IMPLEMENTATION STRATEGIES FOR PARTICLE FILTER BASED TARGET TRACKING

A Thesis
Presented to
The Academic Faculty

by

Rajbabu Velmurugan

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
May 2007

IMPLEMENTATION STRATEGIES FOR PARTICLE FILTER BASED TARGET TRACKING

Approved by:

Dr. James H. McClellan,
Committee Chair
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. David V. Anderson
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Aaron D. Lanterman
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Jeffrey A. Davis
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Brani Vidakovic
School of Industrial and Systems Engineering
Georgia Institute of Technology

Date Approved: 27 March 2007

ACKNOWLEDGEMENTS

I was able to pursue this research work because of the guidance, help, support, and friendship of many people at different stages. I thank Prof. McClellan for having advised and guided me in pursuing this research. By asking the right questions at the right time he helped me to stay focused on my research. I would also like to thank him for having given me the opportunity to come to Georgia Tech. I have learnt a lot about research and teaching from Prof. McClellan. I thank Prof. Anderson and Prof. Lanterman for their suggestions and feedback at different points in my research. I thank Prof. Davis and Prof. Vidakovic for being on my PhD committee and the useful discussions I had with them.

Two of my friends, Volkan Cevher and Shyam Subramanian, at Georgia Tech played an important part in my research. Volkan helped me by providing the motivation and avenues to get started on my research. I thank him for introducing me to particle filters and target tracking, and for the discussions I had with him and his feedback. Shyam helped me in filling some of the missing pieces in my research and introduced me to MITEs. I thank him for his patient explanations and the useful discussions we had.

I thank Prof. Pragasen Pillay at Clarkson University and Pradhyumnan Ramkumar for having introduced me to digital signal processing.

I thank my friends from Prof.McClellan's group: Sam Li, Milind Borkar, Greg Krudysz, Mubashir Alam, Qiang-Le, Ali Cafer, Faisal Shah, and Karan Chopra for both the academic and non-academic discussions, conversations we had. I thank all friends in the Center for Signal and Image Processing (CSIP) group for making it an interesting place to be. I especially thank Nicolas Gastaud, Badri Vellambi, Yeongseon Lee, Maneli Noorkami, Kevin Chan, Raviv Raich, Martin Tobias, Will Leven, Kerkil Choi, Mina Sartipi, Majid Fozunbal, and Amol Borkar for helping me by providing a friendly and social atmosphere. I also thank Deryck Yeung, Dwi Mansjur, and Mohanned Sinnokrot for the useful discussions I had with them.

I also thank Keith May and Sam Smith for helping with the computing facilities and all the staff members at CSIP for their help.

I thank my friends and former colleagues at The MathWorks. I learnt a lot from them which still helps me in various aspects of my research.

I thank all my FRIENDS and FAMILY with out whom I would not have been at Georgia Tech or pursued this research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

This thesis contributes new algorithms and implementations for particle filter-based target tracking. From an algorithmic perspective, modifications that improve a batch-based acoustic direction-of-arrival (DOA), multi-target, particle filter tracker are presented. The main improvements are reduced execution time and increased robustness to target maneuvers. The key feature of the batch-based tracker is an image template-matching approach that handles data association and clutter in measurements. The particle filter tracker is compared to an extended Kalman filter (EKF) and a Laplacian filter and is shown to perform better for maneuvering targets. Using an approach similar to the acoustic tracker, a radar range-only tracker is also developed. This includes developing the state update and observation models, and proving observability for a batch of range measurements.

From an implementation perspective, this thesis provides new low-power and real-time implementations for particle filters. First, to achieve a very low-power implementation, two mixed-mode implementation strategies that use analog and digital components are developed. The mixed-mode implementations use analog, multiple-input translinear element (MITE) networks to realize nonlinear functions. The power dissipated in the mixed-mode implementation of a particle filter-based, bearings-only tracker is compared to a digital implementation that uses the CORDIC algorithm to realize the nonlinear functions. The mixed-mode method that uses predominantly analog components is shown to provide a factor of twenty improvement in power savings compared to a digital implementation. Next, real-time implementation strategies for the batch-based acoustic DOA tracker are developed. The characteristics of the digital implementation of the tracker are quantified using digital signal processor (DSP) and field-programmable gate array (FPGA) implementations. The FPGA implementation uses a soft-core or hard-core processor to implement the Newton search in the particle proposal stage. A MITE implementation of the nonlinear DOA update function in the tracker is also presented.

CHAPTER I

INTRODUCTION

Target tracking, as defined in this thesis, is estimating moving targets' states using noisy measurements obtained at a single observation point or node. The target states consist of target positions and motion parameters in the 2-D space. The measurements can be either single or batch measurements. The individual measurement in a batch is either an acoustic direction-of-arrival (DOA) estimate or a radar range estimate. The state estimation is set up as a state-space-based Bayesian estimation problem where the underlying state and observation models could both be nonlinear and non-Gaussian.

Particle filters or sequential Monte Carlo (SMC) methods use a set of weighted state samples, called particles, to approximate the posterior probability distribution in a Bayesian setup. They provide nearly optimal state estimates in the case of nonlinear and non-Gaussian systems, unlike Kalman filter-based approaches. Because particle filters do not approximate nonlinearities or non-Gaussian noise in the system and use a large number of particles, they tend to be computationally complex.

This thesis addresses two cases relevant to particle filter-based target tracking. The first case is a low-power implementation for a simple particle filter algorithm. The particular algorithm considered is a simple bearings-only single-target tracker that uses the state update to propose particles, and measurements are the target DOAs. The second case is a real-time implementation for a more complicated particle filter tracking algorithm introduced in [28]. This algorithm is a batch-based multi-target tracking algorithm that uses a Gaussian approximation of the full-posterior as the proposal function. The measurements in this case are batches of acoustic DOAs or radar ranges.

The use of batch measurements was introduced by Cevher [28] to alleviate the data association problem in multi-target tracking and is similar to the shape-from-motion algorithm in computer vision. Associating measurements to targets, data association, is a

difficult problem inherent to multi-target tracking. If there are missing measurements or clutter in the measurements the association and tracking become quite difficult. For Kalman filter-based approaches, this problem has been addressed [10] by using joint probabilistic data association (JPDA) techniques [34] and multiple hypothesis tracking (MHT) [101]. A particle filter-based multi-target tracker using the JPDA approach has been developed in [125]. Special pruning techniques to reduce the number of measurement-to-target hypotheses should be used to make such data association approaches practically feasible [85].

From an algorithmic perspective, this thesis improves the acoustic DOA batch-based particle filter algorithm [28] and compares it to the extended Kalman filter (EKF) and the Laplacian filter. As part of this research, we develop a system that uses the DOA tracker along with an initialization algorithm that estimates the number of targets. We also develop a range-only multi-target tracker that uses a batch of radar range measurements. Though not developed in this thesis, the DOA tracker and range-only tracker can be used in a multi-sensor joint tracking system to improve state estimation.

From an implementation perspective, this thesis develops hardware implementations for the efficient use of particle filters in target tracking. When particle filter-based algorithms are to be deployed for real-time and power constrained applications, special implementations are needed. Such implementations have to be efficient in speed, tracking accuracy, and power. The particle filter algorithm has four main stages: particle proposal, particle weight computation, state estimation, and resampling. Parallelization of the particle proposal and weight evaluation stages can be used to reduce execution speed because computations for one particle do not influence the others. However, the resampling and state estimation stages cannot be readily parallelized. Earlier work in [17] addressed this issue by providing special-purpose hardware for the resampling stage. The authors of [17], [5] developed new resampling algorithms and a distributed architecture for parallel resampling. They also addressed finite word length effects in resampling and particle filters. Though [17] presents power dissipation results, the impact of the architecture on the power consumed in computations is not explicitly considered.

One goal of this thesis is to address the issue of low-power computations for particle

filters. In one case, we propose using analog signal processing techniques to implement the nonlinear functions in a particle filter-based bearings-only tracking algorithm to reduce power consumption. These nonlinear functions are implemented in the analog domain using multiple-input translinear element (MITE) networks [88]. The MITE networks used in this research were synthesized by Subramanian [115] in the collaborative analog and digital signal processing (CADSP) group at Georgia Institute of Technology. The configurability of analog circuits is one of the drawbacks preventing analog computational circuits from being used widely. However, recent work [1] has led to the development of field-programmable analog arrays (FPAAs) that use MITEs as the basic component. FPAAs provide configurability and the subthreshold operation of MITEs leads to low-power operation.

Another goal of this thesis is to address the case of a real-time implementation of a batch measurement-based particle filter for multi-target tracking. We derive analytical formulae for the complexity and analyze the fixed-point aspects of the algorithm. We develop a floating-point DSP implementation of the algorithm and analyze its performance. We also present the impact of system parameters on various implementation performance measures of the algorithm such as complexity, speed, and memory requirement. Using the Xilinx System Generator tool, we implement most stages of the batch-based tracker on a field-programmable gate array (FPGA). We compare the FPGA implementation of the batch-based tracker to a FPGA implementation of the bearings-only tracker. The particle proposal and weight evaluation stage in the tracker use a nonlinear function to update the DOA state. This DOA update function is next implemented in the analog domain using MITEs. We present simulation results to compare the performance of the various implementations of the batch-based tracker.

The remaining sections of this chapter provide the necessary background and past research relevant to the research presented in later chapters. In Section 1.1 we provide a brief introduction to Bayesian estimation and present particle filters as a tool to perform Bayesian estimation. A brief review of multi-target tracking and the target tracking models considered in this research are presented in Section 1.2. Analog and digital implementation aspects relevant to the research pursued here are presented in Section 1.3. We conclude

this chapter with an outline of this thesis and specific contributions made by this research in Section 1.4.

1.1 Bayesian Estimation

This section provides a brief introduction to particle filters [46], [103], [48], as their application and implementation is the focus of this research. Particle filters are a tool to perform Bayesian estimation. Consider a system with a hidden state x and an observation y related to the state x. In real world applications the observation y is a random variable as it is corrupted by sensor noise or by errors in the measurement. In a probabilistic setting, any information about the state x before any observation is made can be described using a prior distribution $p(x)$. The conditional probability density of y depends on x and is given by $p(y|x)$ and is called the likelihood function. The likelihood is the model for the system because it tells how y depends on x. Bayes' theorem states that, given the observation y, the conditional distribution of x defines the posterior distribution

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} \tag{1.1}$$

where $p(y)$ is the total probability of the observed data and actually plays the role of a normalization constant. In the discrete case

$$p(y) = \sum p(y|x)p(x)$$

The posterior distribution represents what is known about x after the observation y. The minimum-mean-squared-error (MMSE) estimate or the Bayes' estimate of x given the observation y is the conditional mean of the parameter x given the observation y [100],

$$\hat{x} = E(x|y) = \int_x xp(x|y)dx. \tag{1.2}$$

1.1.1 Importance sampling

The posterior distribution $\pi(x) \triangleq p(x|y)$ considered in Bayes' theorem is often analytically intractable. Monte Carlo methods provide a means to address this difficulty by using a large set of samples drawn from such a distribution to obtain a numerical approximation.

Generally, it is not possible to draw samples directly from an arbitrary probability density function. Several sampling techniques are available to generate samples from arbitrary distributions [102]. Importance sampling is one such technique, where a candidate distribution $q(x)$ is chosen such that it is easy to draw statistically independent samples from $q(x)$. This distribution, called the importance or proposal density, must have a support that contains the support of $\pi(x)$, i.e.,

$$\pi(x) > 0 \Rightarrow q(x) > 0 \text{ for all } x \in \mathbb{R}^{n_x}. \tag{1.3}$$

The Monte Carlo approximation for $\pi(x)$ with N samples can then be defined as

$$\hat{\pi}_N(dx) = \frac{\sum_{i=1}^{N} w(x^{(i)})\delta_{x^{(i)}}(dx)}{\sum_{i=1}^{N} w(x^{(i)})}, \tag{1.4}$$

with the importance weights $w(x^{(i)})$ defined as

$$w(x^{(i)}) = \frac{\pi(x^{(i)})}{q(x^{(i)})}, \tag{1.5}$$

and the indicator function $\delta(dx)$ defined as

$$\delta_{x^{(i)}}(dx) = \begin{cases} 1, & x^{(i)} \in dx \\ 0, & \text{otherwise} \end{cases} \tag{1.6}$$

where dx is a small finite region surrounding a value of interest x.

This procedure, importance sampling, converts samples from the distribution $q(x)$ to the required distribution $\pi(x)$. The normalization in (1.4) is required, as the desired distribution is not completely known but known up to a constant. Using the approximation (1.4) for $\pi(x)$ we can estimate functions of the state dependent on the distribution as

$$f_N = \frac{\sum_{i=1}^{N} w(x^{(i)}) f(x^{(i)})}{\sum_{i=1}^{N} w(x^{(i)})}. \tag{1.7}$$

This estimate in (1.7) is asymptotically unbiased [47] and

$$\lim_{N\to\infty} f_N \to E_{\pi(x)}\{f(x)\}. \tag{1.8}$$

In a probabilistic dynamic system, as the parameter x evolves over time, the evolving posterior density functions are represented as $\pi(x_t)$. The Bayesian importance sampling

procedure described above can be used to draw samples from another distribution and approximate $\pi(x_t)$. In most applications, the difference between $\pi(x_t)$ and $\pi(x_{t+1})$ is small and is caused by incorporating new measurements. Hence, samples used at time t can be recursively re-weighted to construct the set of samples at time $t+1$. Particle filters use such a procedure to sequentially and recursively estimate the posterior distribution as time evolves.

1.1.2 State-space representation

The target-tracking problem can be formulated as a dynamic state-space problem. The target states of interest can be estimated by applying the Bayesian approach outlined in Section 1.1. We are interested in inferring the state x_t at time t based on measurements $y_1, y_2, \ldots, y_t$ at discrete time points $0, 1, \ldots, t$. We assume the following discrete, state-space model in this thesis. The underlying state of interest x_t changes over time according to the system equation

$$x_t = f_t(x_{t-1}, u_t) \tag{1.9}$$

where u_t is a sequence of independent random variables with a known distribution that represents the system noise. At time $t = 0$ the state has a prior density $p(x_0)$. At any time t the measurement y_t is related to the state x_t through the relation

$$y_t = g_t(x_t, v_t) \tag{1.10}$$

where v_t is a sequence of random variables with a known distribution and represents the measurement noise. This equation is referred to as the measurement or observation equation. The sequences u_t and v_t are also assumed to be independent. The functions f_t and g_t and the distributions of the noise terms are assumed known.

If the functions f_t and g_t are linear and the noise distributions are Gaussian, in (1.9) and (1.10), the optimal recursive estimate for x_k can be obtained using the Kalman filter. The posterior distribution will be Gaussian and the recursion for the Kalman filter allows the efficient update of the mean and the variance of the Gaussian distribution with each received measurement [82], [7]. If f_t and g_t are nonlinear functions, the extended Kalman

filter or variations of it can be used to obtain the state estimates [7]. If instead the system and measurement functions are nonlinear and the noises are non-Gaussian, the particle filter provides a recursive solution to estimate x_k, but the Kalman filter fails.

1.1.3 Particle filter

This section presents details of the recursive particle filter; the discussion and derivation follows [103]. Let us denote the cumulative state and measurements up to time t as $\mathbf{X}_t = \{x_j, j = 0, \ldots, t\}$ and $\mathbf{Y}_t = \{y_j, j = 0, \ldots, t\}$. The Bayesian approach to make an inference about the state x_t, given the measurements up to time t, is to calculate the posterior distribution for x_t conditional on the measurements up to time t, $p(x_t|\mathbf{Y}_t)$. The joint posterior density at time t is denoted by $p(\mathbf{X}_t|\mathbf{Y}_t)$. Let $\left\{\mathbf{X}_t^{(i)}, w_t^{(i)}\right\}_{i=1}^{N}$ denote a set of weighted particles that characterizes $p(\mathbf{X}_t|\mathbf{Y}_t)$, where $\left\{\mathbf{X}_t^{(i)}, i = 1, \ldots, N\right\}$ is a set of support points, $\left\{w_t^{(i)}, i = 1, \ldots, N\right\}$ are the associated weights, and N is the number of particles. The weights have to be normalized such that $\sum_i w_t^{(i)} = 1$. Using the Monte Carlo approach (1.4), the joint density can be approximated as follows:

$$p(\mathbf{X}_t|\mathbf{Y}_t) \approx \sum_{i=1}^{N} w_t^{(i)} \delta(\mathbf{X} - \mathbf{X}_t^{(i)}) \tag{1.11}$$

The normalized weights $w_t^{(i)}$ are chosen using the principle of importance sampling described earlier. If the samples are drawn from an importance density $q(\mathbf{X}_t|\mathbf{Y}_t)$, then according to (1.5):

$$w_k^{(i)} \propto \frac{p(\mathbf{X}_t^{(i)}|\mathbf{Y}_t)}{q(\mathbf{X}_t^{(i)}|\mathbf{Y}_t)}. \tag{1.12}$$

If at time $t-1$ we have samples constituting an approximation to $p(\mathbf{X}_{t-1}|\mathbf{Y}_{t-1})$, then as the measurement y_t is received at time t, we wish to approximate $p(\mathbf{X}_t|\mathbf{Y}_t)$ with a new set of samples. If the importance density is chosen to factorize such that

$$q(\mathbf{X}_t|\mathbf{Y}_t) = q(x_t|\mathbf{X}_{t-1}, \mathbf{Y}_t) q(\mathbf{X}_{t-1}|\mathbf{Y}_{t-1}) \tag{1.13}$$

then the samples $\mathbf{X}_t^{(i)} \sim q(\mathbf{X}_t|\mathbf{Y}_t)$ can be obtained by augmenting each of the existing

samples $\mathbf{X}_{t-1}^{(i)} \sim q(\mathbf{X}_{t-1}|\mathbf{Y}_{t-1})$ with the new state $x_t^{(i)} \sim q(x_t|\mathbf{X}_{t-1}, \mathbf{Y}_t)$. The distribution $p(\mathbf{X}_t|\mathbf{Y}_t)$ is expressed in terms of the prior distribution and data likelihood as

$$\begin{aligned} p(\mathbf{X}_t|\mathbf{Y}_t) &= \frac{p(y_t|\mathbf{X}_t, \mathbf{Y}_{t-1})p(\mathbf{X}_t|\mathbf{Y}_{t-1})}{p(y_t|\mathbf{Y}_{t-1})} \\ &= \frac{p(y_t|\mathbf{X}_t, \mathbf{Y}_{t-1})p(x_t|\mathbf{X}_{t-1}, \mathbf{Y}_{t-1})p(\mathbf{X}_{t-1}|\mathbf{Y}_{t-1})}{p(y_t|\mathbf{Y}_{t-1})} \\ &= \frac{p(y_t|x_t)p(x_t|x_{t-1})}{p(y_t|\mathbf{Y}_{t-1})} p(\mathbf{X}_{t-1}|\mathbf{Y}_{t-1}) \qquad (1.14) \\ &\propto p(y_t|x_t)p(x_t|x_{t-1})p(\mathbf{X}_{t-1}|\mathbf{Y}_{t-1}). \qquad (1.15) \end{aligned}$$

The weight update equation can be obtained by substituting (1.13) and (1.15) in (1.12) as

$$\begin{aligned} w_t^{(i)} &\propto \frac{p(y_t|x_t^{(i)})p(x_t^{(i)}|x_{t-1}^{(i)})p(\mathbf{X}_{t-1}^{(i)}|\mathbf{Y}_{t-1})}{q(x_t^{(i)}|\mathbf{X}_{t-1}^{(i)}, \mathbf{Y}_t)q(\mathbf{X}_{t-1}^{(i)}|\mathbf{Y}_{t-1})} \\ &= w_{t-1}^{(i)} \frac{p(y_t|x_t^{(i)})p(x_t^{(i)}|x_{t-1}^{(i)})}{q(x_t^{(i)}|\mathbf{X}_{t-1}^{(i)}, \mathbf{Y}_t)}. \qquad (1.16) \end{aligned}$$

In the target-tracking case, we are only interested in the estimate of $p(x_t|\mathbf{Y}_t)$ at each time step, and only $x_t^{(i)}$ need to be stored, discarding the past histories $\mathbf{X}_{t-1}^{(i)}$ and $\mathbf{Y}_{t-1}^{(i)}$. Hence, the modified weight is then

$$w_t^{(i)} \propto w_{t-1}^{(i)} \frac{p(y_t|x_t^{(i)})p(x_t^{(i)}|x_{t-1}^{(i)})}{q(x_t^{(i)}|x_{t-1}^{(i)}, y_t)} \qquad (1.17)$$

from which the posterior density can be approximated as

$$p(x_t|\mathbf{Y}_t) \approx \sum_{i=1}^{N} w_t^{(i)} \delta(x_t - x_t^{(i)}). \qquad (1.18)$$

The recursive propagation of the weights $w_t^{(i)}$ and support points $x_t^{(i)}$ with the reception of sequential measurements is referred to as the sequential importance sampling (SIS) particle filtering.

The best choice for the importance density function would be the posterior distribution. With the importance density of the form in (1.13), the variance of the importance weights increases over time [48]. This increase in variance leads to the degeneracy of the SIS particle filter. In practice, after a certain number of recursions, all but one particle will have extremely small normalized weights. A suitable measure of variation for the importance

weights or degeneracy of an algorithm is the effective sample size N_{eff} and estimated as [48]:

$$\hat{N}_{\text{eff}} = \frac{1}{\sum_{i=1}^{N} \left(w_t^{(i)}\right)^2} \tag{1.19}$$

where $w_t^{(i)}$ is the normalized weight obtained using (1.16). The value of N_{eff} lies in the interval $1 \leq N_{\text{eff}} \leq N$, so a small N_{eff} indicates degeneracy.

1.1.3.1 Resampling

Resampling is a strategy to overcome the degeneracy of samples in SIS and is a crucial step in particle filtering algorithms when N_{eff} falls below a threshold N_{thr} which is predetermined [48]. Resampling eliminates particles with low importance weights and replicates samples with high importance weights. It involves mapping the set of weighted particles $\left\{x_t^{(i)}, w_t^{(i)}\right\}$ into a new set of particles $\left\{\hat{x}_t^{(i)}, 1/N\right\}$ with uniform weights. The new set $\left\{\hat{x}_t^{(i)}\right\}_{i=1}^{N}$ is generated by resampling, with replacement, N times from an approximate discrete representation of $p(x_t|\mathbf{Y}_t)$ given by

$$p(x_t|\mathbf{Y}_t) \approx \sum_{i=1}^{N} w_t^{(i)} \delta(x_t - x_t^{(i)}) \tag{1.20}$$

so that $P\left\{\hat{x}_t^{(i)} = x_t^{(j)}\right\} = w_t^{(j)}$, where $P()$ represents probability. Several resampling schemes have been proposed in the literature [46], [71]. More recently, efficient hardware for performing resampling has been introduced in [13]. The pseudo-code for one such resampling algorithm is given in Table 1.1.

1.1.3.2 Choice of proposal or importance function

The choice of importance function plays a critical part in the performance and complexity of particle filter algorithms [46], [3]. We provide a brief introduction to two standard choices.

The optimal importance density function that minimizes the variance of importance weights, conditioned upon $x_{t-1}^{(i)}$ and y_t, has been shown to be [48]

$$\begin{aligned} q(x_t|x_{t-1}^{(i)}, y_t) &= p(x_t|x_{t-1}^{(i)}, y_t) \\ &= \frac{p(y_t|x_t, x_{t-1}^{(i)}) p(x_t|x_{t-1}^{(i)})}{p(y_t|x_{t-1}^{(i)})}. \end{aligned} \tag{1.21}$$

Table 1.1: Pseudo-code for a resampling algorithm [71].

---

1. Heapsort the particles $\left\{x_t^{(i)}, w_t^{(i)}\right\}$ in an ascending order according to their weights, $w_t^{(i)} \rightarrow \tilde{w}_t^{(i)}$, to form $\left\{\tilde{x}_t^{(i)}, \tilde{w}_t^{(i)}\right\}$
2. Generate $\omega \sim \mathcal{U}[0,1)$.
3. For $j = 1, 2, \ldots, N$
   a. $u^{(j)} = \frac{j-\omega}{N}$,
   b. Find i, satisfying $\sum_{l=1}^{i-1} \tilde{w}_t^{(i)} < u^{(j)} \leq \sum_{l=1}^{i} \tilde{w}_t^{(i)}$,
   c. Assign $\hat{x}_t^{(j)} = \tilde{x}_t^{(j)}$ and $\hat{w}^{(i)} = 1/N$, and
   obtain the resampled set $\left\{\hat{x}_t^{(i)}, \hat{w}_t^{(i)}\right\}$

Substituting (1.21) in (1.16) results in

$$w_t^{(i)} \propto w_{t-1}^{(i)} p(y_t | x_{t-1}^{(i)}). \tag{1.22}$$

In order to use the optimal importance function, it must be possible to sample from $p(x_t | x_{t-1}^{(i)}, y_t)$ and evaluate $p(y_t | x_{t-1}^{(i)})$. In the general case, one of these may not be straightforward [48]. However, there are some special cases where the use of the optimal importance function is possible. One such case is a class of models for which $p(x_t | x_{t-1}^{(i)}, y_t)$ is Gaussian. In some models, a suboptimal approximation to the optimal importance function is possible [48].

A suboptimal, but simple, choice for the importance function is to use the prior distribution [48],

$$q(x_t | x_{t-1}^{(i)}, y_k) = p(x_t | x_{t-1}^{(i)}). \tag{1.23}$$

Substituting (1.23) in (1.16) yields

$$w_t^{(i)} \propto w_{t-1}^{(i)} p(y_t | x_t^{(i)}). \tag{1.24}$$

The sampling importance resampling (SIR) particle filter introduced in [55] under the name *bootstrap* filter is a simple SIS filter that uses the suboptimal importance density (1.23).

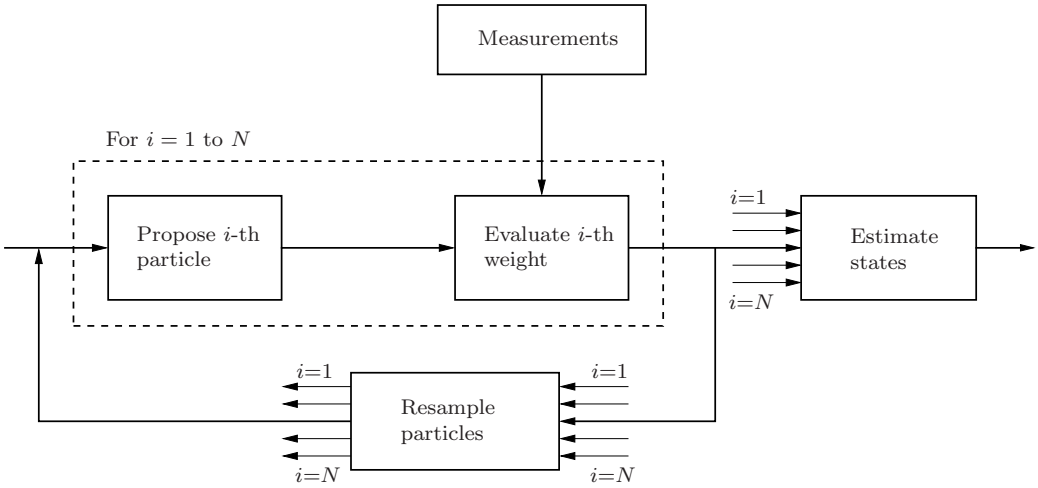


Figure 1.1: Block diagram showing the major computational steps in a sampling importance resampling (SIR) particle filter algorithm.

Pseudo-code for the SIR particle filter algorithm is given in Table 1.2. A block diagram of this is shown in Figure 1.1.

In this thesis we use the SIR filter in a bearings-only single-target tracking algorithm in Chapter 4. On the other hand, the batch-based multi-target tracking in Chapters 2 and 3 uses the optimal importance density function. Furthermore, the multi-target tracker uses a Gaussian approximation of the full-posterior as the importance function to propose particles [48].

1.2 Target Tracking

Target tracking is the processing of measurements obtained from a target in order to maintain an estimate of its current state [7]. It has applications in different fields, and many algorithms have been developed based on the type of measurement or target [11], [8]. The measurements are acoustic, radar, video, or seismic signals obtained at a sensor and can be noisy. They are either used as raw measurements or processed to provide estimates of other parameters that are then used as measurements. The moving target is either single or multiple, vehicle or person. The target state consists of its position in space and its velocity, or in polar coordinates range and angle with respect to the sensor.

Table 1.2: Sampling Importance Resampling (SIR) particle filter.

Given the observed data y_k at k, do

1. For $i = 1, 2, \ldots, N$,
   Sample particles: $x_k^{(i)} \sim q(x_k | x_{k-1}, y_k)$.
2. For $i = 1, 2, \ldots, N$,
   Calculate the importance weights: $w_k^{(i)} = \frac{p(y_k|x_k^{(i)})p(x_k^{(i)}|x_{k-1}^{(i)})}{q(x_k^{(i)}|x_{k-1}^{(i)}, y_k)}$.
   For $i = 1, 2, \ldots, N$,
   Normalize the weights: $\tilde{w}_k^{(i)} = \frac{w_k^{(i)}}{\sum_{j=1}^{N} w_k^{(i)}}$.
3. Calculate the state estimates: $E\{f(x_t)\} = \sum_{i=1}^{N} \tilde{w}_t^{(i)} f(x_t^{(i)})$.
4. Resample $\left\{x_k^{(i)}, \tilde{w}_k^{(i)}\right\}$ to obtain new set of particles $\left\{x_k^{(j)}, w_k^{(j)} = \frac{1}{N}\right\}$.

1.2.1 Past research

Target maneuvers, clutter or missing measurements, and the accuracy of dynamic models are issues that need to be handled in target tracking. In the case of clutter and multiple targets, the association of measurements to targets needs to be handled. This is referred to as data association. Target tracking in clutter has been done using the probabilistic data association filter (PDAF). The PDAF calculates the probabilistic data association (PDA), which consists of the probabilities for all validated measurements corresponding to the targets of interest. Target maneuvers are often handled using multiple models and switching among these models to better represent target dynamics as in the multi-mode (MM) filter or interacting multi-mode (IMM) filters [8], [10].

In multi-target tracking, in addition to data association, the number of targets being tracked also needs to be identified. Several approaches to identify the number of targets have been proposed in the literature [8], [10]. While some approaches explicitly determine the number of targets, other approaches do this implicitly during tracking. For Kalman filter based tracking several strategies have been suggested. These methods form various data association hypotheses and combine or propagate them over time. One technique is the joint

probabilistic data association (JPDA), an extension of the PDA, in which the number of targets is known. This is a target-oriented association filter that evaluates the measurement-to-target association probabilities using the latest set of measurements. Another technique, the multiple hypothesis tracker (MHT), is effective in tracking multiple targets in a cluttered environment [101] where the number of targets is not known ahead of time. This is a measurement-oriented filter that considers the association of sequences of measurements and evaluates the probabilities of all the association hypotheses. The complexity of the MHT increases exponentially with time unless the number of hypotheses is limited using pruning or merging methods [8].

Because particle filters handle nonlinear models and non-Gaussian noise, they have been used in multi-target tracking more recently. The multi-target tracking considered in this research differs from recent work [94], [67] in the observation model used. In [94], the number of targets is estimated by monitoring events represented using regions of interest (ROI) in the surveillance region, tracking is performed using particle filter or the SMC method, and measurement-to-target association is done using a 2-D data assignment algorithm. The measurements are bearing and range at a single sensor. The estimated target states are the positions x and y, velocities v_x and v_y. The observation model used in this thesis is based on image template-matching ideas to perform the data association. This is similar to shape-from-motion algorithms in computer vision or active contour tracking [63].

1.2.2 Bearings-only tracker and batch measurement-based tracker

The estimates in target tracking are obtained by setting up the target tracking problem as a state-space problem as defined in Section 1.1.2. Depending on the dynamic models for target motion and measurements, a Bayesian approach using either the Kalman filter or particle filter can be used. In this thesis, we will concentrate on the particle filter approach and use the EKF approach for comparison. The particle filter is applied to two different tracking problems. The first is a single-target, bearings-only tracking and the second a multi-target, batch measurement-based tracking. In both problems the target is assumed to move with constant velocity, the sensor is stationary, and the geometry is as shown in Figure 1.2.

Though individual measurements in both problems are DOAs or bearings, the assumed observation models are different. A brief description of the main differences between the two problems, as considered in this thesis, is provided here. Later, in Chapters 2 and 4, the individual models will be described in detail.

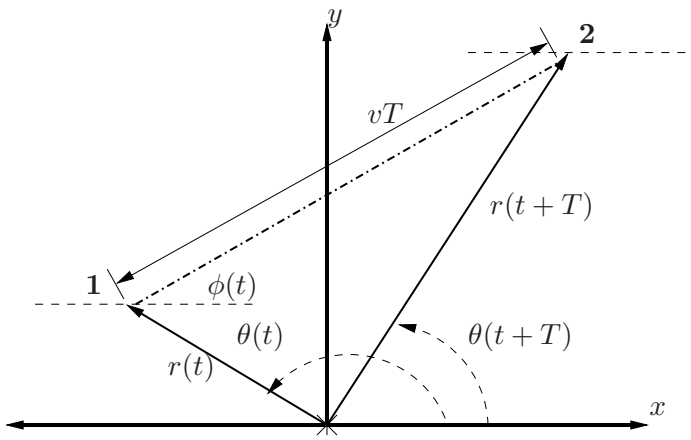


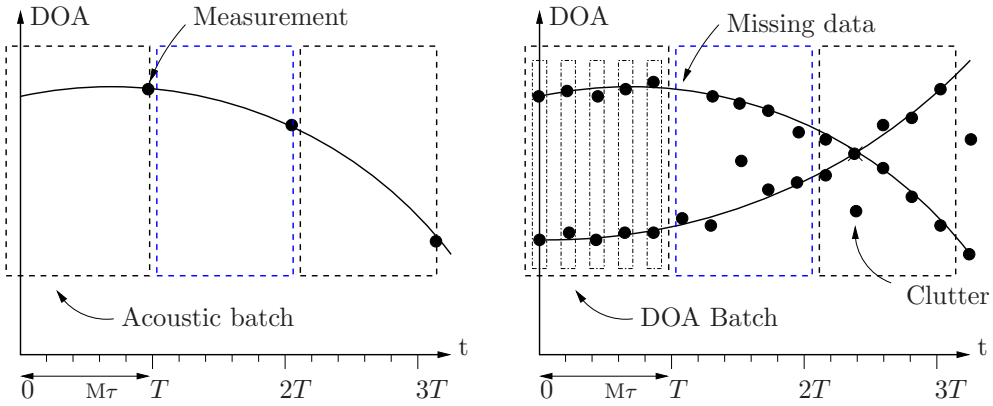
Figure 1.2: Geometry for constant velocity target motion. $\theta(t)$ denotes target DOA, $r(t)$ denotes target range, and $\phi(t)$ denotes the target heading direction when the target is at position denoted by **1** in the x-y plane. The target velocity is represented by v and the period over which the velocity is assumed constant is T.

The bearings-only tracking problem considered here is similar to the approach in [55], but differs from most bearings-only tracking problems where the sensor platform also moves to track targets that maneuver [103]. The target is assumed to follow a constant velocity model in the x-y space. The measurements y_t obtained at a sensor node are bearings or DOAs or the angle the target makes with respect to the sensor node:

$$y_t = \theta_t.$$

These angles are obtained by beamforming the acoustic signal received over a period of T s at the sensor. Measurements θ_t are obtained at every time instant t with a period of T, as shown in Figure 1.3 (a). Using these measurements the target states x, y, v_x, and v_y corresponding to the target x, y coordinates and velocities are estimated at each time instant t. Figure 1.3 compares the observation model in the bearings-only tracker to the batch measurement-based tracker.

In the batch tracker, the target motion model also assumes constant velocity, but the



(a) Bearings-only tracker (b) Batch measurement-based tracker

Figure 1.3: Observation models comparison. (a) Measurements in the bearings-only tracker are DOA's received at time instants T. (b) Measurements in the batch measurement-based tracker are a batch of DOA's. Here, the batch size M is five with two peaks at each instant.

estimated states are the target's motion parameters DOA θ_t, heading direction ϕ_t, and logarithm of velocity over range $q_t = \log v_t/r_t$. The observability of these states using DOA measurements requires a batch of M-DOAs at each time instant t:

$$\mathbf{y}_t = \{y_{t+m\tau}(p_i)\}_{m=0}^{M-1}, \quad p_i = 1, \ldots, P \tag{1.25}$$

with $M \geq 3$, and P is the number of DOA measurement peaks at each time $t + m\tau$, obtained by beamforming the acoustic signal received over a period τ. Given a target's initial condition, the motion parameters are used to compute the target's x-y position as it moves. Figure 1.3 (b) shows the setup in the batch-based multi-target tracker considered in this thesis. In Chapter 2, the individual measurements in the batch are acoustic DOA estimates and in Chapter 3 they are radar range estimates.

The particle filter implementation for the batch-based tracker, compared to the bearings-only tracker, is computationally complex, robust, and handles multiple targets. The complexity and robustness are due to the use of a proposal function that approximates the full-posterior and the nonlinear functions in the state-update and observation models. Table 1.3 shows the main differences between these two trackers and implementations, as considered in this thesis.

With a brief review on particle filters, target tracking, and observation models we have introduced the algorithmic aspects of this thesis. The next section provides an introduction

Table 1.3: Comparison of bearings-only tracker to batch measurement-based tracker.

Characteristic	**Bearings-only tracking**	**Batch-based tracking**
Targets tracked	Single	Multiple
Target motion	Constant velocity	Constant velocity
Target states	Position coordinates (x, y) and velocities (v_x, v_y)	DOA (θ), heading (ϕ), and ratio of velocity over range ($\log v/r$)
Measurement	Single DOAs	Batch of DOAs or ranges
Observation model	Measurements corrupt by Gaussian noise	Measurements corrupted by Poisson clutter or missing measurements
Particle filtering	State update to propose particles Data likelihood to compute particle weights	Full-posterior to propose particles Data likelihood, state likelihood and proposal function to compute weights

to the implementation aspects of this thesis.

1.3 Implementation of Algorithms

Translating an algorithm for real-time implementation requires making specific choices so that the design meets the constraints. Some of the main constraints are speed of execution, power dissipation, accuracy of the results, cost and time involved in the implementation. With the advent of digital integrated circuits (ICs) most algorithms are implemented in the digital domain. While most algorithms are implemented in software executed on a microprocessor, some algorithms and applications require specialized processors or hardware for their implementation. With increasing complexity, the resources consumed by the algorithm also increase. In certain applications more resources lead to increased power dissipation, reduced execution speed, or increased chip area. While there has been significant progress in speed of operation and reduced area, power dissipation still poses a significant challenge [32]. Energy dissipation is critical for certain applications that require a standalone battery-operated sensor or device that needs to perform significant computations, e.g., target tracking in wireless sensor networks.

Table 1.4: Comparison of DSPs to FPGAs.

Characteristics	**DSP**	**FPGA**
Speed	Comparable	Comparable
Power dissipation	Low	High
HW Design flexibility	Low	High
SW Design flexibility	High	Low
Parallelization	Low	High
Ease of implementation	High	Low
Price	Low	High

As mentioned earlier, one goal of this thesis is to address the energy dissipation in the particle filter algorithm by appropriate design choice. The approach we adopt is to use analog circuits in implementing nonlinear functions. We develop a mixed-domain [57], analog and digital domain, approach for a bearings-only target tracker to achieve low-power performance of a particle filter algorithm. More details of this are presented in Chapter 4. Low-power signal processing is an active research area and we provide a brief introduction to this in Section 1.3.1.

In the digital implementation of algorithms, the choices available are to either use a digital signal processor (DSP), a field-programmable gate array (FPGA), or an application-specific IC (ASIC). In this research we focus on DSP and FPGA implementations as these are suitable for the initial design or prototyping of systems. The advantages and disadvantages of using DSPs or FPGAs for signal processing algorithms [9] are shown in Table 1.4. The choice of DSPs or FPGAs for particle filter algorithms is analyzed later in this section.

The tools available for custom digital implementation are sophisticated and flexible compared to the tools available for analog or mixed-signal implementation. We use one such design automation tool, the Xilinx System Generator, to transform Simulink models of the tracker algorithm into hardware description language (HDL) code used in a Xilinx FPGA. While this is not the first FPGA implementation of particle filters, it still provides a platform to compare the implementation generated by the automation tool to an implementation

developed manually by hand. This tool reduces the time spent in system design and hence is suited for complex system designs, such as the batch-based multi-target tracker in this research.

1.3.1 Power-aware design

Power consumed or energy dissipated is an important parameter that needs to be considered while implementing systems for energy-constrained environments. In this section, some basics on power consumption and issues that need to be considered in a power-aware design are discussed. Power-aware design in digital circuits is an active area of research [99]. Power consumption in CMOS digital circuits consists of dynamic and static power. The total power dissipated in a digital circuit has three components, as shown [31]:

$$P = \alpha_{0\to1} C_L V V_{dd} f_{\text{clk}} + I_{sc} V_{dd} + I_{\text{leakage}} V_{dd} \tag{1.26}$$

where $\alpha_{0\to1}$ is the node transition activity factor, C_L the load capacitance, V the swing voltage, V_{dd} the supply voltage, and f_{clk} the clock frequency. In most cases V is the same as V_{dd}. The first term in (1.26) corresponds to switching component power, the second corresponds to direct path short-circuit current I_{sc}, and the third term corresponds to leakage power, which depends on fabrication technology. The switching power is usually higher than the other components. Among several techniques for low-power dissipation in digital circuits, scaling the supply voltage can provide significant savings in power. But, this scaling also places a limit on the maximum speed at which the circuit operates. Hence, an active area of research is to identify optimal regions of operation for low-power operation that provide both the required speed and low-power dissipation [127], [21]. For certain low-power applications where speed is less important, the subthreshold operation of devices is becoming popular. At subthreshold, the leakage power can be significant. Hence reducing this power dissipation in memory devices that are used in subthreshold systems is also an active area of research [20].

The work in [126] provides a good introduction to power consumption in analog circuits and their limits. In analog circuits, the instantaneous power consumed depends upon the (instantaneous) current drawn and the supply voltage V_{dd}. The voltage V_{dd}, however, has

to be above a certain minimum value to provide the necessary dynamic range of operation and to maintain the signal energy level above the thermal noise floor. Reducing voltage to reduce power might lead to increased power dissipation or unstable operation of the circuits and hence is not a valid option. Another approach is to use current-mode operation and operate in the subthreshold region where the current drawn is significantly less. Here, again, care should be taken to maintain the circuit in the valid region of operation with the signal maintained above the noise level. Hence, the low-power operation of analog circuits depends to a large extent on the noise level and the required dynamic range.

1.3.2 Analog computation using MITEs

The digital implementation of linear functions is realized using primitive logic gates such as AND, OR, and NOT gates. These gates use transistors as switches to perform binary operations. Most computations such as addition, subtraction, multiplication, and complex operations are performed using a combination of these logic gates. Nonlinear functions are usually approximated using a combination of these operations, and the required number of devices can be high. On the contrary, the analog implementation of nonlinear functions exploit the relationship between physical quantities such as currents or voltages and the device characteristics of certain devices such as transistors and capacitors. Hence, analog implementation of certain nonlinear functions can be achieved using a few devices [89].

Recent work in the CADSP group at the Georgia Institute of Technology provides an analog approach to implement nonlinear functions using multiple-input translinear elements (MITEs). The MITE was introduced in [88] as a generalization of the bipolar transistor. Both of these circuit elements use their exponential transfer characteristic to implement nonlinear functions or systems [87], [86]. The system parameters are represented as currents and are hence tunable. The optimal number of inputs required in a MITE to implement a function is usually decided by a synthesis procedure. A two-input MITE, whose symbol is shown in Figure 1.4, is defined as a circuit element satisfying the following properties:

1. The current through the input gates is zero.

2. The drain current I and the input-gate voltages V_1 and V_2 are related as

$$I = I_s e^{\frac{\kappa(V_1+V_2)}{U_\mathrm{T}}}, \tag{1.27}$$

where I_s is a pre-exponential scaling constant, κ is a positive dimensionless weight, and U_T is the thermal voltage, kT/q.

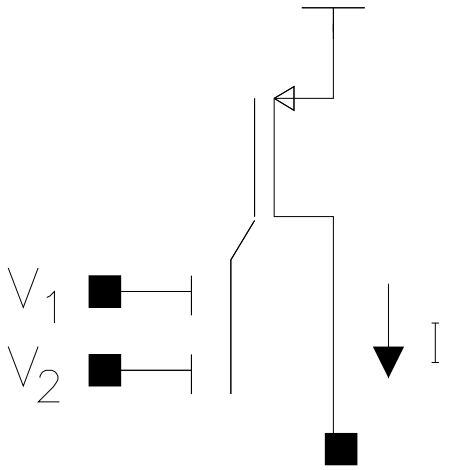


Figure 1.4: Symbol for a two-input MITE.

The synthesis of nonlinear functions using MITEs and their use in particle filter implementation is described in Chapter 4.

1.3.3 Implementation of particle filters

Particle filters are being used in a wide variety of real-time applications. This includes target tracking, navigation [68], robot localization [72], channel estimation [109], and speech modeling [124]. The theoretical aspects, convergence, and performance of particle filter algorithms and applying them to new problems is an active research area. In [16] performance issues and the complexity of particle filters used in bearings-only tracking are studied. The authors suggest adaptively varying the number of particles or variances while tracking. In quantifying the particle filter algorithm's complexity [42] concluded that careful design can mitigate the curse of dimensionality associated with particle filtering. The marginalized particle filter [69] is a combination of the particle filter and the Kalman filter for certain

state-space models where a linear sub-structure exists. This leads to reduced complexity compared to a particle-filter-only approach.

In the past, research on the hardware implementation of particle filters has drawn attention [17], resulting in an FPGA prototype for a particle filter algorithm. As part of that effort the authors developed an architecture for a digital hardware implementation of particle filters along with efficient resampling algorithms. Their initial attempt to use a TI TMS320C54x DSP for a bearings-only tracker application provided a maximum sampling frequency of 1.8 kHz for 1000 particles. By using a Xilinx Virtex II Pro FPGA they achieved a maximum sampling frequency of 50 kHz for a similar application. This later led to an application specific integrated circuit (ASIC) implementation for realizing certain stages in the particle filter algorithm [106], [37], [61]. More recently, a single instruction multiple data (SIMD) architecture that uses N processors to process N particles for particle filters has been presented in [81].

In [109] an FPGA implementation of particle filtering for channel estimation is developed. There, the complexity of particle filters is mitigated through structural modifications such as efficient pipelining and parallel operation. The authors also suggest the use of look-up tables for exponential evaluations and the efficient pipelining of the particle filter stages. The resampling stage in particle filtering is a critical stage for maintaining particle diversity. As resampling requires all particle weights to be available; complete parallelization of the particle filter algorithm is difficult. Efficient methods to address this in hardware can be found in [17], [59], [4].

Data representation is another important issue that needs to be considered while implementing algorithms in hardware. In a digital implementation, the number of bits used to represent quantities of interest is limited. The use of quantization, however, to arrive at finite wordlengths might affect the algorithm's accuracy. Hence, there has been some research on this aspect of the particle filter algorithm. An analysis of finite wordlength effects in a particle filter bearings-only tracker is presented in [15]. In [139], a quantized measurement sequential Monte Carlo (QMSMC) approach for systems with limited communication capability is addressed. The data likelihood function in QMSMC uses quantized

measurements. The work in [139] also compared a quantized measurement extended Kalman filter (QMEKF) and the Kalman filter to QMSMC. Their results show the asymptotic optimality of QMSMC. A more fundamental approach is employed in [68], where the effect of quantization on estimation and filtering is studied, and Cramér-Rao lower bounds (CRLB) are obtained. Here, too, the effect of quantization in the particle filter approach is handled by modifying the data likelihood to use accurate probabilistic models for the quantization noise. Simulations in [68] demonstrate that the particle filter approaches the CRLB when using quantized measurements.

The complexity of models or the large number of particles used can hinder real-time execution of particle filters in certain applications. This happens when the rate of incoming measurements is higher than the update rate of the particle filter. This issue has been addressed by distributing the particles among different observations within an estimation interval. In [73] a weighted mixture of k distributions obtained using k subsets of N/k particles is propagated over time. This approach is used in a particle filter algorithm for robot localization. For large-scale systems, the parallelization of particle filters to achieve real-time constraints has been suggested in [18]. Recent work in [114] uses better particle routing techniques in parallel particle filters to improve execution time. This is applied to likelihood-based estimation of dynamic stochastic general equilibrium (DSGE) models.

Earlier work in the hardware implementation of particle filters address the issue of speed or throughput [17], [4]. Energy efficiency is an important issue in environments that operate under constrained energy or power budgets [62]. The energy dissipated in both communications and computations is critical and needs to be minimized. In Chapter 4, we address the power-aware or energy-efficient implementation of particle filters by adopting a mixed-mode system that performs computations in both the analog and the digital domains.

Most earlier works used FPGAs in particle filter implementation [17], [109]. The particle proposal and weighting stage can be easily parallelized and this can be efficiently implemented in FPGAs. Earlier implementations use the state update as importance function to propose particles and hence the computations in the weight update stage are simple as shown in Section 1.1.3. However, the batch-based algorithm considered in this research,

uses a Gaussian approximation to the full-posterior as the importance function. The use of Newton-Raphson search to identify modes and template-matching to evaluate likelihood, further increases the complexity of the algorithm. We use a DSP and a FPGA to evaluate real-time performance of the batch-based algorithm. The details of this implementation are provided in Chapter 5.

1.4 Contributions and Organization of this Thesis

The contributions of this thesis relate to application of particle filtering for target tracking and hardware implementation of particle filters. From an application perspective, we develop and implement a batch measurement based multi-target tracker that uses acoustic DOAs. We derive and implement the state-space and observation models for a range measurement based multi-target tracker. These two trackers justify use of batch measurements to avoid explicit data-association for multi-target tracking. From an implementation perspective, we consider the batch based tracker and a bearings-only tracker. For low power realization of the bearings-only tracker, we develop mixed-mode implementations that use MITEs for implementing nonlinear functions. The batch-based filter, which is computationally complex compared to the bearings-only tracker, is implemented on a floating-point DSP and most sections of the algorithm are implemented on a FPGA.

In Chapter 2, we present improvements to the batch-based multi-target particle filter DOA tracker introduced by Cevher [28]. We develop a system that uses the particle filter tracker, an acoustic beamformer, and a mode hungry Metropolis-Hastings (MHMH) algorithm for initialization. The particle filter algorithm uses Laplace's method to obtain a Gaussian approximation to the full-posterior. The data-association in multi-target tracking is handled by using a data likelihood similar to the ones used in image template tracking and can handle clutter and missing measurements. We also develop a Laplacian filter and an extended Kalman filter implementation that use batch measurements to perform multi-target tracking. Simulations are presented to demonstrate the performance of particle filter approach in comparison to the other approaches.

The multi-target DOA tracking algorithm's performance can be improved, by developing

a joint tracking system that also uses radar range measurements. Though the joint tracker is not developed in this thesis, a range measurement based tracker is developed in Chapter 3. This uses a batch of radar range estimates and is similar to the DOA measurement based tracker. A new approach to use range-rate information to improve the tracking estimates is also presented. The performance of the algorithm is demonstrated using simulation of various scenarios.

In Chapter 4, we introduce and develop a mixed-mode approach to implement particle filters. In the mixed-mode approach, the nonlinear functions in the bearings-only tracker are implemented using analog MITEs. We demonstrate the tracking performance of the mixed-mode implementation through simulations and compare it to a digital FPGA implementation. The mixed-mode implementation is compared to a digital ASIC implementation based on power dissipation, chip-area, and speed of operation. The digital implementations use the COordinate Rotation DIgital Computer (CORDIC) algorithm to implement the nonlinear functions.

We demonstrate real-time execution of the DOA batch-based particle filter tracking algorithm in Chapter 5. We implement the algorithm on a TI C6713 floating-point DSP, and analyze the required memory-size and the execution times for different stages in the particle filter algorithm. We derive finite wordlengths for desired accuracy and performance, and compare it to a floating-point implementation through simulations. We develop FPGA implementations for certain sections in the particle filter tracker, and present estimates on resources consumed and latencies. We also compare the FPGA implementation of the batch-based tracker to the bearings-only tracker. Using MITEs we implement the DOA update function in the algorithm. We provide tracking results to justify performance of the various implementations.

In Chapter 6, we summarize the contributions of this thesis. We suggest possible extensions to this thesis and areas for future research.

CHAPTER II

ACOUSTIC MULTI-TARGET TRACKING USING DIRECTION-OF-ARRIVAL BATCHES

2.1 Introduction

In this chapter and the next, we develop batch measurement based particle filter multi-target trackers that use direction-of-arrival (DOA)-only or range-only measurements, respectively. The use of batch measurements to alleviate the data association problem inherent in multi-target tracking was introduced by Cevher [28], [22]. The motivation for the separate trackers is to develop a multi-sensor, joint-tracking system that employs both acoustic arrays and radar sensors. This chapter presents a DOA-only tracker and closely follows a related publication [29], which is a joint work by Dr. Volkan Cevher and the author. Later, in Chapter 3 we present details specific to the range-only tracker.

In the literature, target tracking is usually formulated as a state-space problem [7],[46], with an observation equation that relates the target state vector to the measurements, and a state equation that constrains the dynamic nature of the state vector. The state vector contains information related to the target kinematic characteristics, such as DOA, range, velocity, or other motion parameters. The measurements are obtained from acoustic, radar, seismic, or video signals. The performance of the tracking algorithms using state spaces relies heavily on how accurate the models represent the observed natural phenomena [76], [75]. Hence, in many cases, it is important to use nonlinear and non-Gaussian state-space models despite their computational complexity [78].

The multi-target tracking system considered in this chapter uses a batch of DOA measurements to estimate the target states. The DOAs are obtained by beamforming acoustic data received at an acoustic sensor array. Depending on the performance of the beamforming algorithm, the DOAs at certain time instants may be missing or spurious. Such measurements and the maneuvering nature of the targets, leads to nonlinear state-space

models and a non-Gaussian distribution for the noise. In order to better handle this scenario, we use particle filters in the estimation of target states.

As mentioned in Section 1.2, the presence of multiple targets increases the tracking complexity, because data association is needed to sort the received data for each target. In the past, the association problem has been handled in several different ways: (i) probabilistic data association methods estimate the states by summing over all the association hypotheses weighted by the probabilities obtained by the likelihood [7, 6, 34, 92], (ii) smoothness assumptions on the target (motion) states allow a natural ordering of the data [67], (iii) computationally costly ML/EM methods use the likelihood function to search for a global maximum, or (iv) nearest-neighbor methods provide easy heuristics to perform measurement updates. Most of these methods use the mean and covariance approximation of the sufficient statistics for the state, which may be estimated with a Kalman filter; however, for nonlinear state-spaces with general noise assumptions, Monté-Carlo methods or particle filters should be used to adequately capture the dynamic, possibly multi-modal, statistics.

In a particle filter, where the observations arrive in sequence, the state probability density function is represented by discrete state samples (particles) distributed according to the underlying distribution either directly or by proper weighting [78, 46]. Hence, the filter can approximate any statistics of the distribution arbitrarily accurately by increasing the number of particles with proven convergence results [40]. In the particle filtering framework, data association is undertaken implicitly by the state-space model interaction. However, the particle filter suffers from the curse of dimensionality as the number of targets increases [40]. To increase the algorithm's efficiency, various methods have been proposed, such as the partitioning approach [79, 97], or other Bayesian approaches [64].

More recently [125] uses particle filters to address data association and multi-target tracking in the presence of clutter. The authors use a proposal distribution that captures the notion of a soft-gating of the measurements[1]. They present a Monte Carlo joint probabilistic data association filter (MC-JPDAF) that uses particle filters to approximate the

[1]Gating is an approach used in joint probabilistic data association (JPDA) approaches to reduce the number of hypotheses in associating the measurements to targets

distributions in a JPDA filter. They also present two improvements to the standard particle filtering algorithm. In the first approach called sequential sampling particle filter (SSPF), they use a proposal function that sequentially updates the association hypotheses. The second approach uses an independent partition particle filter (IPPF) for the association hypotheses, assuming that the associations are independent over the individual targets. They track slowly maneuvering targets in the x-y plane, using both range and bearing or DOA measurements.

In our work we consider maneuvering targets and use batch measurements to alleviate data association. Further, we do not perform any explicit data association or gating of measurements. The tracking is performed using target states expressed in polar coordinates and the measurements are a batch of DOAs. Though it is based on DOAs the tracking algorithm also estimates the target heading and ratio of velocity-to-range. These states are then used to estimate the target position in the x-y plane. The use of batch measurements to estimate target states is described in Section 2.1.1.

We begin this chapter with a generic description of the batch measurement model. Then, we concentrate on a target tracking system that uses DOAs as measurements. The tracking system consists of three stages. The first stage is a beamforming block that provides the DOA measurements. The second stage is a particle filter based tracking algorithm that uses the DOA measurements to estimate target states. The third stage uses a mode hungry Metropolis-Hastings algorithm to estimate the number of targets and performs automatic initialization. These three stages are iteratively executed to track multiple targets over time.

We also present simulation results comparing the performance of the particle filter tracker to a Laplacian filter and an extended Kalman filter (EKF). The Laplacian filter is an approximate Bayesian filter that uses Laplace's method to obtain the posterior distribution. The EKF is derived using a sliding window implementation to handle the complex observation model that considers DOA batches with clutter and missing DOAs.

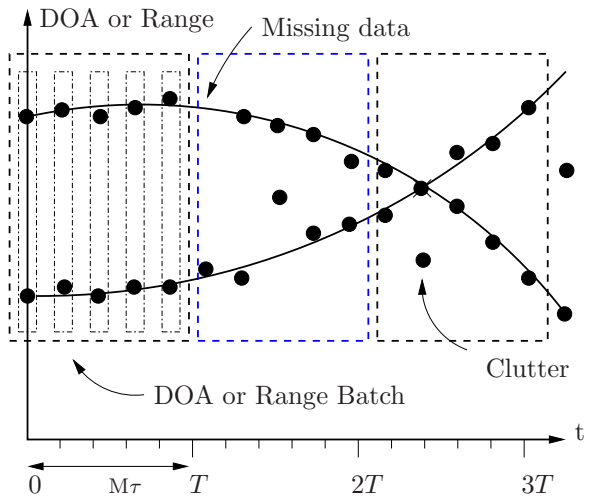


Figure 2.1: Observation model uses a batch of DOA (Chapter 2) or range (Chapter 3) measurements. The measurements are not necessarily ordered. However, when the batch is treated as a measurement vs. time image, there is a natural ordering, which aids in the multiple target tracking by the particle filter. This figure is reproduced from [29].

2.1.1 Batch measurements for multi-target tracking

The particle filter developed here (and in Chapter 3) uses multiple measurements to determine the state vector, based on an image template-matching idea. We denote the collection of M measurements a batch, where M is the batch size. In our problem, a temporal DOA image is first formed when a batch of DOA observations is received from a beamformer that processes the received acoustic data at M τ-second intervals (Figure 2.1). Then, image templates for target tracks are created using the state update function and the target state vectors (Figure 2.2). By determining the best matching template (e.g., most probable target track), the target state vectors are estimated. Because the observations are treated as an image, the data association and DOA ordering problems are naturally alleviated. Moreover, by assuming that the observations are approximately normally distributed around the true target tracks, with constant miss-probability and clutter density, a robust particle filter tracker is formulated. In Chapter 3, we apply a similar image template-matching approach to track multiple targets using radar range measurements processed at a radar sensor.

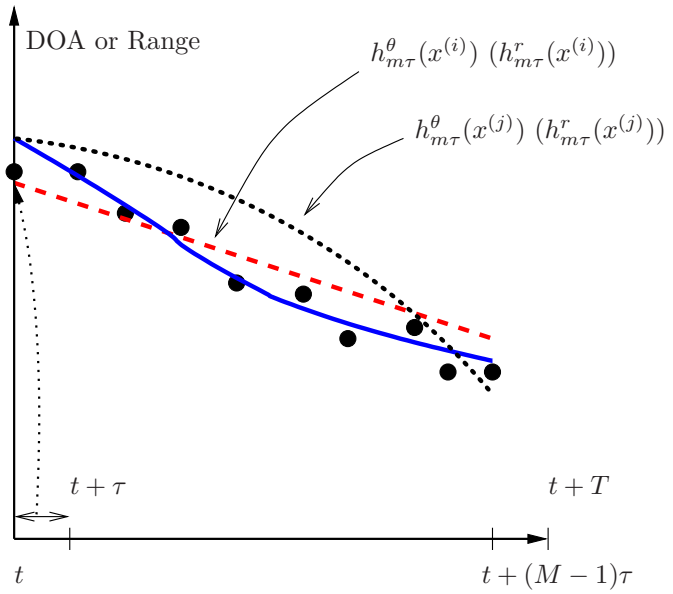


Figure 2.2: Template matching idea is illustrated. The solid line represents the true DOA (range) track. Black dots represent the noisy DOA (range) estimates. The dashed and dotted lines represent the DOA (range) tracks for two proposed particles, $x^{(i)}$ and $x^{(j)}$. These tracks are calculated using the state update function h. Visually, the i-th particle is a better match than the j-th particle; hence, its likelihood is higher. This figure is reproduced from [29].

2.1.2 Acoustic DOA tracker

The target tracker in this chapter uses a particle filter algorithm to track the DOAs of multiple maneuvering targets. Each particle in the filter is created by concatenating partitions, i.e., the state vector for each target. For example, the partition of the k-th target has a state vector that consists of the DOA $\theta_k(t)$, the heading direction $\phi_k(t)$, and the logarithm of velocity over range $Q_k(t) = \log(v_k/r_k(t))$ of the k-th target. The total number of targets (or partitions) K is determined by a mode hungry Metropolis-Hastings block. Hence, given K targets, a particle has K partitions where each target, and each partition, is assumed to be independent.

The particle filter importance function proposes particles for each target independently to increase the efficiency of the algorithm. To derive the proposal function for each target, we use Laplace's method to approximate the posterior of the corresponding partition by a Gaussian around its mode [122, 53, 46]. We calculate the partition modes using a robust Newton-Raphson search method that imposes smoothness constraints on the target motion.

We also present a slower but more robust method for determining the partition posteriors, based on the mode hungry Metropolis-Hastings algorithm [23]. Details of the partition posteriors are given in Section 2.4.

2.2 Tracker System Design

The target tracking system is constructed such that it (i) compensates DOA estimation biases due to rapid target motion [138], (ii) results in higher resolution DOA estimates than just beamforming [138, 97, 25], (iii) is robust to changes in target signal characteristics, and (iv) automatically determines the number of targets. The tracking system consists of three blocks as illustrated in Figure 2.3. Details of the individual blocks are given in the following sections. Here, we discuss the technicalities that lead to this design.

The main objective of the system is to report multiple target DOAs at some period T, after observing the acoustic data at the node microphones. Quite often, target DOAs can change more than a few degrees during an estimation period, e.g., due to rapid target motion. Hence, if we were to just use conventional snapshot DOA estimation methods (e.g., MUSIC, MVDR, etc.) for tracking the targets, the bearing estimates become biased, because the received data is not stationary [66]. This is intuitive, because these methods estimate an average of the target angular spread during their estimation periods [138].

In general, locally linear motion models eliminate this bias by simultaneously estimating the bearing and the target motion parameters for the estimation period T. Conceptually, this is equivalent to aligning the received acoustic data with the motion parameters so that the data becomes stationary for bearing estimation purposes. But, this alignment process relies heavily on the observation model and is computationally costly because of the high volume of the acoustic data used for estimation. In this research, we propose to use a beamformer block to buffer the variability in the observed acoustic signals and to create a compressed set of invariant statistics for our particle filter block.

Therefore, the beamformer block (Figure 2.3) processes the acoustic data at shorter time intervals of $\tau = T/M$ (e.g., $M = 10$), where the targets are assumed to be relatively stationary. This reduces the number of acoustic data samples available for beamforming,

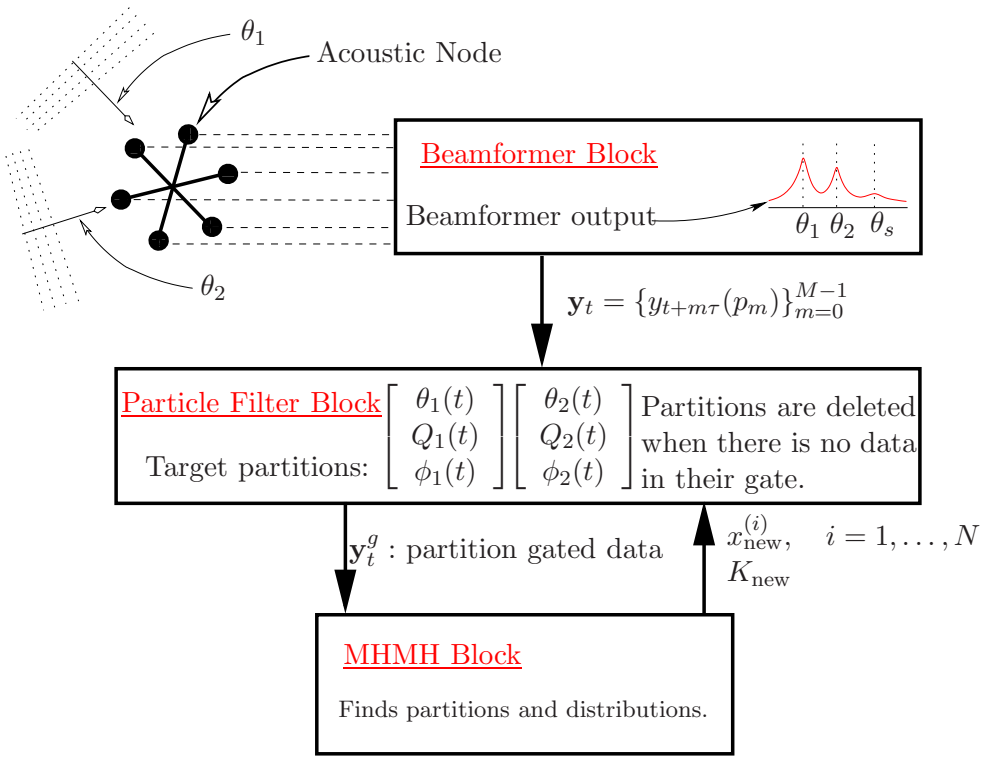


Figure 2.3: Tracking system mechanics demonstrated. The beamformer block monitors the received acoustic signals to adapt to their frequency characteristics for better bearing estimation. It outputs a batch of DOAs that form a sufficient statistic for the particle filter block. The particle filter estimates the tracking posterior and can make various inferences (e.g., mean and mode estimates). It acquires the new partition information from the mode hungry Metropolis Hastings (MHMH) block. It can also delete a partition, if there is no data in the respective partition gate (explained in the text). The MHMH block determines the new partitions and their distributions from the residual DOAs coming from the particle filter (i.e., the gating operation). This figure is reproduced from [29].

resulting in a sequence of noisier DOA estimates. However, these noisier DOA estimates can be smoothed in the particle filter block, because a motion structure is imposed on the batch of DOAs. Moreover, for the state vector defined in Section 2.1.2, a batch of more than three DOAs is sufficient for observability of the state [138]. Since the filter is built on the compressed statistics which are also sufficient to observe the state vector, it achieves a significant reduction in computation when compared to methods that use the acoustic data directly [25].

When the received signal characteristics change, the particle filter tracker formulation is not affected because the beamformer block can absorb variabilities in the acoustic signals. The state-space formulations in this research is similar to other trackers proposed in the literature [138, 97, 25]. The trackers presented in [138, 97] directly employ the classical narrow-band observation model, where targets exhibit constant narrow-band frequency characteristics [66]. The tracker in [25] tries to adapt to varying time-frequency characteristics of the target signals, assuming that the varying frequencies are narrow-band. The tracker in [74] also incorporates an amplitude model for the signals. Because their probability density equations explicitly use an observation equation that relies on the raw measurements, these trackers are hardwired to their observation model. Hence, a complete re-work of the filter equations would be required to track targets with wideband signal characteristics (e.g., [74], [93]).

The third block in Figure 2.3 addresses a fundamental issue for trackers: initialization. It is an accelerated Metropolis-Hastings algorithm modified specifically for unimodal distributions. It takes the ungated DOAs from the particle filter as inputs (Figure 2.4), or if there are no partitions in the particle filter, it takes the DOAs as data from the beamformer block directly. It generates a particle distribution for each potential new target *one at a time*, i.e., it converges on the strongest mode in the, possibly, multi-modal target posterior and sifts out the corresponding DOA data. It then iterates to find other modes until stopping criteria are met. This block creates new partitions for the particle filter, which can also delete its own partitions when conditions described in Section 2.5 are met. Conceptually, this initialization idea is equivalent to the *track-before-detect* (TBD) approach used in the

radar community [12, 105]. Briefly, TBD implies that the tracking or estimation process uses the measurements obtained over a time to estimate states without performing an explicit detection. An implicit detection or estimate of the number of targets is performed as part of the tracking process.

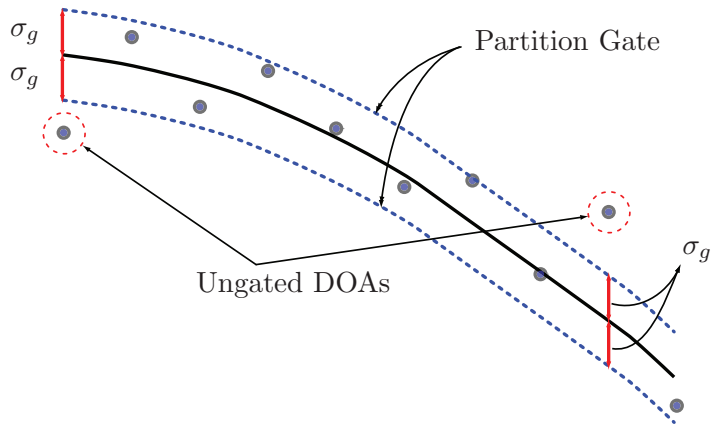


Figure 2.4: Gating operation illustrated. Solid line is the true DOA track. The DOA observations are shown with dots. Given a partition gate size σ_g, the DOAs are ignored (i.e., gated out) if they are more than σ_g away from the solid line.

2.3 Beamformer Block

Beamformers use the collected acoustic data to determine target DOAs and are called narrow-band beamformers if they use the classical narrow-band array observation model [66, 113]. Beamformers are wideband if they are designed for target signals with broadband frequency characteristics [128].

The beamformer block can be designed with the best beamformer for the local characteristics of the acoustic signals. That is, given the observed acoustic signal and its time-frequency distribution, we can choose an optimal beamformer to calculate target DOAs. For example, we can choose multiple beamformers if the received acoustic signal shows both narrow-band and wideband characteristics. The output of the beamformer $\mathbf{y}_t = \{y_{t+m\tau}(p)\}_{m=0}^{M-1}$ is a DOA data cube containing the P_m-highest DOA peaks of the beamformer pattern. In general, the number of DOA peaks P_m at batch index m should be greater than or equal to the number of targets K so that we can also detect new targets while tracking the existing targets. In the discussions presented here, we fix $P_m = P$.

Hence, the output of the beamformer is a DOA data cube $\mathbf{y}_t = \{y_{t+m\tau}(p)\}_{m=0}^{M-1}$. In Figure 2.1, $P = 2$ and $M = 5$. The related work in [29] considers dependence on frequency in the beamformer. Note that the input of the particle filter has the same structure regardless of the target signal characteristics.

Finally, the choice of the parameter τ for beamforming is determined by various physical constraints, including (i) target frequency spread, (ii) target speed, and (iii) a target's affinity to maneuver. For reasonable beamforming, at least two cycles of the narrow-band target signals must be observed. This stipulates that $\tau > 2/F_{\min}$, where $F_{\min}$ is the minimum beamforming frequency for the target. Moreover, to keep the worst case beamforming bias[2] bounded for each DOA by an angle threshold denoted by D, we approximately have $\tau < \frac{2D}{\exp\{Q\}}$, where $\exp\{Q\}$ is the target velocity over range ratio. Lastly, the target motion should satisfy the constant velocity assumption during the output period T. For slow moving ground targets, $T = 1$ s is a reasonable choice. Note that at least three DOA estimates are necessary to determine the state vector. To improve the robustness of the tracker, we use $M > 5$ to decrease the probability that the state is not observable due to missing DOAs. Hence, the parameter τ is bounded by the following

$$\frac{2}{F_{\min}} < \tau < \min\left\{\frac{2D}{\exp\{Q\}}, \frac{T}{M}\right\}. \tag{2.1}$$

2.4 Particle Filter

The details of the particle filter block in Figure 2.3 are discussed here. In Section 2.6, we provide alternative algorithms to replace this block based on the Kalman filter and Laplace's method.

2.4.1 State update model

The particle filter state vector $\mathbf{x}_t$ consists of the concatenation of partitions $x_k(t)$ for each target, indexed by k, and represented as $\mathbf{x}_t = \left[\, x_1^T(t),\ x_2^T(t),\ \cdots\ , x_K^T(t)\, \right]^T$, where K is

[2]The bias is calculated by taking the angular average of the target track. Hence, this bias also depends on the heading direction. The worst case bias happens when the target heading and DOA sum up to π. Moreover, it is also possible to analytically find an expected bias by assuming uniform heading direction, using a similar analysis done in [138].

the number[3] of targets at time t. Each partition has the corresponding target motion parameters $x_k(t) \triangleq [\ \theta_k(t)\ ,\ Q_k(t)\ ,\ \phi_k(t)\ ]^T$, as defined in Section 2.1.2. The angle parameters $\theta_k(t)$ and $\phi_k(t)$ are measured counterclockwise with respect to the x-axis.

The state update equation can be derived from the geometry imposed by the locally constant velocity model, and is the same as in [22]. The resulting state update equation is nonlinear:

$$x_k(t+T) = h_T\left(x_k(t)\right) + u_k(t), \tag{2.2}$$

where $u_k(t) \sim \mathcal{N}\left(0, \Sigma_u\right)$ with $\Sigma_u = \mathrm{diag}\{\sigma^2_{\theta,k}, \sigma^2_{Q,k}, \sigma^2_{\phi,k}\}$ and

$$h_T(x_k(t)) = \begin{bmatrix} \tan^{-1}\left\{\frac{\sin\theta_k(t)+T\exp Q_k(t)\sin\phi_k(t)}{\cos\theta_k(t)+T\exp Q_k(t)\cos\phi_k(t)}\right\} \\ Q_k(t) - \frac{1}{2}\log\left\{1 + 2T\exp\left(Q_k(t)\right)\cos(\theta_k(t)-\phi_k(t)) + T^2\exp(2Q_k(t))\right\} \\ \phi_k(t) \end{bmatrix}. \tag{2.3}$$

The analytical derivations of (2.3) can be found in [138, 25], and [25] also discusses state update equations based on a constant acceleration assumption.

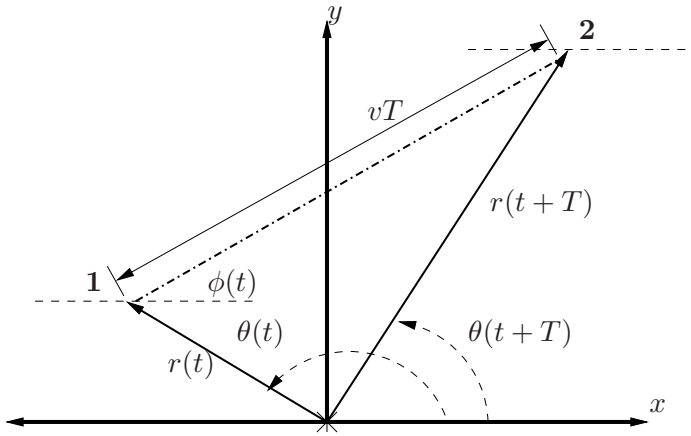


Figure 2.5: Geometry for constant velocity target motion. $\theta(t)$ denotes target DOA, $r(t)$ denotes target range, and $\phi(t)$ denotes the target heading direction when the target is at position denoted by **1** in the x-y plane. The target velocity is represented by v and the period over which the velocity is assumed constant is T. (This figure is same as Figure 1.2).

[3]Explicit time dependence is not shown in the formulations. The parameter K is determined by the MHMH block (Section 2.5).

2.4.2 Observation model

The observation $\mathbf{y}_t$ at time t, consists of the batch DOA estimates from the beamformer block (Section 2.3), given by

$$\mathbf{y}_t = \{y_{t+m\tau}(p)\}_{m=0}^{M-1}, \quad p = 1, \ldots, P \tag{2.4}$$

where m is the batch index. Hence, the acoustic data of length T is segmented into M segments of length τ. Multiple DOA measurements imply the presence of clutter or multiple targets.

The particle filter observation model also includes a clutter model because beamformers can produce spurious DOA peaks as output (e.g., the sidelobes in the power vs. angle patterns) [66]. To derive the clutter model, we assume that the spurious DOA peaks are random with uniform *spatial* distribution on the angle space, and are temporally as well as spatially independent. In this case, the probability distribution for the number of spurious peaks is best approximated by the Poisson distribution with a spatial density [7], [98]. Moreover, the probability density function (pdf) of the spurious peaks is the uniform distribution on $[0, 2\pi)$. However, since the number of peaks in the beamformer output can be user defined (P), and since the beamformer power vs. angle pattern has smoothness properties, we use the following pdf for the spurious peaks:

$$p(\theta|\theta \text{ is spurious}) = \frac{\gamma}{2\pi}, \tag{2.5}$$

where $\gamma \gg 1$ is a constant that depends on the maximum number of beamformer peaks P, the beamformer itself (i.e., the smoothness of the beamformer's steered response), and the number of targets K. Equation (2.5) implies that the natural space of the clutter is reduced by a factor of γ because of the characteristics of our specific system.

2.4.3 Data likelihood function

We now present the data likelihood function for the multi-target tracking scenario considered here. This is similar to the model used in visual tracking [65]. The approach used here was introduced in [28], [22]. This is also the approach used in the implementation of the

batch tracker in Chapter 5. It assumes that the batch of DOAs form a normally distributed cloud around the true target DOA tracks (Figure 2.1) with a constant miss probability of κ; and may have spurious peaks Poisson distributed as derived in (2.5) in Section 2.4.2. Of the P peaks, only one peak corresponds to a target and the remaining peaks either correspond to other targets or clutter. Hence, for a specific time instant within a batch period, with $m = m_i$, we have the observation density to be

$$p(\mathbf{y}_{m_i}(t)|\mathbf{x}_k(t)) \propto 1 + \frac{1-\kappa}{\sqrt{2\pi\sigma_\theta^2(m_i)\kappa\lambda}} \sum_{p_i=1}^{P} \exp\left\{ -\frac{\left(h^\theta_{m_i\tau}\left(x_k(t)\right) - y_{t+m_i\tau}(p_i)\right)^2}{2\sigma_\theta^2(m_i)} \right\}. \quad (2.6)$$

Extending this to a batch of M observations for a single target, we have

$$p(\mathbf{y}(t)|\mathbf{x}_k(t)) \propto \prod_m p(\mathbf{y}_m(t)|\mathbf{x}_k(t)). \quad (2.7)$$

Hence, the observation density for multiple targets can be written as

$$p(\mathbf{y}(t)|\mathbf{x}(t)) \propto \prod_k \prod_m \left\{ 1 + \frac{1-\kappa}{\sqrt{2\pi\sigma_\theta^2(m)\kappa\lambda}} \sum_{p_i=1}^{P} \exp\left\{ -\frac{\left(h^\theta_{m\tau}\left(x_k(t)\right) - y_{t+m\tau}(p_i)\right)^2}{2\sigma_\theta^2(m)} \right\} \right\}, \quad (2.8)$$

where κ is a constant miss probability, $\lambda = \dfrac{\gamma}{2\pi}$ is the Poisson rate of the clutter distribution, superscript θ on the state update function h refers to the DOA component of the state update (2.3), and $\sigma_\theta^2(m)$ is supplied by a beamformer.

A more rigorous and robust approach to derive the data likelihood using the joint probabilistic data association (JPDA) [7, 8] is presented in [29]. Though, there is no significant improvement in performance, the JPDA based approach provides additional parameters that can be used to better model target interactions.

Note that the DOA distribution in (2.8) is not a proper circular normal distribution for an angle space. For angle spaces, the von Mises distribution is used as a natural distribution [50]. The von Mises distribution has a concentration parameter with a corresponding circular variance. It can be shown that for $\sigma_\theta^2 \ll 1$ (high concentration), the von Mises distribution tends to the Gaussian distribution in (2.8) [49]. Because the von Mises distribution has numerical issues for small DOA variances, the Gaussian approximation (2.8) is used in this research. Hence, special care is taken in the implementation to handle angle wrapping issues.

2.4.4 Particle filter proposal function

In our problem of DOA-only multiple target tracking, the proposal function poses difficult challenges because (i) the state vector dimension is proportional to the number of targets K, hence the number of particles to represent posterior can increase significantly as K increases (the curse of dimensionality), (ii) in many cases, the targets maneuver, hence full posterior approximations are required for robust tracking, and (iii) for full posterior approximations, robustly determining the DOA-only data-likelihood is rather hard. We will address each of these challenges in this section. Note that once the proposal function is formulated, the rest of the particle filter structure is well-defined: weighting and resampling.

2.4.4.1 Partitioned sampling

A partitioned sampling approach is used to reduce the curse of dimensionality in the particle filter. The basic idea is as follows. Suppose that we (wrongfully) factor the tracking posterior density as

$$\begin{aligned} p(\mathbf{x}_t|\mathbf{y}_t, \mathbf{x}_{t-T}) &\propto p(\mathbf{y}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{x}_{t-T}) \\ &= \prod_{k=1}^{K} p(\mathbf{y}_t|x_k(t)) \prod_{k=1}^{K} p(x_k(t)|x_k(t-T)) \\ &= \prod_{k=1}^{K} p(\mathbf{y}_t|x_k(t))p(x_k(t)|x_k(t-T)) \\ &\propto \prod_{k=1}^{K} q_k(x_k(t)|\mathbf{y}_t, x_k(t-T)). \end{aligned} \tag{2.9}$$

In this case, the target posterior is conveniently a product of the partition posteriors (i.e., individual target posteriors) $q_k(\cdot|\cdot)$. We can then generate samples for each partition according to its posterior (i.e., $x_k^{(i)} \sim q_k(x_k(t)|\mathbf{y}_t, x_k(t-T))$) and merge them to represent $\mathbf{x}_t$. It can be proved [79] that the resulting particle distribution is the same as when we generate $\mathbf{x}_t$ directly from the full posterior $p(\mathbf{x}_t|\mathbf{y}_t, \mathbf{x}_{t-T})$. However, in the partitioned sampling case, the computational complexity of the state vector generation is linear with respect to the number of targets K as opposed to exponential when the state vector is sampled from the full posterior.

In our problem, we can approximately factor out the tracking posterior to exploit the computational advantage of the partitioned sampling. In addition the target dynamics can be factored out because we assume the targets are moving independently.[4] Unfortunately, the observation density does not factor out, because the observed DOA data cannot be immediately associated with any of the partitions. However, for a given partition, if we assume that the data is only due to that partition and clutter (hence, the DOA data corresponding to other partitions are treated as clutter), we can do the following approximate factorization on the observation likelihood (2.8):

$$\begin{aligned} p(\mathbf{y}_t|\mathbf{x}_t) &\approx \prod_{k=1}^{K} p(\mathbf{y}_t|x_k(t)) \\ &= \prod_{k=1}^{K}\prod_{m=0}^{M-1}\left\{1+\frac{1-\kappa}{\sqrt{2\pi\sigma_\theta^2(m)}\kappa\lambda}\sum_{p_i=1}^{P}\exp\left\{-\frac{\left(h_{m\tau}^{\theta}\left(x_k(t)\right)-y_{t+m\tau}\left(p_i\right)\right)^2}{2\sigma_\theta^2(m)}\right\}\right\}. \end{aligned} \tag{2.10}$$

Hence, for our problem, an approximate partition posterior is the following

$$q_k(x_k(t)|\mathbf{y}_t, x_k(t-T)) \propto p(\mathbf{y}_t|x_k(t))p(x_k(t)|x_k(t-T)), \tag{2.11}$$

where $p(\mathbf{y}_t|x_k(t))$ is given in (2.10) and $p(x_k(t)|x_k(t-T)) = \mathcal{N}\left(h_T(x_k(t-T)), \Sigma_u\right)$ with $\Sigma_u = \text{diag}\{\sigma_\theta^2, \sigma_Q^2, \sigma_\phi^2\}$.

2.4.4.2 A Gaussian approximation for partition posteriors

Now, we derive a Gaussian approximation to (2.11) to be used as a proposal function for the particle filter. Hence, we use the current observed data to propose the filter's particle support, also incorporating target maneuvers if there are any. This approximation can be done in two ways: (i) Laplace's method or (ii) the mode hungry Metropolis-Hastings method. The first method is an order of magnitude faster than the second, but it is not as robust. The filter monitors the target states and can decide to switch between methods to find the partition posteriors.

[4] When targets are moving closely in tandem, there is a possibility that the beamformer block may not resolve them. Hence, they can be treated as a single target. In other cases, the independence assumption still works. However, if high resolution observations (e.g, top down video images of the target plane) are available, it is better to also model the interactions of targets. A good example using Monté-Carlo Markov chain methods can be found in [70].

By default, the filter uses Laplace's method to approximate $p(\mathbf{y}_t|x_k(t))$ in (2.11) and thereby derive the partition proposal functions of the particle filter, denoted as $g_k(x_k(t)|\mathbf{y}_t, x_k(t-T))$. Laplace's method provides an analytical approximation for probability density functions, based on a Gaussian approximation of the density around its mode, where the inverse Hessian of the logarithm of the density is used as a covariance approximation [53]. It can provide adequate approximations to posteriors that are sometimes more accurate than approximations based on third-order expansions of the density functions [122]. The computational advantage of this approach is rather attractive because it only requires first- and second-order derivatives. The condition for accurate approximation is that the posterior be an unimodal density or be dominated by a single mode. Hence, it is appropriate for approximating the partition posteriors of the particle filter.

Laplace's approximation requires the calculation of the data statistics. The Laplacian approximation is described in detail in [22]. For this research, it is implemented with the Newton-Raphson recursion with backtracking for computational efficiency, as explained in Appendix A. Because of the constrained nature of the algorithm's modified cost function, this method is sometimes susceptible to "shadow tracking" and can diverge under certain conditions (refer to Figure 2.6). Shadow tracking refers to the scaled target tracks in x-y that can lead to the same DOA track as illustrated in Figure 2.6. Shadow tracks may cause the Laplacian approximation deviate from the truth because of the motion smoothness constraints used to calculate it.

Any divergence in the Newton-Raphson mode calculation can be detected by monitoring the number of DOA measurements that fall into the gate of the mode estimate (Figure 2.4). If there are less than a threshold number of DOAs (typically M/2) within a σ_g neighborhood of the Newton-Raphson mode's DOA track, then the particle filter uses the MHMH method for determining the mode. After the MHMH iterations are over, if the MHMH corrected mode fails to have the required number of DOAs within its gate, the partition is declared dead. Details of the MHMH method are given in Appendix A. The final expression for the partition proposal functions to be used in the particle filter is given by

$$g_k(x_k(t)|\mathbf{y}_t, x_k(t-T)) \sim \mathcal{N}\left(\mu_g(k), \Sigma_g(k)\right) \tag{2.12}$$

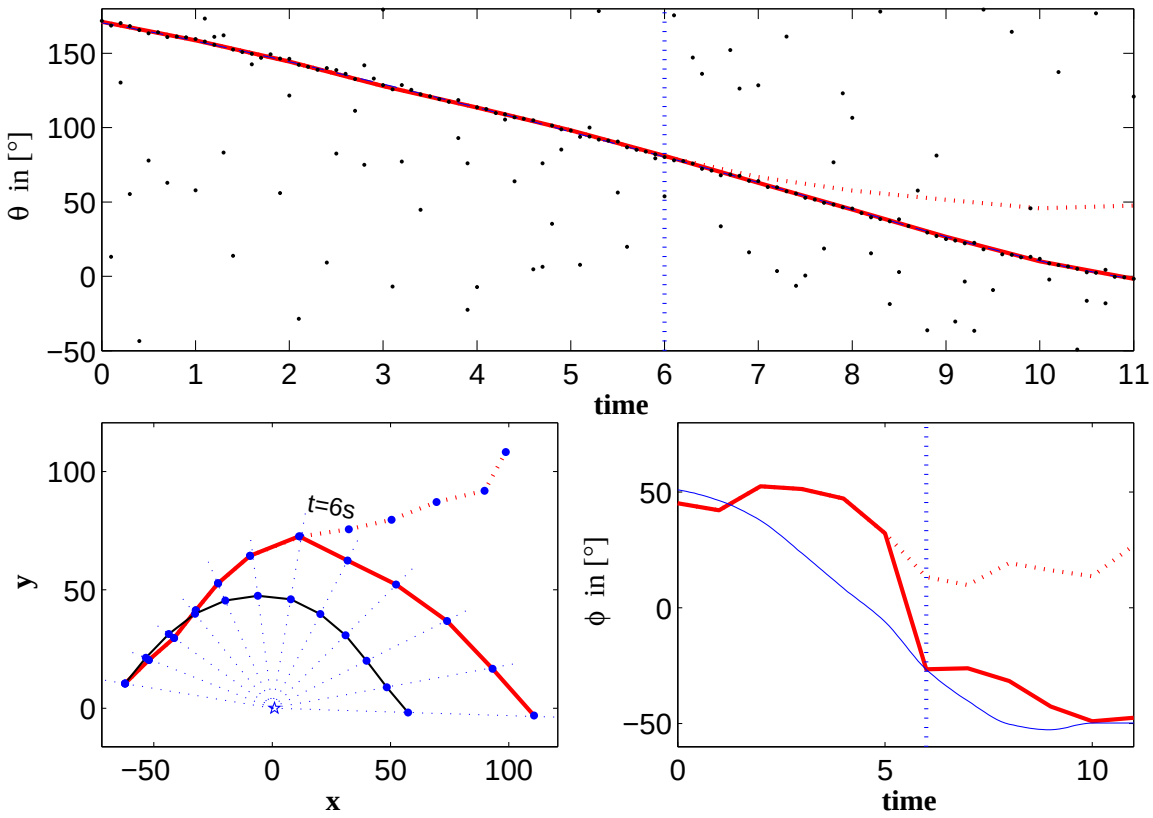


Figure 2.6: The particle filter is susceptible to shadow tracking, when only the Newton-Raphson recursion is used to determine the data-likelihood function. (*Top*) DOA tracking results with MHMH correction (solid line) and without (dotted line). At $t = 6$s, the MHMH automatically corrects the heading bias. (*Bottom Left*) True track (inside), the shadow track (middle) and the diverged track (outside, dotted line) which lacks the MHMH correction. (*Bottom Right*) Filter heading estimates. Note the MHMH correction at time $t = 6$s.

where the Gaussian density parameters are

$$\begin{aligned}\Sigma_g(k) &= \left(\Sigma_y^{-1}(k) + \Sigma_u^{-1}\right)^{-1} \\ \mu_g(k) &= \Sigma_g(k)\left(\Sigma_y^{-1}(k)x_{k,\text{mode}} + \Sigma_u^{-1}h_T(x_k(t-T))\right),\end{aligned} \tag{2.13}$$

where $x_{k,\text{mode}}$ is the mode of $p(\mathbf{y}_t|x_k(t))$, and $\Sigma_y^{-1}(k)$ is the Hessian of $p(\mathbf{y}_t|x_k(t))$ at $x_{k,\text{mode}}$, calculated by either one of the methods in this section. Using this in (2.11) yields (2.13).

2.4.5 Algorithm details

Pseudo-code of the particle filter algorithm is given in Table 2.1. The filter implementation employs an efficient resampling strategy, called "deterministic resampling," first outlined in [71]. This resampling strategy is preferred because of (i) the efficient sorting of the particles and (ii) the small number of random number generations. The deterministic resampling strategy also has known convergence properties [71]. Faster resampling schemes without convergence proofs are also available [13] and these could make a difference in the filter computation, especially when $K = 1$.

Finally, the partitions are managed by the specific interaction between the particle filter and the MHMH block. New partitions are introduced into the particle filter, using the distribution supplied by the MHMH block. First, the particle filter block deletes the DOAs from the observed data that belong to the partitions that it is currently tracking (gating) after estimation (Figure 2.4). Gating here is a simple distance thresholding operation and does not incur significant computation. Then, the remaining DOAs are used by the MHMH block to determine any new partition distributions. For the particle filter to stop tracking a given target, a deletion operation is necessary. The particle filter can delete partitions at either the proposal stage or after estimation. Lastly, note that our implementation of the particle filter does not make partition associations such as partition *split* or *merge*. We leave these decisions to a higher level fusion algorithm.

Table 2.1: Acoustic DOA particle filter tracker pseudo-code.

Given the observed data $\mathbf{y}_t = \{y_{t+m\tau}(p)\}_{m=0}^{M-1}$ in $[t, t+T)$, do

1. For $i = 1, 2, \ldots, N$

   - For $k = 1, 2, \ldots, K$
     sample $x_k^{(i)}(t) \sim g_k(x_k^{(i)}(t)|\mathbf{y}_t, x_k^{(i)}(t-T))$, given by (2.12).
   - Form $\mathbf{x}_t^{(i)} = \left[\, x_1^{(i)}(t),\ x_2^{(i)}(t), \ldots,\ x_K^{(i)}(t) \right]^T$.

2. Calculate the weights

$$w^{*}{}_t^{(i)} = w_{t-T}^{(i)} \frac{p(\mathbf{y}_t|\mathbf{x}_t^{(i)}) p(\mathbf{x}_t^{(i)}|\mathbf{x}_{t-T})}{\prod_k g_k(x_k^{(i)}(t)|\mathbf{y}_t, x_k^{(i)}(t-T))}, \tag{2.14}$$

   where $p(\mathbf{y}_t|\mathbf{x}_t^{(i)})$ is fully joint observation density, given by (2.8).

3. Normalize the weights: $w_t^{(i)} = \dfrac{w^{*}{}_t^{(i)}}{\sum_i w^{*}{}_t^{(i)}}$.

4. Form the estimate: $E\{f(\mathbf{x}_t)\} = \sum_{i=1}^{N} w_t^{(i)} f(\mathbf{x}_t^{(i)})$.

5. Resample the particles:

   - Heapsort the particles in an ascending order according to their weights: $\mathbf{x}_t^{(i)} \rightarrow \tilde{\mathbf{x}}_t^{(i)}$.
   - Generate $\omega \sim \mathcal{U}[0, 1)$.
   - For $j = 1, 2, \ldots, N$
     a. $u^{(j)} = \frac{j-\omega}{N}$,
     b. Find i, satisfying $\sum_{l=1}^{i-1} \tilde{w}_t^{(i)} < u^{(j)} \leq \sum_{l=1}^{i} \tilde{w}_t^{(i)}$,
     c. Set $\mathbf{x}_t^{(j)} = \tilde{\mathbf{x}}_t^{(i)}$.

6. Associate the DOA estimates with partitions using a nearest neighbor approach.

   - Calculate the DOA track for each partition of $\mathbf{x}$, $\mathbf{x} = \sum_{i=1}^{N} w_t^{(i)} \mathbf{x}_t^{(i)}$, using the state update equation.
   - Associate the nearest DOA estimates within a threshold angle distance, with each partition.
   - Report all the unassociated DOA estimates to the MHMH block.

7. Delete partitions with the number of associated DOA estimates less than a threshold number (e.g., M/2).

8. Randomly merge the new partitions coming from the MHMH block, if there are any.

2.5 The MHMH Block

The objective of the MHMH block[5] is to find any new targets in the observed data and determine their probability density distributions. The MHMH block uses the matching pursuit idea by Mallat [80] by consecutively applying the MHMH algorithm on the ungated DOAs until the stopping conditions described below are satisfied. Each iteration of the MHMH block results in a new target state vector distribution and the total number of iterations gives the number of new targets.

The MHMH block employs the mode hungry Metropolis-Hastings sampling algorithm to determine the distribution for the target that has the highest mode in the multi-target observation density. The MHMH sampling algorithm [23] is an accelerated version of the Metropolis-Hastings algorithm [84, 58, 36, 121] that generates samples around the modes of a target density. The pseudo-code for the MHMH sampling algorithm is given in Table 2.2 (reproduced from [23]). For our system, we use a random walk for the candidate generating function $q(\cdot)$ in Table 2.2 with the walk noise variances set to $\sigma_\theta = 0.5^\circ$, $\sigma_Q = 0.01\text{s}^{-1}$, and $\sigma_\phi = 4^\circ$. Cross sampling is applied every $L_{jump} = 2$ iterations on subpartitions of size M_l, which is typically half the number of MHMH particles N_{MHMH}. For our problem, the sampling algorithm is iterated a total of 150 iterations. Finally, the target density $\pi(\cdot)$ is obtained by setting $K = 1$ on the observation density (2.8):

$$\pi(x_i) \propto \prod_m \left\{ 1 + \frac{1-\kappa}{\sqrt{2\pi\sigma_\theta^2(m)}\kappa\lambda} \sum_{p_i=1}^{P} \exp\left\{ -\frac{\left(h_{m\tau}^\theta\left(x_k(t)\right) - y_{t+m\tau}(p_i)\right)^2}{2\sigma_\theta^2(m)} \right\} \right\} \tag{2.15}$$

The pseudo-code of the MHMH block is given in Table 2.3. The algorithm creates a set of particles to input the MHMH sampling algorithm. This initial set consists of some of the residual DOAs and a grid of Q-ϕ values. After the MHMH sampling is over, the mode of these particles is monitored by two conditions. The first condition checks if there are sufficient DOAs in the mode's DOA gate. This ensures that the mode is relatively accurate when used by the particle filter. The second condition makes sure that mode belongs to a physical target as opposed to some alignment of the clutter. If these conditions are

[5]The MHMH block is not to be confused with the MHMH correction of the proposal stage.

Table 2.2: MHMH sampling algorithm.

1. At iteration l, decide if cross-jumping is needed for $x^{(l)}$, i.e., every $L_{jump} = 2$ iterations.
2. If not, then for each particle x_i, $i = 1, 2, \ldots, N_{MHMH}$, use the Metropolis-Hastings scheme:
   - Generate a candidate y_i using $q(x_i, y_i)$,
   - Calculate the acceptance ratio
   $$\alpha(x_i, y_i) = \min\left\{1, \frac{\pi(y_i)q(y_i, x_i)}{\pi(x_i)q(x_i, y_i)}\right\},$$
   - Sample $u \sim \mathcal{U}(0, 1)$,
   - if $u \leq \alpha(x_i, y_i)$, set $x_i^{(l+1)} = y_i$, else, $x_i^{(l+1)} = x_i^{(l)}$.
3. Use the Mode-Hungry scheme:
   - Determine a subpartition of size $M_l < N_{MHMH}$, (e.g.,$M_l = N_{MHMH}/2$),
   - Order the current particles according to their probabilities in descending order: $x_i \rightarrow x_j^*$, where x^* is the ordered particle set,
   - Generate candidates $y^*(1)$ for $x^*(1) = \{x_j^* | j : j = 1, 2, \ldots, N_{MHMH} - M_l\}$, using $q(\cdot, \cdot)$,
   - Calculate the acceptance ratio $\alpha(x^*(1), y^*(1))$, and set $x_j^{(l+1)}$ to $x_j^*(1)$ or $y_j^*(1)$, accordingly for $j = 1, 2, \ldots, N_{MHMH} - M_l$,
   - Distribute M_l candidates $y^*(2)$ from $x^*(1)$ uniformly,
   - Set $x_j^{(l+1)}$ to $y^*(2)$ for $j = N_{MHMH} - M_l + 1, \ldots, N_{MHMH}$.

not satisfied, the MHMH block exits and no new partitions are introduced. Otherwise, we resample N particles with replacement from the MHMH output and declare a new partition in the particle filter. The DOAs belonging to the new mode are gated out, and the procedure is repeated.

2.6 Other Tracker Solutions

This section discusses alternative DOA tracking algorithms to replace the particle filter in Figure 2.3, namely the extended Kalman filter (EKF) and the Laplacian filter. These algorithms incur less computational cost than the particle filter. The Laplacian filter solution can even be implemented in a fully parallel fashion.

Table 2.3: MHMH block pseudo-code.

1. Let $\mathbf{y}_{t,f}^{g}$ be the residual set of DOA observations from the particle filter. If there are no partitions in the particle filter, then $\mathbf{y}_{t,f}^{g} = \mathbf{y}_{t,f}$. Define N_θ as the number DOAs in $\mathbf{y}_{t,f}^{g}$.
2. Set $K_{\text{new}} = 0$. Create a uniform grid for Q and ϕ. Choose N_Q starting values for Q based on the physical motion constraints, and choose N_ϕ heading directions, uniformly spaced in $(-\pi, \pi]$.
3. `while` $N_\theta > 5$,
   - $\{\theta_i\}_{i=1}^{N_\theta} = \{y_{t+m\tau,f}^{g}(p)\}_{m=0}^{4}$.
   - Replicate the Q-ϕ grid N_θ times and combine it with each θ_i. Generate the initial set of particles: $\{x^{(j)}\}_{j=1}^{N_\theta N_Q N_\phi}$.
   - Use MHMH (Table 2.2) to identify one mode (corresponding to one of the targets) of the distribution.
   - If the DOA gate of the highest probability particle has less than 5 DOAs, `break`.
   - If Q value of the highest probability particle is greater than $Q_{th} = 0$, `break`.
   - Resample N particles with replacement from the MHMH output. $K_{\text{new}} = K_{\text{new}} + 1$. Report the resampled set to the particle filter.
   - Delete the DOAs within 3° of the mode DOA track (gating) to obtain the residual $\mathbf{y}_{t,f}^{g}$.

2.6.1 Extended Kalman filter

The EKF solution is a M-scan multiple hypothesis tracking (MHT) filter. Unlike the standard Kalman filter based approaches, the observation equation (2.8) requires M snapshots of data to perform state estimates. Hence, the usage of MHT here differs from traditional approaches[6]. Compared to the particle filter that updates the state every T seconds, the EKF estimates the states every τ seconds, based on the sliding M-DOA snapshots. This is similar to a sliding window approach [7], but assumes that the number of targets is fixed and known since it is estimated by the MHMH block.

In the EKF explicit measurement-to-target associations have to be made. We use the

[6]In traditional MHT, the M-scan associations from one snapshot $t + (m-1)\tau$ are carried over to the next snapshot $t + m\tau$. In this research, the associations during each snapshot $t + m\tau$ depend only on the predicted measurement, based on state x_t, and do not depend on associations at $t + (m-1)\tau$

JPDA to generate a set of hypotheses, relating measurements to targets, during each snapshot [7, 6, 34, 92, 11]. Similar hypotheses are generated for each of the M snapshots, starting at time t. The best hypothesis at each snapshot is chosen, based on the probabilities obtained from the JPDA. These M-best hypotheses are then used to associate the measurements to targets during each batch period. If a measurement-to-target association cannot be found for a target, it is assigned a probability of zero. Using this procedure for each target, a modified innovation term is obtained as

$$\nu_k(t) = \left\{ \beta_m(t) \otimes \left(y_{t+m\tau}(p_k) - h^{\theta}_{m\tau} \hat{x}_k(t|t-\tau) \right) \right\}_{m=0}^{M-1}, \tag{2.16}$$

where $y_{t+m\tau}(p_k)$ is the measurement associated with the target k at time instant $t + m\tau$ and $\beta_m(t)$ is the probability corresponding to the hypothesis at the m-th snapshot, obtained using the JPDA. The state update in the EKF is

$$\hat{x}_k(t|t) = \hat{x}_k(t|t-\tau) + \mathbf{K}_k(t)\nu_k(t), \tag{2.17}$$

where $\mathbf{K}_k(t)$ is the standard Kalman gain. The EKF covariance update is

$$\begin{aligned} \mathbf{P}_k(t|t) = {} & P_{cc} \left(\mathbf{P}_k(t|t-\tau) \right) + \\ & (1 - P_{cc}) \left(\mathbf{P}_k(t|t-\tau) - \mathbf{K}_k(t)\mathbf{M}_k(t)\mathbf{K}_k^T(t) \right) + \widetilde{\mathbf{P}}, \end{aligned} \tag{2.18}$$

where $\widetilde{\mathbf{P}} = \mathbf{K}_k(t)\nu_k(t)\nu_k^T(t)\mathbf{K}_k^T(t)$, $\mathbf{K}_k(t)$ is the Kalman gain, P_{cc} is the probability that the associations are correct, $\mathbf{P}_k(t|t-\tau)$ is the state prediction covariance, and $\mathbf{M}_k(t)$ is the measurement covariance obtained as in the standard EKF [7].

2.6.2 Laplacian filter

The Laplacian filter is a Bayesian filter that uses the product of the partition proposal functions (2.12) as the tracking posterior. Hence, the posterior of the Laplacian filter is a combination of multi-target Gaussian approximations:

$$p(\mathbf{x}_t|\mathbf{y}_t, \mathbf{x}_{t-T}) = \prod_{k=1}^{K} q_k(x_k(t)|\mathbf{y}_t, x_k(t-T)). \tag{2.19}$$

It handles multiple targets using the partition approach and each target posterior is assumed independent. Hence, the filter can be parallelized for faster implementation. Because the

Table 2.4: Simulation parameters - Acoustic target tracker.

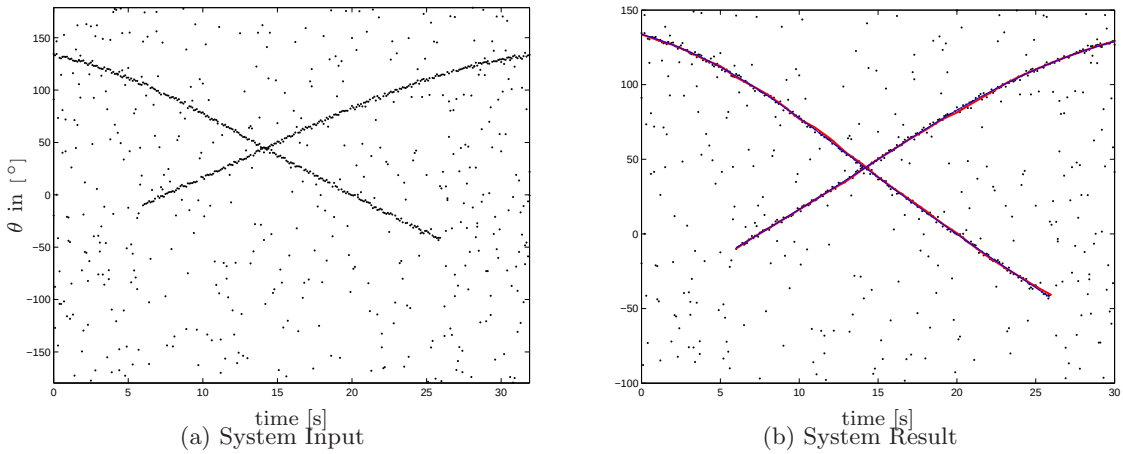
Parameter	**Value**
Number of particles, N	200
θ state noise, $\sigma_{\theta,k}$	1°
Q state noise, $\sigma_{Q,k}$	0.05 s^{-1}
ϕ state noise, $\sigma_{\phi,k}$	10°
Measurement noise, σ_θ	1°
Tracker sampling period, T	1 s
Beamformer batch period, τ	0.1 s
Clutter space parameter, γ	600
Probability of target miss, $\kappa_{0,K}^f$	0.1
Number of batch samples, M	10
Number of DOA peaks, P	4 ($\geq K$)

Laplacian filter uses the Newton-Raphson recursion to calculate the data-likelihood, its complexity is similar to the particle filter for $N < 200$. Every T seconds, the EKF performs M estimates, whereas the particle filter or Laplacian filter perform a single estimate. Furthermore, using M JPDA steps to make explicit target-data associations during each estimate of the EKF also increases its complexity. Considering this, the Laplacian filter is computationally less complex compared to that of the EKF and the particle filter.

2.7 Simulations

This section presents simulation results to demonstrate the performance of the automated target tracking system that uses the DOA tracker and compares the performance of the particle filter to the EKF and the Laplacian filters introduced in the chapter. In addition to these results, we present a tracking scenario that uses prior information of the track or road to improve the target state estimation. Table 2.4 summarizes the simulation parameters used.

Apart from the results presented here, a Matlab based tool was developed to interactively generate scenarios for testing the performance of the particle filter tracking algorithm. This also helped in identifying issues that affected the robustness of the algorithm.



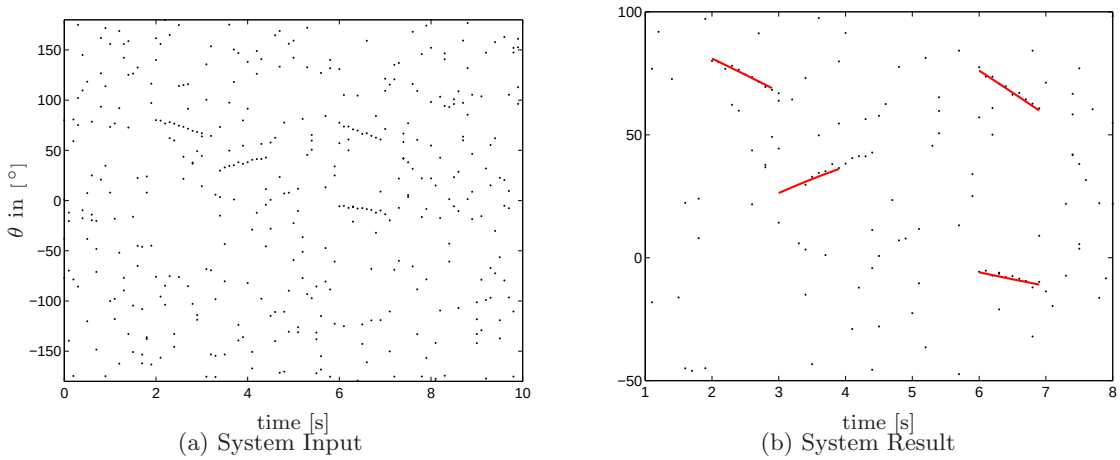
(a) System Input (b) System Result

Figure 2.7: Two targets are successfully initialized and tracked by the automated system. (a) DOA input to the tracking system. The number of peaks $P = 3$ at each time instant. (b) Output of the system. To better illustrate the initialization and tracking, a *zoomed-in* version of the output is shown.

2.7.1 System simulation

Here, we present simulation results for the automated tracking system described in Figure 2.3. We present two scenarios to illustrate the system performance. In the first scenario, two targets are being tracked by the system, where one target appears in the beamformer output between $t = 0$ s and $t = 27$ s, and the other one between $t = 6$ s and $t = 32$ s (Figure 2.7). The targets are successfully initialized by the MHMH block and then deleted by the particle filter. The track coherence is also maintained around $t = 14$ s, because the particle filter also tracks the target motion parameters. In this case, there were no false target initializations by the MHMH block. In the second scenario (Figure 2.8), four short bearing tracks were used as input data to demonstrate the ability to initiate and terminate tracks. It should be noted that, the initialization is sensitive to the number of DOAs used as threshold to determine a valid track.

In Figure 2.9, we provide the results of a simulation to demonstrate the performance of our detection scheme. In this simulation, we randomly picked a target track with a duration of 1 s. We generated the corresponding bearing track for $M = 10$ with varying DOA errors. We then added a spurious track with $M = 10$ by generating random DOA estimates that are uniformly distributed in $[0, 2\pi)$. We repeated this for a Monté-Carlo run of size 100.



(a) System Input (b) System Result

Figure 2.8: Four short target bearing tracks are successfully initialized and killed by the automated system. (a) DOA input to the tracking system. (b) The system can successfully detect the short target tracks. The plot is a *zoomed-in* version to better illustrate the initialization.

During each simulation, we also randomly replaced DOAs in the input data with clutter with probability P_c to simulate missing DOAs. To determine the probability of missed detections in our initialization algorithm, we counted the number of times our initialization algorithm failed to detect the target and then divided that by the size of the Monté-Carlo run. For DOA error variances less than 2°, the detection scheme is quite successful, i.e., the probability of miss is less than 0.1. As the DOA noise increases above the gate size, the detection performance loses its monotonicity.

2.7.2 Comparisons of DOA trackers

In Figures 2.10, 2.11, and 2.12, Monté-Carlo simulations compare the estimates obtained using the particle filter, the EKF, and the Laplacian filter. The state estimation performances of these filters are comparable. The Laplacian filter tends to have less variance than the particle filter in general, because the actual posterior is wider than the Gaussian approximation. Hence, we can consider the Laplacian filter as a mode tracker.

The filters' tracking performance in most cases are similar, when initialized correctly around the true target state as in Figure 2.10. However, the EKF is more sensitive to errors in initialization than the other two filters. The EKF may have difficulty tracking the DOAs correctly, when initialized with the mean of the particle set generated by the MHMH block.

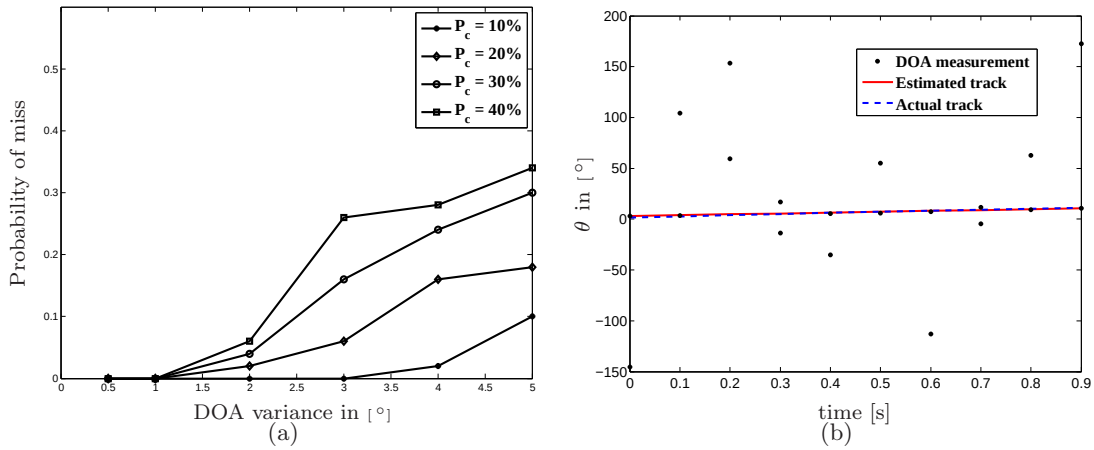
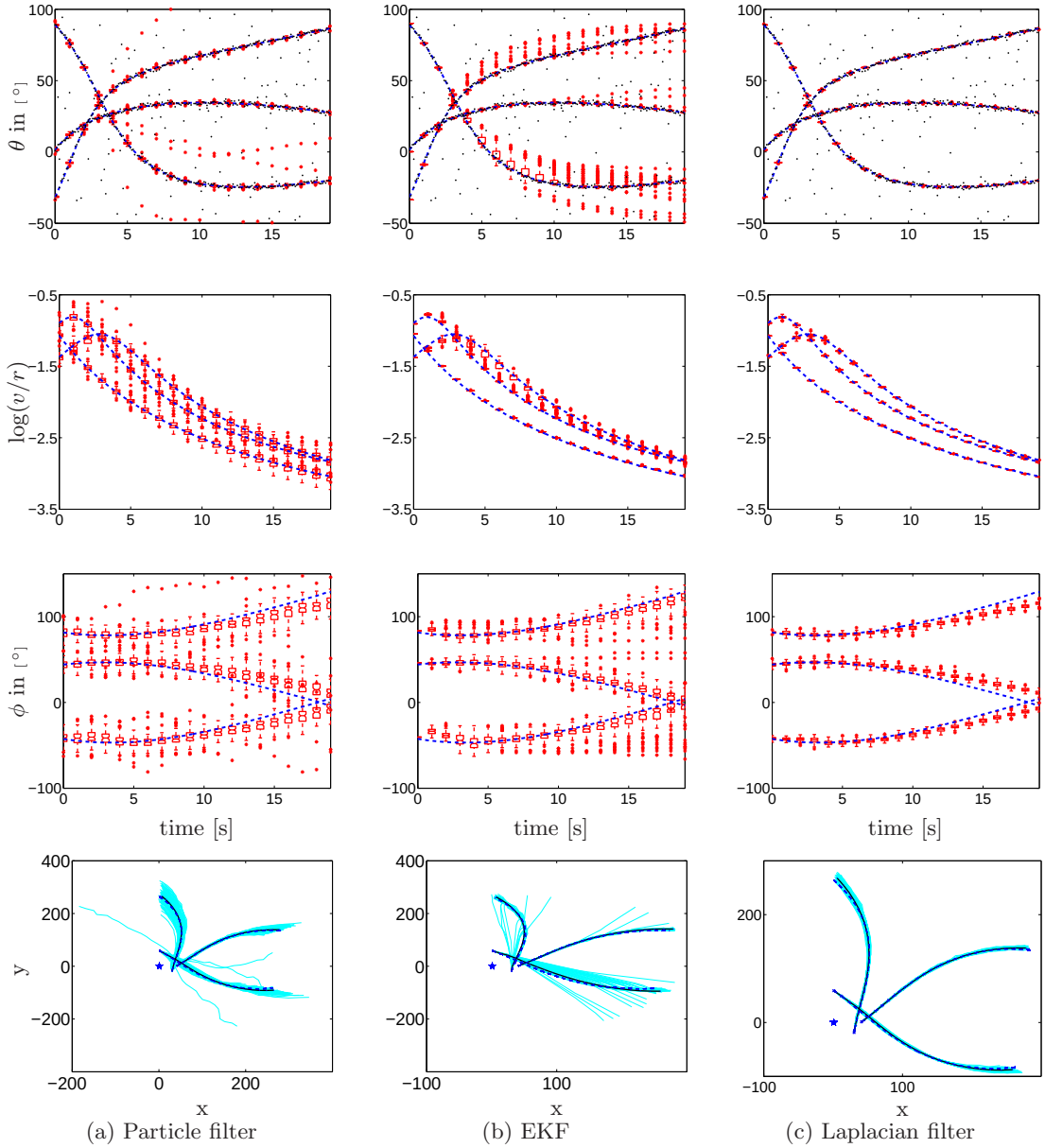


Figure 2.9: (a) Probability of missed detections in the MHMH block with varying noise variance in the measurement data. A partition gate size of $\sigma_g = 3°$ is used. (b) A sample realization with $\sigma_\theta = 2°$ from the Monté-Carlo run that calculates the probability of miss in part (a).

The EKF can completely lose the DOA tracks as in Figure 2.10, if the initial estimates are not accurate, whereas the other two filters can still absorb the discrepancies. There is also a notable decrease in the EKF performance, when targets maneuver as in Figures 2.11 and 2.12. As seen from Figure 2.12, the particle filter is the most robust to target maneuvers compared to the other two methods due to the diversity provided by the particles.

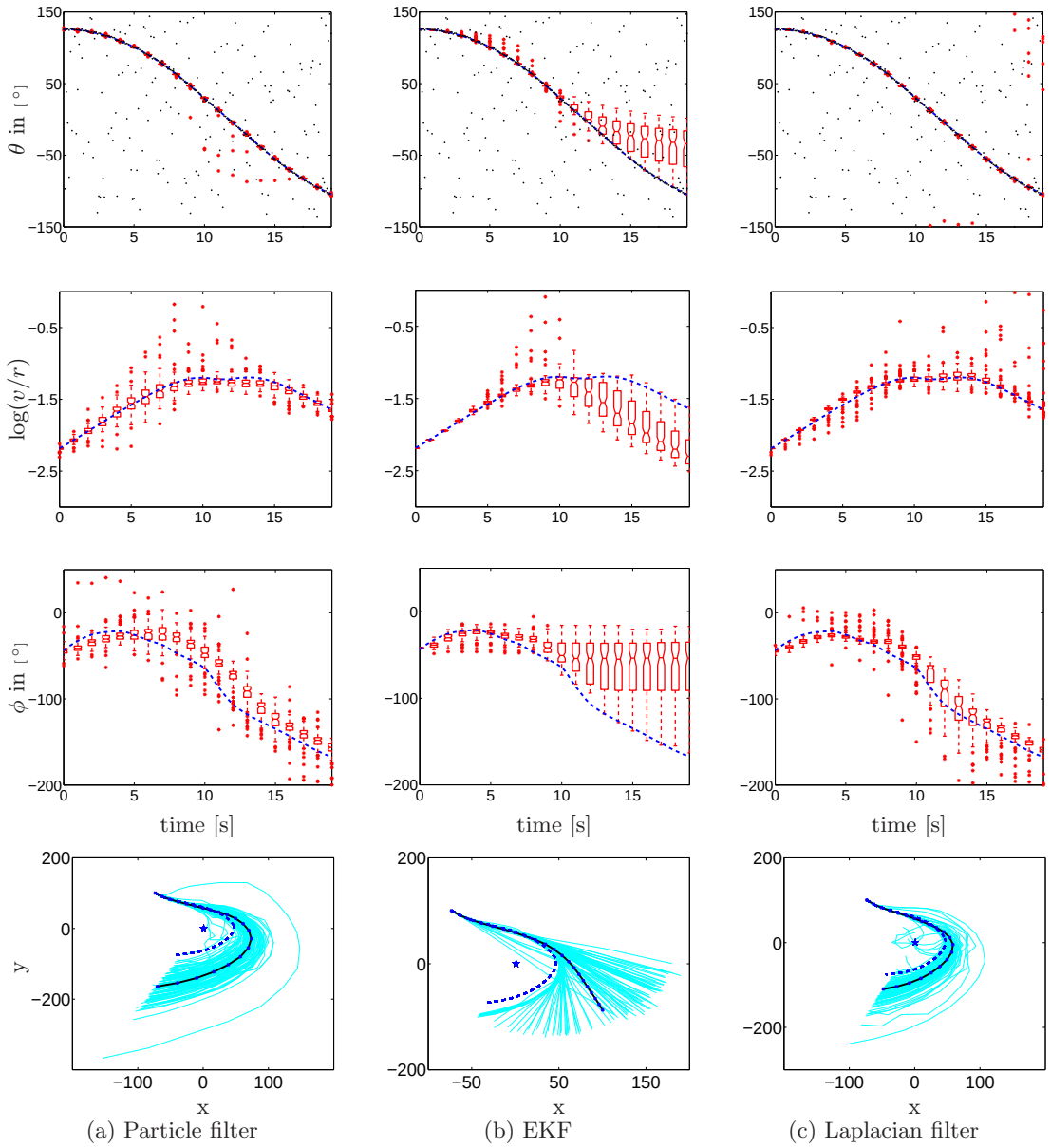
In Figure 2.13 we compare the particle filter, the EKF, and the Laplacian filter presented here with a particle filter that uses the received acoustic signals directly (denoted as DPF) [25]. For the comparison, we use the classical narrow-band observation model to generate the acoustic data as described in [25]; the acoustic SNR at the array is approximately 7-dB in the simulation. For the simulation, we use a constant frequency target at 150 Hz, an 8-microphone circular array with 0.45 wavelength separation, and a sampling frequency of 1000 Hz. For the synthetic target track, process noise is added with variances $\sigma_\theta = 0.5°$, $\sigma_Q = 0.05$, and $\sigma_\phi = 8°$. The process noise explains the abrupt changes in the heading ϕ estimates in Figure 2.13 at each time period.

To use the filters presented in this research, we applied a simple minimum variance distortionless response (MVDR) beamformer on the acoustic data at the center frequency and calculated DOA estimates. The filters being compared use the same noise variances



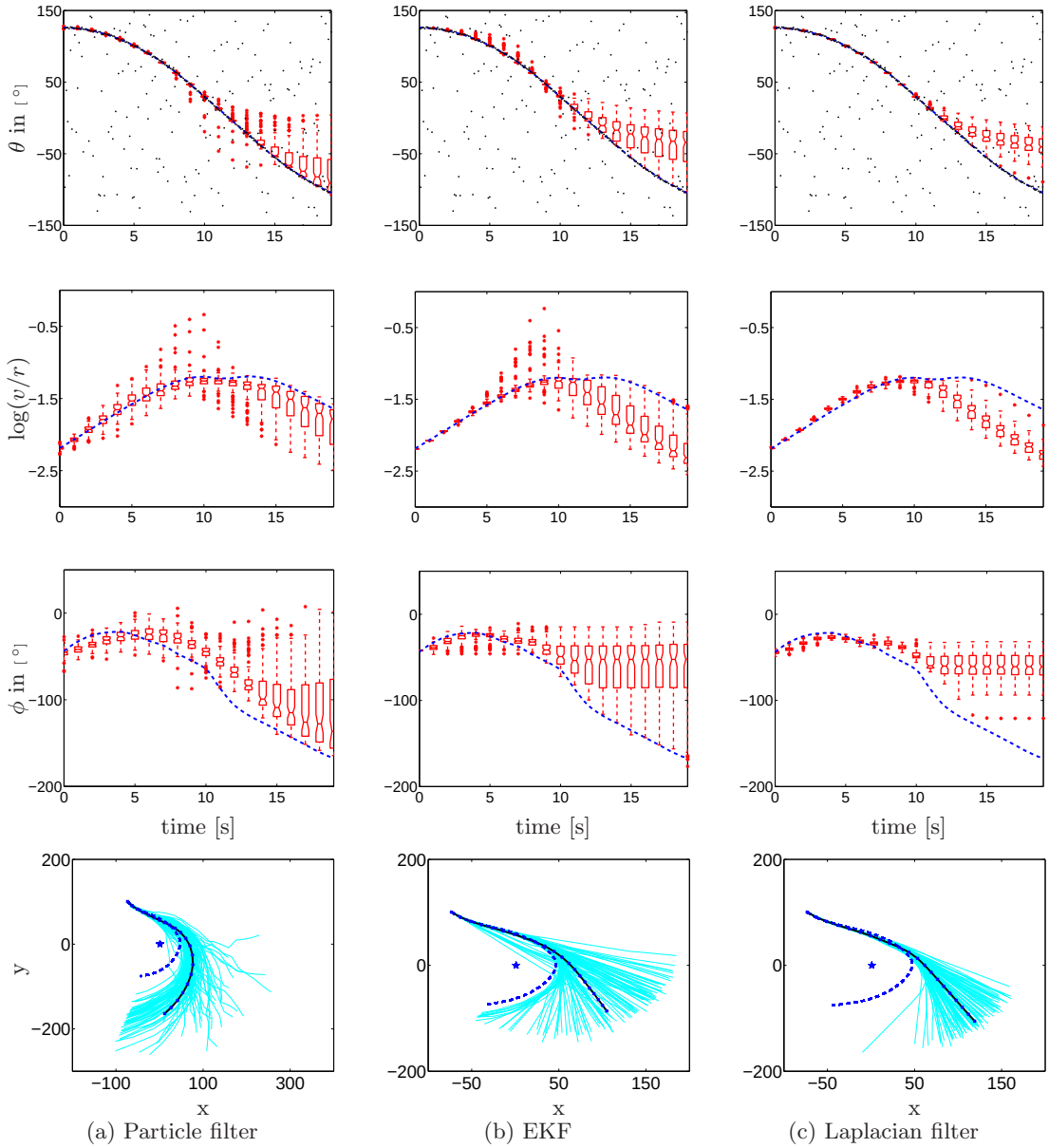
(a) Particle filter (b) EKF (c) Laplacian filter

Figure 2.10: Monté-Carlo run results for each filter using 100 independent noise realizations using Matlab's *boxplot* command. (a) Particle filter. (b) Extended Kalman filter. (c) Laplacian filter. The ground truth is shown with the dashed lines. The first row shows the DOA (θ) estimates, the second row shows the $\log(v/r)$ (Q) estimates, the third row shows the heading (ϕ) estimates. The last row shows the track estimates and their mean. In this example, the Laplacian performs quite well and hence there were no MHMH iterations done in the proposal stage of the particle filter. This is due to the smooth movement of the targets.



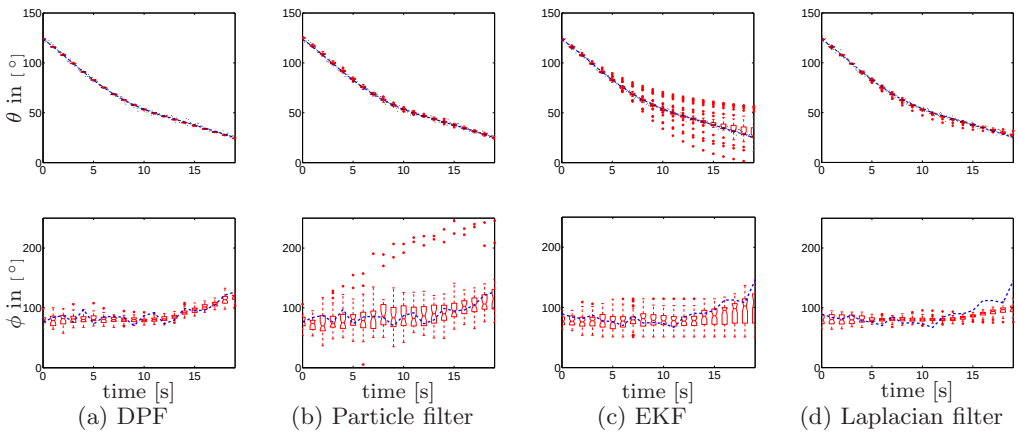
(a) Particle filter (b) EKF (c) Laplacian filter

Figure 2.11: Monté-Carlo run results for each filter using 100 independent noise realizations for a target that maneuvers rapidly. (a) Particle filter. (b) Extended Kalman filter. (c) Laplacian filter. The first row shows the DOA (θ) estimates, the second row shows the $\log(v/r)$ (Q) estimates, the third row shows the heading (ϕ) estimates. The last row shows track estimates and their mean. In this example, the MHMH iterations in the Laplacian calculation were necessary to pull the Laplacian towards the true track because of the abrupt maneuvers at $t = 11$ s and $t = 12$ s. Note that the filter track estimates shadow the true track due to the constant velocity assumption.



(a) Particle filter (b) EKF (c) Laplacian filter

Figure 2.12: Monté-Carlo run results for the same example as in Figure 2.11, but without the MHMH correction. (a) Particle filter. (b) Extended Kalman filter. (c) Laplacian filter. The first row shows the DOA (θ) estimates, the second row shows the $\log(v/r)$ (Q) estimates, the third row shows the heading (ϕ) estimates. The last row shows track estimates and their mean. In this case, the Laplacian filter cannot handle the maneuver. The particle filter can still track the target due to its particle diversity. However, the estimates of the particle filter are now worse than the estimates (Figure 2.11) with the MHMH correction at the proposal stage.



(a) DPF (b) Particle filter (c) EKF (d) Laplacian filter

Figure 2.13: Monté-Carlo comparison of the DOA filters that use batch DOAs to a particle filter that uses the received acoustic signals directly (denoted as DPF, and presented in [25]). (a) DPF. (b) Particle filter. (c) Extended Kalman filter. (d) Laplacian filter. The ground truth is shown in dashed lines, with variances in the estimates indicated by boxplots. The first row shows the DOA (θ) estimates. The second row shows the heading (ϕ) estimates. The DOA estimates for the DPF in (a) are better compared to the other filters. But, the DPF runs three orders of magnitude slower than the particle filter in (b).

for the state update equation. The DPF runs three orders of magnitude slower than the particle filter presented here and performs better because the observation model is perfectly matched by the data. The particle filter has a larger variance partly because of the clutter handling mechanism that creates a noise floor for the pdf (for a single target, the PDF is a raised Gaussian). Not surprisingly, the Laplacian filter show a similar estimation variance as the DPF in bearing estimation because it tracks the mode of the target track with a narrower posterior approximation.

2.7.3 Target tracking with a road prior

Here, we demonstrate an approach to incorporate prior track information into the particle filter tracker for tracking targets on a road [27]. Figure 2.14 illustrates a rather hard tracking scenario, where a single target undergoes various maneuvers along a road. The DOA tracking performance of the particle filter is satisfactory even in the presence of heavy clutter, missing data, and noise (Figure 2.14). However, because of the sudden maneuver at $t = 20$ s, the filter state estimates of Q and ϕ are not as accurate.

If the road information is available along with a calibrated acoustic node, it is optimal to formulate a new tracker, using the target position and velocity as the state vector. However,

it is also possible to incorporate the road information to the DOA-only particle filter tracker, without changing any filter equations. Note that if a target is following a road, its heading direction, in effect, coincides with the road's orientation. Hence, at any given time, as long as the target is on the road, its heading direction will be approximately Gaussian distributed with a mean angle of the road's orientation and some variance (e.g., $(5^\circ)^2$). Denote

$$p_{\{\theta,road\}}(\phi) = \frac{1}{\sqrt{2\pi\sigma^2_{road}}} \exp\left\{-\frac{(\phi - \phi_{\{\theta,road\}})^2}{2\sigma^2_{road}}\right\} \tag{2.20}$$

as the heading prior, calculated using (i) the track information, (ii) the acoustic node position, and (iii) the current DOA θ. Hence, the mean $\phi_{\{\theta,road\}}$ of the heading prior is the orientation of the road, where the line, originating from the node with a slope angle θ, intersects the road.

The heading prior enters the particle filter at the weighting stage:

$$w^{*(i)}_t = w^{(i)}_{t-T} \frac{p(\mathbf{y}_t|\mathbf{x}^{(i)}_t)p(\mathbf{x}^{(i)}_t|\mathbf{x}_{t-T})}{\prod_k g_k(x^{(i)}_k(t)|\mathbf{y}_t, x^{(i)}_k(t-T))} \prod_k p_{\{\theta^{(i)}_k,road\}}(\phi^{(i)}_k), \tag{2.21}$$

where it is assumed that the targets are moving along the same road. Figure 2.15 illustrates the tracking results with the weighting strategy that includes the heading prior. The DOA tracking results are comparable to the case without the prior; however, there is an improvement in the Q and ϕ parameter estimates. An improved method that uses multiple models to incorporate prior road information is discussed in [27].

2.8 Discussions

A framework for an automated, multi-target DOA tracking system based on a batch measurement model was developed in this chapter. The DOA batches are treated as images to naturally handle the data association and ordering issues. The presence of multiple targets is handled using a partition approach. The automated tracking system has a beamformer to estimate DOAs using acoustic data, a mode hungry Metropolis-Hastings block that employs a matching pursuit approach to estimate the number of targets, and a particle filter block to recursively track the targets. Simulation results demonstrate the performance of the batch measurement-based particle filter tracking system in tracking multiple targets. We also present two other filters to perform the tracking: the extended Kalman filter and the

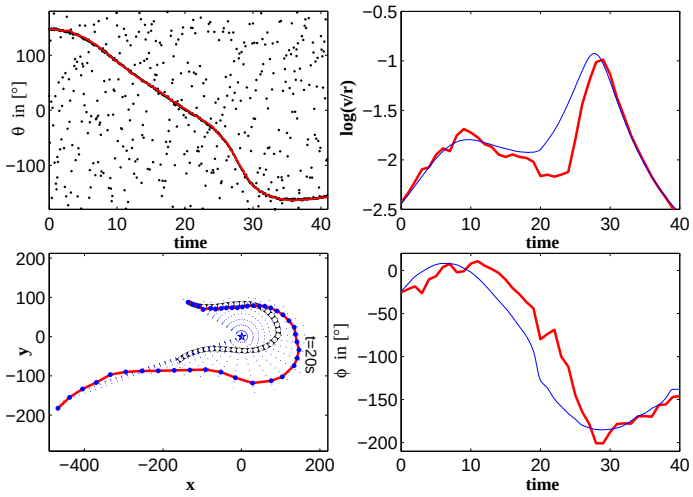


Figure 2.14: (*Top Left*) Particle filter DOA (θ) tracking results (solid line). (*Bottom Left*) True track vs. calculated track. Because of the heading (ϕ) and $\log(v/r)$ (Q) estimation errors, the calculated filter track shadows the true x-y track. Note that the particle filter track is estimated, using the filter outputs and the correct initial position. (*Bottom Right*) Filter heading (ϕ) estimates. Note the sudden target maneuver at time $t = 20$ s. (*Top Right*) Filter $\log(v/r)$ (Q) estimate lags the true target $\log(v/r)$, because the state update function constrains the target movement to constant velocity.

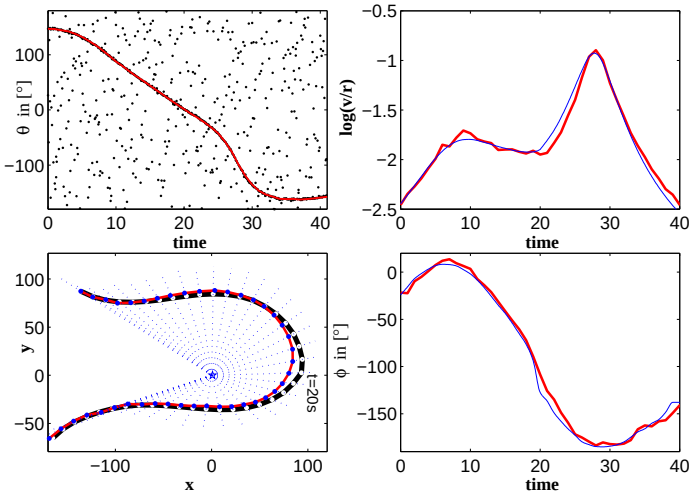


Figure 2.15: The tracking estimates for the scenario shown in Figure 2.14 are improved due to the prior information about the road. The filter input is the same as in Figure 2.14. The use of the prior information about the road has improved the target heading (ϕ) and $\log(v/r)$ (Q) estimates, and as a result the calculated target track closely matches the true track.

Laplacian filter. These filters are compared to the particle filter, for sensitivity to initialization and robustness. When compared to the extended Kalman filter and the Laplacian filter, the particle filter handles the target maneuvers better.

CHAPTER III

RADAR RANGE-ONLY MULTI-TARGET TRACKING

3.1 Introduction

This chapter develops a range-only multi-target particle filter tracker. The particle filter algorithm is similar to that in Chapter 2. Some parts of the discussion in this chapter also appear in a related publication [30], which is a joint work by Dr. Volkan Cevher and the author. The radar range tracking problem has attracted some interest in the literature [103], [104], [110], [45]. This tracking problem is usually formulated using state-space models, where the target's motion is approximated locally as constant velocity motion, and the observations are temporal snapshots of radar range and range-rate estimates. Then, to estimate the state vector consisting of the target's position and velocity using range-only measurements, a mobile platform must be used that executes known maneuvers to guarantee system observability [110]. Otherwise, multiple beacons must be used to track the state vector by virtue of triangulation [45]. In [104] range and range-rate measurements, obtained at a non-stationary observer, are used to estimate the states of a single target. The application in [104] uses a deterministic state model and hence the EKF provides better estimates compared to a particle filter approach. Most past research did not consider multiple targets when using range-only measurements. In our research we use a batch of range measurements to track multiple targets. We achieve this using the image template-matching idea described in Section 2.1.1.

The particle filter algorithm developed here tracks a state vector that consists of the target direction-of-arrival (DOA) $\theta(t)$, the logarithm of the target range $R(t)$, the target speed $v(t)$, and the target heading $\phi(t)$, using a batch of range-only measurements, obtained at a *stationary* sensor. The angles are measured counterclockwise with respect to the x-axis. We prove that the particle filter state vector is observable given at least three range measurements under rotational and planar symmetric ambiguities. The proof makes use of

Stewart's theorem for triangles in geometry.

The motivation for using the state vector of the particle filter is a low-power radio frequency (RF) sensor, implemented at the University of Florida, which transmits a microwave signal to determine the range, velocity, and size of detected targets [41]. The sensor is capable of providing range estimates at 32 ms intervals, with a range resolution of approximately 2 m on a range-Doppler map. Up to a maximum distance of 100 m, the current system is capable of producing range estimates for multiple ground vehicles as well as human targets. The radar hardware is likely to have a larger detection range with hemispherical coverage in the future. Note that the filter equations in this research are developed using range-only measurements so that it is also applicable to amplitude tracking problems when amplitude is proportional to $\frac{1}{r}$ or $\frac{1}{r^2}$, where r represents range. Additional velocity measurements or range-rate measurements can also be incorporated through the data likelihood equation assuming independence of the measurements.

3.2 State-space Description

3.2.1 State update model

The filter state vector $\mathbf{x}_t = \left[\, x_1^T(t),\ x_2^T(t),\ \cdots,x_K^T(t) \,\right]^T$ consists of K partition state vectors $x_k(t)$, one for each target, indexed by k, $k = 1, \ldots, K$. Each partition has the corresponding target motion parameters $x_k(t) \triangleq [\theta_k(t),\ R_k(t),\ v_k(t),\ \phi_k(t)]^T$, where $\theta_k(t)$ is the DOA, $R_k(t)$ is the logarithm of the range $r(t)$, $v_k(t)$ is the speed, and $\phi_k(t)$ is the heading direction. The logarithm of the range is used in the state vector because the range errors are modeled as multiplicative noise.

The state update equation can be derived from the geometry (Figure 3.2) imposed by a motion model on the state vector, and we model the target motion with a locally constant velocity model. The resulting state update equation is nonlinear:

$$x_k(t+T) = h_T\left(x_k(t)\right) + u_k(t), \tag{3.1}$$

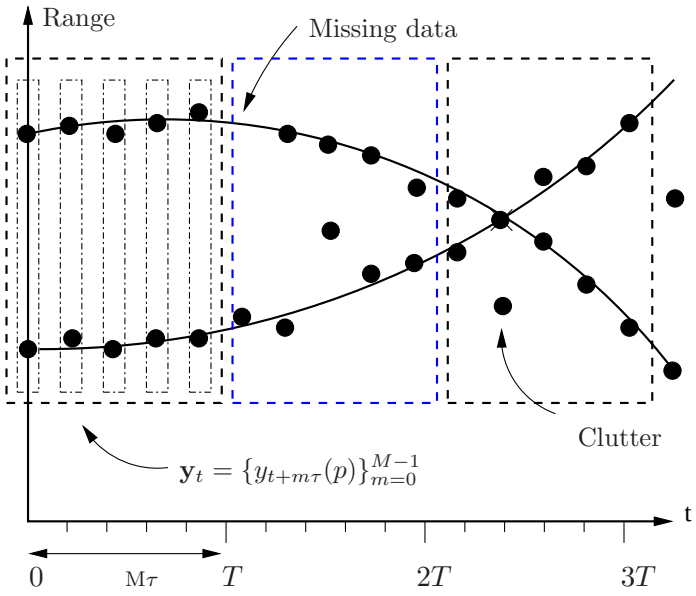


Figure 3.1: Observation model using batch of radar range measurements. Note that the range measurements are not necessarily ordered. However, the image based observation approach provides a natural ordering, when targets are being tracked by the particle filter.

where $u_k(t) \sim \mathcal{N}(0, \Sigma_u)$ with $\Sigma_u = \text{diag}\{\sigma^2_{\theta,k}, \sigma^2_{R,k}, \sigma^2_{v,k}, \sigma^2_{\phi,k}\}$ and

$$h_T(x_k) = \begin{bmatrix} \tan^{-1}\left\{\frac{e^{R_k}\sin\theta_k + Tv_k\sin\phi_k}{e^{R_k}\cos\theta_k + Tv_k\cos\phi_k}\right\} \\ \frac{1}{2}\log\left\{e^{2R_k} + T^2v_k^2 + 2Te^{R_k}v_k\cos(\theta_k - \phi_k)\right\} \\ v_k \\ \phi_k \end{bmatrix}. \tag{3.2}$$

3.2.2 Observation model

The observations $\mathbf{y}_t = \{y_{t+m\tau}(p)\}_{m=0}^{M-1}$ consist of range estimates from a radar sensor at each batch index m. This observation model can be visualized as shown in Figure 3.1. The radar returns over the time-interval τ are used to estimate, possibly multiple, target ranges. A batch of M range estimates are used by the particle filter to estimate the target state every $T = M\tau$ seconds. It is assumed that the measurements in a batch are normally distributed around the true target ranges with variance σ_r^2 and have a constant miss probability κ. A batch may include spurious peaks due to clutter that are Poisson distributed with rate λ.

3.2.3 Observability of the state vector

Figure 3.2 illustrates that it is possible to determine v and $\theta - \phi$ given three range measurements. This follows from Stewart's theorem that can be proved by using the law of cosines on the triangles $\triangle OAB$ and $\triangle OAC$. Applying Stewarts's theorem to the triangle $\triangle OAC$, we get

$$AB \cdot (OC)^2 + BC \cdot (OA)^2 = (AB + BC) \cdot (OB)^2 + AB \cdot (BC)^2 + BC \cdot (AB)^2 \qquad (3.3)$$

With appropriate substitutions of the parameters shown in Figure 3.2, the velocity v can be determined from three range measurements as

$$v = \frac{1}{\tau}\sqrt{\frac{1}{2}\left(r_1^2 + r_3^2\right) - r_2^2}.$$

Note that θ and ϕ cannot be determined uniquely given only the range measurements: for example, the trajectories defined by ABC, $A_1B_1C_1$, and $A_2B_2C_2$ can result in the same range measurements $\{r_1, r_2, r_3\}$. However, if initialized correctly the target states θ, ϕ and v can be estimated sequentially in the target tracker.

3.3 Particle Filter Algorithm

The particle filter algorithm used here is similar to the one used in the DOA-tracker in Section 2.4. Hence, we only present an outline of the steps here. As opposed to the acoustic tracking system, which uses a MHMH block to estimate the number of targets, the range-only tracker assumes the number of targets to be known. All other discussions in Section 2.4 are relevant to the range-only tracker.

3.3.1 Data likelihood function

We derive the data likelihood function considering the output of one batch period $\mathbf{y}_m = y_{t+m\tau}(p)$, where $p = 1, \ldots, P_m$ for each m. The range measurements $\mathbf{y}_m$ may belong to none, i.e.,clutter, or some combination of the targets in the particle filter partitions. Hence the data likelihood function is given as

$$p(\mathbf{y}(t)|\mathbf{x}(t)) \propto \prod_k \prod_m \left\{ 1 + \frac{1-\kappa}{\sqrt{2\pi\sigma_r^2(m)}\kappa\lambda} \sum_{p_i=1}^{P} \exp\left\{ -\frac{\left(h_{m\tau}^r\left(x_k(t)\right) - y_{t+m\tau}(p_i)\right)^2}{2\sigma_r^2(m)} \right\} \right\}, \qquad (3.4)$$

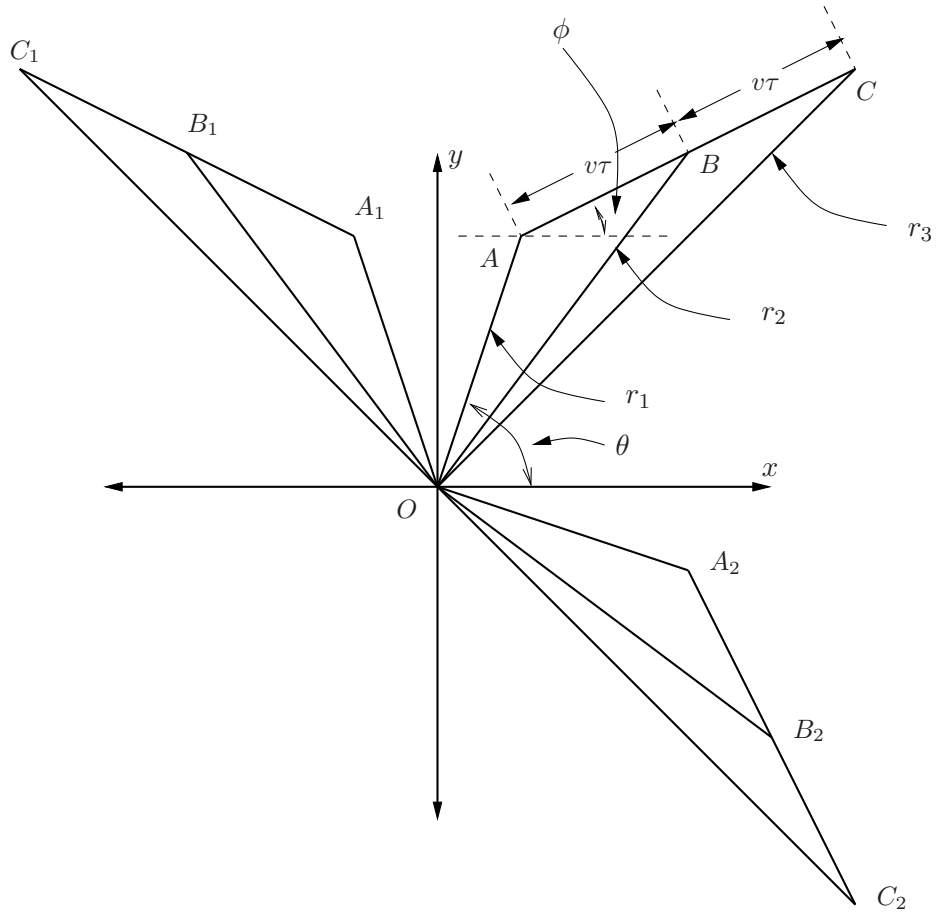


Figure 3.2: Observability of the state vector using three range measurements. The speed, which is assumed constant, can be calculated as $v = \frac{1}{\tau}\sqrt{\frac{1}{2}(r_1^2 + r_3^2) - r_2^2}$. Triangles $\triangle OA_1C_1$ and $\triangle OA_2C_2$ are rotated versions of $\triangle OAC$ that demonstrate the rotational and planar-symmetric ambiguities, respectively.

where κ is a constant miss probability, λ is the Poisson rate of the clutter distribution, the superscript r on the state update function h refers to the range component of the state update (3.2), and $\sigma_r^2(m)$ is the variance of the range estimates provided by the radar sensor.

3.3.2 Proposal function

The arguments supporting the proposal function in Section 2.4.4 for the acoustic tracker are applicable to the range tracker. Hence, we have a partitioned proposal function to propose new particles. For a given partition, if we assume that the data is only due to that partition and clutter (i.e., the range data corresponding to other partitions are treated as clutter), we can factor the data likelihood (3.4). Hence, the partition posterior is the following

$$q_k(x_k(t)|\mathbf{y}_t, x_k(t-T)) \propto p(\mathbf{y}_t|x_k(t))p(x_k(t)|x_k(t-T)), \tag{3.5}$$

where $p(\mathbf{y}_t|x_k(t))$ can be obtained from (3.4) and $p(x_k(t)|x_k(t-T)) = \mathcal{N}\left(h_T(x_k(t-T)), \Sigma_u\right)$ with $\Sigma_u = \text{diag}\{\sigma^2_{\theta,k}, \sigma^2_{R,k}, \sigma^2_{v,k}, \sigma^2_{\phi,k}\}$.

Next, the target partition posterior function in (3.4) is approximated as Gaussian using the Laplace method. Here, as in Section 2.4, the data statistics are computed using the Newton-Raphson recursion (see also Appendix A). The final expression for the partition proposal functions to be used in the particle filter is given by

$$g_k(x_k(t)|\mathbf{y}_t, x_k(t-T)) \sim \mathcal{N}\left(\mu_g(k), \Sigma_g(k).\right) \tag{3.6}$$

The Gaussian density parameters are

$$\begin{aligned} \Sigma_g(k) &= \left(\Sigma_y^{-1}(k) + \Sigma_u^{-1}\right)^{-1}, \\ \mu_g(k) &= \Sigma_g(k)\left(\Sigma_y^{-1}(k)x_{k,\text{mode}} + \Sigma_u^{-1}h_T(x_k(t-T))\right), \end{aligned} \tag{3.7}$$

where $x_{k,\text{mode}}$ is the mode of $p(\mathbf{y}_t|x_k(t))$, and $\Sigma_y^{-1}(k)$ is the Hessian of $p(\mathbf{y}_t|x_k(t))$ at $x_{k,\text{mode}}$, calculated using the Newton-Raphson recursion. The pseudo-code for the particle filter algorithm in the range-only tracker is given in Table 3.1.

3.4 Simulations

This section presents simulation results to demonstrate the performance of the range-only multi-target tracker. Both single- and multi-target tracking scenarios are considered. The

Table 3.1: Range-only particle filter tracker pseudo-code.

Given the observed data $\mathbf{y}_t = \{y_{t+m\tau}(p)\}_{m=0}^{M-1}$ in $[t, t+T)$, do

1. For $i = 1, 2, \ldots, N$

   - For $k = 1, 2, \ldots, K$
     sample $x_k^{(i)}(t) \sim g_k(x_k^{(i)}(t)|\mathbf{y}_t, x_k^{(i)}(t-T))$, given by (3.6).
   - Form $\mathbf{x}_t^{(i)} = \left[\, x_1^{(i)}(t),\ x_2^{(i)}(t), \ldots,\ x_K^{(i)}(t) \right]^T$.

2. Calculate the weights

$$w^{*(i)}_t = w^{(i)}_{t-T} \frac{p(\mathbf{y}_t|\mathbf{x}_t^{(i)}) p(\mathbf{x}_t^{(i)}|\mathbf{x}_{t-T})}{\prod_k g_k(x_k^{(i)}(t)|\mathbf{y}_t, x_k^{(i)}(t-T))},$$

   where $p(\mathbf{y}_t|\mathbf{x}_t^{(i)})$ is fully joint observation density, given by (3.4).

3. Normalize the weights: $w_t^{(i)} = \dfrac{w^{*(i)}_t}{\sum_i w^{*(i)}_t}$.

4. Perform the estimate: $E\{f(\mathbf{x}_t)\} = \sum_{i=1}^N w_t^{(i)} f(\mathbf{x}_t^{(i)})$.

5. Resample the particles:

   - Heapsort the particles in a ascending order according to their weights: $\mathbf{x}_t^{(i)} \to \tilde{\mathbf{x}}_t^{(i)}$.
   - Generate $\omega \sim \mathcal{U}[0, 1)$.
   - For $j = 1, 2, \ldots, N$
     a. $u^{(j)} = \frac{j-\omega}{N}$,
     b. Find i, satisfying $\sum_{l=1}^{i-1} \tilde{w}_t^{(i)} < u^{(j)} \leq \sum_{l=1}^{i} \tilde{w}_t^{(i)}$,
     c. Set $\mathbf{x}_t^{(j)} = \tilde{\mathbf{x}}_t^{(i)}$.

Table 3.2: Simulation parameters - Range-only target tracker

Parameter	**Value**
# particles, N	200
θ state noise, $\sigma_{\theta,k}$	1°
R state noise, $\sigma_{R,k}$	0.1 m
v state noise, $\sigma_{v,k}$	0.1 m/s
ϕ state noise, $\sigma_{\phi,k}$	10°
Measurement noise, σ_r	1 m
Tracker sampling period, T	1 s
# batch samples, M	10
# radar range returns, P	3 ($\geq K$)

synthetic range measurement data is obtained using the constant velocity motion of the targets. The data is perturbed using zero-mean Gaussian noise with variance σ_r^2. The simulation parameters for the results shown in Figures 3.3 and 3.4 are shown in Table 3.2.

Figure 3.3 shows the tracking results for a single-target tracking scenario. The target moves with a constant speed of 14 m/s. The unordered radar range measurements with clutter (Figure 3.3) are used as input to the filter. The tracker is initialized with N particles generated by adding the state noise (Table 3.2) to the true target state at $t = 0$ s. The results show that the tracker can associate the range measurements related to the target motion and can also provide accurate temporal estimates of the single target state. Because of our constant velocity motion assumption, the estimated heading ϕ lags the true target heading. Next, the tracking results for two targets are shown in Figure 3.4. The targets move with a constant speed of 14 m/s and 14.5 m/s. The particle filter was able to resolve the data association issues at times $t = 3$ s and $t = 30$ s. The motion estimates are unique because we have assumed a correct initialization.

The estimates obtained using range-only measurements can be further improved by using additional measurements. For example, define $\alpha_k = v\cos(\theta_k - \phi_k)$ as the range-rate. Then, we can treat the additional range-rate measurements as an independent observation and calculate $p(\boldsymbol{\alpha}|\mathbf{x}_t)$, $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_K]^T$, by using the same joint density association approach as (3.4). The range-rate measurements are then incorporated at the weighting

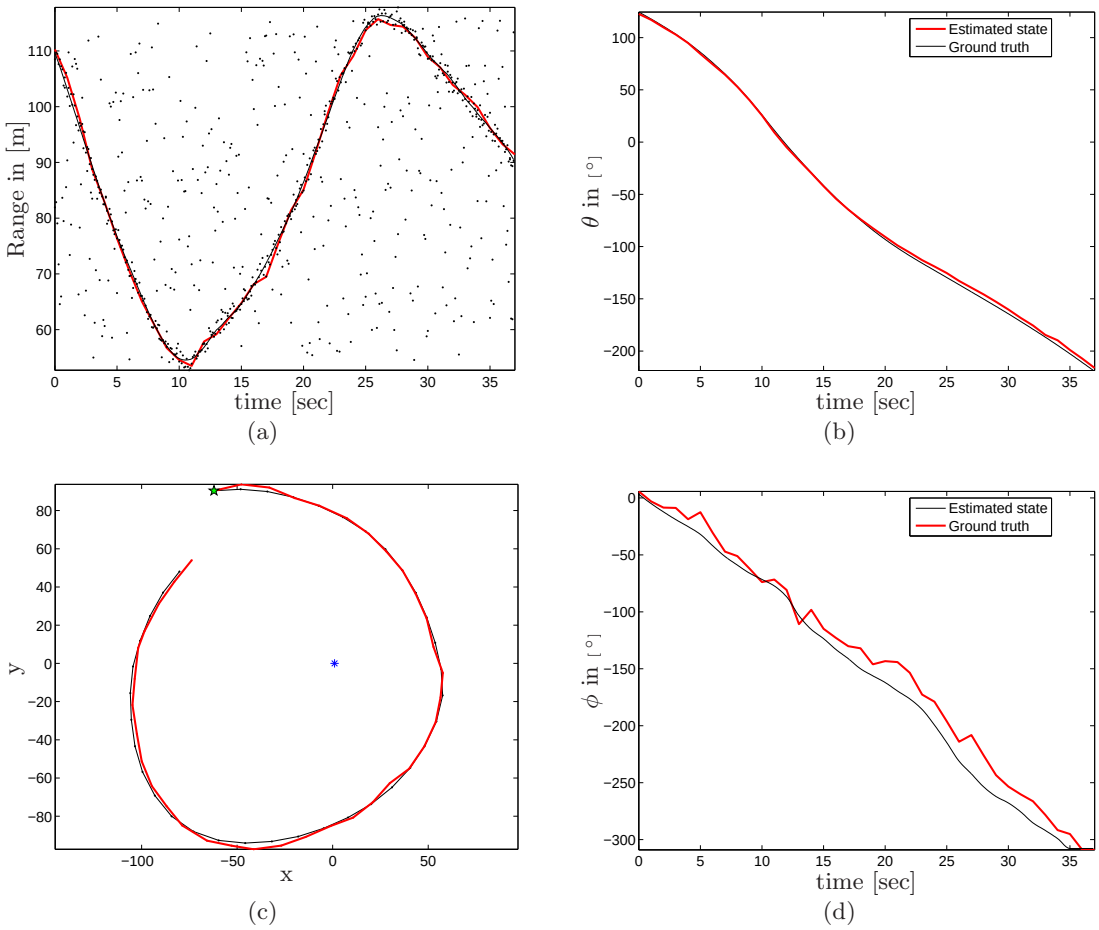


Figure 3.3: Single-target tracking using range-only measurements. (a) Black dots are the range measurements. The estimated target range follows the ground truth. (b) Target DOA (θ) estimates closely follow the true target DOAs. (c) Target track estimates calculated from the state estimates in (a), (b), and (c). The pentagram and the star indicate the target starting position and the radar position, respectively. (d) Target heading (ϕ) estimates have more oscillations than the DOA estimates, because they are more sensitive to the noise in the range measurements.

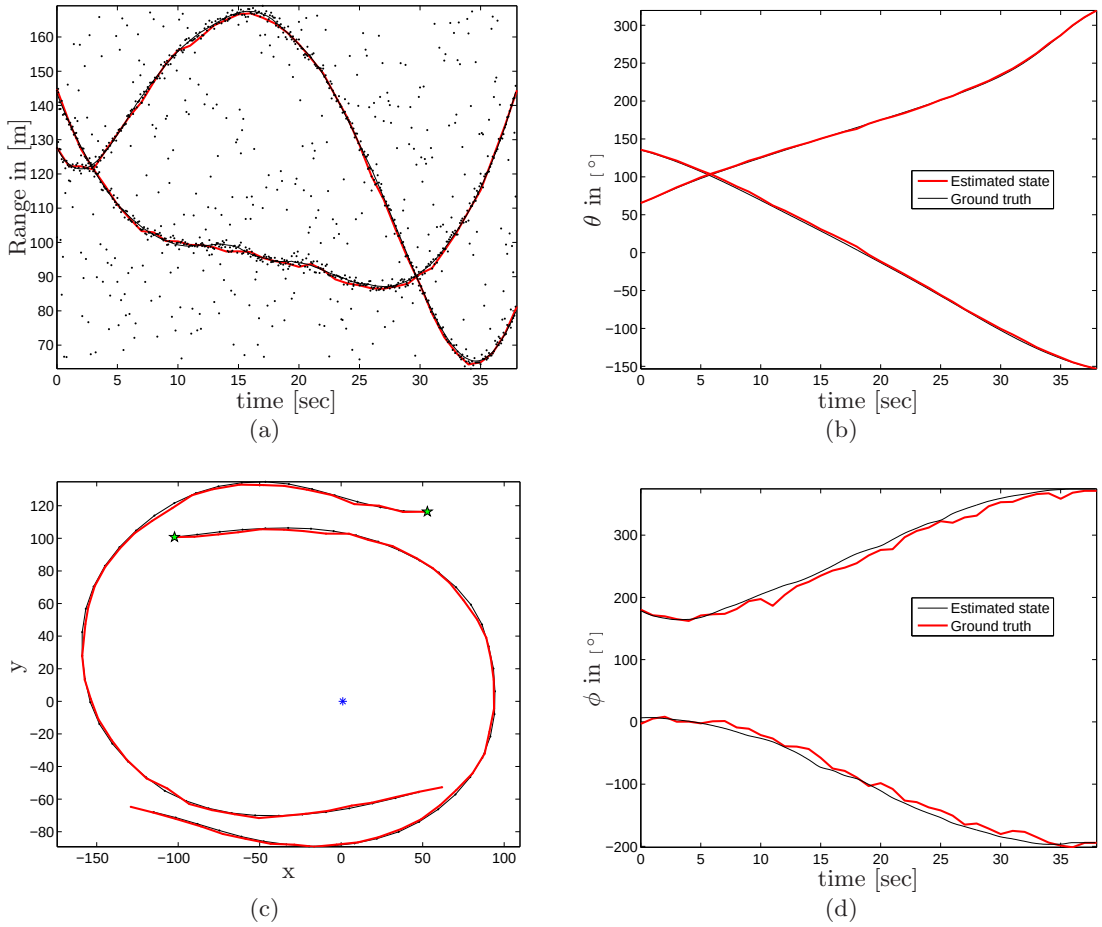


Figure 3.4: Multi-target tracking using range-only measurements. a) Black dots are the range measurements. The estimated target range follows the ground truth. (b) Target DOA (θ) estimates closely follow the true target DOAs. (c) Target track estimates calculated from the state estimates in (a), (b), and (c). The pentagrams and the star indicate the target starting positions and the radar position, respectively. (d) The heading (ϕ) estimates have more oscillations than the DOA estimates, because they are more sensitive to the noise in the range measurements. The filter does a good job in associating the data with the targets because of the joint density approach.

stage of the particle filter algorithm:

$$w^{*(i)}_t = w^{(i)}_{t-T} \frac{p(\mathbf{y}_t|\mathbf{x}^{(i)}_t)p(\mathbf{x}^{(i)}_t|\mathbf{x}^{(i)}_{t-T})}{\prod_k g_k(x^{(i)}_k(t)|\mathbf{y}_t, x^{(i)}_k(t-T))} p(\boldsymbol{\alpha}|\mathbf{x}^{(i)}_t), \tag{3.8}$$

where w^* is the unnormalized weight. The range-rate measurements further improve the tracking estimates as shown in Figure 3.5. The tracking scenario considered here is similar to the single-target tracking scenario described earlier in this section. Here a higher measurement noise variance of $\sigma_r^2 = 4$ m is used with a range-rate variance of $\sigma_\alpha^2 = 1$, and the range measurements have one peak. The use of range-rate measurements improves the heading estimates obtained without the use of range-rate measurements, as shown in Figure 3.5 (d). As a result of this, the estimated target track closely follows the ground truth, as shown in Figure 3.5 (c).

3.5 Discussions

In this chapter, we present a range-only multi-target particle filter tracker based on a batch measurement model. As in the acoustic DOA tracker in Chapter 2 the radar range measurement batches are treated as an image to naturally handle the data association and ordering issues. The presence of multiple targets is handled using a partition approach. The observation likelihoods are calculated jointly and are assigned by using the templates created by the state vectors and the state update equation.

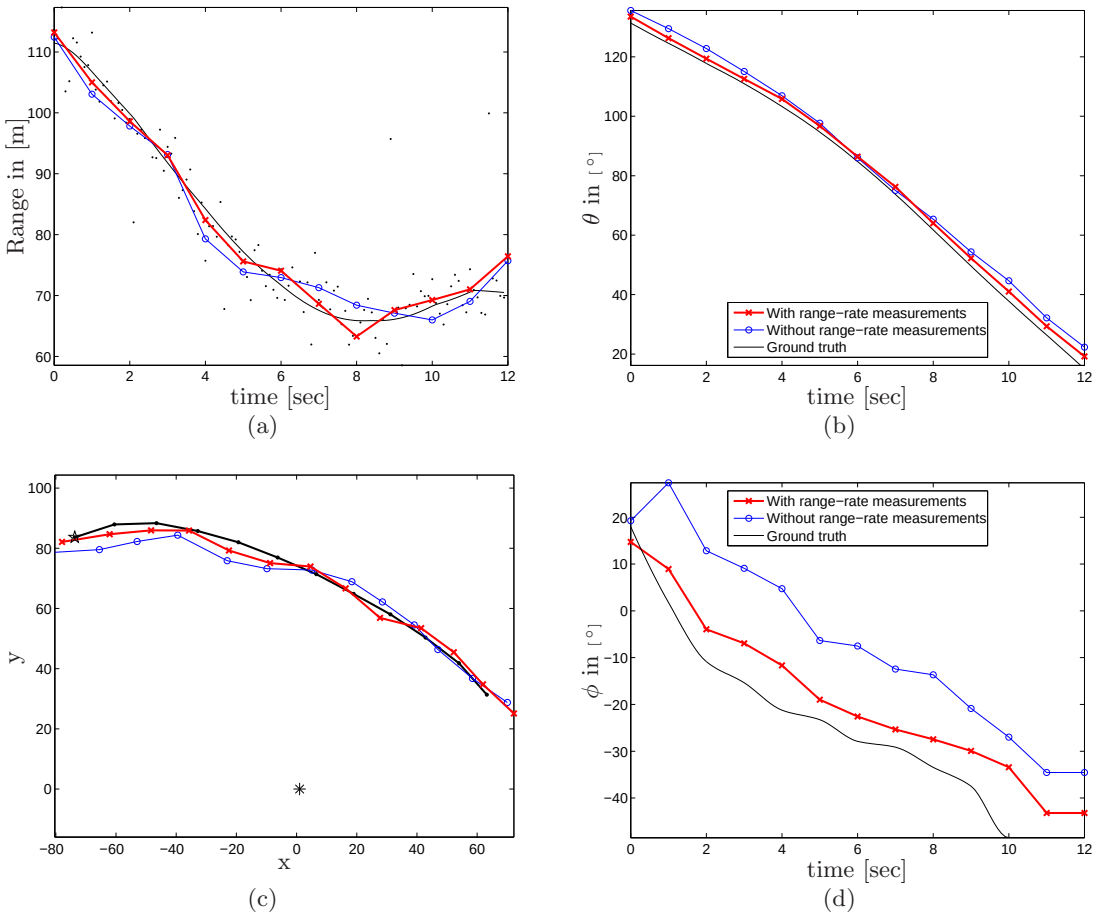


Figure 3.5: Single-target tracking using range and range-rate measurements. Heading estimates (ϕ) with range-rate measurements are closer to the true values than without the range-rate measurements. This scenario is similar to the one in Figure 3.3, but with a single peak in the range measurements and increased measurement noise variance of four ($\sigma_r^2 = 4$ m). (a) Black dots are the range measurements. The estimated target range follows the ground truth. (b) Target DOA (θ) estimates closely follow the true target DOAs. (c) Target track estimates calculated from the state estimates in (a), (b), and (c). The pentagram and the star indicate the target starting position and the radar position, respectively. (d) The heading (ϕ) estimates when using range and range-rate measurements are improved, when compared to the range-only estimates.

CHAPTER IV

MIXED-MODE IMPLEMENTATION OF A PARTICLE FILTER

4.1 Introduction

In this chapter, we develop a mixed-mode implementation of a particle filter and compare it to a digital implementation. This chapter differs from Chapters 2 and 3 where the focus was on the algorithmic aspects of a batch-based particle filter target tracking algorithm. Here we develop hardware implementation strategies for a simple bearings-only particle filter tracker. Specifically, we develop a mixed-mode implementation that uses analog components to realize certain stages in the particle filter. Some sections of this chapter also appear in a related publication [123]. The analog components used in the particle filter were synthesized using MITE networks designed by Shyam Subramanian of the CADSP group in Georgia Institute of Technology. The implementation of the batch measurement-based tracker is discussed in Chapter 5.

The bearings-only tracking problem has been extensively studied in the literature [7], [103], and the scenario considered here is similar to the one in [55], [16]. The specific bearings-only algorithm considered here tracks a single, nonmaneuvering, constant velocity target. The measurements are single, noisy bearings (or angles) obtained at a stationary sensor. The estimated states are the target's position and velocity in the x-y Cartesian space. The particle filter algorithm used is the SIR or *bootstrap* filter, presented in Section 1.1.3, which uses the state update to propose particles.

Past research includes an efficient digital architecture for particle filters that resulted in a Field Programmable Gate Array (FPGA) prototype [17]. The earlier implementations of particle filters in [17] and [4] focused on achieving a faster sampling rate or reducing the execution time. They concentrated on efficient implementation of the resampling stage and exploit parallelization in the particle filter. They adopted a coarse-grain approach where M distributed processing elements are used to process N particles, assuming that

the ratio $\frac{N}{M} \gg 1$. More recently, a single instruction multiple data (SIMD) architecture for particle filters has been presented in [81]. It uses N processors to process N particles and has a time complexity of $O\left((\log N)^2\right)$. This uses a fine-grain approach with the ratio $\frac{N}{M} \approx 1$.

The power dissipated in the computations of the particle filter algorithm needs to be minimized when used in an energy constrained environment, e.g., a network of unattended sensor nodes. Because particle filters do not approximate the nonlinearities in the state-space systems, they are computationally complex. A promising approach to achieve low power dissipation is to have cooperative systems that have both digital and analog components [57]. This is the approach described in this chapter. The use of an analog computer as a co-processor in a digital computer to obtain fast and approximate solutions has recently been demonstrated in [38], [39]. The authors in [38] also show that the analog computer dissipates less power than a general purpose microprocessor or a digital signal processor, when used to solve differential equations.

In the particle filter algorithm considered here, the weight evaluation stage uses nonlinear functions such as the *arctan* and *Gaussian*. These nonlinear functions are implemented using analog circuits composed of multiple-input translinear element (MITE) networks to minimize power dissipation. MITEs were introduced as a circuit primitive in [88] and a brief description is provided in Section 1.3.2. Digital implementation of the nonlinear functions use the COordinate Rotation DIgital Computer (CORDIC) algorithm [129]. In this chapter, we compare the power consumed at the weight evaluation stage of the proposed analog realization with that of a digital ASIC realization. We also address the effect of using an analog-to-digital converter (ADC) and a digital-to-analog converter (DAC) in the mixed-mode implementation. We also compare the mixed-mode implementation to a digital realization based on power consumption, speed of operation, and chip area. Using simulations, we compare the tracking performance of the bearings-only tracker that uses analog circuits to perform the weight evaluation with that of a tracker that uses digital (FPGA) components.

4.2 Particle Filter-based Tracking

4.2.1 Bearings-only tracking

Bearings-only tracking involves estimating the target trajectory using angle measurements at a sensor node. The target is assumed to move in the x-y plane and to follow a constant velocity motion model [7], with a state update period of 1 s. The state transition is described using the relation

$$\mathbf{X}_t = \mathbf{F}\mathbf{X}_{t-1} + \mathbf{\Gamma}\mathbf{u}_t, \tag{4.1}$$

where $\mathbf{X}_t = [x \quad v_x \quad y \quad v_y]_t^T$, $\mathbf{u}_t = [u_x \quad u_y]_t^T$,

$$\mathbf{F} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \text{ and } \mathbf{\Gamma} = \begin{pmatrix} 0.5 & 0 \\ 1 & 0 \\ 0 & 0.5 \\ 0 & 1 \end{pmatrix}. \tag{4.2}$$

Here, x and y are the Cartesian coordinates[1] of the target, v_x and v_y are the corresponding velocities. The parameter $\mathbf{u}_t$ represents the system noise and is Gaussian distributed with covariance $\mathbf{\Sigma}_u = \sigma_u^2 \mathbf{I}_2$, where $\mathbf{I}_2$ is a 2×2 identity matrix.

The angle measurements at a stationary sensor at the origin are given by

$$z_t = \arctan\left\{\frac{y_t}{x_t}\right\} + r_t, \tag{4.3}$$

where r_t represents a Gaussian measurement noise $\mathcal{N}(0, \sigma_r^2)$.

4.2.2 Particle filter

We use particle filters [46], [55] (Section 1.1 has a brief review) for solving the bearings-only tracking problem. Consider the state-space system with the state transition (4.1), where $\mathbf{X}_k$ is the state vector, and with the measurement equation (4.3) where z_t are the noisy measurements related to the state at time t. We are interested in estimating the state $\mathbf{X}_n$ at time n given the measurements z_t for $k = 1, \ldots, n$. The state transition

[1]The notation for x, y in this Chapter 4 differs from the notation in Chapters 1, 2, and 3 where x denotes the state and y the measurement.

density $p(\mathbf{X}_t|\mathbf{X}_{t-1})$ and data likelihood $p(z_t|\mathbf{X}_t)$ can be obtained from (4.1) and (4.3). A Bayesian solution for this filtering problem involves obtaining the posterior distribution

$$\pi_n = p(\mathbf{X}_n|z_n, \mathbf{X}_{n-1}) \propto p(z_n|\mathbf{X}_n)p(\mathbf{X}_n|\mathbf{X}_{n-1}). \tag{4.4}$$

Particle filters use a combination of importance sampling, weight update and resampling to sequentially obtain and update the distribution (4.4). Importance sampling approximates the probability distribution π using a set of N weighted particles $\{\mathbf{X}^{(i)}, w^{(i)}\}$, where $\mathbf{X}^{(i)}$ represents the state and $w^{(i)}$ the corresponding weight for the i-th particle. The weight update stage uses measurements to update the particle weights. The updated particles are used to make inferences on the state $\mathbf{X}_n$. The resampling stage avoids degeneracy by removing particles with low weights and replicating particles with high weights.

Using (4.1) and (4.3), a particle filter algorithm for target tracking similar to the one in [55] can be formulated. The state update is used to propose new particles. As mentioned in Section 1.1.3, this proposal function provides a sub-optimal recursive estimate of the target position in the x-y plane. The pseudocode for the particle filter algorithm is shown in Table 4.1 and a block diagram of the computational flow is shown in Figure 4.1.

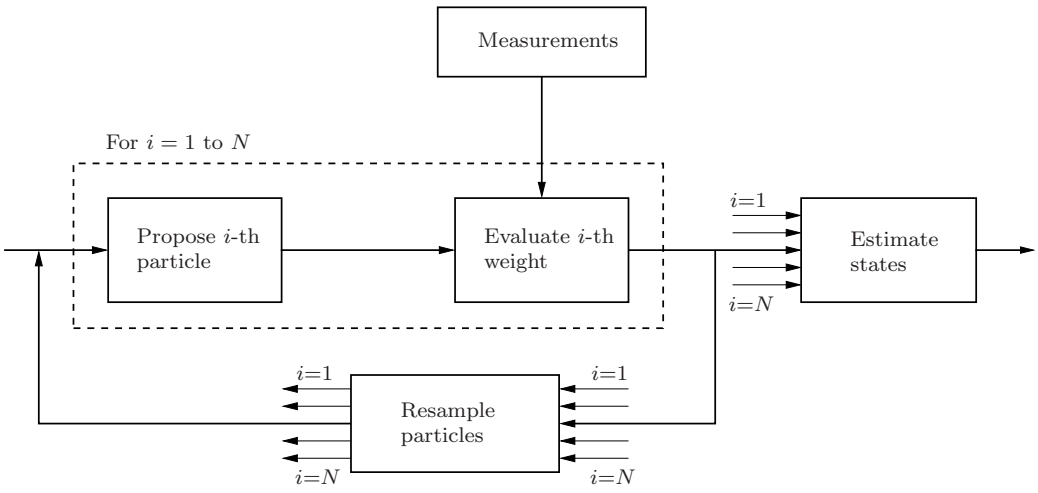


Figure 4.1: Block diagram showing computational flow in a generic particle filter algorithm.

4.3 Particle Filter Implementation

Particle filter algorithms use nonlinear functions and operate on scalar or vector data, as opposed to blocks of data as in speech or image processing. Their complexity is related

Table 4.1: Bearings-only tracker particle filter pseudo-code.

Given the observed data z_k at k,

1. For $i = 1, 2, \ldots, N$ sample or propose particles using the state update (4.1), $\mathbf{X}_t^{(i)} \sim p(\mathbf{X}_t^{(i)} | \mathbf{X}_{t-1}^{(i)})$.

2. For $i = 1, 2, \ldots, N$ calculate the weights,

$$\hat{w}_t^{(i)} = w_{t-1}^{(i)} p(z_t | \mathbf{X}_t^{(i)}), \tag{4.5}$$

   where $p(z_t | \mathbf{X}_t^{(i)})$ is the observation density, given by

$$\frac{1}{\sqrt{2\pi {\sigma_r}^2}} \exp\left\{ - \frac{\left(z_t - \arctan \frac{y_t^{(i)}}{x_t^{(i)}} \right)^2}{2{\sigma_r}^2} \right\}. \tag{4.6}$$

3. Normalize the weights $w_t^{(i)}$ using $\hat{w}_t^{(i)}$.

4. Calculate the state estimates, $E\{\mathbf{X}_t\} = \sum_{i=1}^{N} w_t^{(i)} \mathbf{x}_t^{(i)}$.

5. Resample $\left\{ \mathbf{X}_t^{(i)}, w_t^{(i)} \right\}$ to obtain new set of particles $\left\{ \mathbf{X}_t^{(j)}, w_t^{(j)} = \frac{1}{N} \right\}$.

---

to the number of particles N, the proposal function used, and the nonlinear functions in the model. Most applications that use particle filters perform a Gaussian evaluation at the weighting stage. Depending on the state-space model used, some applications might use additional nonlinear functions in the particle proposal and weight evaluation stage. An example application is the batch measurement-based trackers in Chapters 2 and 3. Except for the state estimate and resampling stage, processing of individual particles can be done in parallel.

In this research, we are interested in reducing the power dissipated while performing computations. For an individual particle in the bearings-only tracker, the weight evaluation stage is computationally more complex when compared to the proposal, state estimation, or resampling stage. So we concentrate on the computations for a single particle in the weight evaluation stage of the bearings-only tracker. As shown in Figure 4.2, the computations include simple operations such as addition and multiplication, as well as nonlinear operations

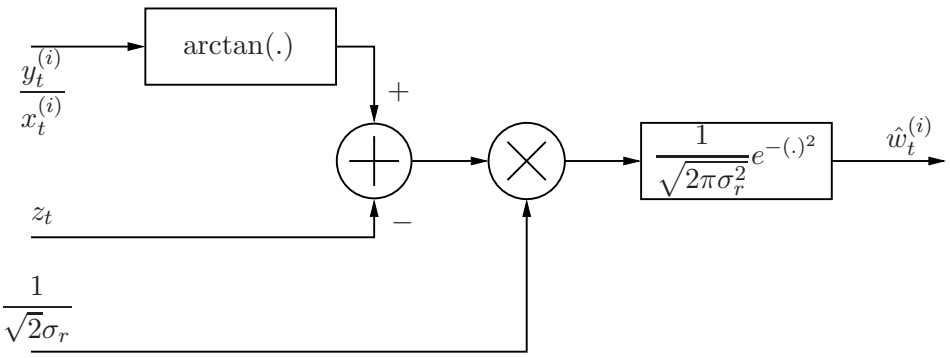


Figure 4.2: Computations at the weight evaluation stage of the particle filter algorithm used in bearings-only tracking.

such as *Gaussian* and *arctan* evaluation. However, we note that, as also mentioned in [17], the resampling stage is still one of the critical stages when considering the particle filter algorithm as a whole.

In this section, we first develop a mixed-mode implementation of the bearings-only target tracker. We then present a digital implementation using an FPGA. This digital implementation is similar to that developed in [17] but without any parallelization.

4.3.1 Mixed-mode implementation

In the mixed-mode implementation of the particle filter algorithm, certain stages use analog components to perform the computations while the remaining stages use digital components. This leads to two possible methods that differ in their analog-digital partition.

In Method-1, shown in Figure 4.3, the weight evaluation stage is implemented in the analog domain and the remaining stages in the digital domain. Data converter blocks, DAC and ADC, are used to transfer data across the domains. The minimum number of bits to be used in the DAC and ADC is dictated by state-space requirements. The lowest value to be represented must be above the noise level in the analog circuit. Increasing the number of bits in these blocks can increase latency and power consumption. Hence a compromise among accuracy, speed, and power consumption has to be made such that the power savings from the analog computations is not offset.

In Method-2, shown in Figure 4.4, the resampling stage alone is implemented in the digital domain and the remaining stages in the analog domain. This method uses fewer

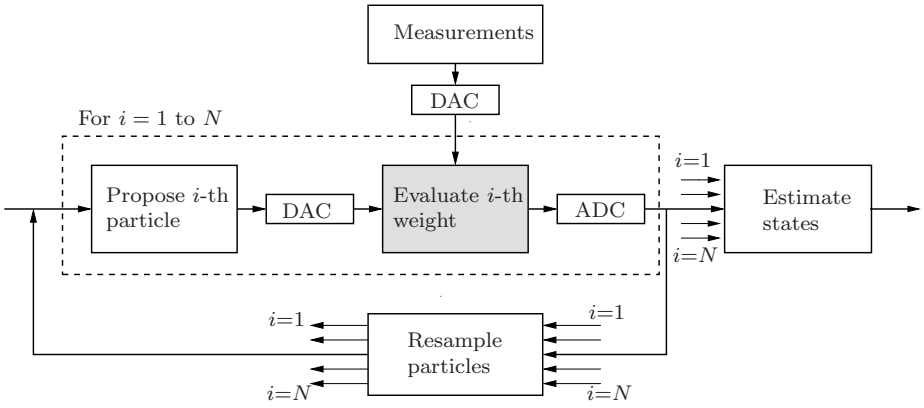


Figure 4.3: Block diagram showing computational flow of the particle filter algorithm in the mixed-mode implementation (Method-1). Highlighted stage is performed in the analog domain.

DACs and ADCs compared to Method-1. The measurements, which are angles in the bearings-only tracker, may be from a source localization algorithm implemented in the analog domain [112]. Performing the operations in the proposal and state estimation stages using analog methods can also provide significant power savings compared to Method-1. The proposal stage uses four additions to implement (4.1). In the analog domain, the addition of two signals is based on Kirchoff's current law (KCL) and is performed by connecting the wires carrying the corresponding currents. In the digital domain, addition is performed using adder circuits. An analog approach to generate Gaussian random noise, used in the proposal stage, is given in [2]. The state estimation, Step 4 in Table 4.1, can be achieved by using a vector-matrix multiplier in the analog domain [35]. The resampling stage assumes the availability of an analog memory whose access is controlled using a digital controller [56].

4.3.2 Nonlinear function realization using MITEs

The two-input MITE introduced in Section 1.3.2 is used to realize the nonlinear or transcendental functions in the weighting stage (4.6) as shown in Figure 4.2. The transcendental functions needed in the bearings-only tracking algorithm are the inverse tangent *arctan* and the *Gaussian* ($\exp(-x^2/2)$). The approximations and the corresponding implementations of these functions are considered here. The synthesis of MITE networks to realize these functions was performed by Shyam Subramanian [115] in the CADSP group at the Georgia

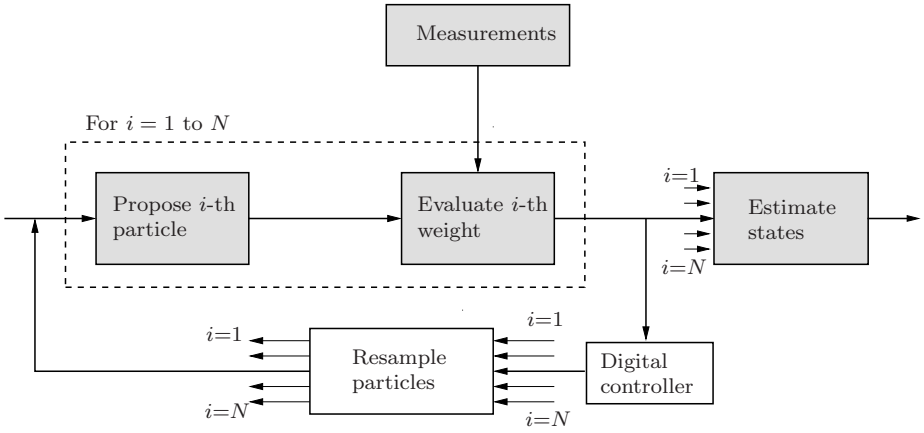


Figure 4.4: Block diagram showing computational flow of the particle filter algorithm in the mixed-mode implementation (Method-2). Highlighted stages are performed in the analog domain.

Institute of Technology.

The implementation of nonlinear functions using translinear circuits is discussed in [107]. Elementary operations like addition and subtraction are easily performed in any current-mode system (using KCL and a current mirror, respectively). Translinear circuits, in particular, can also perform other elementary operations like multiplication, division, and exponentiation (with rational exponents). Hence, any algebraic function can be synthesized by simply expressing it in terms of these elementary operations. Transcendental functions like $\exp(x), \log(x), \text{and } \arctan(x)$ are implemented by suitably approximating them using algebraic functions. Various techniques exist for approximating nonlinear functions by rational functions [51]. Approximation with minimax or near-minimax error is one of the more common methods for approximation over an interval. Remez's algorithm [51] is used to determine the minimax rational approximation, while numerous other techniques exist to get near-minimax approximations. For example, Maple's 'minimax' command implements Remez's algorithm. The approximations used for the inverse tangent *arctan* and the *Gaussian* $\exp(-x^2/2)$ functions are shown in Table 4.2 and the corresponding implementations are considered next.

Table 4.2: Approximations used in the analog implementation of nonlinear functions.

Function	**Approximation**	**Range**
$y = \frac{2}{\pi}\arctan(x)$	$f(x) = \frac{x}{0.63 + \sqrt{0.88 + x^2}}$	$\lvert x\rvert < \infty$
$y = c\exp\left(-\frac{x^2}{2a^2}\right)$	$f(x) = zc\frac{1 - n_1(r/a)}{1 - d_1(r/a) + d_2(r^2/a^2)}$ $z, c, n1, d1, d2$ are constants	$x \in [-4a, 4a] \Rightarrow r \in [2a, 18a]$

4.3.2.1 Implementation of the inverse tangent function

The function ϕ to be approximated is as follows (normalized so that $\phi(\infty) = 1$):

$$\phi(x) = \frac{2}{\pi}\arctan(x), \text{ where } |x| < \infty. \tag{4.7}$$

An approximation of ϕ using algebraic functions, given in [107], is as follows:

$$y = f(x) = \frac{x}{0.63 + \sqrt{0.88 + x^2}}, \text{ where } |x| < \infty. \tag{4.8}$$

The maximum error obtained using the approximation (4.8) is less than 0.05% of the maximum value.

The implementation of f in (4.8) using MITEs is done through the following steps:

1. **Scaling.** Since the input and output variables are represented by currents, to maintain dimensional consistency, the substitutions $x \mapsto I_x/I_a$ and $y \mapsto I_y/I_a$ are done. Hence, we have
$$I_y = \frac{I_x I_a}{0.63 I_a + \sqrt{0.88 I_a^2 + I_x^2}} \tag{4.9}$$
where I_x is the input current, I_y the output current, and I_a the reference current.

2. **Current splitting.** Since the input x can take both positive and negative values and since the currents through MITEs must necessarily be positive, we use a current splitter [52] to produce currents I_{x+} and I_{x-} satisfying $I_{x+} - I_{x-} = I_x$ and $I_{x+} I_{x-} = I_a^2$.

3. **Block reduction.** The equation to be implemented thus becomes

$$I_y = \frac{I_{x+}I_a - I_{x-}I_a}{0.63I_a + \sqrt{-1.12I_a^2 + I_{x+}^2 + I_{x-}^2}}$$
$$= \left(\frac{I_{x+}I_a}{\left(0.63I_a + \left(\sqrt{I_r I_a}\right)\right)}\right) - \left(\frac{I_{x-}I_a}{\left(0.63I_a + \left(\sqrt{I_r I_a}\right)\right)}\right) \quad (4.10)$$
$$\text{where } I_r = \frac{I_{x+}^2}{I_a} + \frac{I_{x-}^2}{I_a} - 1.12I_a.$$

The parentheses show the order in which the operations are implemented. Each of the blocks (representing the operation in the parentheses) is implemented using procedures described in [86].

4. **Consolidation.** As described in [87], reduntant MITEs are removed using *consolidation* to arrive at the final circuit shown in Figure 4.5(a).

4.3.2.2 Implementation of the Gaussian

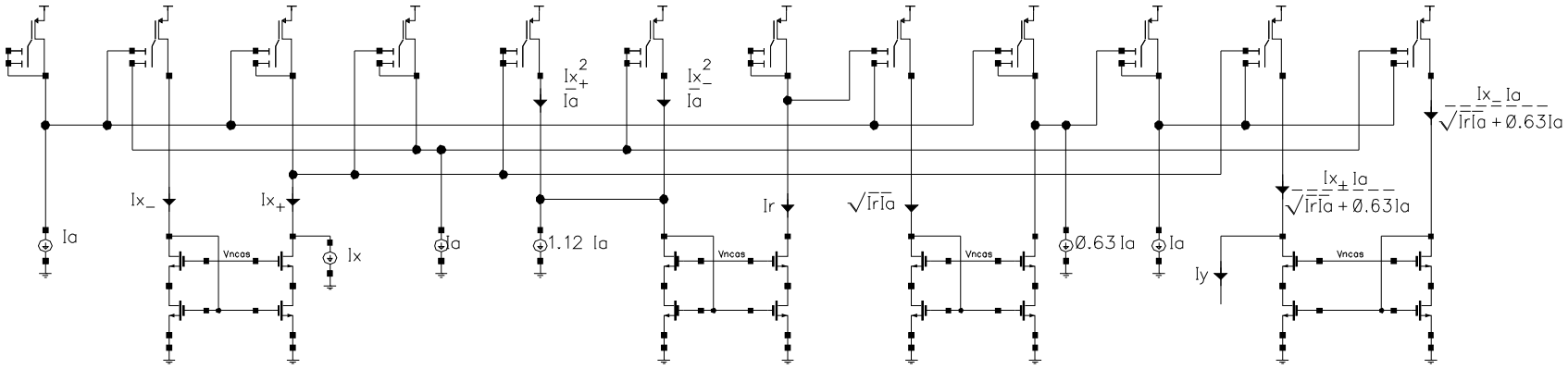
After scaling and normalization, the Gaussian is transformed into

$$I_y = I_c \exp\left(-\frac{I_x^2}{2I_a^2}\right), \quad (4.11)$$

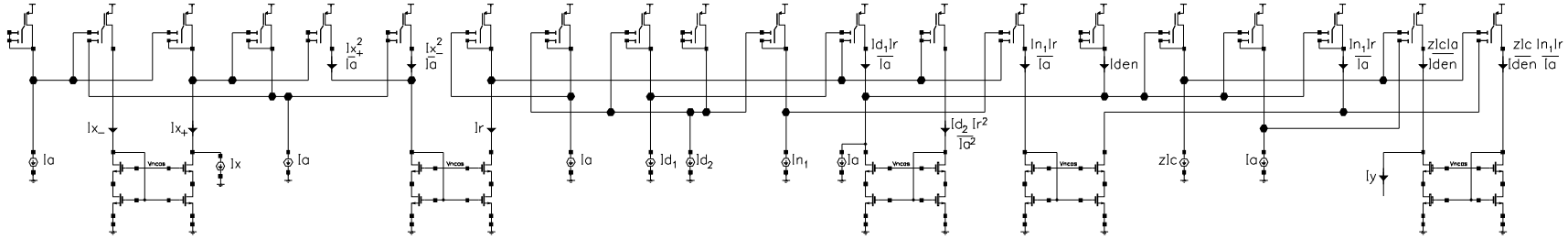
where I_y represents the output current, I_x the input current, I_a the reference current, and I_c the scaling current. A current splitter converts I_x into two positive currents I_{x+} and I_{x-} satisfying $I_{x+} - I_{x-} = I_x$ and $I_{x+}I_{x-} = I_a^2$. Hence, we have $I_x^2 = I_{x+}^2 + I_{x-}^2 - 2I_a^2 = I_r I_a - 2I_a^2$, where $I_r = I_{x+}^2/I_a + I_{x-}^2/I_a$. Thus, $I_y = eI_c \exp(-I_r/(2I_a))$. It should also be noted that if the implementation is to be valid for $I_x \in [-bI_a, bI_a]$, then it suffices to approximate $\exp(-I_r/(2I_a))$ over the interval $I_r \in [2I_a, (2+b^2)I_a]$. The minimax rational approximation for $b = 4$ with numerator and denominator degrees equal to 1 and 2, respectively, was found using Remez's algorithm and is given by :

$$I_y = zI_c \frac{I_a - I_{n1}(I_r/I_a)}{I_a - I_{d1}(I_r/I_a) + I_{d2}(I_r^2/I_a^2)} \quad (4.12)$$

where $I_r = (I_{x+}^2 + I_{x-}^2)/I_a$, $I_{n1} = 0.07195I_a$, $I_{d1} = 0.2913I_a$, $I_{d2} = 0.1641I_a$, and $z = 1.245$. The computations that involve multiplication are done using a new synthesis procedure especially meant for such "product–of–power–law" computations [115]. The currents I_{n1}, I_{d1}, I_{d2} are set using programmable floating–gate MOSFETs [108]. The final circuit is shown in Figure 4.5(b).



(a) The MITE circuit used to implement the arctan function.



(b) The MITE circuit used to implement the Gaussian function.

Figure 4.5: (a) The MITE circuit used to implement the *arctan* function. The current $I_r = \frac{I_{x+}^2}{I_a} + \frac{I_{x-}^2}{I_a} - 1.12I_a$ and the output is given by $I_y = \frac{(I_{x+}I_a - I_{x-}I_a)}{(\sqrt{I_a I_r} + 0.63I_a)}$. (b) The MITE circuit used to implement the *Gaussian* function. The intermediate variables used are given by $I_r = I_{x+}^2/I_a + I_{x-}^2/I_a$, and $I_{den} = I_a - (I_{d1}I_r)/I_a + (I_{d2}I_r^2)/I_a^2$. The output is $I_y = zI_c(I_a - (I_{n1}I_r)/I_a)/I_{den}$.

4.3.3 FPGA implementation of particle filter

Having looked at a mixed-mode implementation of the bearings-only tracker, we present a fully digital FPGA implementation of the tracker in this section. The main motivations for the FPGA implementation are (i) to obtain qualitative and quantitative comparison of the two implementations (ii) to compare the FPGA implementation of the bearings-only particle filter tracker to the batch-based tracker in Chapter 5. We use the Xilinx System Generator for DSP [136] to obtain an FPGA implementation for the bearings-only tracker. System Generator is a high-level software tool to generate VHDL code for Xilnx FPGAs from Simulink models or Matlab code. Otherwise, the FPGA implementation is similar to that in [17], [5], but without any parallelization.

Most components in the bearings-only particle filter tracker are built from the basic blocks provided in the System Generator library. Certain functions, such as the *arctan*, *exponential*, *ceil*, and *Gaussian* are approximated or implemented by modifying blocks in the standard library. The *arctan* and *Gaussian* functions are implemented using the CORDIC block. Each of the individual blocks have delays that need to be handled by using appropriate delay blocks while integrating them into the tracker.

Another important aspect of the digital FPGA implementation is the choice of word length. In our case, we restrict the target states to be in the first-quadrant, i.e., positive values for both x and y positions, and bearing measurements from 0° to 90°. With these assumptions, the word length is chosen to be 17 bits with a fractional part of 15 bits. One rationale behind this choice is to make a fair comparison with an earlier implementation [17], [4].

Though individual stages were integrated to obtain the tracker, a complete tracking system was not realized. The implementation lacks a higher level module to recursively propagate the particles and update the state estimates over time, because this would involve developing a top-level VHDL module to control the memory reads and writes. The System Generator setup is not well suited to design this control operation, so it is not pursued here. The implementation presented here performs the computations for N particles in a single iteration of the particle filter. The individual blocks in the particle filter implementation

shown in Figure 4.6 are described next. The System Generator model used to realize the FPGA implementation is shown in Figure 4.7.

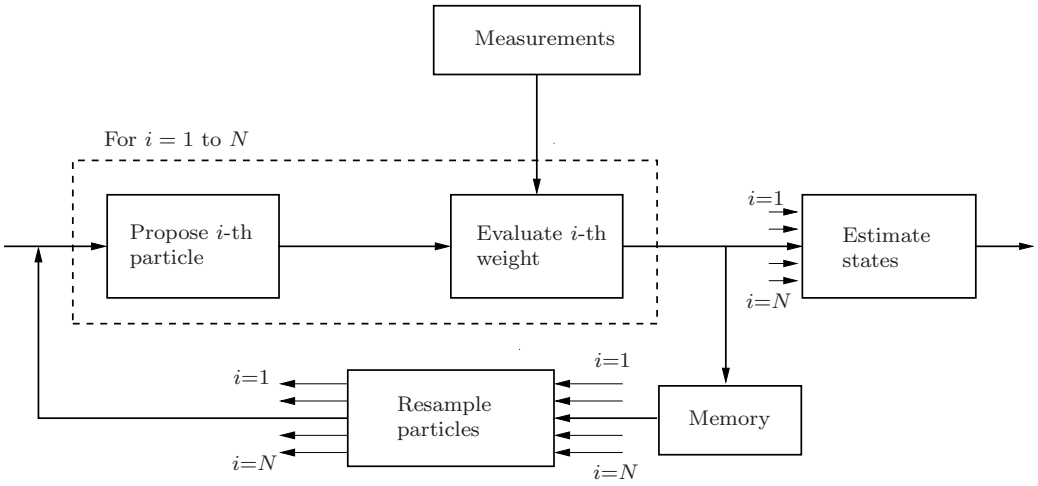


Figure 4.6: Particle filter flow in a digital or FPGA implementation

The proposal stage in the bearings-only tracker performs a state update as in (4.1). From an implementation perspective, the significant operation in this stage is the additive white Gaussian noise (AWGN) generator. We use the AWGN block from the System Generator library, which is based on the Box-Mueller algorithm. Other operations are mostly additions and subtractions, and the block diagram of this stage is shown in Figure 4.8.

The weight evaluation stage has most of the computations in the bearings-only tracker. The particle weights are evaluated using the data likelihood expression in (4.6). The System Generator model of this stage is shown in Figure 4.9. It consists of two significant operations, i.e., evaluating the *arctan* and *Gaussian* functions. The *arctan* function is implemented by modifying the CORDIC block provided in the System Generator library. The *Gaussian* evaluation is performed using an exponential. This exponential is implemented as [17]:

$$\exp(x) = \sinh(x) + \cosh(x),$$

where the hyperbolic functions are implemented by configuring the CORDIC block [90]. Since the range of values that can be input to a CORDIC block is restricted, the exponential has to be further approximated as

$$\exp(x) = \exp(x_{\text{integer}}) + \exp(x_{\text{fractional}}).$$

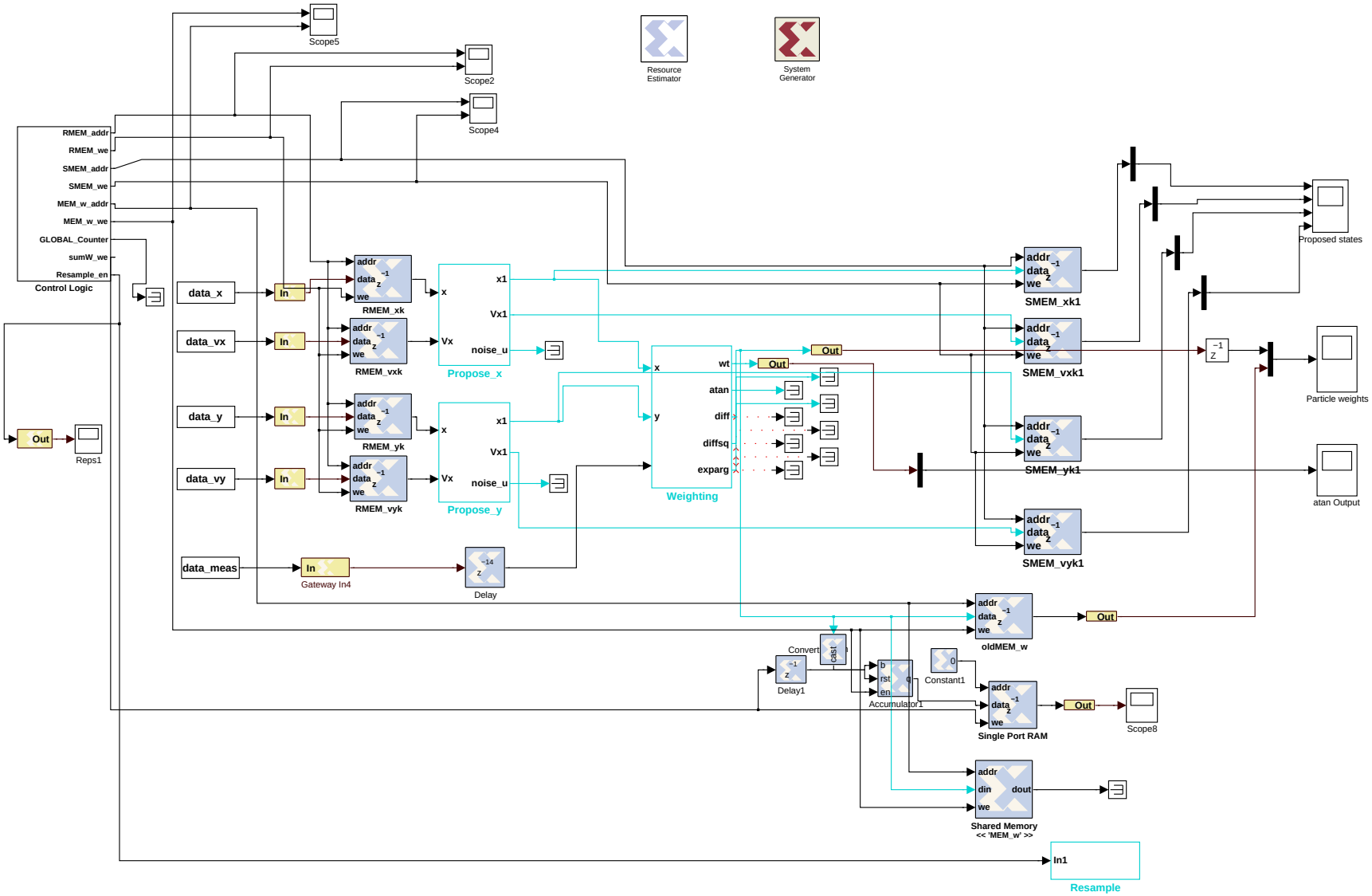


Figure 4.7: System Generator model of the bearings-only particle filter tracker. The blocks represent HDL functionality in Xilinx FPGAs. The implementation is for evaluating particle weights during one time step. The high-level architecture is similar to the approach in [5].

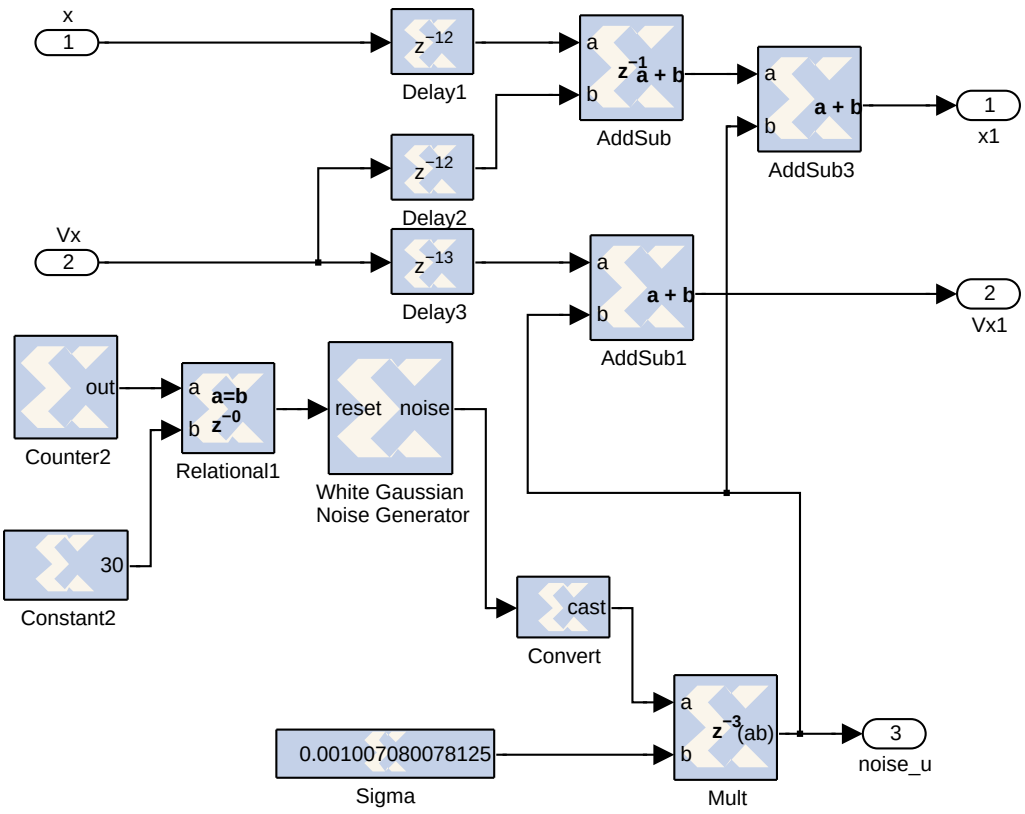


Figure 4.8: System Generator model of the proposal stage in the bearings-only tracker. This corresponds to the "Propose_x" block in Figure 4.7. The "Propose_y" block uses a similar model with inputs y and V_y.

The CORDIC block in these functions introduces significant delay when compared to other blocks in the implementation. This delay depends on the word length of the input data. An alternative to using the CORDIC blocks is to use a look-up table to evaluate these nonlinear functions. However, the memory requirements for such an approach increases with the required accuracy.

The weight normalization stage is closely associated with the weighting stage, because the sum of the particle weights affects the word length. Normalization involves division which also requires a CORDIC block in the FPGA implementation.

The resampling stage is an important step in the particle filter algorithm and the most time consuming operation in the algorithm. Several efficient ways for resampling have been proposed [14] using a distributed architecture. However, in this thesis we consider a simple approach and implement a modified residual systematic resampling algorithm [5] as shown

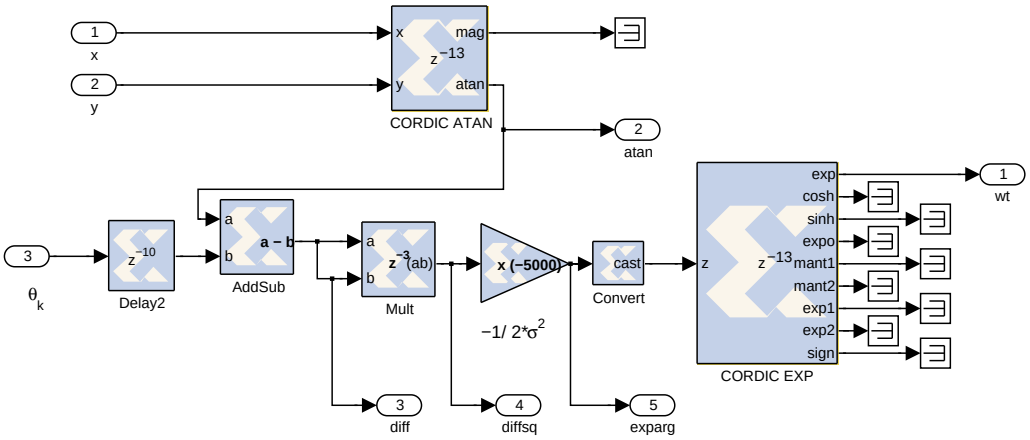


Figure 4.9: System Generator model of the weight evaluation stage in the bearings-only tracker. This corresponds to the "Weighting" block in Figure 4.7.

in Table 4.3. The FPGA implementation involves logic operations and control circuits to move particles among memory locations. The strategy used in resampling is to compute the number of times a particle has to be replicated based on its weight. The implementation has one random number generation. The use of K in Step 2 of the pseudocode in Table 4.3 allows for non-normalized weights to be used and hence reduces the number of divide operations, otherwise required to normalize the weights.

The proposal and weighting stages can be performed sequentially, whereas the weight normalization and resampling cannot start until all the particle weights have been evaluated. Hence, the resampling stage determines the speed of operation, or sampling rate, of the particle filter algorithm, i.e., the rate at which the estimates can be produced. Any improvement in the resampling stage can provide significant speed up, especially if the proposal and weighting stages can be parallelized. But, resampling is not the focus of this research, because it has been analyzed already and an ASIC implementation for it was presented in [60].

The various stages are implemented using the System Generator tool and targeted to the Xilinx Virtex II Pro FPGA device [135]. The device details are shown in Table 4.4. Based on this implementation, the estimated resource utilization by each stage for a single particle evaluation is shown in Table 4.5. Nearly 15% of the resources in the FPGA device are utilized. This resource utilization does not represent an optimal choice and has room for

Table 4.3: Residual Systematic Resampling adapted from [5].

1: $U \sim \mathcal{U}[0,1]$ {*Draw* a random variable from an uniform distribution}
2: $K = N/W_n$ {W_n is the sum of non-normalized weights}
3: $ind_r = 0, ind_d = M - 1$
4: **for** i=1 to N **do**
5: $temp = w_n^{(i)}.K - U$
6: $r^{ind_r} = \lceil temp \rceil$ {r is the # a particle is replicated}
7: $U = temp - r^{ind_r}$
8: **if** $r^{ind_r} > 0$ **then**
9: $i_r^{(ind_r)} = i$, $ind_r = ind_r + 1$
10: **else**
11: $i_d^{(ind_d)} = i$, $ind_d = ind_d - 1$
12: **end if**
13: **end for**

improvement. For example, the single-port block RAMs used in between the various stages can be replaced by dual-port block RAMs [5]. The RMEM and SMEM blocks in Figure 4.7 are single-port block RAMs. But, using dual-port block RAMs requires a more sophisticated control strategy. Since the aim of this implementation is to perform comparisons, such an approach was not adopted. The proper operation of the particle filtering algorithm also relies on accurately estimating the delay at each stage of the algorithms. The delay, in clock cycles, is shown in Table 4.6. Based on a device clock frequency of 100 MHz and the latencies shown in Table 4.6, the sampling rate of the particle filter algorithm is nearly 33 kHz.

Table 4.4: Xilinx Virtex II Pro FPGA - XC2VP30 - Device details.

Resource	# Logic cells[†a]	# Slices[†b]	Block RAM	Clock frequency
	30816	13696	2448 kb	100 MHz

[†a] Logic cell ≈ one 4-input LUT + one Flip-Flop + Carry logic
[†b] Each slice includes two 4-input function generators, carry logic, arithmetic logic gates, wide function multiplexers, and two storage elements.

Table 4.5: FPGA resource utilization for bearings-only tracker.

Resource	**PF Stage**			
	Proposal	**Importance**	**Resampling**	**Overall**
# Slices	2700	1215	374	4635
# Flip-flops	2500	892	373	4060
# Block RAMs	0	0	0	0
# LUTs	3500	2209	653	7065
# Emb. Mult.	8	0	0	8

Table 4.6: Time delay at various stages of bearings-only tracker.

	PF Stage			
	Proposal	**Importance**	**Resampling**	**Overall**
Latency (Clock cycles)	13	32	$\mathbf{N+5}$	$\mathbf{3N}+50$

4.4 Simulations

This section presents simulation results to demonstrate the functional accuracy of the weighting stage implemented using MITEs in the analog domain and using an FPGA in the digital domain. In Section 4.5, the weighting stage, implemented as a MITE circuit or an FPGA block, is used in a Matlab simulation of the bearings-only tracker. This is achieved using Matlab scripts that execute a call to an analog (Spectre [19]) or a digital (Modelsim [83]) circuit simulator tool as part of a Matlab simulation.

4.4.1 Mixed-mode implementation

Our mixed-mode simulations use Method-1 (Figure 4.3) for the bearings-only tracker. The analog implementation of the *arctan* and *Gaussian* functions use MITEs as explained in Section 4.3.1. The circuit blocks are simulated for various values of the input currents and the reference currents to determine the accuracy of the implementation. Models for the AMI 0.5 μm CMOS process were used in the simulations. The results for the standalone functions are shown in Table 4.7. The error (as a percentage of the maximum) and power values correspond to the ranges of the input and reference currents shown in Table 4.7. The reference currents must be chosen such that the MITE networks remain in their valid region of operation.

The analog circuit for the weighting stage of the bearings-only tracker is simulated and

Table 4.7: Characteristics of the MITE implementation - *arctan* and *Gaussian* functions.

Circuit	*arctan*		*Gaussian*	
	Minimum	Maximum	Minimum	Maximum
Ref. I_a (nA)	0.5	20	0.1	10
Input current	$-10I_a$	$10I_a$	$-10I_a$	$10I_a$
Power (μW)	0.361	14.45	1.097	109.7
Error (%)	0.31	0.71	2.04	2.8

Table 4.8: Simulation parameters - Mixed-mode bearings-only tracker.

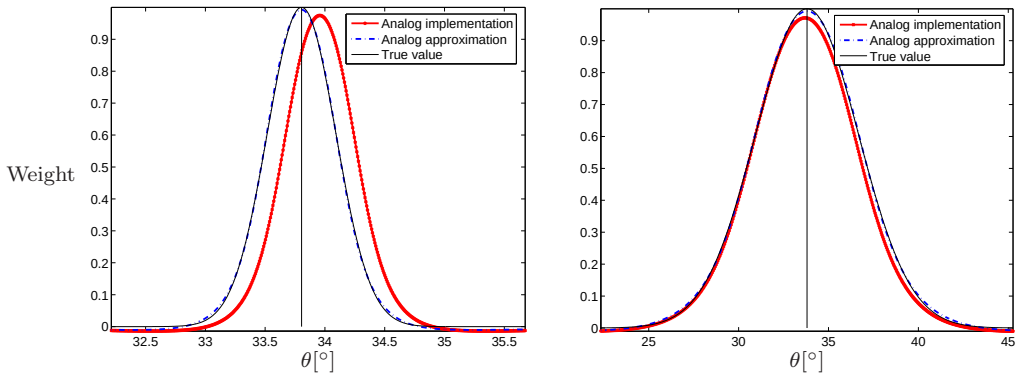
Tracker parameters			**Circuit parameters** (variables in (4.6))		
N	σ_r	σ_u	$I_{\text{in}}(y_k^{(i)}/x_k^{(i)})$	Ia_{arctan}	$Ia_{\text{Gaussian}}(\sigma_r)$
1000	0.29°	0.001	6.36nA to 7.18nA	10nA	0.1nA

the results compared to a Matlab simulation. The range for the input currents is obtained from the Matlab simulation. The simulation parameters used are similar to those in [55] and are shown in Table 4.8. Figure 4.10 compares the weights obtained from the *Analog implementation* to the *True value* obtained using (4.5) and the *Analog approximation* values obtained using (4.10) and (4.12), for two cases. The error introduced by the *arctan* block shifts the mean of the *Gaussian* distribution in the weight evaluation.

Depending on the measurement noise variance σ_r^2, the shift in the Gaussian may affect the state estimate. If the variance σ_r^2 is comparable to the error in the MITE implementation of the *arctan* circuit, the error in the computation will lead to biased estimates. The influence of σ_r on the output of the weighting stage is shown in Figure 4.10 for two values that differ by a factor of ten. In Section 4.5.1 we present the corresponding effect in a bearings-only tracking scenario. Another issue is that the implemented *arctan* block has quadrant ambiguity and hence additional logic needs to be included to put the results in the correct quadrant.

4.4.2 Digital implementation

The digital or FPGA implementation for the weighting stage of the bearings-only tracker is simulated and the results have been compared to simulink simulation outputs. The range for the input currents is obtained from Matlab simulations. The simulation parameters



(a) $\sigma_r = 0.29°$ (b) $\sigma_r = 2.8°$

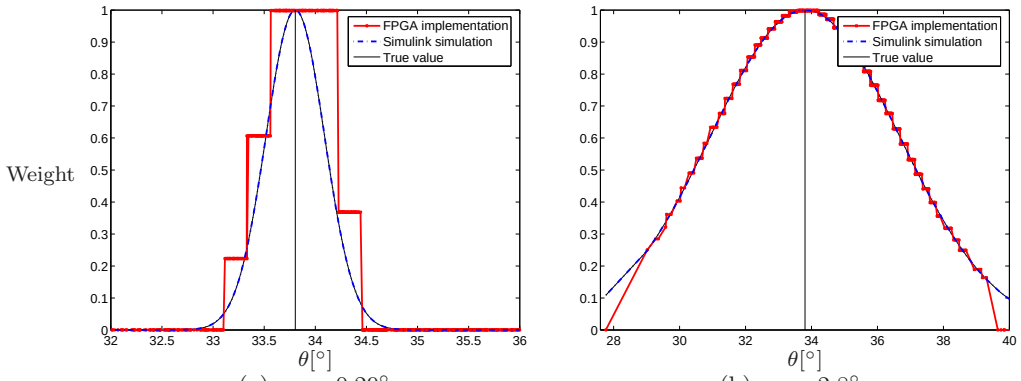
Figure 4.10: Comparison of the evaluated weights ($\hat{w}_t^{(i)}$ in Figure 4.2) using the analog (MITE) implementation for two values of noise variance. The straight line at 33.8° represents the true mean. (a) The standard deviation $\sigma_r = 0.29°$ is of the same order as the error in *arctan* function, leading to a shift in the *Gaussian* mean. (b) The standard deviation $\sigma_r = 2.8°$ is much larger than the error in the *arctan* function.

used are similar to those in the analog simulation and are shown in Table 4.8. The plot in Figure 4.11 compares the weights obtained from the *FPGA implementation* to results from a *Simulink simulation* and to the *True value* obtained using Matlab. The error introduced by the *arctan* block is low compared to that in the analog implementation. Hence, there is no shift in the mean of the *Gaussian* distribution in the weight evaluation.

Depending on the measurement noise variance σ_r, the shift in the Gaussian may or may not affect the state estimate. If this variance is comparable to the error in the FPGA implementation of the *arctan* circuit, the error in the computation will lead to biased estimates. The influence of small or large σ_r on the output of the weighting stage is shown in Figure 4.11. In Section 4.5.2 we present the corresponding effect in a bearings-only tracking scenario.

4.5 Simulations: Bearings-only Tracking

In this section, the weighting stage implemented as a MITE circuit, or an FPGA block, is used in a Matlab simulation of the bearings-only tracker. This is achieved using Matlab scripts that execute a call to an analog (Spectre [19]) or a digital (Modelsim [83]) circuit simulator tool as part of a Matlab simulation. The results shown here are for a simple bearings-only tracker where the measurements are target DOAs at individual time instants.



(a) $\sigma_r = 0.29°$ (b) $\sigma_r = 2.8°$

Figure 4.11: Comparison of the evaluated weights ($\hat{w}_t^{(i)}$ in Figure 4.2) using the digital (FPGA) implementation for two values of noise variance. The straight line at 33.8° represents the true mean. (a) The standard deviation $\sigma_r = 0.29°$. (b) The standard deviation $\sigma_r = 2.8°$. Since, the error in FPGA implementation of the *arctan* block is significantly smaller than either of the standard deviations $\sigma_r = 2.8°$ and $\sigma_r = 0.29°$, the *Gaussian* mean is aligned with the true mean.

The tracking scenario considered here and the parameters used are similar to those in [55] and are shown in Table 4.8. A single target trajectory and measurements that follow (4.1) and (4.3) is simulated for two different values of measurement noise. The sensor is located at the origin in the x-y plane. The target's actual initial state is $\mathbf{X} = [0.250 \quad 0.001 \quad 0.240 \quad -0.005]^T$, the system noise is $\mathcal{N}(0, 0.001^2)$, and the measurement noise is either $\mathcal{N}(0, 0.29^2)$ or $\mathcal{N}(0, 2.8^2)$. The true target trajectory in the x-y plane is shown in Figure 4.12.

4.5.1 Tracking using the mixed-mode implementation

Tracking estimates from the *Analog implementation* are shown in Figures 4.13, 4.14, and 4.15. The error introduced when σ_r is small can lead to divergence of the estimates with time. Although, particle filters can mitigate this error because they use particles spread over a certain region, the error in computation might dominate and cause the particle distribution to be biased. However, if the noise variance is high compared to the error in the *arctan* computation, the estimates will be close to the ones obtained from a Matlab simulation.

During each iteration, inputs were presented as a data file of $\frac{y_t}{x_t}$ quotient values and

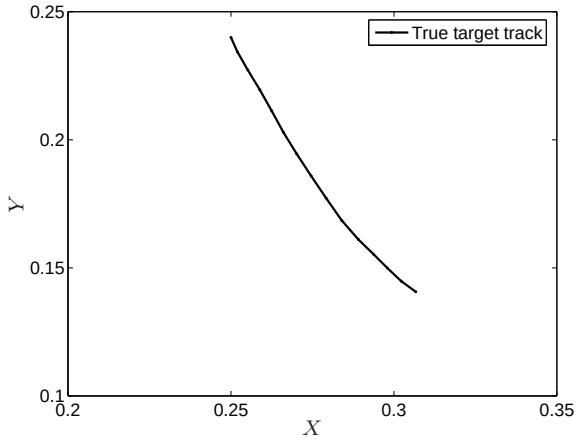


Figure 4.12: The true trajectory, in the x-y plane, of the target being tracked. The sensor (not shown in the figure) is located at the origin.

bearing measurement z_t. x_t and y_t represent the x and y components of the particle state vector $\mathbf{X}$, respectively. A Spectre simulation script uses these values to compute the particle weights $\hat{w}_t^{(i)}$ using the analog MITE circuit implementation of the weighting stage. These weights are used to continue the Matlab simulation.

The error in the *arctan* computation can vary from 0.18° to 0.41°. In Figures 4.13 (a), 4.14 (a), and 4.15 (a) the results correspond to a standard deviation of 0.29°. This value of the standard deviation is comparable to the error in the *arctan* computation. This results in a small bias in the x-y estimates, because of the shift in the mean of the *Gaussian* output, or weights, which was shown in Figure 4.10 (a). Hence, the resulting state estimates (*Analog implementation*) do not closely match the *Matlab estimate*. However, in spite of the bias they closely follow the *Ground truth* values. This might be an effect of the bias pulling the estimate towards the ground truth. The results in Figures 4.13 (b), 4.14 (b), and 4.15 (b) correspond to a standard deviation of 2.80°. This noise value is high compared to the error values in the *arctan* computation and the estimated weights are not biased as shown in Figure 4.10 (b). Hence, the corresponding estimates (*Analog implementation*) closely follow the *Matlab estimate*. In Figure 4.15, the difference between the *Ground truth* and the *Matlab estimate* is because of the sensitivity of the bearings-only tracker to the small state noise on the velocity components and the distance of the target from the sensor location.

The error in the magnitude of the analog *Gaussian* output does not affect the estimates. Since the computed weights $\hat{w}_t^{(i)}$ are normalized to obtain $w_t^{(i)}$, the individual difference in weights does not affect the final estimates significantly. However, the MITE reference currents have to be selected appropriately to prevent biased values from the *Gaussian* computation.

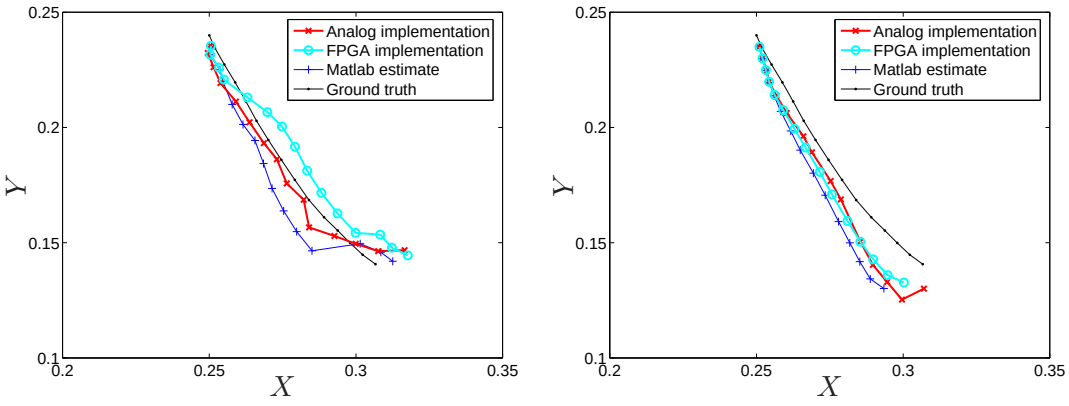
4.5.2 Tracking using the FPGA implementation

As mentioned earlier, the complete particle filter algorithm was not implemented in an FPGA. However, the FPGA implementation of the weighting stage was simulated using a VHDL implementation of this stage along with a Matlab implementation. This was achieved by realizing the weighting stage as a simulation file in Modelsim. This file was then used as part of the Matlab simulation to compute weights. The measurements and particle states were input to this simulation file and the outputs were fed back to the Matlab simulation.

The true trajectory of the target being tracked is shown in Figure 4.12 and the scenario is described in the beginning of Section 4.5. The target state estimates using the *FPGA implementation* were compared to the *Analog implementation* and to the *Matlab estimate*; the results for two different values of measurement noise are shown in Figures 4.13, 4.14, and 4.15. The tracking estimates using the *FPGA implementation* closely follow the *Matlab estimate*. This is expected, because the evaluation of weights using the FPGA implementation does not introduce any bias in the weights as shown in Figure 4.11 and as discussed in Section 4.4.2. In Figure 4.15, the difference between the *Ground truth* and the *Matlab estimate* is because of the sensitivity of the bearings-only tracker to the small state noise on the velocity components and the distance of the target from the sensor location.

4.6 Comparison of the Mixed-mode Implementation to a Digital Implementation

In this section, we compare the mixed-mode implementation to a digital implementation. To do a fair comparison, the digital implementation corresponds to an ASIC implementation, not the FPGA implementation.



(a) $\sigma_r = 0.29°$ (b) $\sigma_r = 2.8°$

Figure 4.13: Comparison of x-y track estimates from the analog (MITE) implementation to the digital (FPGA) implementation and the Matlab simulation. (a) Results for a noise standard deviation of $\sigma_r = 0.29°$, which is comparable to the error in the analog *arctan* computation. (b) Results for a noise standard deviation of $\sigma_r = 2.8°$, which is high compared to the error in the analog *arctan* computation.

4.6.1 Power dissipation

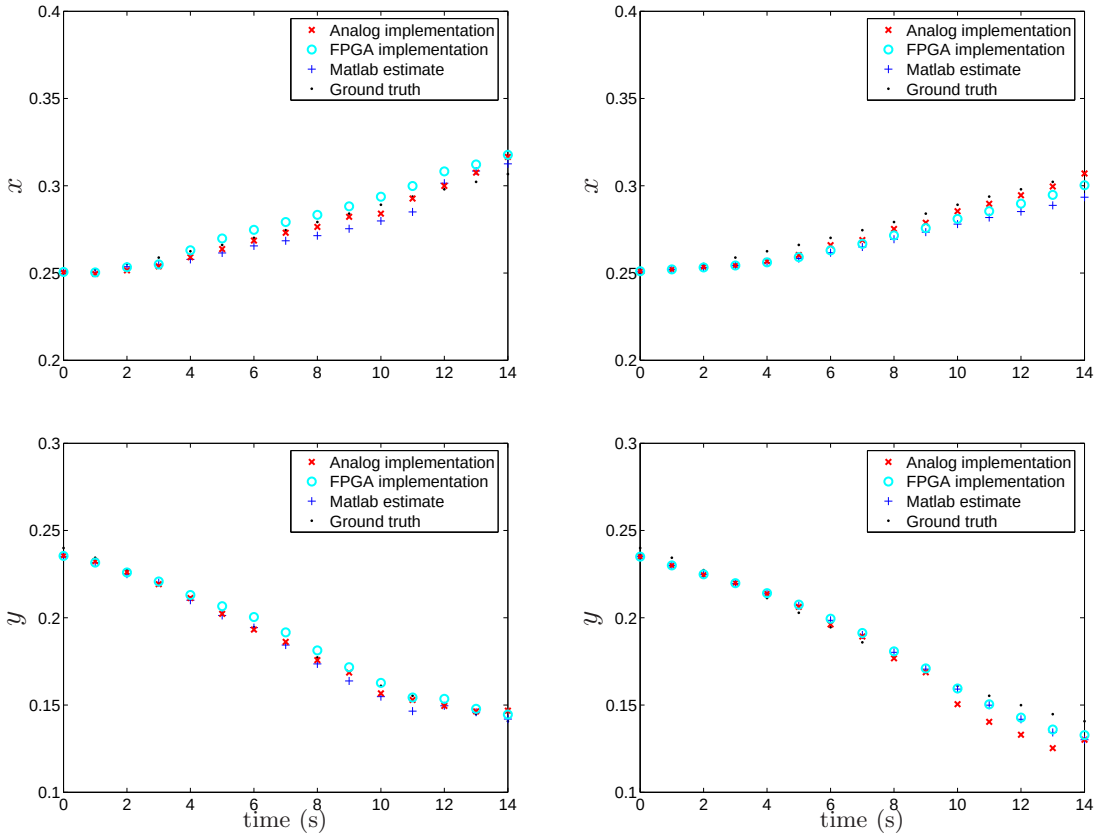
From (4.5) and (4.6), note that the significant computations in the particle weighting stage of the particle filter algorithm involve the evaluation of *arctan* and *Gaussian* functions. Hence, the power consumed in these computations is considered here.

4.6.1.1 Analog MITE implementation

The analog implementations of the *arctan* and *Gaussian* functions use MITEs as explained in Section 4.3.2. The instantaneous power consumed by these circuits depends on the instantaneous value of the input current, I_x. For the *arctan* circuit, the current drawn from the supply can be shown to be

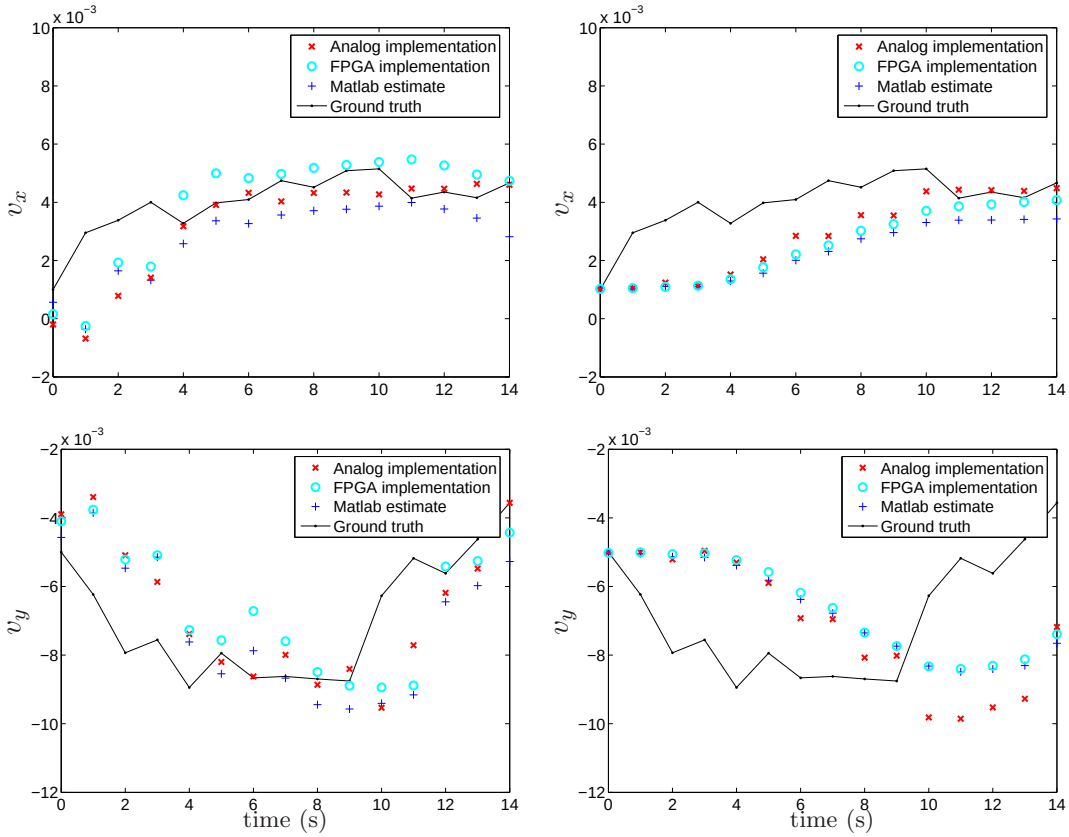
$$I_a \left(9.26 + \frac{2I_x^2}{I_a^2} + \sqrt{4 + \frac{I_x^2}{I_a^2}} + 2\sqrt{\frac{I_r}{I_a}} + \frac{\sqrt{4 + I_x^2/I_a^2}}{\sqrt{I_r/I_a} + .63} \right)$$

where I_a is the reference current and I_r is as shown in (4.10). Assuming $|I_x|_{\max} = 10I_a$, the maximum power dissipated is $240.82 I_a V_{dd}$. For a typical value of $I_a = 10$ nA with a 3 V supply, the maximum power consumed is 7.23 μW. A similar expression can be derived for the *Gaussian* block. Using the notations in (4.11) and setting the scaling current I_c to be equal to the reference current I_a, $I_c = I_a$, the maximum power consumed is $3655.14 I_a V_{dd}$. For $I_a = 0.1$ nA the power consumed is 1.097 μW. These values are shown in Table 4.7.



(a) $\sigma_r = 0.29°$ (b) $\sigma_r = 2.8°$

Figure 4.14: State estimation results for the x and y coordinates from the analog (MITE) implementation compared to the digital (FPGA) implementation and the Matlab simulation. (a) Results for a noise standard deviation of $\sigma_r = 0.29°$, which is comparable to the error in the analog *arctan* computation. (b) Results for a noise standard deviation of $\sigma_r = 2.8°$, which is high compared to the error in the analog *arctan* computation.



(a) $\sigma_r = 0.29°$ (b) $\sigma_r = 2.8°$

Figure 4.15: State estimation results for the x-y velocity components from the analog (MITE) implementation compared to the digital (FPGA) implementation and the Matlab simulation. (a) Results for a noise standard deviation of $\sigma_r = 0.29°$, which is comparable to the error in the analog arctan computation. (b) Results for a noise standard deviation of $\sigma_r = 2.8°$, which is high compared to the error in the analog arctan computation.

4.6.1.2 Digital implementation

As introduced in Section 1.3.1, power consumption in CMOS circuits consists of dynamic and static power (1.26). Ignoring the negligible static power, the average dynamic power dissipated can be approximated as [31]:

$$P = \alpha_{0\rightarrow 1} C_L V_{dd}^2 f_{\text{clk}} \tag{4.13}$$

where $\alpha_{0\rightarrow 1}$ is the node transition activity factor, C_L the load capacitance, V_{dd} the supply voltage, and f_{clk} the clock frequency.

Digital implementations of the *arctan* and *Gaussian* functions use the CORDIC algorithm [129], which is an iterative algorithm that can be used to implement trigonometric and hyperbolic functions. The *Gaussian* function can be implemented using the relation $e^x = \sinh(x) + \cosh(x)$. A single CORDIC block can evaluate both sinh and cosh functions simultaneously. There are several implementation architectures for the CORDIC block depending on the performance requirements. The basic computation, in a single iteration or pipeline, involves two shifts, three additions, and one look-up operation. Its complexity and speed depends on the required number of bits of precision.

A VHDL implementation of a pipelined CORDIC block [96] was simulated, and the estimated power consumption using the AMI 0.35 μm CMOS process is shown in Table 4.9. This simulation was performed to obtain a baseline for comparison. Low-power implementation of CORDIC algorithms is an on going research area. Hence, for reference, results from a low-power CORDIC implementation using MOS Current Mode logic (MCML) demonstrated in [91] is also shown. The implementation specifications in [91], however, were a bit different.

4.6.1.3 Comparison of power dissipation in the mixed-mode to a digital implementation

As explained in Section 4.3.1, the mixed-mode Method-1 uses DACs and ADCs to transfer data between the digital and analog domains. Hence, the power dissipation in these devices has to be considered as part of the total power dissipation in the mixed-mode. Here, we provide a brief analysis of its impact. It is difficult to quantify the exact speed of operation

Table 4.9: Characteristics of the digital implementation - CORDIC algorithm.

Implementation	Digital	MCML [91]
$\mathbf{V_{DD}}$(V)	3.3	1.0
Clock Frequency (MHz)	1	125
Output precision (bits)	12	8
Power (μW) [†a]	550×2	4330×2

[†a] The factor of 2 is to account for the two CORDIC blocks needed to implement the *arctan* and *Gaussian* functions.

of the MITE circuits when used in implementation Method-1. However, they can operate at a relatively low frequency. Based on the delay in the circuit this frequency range is in the 10–100 kHz range. The accuracy that can be obtained using these circuits is nearly 8 bits. A TI 10-bit DAC [117] with an update rate of 75 kHz dissipates 0.75 mW, while an 8-bit ADC [118] with 70 kHz throughput has a minimum power dissipation of 0.18 mW. Using this approximate analysis, the DAC and ADC conversion dissipates nearly 1 mW power. Hence, for N particles this leads to $2N$ mW of power dissipation in the data converters for the mixed-mode Method-1. For $N = 1000$ this corresponds to a power dissipation of 2W. This assumes voltage as output from the DAC and input to the ADC. On the other hand MITE circuits operate in a current mode, so a voltage-to-current conversion is required.

The MITE implementation of the *arctan* and *Gaussian* functions together dissipates nearly 115 μW (Table 4.7), for a typical value of $I_a = 10$ nA. The combined digital implementation, operating at a lower frequency (Table 4.9) dissipates 1100 μW. This shows that the MITE implementation dissipates ten times less power than a digital implementation of the transcendental functions considered. However, if the ADCs and DACs are included, the total power dissipated in the mixed-mode is nearly 1115 μW which is of the same order as the digital implementation.

On the other hand, if Method-2 in Figure 4.4 is used the power dissipated in the data converters can be removed. Implementing the proposal and state estimation stages in the analog domain can provide additional savings. While it is difficult to give exact power savings without an implementation, we can provide an estimate. As shown in Figure 4.8, the

Table 4.10: Comparison of estimated power dissipation - Mixed-mode to a digital implementation of the bearings-only tracker.

Operation	**Mixed-mode**		**Digital**
	Method-1	Method-2	(ASIC)
Nonlinear functions	115 μW	115 μW	1100 μW
ADC/DAC	1000 μW	–	–
Addition	20 μW	–	20 μW
Multiply accumulate	$10 \times P_{\text{MAC}}$	$P_{\text{MAC}}{}^{\dagger a}$	$10 \times P_{\text{MAC}}$
Overall	1135 μW+ $10 \times P_{\text{MAC}}$	115 μW+ P_{MAC}	1120 μW+ $10 \times P_{\text{MAC}}$

$^{\dagger a}$ P_{MAC} denotes power dissipation in analog multiply accumulate operation.

particle proposal stage involves six adders. For a digital implementation of a 16-bit ripple-carry adder operating at 2 MHz, the power dissipation is 0.21 mW (See [33], Paper 4.7). Considering a lower frequency of operation, in the kHz range, the power consumed will be in the μW range. In an analog implementation, addition can be achieved by connecting the wires that carry the corresponding values, and hence no additional power is dissipated. As mentioned earlier, state estimation, which is a multiply-accumulate (MAC) operation, can be performed using a vector-matrix multiplier [35]. At frequencies of operation less than 10 MHz, when compared to a digital implementation of a MAC an analog implementation can provide a factor of 1000 in power savings [35]. For frequencies of operation in the kHz range, the analog circuit can lead to approximately a factor of ten in power savings. Hence, using the implementation in Method-2, in addition to the *arctan* and *Gaussian* functions that provide a factor of ten power savings, the proposal and state estimation stage provide a factor of nearly ten in power savings. Overall, the implementation in Method-2 will dissipate approximately 20 times less power when compared to a digital implementation. Table 4.10 summarizes the results comparing the estimated power dissipation.

4.6.2 Chip area, speed, and accuracy

As for chip area, the MITE implementation of the *arctan* and *Gaussian* functions consumes nearly 0.38 mm^2. This area is of the same order as a digital implementation of the CORDIC

algorithm in [137]. The implementation specifics in [137] are a bit different, but a single CORDIC implementation takes up 0.114 mm^2 area. A chip using the MITE implementation for the *arctan* amd *Gaussian* blocks has been sent for fabrication by Shyam Subramanian. As mentioned earlier, it is difficult to quantify the operating speed of the MITE implementation. However, based on approximate time delays obtained from simulations, it is in the 100 kHz range.

The accuracy of the output results, based on the error estimates in Table 4.7, is approximately 6–8 bits. The precision or accuracy of a digital implementation using the CORDIC algorithm depends on the number of bits used and number of iterations in the algorithm. Hence, higher accuracy could be obtained with the digital implementation if it were needed.

4.7 Discussions

In this chapter, we presented a mixed-mode implementation strategy for particle filters. We used multiple-input translinear element (MITE) networks to implement the nonlinear *arctan* and *Gaussian* functions in the weight evaluation stage of a bearings-only tracker. We presented simulation results to show the accuracy of these functions and their effect on target tracking. If the error in the output of the *arctan* function is smaller than the measurement noise standard deviation, its impact on the target tracking estimates will be minimal. We also showed that the MITE implementation of the functions dissipate approximately ten times less power when compared to a digital implementation of the CORDIC algorithm, which is used to evaluate the nonlinear functions. If the MITE implementation is used in the mixed-mode Method-1, the use of data converters can offset the power savings obtained from the analog implementation. However, if the mixed-mode Method-2 is used it leads to more power savings.

In addition to the mixed-mode implementation, we presented an FPGA implementation for the bearings-only tracker. This implementation is similar to that in [17], [4]. However, in this thesis we realized this using the Xilinx System Generator tool, which significantly reduced the time spent in developing the FPGA implementation. The motivation for this

FPGA implementation, is to provide a baseline for comparing the batch-based tracker presented in Chapter 5. We also compared the tracking estimates of the FPGA implementation to those of the mixed-mode implementation of the bearings-only tracker through simulations.

CHAPTER V

IMPLEMENTATION OF A BATCH-BASED PARTICLE FILTER TRACKER

5.1 Introduction

In this chapter, we concentrate on the implementation of the batch-based multi-target tracker introduced by Cevher [28] and presented in Chapters 2 and 3. Our goal is to demonstrate real-time performance of the batch-based DOA tracker and to address aspects that are unique to the algorithm. We first derive the complexity of the algorithm. Next we develop and analyze a fixed-point implementation of the algorithm. This provides a guideline for wordlength selection. We implement the algorithm in a floating-point DSP and present performance metrics such as speed of operation, memory utilization, and estimation accuracy. Using an approach similar to that in Section 4.3 we implement most stages of the batch-based tracker using the Xilinx System Generator. The particle proposal and weight evaluation stages in the tracker use nonlinear functions to update the DOA state. This DOA update function is next implemented in the analog domain using MITEs. The MITE networks were synthesized by Subramanian [115] in the CADSP group at Georgia Institute of Technology.

From an implementation perspective, the batch-based tracker is different from earlier work in [17], [4], [109] in two main aspects. The first difference is in the state update function. The state update function in previous work is linear and computationally less complex when compared to that in the batch-based tracker. The second difference is in the proposal function and as a result in the weight evaluation stage. Earlier work uses the state update to propose particles, and hence the weight evaluation stage only has to evaluate the data likelihood. In comparison, as described in Section 2.4, the batch-based tracker uses a Gaussian approximation to the full-posterior distribution as the proposal function. Because of this proposal function, the computation of weights requires the evaluation of

state and proposal distributions in addition to the data likelihood. These computations lead to additional complexity in the proposal stage. The Gaussian approximation is obtained using Laplace's method and also uses a Newton search. We present a soft- and hard-core FPGA implementation of the Newton search algorithm and discuss the advantages and disadvantages of these approaches. Furthermore, the data likelihood evaluation in the weighting stage handles a batch of M measurements, leading to a product of Gaussians. The batch-based tracker, however, has one advantage in that it uses a relatively smaller number of particles.

5.2 A System View of Batch-based Particle Filter Tracker

In this section, we provide a system view of the batch-based particle filter multi-target tracker developed in Chapter 2. Figure 5.1 shows a system view to better relate the research aspects in Chapter 2 and this chapter. While the *Algorithm Level* corresponds to Chapter 2, the *Architecture Level* and *Circuit Level* correspond to this chapter.

A brief description of the stages in the tracker relevant to the implementation is given here. This will be used in the later sections of this chapter, where details of the DSP, FPGA, and analog MITE implementations will be discussed. While a complete tracker was implemented on a DSP, only sections of the tracker were implemented using the FPGA and analog components.

5.2.1 Particle proposal stage

The particle proposal stage (Figure 5.2) in the DOA tracker uses an approximation of the full-posterior to propose particles, as detailed in Section 2.4. This requires performing a state update of the particles and approximating the data likelihood using the Laplace method. Evaluating the state likelihood involves evaluating (2.3) which is shown as a block diagram in Figure 5.3 and repeated here for convenience.

$$x_k(t+T) = h_T\left(x_k(t)\right) + u_k(t), \tag{5.1}$$

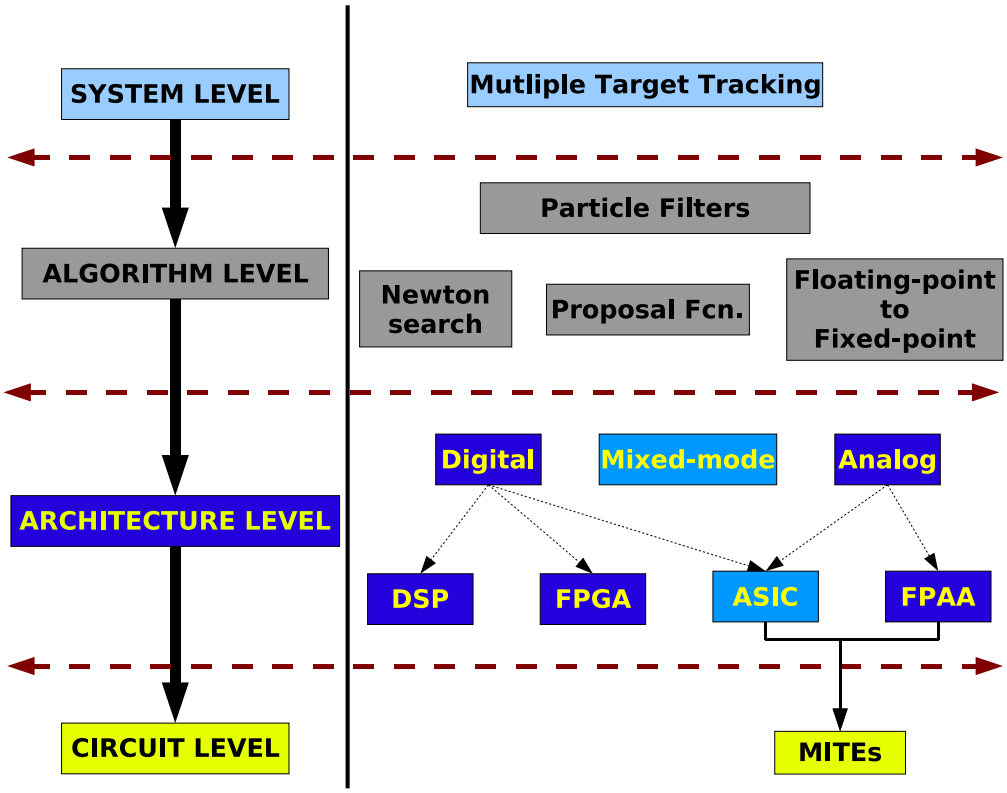


Figure 5.1: A system view of the design choices considered in developing the particle filter-based multi-target tracker. The *Algorithm Level* aspects are handled in Chapter 2. The *Architecture Level* and *Circuit Level* are handled in this chapter.

where $u_k(t) \sim \mathcal{N}(0, \Sigma_u)$ with $\Sigma_u = \text{diag}\{\sigma^2_{\theta,u}, \sigma^2_{Q,u}, \sigma^2_{\phi,u}\}$ and

$$h_T(x_k(t)) = \begin{bmatrix} \tan^{-1}\left\{\frac{\sin\theta_k(t)+T\exp Q_k(t)\sin\phi_k(t)}{\cos\theta_k(t)+T\exp Q_k(t)\cos\phi_k(t)}\right\} \\ Q_k(t) - \frac{1}{2}\log\left\{1 + 2T\exp\left(Q_k(t)\right)\cos(\theta_k(t)-\phi_k(t)) + T^2\exp(2Q_k(t))\right\} \\ \phi_k(t) \end{bmatrix}. \tag{5.2}$$

Approximating the data likelihood using the Laplace method requires a Newton-Raphson search algorithm to be implemented (see Section 2.4.4 and Appendix A). This involves computing the inverse Hessian and gradient of the cost function. These operations will be sensitive to finite-precision effects so this needs to be handled appropriately. One approach to simplify this is to use the state update as a proposal function and hence avoid the Laplace method for approximating the data likelihood. Another approach, if feasible, is to implement the Newton search in a floating-point co-processor.

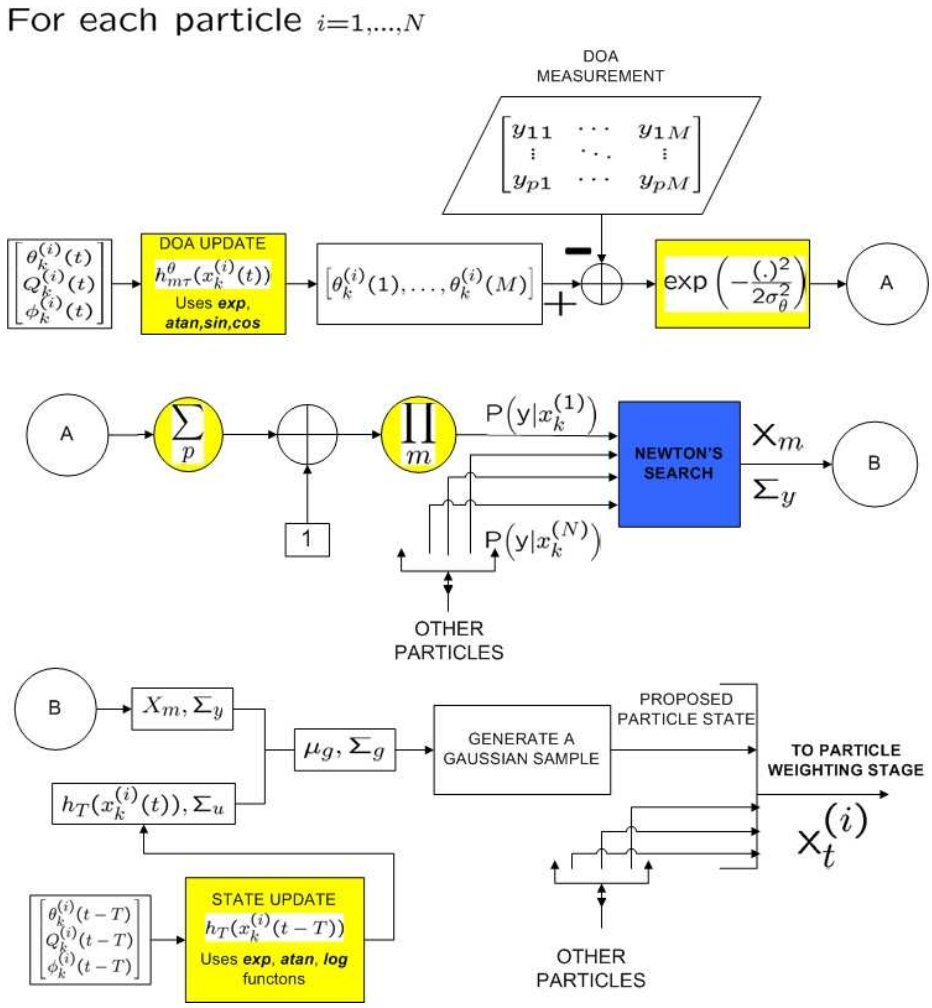


Figure 5.2: Block diagram showing the flow of computations in the particle proposal stage.

5.2.2 Particle weight evaluation stage

The weight evaluation stage (Figure 5.4) of the batch-based tracker in Section 2.4 is computationally complex when compared to that in the bearings-only tracker (Figure 4.2) in Section 4.2. As shown in (2.14) from Chapter 2, the particle weights are calculated using

$$w_t^{*(i)} = w_{t-T}^{(i)} \frac{p(\mathbf{y}_t|\mathbf{x}_t^{(i)})p(\mathbf{x}_t^{(i)}|\mathbf{x}_{t-T})}{\prod_k g_k(x_k^{(i)}(t)|\mathbf{y}_t, x_k^{(i)}(t-T))}. \tag{5.3}$$

Here, the state transition distribution, proposal distribution, and data likelihood are all Gaussians. The evaluation of the state distribution $p(\mathbf{x}_t^{(i)}|\mathbf{x}_{t-T})$ and the proposal distribution $g_k(x_k^{(i)}(t)|\mathbf{y}_t, x_k^{(i)}(t-T))$ only involve subtraction and product operations. In Figure 5.4 these functions are denoted as $L_x^{(i)}$ and $L_g^{(i)}$. Since the covariances Σ_u and Σ_g are

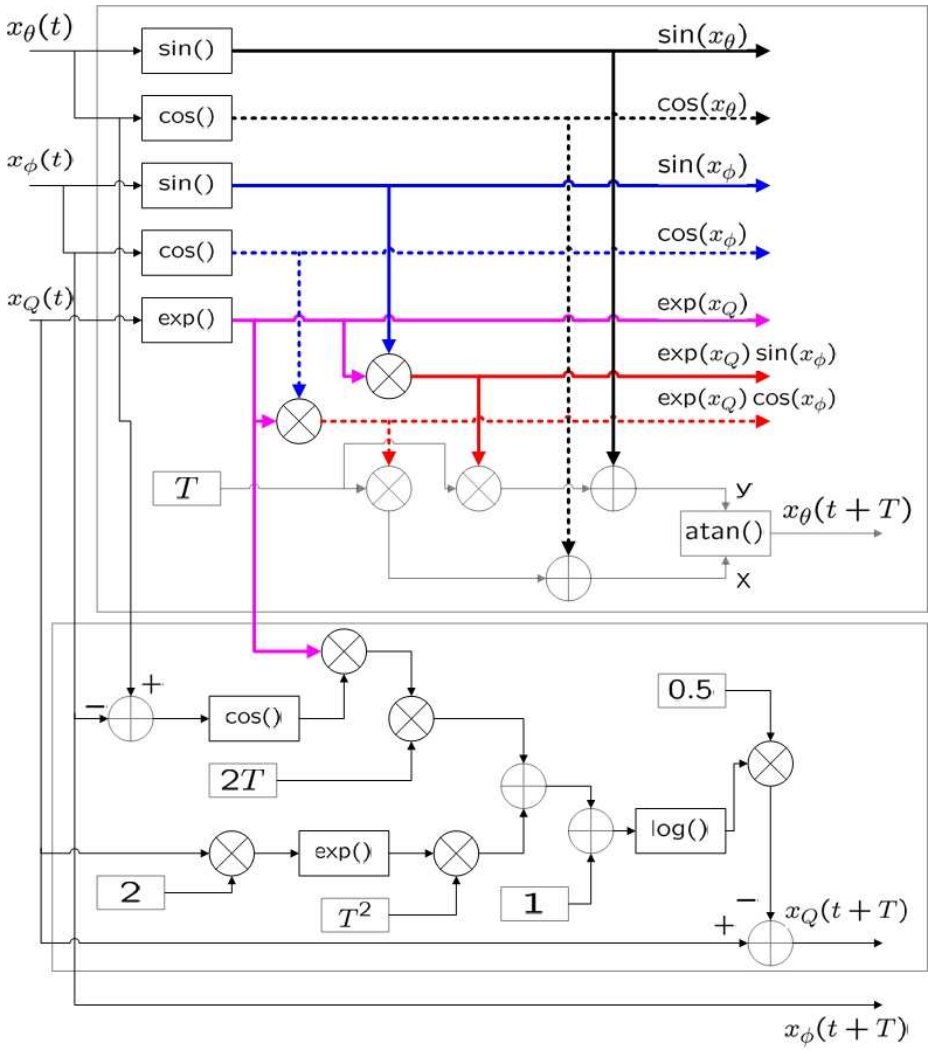


Figure 5.3: Block diagram showing the flow of computations in the particle state update stage.

diagonal matrices they do not require matrix operations. However, evaluating the data likelihood (5.4) involves M DOA updates (Figure 5.6) and M exponential evaluations. The computations in the data likelihood evaluation are shown in Figure 5.5 and the relevant equation (2.8) from Chapter 2 is repeated here for convenience:

$$p(\mathbf{y}(t)|\mathbf{x}(t)) \propto \prod_k \prod_m \left\{ 1 + \frac{1-\kappa}{\sqrt{2\pi\sigma_\theta^2(m)\kappa\lambda}} \sum_{p_i=1}^{P} \exp\left\{ -\frac{\left(h_{m\tau}^\theta\left(x_k(t)\right) - y_{t+m\tau}(p_i)\right)^2}{2\sigma_\theta^2(m)} \right\} \right\} \tag{5.4}$$

The DOA update function $h_T^{\theta}(x(t))$, with $x = [\theta(t)\ Q(t)\ \phi(t)]^T$ in (5.4), is also nonlinear with some of the functions shown in Table 5.1.

$$h_T^{\theta}(x(t)) = \theta(t+T) = \tan^{-1}\left\{\frac{\sin\theta(t) + T\exp\left(Q(t)\right)\sin\phi(t)}{\cos\theta(t) + T\exp\left(Q(t)\right)\cos\phi(t)}\right\}. \tag{5.5}$$

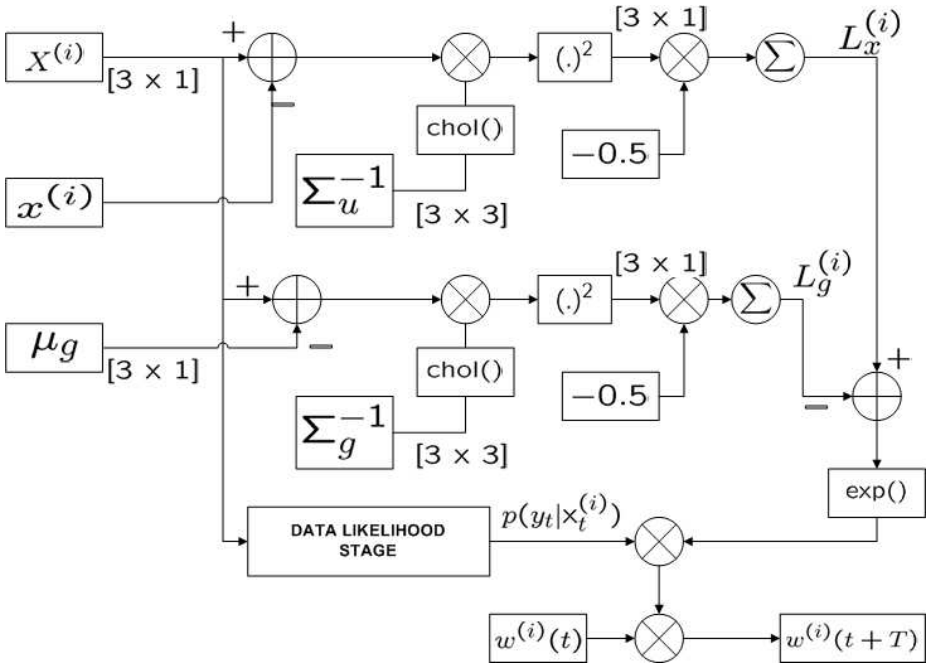


Figure 5.4: Block diagram showing the flow of computations in the particle weight evaluation stage.

Table 5.1: Functions to be implemented for the batch-based DOA target tracker

Function	**Expected range for** x †a
Gaussian $[\exp(-x^2)]$	$x - 4\sigma_\theta \leq x \leq x + 4\sigma_\theta$
$\exp(x)$	$-4.5 \leq x < 0$
$\log(x)$	$0 < x < 2$
$\arctan(x)$	$-100 < x < 100$
$\sin(x)$	$-\pi \leq x < \pi$
$\cos(x)$	$-\pi \leq x < \pi$
square-root $[\sqrt{x}]$	–
multiplies $(*)$	–
division $(/)$	–

†a These are approximate range values, obtained from simulations in Section 2.7.

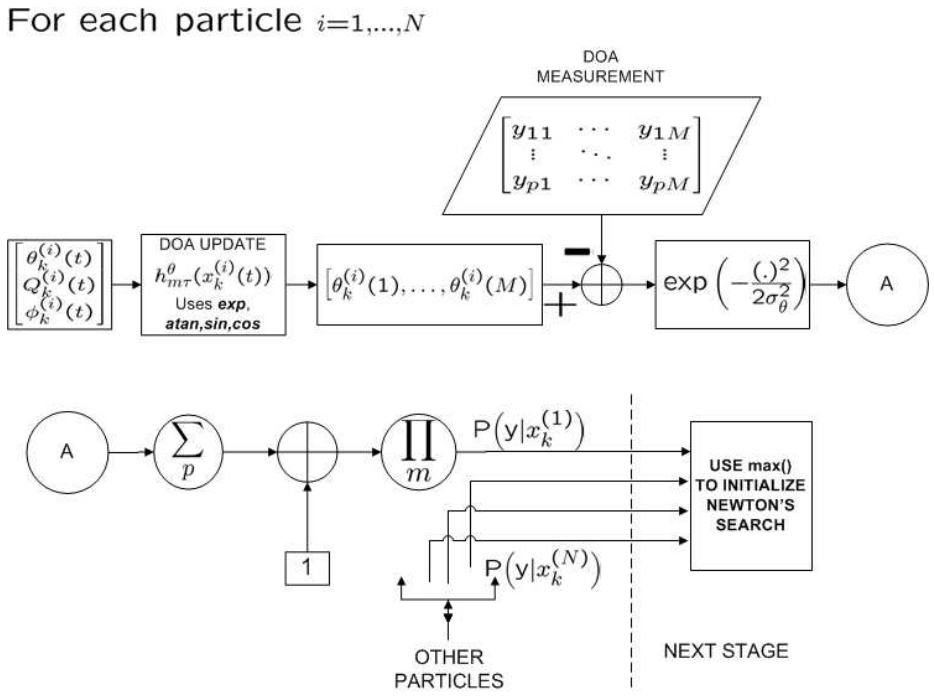


Figure 5.5: Block diagram showing the flow of computations in the data likelihood evaluation.

5.2.3 Implementation strategy

Several factors affect the design strategy to be adopted in implementing algorithms for target tracking. Some of these factors are

- Real-time constraints
- Accuracy of the estimates
- Cost of implementation
- Power dissipation and area constraints
- Flexibility of the implementation with respect to design modifications.

Our goal is to explore some of the available choices and to evaluate the above attributes in implementing the particle filter-based tracker, but we will not try to optimize the implementation for all these factors. The main motivation is to develop an implementation strategy for particle filters, which tend to be computationally complex.

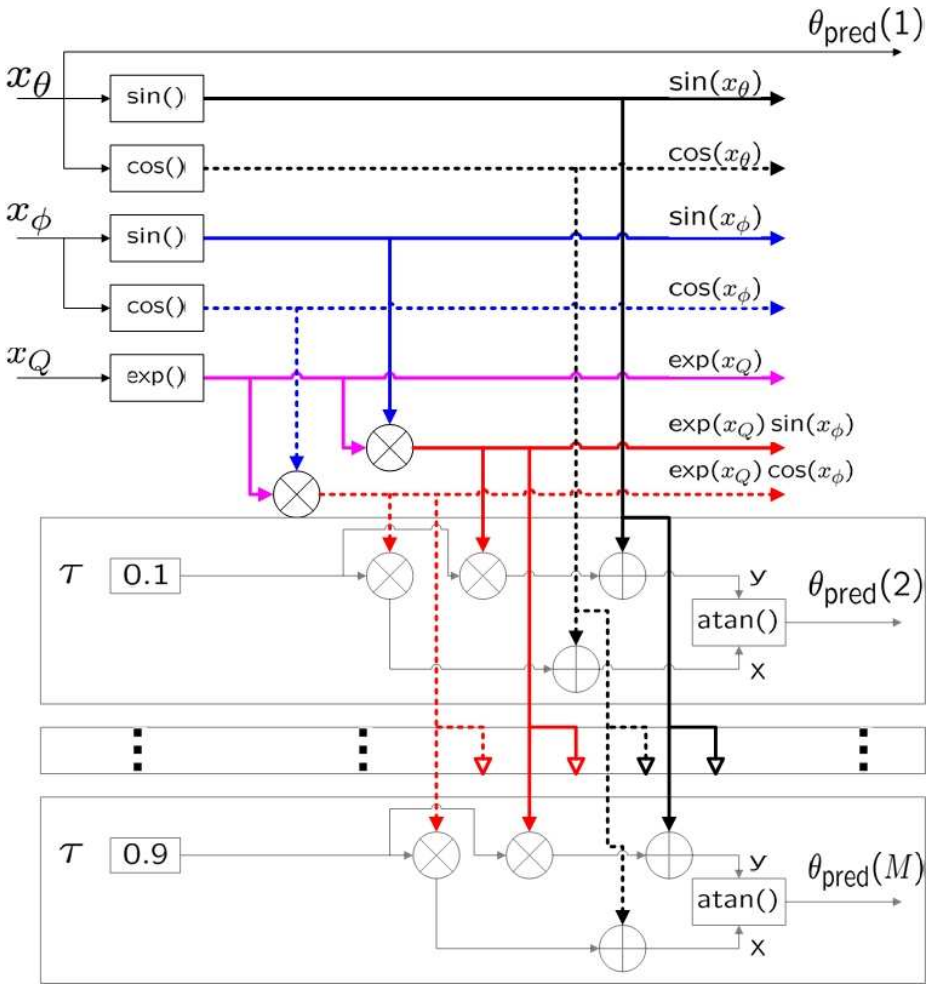


Figure 5.6: Block diagram showing the flow of computations in the particle DOA update stage.

In the digital domain, we first demonstrate that the complex batch-based particle filter multi-target tracker can be executed in real-time using a floating-point DSP. Next, we concentrate on an FPGA implementation for certain stages in the tracker. The proposal stage and weight evaluation stage are realized with an FPGA. The FPGA implementation of the data likelihood is used in evaluating weights in the batch-based tracker as part of a Matlab simulation. This is similar to our approach in Section 4.5 for the bearings-only tracker. We provide resource utilization in the FPGA and analyze the implications of the batch measurements from an implementation perspective. We compare this to the FPGA implementation of the bearings-only tracker presented in Chapter 4. We also develop soft- and hard-core applications to implement the Newton-Raphson search on the FPGA.

Table 5.2: Notation for computational complexity analysis.

Notation	Description	Typical Values
N	Number of particles	100
N_s	State vector size	3
K	Number of targets	3
M	# batch samples for DOA estimation	10 (> 5)
P	# DOA peaks in a beamformer	3 ($\geq K$)
F	# beamforming frequencies	1
N_{iter}	# iterations in Newton-Raphson search	15

In the analog domain, we develop a MITE network to perform the DOA update (5.5). We modify (5.5) into a form that requires less computation stages in the MITE implementation. Because of this modification, the sine and cosine functions need not be implemented explicitly, which leads to a reduction in the number of MITEs required for the DOA update stage.

5.3 Computational Complexity

This section presents the computational complexity of the particle filter algorithm described in Section 2.4. Table 5.2 summarizes the notation for the parameters required to derive the implementation complexities shown in Table 5.3.

The choice of our proposal function drives the complexity and execution time of the tracking algorithm. As expected complexity increases with the number of targets and the number of particles used. The complexity analysis (Table 5.3) and simulation results suggest that the overall complexity of the algorithm is of $O(NPKM)$. This is also apparent from the linear increase in the total computational time versus N in Figure 5.7, for fixed P, K,and M. The complexity of the particle proposal stage (Step 1 in Table 2.1) and the weight evaluation stage (Steps 2 and 3 in Table 2.1) are comparable. For a small number of particles, up to $N = 500$, the execution time of the Newton search is considerably more than the other operations. The Newton search does not depend on the number of particles used, but does depend on the number of targets. The number of iterations in the Newton search is typically between 10 and 20 because of the adaptive step-size and the stopping conditions.

Table 5.3: Computational complexity analysis - Batch-based particle filter tracker.

Description (Equation)	**Number of Operations**				**Complexity**
	Adds	*Mult.*	*Trans.*	*Other*	
Propose particles (Table 2.1 - Step 1)					
State $\mathbf{x_t}$ update (2.3)	$6NK$	$12NK$	$8NK$	–	$O(NK)$
Evaluate mode $\mathbf{x_{mode}}$	$12NM$	$6NM$	$4NM$	$3NM$	$O(NM)$
Newton-Raphson recursion	$K^2N_s^2M$	K^2M	$10K$	KPM †a	$O(MK^2)$
Evaluate μ_g and Σ_g (2.13)	$NK^2N_s^2$	$2NK^2N_s^2$	–	–	$O(NK^2)$
Sample particles	$NK^2N_s^2$	$NK^2N_s^2$	–	NKN_s †b	$O(NK^2)$
Update particle weights (Table 2.1 - Steps 2,3)					
Evaluate state-likelihood, proposal (2.12)	$NK^2N_s^2$	$NK^2N_s^2$	$2NKN_s$	NKN_s †a	$O(NK^2)$
Evaluate data-likelihood (2.10)	$3NKPM$	$10NKM$	$NKPM$	$NKPM$ †a	$O(NKPM)$
Evaluate weights	$3N$	$2N$	N	–	$O(N)$
Estimate states (Table 2.1 - Step 4)	NKN_s	NKN_s	–	–	$O(NK)$
Resample (Table 2.1 - Step 5)	$2N$	–	–	$2N$ †c	$O(N)$
Overall					$O(NPKM)$

†a Modulo operations, †b Random number generation, †c Compare operations.

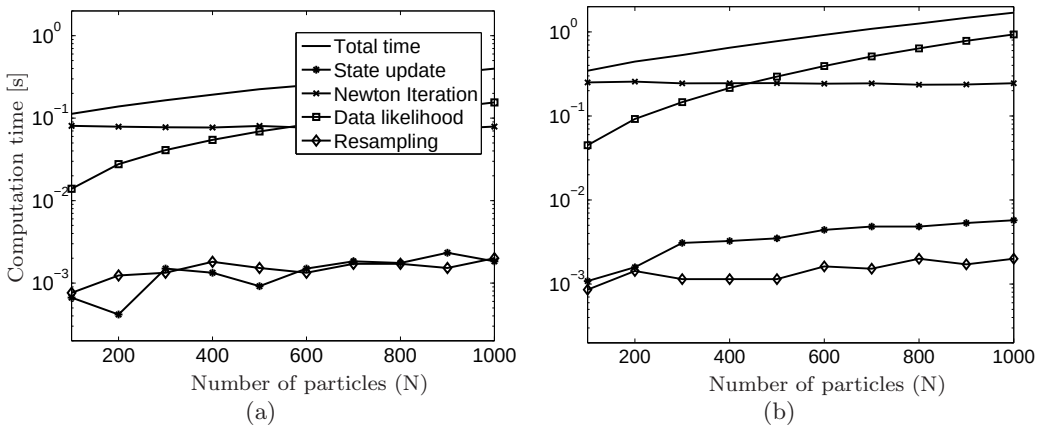


Figure 5.7: Execution times estimated from Matlab simulations for the various particle filter stages. (a) Single-target case. (b) Three-target case.

5.4 Fixed-point Analysis

Fixed-point analysis of algorithms is important to characterize their behavior when implemented in hardware. Hardware realization using fixed-point is significantly faster and consumes less power when compared to a floating-point realization.

As shown in Section 2.7, the DOA tracking algorithm has acceptable tracking performance when executed on a general purpose CPU that performs floating-point computations. To use this algorithm in fixed-point hardware, simulations are needed to measure the performance loss resulting from the word length of the fixed-point representation. Fixed-point simulations can be performed using the Matlab Fixed-point Toolbox. Following the Matlab notation [77], a signed or unsigned fixed-point number is represented using the notation $[sW, F]$ or $[uW, F]$, where s or u identifies it as a signed or unsigned quantity, W is the total number of bits (word length) and F indicates the number of bits to the right of the binary point (fraction length). More details on fixed-point numerics can be found in standard texts [130].

Fixed-point numerics Using a finite number of bits to represent numerical quantities introduces errors while performing common arithmetic operations and function evaluations. Two types of errors that can occur are (i) overflow or underflow (ii) round-off errors. If the quantity to be represented is outside the range of values that can be represented using

a chosen fixed-point representation, then an overflow has occurred. Overflows can occur during operations that accumulate results. For the proper behavior of an algorithm, these errors need to be avoided by choosing the mantissa and fraction lengths appropriately. The word length must accommodate the dynamic range of variables used in the algorithm. In the case of DOA target tracking, the dynamic range depends on the assumed tracking scenario such as target velocity, distance from the sensor, and noise variances. Round-off errors occur as a result of the minimum resolution that can be obtained in the fixed-point representation. It is difficult to avoid such errors, but care should be taken so that these errors do not accumulate. Though particle filters are recursive, the accumulation of fixed-point errors is mitigated by the resampling stage.

In the following sections, finite word length effects at individual stages of the particle filter target tracker are presented to justify the choice of word length used. In the fixed-point simulation considered here:

- All mathematical functions perform floating-point computations on fixed-point data. The fixed-point hardware implementation of these functions will have additional implications. In particular, it will affect the execution time, accuracy, and resolution of the output.

- If a word length of W is used to represent variables, then the output during a product operation can be $2W$ long and is not considered as an overflow.

Particle proposal stage The tracking algorithm considered uses an approximation of the full posterior as its proposal function. This involves determining the data likelihood and then using the state update to determine the proposal distribution as shown in Table 2.1.

State update The state update equation in (5.2) is nonlinear and uses exponential, arctangent, sine, and cosine functions. The range of values for θ and ϕ is $-\pi$ to π. The range of values for Q ($\log(v/r)$) depends on the target velocity and tracking field. Based on the parameters shown in Table 5.4 and physical constraints, the range of values for Q is -4.5 to 0. The dynamic range of θ, ϕ, and Q requires a minimum mantissa length of 4

bits to represent the integer part. The covariance of the assumed state noise Σ_u^2 places a limit on the number of bits used in the fractional part. For the simulation parameters shown in Table 5.4, $\sigma_{\theta,k}^2$ requires a fraction length of 14 bits. We choose a wordlength of $[s16, 10]$, based on simulations, to provide better estimates.

Gaussian approximation to proposal distribution This is done using Laplace's method described in Section 2.4.4. Evaluating the data likelihood involves modifications to the floating-point algorithm. The cumulative sum operation over the number of peaks (P) and the product over number of snapshots (M) are evaluated by appropriately handling the intermediate results to avoid overflow. Other modifications relate to the use of the exponential function in the data likelihood and the noise variance σ_θ^2, which is small. The conclusion is that we require a fraction length of 14 bits.

Another aspect of the proposal stage that is sensitive to finite wordlength is the Newton search to identify modes. The tolerance values, stopping criteria, and line search algorithm in the Newton search are sensitive to finite wordlength. The inverse of the Hessian matrix used in the Newton search depends on the state noise variances. In the target tracking scenario considered, the impact of the state noise variance on the DOAs is very small and hence the inverse can have large values requiring a mantissa of up to 11 bits. Each step involves computing the cost function $J(x)$, its Jacobian $G(x)$, and Hessian $H(x)$. All of these involve nonlinear function evaluations that affect the choice of word length.

Evaluate particle weights This is shown in (5.3) and involves exponential function evaluations that could be affected by limited word length. The second step, is the normalization of the weights. Though this is an elementary operation, it can affect the word length choice, because it involves a cumulative sum of N terms that could be large. Scaling the weights by N can avoid this issue. The minimum word length used to represent the particle weights is determined by the number of particles. In the target tracking simulations, particle weights are represented using a word length of 16 as $[u16, 15]$.

Estimate states An estimate of the states can be obtained as the weighted mean of the particles. Implementing this involves an inner product evaluation where the wordlength can grow as the number of terms increases. In the particle filtering algorithm, because the particle weights are normalized and sum up to one, this is not a concern.

Resampling This is a critical step in the performance of the particle filter. The use of finite word length affects this stage in a subtle, but significant, way. Using the normalized weights, the particles that need to be replicated and the number of times they need to be replicated can be determined. At the end of resampling, the sum of the particle weights needs to be one and the number of particles N should remain the same. This is difficult to obtain because of the finite number of bits used to represent the weights. A strategy to satisfy both criteria needs to be used. Efficient ways of doing this in hardware can be found in [59].

Fixed-point simulation results This section summarizes results obtained from fixed-point simulations of the multi-target DOA tracker. The parameters used in the simulations are shown in Table 5.4.

In the scenario considered, the minimum fixed-point wordlength required for acceptable tracking performance is $[s16, 10]$. Figure 5.8 compares the state estimates obtained using different fixed-point word lengths to a floating-point implementation of a single-target tracking scenario. As can be seen, with decreasing fraction lengths, the error in the state estimates increases. This is mainly caused by round-off errors that occur when shorter fraction lengths are used. The tracking result for a three-target case is shown in Figure 5.9. Increasing the number of targets increases the execution time. It does not affect the errors resulting from fixed-point computation.

Simulations performed with wordlengths and fraction lengths lower than those shown in Figures 5.8 and 5.9 either cause overflow or lead to particle divergence. The root-mean square errors (RMSE) using varying word lengths, for the single-target case are given in Table 5.5, while the multi-target case is summarized in Table 5.6. Though the RMSE of the estimates for θ and Q increases with decreasing fraction lengths, a similar trend is not

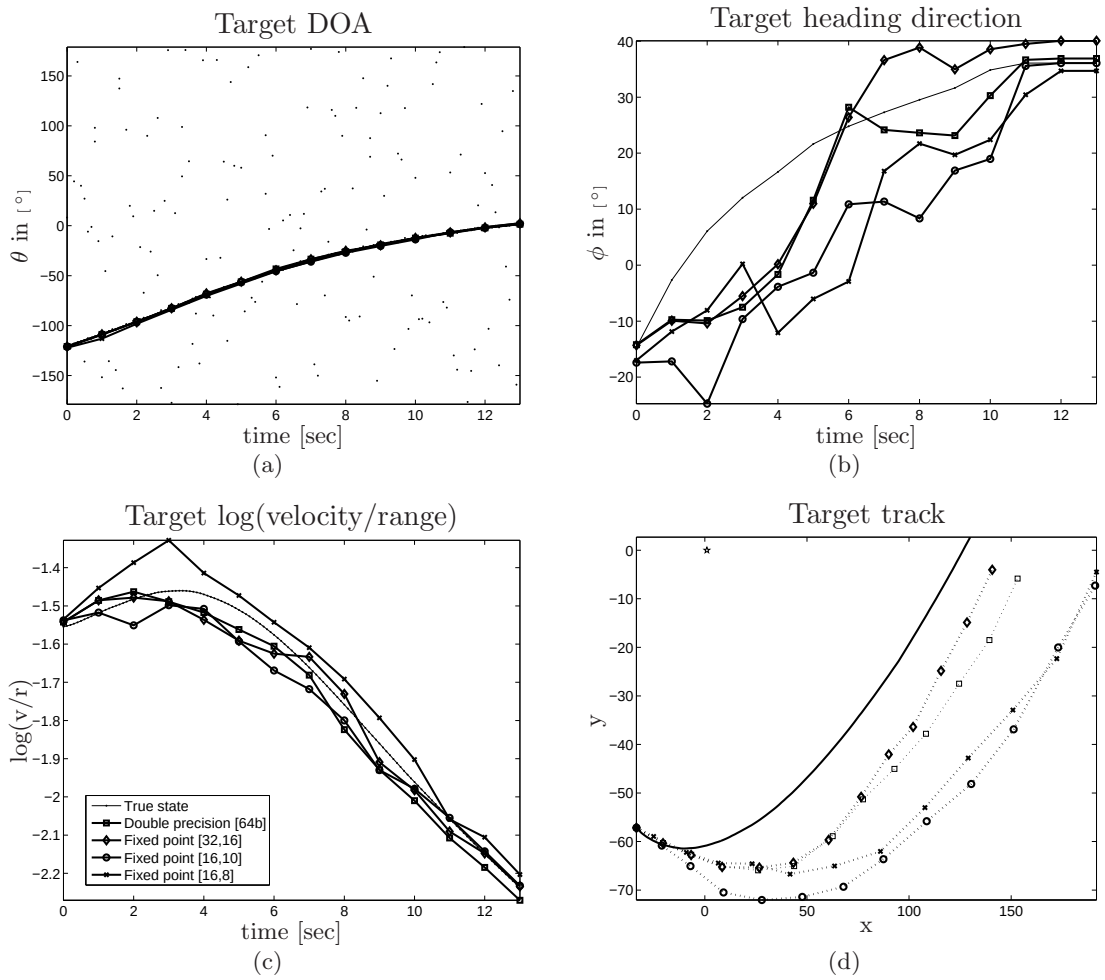


Figure 5.8: Comparison of tracker estimates for a single-target case, using particle filter with varying fixed-point wordlengths. (a) The estimated target DOAs closely follow the true DOAs. The *dots* are the noisy DOA measurements. (b) The estimated target heading directions. (c) The estimated target state log(velocity/range). (d) The target tracks obtained from the estimated states are compared to the true track. The *star* indicates the sensor location.

Table 5.4: Simulation parameters - Fixed-point simulations.

Parameter	**Value**
[Word length, Fraction length]	[16, 10]
Number of particles, N	200
State DOA noise variance, $\sigma^2_{\theta,k}$	$(1\,^\circ)^2$
State Q noise variance, $\sigma^2_{Q,k}$	$(0.05\text{s}^{-1})^2$
State heading noise variance, $\sigma^2_{\phi,k}$	$(10\,^\circ)^2$
Measurement noise variance, σ^2_{θ}	$(1\,^\circ)^2$
Tracker sampling period, T	1 s
Beamformer batch period, τ	0.1 s
Poisson clutter rate, λ	$\frac{2\pi}{600}$
Probability of target miss $\kappa^f_{0,K}$	0.1
Batch samples for DOA estimation, M	10
Number of DOA peaks, P	4 ($\geq K$)
Beamformer center frequencies, F	1
Target velocities, v	10 to 15 m/s
Surveillance region	$[600, 600]^2$ m^2

seen in ϕ because it decreases. The error in the heading estimate ϕ is high compared to the DOA θ and $\log(v/r)$ (Q) states. This is also true with the floating-point RMSE values shown in Tables 5.5 and 5.6 because the heading state noise ($10\,^\circ$) is assumed large to allow for target maneuvers.

Table 5.5: Tracking performance (RMSE) for varying wordlengths and fraction lengths - Single-target case.

Data type	State estimates - RMSE		
	$\theta[^\circ]$	$Q[\text{s}^{-1}]$	$\phi[^\circ]$
Floating-Point [64b]	0.7147	0.0427	9.9008
Floating-Point [32b]	1.0123	0.0637	10.2219
Fixed-Point [$s32, 16$]	0.7428	0.0400	9.7948
Fixed-Point [$s16, 10$]	0.9294	0.0505	17.4354
Fixed-Point [$s16, 8$]	1.7882	0.0646	15.9137

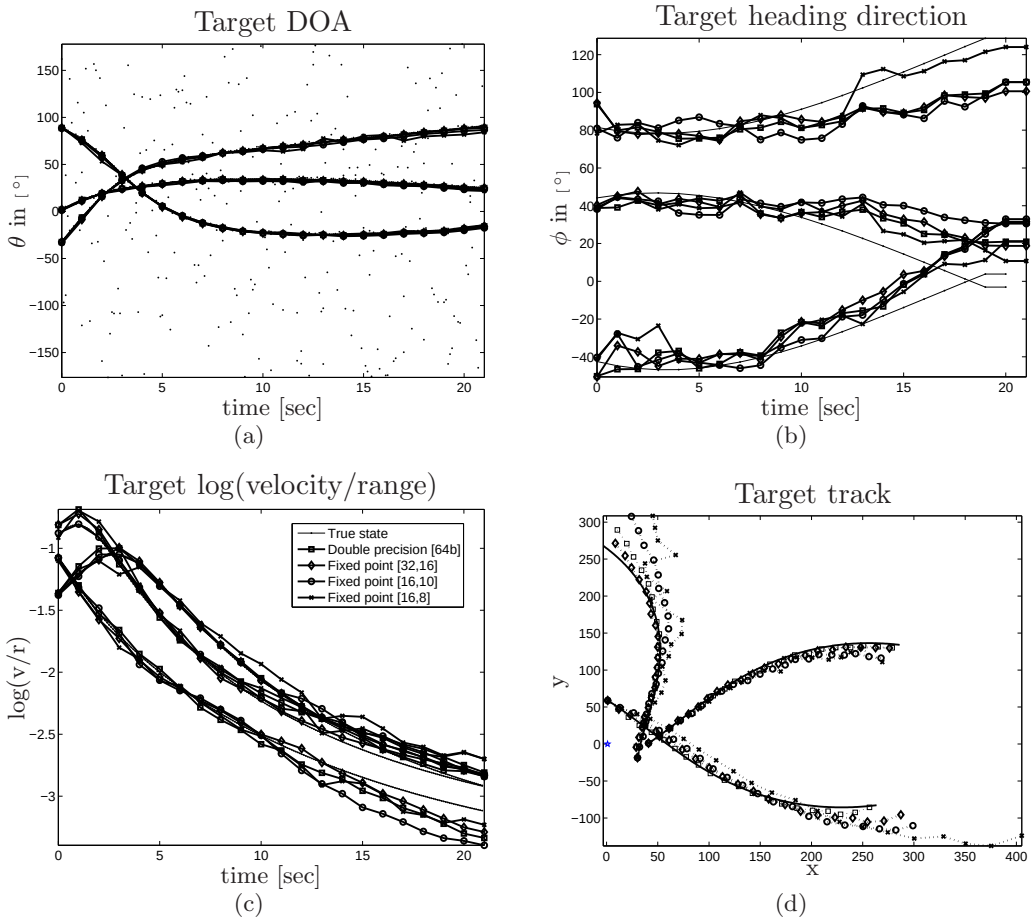


Figure 5.9: Comparison of tracker estimates for a multi-target case, using a particle filter with varying fixed-point wordlengths. (a) The estimated target DOAs closely follow the true DOAs. The *dots* are the noisy DOA measurements. (b) The estimated target heading directions. (c) The estimated target state log(velocity/range). (d) The target tracks obtained from the estimated states are compared to the true track. The *star* indicates the sensor location.

Table 5.6: Tracking performance (RMSE) for varying word lengths and fraction lengths - Multi-target case.

Data type	State estimates - RMSE		
	$\theta[^\circ]$	$Q[\mathrm{s}^{-1}]$	$\phi[^\circ]$
Floating-Point $[64b]$	1.3330	0.1384	22.7141
Fixed-Point $[s32, 16]$	1.3949	0.0953	24.0729
Fixed-Point $[s16, 10]$	3.1371	0.1885	27.7010
Fixed-Point $[s16, 8]$	4.5096	0.1902	16.0695

In practical implementations, the power consumed might be critical because power dissipation increases with wordlength. Hence, a compromise between estimation error and bit length used (power consumed) needs to be made. A trade-off among word length, execution time, and memory usage will also be critical in real-time execution. Based on the simulations and considering that we are interested in both DSP and FPGA implementations of the batch-based tracker we choose a wordlength of $[s16, 10]$.

5.5 Floating-point DSP Implementation

The particle filter based target tracker that uses batch measurement has been implemented in a floating-point DSP. The DSP board used is a DSP starter kit (DSK) [111] for the Texas Instrument (TI) C6713 chip [120]. The objective of this implementation is to demonstrate the feasibility of a real-time batch measurement-based tracker, and to quantify its performance in terms of accuracy, speed, and memory requirements.

As a first step in this process, the algorithm is implemented in C, based on the Matlab implementation used in Section 2.4. The implementation is for a floating-point processor and hence there are no round-off errors that affect the implementation. It is possible that the implementation could be optimized further by improving memory usage and the implementation of certain functions. Where possible, library functions provided by TI in their standard libraries were used.

The size of memory sections in the C6713 chip are shown in Table 5.7. The internal or on-chip memory of 256 KB is a unified program and data memory [116]. The memory size needed for storing particle states, weights, and state estimates can be estimated from the

system parameters. Table 5.8 summarizes the parameter values used to derive the memory required for individual stages, which are shown in Table 5.9. In a practical implementation the dependence on duration D will not be relevant, because data can stream in and the estimates can be streamed out and need not be stored in the DSP. The code size for an optimized implementation using the IRAM is 45.34 KB. Most of the code is devoted to the weight evaluation and proposal stages of the particle filter.

Table 5.7: Memory sections and sizes in the C6713-DSK [111].

Type	**Section**	**Size**
Internal	IRAM	192 KB
Internal	L2_CACHE	64 KB
External	SDRAM	16 MB

Table 5.8: Notation for data-size analysis.

Notation	**Description**	**Values**
N	Number of particles	100
N_s	State vector size	3
K	Number of targets	1
M	# batch samples for DOA estimation	10
P	# DOA peaks in a beamformer	2
S	Size in bytes (data-type)	4 (float)
D	Total duration	20 s

Table 5.9: Estimated data size, using the notations and values in Table 5.8.

Stage	**Size**	$K = 1, N = 1000, P = 2$	$K = 3, N = 100, P = 4$
Measurement data	$PMD \cdot S$	1.56 KB	3.12 KB
Estimated states	$N_s KD \cdot S$	0.23 KB	0.70 KB
Initial particles	$NN_s K \cdot S$	11.71 KB	3.51 KB
Particle states	$4N_s NK \cdot S$	46.87 KB	14.06 KB
Particle weights	$2N \cdot S$	7.81 KB	0.78 KB
Required heap		68.18 KB	22.17 KB

The memory used in the implementation is an important aspect when implementing on a DSP. Most DSPs have two types of memory. One is a smaller, on-chip memory or internal memory, and the other is off-chip or external memory. Performing all the processing in

Table 5.10: Size of memory sections in the IRAM, for $N = 1000, K = 1, P = 2$.

Section	**Size**
.heap (data)	88 KB
.text (code)	45.34 KB
.bios	17.21 KB
.cinit (data)	17.04 KB
.far (data)	14.05 KB
others	8.7 KB
Total	190.16 KB

the on-chip memory can provide significant advantage in terms of speed of operation and power dissipation. However, the smaller size of this memory makes this hard to do. In the algorithm considered here, the code size is rather large and occupies a significant portion of the internal memory (IRAM) as shown in Table 5.10. As long as the number of particles or number of targets is within a certain limit, all processing can be done using the internal memory. For our implementation, using the *Notation* in Table 5.8 and *Size* in Table 5.9, we were able to fit $N = 1000$, with a single-target on the IRAM. The corresponding data size is 68.18 KB.

In Table 5.10, the size of *.text* and *.bios* sections depend on the programming and algorithm code flow and are not affected by the parameters relevant to the data in the algorithm. On the other hand, the size of *.cinit* and *.far* sections depend on the parameters relevant to the algorithm. The size of *.heap* is a user-controllable parameter in the DSP implementation. The *heap* memory is used to store pre-allocated data. In our implementation, we use this for the data shown in Table 5.9. The *.heap* size of 88 KB in Table 5.10 was chosen to fit the 68.18 KB of data size shown in Table 5.9. Hence, the size of the *heap* memory can be adjusted to fit the required data size. The size of the *heap* memory is limited by the size of the IRAM and memory required by the other sections in Table 5.10. It should also be noted that apart from the code and data size used by the algorithm, there is an overhead attached to running these algorithms on a DSP. This overhead includes the DSP/BIOS and the loading of other libraries as shown in Table 5.10. There are ways to reduce this overhead by a careful design of the memory layout [119]. If the IRAM is not large enough

to accommodate the relevant sections, the external memory has to be used and that leads to reduced speed of operation. However, careful design of the code and data accesses can lead to better performance even while using the external memory.

As a simple rule of thumb, if all processing can be performed using the internal memory, the algorithm can run at least five to ten times faster than an implementation that runs in external memory. Figure 5.10 shows the total execution time versus the number of particles and Figure 5.11 shows the execution times for different stages in the algorithm. The execution time with either optimization enabled or disabled while compiling the code for the DSP is compared to that of Matlab execution times in both figures. As expected the proportion of time spent in the various stages of the algorithm is consistent between the DSP and Matlab implementations. One aspect that can be tuned in the DSP implementation is the operating clock speed. The DSP considered here can operate at a maximum clock speed of 225 MHz. The execution times shown in Figures 5.10 and 5.11 assume a clock frequency of 225 MHz. The execution times for the DSP include both the CPU cycles and the time spent at the system level to perform the computations, i.e., the time spent in memory accesses. The execution times in clock cycles were obtained using the device accurate simulator in Code Composer Studio (CCS) and verified using the actual device. The simulator is used to accurately quantify the execution times of individual functions.

Table 5.11: Clock cycles (TI-C6713) depending on optimization.

Number of Particles	**Clock Cycles**	
	Without optimization	With optimization
100	10,387,044	8,443,120
200	17,685,104	8,870,869
300	27,070,784	22,085,757
400	35,756,373	29,173,871
500	44,477,691	36,223,795
600	51,846,031	42,316,446

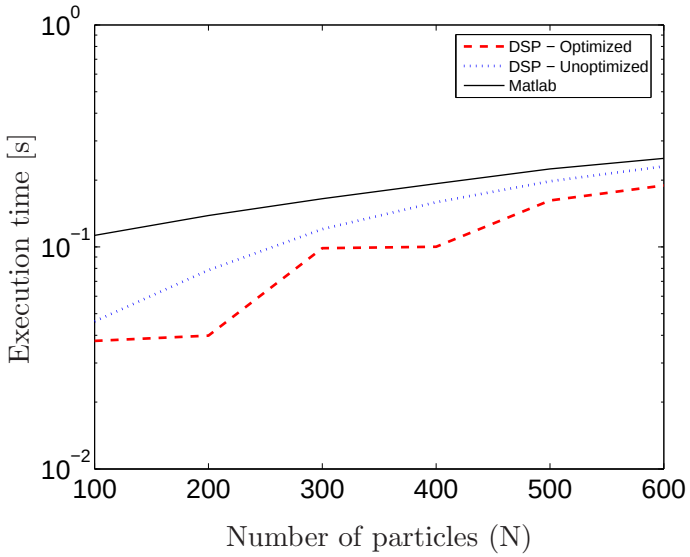


Figure 5.10: Comparison of total execution time for the particle filter algorithm running on a TI-C6713 DSP to Matlab. The operating clock frequency for the DSP is 225 MHz and the Matlab code was run on a PC operating at 1.69 GHz.

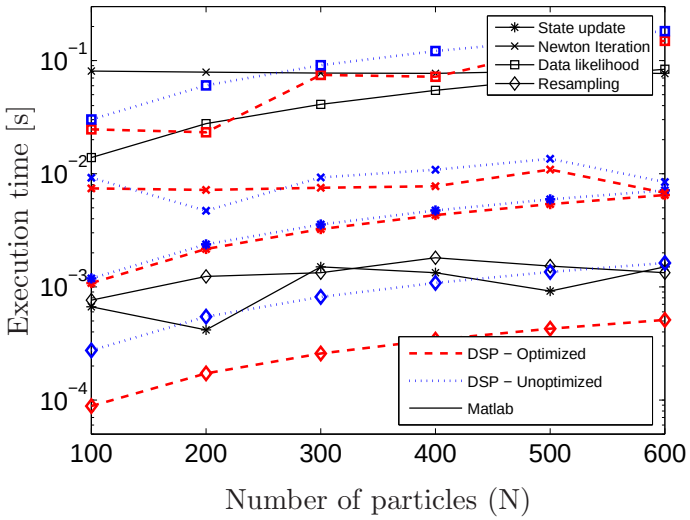


Figure 5.11: Comparison of execution times for various stages in the particle filter algorithm running on a TI-C6713 DSP to Matlab. The operating clock frequency for the DSP is 225 MHz and the Matlab code was run on a PC operating at 1.69 GHz.

5.6 FPGA Implementation of Certain Sections

In this section, we develop an FPGA implementation for the particle state update, data likelihood evaluation, and particle weight evaluation stages. We also present an FPGA based soft- and hard-core strategy to implement the Newton-Raphson search in the particle proposal stage. As mentioned in Section 4.3.3, the FPGA implementation is done using the Xilinx System Generator [136]. Based on results from the Matlab fixed-point simulations presented in Section 5.4, the fixed-point word length used is 16 bits with 10 bits for the fractional part. In Section 5.8, we provide simulation results of using the FPGA implementation of the data likelihood as part of a Matlab simulation to perform target tracking. The functionality of the individual stages implemented as System Generator models was verified by comparing with Matlab and Simulink results.

5.6.1 Particle state update

The particle state update function in (5.2) and shown in Figure 5.3 was implemented as a System Generator model. The implementation only involves updating the states θ and Q, because the heading state ϕ is propagated directly. The resource utilization for this stage is shown in Table 5.12 and the latency of this stage is shown in Table 5.13. The trigonometric, exponential, arctangent, and logarithmic functions were implemented using the CORDIC algorithm. A single realization of the CORDIC algorithm evaluates both sine and cosine functions of a parameter, leading to reduced resource usage. The CORDIC implementation in the System Generator uses a fully parallel and pipelined architecture. The approximate number of slices used (Xilinx convention as in Table 4.4) in implementing a CORDIC algorithm to realize the transcendental functions varies from 300 to 600 slices.

Table 5.12: FPGA resource utilization - Particle state update in the batch-based tracker.

Resource	**State update stage**		
	Q update	θ update	**Overall**
# Slices	2491	1877	4368
# Flip-flops	2225	1480	3705
# LUTs	4624	3551	8175
# Emb. Mult.	0	0	0

Table 5.13: Time delay at the state update stage of the batch-based tracker.

Latency in clock cycles		
Q update	θ update	**Overall**
49	34	49

5.6.2 Data likelihood evaluation

The data likelihood function to be evaluated is in equation (5.4) and the corresponding block diagram is shown in Figure 5.5. This function will be used in both the particle proposal stage and the weight evaluation stage, and plays a significant role in the particle filter performance.

Apart from the number of nonlinear functions to be evaluated, the use of batch-measurements is a significant difference between this tracker and the bearings-only tracker. For a single-target, there are two ways in which the batch computations in (5.4) can be performed. They can be done in parallel using M computation units or using a single unit with sequential updating of the product term.

The resources required for the parallel and sequential implementation are shown in Table 5.14. The resources shown are for a single target ($K = 1$), one peak ($P = 1$) in the measurement, and a batch size of ten ($M = 10$). The parallel version of implementing the data likelihood consumes nearly M times more resources when compared to the recursive implementation. The control circuitry in the recursive version to obtain the cumulative product accounts for the small increase in resources. If a parallel implementation is pursued, it will not fit on the FPGA device Xilinx XC2VP30 (Table 4.4) considered in this research. The resources for k targets with p peaks in the measurement batch, assuming that the computations for the p peaks are performed in parallel, can be approximately estimated by scaling the resources shown for the single target case in Table 5.14 by a factor of kp. While the parallel implementation results in increased resource consumption, the sequential implementation adds an additional latency of $5M$ cycles leading to an increase in the execution time of the data likelihood evaluation. The latency for the parallel implementation is shown in Table 5.16. If sufficient resources are available, a parallel implementation will

provide significant improvement in speed. The sequential approach adopted in this research uses a memory unit to store the intermediate M values. The cumulative product is not computed until all M values are available. However, pipelining this operation can further reduce the execution time.

Table 5.14: FPGA resource utilization - Data likelihood evaluation in the batch-based tracker.

Resource	**Data likelihood**	
	Parallel	Recursive
# Slices	29960	3435
# Flip-flops	28180	3311
# Block RAMs	–	–
# LUTs	57610	6588
# Emb. Mult.	–	–

5.6.3 Particle weight evaluation

The evaluation of the particle weights using (5.3) involves evaluating the state transition distribution, proposal distribution, data likelihood, and an exponential as shown in Figure 5.4. The operations involved in evaluating the state and proposal distribution probabilities are identical and hence the same blocks can be reused. Further, the delay in the data likelihood evaluation is large when compared to the evaluation of state or proposal distribution probabilities as shown in Table 5.16. Hence, the sequential evaluation of these distributions does not introduce any additional delay. The resource utilization for evaluating the state or proposal distribution probabilities is shown in Table 5.15. The resources for the data likelihood are shown in Table 5.14.

Table 5.15: FPGA resource utilization - Particle weight evaluation in the batch-based tracker.

Resource	Evaluate distributions State or proposal	Evaluate Data likelihood	**Weight evaluation**
# Slices	4906	3435	8869
# Flip-flops	6306	3311	10029
# LUTs	7889	6588	15484
# Emb. Mult.	3	0	3

Table 5.16: Time delay at the weight evaluation stage of the batch-based tracker.

Latency in clock cycles				
State distribution	Proposal distribution	Data likelihood	Weight evaluation	**Overall**
37	37	119	15	134

5.6.4 Newton-Raphson search

As explained in Section 2.4.4, Laplace's method is used to approximate the data likelihood as a Gaussian. This is achieved by executing a Newton-Raphson search algorithm to identify the mode and curvature of the distribution. As described in Appendix A, the Newton-Raphson recursion is given by the familiar expression $x_k^{l+1} = x_k^l - \nu_l H_l^{-1} G_l$, where ν_l is the algorithm step size, $G = \partial J/\partial x_k$ the gradient, and $H = \partial^2 J/\partial x_k \partial x_k^T$ the Hessian. The cost function J (reproduced from Appendix A) that needs to be minimized is:

$$\begin{aligned} J(x_k(t)) \;=\; & -\sum_{m=0}^{M-1}\sum_{p=1}^{P} \exp\left\{-\frac{\left(h_{m\tau}^{\theta}(x_k(t)) - y_{t+m\tau}(p)\right)^2}{2\sigma_{\theta}^2(m)}\right\} \\ & + \frac{1}{2}\left(x_k(t) - x_{0,k}\right)^T \Sigma^{-1}\left(x_k(t) - x_{0,k}\right). \end{aligned} \tag{5.6}$$

A generic Newton-Raphson search itself is not difficult to implement in an FPGA. Mathematical operations such as division and square-root are implemented in hardware using Newton-Raphson search [44]. However, the cost function (5.6) contains nonlinear function evaluations, which significantly increase the complexity. Another aspect that can affect the iteration is the use of fixed-point data. Unless special care is taken care to address this, the convergence of the Newton search will be affected. Furthermore, its complexity is independent of the number of particles N and does not lead to any parallelization. So, we implement the Newton search in software using floating-point data. In most FPGAs there are two choices to implement an algorithm as a software application. One choice is to use a soft-core processor and the other is to use a hard-core processor. The Xilinx device [135] we use offers both these functionalities. The soft-core is a MicroBlaze core [134] from Xilinx and the hard-core is Power PC based [131].

One advantage of using a soft- or a hard-core to implement the Newton-Raphson search

is that the C code developed for the DSP implementation in Section 5.5 can be reused with minor modifications. However, using this code as part of an FPGA, either as part of the soft- or hard-core, requires appropriate compilation. We achieve this using the Xilinx Platform Studio (XPS) for embedded development kit (EDK) [132].

The advantage of using a soft-core implementation, over a hard-core, is that the components of the designed CPU can be user specified. Further, the MicroBlaze can be configured to have a floating point unit (FPU) that can provide faster floating-point operations. However, the disadvantage of the current version of the MicroBlaze supported on the board (Xilinx XCV2P30) we use is that the maximum size of the local memory (block RAM (BRAM)) available to store the application is 64 KB. In our application, this is smaller than the code and data size for the Newton-Raphson search algorithm. The compiled size of the algorithm is 110 KB, so we cannot use the MicroBlaze soft-core for the Newton-Raphson search. However, we verified the functionality of the MicroBlaze soft-core by executing the DOA update function (5.5), which occupies less memory than the Newton search.

Next, we used a hard-core processor to achieve software implementation of the Newton search on the FPGA. The hard-core processor in the FPGA considered in this research is the Power PC 405 from Xilinx. This is a 32-bit implementation of the Power PC CPU. The floating-point operations on this hard-core are emulated using software. Hence, it cannot provide any significant speedup in the floating-point operations. But, the latest version of the FPGAs have a hard-core with the facility to incorporate an FPU that can speedup floating-point operations [133].

We performed a functional verification of the Newton-Rapshon search by executing it on the hard-core, but we did not perform any specific performance or time profiling of the Newton-Raphson algorithm. A soft- or hard-core implementation on a FPGA provides an opportunity to improve the performance of the software application by using hardware blocks for specific operations, e.g., matrix multiplication, matrix inverse, data likelihood evaluation. Similar approaches to improve performance in a different application were presented in [54].

5.6.5 Summary of resource and latency in batch-based particle filter

In this section, we present a summary of the estimated resource utilization and latency for the entire batch-based particle filter algorithm. These resources are for a single particle evaluation in the single-target case, and were obtained using the resource utilization block in the Xilinx System Generator. The resources and latency for the Newton search algorithm, implemented in hard-core, are not included in this summary. Since the entire batch-based tracker was not implemented in the FPGA, the estimates provided here are only approximate. When compared to the resources utilized by the bearings-only tracker (Table 4.5), the batch-based tracker utilizes at least four times more resources. The total number of slices in Table 5.17 is more than the available resource (Table 4.4) in the FPGA device considered in this research. However, recent FPGA devices are even larger, and would be able to accommodate the batch-based particle filter tracker.

Assuming that the resampling algorithm is the same as in Section 4.3.3, the overall latency of $3N+302$ in the batch-based tracker depends on N. The additional latency of $2N$ comes from the assumption that a single-port block RAM is used between the particle filter stages (Section 4.3.3 and [5]). As mentioned earlier and from simulations in Section 2.7, the batch-based tracker requires relatively less particles ($100 < N \leq 1000$) to provide reasonable estimates. Based on a clock frequency of 100 MHz, the delay or sampling frequency corresponding to two different N values is given in the Table 5.19. For comparison, the sampling frequency of the bearings-only tracker (from Table 4.6) is also presented. The latency in the resampling stage is the same in both trackers and depends on the resampling algorithm. For large values of N the additional latency from the individual stages of the batch tracker are not significant. Hence, meeting real-time constraints depends more on N. For smaller values of N the overall latency is comparable to that of the bearings-only tracker that requires a large number of particles. A sampling frequency of 100 kHz is sufficient to generate estimates at every time period $T = 1$ s.

Table 5.17: FPGA resource utilization - Batch-based particle filter tracker.

Resource	**PF Stage**			
	Proposal[†a]	**Weight evaluation**	**Resampling**	**Overall**
# Slices	7803	8869	374	17046
# Flip-flops	7016	10029	373	17418
# LUTs	14763	15484	653	30900
# Emb. Mult.	0	3	0	3

[†a] Excluding the Newton search and Gaussian noise generation.

Table 5.18: Time delay at various stages of the batch-based tracker.

Latency in clock cycles				
Proposal[†a]		**Weight evaluation**	**Resampling**	**Overall**
State update	Data likelihood			
49	119	134	$N+5$	$3N+302$

[†a] This does not include the Newton-Raphson search.

5.7 Analog implementation of the DOA update function

In this section, we develop an analog implementation using MITE networks for the DOA update function (5.7). This function was chosen, because it is required in both the particle proposal and weight evaluation stages of the batch-based tracker and has more nonlinear operations. The motivation to realize this function using MITEs is to continue the mixed-mode implementation in Section 4.3 and better understand the feasibility of using MITEs in particle filter algorithms. The DOA update function is reproduced here

$$h_T^{\theta}(\mathbf{x}(t)) = \theta(t+T) = \tan^{-1}\left\{\frac{\sin\theta(t) + T\exp\left(Q(t)\right)\sin\phi(t)}{\cos\theta(t) + T\exp\left(Q(t)\right)\cos\phi(t)}\right\}. \tag{5.7}$$

This can be modified into a form that requires fewer computational stages in the analog MITE implementation. Ignoring the time index of the states and state vector $\mathbf{x}$ in (5.7),

Table 5.19: Comparison of bearings-only and batch-based particle filter trackers, based on sampling frequency or estimation rate.

	Batch-based tracker		**Bearings-only tracker**	
Latency	$3N+302$ (Table 5.18)		$3N+50$ (Table 4.6)	
Sampling frequency	$N=200$	$N=1000$	$N=200$	$N=1000$
	100 kHz	30 kHz	285 kHz	33 kHz

we perform the following rearrangement of the inverse tangent:

$$\begin{aligned} h_T^{\theta}(\mathbf{x}) &= \arg(e^{j\theta} + Te^{Q}e^{j\phi}) \\ &= \arg(e^{j\theta} + ce^{j\phi}), \quad \text{where } c = Te^{Q} \\ &= \arg\Big(e^{j(\frac{\phi+\theta}{2})}\Big(e^{-j(\frac{\phi-\theta}{2})} + ce^{j(\frac{\phi-\theta}{2})}\Big)\Big) \\ &= \frac{\phi+\theta}{2} + \arg\Big((c+1)\cos(\frac{\phi-\theta}{2}) + j(c-1)\sin(\frac{\phi-\theta}{2})\Big) \\ &= \frac{\phi+\theta}{2} + \tan^{-1}\Big(\frac{(c-1)}{(c+1)}\tan\Big(\frac{\phi-\theta}{2}\Big)\Big). \end{aligned} \tag{5.8}$$

This representation replaces two sine and two cosine evaluations by one tangent evaluation resulting in a reduced number of MITEs in the analog implementation. A similar representation in the digital domain using FPGAs will not provide any significant savings, because the CORDIC algorithm can produce the sine and cosine values of an argument simultaneously. Next, we approximate the arctangent term in (5.8) using algebraic functions. The synthesis procedure for the MITE network to implement this function is similar to that followed in Section 4.3.1 for the *arctan* and *Gaussian* functions.

As was the case in the arctangent evaluation in Section 4.4.1 of Chapter 4, there is an quadrant ambiguity in the DOA update function. Additional circuitry is needed to appropriately handle this mapping.

The MITE implementation of the DOA update obtained using the synthesis procedure introduces a bias or error in the updated DOAs. The maximum error is on the order of 5–10% (nearly 9–18 °). A small uniform bias or error in the updated DOAs may not affect the weight evaluation stage, assuming the error is of the same order among all particles at a specific time instant. However, its impact on the particle proposal stage, which decides the particle support, is more serious and can lead to erroneous estimates. A large error will affect both the weight evaluation and the particle proposal. At this time, it is difficult to isolate or explain the cause for the error or deviation in the analog circuit output. Hence, we use the mathematical approximation in Table 5.20 used in implementing the MITE circuit, as part of a Matlab simulation in Section 5.8.3.

Table 5.20: Approximations used in the analog implementation of nonlinear functions in the batch-based tracker.

Function	**Approximation**	**Range**
$y = \frac{2}{\pi}\arctan(x)$	$f(x) = \frac{x}{0.63 + \sqrt{0.88 + x^2}}$	$\lvert x\rvert < \infty$
$y = \tan(\pi x)$	$f(x) = \frac{88}{3\pi}\frac{x(1-x^2)}{(1-4x^2)(3-x^2)}$	$\lvert x\rvert < 1$
$y = \frac{1}{\pi}\tan^{-1}\left(\frac{(c-1)}{(c+1)}\tan(\pi x)\right)$	$y = \frac{\alpha f_1(x)}{1.26(0.25-x^2) + \sqrt{3.52(0.25-x^2)^2 + f_1^2(x)}}$ where $\alpha = \frac{1}{2}$, and $f_1(x) = \frac{44}{3\pi}\frac{(c-1)}{(c+1)}\frac{x(1-x^2)}{(3-x^2)}$	$\lvert x\rvert < 1$

Assuming a V_{dd} of 3 V the MITE implementation of the DOA update function dissipates 2.34 μW, which is quite low. The DOA update function can be used in the data likelihood (5.4) evaluation and the state update function (5.2), which can lead to further power savings. However, the accuracy of the DOA update function has to be improved to perform the batch-based target tracking.

5.8 Simulations

In this section, we present tracking results from using the DSP, FPGA, and analog MITE implementations. While the entire batch-based particle filter algorithm was developed and implemented in the DSP, only sections of the algorithm were developed for the FPGA and analog implementations. The tracking results for the FPGA implementation are obtained by using a ModelSim description of the data likelihood as part of a Matlab simulation. The tracking results for the analog implementation are obtained by using the approximation of the MITE implementation for the DOA update stage in a Matlab simulation. The difficulty in interfacing and the time taken in running simulations in the different development environments or simulation tools makes it difficult to perform Monte Carlo simulations. However, the simulation results presented here provide functional verification of the various implementations presented in this chapter. The real-time performance of the algorithm on the DSP has been presented in Section 5.5.

Table 5.21: Simulation parameters - Target tracking using DSP, FPGA, and MITEs.

Parameter	**DSP implementation**	**FPGA or MITE simulation**
Number of particles, N	100	100
θ state noise, $\sigma_{\theta,k}$	1°	1°
Q state noise, $\sigma_{Q,k}$	0.05 s^{-1}	0.05 s^{-1}
ϕ state noise, $\sigma_{\phi,k}$	10°	10°
Measurement noise, σ_θ	1°	1°
Tracker sampling period, T	1 s	1 s
Beamformer batch period, τ	0.1 s	0.1 s
Clutter space parameter, γ	900	600
Probability of target miss, $\kappa^f_{0,K}$	0.1	0.1
Number of batch samples, M	10	10
Number of DOA peaks, P	4	1

5.8.1 DSP implementation

We performed both single- and multi-target tracking using the DSP implementation presented in Section 5.5. The simulation parameters for the tracking results presented in Figure 5.12 are given in Table 5.21. Inputs to the DSP algorithm, such as the measurement data and the initial set of particles, are provided as header files. Output estimates are obtained by executing the algorithm and the results were written to a text file. These results are compared to the Matlab results in Figure 5.12 and are found satisfactory. The data and parameters used in the DSP implementation are represented using 32-bit floating-point representation (single-precision), whereas, the Matlab simulations use a 64-bit floating-point representation (double-precision). However, this difference does not cause any degradation in the performance of the tracker.

5.8.2 FPGA implementation

As explained in Section 5.6, the data likelihood function was implemented in the FPGA. Following a procedure similar to that in Section 4.5 we use this FPGA implementation in a Matlab simulation. This is done by executing a call to the ModelSim simulation script for the data likelihood evaluation. The accuracy of the results also depends on the choice of word length in the CORDIC block for implementing the nonlinear functions in Figure 5.5.

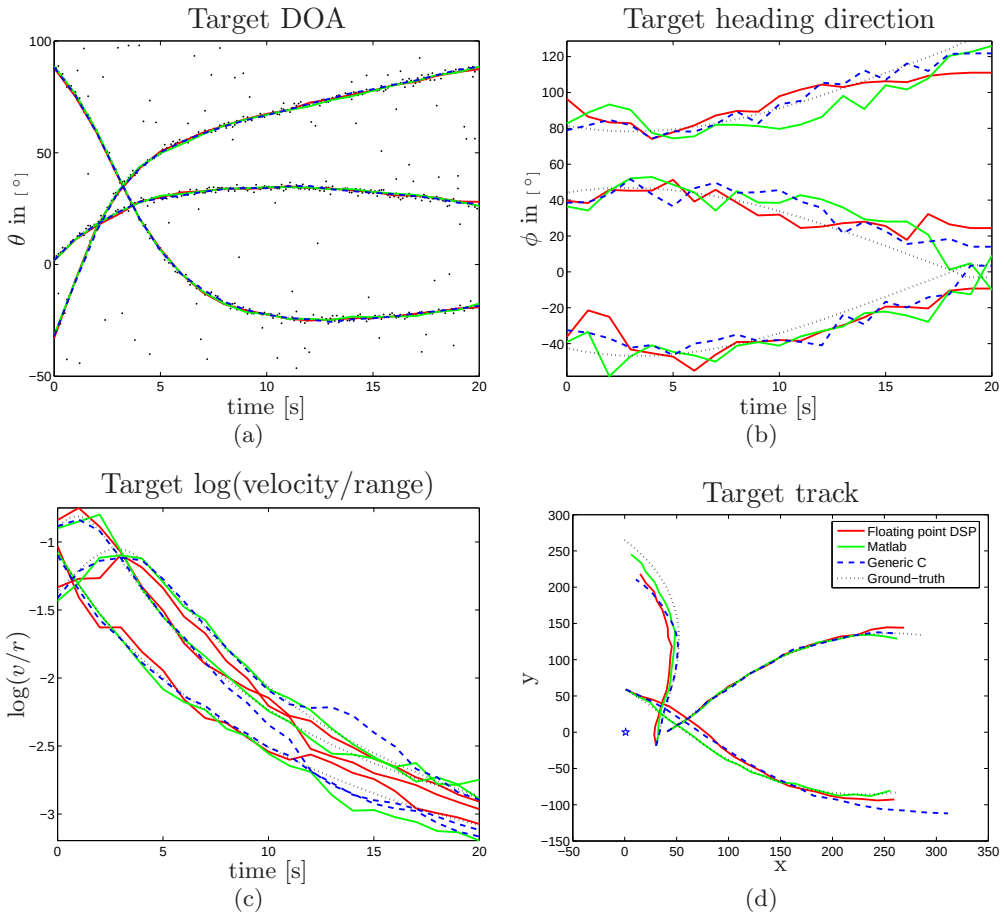


Figure 5.12: Comparison of estimation results for a multi-target case using the DSP, Matlab, and C implementation. Ground-truth is represented using *dotted* lines. State estimates are represented as *solid* or *dashed* lines. (a) Target DOA θ estimates. The *black dots* denote DOA measurements. (b) Target heading ϕ estimates. (c) Target $\log(v/r)$ estimates. (d) Target tracks calculated using the state estimates. The *star* denotes the sensor location. The target state estimates obtained using the DSP implementation are compared to the ground-truth as well as to the results from the Matlab simulation and a generic-C implementation.

In our implementation the wordlength used was 16 bits with 10 bits for the fractional part. The results from a simulation for a single-target tracking case with the parameters shown in Table 5.21 are shown in Figure 5.13.

5.8.3 Analog implementation

As mentioned in Section 5.7, the MITE implementation of the DOA update introduces a bias or error in the updated DOAs. Hence, we use the mathematical approximation in Table 5.20 used to implement the MITE circuit as part of a Matlab simulation. The tracking scenario considered is the same as in the FPGA simulation in Section 5.8.2, whose simulation parameters are shown in Table 5.21. The results from this simulation are shown in Figure 5.14.

5.9 Discussions

In this chapter, we presented our implementation approach using three possible platforms, either for the entire algorithm or sections of the algorithm, to be used in batch-based multi-target particle filter tracking.

In the digital domain, we demonstrated a real-time implementation of the algorithm on a TI-C6713 floating-point DSP. We showed that optimizing the code in a DSP implementation can speed-up the execution. As for the FPGA implementation we presented resource utilization and latency in the batch-based algorithm. We also presented soft- and hard-core approaches for the Newton search used in the algorithm. Based on our estimate, even though the entire FPGA implementation will not fit on a XCV2P30 device, more recent FPGA devices are large enough to implement this algorithm. The main reason for the increased resources is the use of pipelined CORDIC blocks for all the nonlinear operations in the algorithm. Hence, an optimal design of the FPGA implementation can improve both the speed of operation and resource utilization, by reusing CORDIC blocks and other functions with appropriate control circuitry.

In the analog domain, we implemented the DOA update function using MITEs. Although, the accuracy of the results obtained from the analog circuit simulation might not

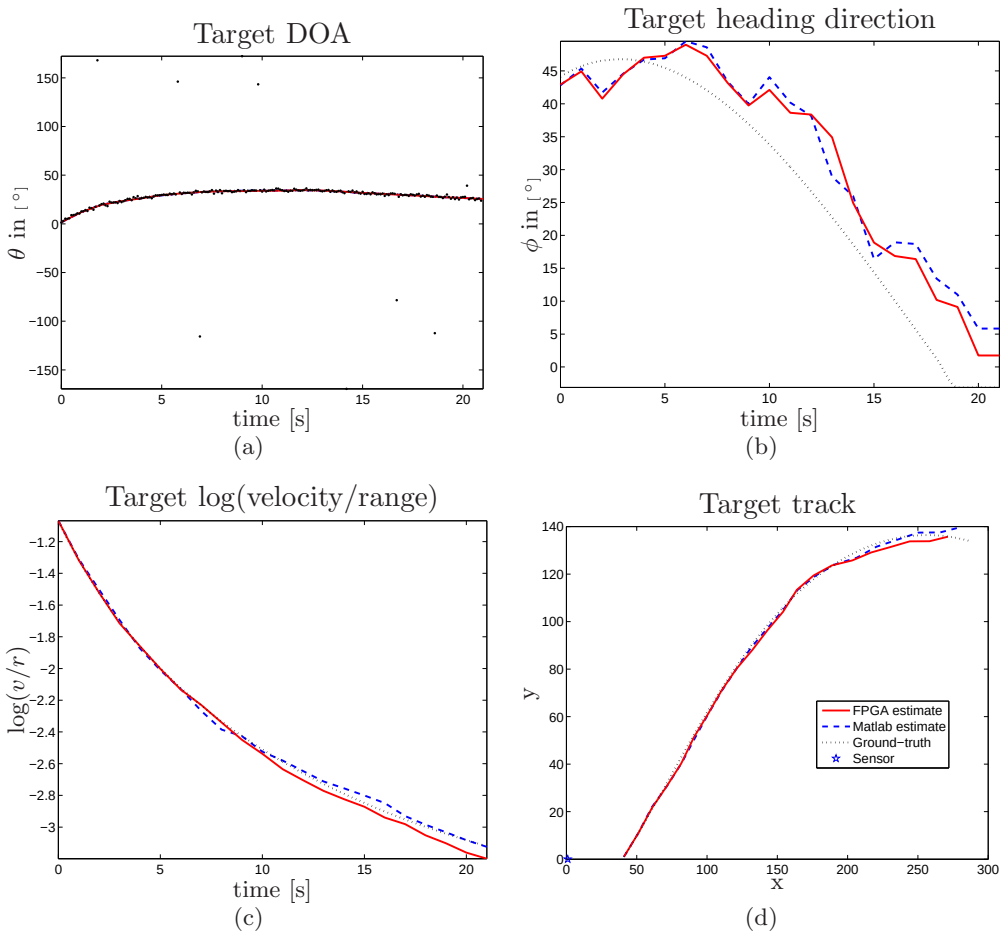


Figure 5.13: Comparison of estimation results for a single-target case using the FPGA implementation and Matlab. The target state estimates obtained using an FPGA implementation of the data likelihood stage are compared to the ground-truth as well as to the results from a Matlab simulation. Ground-truth is represented using *dotted* lines. State estimates are represented as *solid* or *dashed* lines. (a) Target DOA θ estimates. The *black dots* denote DOA measurements. (b) Target heading ϕ estimates. (c) Target $\log(v/r)$ estimates. (d) Target tracks calculated using the state estimates. The *star* denotes the sensor location.

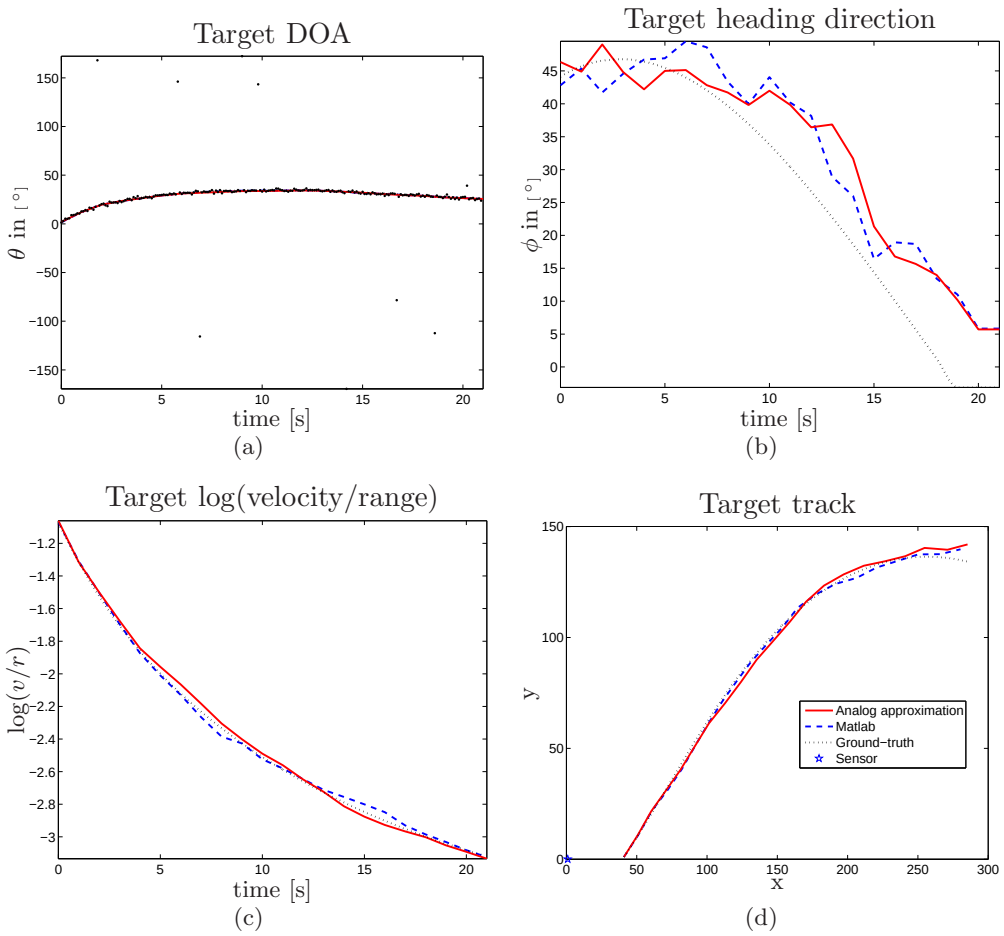


Figure 5.14: Comparison of estimation results for a single-target case using MITE implementation and Matlab. The target state estimates obtained using the approximation (Table 5.20) for the DOA update function used in the MITE implementation are compared to the ground-truth and the results from a Matlab simulation. Ground-truth is represented using *dotted* lines. State estimates are represented as *solid* or *dashed* lines. (a) Target DOA (θ) estimates. The *black dots* denote DOA measurements. (b) Target heading (ϕ) estimates. (c) Target $\log(v/r)$ estimates. (d) Target tracks calculated using the state estimates. The *star* denotes the sensor location.

be sufficient for the tracking algorithm, the significantly low power dissipation in this function justifies the use of MITEs for nonlinear functions. As discussed in Chapter 4, it is difficult to quantify the operating speed of the analog implementation. Also, if this analog implementation is to be used in a mixed-mode approach as shown in Section 4.3, the use of large number of DACs and ADCs might offset the power savings obtained from the analog implementation. However, when compared to the bearings-only tracker, the batch-based tracker uses fewer particles. Hence, a mixed-mode approach may be more suitable for the batch-based tracker, if the accuracy of the implementation can be improved.

CHAPTER VI

CONCLUSIONS

In this Chapter, we first summarize the contributions of this thesis. Next, we present directions for future research in using particle filters for target tracking and in implementing particle filter algorithms.

6.1 Contributions

The contributions of this thesis can be classified into two separate parts. The first part relates to the application of particle filters in multi-target tracking. This includes contributions from Chapters 2 and 3. These two chapters also involved joint work by the author with Dr.Volkan Cevher ([29], [30], and [27]). The second part relates to efficient implementation methods for particle filters in target tracking. This includes contributions from Chapters 4 and 5. Even though the implementations discussed were for target tracking, they should be useful for other applications that employ particle filters. The analog MITE implementations in these chapters were developed by Shyam Subramanian ([123]).

In Chapter 2, the acoustic batch-measurement-based particle filter for multi-target tracking introduced by Cevher [28] was improved. Improvements included reducing the execution time and improving the robustness of the algorithm. The tracking algorithm used a batch of direction-of-arrivals (DOAs) as measurements and follows a template-matching approach to alleviate the data association problem inherent in multi-target tracking. We also developed an automated tracking system that consists of the tracking algorithm, and an initialization algorithm that estimates the number of targets. The particle filter algorithm, when compared to a Laplacian filter and an extended Kalman filter, performed better for maneuvering targets. The target tracking results in this chapter justified the use of acoustic batch DOA-based particle filter multi-target tracking.

The acoustic DOA-based particle filter tracker in Chapter 2 was modified and extended to a range-only tracker in Chapter 3. The observations here were radar range measurements

obtained at a stationary sensor. We showed that the observability of target states required at least three range measurements. These range measurements were used in tracking multiple, maneuvering targets. The range-only tracker estimates were improved using prior road information. The estimation results for the range-only tracker also justified the use of batch measurements.

In Chapter 4, we developed mixed-mode implementation strategies for particle filter-based tracking algorithms to achieve low-power dissipation. The mixed-mode implementations used analog multiple-input translinear elements (MITEs) to realize nonlinear functions, such as the *arctan* and the *Gaussian*, used in a standard particle filter bearings-only tracking algorithm. MITEs operate in the subthreshold region, drawing lower currents and thus providing low-power operation. While the MITEs provided power savings, depending on the implementation in which they were used, the overall mixed-mode implementations did not offer significant power saving. In one implementation, Method-1, the use of data converters offsets the power savings obtained by the analog implementation. A second implementation, Method-2, that uses mostly analog components can provide nearly 20 times less power dissipation when compared to a digital implementation. We also developed a FPGA implementation of the bearings-only tracker, similar to that in [5]. We used the Xilinx System Generator tool to obtain the FPGA implementation. The use of this tool reduced the time spent in developing the system, and provided estimates of the FPGA resources consumed. By using simulations, we showed the sensitivity of the tracking results to inaccuracies in the analog implementation. For comparatively large measurement noise variances, the inaccuracy in the analog implementation did not affect the estimation results.

In Chapter 5, we developed various implementations for the batch-measurement-based particle filter tracker presented in Chapter 2. We developed a floating-point DSP implementation of the algorithm, and demonstrated that it can operate in real-time. We also presented estimates on the required memory size, based on the number of particles, targets, and other parameters of the tracking algorithm. Using fixed-point simulations, we showed that a fixed-point word length of 16 bits with 10 bits for the fractional part will be sufficient for a fixed-point implementation of the batch-based tracker. We used this

word length in developing FPGA implementations for certain sections of the batch-based tracker. We showed that the resources consumed by this tracker are at least four times more than that of the bearings-only tracker. The FPGA device we considered, the Xilinx Virtex II Pro (XCV2P30), will not be able to fit the batch-based tracker. But newer versions of these FPGAs can accommodate this algorithm. Using 200 particles, the batch tracker can operate at approximately 100 kHz. Though not all stages of the particle filter tracking algorithm are part of the resource and speed estimate, there is enough room for improvement and an FPGA implementation is feasible. We also showed that a soft- or hard-core in the FPGA can be used to implement the Newton-Rapshon search in the particle filter algorithm. In the analog domain, we used MITEs to implement the DOA update function used in the algorithm. While the MITE implementation has a low-power dissipation of 2.34 μW, the accuracy of the results are not sufficient to be used in the tracking algorithm. The accuracy of the DSP, FPGA, and MITE implementations were all verified through comparison with Matlab simulations. While the complete algorithm implemented on the DSP was used in the verification, the FPGA and MITE implementation of the weighting stage alone was used in the verification. Further, the MITE implementation to verify target tracking used the analog approximation and not the actual analog circuit.

6.2 Future research

Here, we briefly present directions for future research that could be an extension to the research performed in this thesis.

Particle filters in multi-target tracking The proposal function of the particle filter algorithm used in the multi-target tracking considered in this thesis used a Newton-Raphson search to identify the mode of the data likelihood function. Another approach to identify this mode, when using batch measurements, would be to use a random sampling consensus (RANSAC) algorithm [26]. RANSAC can be used to identify the particle that best predicts the batch of received measurements. The use of RANSAC avoids computing the gradient and Hessian required in a Newton-Raphson search.

The DOA-only and range-only trackers presented in this thesis can be used in a joint

tracker that uses both DOA and range measurements. This joint tracker would provide improved estimates on all the estimated states of the target being tracked.

Mixed-mode implementation of particle filters While this thesis developed two mixed-mode implementation strategies, to make a robust comparison with a digital implementation the actual mixed-mode implementations have to be physically realized. As a first step, this would involve using mixed-mode simulators that can also accommodate the use of ADCs and DACs, and would also include the impact of circuit level noise in the digital-to-analog interface layer for the MITE implementation. Next, the physical realization of the mixed-mode implementations would help in more accurately quantifying the speed of operation, power dissipation, and other device-level characteristics of the mixed-mode implementation.

FPGA implementation of batch-based particle filters In this thesis, sections of the batch-based particle filter tracker were implemented on an FPGA using the Xilinx System Generator tool. Implementing all the sections in an FPGA, using soft- or hard-cores to implement the Newton-Rapshon search, and integrating all these sections to perform target tracking will be a challenging, system-on-chip implementation for target tracking. Another aspect of this would be to reduce the complexity of the algorithm by using look-up table implementations for the nonlinear functions, instead of using CORDIC blocks as was the case in this thesis.

While the Xilinx System Generator is a good tool for prototyping, it does not generate an optimal implementation. Hence, other approaches to creating an FPGA implementation can be considered for a practical implementation. One interesting approach would be to use System-C to implement the algorithm and then translate it into a generic hardware description language (HDL) implementation that can be used in FPGA or ASIC implementations.

Analog implementation in target tracking The mixed-mode implementation Method-2 in this thesis uses analog implementation for most stages in the particle filter algorithm. A physical realization of this implementation will be a challenging analog circuit

implementation that would likely provide significant power savings. This implementation can use analog approaches to estimate DOAs, the MITE implementations for the nonlinear functions, a vector-matrix multiplier in the state estimation, and analog memory sections that use a digital controller in performing the resampling.

The idea of particle filters is to obtain discrete representation of distributions, and use this representation in a Bayesian framework to perform estimation. Hence, analog implementations as suggested in this thesis can be used to perform nonlinear function computations, that can provide power savings. However, a more challenging approach would be to use analog circuits to perform the numerical integrations involved in Bayesian estimation to approximate posterior distributions.

APPENDIX A

DATA-LIKELIHOOD APPROXIMATION

A.1 Newton-Raphson for Laplace's Approximation

To use Laplace's method [53] in Section 2.4.4, we need to determine the mode $x_{k,\text{mode}}$ and the Hessian H (calculated at the mode) of the density $p(\mathbf{y}_t|x_k(t))$, given by (2.11). To calculate these parameters, a Newton-Raphson search algorithm can be used on the negative log-likelihood of the data. In this appendix, we will first present a numerically robust cost function to optimize for calculating the parameters required by the partition proposal functions [22]. We then give the modifications of the Newton-Raphson algorithm that lead to faster convergence.

A.1.1 Modifications to the Cost Function

Define L^- as the negative log of $p(\mathbf{y}_t|x_k(t))$ in (2.7):

$$L^-(x_k(t)) \doteq -\sum_{m=0}^{M-1} \log\left\{1 + \frac{1-\kappa}{\kappa\lambda\sqrt{2\pi\sigma_\theta^2(m)}} \sum_{p=1}^{P} \exp\left\{-\frac{\left(h_{m\tau}^{\theta}\left(x_k(t)\right) - y_{t+m\tau}(p)\right)^2}{2\sigma_\theta^2(m)}\right\}\right\}, \tag{A.1}$$

where $\doteq$ means equality up to a constant. Unfortunately, the Newton-Raphson recursion, based on L^- (A.1), has numerical sensitivity issues. Moreover, the exact local Hessian of the partition data-likelihood is not a good covariance approximation, because it is not guaranteed to be a positive definite matrix.

As an alternative cost function to determine the required mode and the Hessian, we propose to use the following cost function

$$\begin{aligned} J(x_k(t)) = &-\sum_{m=0}^{M-1}\sum_{p=1}^{P} \exp\left\{-\frac{\left(h_{m\tau}^{\theta}(x_k(t)) - y_{t+m\tau}(p)\right)^2}{2\sigma_\theta^2(m)}\right\} \\ &+ \frac{1}{2}\left(x_k(t) - x_{0,k}\right)^T \Sigma^{-1}\left(x_k(t) - x_{0,k}\right). \end{aligned} \tag{A.2}$$

This cost function consists of two terms: the first term has the same minima as the negative log-likelihood function (A.1); and the second term is a regularization term that forces the

minima to lie close to some vector $x_{0,k}$ with respect to a weighted distance measure Σ. Nominally, the mode should be within the particle cloud coming from the previous iteration after being propagated through the state update. Hence, an easy way to determine the mode would be to choose the k^{th} partition of the particle that best explains the current data set (denoted as $x_{0,k}$ in (A.2)). Unfortunately, when the targets maneuver, $x_{0,k}$ may fall outside actual data observations. This necessitates the correction accomplished by the Newton algorithm to determine the actual mode $x_{k,\text{mode}}$.

The regularization in (A.2) is introduced to constrain the solution space to lie in the Σ-neighborhood of $x_{0,k}$, and, at the same time, impose smoothness on the target motion. Note that the cost function J depends only on the angular distances without the regularization term. With the regularization term, the gradients of J will not produce physically infeasible (motion) changes in the states $Q_k(t)$ and $\phi_k(t)$ of $x_k(t)$ to account for small angle errors, while determining $x_{k,\text{mode}}$. In addition, the parameter Σ of the regularization term also bounds the covariance of the data-likelihood approximation for numerical stability. A sensible choice for Σ is a constant α (e.g., $\alpha = 2$) times the state noise covariance Σ_u, where the constant α corresponds to how much more variation than Σ_u the target states can have.[1]

A.1.2 The Modifications to the Newton-Raphson Recursion

If we define $G = \partial J/\partial x_k$ and $H = \partial^2 J/\partial x_k \partial x_k^T$, the Newton-Raphson recursion is given by the familiar expression $x_k^{l+1} = x_k^l - \nu_l H_l^{-1} G_l$, where ν_l is the algorithm step size. Although time-consuming, it is straightforward to derive analytical expressions for G and H. Similar calculations can be found in [138] as well as in [28, 24]. We note that even with the available analytical cost function J, the calculation of the Hessian poses problems. If the Hessian of (A.2) is directly calculated from the exact formulas, it is possible to show that the resulting expression for H is not guaranteed to be positive definite, and modifications are necessary to make the Newton correction $H^{-1}G$ effective at each iteration. Hence, while calculating the final expression of the Hessian, terms including second order derivatives are

[1] The parameter α values should be chosen according to the knowledge of how fast the targets of interest can maneuver.

neglected. In this case, the Hessian is a function of the outer product of the gradient, and it is possible to prove that it is positive definite.

When the Newton-Raphson recursion converges, it does so with a quadratic convergence rate [95]. To further speed up the convergence rate of this algorithm, the step size ν_l can be changed adaptively, as long as we make sure that the cost function is always decreasing. The optimum choice for the step size ν_l is the minimizer of the following cost function:

$$\nu_l = \min_{\nu} \phi(\nu), \quad \phi(\nu) = J(x_k^l - \nu H_l^{-1} G_l). \tag{A.3}$$

To avoid the computational burden of (A.3), line search algorithms are used. The line search algorithms can be derived based on at least one of the two Wolfe conditions: (i) the sufficient decrease condition and (ii) the curvature condition [95, 43]. For the problem considered here, a *backtracking* line search algorithm is used to identify the largest acceptable ν_l based on the sufficient decrease condition. The backtracking line search algorithm, adapted from [43], for determining the Newton step size is summarized in Table A.1.

Table A.1: Newton-Raphson algorithm with backtracking step size selection.

At the l^{th} iteration of Newton-Raphson algorithm:

1. Calculate the descent direction $p_l = -H_l^{-1} G_l$. Then,
   - Set $\nu^0 = 1$;
   - While $J(x_k^l + \nu^m p_l) > J(x_k^l) + 10^{-4} \nu^m G_l^T p_l$ (*sufficient decrease condition*), do
     - Choose the contraction factor $\rho \in [0.1, 0.5]$,
     - $\nu^m = \rho \nu^{m-1}$.
   - Set $\nu_l = \nu^m$.
2. $x_k^l = x_k^{l-1} + \nu_l p_l$.
3. Continue until the stopping condition.
4. Set $x_{k,\text{mode}} = x_k^l$ and $\Sigma_y(k) = H_l^{-1}$ in Eqn. (2.13).

Lastly, it is also crucial for the Newton-Raphson algorithm to use a good stopping condition for terminating the search. Among various choices outlined in [43], we tested two conditions: (i) relative change in the cost function and (ii) relative change in the state x_k. We identified that both choices are adequate to terminate the search in approximately ten

iterations for the partition mode calculation.

A.2 Mode Hungry Approach for the Partition Data-Likelihood

To determine the mode $x_{k,\text{mode}}$ and the Hessian H in Section 2.4.4, the mode hungry Metropolis-Hastings iterates on the particles that belong to the k-th partition after they are propagated through the state update function. The MHMH generic iteration can be found in Table 2.2 in Section 2.5. The mode of the resulting distribution is taken as $x_{k,\text{mode}}$. The Hessian can be calculated using the weights of the resulting particles and

$$H = \sum_{j=1}^{N} w_*^{(j)} \left(x_k^{(j)} - x_{k,\text{mode}} \right) \left(x_k^{(j)} - x_{k,\text{mode}} \right)^T \tag{A.4}$$

where $w_*^{(j)}$ is the normalized weight of the j-th particle after the MHMH. The Hessian calculated this way is not an accurate estimate of the actual Hessian due to (i) presence of multiple targets, (ii) spurious DOA's, and (iii) finite number of MHMH iterations.[2] We estimate the Hessian by using the weights of the ten closest particles to the mode by using (A.4).

[2]If the number of particles tend to infinity, they are likely to cover the other target partitions. Hence, the estimates will naturally be biased.

REFERENCES

[1] Abramson, D., "MITE architectures for reconfigurable analog arrays," Master's thesis, Georgia Institute of Technology, Atlanta, GA, November 2004.

[2] Aouini, S. and Roberts, G. W., "A predictable robust fully programmable analog Gaussian noise source for mixed-signal/digital ate," in *Proc. IEEE Intl. Test Conf.*, October 2006.

[3] Arulampalam, M. S., Maskell, S., Gordon, N., and Clapp, T., "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking," *IEEE Trans. Signal Processing*, vol. 50, no. 2, pp. 174–188, 2002.

[4] Athalye, A., Bolić, M., Hong, S., and Djurić, P. M., "Architecture and memory scheme for sampling and resampling in particle filters," in *Proc. IEEE 11th Digital Signal Processing Workshop*, (Taos Ski Valley, New Mexico, USA), Aug. 2004.

[5] Athalye, A., Bolić, M., Hong, S., and Djurić, P. M., "Generic hardware architectures for sampling and resampling in particle filters," *EURASIP J. Applied Signal Processing*, vol. 17, pp. 2888–2902, 2005.

[6] Bar-Shalom, Y., "Tracking methods in a multitarget environment," *IEEE Trans. Automatic Control*, vol. AC-23, pp. 618–626, Aug. 1978.

[7] Bar-Shalom, Y. and Fortmann, T., *Tracking and Data Association*. Academic-Press, 1988.

[8] Bar-Shalom, Y. and Li, X.-R., *Multitarget-Multisensor Tracking: Principles and Techniques*. YBS, 3 ed., 1995.

[9] Berkeley Design Technology, Inc.(BDTI), "Comparing FPGAs and DSPs for high-performance DSP applications." White paper at "http://www.bdti.com/articles/info_articles.htm", November 2006.

[10] Blackman, S. and Popoli, R., *Design and Analysis of Modern Tracking Systems*. Norwood, MA: Artech House, 1999.

[11] Blackman, S. S., *Multiple-Target Tracking with Radar Application*. Artech House, 1986.

[12] Boers, Y. and Driessen, J. N., "Multitarget particle filter track before detect application," *IEE Proc. Radar, Sonar, and Navigation*, vol. 151, no. 6, pp. 351–357, Dec. 2004.

[13] Bolić, M., Djurić, P. M., and Hong, S., "New resampling algorithms for particle filters," in *ICASSP*, 2003.

[14] Bolić, M., Djurić, P. M., and Hong, S., "Resampling algorithms and architectures for distributed particle filters," *IEEE Trans. Signal Processing*, vol. 53, pp. 2442–2450, July 2005.

[15] Bolić, M., Hong, S., and Djurić, P. M., "Finite precision effect on performance and complexity of particle filters for bearing-only tracking," in *IEEE 36th Asilomar Conf. Signals, Systems and Computers*, vol. 1, (Pacific Grove, CA, USA), pp. 838–842, Nov 2002.

[16] Bolić, M., Hong, S., and Djurić, P. M., "Performance and complexity analysis of adaptive particle filtering for tracking applications," in *IEEE 36th Asilomar Conf. Signals, Systems and Computers*, vol. 1, (Pacific Grove, CA, USA), pp. 853–857, Nov 2002.

[17] Bolić, M., *Architectures for efficient implementation of particle filters*. PhD thesis, Stony Brook University, New York, Aug 2004.

[18] Brun, O., Teuliere, V., and Garcia, J.-M., "Parallel particle filtering," *J. Parallel and Distributed Computing*, vol. 62, pp. 1186–1202, July 2002.

[19] Cadence Design Systems, Inc.,, San Jose, CA, USA, *Virtuoso Spectre Circuit Simulator User Guide*, Product version 5.1.41 ed., July 2004.

[20] Calhoun, B. H. and Chandrakasan, A., "A 256kb sub-threshold SRAM in 65nm CMOS," in *Proc. IEEE Intl. Solid-State Circuits Conf.*, pp. 628–629, 678, Feb 2006.

[21] Calhoun, B. H., Wang, A., and Chandrakasan, A., "Modeling and sizing for minimum energy operation in subthreshold circuits," *IEEE J. Solid-State Circuits*, vol. 40, pp. 1778–1786, Sep 2005.

[22] Cevher, V. and McClellan, J. H., "An acoustic multiple target tracker," in *Proc. IEEE/SP 13th workshop on Statistical Signal Processing*, (Bordeaux, FR), 17–20 July 2005.

[23] Cevher, V. and McClellan, J. H., "Fast initialization of particle filters using a modified Metropolis-Hastings algorithm: Mode-Hungry approach," in *ICASSP 2004*, (Montreal, CA), 17–22 May 2004.

[24] Cevher, V. and McClellan, J. H., "Acoustic node calibration using a moving source," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 42, pp. 585–600, April 2006.

[25] Cevher, V. and McClellan, J. H., "General direction-of-arrival tracking with acoustic nodes," *IEEE Trans. Signal Processing*, vol. 53, no. 1, pp. 1–12, January 2005.

[26] Cevher, V., Shah, F., Velmurugan, R., and McClellan, J. H., "A multi target bearing tracking system using random sampling consensus," in *IEEE Aerospace Conf.*, March 2007.

[27] Cevher, V., Velmurugan, R., and McClellan, J. H., "Multi target direction-of-arrival tracking using road priors," in *IEEE Aerospace Conference*, (Big Sky, Montana), 4–11 March 2006.

[28] Cevher, V., *A Bayesian Framework for Target Tracking using Acoustic and Image Measurements*. PhD thesis, Georgia Institute of Technology, Atlanta, GA, Jan 2005.

[29] Cevher, V., Velmurugan, R., and McClellan, J. H., "Acoustic multi target tracking using direction-of-arrival batches." Accepted in IEEE Tran. Signal Processing, October 2006.

[30] Cevher, V., Velmurugan, R., and McClellan, J. H., "A range-only multiple target particle filter tracker," in *Proc. IEEE Intl. Conf. Acoustics, Speech, and Signal Proc. ICASSP'06*, May 2006.

[31] Chandrakasan, A. P. and Brodersen, R. W., *Low Power Digital CMOS Design*. Kluwer Academic Publishers, 1995.

[32] Chandrakasan, A. P. and Brodersen, R. W., "Minimizing power consumption in digital CMOS circuits," *Proceedings of the IEEE*, vol. 83, pp. 498–523, April 1995.

[33] Chandrakasan, A. P. and Brodersen, R. W., eds., *Low Power CMOS Design*. IEEE Press, 1998.

[34] Chang, K. and Bar-Shalom, Y., "Joint probabilistic data association for multitarget tracking with possibly unresolved measurements," *IEEE Trans. Automatic Control*, vol. AC-29, pp. 585–594, July 1978.

[35] Chawla, R., *Power-Efficient Analog Systems To Perform Signal-Processing Using Floating-Gate MOS Device For Portable Applications*. PhD thesis, Georgia Institute of Technology, Atlanta, GA, USA, December 2004.

[36] Chib, S. and Greenberg, E., "Understanding the Metropolis-Hastings algorithm," *The American Statistician*, vol. 49, no. 4, pp. 327–335, 1995.

[37] Chin, S.-S. and Hong, S., "VLSI design of high-throughput processing element for real-time particle filtering," *Intl. Symp. on Signals, Circuits and Systems*, vol. 2, pp. 617–620, Jul 2003.

[38] Cowan, G. E. R., *A VLSI Analog Computer / Math Co-processor for a Digital Computer*. PhD thesis, Columbia University, New York, NY, USA, 2005.

[39] Cowan, G. E. R., Melville, R. C., and Tsividis, Y. P., "A VLSI analog computer/digital computer accelerator," *IEEE J. Solid-State Circuits*, vol. 41, pp. 42–53, January 2006.

[40] Crisan, D. and Doucet, A., "A survey of convergence results on particle filtering methods for practitioners," *IEEE Trans. Signal Processing*, vol. 50, no. 3, pp. 736–746, March 2002.

[41] Damarla, R. T., Hao, V., Reiff, C., and Kurtz, J., "Fusion of acoustic and radar data for tracking vehicles," in *Military Sensing Symposium*, (Laurel, MD), Aug 2005.

[42] Daum, F. and Huang, J., "Mysterious computational complexity of particle filters," in *Proc. of the SPIE*, vol. 4728, (Orlando, FL, USA), pp. 418–426, Apr 2002.

[43] Dennis, J. E. and Schnabel, R. B., *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1996.

[44] Deschamps, J.-P., Bioul, G. J. A., and Sutter, G. D., *Synthesis of Arithmetic Circuits: FPGA, ASIC and Embedded Systems*. Wiley-Interscience, 2006.

[45] Doraiswami, R., "A novel Kalman filter-based navigation using beacons," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 32, pp. 830–840, April 1996.

[46] Doucet, A., Freitas, N., and Gordon, N., eds., *Sequential Monte Carlo Methods in Practice*. Springer-Verlag, 2001.

[47] Doucet, A., "On sequential simulation-based methods for bayesian filtering," Tech. Rep. CUED/F-INFENG/TR. 310, Cambridge University Department of Engineering, 1998.

[48] Doucet, A., Godsill, S., and Andrieu, C., "On sequential Monte Carlo sampling methods for Bayesian filtering," *Statistics and Computing*, vol. 10, no. 3, pp. 197–208, 2000.

[49] Edgoose, T., Allison, L., and Dowe, D. L., "An MML classification of protein sequences that knows about angles and sequences," in *Pacific Symp. Biocomputing 98*, pp. 585–596, Jan. 1998.

[50] Evans, M., Hastings, N., and Peacock, B., *Statistical Distributions, 3rd ed.* Wiley, 2000.

[51] Fike, C. T., *Computer Evaluation of Mathematical Functions*. Series in Automatic Computation, Prentice Hall, Inc., 1968.

[52] Frey, D., "Current mode class AB second order filter," *Electronics Letters*, vol. 30, pp. 205 – 206, Feb. 1994.

[53] Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. B., *Bayesian Data Analysis*. Chapman Hall/CRC, 2004.

[54] Gonzalez, I. and Gomez-Arribas, F., "Ciphering algorithms in MicroBlaze-based embedded systems," *IEE Proc.-Comput. Digit. Tech.*, vol. 153, pp. 87–92, March 2006.

[55] Gordon, N. J., Salmond, D. J., and Smith, A. F. M., "Novel approaches to nonlinear/non-Gaussian Bayesian state estimation," *IEE Proceedings-F*, vol. 140, pp. 107–113, Apr 1993.

[56] Harrison, R. R., Bragg, J. A., Hasler, P., Minch, B. A., and Deweerth, S. P., "A CMOS programmable analog memory-cell array using floating-gate circuits," *IEEE Trans. Circuits Syst. II*, vol. 48, pp. 4–11, January 2001.

[57] Hasler, P., "Low-power programmable signal processing," in *Proc. 5th IEEE Intl. Workshop on System-on-Chip for Real-Time Appl.*, (Banff, Alberta - Canada), pp. 413–418, July 2005.

[58] Hastings, W., "Monte Carlo sampling methods using Markov chains and their applications," *Biometrika*, vol. 57, pp. 97–109, 1970.

[59] Hong, S., Bolić, M., and Djurić, P. M., "An efficient fixed-point implementation of residual resampling scheme for high-speed particle filters," *IEEE Signal Processing Letters*, vol. 11, no. 5, pp. 482 – 5, 2004.

[60] Hong, S., Chin, S.-S., Djurić, P. M., and Bolić, M., "Design and implementation of flexible resampling mechanism for high-speed parallel particle filters," *Journal of VLSI Signal Processing*, vol. 44, pp. 47–62, August 2006.

[61] Hong, S., Liang, X., and Djurić, P. M., "Reconfigurable particle filter design using dataflow structure translation," *2004 IEEE Workshop on Signal Processing Systems Design and Implementation*, pp. 325 – 30, 2004.

[62] Ihler, A. T., III, J. W. F., and Willsky, A. S., "Particle filtering under communications constraints," in *Proc. 13th IEEE/SP workshop on Statistical Signal Processing*, (Bordeaux, France), July 2005.

[63] Isard, M. and Blake, A., *Active Contours*. Springer, 2000.

[64] Isard, M. and MacCormick, J., "BraMBLe: A Bayesian multiple-blob tracker," in 8^{th} *Int. Conf. on Computer Vision*, 2001.

[65] Isard, M. and Blake, A., "CONDENSATION – conditional density propagation for visual tracking," *Int. J. Computer Vision*, vol. 29, pp. 5–28, August 1998.

[66] Johnson, D. H. and Dudgeon, D. E., *Array Signal Processing: Concepts and Techniques*. Prentice Hall, 1993.

[67] Karlsson, R. and Gustafsson, F., "Monte Carlo data association for multiple target tracking," in *IEE Target Tracking: Algorithms and Applications*, (Netherlands), Oct 2001.

[68] Karlsson, R., *Particle Filtering for Positioning and Tracking Applications*. PhD thesis, Linkoping University, Linkoping, Sweden, January 2005.

[69] Karlsson, R., Schön, T., and Gustafsson, F., "Complexity analysis of the marginalized particle filter," *IEEE Trans. Signal Processing*, vol. 53, pp. 4408–4411, Nov 2005.

[70] Khan, Z., Balch, T., and Dellaert, F., "An MCMC-based particle filter for tracking multiple interacting targets," Tech. Rep. GIT-GVU-03-35, College of Computing, Georgia Institute of Technology, Oct. 2003.

[71] Kitagawa, G., "Monte-Carlo filter and smoother for non-Gaussian nonlinear state space models," *J. Comp. and Graph. Stat.*, vol. 5, no. 1, pp. 1–25, Mar 1996.

[72] Kwok, C., Fox, D., and Meila, M., "Adaptive real-time particle filters for robot localization," in *Proc. IEEE Intl. Conf. Robotics and Automation ICRA '03*, vol. 2, pp. 2836 – 2841, September 2003.

[73] Kwok, C., Fox, D., and Meila, M., "Real-time particle filters," *Proc. of the IEEE*, vol. 92, pp. 469– 484, March 2004.

[74] Larocque, J. R., Reilly, J. P., and Ng, W., "Particle filters for tracking an unknown number of sources," *IEEE Trans. Signal Processing*, vol. 50, no. 12, pp. 2926–2937, Dec. 2002.

[75] Li, X. and Jilkov, V. P., "A survey of maneuvering target tracking - Part III: Measurement models," in *Proc. SPIE Conf. on Signal and Data Processing of Small Targets*, vol. 4473, pp. 423 – 446, July 2001.

[76] Li, X. and Jilkov, V. P., "Survey of maneuvering target tracking - Part I: Dynamic models," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 39, pp. 1333 – 1364, October 2003.

[77] Linebarger, D. A. and Bryan, T. A., "An introduction to fixed-point signal processing," in *Proc. IEEE 11th Digital Signal Processing Workshop*, (Taos Ski Valley, New Mexico, USA), August 2004.

[78] Liu, J. S. and Chen, R., "Sequential Monte Carlo methods for dynamic systems," *J. American Statistical Association*, vol. 93, pp. 1032–1044, Sep 1998.

[79] MacCormick, J. and Isard, M., "Partitioned sampling, articulated objects, and interface-quality hand tracking," in *Proc. of the Europen Conference on Computer Vision*, 2000.

[80] Mallat, S. and Zhang, S., "Matching pursuits with time-frequency dictionaries," *IEEE Trans. Signal Processing*, vol. 41, no. 12, pp. 3397–3415, Dec. 1993.

[81] Maskell, S., Alun-Jones, B., and Macleod, M., "A single instruction multiple data particle filter," in *Proc. IEEE Nonlinear Statistical Signal Processing Workshop*, (Cambridge, UK), September 2006.

[82] Maybeck, P. S., *Stochastic models, estimation, and control*, vol. 1. New York, NY: Academic Press, 1979.

[83] Mentor Graphics Corporation, *ModelSim Xilinx User's Manual*, Product version 6.1e ed., March 2006.

[84] Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E., "Equations of state calculations by fast computing machines," *J. Chemical Physics*, vol. 21, pp. 1087–1092, 1953.

[85] Miller, M. L., Stone, H. S., and Cox, I. J., "Optimizing Murty's ranked assignment method," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 33, pp. 851–862, July 1997.

[86] Minch, B. A., "Construction and transformation of multiple-input translinear element networks," *IEEE Trans. Circuits Syst. I*, vol. 50, pp. 1530–1537, Dec. 2003.

[87] Minch, B. A., "Synthesis of static and dynamic multiple-input translinear element networks," *IEEE Trans. Circuits Syst. I*, vol. 51, pp. 409–421, Feb. 2004.

[88] Minch, B. A., Diorio, C., Hasler, P., and Mead, C. A., "Translinear circuits using subthreshold floating-gate MOS transistors," *Analog Integrated Circuits and Signal Processing*, vol. 9, no. 2, pp. 167–179, 1996.

[89] Minch, B. A., *Analysis, Synthesis, and Implementation of Networks of Multiple-Input Translinear Elements*. PhD thesis, California Institute of Technology, May 1997.

[90] Muller, J.-M., *Elementary Functions: Algorithms and Implementation*. Birkhäuser Boston, 1997.

[91] Musicer, J. and Rabaey, J. M., "MOS current mode logic for low power, low noise CORDIC computation in mixed-signal environments," in *Proc. Intl. Symp. Low Power Electronics and Design*, pp. 102–107, 2000.

[92] Ng, L. and Bar-Shalom, Y., "Multisensor multitarget time delay vector estimation," *IEEE Trans. ASSP*, vol. ASSP-34, pp. 669–677, Aug. 1986.

[93] Ng, W., Reilly, J. P., Kirubarajan, T., and Larocque, R.-R., "Wideband array signal processing using MCMC methods," *IEEE Trans. Signal Processing*, vol. 53, pp. 411–426, February 2005.

[94] Ng, W., Li, J., Godsill, S., and Vermaak, J., "Tracking variable number of targets using sequential monte carlo methods," in *Proc. IEEE/SP 13th workshop on Statistical Signal Processing*, (Bordeaux, France), Jul 2005.

[95] Nocedal, J. and Wright, S. J., *Numerical Optimization*. Springer-Verlag, 1999.

[96] Opencores. http://www.opencores.org, January 2007. Last accessed February, 2007.

[97] Orton, M. and Fitzgerald, W., "A Bayesian approach to tracking multiple targets using sensor arrays and particle filters," *IEEE Trans. Signal Processing*, vol. 50, no. 2, pp. 216–223, February 2002.

[98] Papoulis, A. and Pillai, S. U., *Probability, random variables and stochastic processes*. McGraw Hill, 2002.

[99] Pedram, M. and Rabaey, J. M., eds., *Power aware design methodologies*. Norwell, MA, USA: Kluwer Academic Publishers, 2002.

[100] Poor, H., *An Introduction to Signal Detection and Estimation*. Springer-Verlag, 1994.

[101] Reid, D. B., "An algorithm for tracking multiple targets," *IEEE Trans. Automat. Contr.*, vol. 24, pp. 843–854, Dec 1979.

[102] Ripley, B., *Stochastic Simulation*. John Wiley & Sons Inc., 1987.

[103] Ristic, B., Arulampalam, S., and Gordon, N., *Beyond the Kalman Filter: Particle Filters for Tracking Applications*. Artech House, 2004.

[104] Ristic, B., Arulampalam, S., and McCarthy, J., "Target motion analysis using range-only measurements: algorithms, performance, and application to ISAR data," *Elsevier Signal Processing*, vol. 82, pp. 273–296, 2002.

[105] Rutten, M. G., Ristic, B., and Gordon, N. J., "A comparison of particle filters for recursive track-before-detect," in *8th Int. Conf. on Info. Fus.*, vol. 1, pp. 169–175, July 2005.

[106] Sadasivam, M. and Hong, S., "Application specific coarse-grained FPGA for processing element in real time parallel particle filters," *Proc. 3rd IEEE Intl. Workshop on System-on-Chip for Real-Time Applications*, pp. 116 – 19, 2003.

[107] Seevinck, E., *Analysis and Synthesis of Translinear Integrated Circuits*. Amsterdam: Elsevier, 1988.

[108] Serrano, G., Smith, P. D., Lo, H. J., Chawla, R., Hall, T. S., Twigg, C., and Hasler, P., "Automatic rapid programming of large arrays of floating-gate elements," in *Proc. Intl. Symp. on Circuits and Syst.*, vol. 1, pp. I–373 – I–376, May 2004.

[109] Shabany, M., Shojania, H., Zhang, J., Omidi, J., and Gulak, P., "VLSI architecture of a wireless channel estimator using sequential Monte Carlo methods," *2005 IEEE 6th Workshop on Signal Processing Advances in Wireless Communications*, pp. 450–454, Jun 2005.

[110] Song, T. L., "Observability of target tracking with range-only measurements," *IEEE Journal of Oceanic Engineering*, vol. 24, pp. 383–387, July 1999.

[111] Spectrum Digital, Inc., Stafford, TX, USA, *TMS320C6713 DSK Technical Reference*, November 2003. Document ID 506735-0001 Rev. B.

[112] Stanacevic, M. and Cauwenberghs, G., "Micropower gradient flow acoustic localizer," *IEEE Trans. Circuits Syst. I*, vol. 52, pp. 2148–2157, Oct 2005.

[113] Stoica, P. and Nehorai, A., "Music, maximum likelihood, and Cramér-Rao bound," *IEEE Trans. ASSP*, vol. 37, no. 5, pp. 720–741, May 1989.

[114] Strid, I., "Parallel particle filters for likelihood evaluation in DSGE models: An assesment," report, Stockholm School of Economics, Stockholm, Sweden, June 2006. version 2006-06-19, accessed on 2006-12-17 at "http://people.su.se/ pzaga/research/particle_strid.pdf".

[115] Subramanian, S., Anderson, D. V., Hasler, P., and Minch, B. A., "Optimal synthesis of MITE translinear loops." Proc. Intl. Symp. on Circuits and Syst., May 2007. accepted.

[116] Texas Instruments, Dallas, TX, USA, *How to Begin Development Today With the TMS320C6713 Floating-Point DSP*, October 2002. TI Document ID SPRA809A.

[117] Texas Instruments, Dallas, TX, USA, *TLC5615C, TLC5615I – 10-BIT DIGITAL-TO-ANALOG CONVERTERS*, August 2003. TI Document ID SLAS142D.

[118] Texas Instruments, Dallas, TX, USA, *ADS7830 - ANALOG-TO-DIGITAL CONVERTER*, March 2005. TI Document ID SBAS302A.

[119] Texas Instruments, Dallas, TX, USA, *DSP/BIOS Sizing Guidelines for TMS320C2000/C5000/C6000 DSPs*, May 2006.

[120] Texas Instruments, Dallax, TX, USA, *TMS320C6713B Floating-Point Digital Signal Processor*, June 2006. TI Document ID SPRS294B.

[121] Tierney, L., "Markov chains for exploring posterior distributions," *The Annals of Statistics*, vol. 22, no. 4, pp. 1701–1728, 1994.

[122] Tierney, L. and Kadane, J. B., "Accurate approximations for posterior moments and marginal densities," *J. American Statistical Association*, no. 81, pp. 82–86, 1986.

[123] Velmurugan, R., Subramanian, S., Cevher, V., Abramson, D., Odame, K. M., Gray, J. D., Lo, H., McClellan, J. H., and Anderson, D. V., "On low-power analog implementation of particle filters for target tracking," in *Proc. 14th European Signal Processing Conf. EUSIPCO 2006*, EUSIPCO, September 2006.

[124] Vermaak, J., Andrieu, C., Doucet, A., and Godsill, S. J., "Particle methods for Bayesian modeling and enhancement of speech signals," *IEEE Trans. Speech Audio Processing*, vol. 10, pp. 173–185, March 2002.

[125] Vermaak, J., Godsill, S. J., and Pérez, P., "Monte Carlo filtering for multi-target tracking and data association," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 41, pp. 309 – 332, January 2005.

[126] Vittoz, E. A., "Low-power design: Ways to approach the limits," in *Proc. IEEE Intl. Solid-State Circuits Conf.*, pp. 14–18, Feb 1994.

[127] Wang, A. and Chandrakasan, A., "A 180-mv subthreshold FFT processor using a minimum energy design methodology," *IEEE J. Solid-State Circuits*, vol. 40, pp. 310–319, 599, Jan 2005.

[128] Wang, H. and Kaveh, M., "On the performance of signal-subspace processing-part II: Coherent wide-band systems," *IEEE Trans. Acous., Speech, and Signal Processing*, vol. ASSP-35, pp. 1583–1591, Nov. 1987.

[129] Wang, S., Piuri, V., and Swartzlander Jr., E. E., "A unified view of CORDIC processor design," in *Proc. IEEE 39th Midwest Symp. Circuits and Syst.*, vol. 2, pp. 852–855, 1996.

[130] Wilkinson, J. H., *Rounding Errors in Algebraic Processes*. National Physical Laboratory (Great Britain). Notes on applied science no. 32, London: Her Majesty's Stationery Off., 1963.

[131] Xilinx, San Jose, CA, *PowerPC Processor Reference Guide Embedded Development Kit*, September 2003. EDK 6.1, version 1.1.

[132] Xilinx, San Jose, CA, *Embedded System Tools Reference Manual - Embedded Development Kit EDK 8.1i*, October 2005. Xilinx ID: UG111 (v5.0).

[133] Xilinx, San Jose, CA, *PowerPC 405 Processor Block Reference Guide - Embedded Development Kit*, July 2005. Xilinx ID: UG018 (v2.1).

[134] Xilinx, San Jose, CA, *MicroBlaze Processor Reference Guide - Embedded Development Kit EDK 8.1i*, February 2006. Xilinx ID: UG081 (v5.4).

[135] Xilinx, San Jose, CA, *Xilinx University Program Virtex-II Pro Development System*, May 2006. Online at: "http://www.xilinx.com/univ/XUPV2P/Documentation/XUPV2P_User_Guide.pdf".

[136] Xilnx, San Jose, CA, *Xilinx System Generator for DSP v8.1 User Guide*, April 2006. Online at: "http://www.xilinx.com/support/sw_manuals/sysgen_ug.pdf".

[137] Zhang, R., Han, J. H., Erdogan, A., and Arslan, T., "Low power CORDIC IP core implementation," in *Proc. IEEE Intl. Conf. Acoustics, Speech, and Signal Proc. ICASSP'06*, vol. 3, pp. 956–959, May 2006.

[138] Zhou, Y., Yip, P. C., and Leung, H., "Tracking the direction-of-arrival of multiple moving targets by passive arrays: Algorithm," *IEEE Trans. Signal Processing*, vol. 47, pp. 2655–2666, Oct 1999.

[139] Zou, J., *On Systems with Limited Communication*. PhD thesis, Harvard University, Cambridge, Massachusetts, May 2004.

VITA

Rajbabu Velmurugan was born in Vridhachalam, India in November 1973. He received his B.E. degree from Government College of Technology, Coimbatore, India in 1995 and M.S. degree from Clarkson University, Potsdam, NY, USA in 1998, both in electrical engineering. He will receive his Ph.D. in electrical engineering from Georgia Institute of Technology, Atlanta, GA, USA in May 2007. He worked in Larsen & Toubro Ltd., India from 1995 to 1996 and in The MathWorks, Natick, USA from 1998 to 2001. His research interests include particle filtering techniques, design and implementation of real-time signal processing systems, and array signal processing. He is also interested in teaching and actively involved in projects related to signal processing education.