Algorithmic and Architectural Optimizations for Computationally Efficient Particle Filtering

Aswin C. Sankaranarayanan, *Student Member, IEEE*, Ankur Srivastava, *Member, IEEE*, and Rama Chellappa, *Fellow, IEEE*

***Abstract*—In this paper, we analyze the computational challenges in implementing particle filtering, especially to video sequences. Particle filtering is a technique used for filtering nonlinear dynamical systems driven by non-Gaussian noise processes. It has found widespread applications in detection, navigation, and tracking problems. Although, in general, particle filtering methods yield improved results, it is difficult to achieve real time performance. In this paper, we analyze the computational drawbacks of traditional particle filtering algorithms, and present a method for implementing the particle filter using the Independent Metropolis Hastings sampler, that is highly amenable to pipelined implementations and parallelization. We analyze the implementations of the proposed algorithm, and, in particular, concentrate on implementations that have minimum processing times. It is shown that the design parameters for the fastest implementation can be chosen by solving a set of convex programs. The proposed computational methodology was verified using a cluster of PCs for the application of visual tracking. We demonstrate a linear speedup of the algorithm using the methodology proposed in the paper.**

***Index Terms*—Auxillary variable, design methodologies, Monte Carlo Markov chain (MCMC), particle filter, resampling, visual tracking.**

I. Introduction

FILTERING is the problem of estimation of an unknown quantity, usually referred to as state, from a set of observations corrupted by noise, and has applications in a broad spectrum of real-life problems including GPS navigation, tracking, etc. The specific nature of the estimation/filtering problem 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. Generally, analytical solutions for estimation are possible in constrained and special scenarios. For example, Kalman filtering [1] is an optimal analytic filter when the models are linear and the corrupting noise processes are Gaussian. For nonlinear systems driven by non-Gaussian processes, the extended Kalman filter or the iterated extended Kalman filter are used as approximations to the optimal filtering scheme. Another popular tool for solving the inference problems for nonlinear systems is particle filtering [2], [3].

Particle filtering has been applied to a wide variety of problems such as tracking, navigation, detection [4], [5] and video-based object recognition. This generality of particle filters comes from a sample (or particle) based approximation of the posterior density of the state vector. This allows the filter to handle both the nonlinearity of the system, as well as the non-Gaussian nature of noise processes. However, the resulting algorithm is computationally intensive and, hence, the need for efficient implementations of the algorithm, tuned specifically towards hardware- or multiprocessor-based implementations.

Many methods for algorithmic and hardware implementations of particle filtering have been proposed in the literature. The authors of [6] identify resampling algorithms as the main computational step in the algorithm that is completely independent of the underlying application. They also propose new resampling algorithms that reduce the complexity of hardware implementations. Architectures for efficient distributed pipelined implementations using FPGAs have been proposed in [7]. A detailed analysis of the basic problem, addressing many hardware and software issues, can also be found in [8] and [9].

The resampling algorithms presented in the above references are modifications of the basic *systematic resampling* algorithm presented in [10], which, by itself, creates bottlenecks in a streamlined implementation. In [11], the authors propose a methodology to overcome this limitation by rederiving the basic theory, with an alternate resampling algorithm which is similar to the Monte Carlo Markov chain (MCMC) tracker for interacting targets in video presented in [12]. There have been a number of resampling schemes that have been proposed in the literature. Liu and Chen [13] list and compare a number of such schemes. Of sufficient interest and relevance are the so-called *local Monte Carlo methods* that are described in [13].

A. Motivation

Specifically, this paper analyzes the computational challenges in the implementations of particle filters, and *provides a general design methodology for particle filtering using pipelining and parallelization*; these are constructs that are commonly used in both hardware and multiprocessor-based systems.

Particle filtering involves three main modules: proposition, weight evaluation, and resampling modules. Standard implementations of particle filtering typically use what is commonly

Manuscript received September 29, 2006; revised January 4, 2008. This work was supported in part by the National Science Foundation-ITR Grant 03-24313 and in part by a Task order from ARL monitored by Alion Science. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Magdy Bayoumi.

A. C. Sankaranarayanan and R. Chellappa are with the Center for Automation Research and the Electrical and Computer Engineering Department, University of Maryland, College Park MD 20742 USA (e-mail: aswch@cfar.umd.edu; rama@cfar.umd.edu).

A. Srivastava is with the Electrical and Computer Engineering Department, University of Maryland, College Park, MD 20742 USA (e-mail: ankurs@glue.umd.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIP.2008.920760

1057-7149/$25.00 © 2008 IEEE

known as *systematic resampling* (SR). Systematic resampling poses a significant challenge for pipelined implementations as it can only begin when all the weights are computed at the weight computation stage, and the cumulative sum of the weights is available. This means that any pipelined implementation would start the resampling only after all the weights are computed. This increases the latency of the whole implementations.

In this paper, we present algorithmic and implementation schemes for particle filters for speeding up the basic computations, thereby making particle filtering-based solutions amenable to real time constraints. We demonstrate a computational methodology where the need for the knowledge of cumulative sum of weights is removed. This implies that, in contrast to traditional particle filtering implementation, the proposed approach does not suffer any bottlenecks in pipelining. Further, this allows us to speedup the filter and reduce its latency through pipelining and parallelization. We further demonstrate the performance of these implementations using a cluster of PCs. This allows us to achieve speedups that are linear in the number of cluster nodes.

B. Specific Contributions

This paper address the computational challenges in hardware and multiprocessor implementations of particle filters. In this regard, we make the following contributions.

1) **Algorithmic Enhancements:** In order to avoid the SR step, we propose the use of the independent Metropolis Hastings algorithm (MHA) [14] for resampling. We show that this algorithmic modification is much more amenable to pipelining and parallelization.
2) **Auxiliary Particle filtering:** Further, we show that many of the problems associated with the proposed methodology can be further reduced with the use of auxiliary particle filters [15]. This allows for complete freedom in the choice of proposal density, which could be an important design issue.
3) **Minimum Time Implementations:** We present pipelineable and parallel architectures for implementing the proposed algorithm. We formulate a set of convex programs for obtaining the design specification of the fastest implementation of the algorithm. We also prove that given a constraint on the execution speed of the algorithm, the minimum resources required for the implementation can be formulated as a convex program.
4) We analyze the pipelining and parallelizability of the proposed implementation using a cluster of PCs for tracking a vehicle in a video stream. We achieve speedups in computation that are linear in the number of cluster nodes.

The rest of the paper is organized as follows. We first present the traditional particle filtering algorithm in Section II. In Section III, we present the MCMC sampling theory and use it to propose a computational methodology in Section IV. Section V analyzes the implementations using the proposed methodology. Finally, in Section VI, we demonstrate the performance of the proposed implementations for the problem of tracking in videos using a cluster of PCs.

II. Particle Filtering

In particle filtering, we address the problem of Bayesian inference for dynamical systems. Let $\mathcal{X} \subseteq \mathbb{R}^d$ and $\mathcal{Y} \subseteq \mathbb{R}^p$ denote the state space and the observation space of the system respectively. Let $x_t \in \mathcal{X}$ denote the state at time t, and $y_t \in \mathcal{Y}$ the noisy observation at time t. We model the state sequence $\{x_t\}$ as a Markovian random process. Further we assume that the observations $\{y_t\}$ to be conditionally independent given the state sequence. Under these assumptions, the system is completely characterized by the following.

- $p(x_t|x_{t-1})$: The *state transition probability density function*, describing the evolution of the system from time $t-1$ to t. Alternatively, the same could be described with a *state transition model* of the form $x_t = h(x_{t-1}, n_t)$, where n_t is a noise process.
- $p(y_t|x_t)$: Observation likelihood density, describing the conditional likelihood of observation given state. As before, this relationship could be in the form of an *observation model* $y_t = f(x_t, \omega_t)$ where ω_t is a noise process independent of n_t.
- $p(x_0)$: The prior state probability at $t = 0$.

Given statistical descriptions of the models and noisy observations, we are interested in making inferences about the state of the system at current time. Specifically, given the observations till time $t, y_{1:t} = \{y_1, \ldots, y_t\}$, we would like to estimate the posterior density function $\pi_t = p(x_t|y_{1:t})$. With the posterior, we aim to make inferences $I(f_t)$ of the form

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

where f_t is some function of interest. An example of such an inference could be the conditional mean, where $f_t(x_t) = x_t$.

Under Markovian assumption on the state space dynamics and conditional independence assumption on the observation model, the posterior probability is recursively estimated using the *Bayes Theorem*

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

Note that there are no unknowns in (2) since all terms are either specified or computable from the posterior at the previous time step. The problem is that this computation (including the integrations) need not have an analytical representation. However, foregoing the requirement for an analytic solution, particle filtering approximates the posterior π_t with a discrete set of particles or samples $\left\{x_t^{(i)}\right\}_{i=1}^N$ with associated weights $\left\{w_t^{(i)}\right\}_{i=1}^N$ suitably normalized so that $\sum_{i=1}^N w_t^{(i)} = 1$. The approximation for the posterior density is given by

$$\hat{\pi}_t(x_t) = \sum_{i=1}^N w_t^{(i)} \delta_{x_t}\left(x_t^{(i)}\right) \tag{3}$$

where $\delta_{x_t}(\cdot)$ is the Dirac Delta function centered at x_t. The set $S_t = \left\{x_t^{(i)}, w_t^{(i)}\right\}_{i=1}^N$ is the weighted particle set that represents the posterior density at time t, and is estimated recursively from S_{t-1}. The initial particle set S_0 is obtained from sampling the prior density $\pi_0 = p(x_0)$.

We first discuss the so called *importance function* $g(x_t|x_{t-1}y_t)$, an easy to sample function whose support encompasses that of π_t. The estimation of $I(f_t)$, as defined in (1) can be recast as follows:

$$\begin{aligned} I(f_t) &= \int f_t(x_t)\frac{p(x_t|y_{1:t})}{g(x_t|x_{t-1}y_t)}g(x_t|x_{t-1}y_t)dx_t \\ &= \int f_t(x_t)w(x_t)g(x_t|x_{t-1}y_t)dx_t \end{aligned} \tag{4}$$

where $w(x_t)$ is defined as the so called *importance weight*

$$w_t = \frac{p(x_t|y_{1:t})}{g(x_t|x_{t-1}y_t)}. \tag{5}$$

Particle filters sequentially generate S_t from S_{t-1} using the following steps.

1) **Importance Sampling:** Sample $x_t^{(i)} \sim g\left(x_t|x_{t-1}^{(i)}y_t\right)$, $i = 1,\ldots,N$. This step is also called the *proposal step* and $g(\cdot)$ is sometimes called the *proposal density*.
2) **Computing Importance Weights:** Compute the unnormalized importance weights $\tilde{w}_t^{(i)}$

$$\tilde{w}_t^{(i)} = w_{t-1}^{(i)}\frac{p\left(y_t|x_t^{(i)}\right)}{g\left(x_t^{(i)}|x_{t-1}^{(i)}y_t\right)}, \quad i = 1,\ldots,N. \tag{6}$$

3) **Normalize Weights:** Obtain the normalized weights $w_t^{(i)}$

$$w_t^{(i)} = \frac{\tilde{w}_t^{(i)}}{\sum_{j=1}^{N}\tilde{w}_t^{(j)}}, \quad i = 1,\ldots,N. \tag{7}$$

4) **Inference Estimation:** An estimate of the inference $I(f_t)$ is given by

$$\hat{I}_N(f_t) = \sum_{i=1}^{N} f_t\left(x_t^{(i)}\right)w_t^{(i)}. \tag{8}$$

This sequence is performed for each time iteration to get the posterior at each time step. A basic problem that the above algorithm suffers from is that, after a few time steps, all importance weights except a few go to zero. These weights will remain to be zero for all future time instants [as a result of (6)], and do not contribute to the estimation of $\hat{I}_N(f_t)$. Practically, this degeneracy is undesirable and is a waste of computational resource. This is avoided with the introduction of a *resampling* step. Resampling essentially replicates particles with higher weights and eliminates those with low weights. This can be done in many ways; [2], [10], and [16] list many resampling algorithms. The most popular one, originally proposed in [2], samples N particles from the set $\left\{x_t^{(i)}\right\}$ (samples generated after proposal) according to the multinomial distribution with parameters $w_t^{(i)}$ to get a new set of N particles $\tilde{S}_t$. The next iteration uses this new set $\tilde{S}_t$ for sequential estimation. We discuss some additional sampling algorithms in Section II-B.

A. Choice of Importance Function

Crucial to the performance of the filter, is the choice of the importance function $g(x_t|x_{t-1}y_t)$. Ideally, the importance function should be close to the posterior. If we choose $g(x_t|x_{t-1}y_t) \propto p(y_t|x_t)p(x_t|x_{t-1})$, then we would obtain the importance weights w_t identically equal to 1 and the variance of the weights would be zero. For most applications, this density function is not easy to sample from. This is largely due to the nonlinearities in the state transition and observation models. One popular choice is to use the state transition density $p(x_t|x_{t-1})$ as the importance function. In this case, the importance weights are given by

$$w_t \propto w_{t-1}p(y_t|x_t). \tag{9}$$

Other choices include using cleverly constructed approximations to the posterior density [17].

B. Resampling Algorithms

In the particle filtering algorithm, the resampling step was introduced to address degeneracies resulting due to the importance weights getting skewed. Among resampling algorithms, the SR technique is popularly used. The basic steps of SR [16] are recounted as follows.

• For $j = 1,\ldots,N$:

1) Sample $J \sim \{1,\ldots,N\}$, such that $Pr[J = i] = a^{(i)}$, for some choice of $\{a^{(i)}\}$.

2) The new particle $\tilde{x}_t^{(j)} = x_t^{(J)}$ and the associated weight is $\tilde{w}_t^{(j)} = w_t^{(J)}/a_t^{(J)}$.

• The resampled particle set is $\tilde{S}_t = \left\{\tilde{x}_t^{(i)}, \tilde{w}_t^{(i)}\right\}_{i=1}^{N}$.

If $a^{(i)} = w_t^{(i)}$, the resampling scheme is the one used in [2]. Other choices are discussed in [16].

Particle filtering algorithms that use sequential importance sampling (SIS) and SR are collectively called SISR algorithms. Computationally, SR is a tricky step, as it requires the knowledge of the normalized weights. *Resampling based on SR cannot start until all the particles are generated and the value of the cumulative sum is known. This is the basic limitation that we overcome by proposing alternative techniques.*

III. Independent Metropolis Hastings Algorithm

In this section, we introduce Monte Carlo sampling techniques, discuss in detail the MHA and its derivative, the independent Metropolis Hastings algorithm (IMHA) [14]. Further, we "redesign" the basic particle filtering algorithm using these techniques for sampling.

Particle filtering is a special case of more general MCMC-based density sampling techniques, specifically suited for dynamical systems. The MHA [18], [19] is considered the most general MCMC-based sampling. Popular samplers such as the Metropolis sampler [20] or the Gibbs sampler [21] are special cases of this algorithm.

The MHA and the particle filter both address the issue of generating samples from a distribution whose functional form is known (upto a normalizing factor) and 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 is computationally more favorable than systematic resampling.

A. Metropolis Hastings Algorithm

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. The MHA generates samples from the desired density (say $p(x), x \in \mathcal{X}$) by generating samples from an easy to sample *proposal distribution*, say $q(x|y)$, $x \in \mathcal{X}$, $y \in \mathcal{X}$. MHA produces a sequence of states $\{x^{(n)}, n \geq 0\}$, which by construction is Markovian in nature, through the following iterations.

1) Initialize the chain with an arbitrary value $x^{(0)} = x_0$. Here, x_0 could be user specified.

2) Given $x^{(n)}, n \geq 0$, generate $\hat{x} \backsim g(\cdot|x^{(n)})$, where g is the sampling or proposal function.

3) Accept $\hat{x}$ with probability $\alpha(x^{(n)}, \hat{x})$ as defined as follows:

$$\alpha(x^{(n)}, \hat{x}) = \min\left\{\frac{p(\hat{x})}{p(x^{(n)})}\frac{g(x^{(n)}|\hat{x})}{g(\hat{x}|x^{(n)})}, 1\right\}. \tag{10}$$

That is, for a uniform random variable $u \backsim U[0,1]$

$$x^{(n+1)} = \begin{cases} \hat{x}, & \text{if } u \leq \alpha(x^{(n)}, \hat{x}) \\ x^{(n)}, & \text{otherwise.} \end{cases} \tag{11}$$

Under mild regularity conditions, it can be shown that 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 [14].

The MHA is used to generate a MCMC whose invariant distribution is the distribution $p(x)$. However, there is an initial phase when the chain is said to be 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 diminishes 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 $q(x|y)$ 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 heuristics that estimate the number of burn-in samples (say N_b). Samples that are in the burn-in period are discarded.

B. Independent Metropolis Hastings Algorithm

The IMHA is a special case of the general MHA where the proposal function $q(x|y)$ is set as $q(x)$. This makes the proposal function independent of the previously accepted sample in the chain. This would mean that the acceptance probability (10) $\alpha(x^{(n)}, \hat{x})$ of a proposal $\hat{x} \in \mathcal{X}$ with the chain at $x^{(n)} \in \mathcal{X}$

$$\alpha(x^{(n)}, \hat{x}) = \min\left\{\frac{p(x^{(n)})}{g(x^{(n)})}\frac{g(\hat{x})}{p(\hat{x})}, 1\right\}. \tag{12}$$

The IMH algorithm has strong convergence properties. Under mild regularity conditions, it has been shown to converge at a uniform rate independent of the value x_0 used to initialize the chain. A study of such convergence properties can be found in [14] and [22].

Both IMHA and SISR are algorithms designed to generate samples according to a probability density function, with the SISR suited specifically to the sequential nature of dynamical systems. In this regard, the key difference between the IMHA and the SISR algorithm lies in the fact that the SISR algorithm requires the knowledge of cumulative sum of weights [the term $\sum_{j=1}^{N} \tilde{w}_t^{(j)}$ in (7)]. This is important as the cumulative sum can only be computed when the weights corresponding to the whole particle set is known. Hence, SR can only begin after all particles are generated and their weights are computed. In contrast, the IMHA poses no such bottlenecks. In Section IV, we exploit this property to design a filter that does not suffer from the bottlenecks introduced by SR.

IV. Proposed Methodology

The bottlenecks introduced by the SR technique can be overcome by using the IMHA for resampling. However, there are some basic issues that needs to be resolved before we achieve this. To begin with, the generation of particles using importance sampling works differently for the two algorithms. Particle filtering allows for the importance function to be defined locally for each particle. Mathematically, the ith particle at time t is generated from an importance function, represented as $g\left(x_t|x_{t-1}^{(i)}y_t\right)$, parametrized by $x_{t-1}^{(i)}$. This poses a problem in the application of IMHA to estimate the posterior, because the concept of importance functions associated with each particle does not extend to IMHA. 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 IMHA, the importance function remains the same.

Given a set of unweighted samples $\left\{x_{t-1}^{(i)}, i = 1, \ldots\right\}$ sampled from the posterior density $p(x_{t-1}|y_{1:t-1})$ at time $t-1$, we can approximate the posterior by

$$p(x_{t-1}|y_{1:t-1}) \approx \frac{1}{N}\sum_{i=1}^{N} \delta_{x_{t-1}}\left(x_{t-1}^{(i)}\right) \tag{13}$$

where $\delta_{x_{t-1}}(\cdot)$ is the Dirac delta function on x_{t-1}. Using (2) and (13), we can approximate the posterior at time t

$$p(x_t|y_{1:t}) \approx \frac{p(y_t|x_t)}{p(y_t|y_{1:t-1})}\frac{1}{N}\sum_{i=1}^{N} p\left(x_t|x_{t-1}^{(i)}\right). \tag{14}$$

Sampling from this density can be performed using MHA or IMHA. The issue of choice of importance function now arises. The importance function typically reflects and exploits the knowledge of application domain or could be a clever approximation to the posterior. For this reason, we would like to reuse the importance function corresponding to the underlying model.

Keeping this in mind, we propose a new importance function of the form

$$g'(x_t|y_t) = \sum_{i=1}^{N} \frac{1}{N} g\left(x_t|x_{t-1}^i y_t\right). \tag{15}$$

Note that $g'(x_t|y_{1:t})$ qualifies to be an importance function for use in IMHA, given its dependence on only one state variable. To sample from $g'(x_t|y_t)$, we need to first sample $I \backsim U[1,2,\ldots,N]$, and then sample from $g\left(\cdot|x_{t-1}^I y_t\right)$. The sampling of I can be done deterministically given the ease of sampling from uniform densities over finite discrete spaces. *Finally, although the new importance function is functionally different from the one used in the SISR algorithm, the generated particles will be identical.*

The overall algorithm proceeds similar to IMHA. We first propose particles using the new importance function $g'(x_t|y_{1:t})$. The acceptance probability now takes the form

$$\alpha(x_t, \hat{x}) = \min\left\{\frac{w'(\hat{x})}{w'(x_t)}, 1\right\} \tag{16}$$

$$w'(x_t) = p(y_t|x_t) \frac{\sum_{i=1}^{N} p\left(x_t|x_{t-1}^{(i)}\right)}{\sum_{i=1}^{N} g\left(x_t|x_{t-1}^{(i)} y_t\right)}. \tag{17}$$

Further, if the choice of the importance function were the same as the state transition model, i.e, $g(x_t|x_{t-1}y_t) = p(x_t|x_{t-1})$, then the acceptance probability becomes a ratio of likelihoods

$$\alpha\left(x_t^{(n)}, \hat{x}\right) = \min\left\{\frac{p(y_t|\hat{x})}{p\left(y_t|x_t^{(n)}\right)}, 1\right\}. \tag{18}$$

We can now avoid the systematic resampling of traditional particle filtering algorithms. The intuition is that we will use IMHA to generate unweighted particle set/stream from the desired posterior.

As before, we have an unweighted particle set S_{t-1}, that contains particles approximating the posterior at time $t-1$, $\pi_{t-1}(x_{t-1})$. We aim to estimate an approximation to the posterior at time t. As before, the algorithm is initialized with S_0 containing samples from the prior $p(x_0)$. The main steps are stated as follows.

• **Importance Sampling (step 1):** Generate $N + N_b$ indices $J(i), i = 1,\ldots,N+N_b$ 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. between $1 \ldots N$ with uniform density.

• **Importance Sampling (step 2):** From the particle set $S_{t-1} = \left\{x_{t-1}^{(i)}, i = 1,\ldots,N\right\}$ at time $t-1$, propose $N+N_b$ particles to form the set $\hat{S}_t = \left\{\hat{x}_t^{(i)}, i = 1,\ldots,N+N_b\right\}$ using the rule

$$\hat{x}_t^{(i)} \backsim g\left(\cdot|x_{t-1}^{J(i)} y_t\right). \tag{19}$$

• **Compute Importance Weights:** For each particle in $\hat{S}_t$, evaluate the importance weights $w'^{(i)}_t$, for each i using (17).

• **Inference:** Estimate the expected value of functions of interest. Compute

$$\hat{I}_t(f_t) = \frac{\sum_{i=1}^{N+N_b} f\left(x_t^{(i)}\right) w'^{(i)}_t}{\sum_{i=1}^{N+N_b} w'^{(i)}_t}. \tag{20}$$

Note that samples discarded during burn-in can still be used in the computation of (20) as the unnormalized particle set $\left\{x_t^{(i)}, w'^{(i)}_t, i = 1,\ldots,N+N_b\right\}$ is still properly weighted (when normalized) [23].

• **MCMC Sampler:** Use the IMH sampler to parse through the set $\hat{S}_t$, to generate a new unweighted set of particles using the following steps.

1) Initialize the chain with $x_t^{(1)} = \hat{x}_t^{(1)}$ the first particle proposed.

2) For $i = 2,\ldots,N+N_b$

$$x_t^{(i)} = \begin{cases} \hat{x}_t^{(i)}, & \text{with prob. } \alpha\left(x_t^{(i-1)}, \hat{x}_t^{(i)}\right) \\ x_t^{(i-1)}, & \text{with prob. } 1-\alpha\left(x_t^{(i-1)}, \hat{x}_t^{(i)}\right) \end{cases} \tag{21}$$

where $\alpha(\cdot,\cdot)$ is the acceptance probability as defined in (10).

Discarding the first N_b samples for burn in, the remaining N samples form $S_t = \left\{x_t^{(i)}, i = N_b+1,\ldots,N\right\}$, the approximation of $p(x_t|y_{1:t})$.

We can now compare the algorithm given above with the classical SISR discussed in Section II. Note that the SISR algorithm involves a weight normalization step (7). However, the proposed algorithm works with ratios of unnormalized weights and requires no such normalization. This allows for the following advantages in the proposed methodology.

- The IMH sampler works with ratios of importance weights. This obviates the need for knowledge of normalized importance weights, as we can work with unnormalized weights. This allows the IMH sampler to start parsing through the particles as they are generated, and not wait for the entire particle set to be generated and the importance weights computed.
- In contrast, in SISR, the resampling can begin only when all particles are generated and the cumulative sum or normalized weights are known.

The ability to resample particles as they are generated allows for faster implementations. This is analyzed further in Section V.

A. Drawbacks of the Proposed Framework

The proposed framework overcomes the drawbacks of the SISR algorithm by adopting an MCMC sampling strategy as opposed to the traditional SR technique. However, the new framework does introduce extra computations that add to increased overall complexity. We discuss these drawbacks and an alternate formulation that can circumvent this issue.

Consider the expression for weight computation, given in (17). The expression involves computing the summations $\sum_{i=1}^{N} p\left(x_t|x_{t-1}^{(i)}\right)$ and $\sum_{i=1}^{N} g\left(x_t|x_{t-1}^{(i)}y_t\right)$, which require additional computation time. The computation of both terms does not present a severe bottleneck, as it can be easily pipelined. Further, when the proposal density matches the state transition model, the terms cancel each other out.

Nonetheless, it is possible to circumvent this problem using the auxiliary particle filtering paradigm [3], [15].

B. Auxiliary Particle Filters

Auxiliary particle filtering refers to techniques that extend the state space of the problem to include a particle index. Consider the new state space $\{x_t, k\}$, where $k \in [1, \ldots, N]$ denotes the particle index. The posterior $p(x_t k|y_{1:t})$ is defined as

$$p(x_t k|y_{1:t}) \propto p(y_t|x_t)p\left(x_t|x_{t-1}^{k}\right). \tag{22}$$

Marginalizing (22) over the state k gives the expression in (14) for $p(x_t|y_{1:t})$.

Let us further assume that we sample the joint space using a proposal $g(x_t k|x_{t-1}y_t)$, i.e, $\left(x_t^{(i)}, k^{(i)}\right) \sim g(\cdot|x_{t-1}y_t)$. The unnormalized weights can be constructed as

$$w_t^{(i)} = \frac{p\left(y_t|x_t^{(i)}\right)p\left(x_t^{(i)}|x_{t-1}^{k^{(i)}}\right)}{g\left(x_t^{(i)}, k^{(i)})|x_{t-1}^{(i)}y_t\right)}. \tag{23}$$

As before, we can resample using an MCMC chain, and the expression for acceptance probability equals the ratio of unnormalized weights as given in (10). At the inference step, we first marginalize across the particle index state k. However, it is easy to see that the marginalization is identical to discarding the particle index information at each particle, given the nature of the particle-based representation of the underlying density. In a nutshell, the use of auxiliary variable allows us to completely avoid the summation of (17) and the associated computational cost.

Finally, there exist many choices for the proposal density in the extended state space. A discussion on this can be found in [15].

V. Implementation Based on Proposed Methodology

In this section, we present approaches for implementing the theory presented in Section III. We assume that the basic computational blocks for importance sampling, computation of importance weight and parsing of particles as per the IMH algorithm are available. We use these blocks to propose three implementations: a sequential implementation and two parallel implementations.

A. Sequential Implementation

Fig. 1 illustrates a straight-forward implementation of the proposed algorithm. It consists of the following blocks.

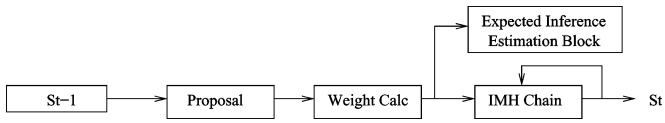


Fig. 1. Sequential implementation.

Proposal Block: The proposal block takes S_{t-1}, the particles from the previous time step and proposes new particles $x_t^{(i)}$ (one particle at a time) by sampling the proposal function. For the IMHA-based algorithm, this amounts to generating a uniform number $J(i) \sim U[1, 2, \ldots, N]$ to randomly pick one particle from S_{t-1}, say $\left\{x_{t-1}^{J(i)}\right\}$. The particle $x_t^{(i)}$ is obtained from sampling $g\left(x_t|x_{t-1}^{J(i)}y_t\right)$. We assume that this block proposes particle one at a time. When we use the auxiliary variable framework, this involves sampling both the state $x_t^{(i)}$ and the associated particle index state $k^{(i)}$ from a proposal function $g(x_t k|x_{t-1}y_t)$.

Weight Calculator: This block is an implementation of (17) [or (23) when we use auxiliary variables].

IMH Chain: This block implements (16) such that the acceptance probability α is calculated for the new particle and the previously accepted particle. Further, an uniform random-number $u \sim U[0, 1]$ is generated and if it is smaller than α then the new particle is retained in S_t, else the last accepted particle in the chain is replicated once more.

Inference Estimation Block: This block estimates the inference function (1). The computation can be performed in parallel with the IMH chain, and has no effect on the overall computation.

The characteristics of this basic implementation are as follows.

- **Sequential Processing of Particles:** Each block in the implementation processes one particle at a time. So, to process Q particles, each block needs to run Q times. Note that, if we need to generate N particles to represent the posterior density, then we will have to iterate $N + N_b$ times where N_b is the burn-in period. The last N particles in the IMH chain is the sample set S_t.
- **Pipelining:** By pipelining the blocks, processing in each block can be made to overlap in time, leading to an overall increase in the throughput of the system.
- **Computation Time:** We now estimate the time required to process $Q = N + N_b$ particles under this implementation. Let us suppose that the target application is such that the proposal block can generate one particle every T_p time units. The weight computation block generates the weight of a particle in T_w time units, and the IMH chain process particles once in every T_d time units. Further, we assume that the overall time required to process is not constrained by the inference block (and, therefore, ignored in this analysis). Under this setting, we can compute the total time required to process Q particles.

The implementation in Fig. 1 will take $T_p + T_w + T_d$ time units to produce the first particle $x_t^{(1)}$. Thereafter, it will be able to produce one particle every $\max(T_d, T_p, T_w)$ time units. The total latency for generating $N_b + N$ particles would be $(N_b + N - 1)\max(T_d, T_p, T_w) + T_p + T_w + T_d$ time units.

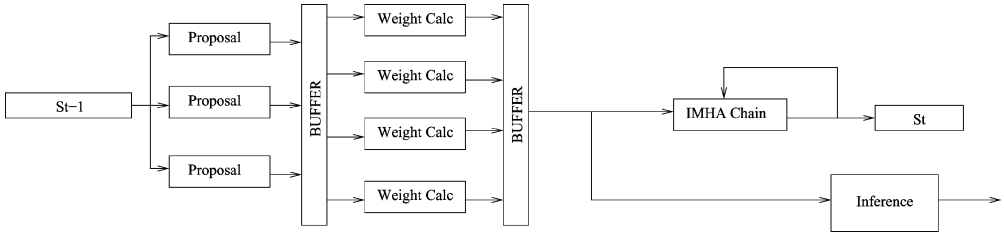


Fig. 2. Parallel implementation with a single IMH chain.

This basic sequential implementation can be made faster by replicating the proposal, weight computation and the IMH chain blocks. In order to exploit the parallelism in processing of particles, we present a refinement of the sequential implementation.

B. Parallel Implementation: Single Chain

Fig. 2 illustrates the parallel implementation of the proposed algorithm. We still retain a single IMH Chain, though the proposal and the weight computation blocks are replicated. Having multiple IMH chains introduces additional issues involving burn-in in each chain. For this reason, we first restrict ourselves to single chain implementations. We relax this restriction later in Section V-C. Let the number of proposal blocks be R_p and the number of weight computation blocks be R_w. We would like to compute the total time required to process Q particles as a function of R_p and R_w (and the latency of the blocks T_p, T_w, and T_d). Further, we would like to choose specific values of R_p and R_w to achieve the smallest total processing time.

The total computational time is determined by bottlenecks in processing created due to differing rates of processing of particles at each stage. The rate at which the proposal blocks process particles is R_p/T_p, the weight computation blocks at R_w/T_w and the IMHA blocks at $1/T_d$. The total computational time is predominantly dependent on which of the three rates is the smallest.

1) Case A: $R_p/T_p \leq R_w/T_w \leq 1/T_d$: In this scenario, the proposal blocks have the smallest rate of processing, followed by the weight computation blocks. Suppose we need to process Q particles, then the proposal blocks by themselves will need $(Q/R_p)T_p$ time units to process all particles. The weight computation and IMHA processing happen in parallel. Given the quicker processing rate at both weight computation and IMHA, by the time the last set of R_p particles is processed at the proposal blocks, all earlier particles have already been processed through the weight computation blocks. The amount of time required to process the last set of R_p particles at the weight computation blocks and the IMHA block is $R_pT_w/R_w + R_wT_d$. Allowing R_p and R_w to take values over the real line (and not just positive integers) the total time for processing τ_A is

$$\tau_A(R_p, R_w) = \frac{Q}{R_p}T_p + \frac{R_p}{R_w}T_w + R_wT_d. \tag{24}$$

We are now interested in computing the values of R_p and R_w that minimize τ_A, keeping in mind that such solutions must satisfy the assumptions of Case A. To begin with, we note that both R_p and R_w take positive values. This allows a natural change of coordinate frames of the form

$$\begin{aligned}\tilde{R}_p &= \log(R_p)\\ \tilde{R}_w &= \log(R_w).\end{aligned} \tag{25}$$

In terms of $\tilde{R}_p$ and $\tilde{R}_w$, the expression for τ_A can be written as

$$\tau_A(\tilde{R}_p, \tilde{R}_w) = QT_P^{-\tilde{R}_p} + T_we^{\tilde{R}_p - \tilde{R}_w} + T_de^{\tilde{R}_w}. \tag{26}$$

The constraints for the minimization come from the assumptions made on the ordering of the rates in Case A

$$\begin{aligned}\tilde{R}_p - \tilde{R}_w - \log\left(\frac{T_p}{T_w}\right) &\leq 0\\ \tilde{R}_w - \log\left(\frac{T_w}{T_d}\right) &\leq 0.\end{aligned} \tag{27}$$

Finally, R_p and R_w are naturally bounded by the value of Q. This leads a convex optimization problem with inequality constraints stated as

$$\min_{\tilde{R}_p, \tilde{R}_w} \tau_A(\tilde{R}_p, \tilde{R}_w) = QT_pe^{-\tilde{R}_p} + T_we^{\tilde{R}_p - \tilde{R}_w} + T_de^{\tilde{R}_w} \tag{28}$$

$$\begin{aligned}\tilde{R}_p - \tilde{R}_w - \log\left(\frac{T_p}{T_w}\right) \leq 0 \quad \tilde{R}_p - \log Q \leq 0\\ \tilde{R}_w - \log\left(\frac{T_w}{T_d}\right) \leq 0 \quad \tilde{R}_w - \log Q \leq 0.\end{aligned} \tag{29}$$

We now note that the expression for τ_A is convex in both $\tilde{R}_p$ and $\tilde{R}_w$. Further, the inequality constraint is also convex in $\tilde{R}_p$ and $\tilde{R}_w$. One can use a host of techniques [24] designed specifically for convex optimization.

2) Case B: $R_w/T_w \leq R_p/T_p$, $R_w/T_w \leq 1/T_d$: Using a line of reasoning identical to Case A, we can derive an expression for the amount of time τ_B needed to process Q particles, as a function of R_p and R_w

$$\tau_B(R_p, R_w) = \frac{R_w}{R_p}T_p + \frac{Q}{R_w}T_w + R_wT_d. \tag{30}$$

We note that a value of R_p greater than R_w is impractical leading to a constraint on R_p of the form $R_p \leq R_w$. As before, we can recast the set of equations in terms of $\tilde{R}_p$ and $\tilde{R}_w$ [as defined in (25)] to get the cost and constraint equations

$$\min_{\tilde{R}_p, \tilde{R}_w} \tau_B(\tilde{R}_p, \tilde{R}_w) = T_pe^{\tilde{R}_w - \tilde{R}_p} + QT_we^{-\tilde{R}_w} + T_de^{\tilde{R}_w} \tag{31}$$

TABLE I
EXPRESSIONS FOR TOTAL TIME TAKEN TO PROCESS Q PARTICLES FOR BOTTLENECKS AT VARIOUS STAGES IN THE PIPELINE

Case	Rate Ordering	Cost	Constraint
A	$R_p/T_p \leq R_w/T_w \leq 1/T_d$	$QT_p e^{-\tilde{R}_p} + T_w e^{\tilde{R}_p - \tilde{R}_w} + T_d e^{\tilde{R}_w}$	$\tilde{R}_p - \tilde{R}_w - \log\left(\frac{T_p}{T_w}\right) \leq 0 \quad \tilde{R}_p - \log Q \leq 0$ $\tilde{R}_w - \log\left(\frac{T_w}{T_d}\right) \leq 0 \quad \tilde{R}_w - \log Q \leq 0$
B	$R_w/T_w = \min(R_p/T_p, R_w/T_w, 1/T_d)$	$T_p e^{\tilde{R}_w - \tilde{R}_p} + QT_w e^{-\tilde{R}_w} + T_d e^{\tilde{R}_w}$	$\tilde{R}_w - \tilde{R}_p - \log\left(\frac{T_w}{T_p}\right) \leq 0 \quad \tilde{R}_p - \log Q \leq 0$ $\tilde{R}_w - \log\left(\frac{T_w}{T_d}\right) \leq 0 \quad \tilde{R}_w - \tilde{R}_p \leq 0$
C	$R_p/T_p \leq 1/T_d \leq R_w/T_w$	$QT_p e^{-\tilde{R}_p} + T_w + T_d e^{\tilde{R}_p}$	$\tilde{R}_p - \log\left(\frac{T_p}{T_d}\right) \leq 0 \quad \tilde{R}_p - \log Q \leq 0$ $-\tilde{R}_w + \log\left(\frac{T_w}{T_d}\right) \leq 0 \quad \tilde{R}_w - \log Q \leq 0$
D	$1/T_d = \min(R_p/T_p, R_w/T_w, 1/T_d)$	$T_p + T_w + QT_d$	$-\tilde{R}_p + \log\left(\frac{T_p}{T_d}\right) \leq 0 \quad \tilde{R}_p - \log Q \leq 0$ $-\tilde{R}_w + \log\left(\frac{T_w}{T_d}\right) \leq 0 \quad \tilde{R}_w - \log Q \leq 0$

$$\tilde{R}_w - \tilde{R}_p - \log\left(\frac{T_w}{T_p}\right) \leq 0 \quad \tilde{R}_w - \log Q \leq 0$$
$$\tilde{R}_w - \log\left(\frac{T_w}{T_d}\right) \leq 0 \quad \tilde{R}_p - \tilde{R}_w \leq 0. \tag{32}$$

Both the cost function and the inequality constraints are convex in $\tilde{R}_p$ and $\tilde{R}_w$.

3) Case C: $R_p/T_p \leq 1/T_d \leq R_w/T_w$: In Case C, the main bottleneck is in the proposal block, followed by the IMH chain. Accordingly, the total time τ_C for processing of Q particles is

$$\tau_C(R_p, R_w) = \frac{Q}{R_p} T_p + T_w + R_p T_d. \tag{33}$$

Using the transformation of variables in (25), we can write down expressions for both the cost τ_C and the constraints

$$\min_{\tilde{R}_p, \tilde{R}_w} \tau_C(\tilde{R}_p, \tilde{R}_w) = QT_p e^{-\tilde{R}_p} + T_w e^{\tilde{R}_p - \tilde{R}_w} + T_d e^{\tilde{R}_w} \tag{34}$$

$$\tilde{R}_p - \tilde{R}_w - \log\left(\frac{T_p}{T_w}\right) \leq 0 \quad \tilde{R}_p - \log Q \leq 0$$
$$\tilde{R}_w - \log\left(\frac{T_w}{T_d}\right) \leq 0 \quad \tilde{R}_w - \log Q \leq 0. \tag{35}$$

As before, both the cost and the inequality constraints are convex over $\tilde{R}_p$ and $\tilde{R}_w$.

4) Case D: $1/T_d = \min(R_p/T_p, R_w/T_w, 1/T_d)$: The final scenario is when the main bottleneck is at the IMH chain. The expression for total time τ_D is given as

$$\tau_D(R_p, R_w) = T_p + T_w + QT_d \tag{36}$$

τ_D is not dependent on the choice of R_p and R_w. So, the whole feasibility set forms the solution set when we optimize for minimum processing time. For completeness, we again formulate it as a convex program with the following cost and constraints:

$$\min_{\tilde{R}_p, \tilde{R}_w} \tau_D(\tilde{R}_p, \tilde{R}_w) = T_p + T_w + QT_d \tag{37}$$

$$-\tilde{R}_p + \log\left(\frac{T_p}{T_d}\right) \leq 0 \quad \tilde{R}_p - \log Q \leq 0$$
$$-\tilde{R}_w + \log\left(\frac{T_w}{T_d}\right) \leq 0 \quad \tilde{R}_w - \log Q \leq 0. \tag{38}$$

As stated above, in Case D, all points in the feasible set form the solution set.

Depending on the exact location of the bottleneck, it is possible to have upto six different scenarios. However, some of these scenarios collapse to identical expressions for the total cost leading to the four cases A through D discussed above. The expressions for the cost and the associated constraints are summarized in Table I. We note that each case results in a convex cost function and convex inequality constraints. This allows us to design an algorithm for determining the global minima for total computation time for processing Q particles given values of T_p, T_w, and T_d.

1) Given values of T_d and T_w, formulate FOUR convex programs associated with the four cases illustrated in Table I.

2) Solve each convex program to obtain minimum times $\tau_{i,\min}, i \in \{A, B, C, D\}$ and associated values of $\tilde{R}_p$ and $\tilde{R}_w$.

3) Choose the configuration that gives the least total processing time.

The above algorithm allows us to obtain design specifications with minimum processing time given values of T_p, T_w, T_d, and Q. Note that the basic computation tools used are optimization techniques for convex programs. Convex optimization is a well studied problem, and there are techniques that solve convex programs very efficiently and reliably [24]. Further, convex programs have very desirable properties with respect to local minima. All local minima are also global minima, and further the set of all local (global) minima form a convex set themselves. Finally, we note that analytic solutions to the convex program are highly dependent on the individual values of Q, T_p, T_w, and T_d.

It is possible that the four convex program may not have unique solutions. Ambiguity in choice of R_p and R_w over the solution set can be resolved, if we have additional considerations such as resource or energy constraints. *It is noted that the set of all solutions to a convex program is also convex* [24]. *This property could be effectively used to design alternate cost functions to resolve the ambiguity in the choice of R_p and R_w.*

C. Parallel Implementation: Multiple Chains

Fig. 3 shows a parallel implementation of the proposed algorithm with multiple IMH chains. The implementation basically replicates the structure proposed in Fig. 2 multiple times. This implementation gives speedup proportional to the number of IMH chains.

Let P be the number of IMH chains. Under this implementation, to generate a set S_t with N particles, each chain would

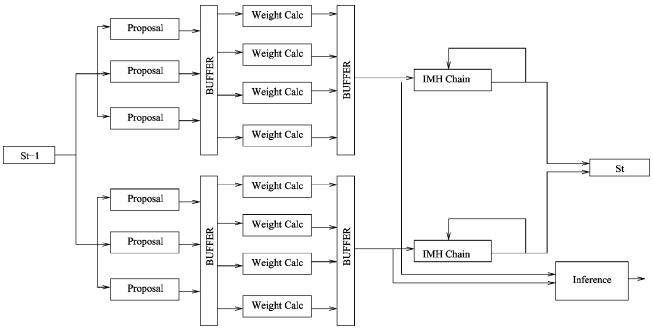


Fig. 3. Parallel implementation with multiple IMH chains.

need to generate only N/P particles, excluding that required for burn in, leading to a total of $N_b + N/P$ particles at each IMH chain. Hence, the time required for obtaining an N-particle set is equal to the time required to process $N_b + N/P$ particles in the implementation as per Fig. 2. With this, we can easily compute the total time required to generate S_t for different scenarios using the same analysis as before, and restricting the total number of particles per IMH chain to $N_b + N/P$.

VI. Experimental Verification

The design methodologies proposed in this paper were verified for two applications: a synthetic example originally discussed in [2] and for the problem of visual tracking. The testbed was the UMIACS red/blue cluster. The red cluster consists of 16 PII (400 MHz) PCs running Redhat 7.3, with each PC having a RAM of 1GB. The blue cluster consists of 12 PIII (550 Mhz). We used MPICH [25], [26], an implementation of the message passing interface (MPI) for communication between threads.

We chose to implement over a multiprocessor cluster framework as the underlying theory applies both to hardware-based design as well as to clusters. In general, MPI has large overheads; however, such overheads are common and identical to both SISR, as well as the MCMC-based schemes. The conclusions from our experimental observations still remain the same.

Further, as mentioned earlier, computation of the burn-in period is a hard problem by itself. However, in sequential estimation, the proposal density is in general a good guess of the posterior. In such cases, the adverse effects of burn-in period are reduced. For the experiments below, we set $N_b = 0$.

A. Visual Tracking

We implemented the particle filter-based online tracking algorithm presented in [27] using the red cluster. We discuss the finer details of the filter and its implementation below.

1) Model Details: We first summarize the tracking algorithm, detailing its computational aspects. A typical tracking example in shown in Fig. 4. The models defining the dynamical system is described as follows.

- **State Space:** The state x_t is a 6-D vector ($\mathcal{X} = \mathbb{R}^6$) defining affine deformations of a rectangular template.
- **State Transition Model:** A simple random walk model with Gaussian noise is used to model the state transition

$$x_t = x_{t-1} + n_t, \quad n_t \sim N(0, \Sigma_n). \tag{39}$$

- **Observation Model:** The frame of the video at time t forms the observation. The likelihood model $p(y_t|x_t)$ involves comparing the appearance model A_t suitably deformed by the state x_t with the observation y_t. The appearance model employs a mixtures of Gaussians with three mixtures to model the appearance. Each value of x_t defines a patch (parallelogram shaped) on the image y_t. Let $z_t = T(y_t; x_t)$ be the patch defined by x_t over y_t. Then

$$p(y_t|x_t) = p(y_t|x_t A_t) = p(z_t|A_t). \tag{40}$$

- **Proposal Density:** The algorithm uses the proposal density to be the same as the state transition model.

2) Implementation Details: An estimate of T_p, T_w, and T_d was first obtained by running each block over a single PC many times over and averaging the individual runs.

- $T_p = 8\ \mu$s, $T_w = 1.1$ ms, $T_d = 1\ \mu$s.

With this, we can see that the main computational bottleneck for this particular application is the evaluation of weights. The weight computation has an unusually large latency, primarily because it involves retrieval from the memory. Given a particle, to compute the weight we need to first obtain the template $z_t^{(i)} = T\left(y_t; x_t^{(i)}\right)$. This involves retrieval of elements from the memory (containing the current frame). Further, the evaluation $p\left(z_t^{(i)}|A_t\right)$ involves evaluation of the mixture of Gaussian, which is far more complicated than the simple proposal and the IMHA blocks. For this reason, we fixed $R_p = R_d = 1$ and analyzed the performance of the architecture for various values of R_w. The implementation of the proposed methodology over the cluster was as follows. Each block (as shown in Fig. 3) was assigned a cluster node for itself, i.e, a total of $R_p + R_w + R_d$ cluster nodes were employed, with R_p of them performing particle proposal, R_w of them computing weights and R_d, the IMH chains. Communication between these PCs was performed using MPICH libraries. Holding the values of R_p and R_w at unity ($R_p = 1 = R_d$), the tracker was tested for varying number of weight computation blocks.

For comparison purposes, we also implemented traditional SISR with the same specifications as the proposed methodology. The main difference was that the node that performed resampling would now wait till all particles are delivered from the weight computation blocks before starting the SR algorithm.

3) Results: The algorithm was used to process 20 frames of a video sequence, tracking a car. Fig. 4 shows typical tracking results. The filter was run with 840 and 1680 particles, with the number of cluster nodes for weight computation R_w varying from 1 to 6. $R_w = 1$ corresponds to the sequential implementation, and $R_w > 1$ corresponds to the parallel implementation with a single chain. Under the same setup, we tried an implementation of SISR, replacing the IMH Chain with a systematic resampler. The main difference between the algorithms is that the systematic resampler could begin only when all particles were processed and the normalized weights are known.

Fig. 5(a) shows the actual time taken (in seconds) to process 20 frames of video, with 840/1680 particles for the proposed algorithm and SISR. Note the $1/x$-like decay exhibited by the time taken by the proposed algorithm. Fig. 5(b) shows the speedup of each algorithm when we add more and more computing nodes. The $1/x$-like behavior now translates to

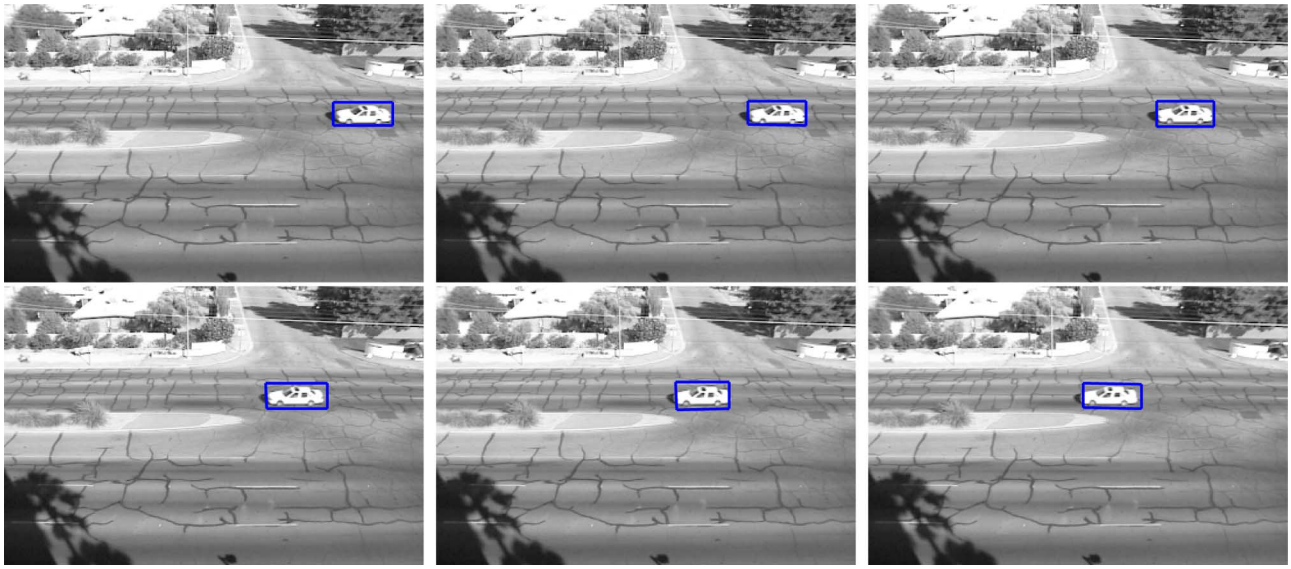


Fig. 4. Frames 1, 4, 8, 12, 16, and 20 of the tested set. The output of the tracker is inlaid on top.

a linear increase in speedup with the number of processing nodes. The two plots demonstrate the pipelinability of the proposed algorithm. It can be seen that the speedup tapers-off as number of cluster nodes increases. This is attributed to increasing communication delays between the nodes. There are no standard models for communication delays when using MPICH. As we use more and more processors, interprocessor communication becomes the dominant source of delay, and further parallelization does not help.

B. Synthetic Example

We applied the design methodology and implementation strategies proposed in this paper for a synthetic example. The problem specifications were first introduced in [2]. The system has a scalar state space, i.e, $\mathcal{X} = \mathbb{R}$. The state transition model is defined by

$$\begin{aligned} x_t &= x_{t-1} + \frac{25x_{t-1}}{1+x_{t-1}^2} + 8\cos(2(t-1)) + w_t \\ w_t &\sim N(0,10). \end{aligned} \tag{41}$$

The observation model is given by the equation

$$y_t = \frac{1}{20}x_t^2 + v_t, \quad v_t \sim N(0,1). \tag{42}$$

We could then estimate the times $T_p : T_w : T_d$ to be in the proportion 19.2:1:7. For filtering with $Q = 840$ particles, we can now formulate and solve the four convex programs. The convex programs were solved using the Matlab's optimization toolbox. The constraints that are active (the constraints that are satisfied with equality at a feasible point are called *active*) at the minima were noted to give a qualitative interpretation to the result.

Case A: Minimum is achieved with the following two active constraints:

$$\frac{R_p}{T_p} = \frac{R_w}{T_w} = \frac{1}{T_d}. \tag{43}$$

Note that this is the rate balancing condition. The corresponding minimum time is

$$\tau_{A,\min} = QT_d + T_p + T_w. \tag{44}$$

Case B: Again, the minimum is achieved at the boundary with the same active constraints

$$\begin{aligned} R_w &= R_p \\ \frac{R_w}{T_w} &= \frac{1}{T_d}. \end{aligned} \tag{45}$$

This gives us a minimum time of

$$\tau_{B,\min} = T_p + QT_d + T_w. \tag{46}$$

Case C: Note that the cost function is independent of R_w. The minimum is achieved at the following active constraint:

$$\frac{R_p}{T_p} = \frac{1}{T_d} \tag{47}$$

giving a minimum time

$$\tau_{C,\min} = QT_d + T_w + T_p. \tag{48}$$

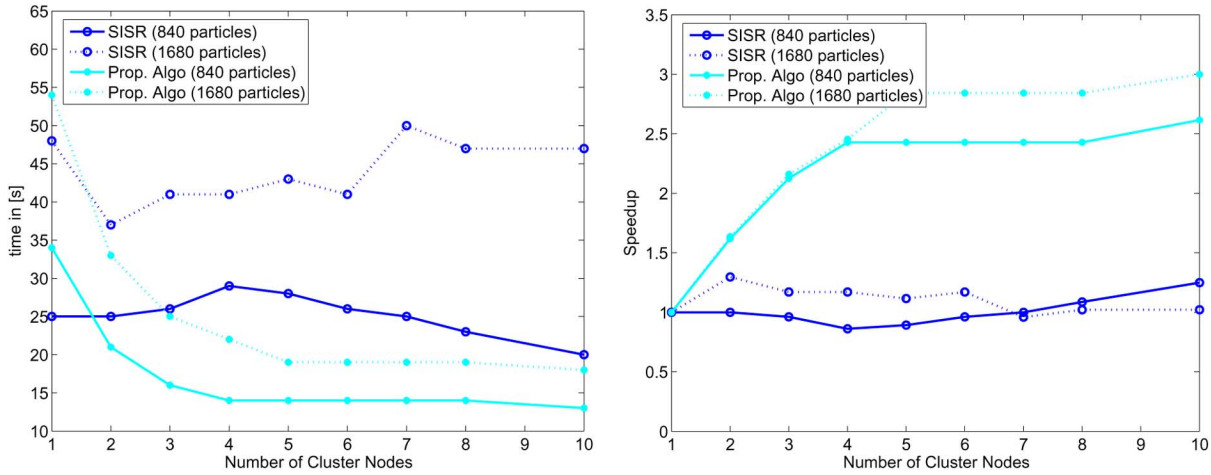
Case D: The cost function is constant over the feasible set. Hence, the minimum time is

$$\tau_{D,\min} = T_p + T_w + QT_d. \tag{49}$$

It turns out that all four convex program give the same minimum time, and this also corresponds to the solution given when the rates are balanced as in (43). This is interesting as balanced rates have an intuitive appeal.

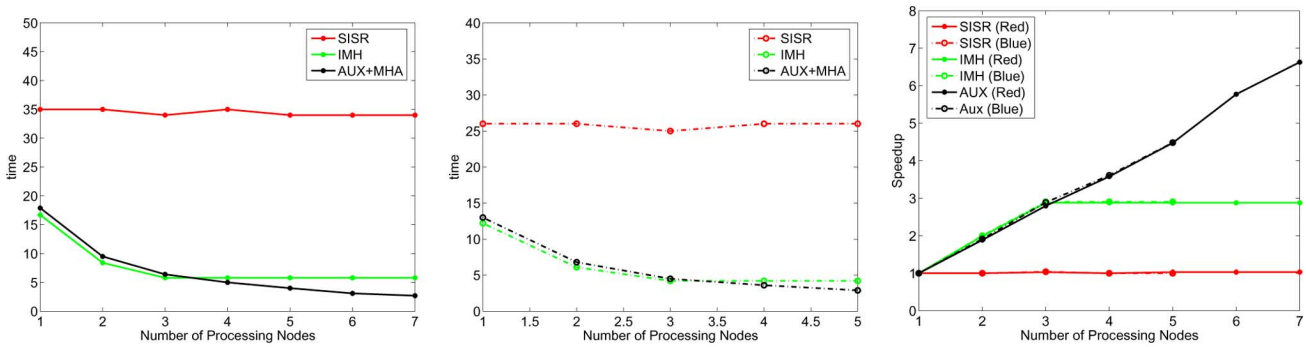
We implemented three filters and tested them on the Red and Blue clusters. The first two filters were those using SISR and IMH for resampling, with the proposal density, being same as the state transition model. The third filter was based on auxiliary particles, with a complicated proposal density defined as follows:

$$\begin{aligned} g(x_t, k|x_{t-1}y_t) &\propto g\left(x_t|x_{t-1}^{(k)}y_t\right) g(k) \\ g(k) &= c \\ g\left(\cdot|x_{t-1}^k y_t\right) &\sim N\left(\pm\sqrt{20|y_t|} + \hat{x}_{t|t-1}^{(k)}, 1\right) \\ \text{where } \hat{x}_{t|t-1}^{(k)} &= x_{t-1}^{(k)} + \frac{25x_{t-1}^{(k)}}{1+\left(x_{t-1}^{(k)}\right)^2} \\ &\quad + 8\cos(2(t-1)). \end{aligned} \tag{50}$$



(a) (b)

Fig. 5. (Left) Actual time (in seconds) taken to process 20 frames, with a filter of 840 particles, with varying number of R_w. (Right) Speedup obtained by replication of the weight computation node. Note the linear speedup obtained with the proposed algorithm. (a) Timing. (b) Speedup.



(a) (b) (c)

Fig. 6. Timing and speedup over the synthetic example. Saturation occurs after $R_p = 3$ because of the shift of the bottleneck from the proposal block to the IMH sampler. (a) Timing (red cluster). (b) Timing (blue cluster). (c) Speedup.

This particular proposal density samples the auxiliary state randomly, and *mixes* the observation with the predicted state to concentrate more particles near the posterior modes.

Fig. 6 shows the actual time for computation and the achieved speedup with parallelization for the three filters, tested on both clusters. We tested the algorithm for varying R_p as the bottleneck is initially in the proposal stage. However, for $R_p > T_p/T_d \approx 3$, the bottleneck shifts to the IMH sampler and further increase in the value of R_p does not produce any significant gains in the overall processing time. This is reflected in the saturation of the plots associated with the proposed algorithm (IMH) in Fig. 6. In contrast, in SISR the resampling begins only when all the particles are generated. The overall time for processing does not scale as well. Finally, auxiliary particle filtering scales linearly with the number of processing nodes, and offers the best speedup.

The resampling method and the associated implementation schemes proposed in the paper allows for a pipeline that is free of bottlenecks. Further, implementations using the proposed methodologies show a speedups that increases linearly with the number of processing nodes utilized. This allows for us to parallelize the algorithm to achieve the desired runtime rate. In contrast, implementations based on SISR do not scale that easily with the number of the processing nodes used.

VII. Conclusion

In this paper, we address the computational challenges in implementing particle filters. We provide a methodology that uses the Independent Metropolis Hastings sampler. It is shown that the traditional bottleneck introduced by the systematic resampler is removed. This allows for a bottleneck-free pipelined implementation. The proposed algorithm works independent of the underlying application. Further, by using the auxiliary filter paradigm, we obtain an alternate design that does not suffer (in complexity) in the presence of arbitrary proposal function. Finally, a set of convex programs is used to compute the design specifications in terms of resources employed in each stage of processing to achieve the minimum time required to process a certain number of particles. We validate our propositions using a cluster of PCs for the problem of visual tracking and show that implementations of the proposed methodology achieve speedup that is linear with the number of processing elements.

References

[1] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Trans. ASME J. Basic Eng.*, vol. 82, 1960.

[2] N. Gordon, D. Salmon, and A. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation," *Proc. Inst. Elect. Eng. F, Radar Signal Process.*, vol. 140, pp. 107–113, 1993.

[3] A. Doucet, N. Freitas, and N. Gordon, *Sequential Monte Carlo Methods in Practice*. New York: Springer-Verlag, 2001.

[4] M. Isard and A. Blake, "Contour tracking by stochastic propagation of conditional density," in *Proc. Eur. Conf. Computer Vision*, 1996, pp. 343–356.

[5] Q. Gang and R. Chellappa, "Structure from motion using sequential Monte Carlo methods," *Int. J. Comput. Vis.*, vol. 50, no. 1, pp. 5–31, 2004.

[6] M. Bolic, P. M. Djuric, and S. Hong, "Resampling algorithms for particle filters: A computational complexity perspective," *EURASIP J. Appl. Signal Process.*, no. 15, pp. 2267–2277, 2004.
[7] M. Bolic, P. M. Djuric, and S. Hong, "Resampling algorithms and architectures for distributed particle filters," *IEEE Trans. Signal Process.*, vol. 53, no. 7, pp. 2442–2450, Jul. 2005.
[8] M. Bolic, "Architectures for efficient implementation of particle filters," Ph.D. dissertation, Dept. Elect. Eng., State Univ. New York, Stony Brook, 2004.
[9] A. Athalye, M. Bolic, S. Hong, and P. M. Djuric, "Generic hardware architectures for sampling and resampling in particle filters," *EURASIP J. Appl. Signal Process.*, vol. 17, pp. 2888–2902, 2005.
[10] A. Doucet, S. Godsill, and C. Andrieu, "On sequential Monte Carlo sampling methods for Bayesian filtering," *Statist. Comput.*, vol. 10, pp. 197–208, 2000.
[11] A. C. Sankaranarayanan, R. Chellappa, and A. Srivastava, "Algorithmic and architectural design methodology for particle filters in hardware," in *Proc. Int. Conf. Computer Design*, 2005, pp. 275–280.
[12] Z. Khan, T. Balch, and F. Dellaert, "MCMC-based particle filtering for tracking a variable number of interacting targets," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 11, pp. 1805–1819, Nov. 2005.
[13] J. S. Liu and R. Chen, "Sequential Monte Carlo methods for dynamic systems," *J. Amer. Statist. Assoc.*, vol. 93, pp. 1032–1044, 1998.
[14] C. P. Robert and G. Casella, *Monte Carlo Statistical Methods*. New York: Springer-Verlag, 1999.
[15] M. K. Pitt and N. Shephard, "Filtering via simulation: Auxiliary particle filters," *J. Amer. Statist. Assoc.*, vol. 94, no. 446, pp. 590–591, 1999.
[16] J. S. Liu, R. Chen, and T. Logvinenko, "A theoretical framework for sequential importance sampling with resampling," in *Sequential Monte Carlo Methods in Practice*, A. Doucet, N. de Freitas, and N. Gordon, Eds. New York: Springer-Verlag, 2001.
[17] A. Doucet, "On sequential simulation-based methods for Bayesian filtering," Tech. Rep., Dept. Eng., Univ. Cambridge, Cambridge, U.K., 1998.
[18] S. Chib and E. Greenberg, "Understanding the Metropolis Hastings algorithm," *Amer. Statist.*, vol. 49, pp. 327–335, 1995.
[19] W. K. Hastings, "Monte Carlo sampling methods using Markov chains and their applications," *Biometrika*, vol. 57, pp. 97–109, 1970.
[20] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equations of state calculations by fast computing machines," *J. Chem. Phys.*, vol. 21, pp. 1087–1091, 1953.
[21] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distributions and Bayesian restoration of images," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-6, no. 6, pp. 721–741, Nov. 1984.
[22] K. L. Mengerson and R. L. Tweedie, "Rates of convergence of the Hastings and Metropolis algorithms," *Ann. Statist.*, vol. 24, no. 1, pp. 101–121, Feb. 1996.
[23] H. Tjelmeland, "Using all Metropolis-Hastings proposals to estimate mean values," Tech. Rep., Dept. Math. Sci., Norwegian Univ. Sci. Technol., 2004.
[24] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York: Cambridge Univ. Press, 2004.
[25] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A high-performance, portable implementation of the MPI message passing interface standard," *Parallel Comput.*, vol. 22, no. 6, pp. 789–828, Sep. 1996.
[26] W. D. Gropp and E. Lusk, *User's Guide for* `mpich`*, a Portable Implementation of MPI*, ANL-96/6, Math. Comput. Sci. Div., Argonne Nat. Lab., 1996.
[27] S. Zhou, R. Chellappa, and B. Moghaddam, "Visual tracking and recognition using appearance-adaptive models in particle filters," *Trans. Image Process.*, vol. 13, no. 11, pp. 1434–1456, Nov. 2004.

Aswin C. Sankaranarayanan (S'04) received the B.Tech. degree from the Indian Institute of Technology, Madras, in 2003. He is currently pursuing the Ph.D. degree at the Department of Electrical and Computer Engineering, University of Maryland, College Park.

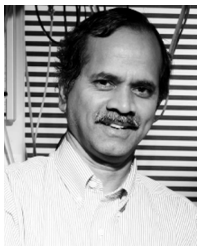
His research interests are in computer vision, statistics, geometry, and signal processing.

Mr. Sankaranarayanan was selected as a Future Faculty Fellow in the A. J. Clarke School of Engineering in Spring 2007. He was also a participant in IBM's Emerging Leaders in Multimedia Workshop held at the T. J. Watson Research Center in October 2007.

Ankur Srivastava (M'02) received the B.S. degree from the Indian Institute of Technology, Delhi, the M.S. degree from Northwestern University, Evanston, IL, and the Ph.D. degree from the University of California, Los Angeles (UCLA), in 1998, 2000, and 2002, respectively.

Since Fall 2002, he has been an Assistant Professor with the University of Maryland, College Park. His research interests include many aspects of design automation including logic and high-level synthesis, low-power issues, fabrication variability, and low-power issues pertaining to sensor networks and computer vision.

Dr. Srivastava received the outstanding Ph.D. Award from the Computer Science Department of UCLA in 2002. He is a Member of ACM.



Rama Chellappa (S'78–M'79–SM'83–F'92) received the B.E. (Hons.) degree from the University of Madras, Madras, India, in 1975 and the M.E. (Distinction) degree from the Indian Institute of Science, Bangalore, in 1977. He received the M.S.E.E. and Ph.D. degrees in electrical engineering from Purdue University, West Lafayette, IN, in 1978 and 1981, respectively.

Since 1991, he has been a Professor of electrical engineering and an affiliate Professor of Computer Science at the University of Maryland, College Park. Recently, he was named the Minta Martin Professor of Engineering. He is also affiliated with the Center for Automation Research (Director) and the Institute for Advanced Computer Studies (permanent member). Prior to joining the University of Maryland, he was an Assistant Professor (1981 to 1986) and an Associate Professor (1986 to 1991) and Director of the Signal and Image Processing Institute (1988 to 1990) with the University of Southern California (USC), Los Angeles. Over the last 27 years, he has published numerous book chapters and peer-reviewed journal and conference papers. He has also coedited and coauthored many research monographs. His current research interests are face and gait analysis, 3-D modeling from video, automatic target recognition from stationary and moving platforms, surveillance and monitoring, hyperspectral processing, image understanding, and commercial applications of image processing and understanding.

Dr. Chellappa has served as an Associate Editor of four IEEE journals. He was a co-Editor-in-Chief of *Graphical Models and Image Processing*. He also served as the Editor-in-Chief of the IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE. He served as a member of the IEEE Signal Processing Society Board of Governors and as its Vice President of Awards and Membership. He has received several awards, including the National Science Foundation Presidential Young Investigator Award in 1985, three IBM Faculty Development Awards, the 1990 Excellence in Teaching Award from the School of Engineering at USC, the 1992 Best Industry Related Paper Award from the International Association of Pattern Recognition (with Q. Zheng), and the 2000 Technical Achievement Award from the IEEE Signal Processing Society. He was elected as a Distinguished Faculty Research Fellow (1996 to 1998), and as a Distinguished Scholar-Teacher (2003) at University of Maryland. He co-authored a paper that received the Best Student Paper in the Computer Vision Track at the International Association of Pattern Recognition in 2006. He is a co-recipient of the 2007 Outstanding Innovator of the Year Award (with A. Sundaresan) from the Office of Technology Commercialization at University of Maryland and received the A. J. Clark School of Engineering Faculty Outstanding Research Award. He was recently elected to serve as a Distinguished Lecturer of the Signal Processing Society and recevied its Meritorious Service Award. He is a Golden Core Member of the IEEE Computer Society and received its Meritorious Service Award in 2004. He is a Fellow of the International Association for Pattern Recognition. He has served as a General Chair and the Technical Program Chair for several IEEE international and national conferences and workshops.