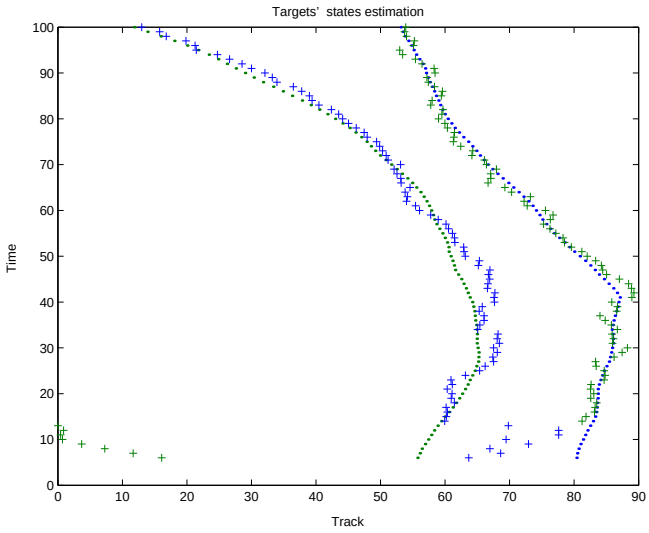

Figure 4.6: State estimation when targets' trajectories do not intersect and do not cross the boundaries. 'Dots' denote exact positions, 'crosses' denote estimations.

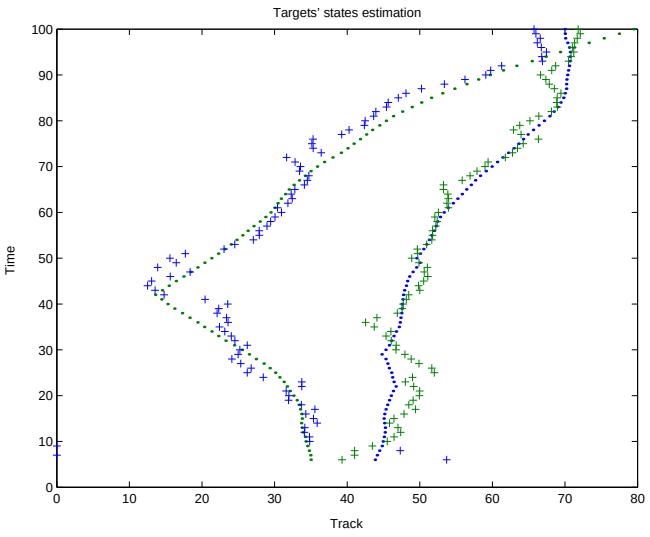

Figure 4.7: State estimation when targets' trajectories intersect. 'Dots' denote exact positions, 'crosses' denote estimations.

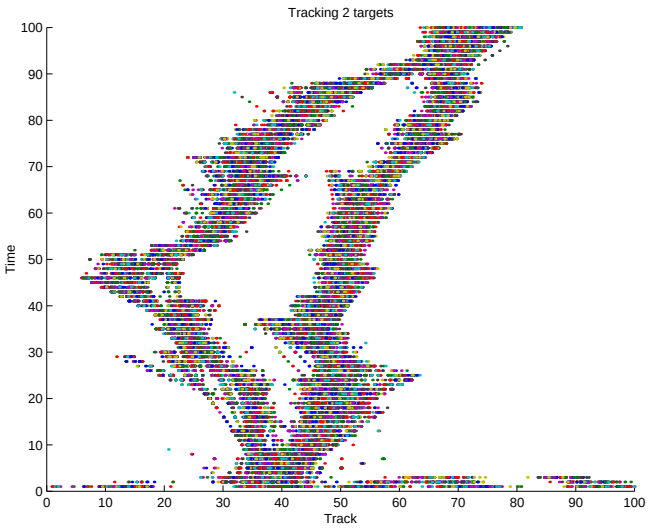

Figure 4.8: Tracking of two targets when their trajectories intersect.

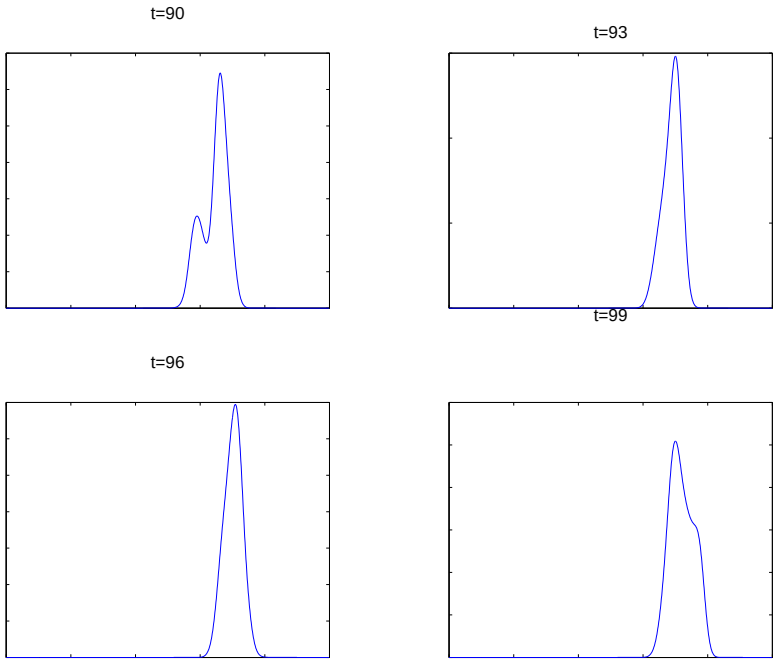

Figure 4.9: Approximation of the PHD as an unnormalized density function for times close to the intersection time.

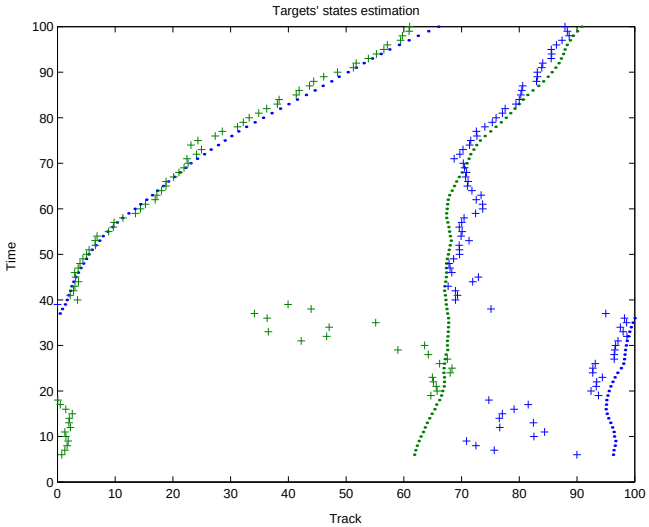

Figure 4.10: State estimation when targets' trajectories cross the boundaries. 'Dots' denote exact positions, 'crosses' denote estimations.

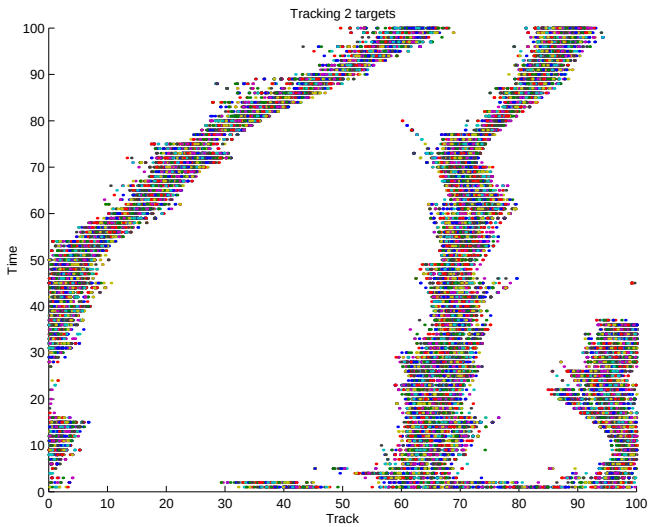

Figure 4.11: Tracking of two targets when one crosses the boundary.

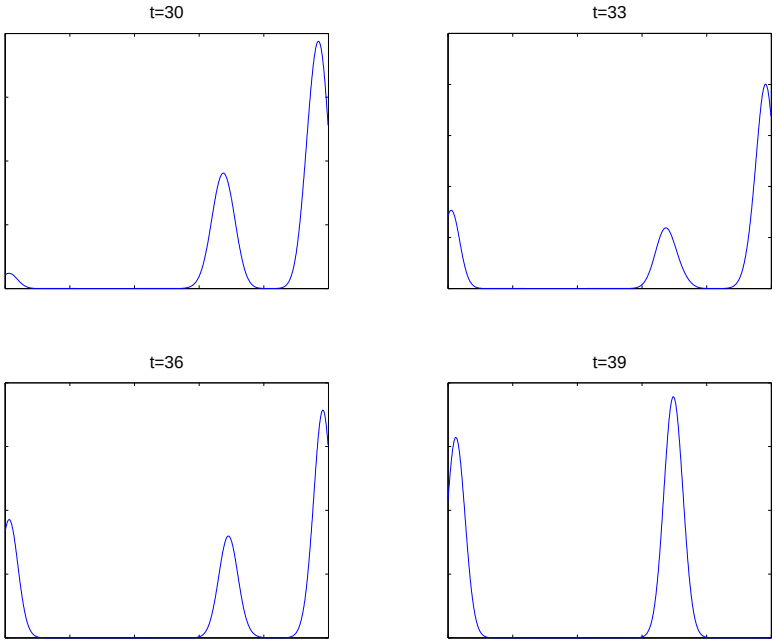

Figure 4.12: Approximation of the PHD as an unnormalized density function for times close to the crossing time.

4.6 Conclusion

In this paper, a new method for targets' states estimation was proposed as a part of a solution to the problem of tracking and identifying multiple targets. The recently proposed Probability Hypothesis Density (PHD) was used to track multiple targets when observations are gathered from multiple sensors. This new method for estimating targets' states is based on the assumption that the PHD can be represented as a mixture of Gaussian densities with unknown means and covariance matrices. The number of Gaussian densities in this mixture is the expected number of targets. Using this representation, the likelihood function can be constructed, and maximization of the likelihood gives the values of the means (for the Gaussian densities) that can serve as estimates for the targets' states. To maximize the likelihood, the Expectation-Maximization (EM) algorithm was employed. Being widely used for the mixture-density parameter estimation problem, the EM algorithm is particularly attractive when the mixture is Gaussian, since analytical expressions for the expectation step can be derived in this case.

The proposed method was tested on a simple racing car model. The results of this simple model provided examples of specific cases in which the targets' states fail to be resolved. Estimations are poor in cases where targets' trajectories either intersect, get very close to each other or cross the boundary. This failure is due to the fact that the EM algorithm is highly sensitive to the output of the PHD. In the case where targets are close to each other, the unnormalized density function (ie., the output of the PHD) collapses to a single peak, making it impossible for the EM algorithm to resolve more than one target's state. In a similar fashion, when a target crosses the boundary, an additional local maximum of the unnormalized density function comes into existence, which again obstructs the work of the

EM algorithm. Additional analysis of the coupling between the PHD and targets' states estimation algorithm should be done in order to overcome these obstacles.

4.7 Future Work

There are a number of possible ways of extending the work that has been presented in this paper. Updating the motion model is a simple way to eliminate the boundary conditions that cause problems with estimating the targets' states. This would eliminate one of two ways that the state estimation algorithm can fail. Aside from this extension to the motion model, there has been some interest in comparing various state estimation methods and testing the ways in which those methods interact with the PHD. In addition, further exploration remains to be made of the PHD.

Updating the motion model is a relatively simple way to overcome the problem posed by the extra maximums of the unnormalized density function that appear when targets cross the boundaries. A variant of the racing car problem, which eliminates the looping at the endpoints, could be used to confirm that the EM algorithm resolves the targets' states correctly when the states are not near each other. Tracking failures of the EM algorithm due to the cyclic boundary conditions are presented in the Results section and illustrated in Figures (4.7) and (4.10).

There are other methods of estimating the targets' states which are not discussed in this manuscript and which can serve to extend this preliminary research. The most common of these is the Nearest Neighbour Clustering method, which was briefly mentioned in the State Estimation and the EM Algorithm section. However, there are also more modern approaches, such as the Cheeseman Clustering and the Fourier/Wavelet based metric minimization methods. A careful and unbiased comparison of these competing methods would be of interest to tracking practitioners in both industry and academia.

4.8 Appendix

The following code is the main routine that is used to track and estimate the targets' state.

```
%-------------------------------------------------------------------------------------------------
% Main Script
%-------------------------------------------------------------------------------------------------

clear;                          % clear all variables
global N;                       % number of particles
global M;                       % number of targets
global PD;                      % probability of detecting a target
global ClM;                     % mean for Poisson distribution; used to generate clutter

% assigning values to the global variables
ClM=5;
PD=1;
N=5000;

M=floor(5*rand)+1;              % the number of targets
Tar=init_states(M);             % getting initial positions of the targets
[Mk,Xk]=init;                   % getting particles and guessing the number of targets
TS = 100;                       % number of time steps

MM=zeros(1,TS);                 % array for storing the mass of the system at each time step
XX=zeros(N,TS);                 % two dimensional array for storing particles at each time step
TT=zeros(M,TS);                 % two dimensional array for storing targets' states at each time step
```

```
stop_flag=-1;                    % a stopping flag

for t=1:TS
  Tar=move(Tar);                 % moving targets
  Obs=observe(Tar);              % observing targets
  [Mk,Xk]=update1(Mk,Xk);        % updating mass and particles,  k|k   -> k+1|k
  [Mk,Xk]=update2(Mk,Xk,Obs);    % updating mass and particles,  k+1|k -> k+1|k+1
  MM(t)=Mk;                      % saving the mass of the system
  XX(:,t)=Xk(1,:)';              % saving particles
  TT(:,t)=Tar(1,:)';             % saving targets

  % EM is applied only after the 6th time step; this 'if' statement provides initial guess
  if(t==6)
    Me=round(mean(MM(1:t)));     % system mass is estimated as an average value of the previous masses
    Mo=round(mean(MM(1:t-1)));   % old estimation for the mass of the system
    Me=2;                        % we fixed the mass, estimation failed when mass varied
    [a,m,d]=get_init_guess1d(Me); % initial guess for EM algorithm

  % this was written to handle the case when the mass changes
  elseif(t>6)
    Me=round(mean(MM(1:t)));     % system mass is estimated as an average value of the previous masses
    Mo=round(mean(MM(1:t-1)));   % old estimation for the mass of the system
    Me=2;                        % we fixed the mass, estimation failed when mass varied
    Mo=2;                        % we fixed the mass, estimation failed when mass varied

    % 'if' statement below will never be called, since we keep Mo and Me equal to 2
    % it was written to handle the case when the mass of the system changes

    if(Me<Mo)                            % if mass becomes less we eliminate elements from a,m,d
      ind=1:length(a);                   % getting indices for the array a
      [s1,s2]=sort(a);                   % sorting a
      keep=setdiff(ind,s2(Mo-Me));       % getting indices for the elements that will be kept
      a=a(keep);                         % updating a
      a=a/sum(a);                        % sum of the elements of a should be 1
      m=m(:,keep);                       % updating m
      d=d(:,keep);                       % updating d
    elseif (Me>Mo)                       % if mass becomes greater
      [mv,mi]=max(a);                    % find the maximum element of array a
      a=a([1:Mo mi*ones(1,Me-Mo)]);      % resampling this elements
      a=a/sum(a);                        % sum of the elements of a should be 1
      m(:,Mo+1:Me)=m(:,mi*ones(1,Me-Mo)); % resampling the corresponding element of m
      d(:,Mo+1:Me)=d(:,mi*ones(1,Me-Mo)); % resampling the corresponding element of d
    end;
  end;

  % EM is applied only after the 6th time step
  if(t>=6)
    [a,m,d]=get_it1d(a,m,d,Xk(1,:));     % applying EM
    RES(1,1:M,t)=Tar(1,:);               % saving targets' positions
    RES(2,1:Me,t)=m;                     % saving estimated targets' positions
  end;
end;
```

Next, the functions implementing the motion and observation models are presented.

```
function y=move(x)
%-------------------------------------------------------------------------------------------------
% implements the following motion model:
% X_k+1=X_k+V_k
% V_k+1=(V_k+N(0,0.1))*(2B(p)-1)
%-------------------------------------------------------------------------------------------------
% x is a collection of two dimensional vectors (X_k,V_k) represented as a 2xN matrix

N=size(x);                                          % size of x
N=N(2);                                             % number of vectors
```

```
y(1,:)=x(1,:)+x(2,:);                                    % updating position
y(2,:)=(x(2,:)+0.1*randn(1,N)).*(2*(rand(1,N)>0.02)-1);   % updating velocity; here p=0.02

% we have wrapping boundary condition, i.e. 100=0
y(1,:)=y(1,:)-100*(y(1,:)>=100)+100*(y(1,:)<0);


function y=observe(T)
%-------------------------------------------------------------------------------------------------------
% implements the following observation model:
% X_o=X+N(0,3)
% plus clutter
%-------------------------------------------------------------------------------------------------------
% T is a set of targets represented as a 2xM matrix

global PD;                                        % probability of detecting a target
n=size(T);                                        % size of T
n=n(2);                                           % number of targets
Pd=(rand(1,n)>(1-PD));                            % Pd indicates what targets are observed
cl=clutter;                                       % generating clutter
cn=length(cl);                                    % number of clutter points
ot=sum(Pd);                                       % number of observed targets
y=zeros(1,cn+ot);                                 % number of observations (clutter+observed targets)
y(1:cn)=cl;                                       % storing clutter

% making sure we observe at least one target
if(ot~=0)                                         % storing observed targets
  y(cn+1:cn+ot)=T(1,find(Pd>0))+3*randn(1,ot);
end;
```

The following functions implement the motion and observation update steps.

```
function [M,X]=update1(m,x)
%-------------------------------------------------------------------------------------------------------
% first update
%-------------------------------------------------------------------------------------------------------
M=m;                                          % mass remains the same
X=move(x);                                    % particles move according to the motion mod


function [M,X]=update2(m,x,Z)
%-------------------------------------------------------------------------------------------------------
% second update
%-------------------------------------------------------------------------------------------------------
% m is the mass of the system
% x contains particles
% Z contains observations

global N;                                     % number of particles
global PD;                                    % probability of detecting a target
global ClM;                                   % mean for Poisson distribution; used for clutter
cc=ClM/100;                                   % useful constant

Ii=I(Z,x(1,:),m);                             % computing integral
M=sum(Ii./(cc+Ii))+(1-PD)*m;                  % updating the mass of the system

% the following code generates weights used for particles resampling
tmp=zeros(length(Z),N);
for i=1:length(Z)
  dif=min(Z(i)-x(1,:),100-Z(i)+x(1,:));       % accounting for wrapping boundary condition
  tmp(i,:)=PD*normpdf(0,dif,3)/(cc+Ii(i));    % main part of the weights formula
end;

S=size(tmp);
% need to handle the case of one target separately
```

```
if(S(1)>1)
  w=sum(tmp)+(1-PD)*m;
else
  w=tmp+(1-PD)*m;
end;

w=w/sum(w);                          % normalizing weights
reg=cumsum(w);                       % generating intervals of lengths corresponding to the weights
samp=rand(1,N);                      % just random numbers

% we use the following algorithm for resampling:
% if the k-th random number (from 'samp') hits the j-th interval (in 'reg')
% then new k-th particle is the old j-th particle
X=zeros(size(x));
for i=1:N
  X(:,i)=x(:,bisearch(reg,samp(i)));   % resampling
end;


function m=bisearch(x,x0)
%-------------------------------------------------------------------------------------------------
% binary search
% searching for the element of array x that is the closest to x0
%-------------------------------------------------------------------------------------------------

N=length(x);                         % length of x
flag=1;                              % useful variable
l=1; r=N;                            % initially, left position is 1 and right position is N

while(l<=r & flag==1)
  m=floor((r+l)/2);                  % computing the middle point
  if(x(m)==x0)                       % if we find x0 exactly
    flag=0;                          % stop the loop
  elseif (x0 < x(m))                 % if x0 lies in the left half
    r=m-1;                           % update the right position
  else
    l=m+1;                           % update the left position
  end;
end;

if(flag==1)                          % a little adjustment
    m=l;
end;


function y=I(z,x,m)
%-------------------------------------------------------------------------------------------------
% computes integral
%-------------------------------------------------------------------------------------------------

global N;                            % number of particles
global PD;                           % probability of detecting a target
y=zeros(size(z));

for i=1:length(z);
    y(i)=sum(normpdf(z(i),x,3));
end;
y=PD*m*y/N;
```

Note, the above group of functions implements the particle filter. Here are functions implementing the EM algorithm. The first is the function that computes the initial guess.

```
function [a,m,d]=get_init_guess1d(M)
%-------------------------------------------------------------------------------------------------
% provides initial guess for EM algorithm; 1-dimensional case
```

```
% M - estimated number of targets
%-------------------------------------------------------------------------------------------------

a=ones(1,M)/M;                          % each a_i is just 1/M
m=100*rand(1,M);                        % means are distributed uniformly; only position is estimated
d=5*ones(1,M);                          % variance is 5^2
```

The following function implements one iteration of the EM algorithm.

```
function [a,m,d]=update1d(a_i, m_i, d_i, x)
%-------------------------------------------------------------------------------------------------
% updates values of a,m,d (1-dimensional case) using the corresponding formulae
%-------------------------------------------------------------------------------------------------

N=length(x);                            % number of particles
Pm=P1(a_i, m_i, d_i, x);                % computing P_l_i
Sp=sum(Pm');
a=Sp/N;                                 % computing a
mn=length(a_i);
m=zeros(1,mn);

% computing means
for L=1:mn
  m(L) = dot(x,Pm(L,:))/Sp(L);
end;

d=zeros(size(d_i));

% computing variances
for L=1:mn
  d(L)=dot(Pm(L,:),(x-m(L)).^2)/Sp(L);
end;
```

And the last function implements the EM algorithm itself.

```
function [a,m,d]=get_it1d(a,m,d,x)
%-------------------------------------------------------------------------------------------------
% computes optimal a,m,d (1-dimensional case)
%-------------------------------------------------------------------------------------------------

a_o=a;m_o=m;d_o=d; % storing old values
flag=1;
epsilon=0.0001;

% iterating while distance is greater than epsilon; flag is used only for the first step
while(dist(a,m,d,a_o,m_o,d_o)>epsilon | flag==1)
 if flag==1
    flag=0;
 end;
    a_o=a;m_o=m;d_o=d;
    % updating values of a,m,d
    [a,m,d]=update1d(a_o,m_o,d_o,x);
end;


function y=dist(a1,m1,d1,a2,m2,d2)
%-------------------------------------------------------------------------------------------------
% distance function (it's just a sum of norms)
%-------------------------------------------------------------------------------------------------

y=norm(a1-a2)+norm(m1-m2)+norm(d1-d2);
```

Bibliography

[1] RONALD MAHLER, *Approximate Multisensor-Multitarget Joint Detection, Tracking, and Identification Using a First-Order Multitarget Moment Statistic*, Technical Report, Lockheed Martin, 2000.

[2] I.R. GOODMAN, R.P.S. MAHLER, AND H.T. NGUYEN, *Mathematics of Data Fusion*, Kluwer Academic Publishers, 1997.

[3] JEFF A. BILMES, *A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models*, Technical report, International Computer Science Institute, Berkeley California, 1998.

[4] BA-NGU VO, SUMEETPAL SINGH, ARNAUD DOUCET, *Sequantial Monte Carlo Implementation of the PHD Filter for Mulit-Target Tracking*. To appear in the proceedings of Fusion 03.