Algorithmic and Architectural Design Methodology for Particle Filters in Hardware

Aswin C Sankaranarayanan, Rama Chellappa* and Ankur Srivastava

Electrical and Computer Engineering Department, University of Maryland at College Park

{aswch,rama}@cfar.umd.edu, ankurs@glue.umd.edu

Abstract

In this paper we present algorithmic and architectural methodology for building Particle Filters in hardware. Particle filtering is a new paradigm for filtering in presence of non-Gaussian non-linear state evolution and observation models. This technique has found wide-spread application in tracking, navigation, detection problems especially in a sensing environment. So far most particle filtering implementations are not lucrative for real time problems due to excessive computational complexity involved. In this paper, we re-derive the particle filtering theory to make it more amenable to simplified VLSI implementations. Furthermore, we present and analyze pipelined architectural methodology for designing these computational blocks. Finally, we present an application using the Bearing Only Tracking Problem and evaluate the proposed architecture and algorithmic methodology.

1 Introduction

1.1 Problem of Particle Filtering

Filtering is the problem of estimation of an unknown quantity, usually referred to as state, from a set of observations corrupted by noise. Filtering (example, Kalman filtering) has been applied to a broad spectrum of real-life problems including GPS navigation, tracking etc. One such application is video based tracking of the location, velocity, identity (objectified as states) of a target when it is observed using a camera. We need to estimate these states (in some optimal sense) from each frame of the video. In this application, the states evolve through the motion of the target. Capturing this state information enables us to predict the location (state) of the target in future. Such prediction problems depend strongly on the concept of filtering.

The specific nature of the estimation/filtering depends greatly on the state we need to estimate, the evolution of the state with time (if any) and the relation of this state to the observations and the sources of noise. The model that captures the evolution of states is called the motion model or the State Transition Model. The observation model captures the relationship between an observation and a state. Conceptually, filtering makes a prediction and refines/validates it with an observation over a long period of time. This prediction and refinement has to performed in optimal sense (formally defined later).

Generally, analytical solutions for estimation is only possible in constrained scenarios. Kalman filtering (a purely analytical filter) [8] is an optimal filter when the state transition and the observation models are linear and the corrupting observation noise are Gaussian in nature. However, in a practical scenarios, the state transition and the observation models are usually nonlinear and the corrupting noise non-Gaussian, analytical solutions are no longer possible.

In order to address this nonlinear nature of practical problems, a new concept in filtering has gained attention [5, 2]. Popularly known as particle filtering, it uses Monte Carlo simulation as a core estimation framework. Monte Carlo Simulations are a popular tool for estimating analytically intractable quantities. Conceptually particle filtering performs the following operation.

Given the prior knowledge of the sequence of states that the system had been into $x_0, x_1, x_2...x_t$, a sequence of observations $y_1...y_t$ of the system, the posterior probability is defined as $p(x_t|y_0, y_1, \ldots, y_t)$. This posterior probability contains a lot of information about the system under observation and needs to be estimated. Particle filtering approximates this density function with a discrete set of samples called particles and the weights associated with these samples. This approximation is performing using Monte Carlo Sampling.

Because particle filtering uses Monte Carlo Sampling as basic approximation step of the posterior probability, it can easily capture the nonlinearities in the system under observation. Details about the mathematical framework would be presented in the later sections.

1.2 Particle Filtering in Hardware: A New Application Domain

Particle filtering has been applied to several real life problems of tracking, navigation, detection [7, 3, 10] especially in a sensing framework. There have been numerous software implementations of the particle filtering scheme tailored to various applications. Because particle filtering is a Monte Carlo Simulation based estimation framework, the computations tend to be very slow. Therefore, in this paper, we present hardware architectures for particle filters for speeding up the basic computations, thereby making particle filtering based solutions amenable to real time constraints. Particle filtering is more of an application domain rather than an application. Underlying computations in particle filtering vary from one application to the other. Therefore this paper proposes generic algorithmic and architectural strategies for particle filtering in hardware. The proposed techniques could be either used in an automated particle filtering synthesis tool or used by individual designers while developing their own application specific particle filtering implementations.

1.3 Specific Contributions

Specifically, this paper introduces particle filtering as a new problem for hardware synthesis. It investigates the traditional SISR based particle filtering. SISR requires latency and resource hungry resampling step. Therefore in-order to improve the hardware complexity of particle filtering we make the following contributions

1. Algorithmic Enhancements: In order to avoid the resampling step, we re-derive the theory of particle filtering using Markov Chain Monte Carlo technique. Our re-derived particle filtering technique is simpler in hardware complexity

2. Hardware Architecture: We propose two hardware architectures (sequential and parallel) for the algorithm we propose and analyze its latency and resource requirements.

Finally, in experimental results we take the *Bearings Only Tracking Problem* to demonstrate an application. We compare the traditional and re-derived algorithms and also analyze the hardware complexity for the associated particle filtering technique.

The rest of the paper is organized as follows. We first present the traditional particle filtering algorithm. This is followed by a brief analysis of the drawbacks of this algorithms as hardware architectures. In section 4, present the theory behind an alternate Monte Carlo Sampling technique called the Metropolis

Hastings algorithm and its relation to particle filtering. Section 5 analyses the proposed hardware architecture determining time estimates. In Section 6, we present an application of the proposed architecture to the problem of bearings-only tracking.

2 Particle Filtering Theory

In this section we will briefly outline the computations that need to be performed in the particle filtering scenario. Let x_t denote the state of the system in time t that we are trying to *infer*. This is modeled as a *Markov* process of initial probability distribution $p(x_0)$. We also have a sequence of of observations from time 0 to t denoted by $\{y_t; t \in \mathbf{N}\}$ which would assist us in making this inference. The theory assumes that the following information about the system under observation is known a-priori. This information is specific to the application and therefore particle filtering is not a specific algorithm but belongs to a class/domain of algorithms.

1. $p(x_t|x_{t-1})$: State transition probability for the system. This describes the way a system state at time $t - 1$ goes to time t

2. $p(y_t|x_t)$: Observation model, the probability that we obtain a certain observation given a certain state of the system

3. $p(x_0)$: The probability of the initial state of the system, ie. at t=0

In order to infer the current state of the system at time t based on the observations made so far, we need to evaluate the posterior probability of the system given as $p(x_t|y_{1:t})$. This posterior probability would be used to draw an inference about the system, which is mathematically represented as

$$I(f_t) = \mathbf{E}_{p(x_t|y_{1:t})}[f_t(x_t)] \tag{1}$$

where $f_t(\cdot)$ is some function of interest. Examples of such an inference could be the conditional mean, where $f_t(x_t) = x_t$. The posterior probability is estimated using a recursive technique based on the well known *Bayes Theorem*

$$p(x_t|y_{1:t}) = \frac{p(y_t|x_t)p(x_t|y_{1:t-1})}{p(y_t|y_{1:t-1})} \tag{2}$$

In order to compute the posterior probability, we will need to compute the quantity $p(x_t|y_{1:t-1})$. Computation of this quantity is called the *prediction* step and is mathematically represented as follows

$$p(x_t|y_{1:t-1}) = \int p(x_t|x_{t-1})p(x_{t-1}|y_{1:t-1})dx_{t-1} \tag{3}$$

Moreover equation 2 also requires us to know $p(y_t|x_t)$ (defined by the observation model) and $p(y_t|y_{1:t-1})$. Note that, in reality there are no unknowns in equation 2 since all parameters are either specified or computable (like equation 3). The problem is that this computation may not have an analytical representation. Therefore particle filtering approximates the desired posterior $p(x_t|y_{1:t})$ with a set of particles or samples $S_t = \{x_t^{(i)}, w_t^{(i)}\}_{i=1}^N$, with associated weights. Conceptually, these particles could be understood as samples from the posterior density with the weights as unbiasing terms to account for sampling. This computation is performed recursively, starting from time $t = 0$. This algorithm is popularly known as SISR.

- **Initialization:** At time $t = 0$, using Markov Chain Monte Carlo (MCMC) techniques, sample $\{x_0^{(i)}\}, i = 1, \ldots, N$ from the initial density function $p(x_0)$.

- **Particle Proposition** Based on the observation y_t and the state of the system at time $t-1$ (represented by particles $\{x_{t-1}^i, w_{t-1}^i\}, i = 1, \ldots, N$, *propose* a set of new particles $\{x_t^{(i)}, i = 1, \ldots, N\}$ from the proposal function or the *importance density function* $g(x_t|x_{t-1}^{(i)} y_t)$.

- **Importance Weights:** With each proposed particle $x_t^{(i)}$, associate an unnormalized weight $\tilde{w}_t^{(i)}$ defined as:

$$\tilde{w}_t^{(i)} = w_{t-1}^{(i)} \frac{p(y_t|x_t^{(i)})p(x_t^{(i)}|x_{t-1}^{(i)})}{g(x_t^{(i)}|x_{t-1}^{(i)} y_t)} \tag{4}$$

- Normalize $\tilde{w}_t^{(i)}$ to get $w_t^{(i)}$

$$w_t^{(i)} = \frac{\tilde{w}_t^{(i)}}{\sum_{j=1}^N \tilde{w}_t^{(j)}} \tag{5}$$

- **Compute the Expected Inference:** The proposed particles x_t^i and the associated weights w_t^i are properly weighted with respect to the desired posterior. Therefore

$$E_p[f(x_t)] = E_g[f(x_t)w(x_t)] = \lim_{N \to \infty} \sum_{i=1}^N f(x_t^{(i)})w_t^{(i)} \tag{6}$$

- **Resampling:** Resampling is a necessary step that replicates particles with higher probability and reduces the number of particles with lower probability. By doing this, we rejuvenate the particle set to obtain better results for future time instants. The resulting particle set is then used in the next time step to predict the posterior probability subsequently. Resampling involves sampling a new set of particles from the set $\{x_t^{(i)}, i = 1, \ldots, N\}$ according to the weights $\{w_t^{(i)}, i = 1, \ldots, N\}$. The new set of particles are assigned identical weights $(= \frac{1}{N})$. Without resampling the particle set would have a very large variance thereby reducing the quality of the estimate.

- Goto the next iteration

Figure 1 illustrates the based block diagram of this particle filtering scheme.

3 Hardware Complexity of SISR

In order to speed up the particle filtering technique, it has become imperative to perform it in hardware. Effective algorithmic and architectural approaches are therefore desired. The particle filtering technique, as presented in the previous section (and illustrated in figure 1), essentially comprises of a hardware block which is executed iteratively from one observation interval to another y_{t-1} to y_t. During this interval the particles from the previous time step are processed as discussed earlier to generate particles for the current time step. The SISR algorithm presented above has several application specific basic blocks including the proposal block, importance weights computer, normalizer, re-sampler and inference block. Although this implementation assists in speeding up the particle filtering scheme there are several disadvantages, as outlined below.

1. Complexity in Resampling: The resampling block is a complicated hardware implementation and can therefore impact the speed and resource utilization. Moreover it is not amenable to pipelining (since we cannot start resampling until all the samples are ready).

2. Flexibility: Having hardware implementations implies loosing out on flexibility. But there are certain aspects of flexibility that must not be lost from a particle filtering perspective. It has been discussed that we start with N particles from the previous iterations, propose, evaluate and resample exactly N particles. Typically, hardware implementations would end up fixing N to a constant. This can adversely affect the quality of particle filtering. Typically, we would be interested in proposing $Q \geq N$ particles and finally deriving the inference from Q particles. We would also like Q to be a variable chosen depending on the need and situation. For example in visual tracking applications, in the event of an occlusion we would like to propose more than N particles. Note that in such scenarios we would still use only N samples to represent the posterior. The degree of flexibility offered by SISR based implementations is very low.

In order to address these problems, we re-derive the particle filtering scheme using an alternative approach in which re-sampling is not needed and flexibility (as described above) can still be obtained.

4 MCMC based Particle Fildtering

The particle filtering (PF) technique which helps in evaluating the posterior probability (equation 2) and the state expected inference (equation 1) is essentially a sequential Monte Carlo technique. In a way, these are special cases
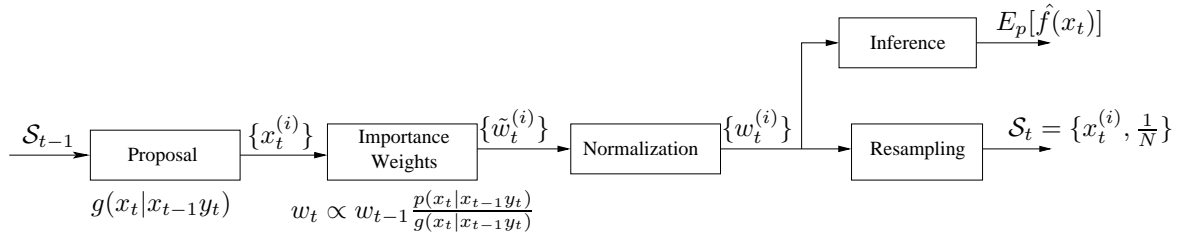
Figure 1. Figure shows commonly used SIS-R algorithm with resampling done at the end of every estimation cycle.

of more general Monte Carlo Markov Chain (MCMC) based density sampling technique. The Metropolis Hastings Algorithm (MHA) [1, 6] is considered the most general MCMC based sampling. Most popular MCMC samplers such as the Metropolis Sampler [9] or the Gibbs Sampler [4] are special cases of this algorithm.

The MHA and the PF both address the issue of generating samples from a distribution whose functional form is known (upto a normalizing factor) and which is difficult to sample. In this section, we present a hybrid sampler that uses the sampling methodologies adopted in MCMC samplers (specifically, the MHA algorithm) for the problem of estimating posterior density functions. We later show that such a scheme removes the adverse effects of the resampling block in the traditional PF architecture. It is also shown that such an architecture has more desirable properties in its scalability and flexibility.

In this section, we present the *Metroplis-Hastings Algorithm* and its derivative, the so called *Independent Metropolis-HAstings Chain* and show its applicability for particle filtering. For brevity, these derivations are provided briefly.

4.1 Metropolis Hastings Algorithm (MHA)

We first present the general theory of MCMC sampling using the MHA Algorithm and then state the conditions under which the general theory fits into the particle filtering algorithm presented before. It could be recalled that the particle filtering scheme needs to generate samples of the form x_t^i, w_t^i where these samples represent the posterior density. Similarly, MHA generates these samples from the desired density (say $p(x)$) by generating samples from an easy to sample distribution say $q(x|y)$. MHA produces a sequence of state $\{x^{(n)}\}$, which by construction is Markovian in nature, through the following iterations.

1. Initialize the chain with an arbitrary value $x^{(0)} = x_0$.

2. Given $x^{(n)}$, generate $\hat{x} \backsim g(\cdot|x^{(n)})$, where g is the sampling or proposal function.

3. Accept $\hat{x}$ with probability $\alpha(x^{(n)}, \hat{x})$, that is, for a uniform random variable $u \backsim \mathcal{U}[0, 1]$

$$x^{(n+1)} = \begin{cases} \hat{x} & \text{if} \quad u \leq \alpha(x^{(n)}, \hat{x}) \\ x^{(n)} & \text{otherwise} \end{cases}$$

$$\alpha(x^{(n)}, \hat{x}) = \min \left\{ \frac{p(\hat{x})}{p(x^{(n)})} \frac{g(x^{(n)}|\hat{x})}{g(\hat{x}|x^{(n)})}, 1 \right\}$$

Theorem: Under mild regularity conditions, the Markov Chain $\{x^{(n)}\}$ as constructed by the MHA converges and has $p(x)$ as its invariant distribution, independent of the value x_0 chosen to initialize the chain.

The MH algorithm is used to simulate a Markov Chain whose invariant distribution is the desired distribution $p(x)$. However, there is an initial phase when the chain is said to in a transient state, due to the effects of the initial value x_0 chosen. However, after sufficient samples the effect of the starting value diminished and can be ignored. The time during which the chain is in a transient state is referred to as *burn-in* period. This is usually dependent on both the desired function $p(x)$, the proposal function $g(x)$ and most importantly, on the initial state x_0. In most cases, an estimation of this burn in period is very difficult. It is usually easier to make a conservative guess of what it could be. There are also heuristics that estimate the number of burn in samples (say N_b). We shall assume that such an estimate of N_b is available.

4.2 Independant MH Sampling and Particle Filter

Independent MH Sampling is a special case of the general MH algorithm where the proposal function $q(x|y) = q(x)$. This would mean that the acceptance probability, $\alpha(z_1, z_2)$ becomes,

$$\alpha(z_1, z_2) = \min \left\{ \frac{p(z_2)}{g(z_2)} \frac{g(z_1)}{p(z_1)}, 1 \right\} \qquad (7)$$

There is a close resemblance between the IMHA and the SISR algorithm presented in the previous section. Note that the ratio $p(x)/g(x)$ that appears in the accpetance probability is analogous to unnormalized importance weight defined in equation (4) when the desired density $p(x)$ equals the posterior density (of the Bayesian Inference problem) $p(x_t|y_{1:t})$ and the importance function $g(x)$ is the importance function suitably defined.

To begin with, we present an alternative interpretation of particle filtering and modify the underlying theory to bring it in line with the Independent MHA. Particle filtering generates independent (conditionally-independent, because of resampling) tracks in the state space and weighs these tracks to account for both for the posterior density and the importance function. Resampling, destroys the independence of the tracks by deleting tracks with lower weights and replicating tracks with higher weights. The concept of tracks (independent or otherwise) poses a problem in application of MHA to estimate the posterior, because the concept of importance function becomes ambiguous. This is due to the fact that, in a general scenario, each track has an importance function defined with respect to the evolution of the track till the current time instant. Mathematically, the ith track at time t uses an importance function, given as $g(x_t|x_{t-1}^{(i)}y_t)$. This importance function, in nature, is local to the ith track. In contrast, the MHA algorithm requires the importance function to depend functionally only on the last accepted sample in the chain, and in the case of the independent MH Chain the importance function is maintained constant.

To remove the concept of tracks we invoke the *plug-in principle* on the posterior density function at time $t - 1$. Given a set of unweighted samples $\{x_{t-1}^{(i)}, i = 1, \ldots\}$ sampled from the posterior density $p(x_{t-1}|y_{1:t-1})$ at time $t - 1$, we can approximate the posterior by

$$p(x_{t-1}|y_{1:t-1}) \approx \frac{1}{N} \sum_{i=1}^{N} \delta_{x_{t-1}}(x_{t-1}^{(i)}) \qquad (8)$$

where $\delta_{x_{t-1}}(\cdot)$ is the Dirac Delta function on x_{t-1}. Using equations(2,3,8) , we can approximate the posterior at time t,

$$p(x_t|y_{1:t}) \approx \frac{p(y_t|x_t)}{p(y_t|y_{1:t-1})} \frac{1}{N} \sum_{i=1}^{N} p(x_t|x_{t-1}^{(i)}) \qquad (9)$$

The posterior density function $p(x_t|y_{1:t})$, as defined, needs to be sampled (just like particle filtering). This sampling can be performed using MHA. The issue of choice of importance function now arises. The choice of importance function is made when the system is being designed. To generate samples that are similar to those generated by the SISR algorithm we propose an importance function, that is a mixture of the ones used in the SISR algorithm

$$g'(x_t|y_t) = \sum_{i=1}^{N} \frac{1}{N} g(x_t|x_{t-1}^i y_t) \qquad (10)$$

3

Though functionally, the new importance function is different from the one is used in the SISR algorithm, the particles proposed will be identical. This is due to the fact that to sample from $g'(\cdot|y_t)$, we need to first sample $I \backsim \mathcal{U}[1, 2, \ldots, N]$, and then sample from $g(\cdot|x_{t-1}^I y_t)$.

The acceptance probability now takes the form

$$\alpha(x_t, \hat{x}) = \min\left\{\frac{w'(\hat{x})}{w'(x_t)}, 1\right\} \tag{11}$$

$$w'(x_t) = p(y_t|x_t)\frac{\sum_{i=1}^{N} p(x_t|x_{t-1}^{(i)})}{\sum_{i=1}^{N} g(x_t|x_{t-1}^{(i)} y_t)} \tag{12}$$

We can now avoid the resampling step defined in traditional PF algorithms. The intuition behind such a algorithm is that we will use an independent MH sampler to generate unweighted particle set/stream from the desired posterior.

The main details of the algorithm is presented below.

1. Given unweighted particle set, S_{t-1} sampled from the posterior at time $t - 1$, we need to obtain a particle set sampled from the posterior at time t.

2. Generate $N + N_b$ indices $I_i, i = 1, \ldots, N + N_b$ uniformly from the set $\{1, 2, 3, \ldots, N\}$, where N_b is an estimate of the *burn in* period and N is the number of particles required. This essentially implies that we generate $N + N_b$ random number between $1..N$ with uniform density.

3. From the particle set $S_{t-1} = \{x_{t-1}^{(i)}, i = 1, \ldots, N\}$ at time $t - 1$, propose $N + N_b$ particles $\hat{S}_t = \{\hat{x}_t^{(i)}, i = 1, \ldots, N + N_b\}$ using the rule:
$$\hat{x}_t^{(i)} \backsim g(\cdot|x_{t-1}^{I_i} y_t)$$
This essentially implies that we randomly select one of the x_{t-1} particles (denoted by $x_{t-1}^{I_i}$). Using this, the proposal function g and the observation y_t we propose the new particle x_t.

4. For each particle in $\hat{S}_t$, evaluate the weight $w'_t^{(i)}$, for each i using equation (12).

5. **Inference:** Estimate expected value of functions of interest (say $f(\cdot)$). *Note that burn in does not affect the inference as the unnormalized particle set $\{x_t^{(i)}, \tilde{w}_t^{(i)}, i = 1, \ldots, N + N_b\}$ is still properly weighted.* Compute
$$E_p[f(x_t)] = \frac{\sum_{i=1}^{N+N_b} f(x_t^{(i)})\tilde{w}_t^{(i)}}{\sum_{i=1}^{N+N_b} \tilde{w}_t^{(i)}}$$

6. **MCMC based Discriminator:** Use the Independent MH sampler to parse through the set $\hat{S}_t$. While parsing, generate a random number with uniform density between 0-1. Then calculate the acceptance probability using equations (11,12) between the current accepted particle and the particle under consideration. If the acceptance probability is more than the random number then accept the new particle as the current particle else reject it and goto the next particle. In the end we will have a sequence of $N + N_b$ accepted particles. Choose the last N of these as unweighted samples from $p(x_t|y_{1:t})$.

The advantage of this algorithm is that it helps us avoid the expensive resampling step that SISR based implementation would need. Moreover, as will be illustrated later, this formulation of particle filtering is also flexible.

5 MCMC Based Hardware Architecture

Implementing particle filtering in hardware comes at the price of flexibility. In a particle filtering sense, flexibility is desired in the following sense. Several times, we would like to be able to propose more than N particles for generating the posterior density. Since Particle Filtering is a Monte Carlo simulation based technique, having more particles means better inference. Having hardware implementations implicitly forces us to give up this flexibility. As would be illustrated later, this is not the case in the hardware architecture based on MSA. Our architecture can trivially consider proposition of more than N particles and choosing best N out of those.
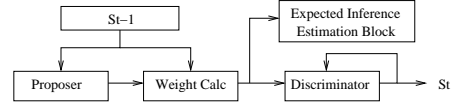

Figure 2. Sequential Architecture

5.1 Sequential Architecture

Figure 2 illustrates the basic architecture assuming the parallelism in particle proposition is not exploited.

Proposal Block: The proposal block essentially takes the N particles from the previous iteration and proposes new particles x_t by sampling the density function $g(x_t|x_{t-1}, y_t)$. Essentially there is a uniform random number generator that randomly selects x_{t-1} and uses it to propose x_t. The proposition block is essentially a hardware implementation of this density function g. We can assume that this proposition block proposes one particle at a time.

Weight Calculator: This block essentially is an implementation of equation 12 and consists of a sequence of adders and a divider. Note that particle proposition block and the weight computation block can be pipelined.

Discriminator: The Discriminator block essentially is an implementation of equation 11 in which the parameter α is calculated for the new particle x_t and the previous best particle. A random-number is generated and if it is smaller than α then the new particle is chosen as the current best else it is discarded.

Expected Inference Estimation Block: This block estimates the inference function (essentially equation 1). This can also be pipelined with proposal and weight computation block.

The characteristics of this basic architecture are as follows

1. Sequential: It proposes one particle at a time and therefore needs to be sequentially executed for generating N particles. The discriminator block must process one particle at a time (Markov Chain) therefore it evaluates all particles sequentially. Note that, if we need to generate N particles to represent the posterior density, then we will have to iterate this architecture $N + N_b$ times where N_b is the statistical burn-in period. After the burn in period, we generate N new particles. The previous N best particles selected by the discriminator would represent samples of the posterior density.

2. Flexible:On several occasions, we would be interested in proposing $Q \geq N$ particles and choosing N particles to carry over to the next iteration. This helps in improving the inference (equation 1). The presented architecture can trivially be extended, if we need to process Q particles and select N best of those. We keep proposing $Q + N_b$ particles and select the last N chosen by the discriminator. Therefore, even though this particle filtering scheme has been implemented in hardware, it is flexible enough to process more than N particles.

3. Resampling in a traditional sense is not needed

This basic sequential architecture can be made faster by pipeline. This degree of pipelining certainly depends on the nature of the importance function g, the state transition function $p(x_t|x_{t-1})$ which in-turn depend on the application. Next, in order to exploit the parallelism in proposal of particles, we present another architecture which is a refinement of the sequential architecture.

5.2 Parallel Architecture

Figure 3 illustrates the parallel implementation of the architecture. In this architecture, we have replicated the sequential architecture several times. Each of the individual sequential architecture works independently. Let us suppose we need to obtain N samples from the posterior. If we have replicated the sequential architecture P times then each one would be used to obtain N/P samples. The only difference lies in the fact that each sequential architecture will have its own statistical burn-in period, therefore the overall latency for each sequential architecture would be $N_b + N/P$. Note that this architecture is also flexible when it comes to processing $Q \geq N$ particles for improved expected inference.
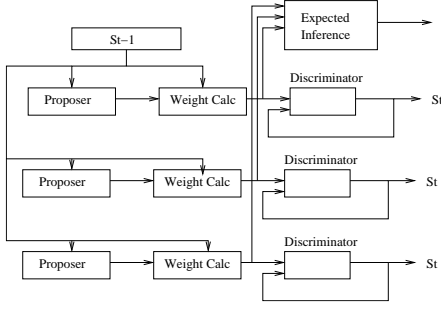
Figure 3. Parallel Architecture

5.3 Pipelining and Latency Analysis

The basic sequential and parallel architectures presented above can be optimized further for speed by using techniques like pipelining. This pipelining depends upon the nature of the proposal block, the weight computation block and the discriminator. Let us suppose that the target application is such that the proposal block can generate one particle every T_p clock steps with an initial latency of K_p. The weight computation block generates the weight of a particle in K_w clocks (latency) and has a throughput of T_w. The Discriminator block is a purely sequential system that depends on the computation of the previous step. Therefore its throughput and latency must be the same $= T_D$. Based on these parameters, and also on the assumption that the latency of particle filtering is not constrained by the expected inference block (and therefore ignored in this analysis), we present an analysis on how fast the MHA based hardware architecture can be made. In this sub-section, this analysis is presented for the sequential architecture only. The analysis for the parallel architecture would follow trivially.

Given the parameters, let us suppose would would like to process Q particles. Since there is a burn in associated, the total number of particles that we need to process is $N_b + Q$. The basic architecture in figure 2 will take $K_p + K_w + T_D$ to produce the first particle x_t. Thereafter, it will be able to produce one particle every $\max(T_D, T_p, T_w)$ clock steps. The total latency for generating $N_b + Q$ particles would be $(N_b + Q - 1)\max(T_D, T_p, T_w) + K_p + K_w + T_D$ clock steps.

This can be made faster by replicating the proposition block and the weight calculation block. We cannot replicate the discriminator since all particles need to be processed sequentially. In order to evaluate such a replication procedure we make the following evaluations.

Infinite Resources Assumption Assuming that the number of resources are infinite. Therefore we can generate the $Q + N_b$ particles and the associated weights in just on step of $K_p + K_w$ clocks. The infinite resources does not affect affect the discriminator since it has to compare all particles sequentially. Therefore the total latency in the infinite resource assumption is

$$K_p + K_w + (Q + N_b)T_D \quad (13)$$

This is the fastest speed that this particular implementation of particle filtering scheme can obtain.

Finite Resource Constraints

As indicated earlier, replicating the proposal and the weight computation blocks can lead to speeding up of the particle filtering scheme. But this replication must be done judiciously. In this discussion we analyze such a resource constrained scenario. In order to make such an analysis, we make the following assumptions.

1. The proposition block and the weight calculation block must be replicated in tandem, i.e. if we have two proposition blocks we must have two weight computation blocks, one dedicated for the other

2. Each proposition and weight calculation pair will therefore have an initial latency of $K = K_w + K_p$ and a throughput of $T = \max(T_p, T_w)$ for generating a particle and the associated weight

3. We replicate this pair R times. Therefore our proposal and weight computation hardware generates R particles per T clock steps with an initial latency of K.

4. The discriminator block stays the same. We also assume that the buffer space in between these blocks (which enables data transfer) is not a constraint

Under these assumptions, we would like to determine the smallest value of $R \geq 1$ such that the overall latency of processing $N_b + Q$ particle is equal to equation 13 (the fastest speed). Since we have replicated the basic proposal-weight calculation par R times, we get R particles every T clock steps with an initial latency of K clocks. In order to generate the weight of $Q + N_b$ particles, we would take $K + ((Q + N_b - R)/R)T$ clocks. The discriminator processes one particle every T_D clock steps. After the first batch of R particles are generated it takes $T_D * R$ clocks to process them. If after this latency a new batch of particles has arrived then it starts processing those, else it waits for the next batch. Therefore, it generates two scenarios

$T_D * R \leq T$ This case implies that when a batch of R particles has been processed, the discriminator must wait for a new batch to arrive. In this case, the discriminator is not the bottleneck. Therefore the total latency for processing $Q + N_b$ particles is $K + ((Q + N_b - R)/R)T + R * T_D$. Here $K + ((Q + N_b - R)/R)T$ is the latency for proposing and calculating the weight for $Q + N_b$ particles. While these particles are being generated, the discriminator is working in parallel. Therefore in the end the discriminator takes $R * T_D$ to process the last batch of R particles. All others have already been processed. The smallest value of this latency happens when $R = T/T_D$. At this point the value of this latency is $K + (Q + N_b)T_D$ which is the fastest that can be achieved (in the infinite resource case).

$T_D * R > T$ In this case the latency is decided by the discriminator. This scenario implies that every time the discriminator *finishes up* processing R particles, new particles are already waiting. Note that this scenario can happen in two situations, 1) $T_D > T$ and 2) $T_D \leq T$.

Case 1 $T_D > T$ If the discriminator takes more clock steps to process one particle than it takes for the proposal-weight computation pair to generate one (essentially T clocks) then any form of replication is wasteful and R should be 1. In this case the total latency is $(N_b + Q - 1)\max(T_D, T_p, T_w) + K_p + K_w + T_D$ (as indicated earlier). Since $T_D > T$, where $T = Max(T_p, T_w)$, the latency becomes $K + (Q + N_b)T_D$ which is the fastest speed.

Case 2 $T_D \leq T$ In this scenario, it takes the discriminator faster to process a particle than the proposal-weight computation pair to generate one. But $T_D * R > T$ due to the choice of R. This scenario ensures that every time the discriminator *finishes up* processing R particles, new particles are already waiting. Therefore the total latency of processing $Q + N_b$ particles is decided completely by the discriminator. Thus the overall latency is simply $K + (Q + N_b)T_D$ (the fastest speed). Interestingly this latency is independent of R. All we need to do is choose an R such that $T_D * R > T$. The smallest value of R that ensures this is $R = T/T_D$ which is the same as the case when $T_D * R \leq T$.

In conclusion we can state that the sequential architecture gives the fastest speed of $K_w + K_p + (Q + N_b)T_D$ if

1. If $T_D > T$ then R = 1

2. Else R $= T/T_D$

Note that this analysis assumes a sequential architecture. Since the parallel architecture is nothing but a replication of sequential machines, this analysis can be trivially extended to the parallel case. We omit it for brevity. This ends the discussion on the architecture which avoids resampling and is also flexible.

6 Experimental Results

Through the experimental results we intend to illustrate 1) One possible application of the particle filtering scheme 2) Compare the quality of result obtained by the SISR and the MHA based approaches

6.1 An Application

We demonstrate the efficiency of our proposed architecture on the bearings-only tracking problem. The problem is that of estimating the motion of a target based on noisy Direction of Arrival (DOA) observations [10]. The state of the system at time t is given by the three-tuple $x_t = (\theta_t, q_t, \phi_t)$, where θ_t is the estimated DOA at time t, q_t is the ratio of the velocity v_t and range r_t, and ϕ_t is the headings direction (see figure (4)).
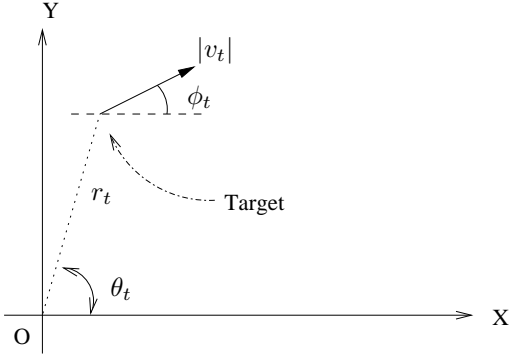
Figure 4. Figure showing the state describing the bearings only tracking problem

Figure 5. Figure shows DOA mean estimates for the MHA and the PF based algorithms.

The observation y_t is a frame of noisy measurement of the state θ_t. Based on a sequence of observations we would like to draw some inference about the current state of the system. As mentioned earlier the posterior density is completely defined with the likelihood $p(y_t|x_t)$, the state transition density $p(x_t|x_{t-1})$ and the initial state density function $p(x_0)$.

The observation model is of the following form (we omit the details and derivations)

$$\log p(y_t|x_t) = K - \frac{(y_t - \theta_t)^T(y_t - \theta_t)}{2\sigma_o^2} \qquad (14)$$

The state evolution happens according to the following equations.

$$
\begin{aligned}
\tan\theta_t &= \frac{\sin\theta_{t-1} + q_{t-1}\sin\phi_{t-1}}{\cos\theta_{t-1} + q_{t-1}\cos\phi_{t-1}} + v_{1,t} \\
q_t &= \frac{q_{t-1}}{\sqrt{1 + q_{t-1}^2 + 2q_{t-1}\cos(\theta_{t-1} - \phi_{t-1})}} + v_{2,t} \\
\phi_t &= \phi_{t-1} + v_{3,t} \\
v_{i,t} &\sim \mathcal{N}(0, \sigma_i^2), i = 1, \ldots, 3
\end{aligned}
\qquad (15)
$$

The state transition density $p(x_t|x_{t-1})$ can now be defined appropriately from equation (15). The prior $p(x_0)$ is assumed to a Gaussian density with mean μ_0 and variance Σ_0.

The importance function $g(x_t|x_{t-1}y_t)$ is chosen to be the state transition density $p(x_t|x_{t-1})$. The choice of such an importance function has a significant effect on the hardware architecture. The computation of the acceptance probability (equations (11,12)) reduces to evaluating the likelihood function $p(y_t|x_t)$.

$$
\begin{aligned}
\alpha(x_t, \hat{x}) &= \min\left\{ \frac{w'(\hat{x})}{w'(x_t)}, 1 \right\} \\
w'(x_t) &= p(y_t|x_t)
\end{aligned}
\qquad (16)
$$

This significantly simplifies the weight computation block. Further, given the exponential nature of the weight, we can save both hardware and time, if we deal with "log" quantities instead. That is compute $\log w'(x_t)$ and $\log w'(\hat{x})$, and subtract the two to get the log of the acceptance probability.

$$\log \alpha(x_t, \hat{x}) = \min\left\{ \log w'(\hat{x}) - \log w(x_t), 0 \right\} \qquad (17)$$

From equation (14), we can compute $\log w'(\hat{x})$ and $\log w'(x_t)$. This usage of logarithms can save us the time required to perform an exponential operator and a division for each particle we process. Note that in order for the Markov Chain to accept/reject a particle it must generate a uniform random number between 0-1 and make its decision based on Step 6 of the algorithm in the section describing MHA. If the weights are computed in the log domain, the uniform random number must also be generated in the log domain.

Figure (5) shows the mean estimates for DOA for the the MHA and the SISR algorithm. Statistically, both have the similar behavior.

7   Conclusion and Future Work

In this paper we present several techniques for simplified VLSI implementation of the particle filtering technique. An interesting course of future work would be to extend it while considering energy limitations.
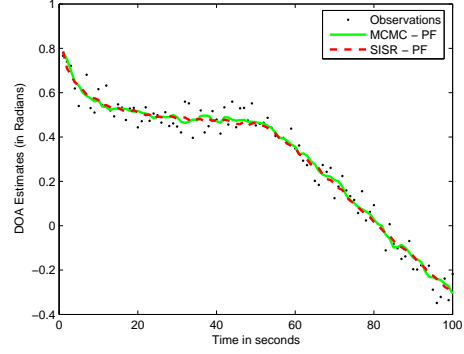
References

[1] S. Chib and E. Greenberg. Understanding the metropolishastings algorithm. In *American Statistician*, volume 49, page 327335, 1995.

[2] A. Doucet, N. Freitas, and N Gordon. Sequential monte carlo methods in practice, new york: Springer-verlag. 2001.

[3] Q. Gang and R. Chellappa. Structure from motion using sequential monte carlo methods. In *International Journal of Computer Vision*, volume 50(1), pages 5–31, 2004.

[4] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions and bayesian restoration of images. In *IEEE Trans. Pattern Analysis and Machine Intelligence*, volume 6, pages 721–741, 1984.

[5] N. Gordon, D. Salmon, and A Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. In *IEE Proceedings*, volume 140, page 107113, 1993.

[6] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. In *Biometrika*, volume 57, pages 97–109, 1970.

[7] M. Isard and A. Blake. Contour tracking by stochastic propagation of conditional density. In *European Conference on Computer Vision*, page 343356, 1996.

[8] R. E. Kalman. A new approach to linear filtering and prediction problems. In *Transactions of the ASME Journal of Basic Engineering, 82*, 1960.

[9] N. Metropolis, A. W. Rosenbluth, M. N Rosenbluth, A. H. Teller, and E. Teller. Equations of state calculations by fast computing machines. In *Journal of Chemical Physics*, volume 21, pages 1087–1091, 1953.

[10] M. Orton and W. Fitzgerald. A bayesian approach to tracking multiple targets using sensor arrays and particle filters. In *IEEE Transactions on Acoustics, Speech and Signal Processing*, volume 50, pages 216–223, February 2002.