

January 2008

# A parallel color-based particle filter for object tracking

Henry Medeiros

Johnny Park

Avinash Kak

Follow this and additional works at: <http://docs.lib.purdue.edu/ecepubs>

---

Medeiros, Henry; Park, Johnny; and Kak, Avinash, "A parallel color-based particle filter for object tracking" (2008). *Department of Electrical and Computer Engineering Faculty Publications*. Paper 59.  
<http://dx.doi.org/http://dx.doi.org/10.1109/CVPRW.2008.4563148>

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact [epubs@purdue.edu](mailto:epubs@purdue.edu) for additional information.

# A Parallel Color-Based Particle Filter for Object Tracking

Henry Medeiros, Johnny Park, and Avinash Kak  
School of Electrical and Computer Engineering  
Purdue University, West Lafayette, Indiana 47907-2035  
{hmedeiro, jpark, kak}@purdue.edu

## Abstract

*Porting well known computer vision algorithms to low power, high performance computing devices such as SIMD linear processor arrays can be a challenging task. One especially useful such algorithm is the color-based particle filter, which has been applied successfully by many research groups to the problem of tracking non-rigid objects. In this paper, we propose an implementation of the color-based particle filter suitable for SIMD processors. The main focus of our work is on the parallel computation of the particle weights. This step is the major bottleneck of standard implementations of the color-based particle filter since it requires the knowledge of the histograms of the regions surrounding each hypothesized target position. We expect this approach to perform faster in an SIMD processor than an implementation in a standard desktop computer even running at much lower clock speeds.*

## 1. Introduction

As the demand for low-power, portable, networked, and mobile computing devices continues to increase, it is natural that the services provided by such devices grow in number and in complexity. However, due to power consumption constraints, the operating speed of these devices is bounded to be much lower than that of standard desktop computers. To support these new, more complex applications, running in lower clock speed processors, alternative processing architectures are being employed. Since these architectures are fundamentally different from that of the general purpose processors, it is often the case that existing algorithms need to be redesigned in order to be implemented in these systems.

In the specific case of vision systems, object tracking is a building block for a number of applications. As a consequence, many successful approaches have been devised for visual tracking. One such successful approach

is the color-based particle filter. Over the past decade, many research groups have successfully employed the particle filter [2] to track non-rigid objects based on their color histograms [6, 19, 22, 23, 28]. In this approach, a reference histogram of the target is initially provided to the tracker which then searches each subsequent frame for the most likely new location of the target using Bayesian estimation. The results obtained so far by these researchers show that the method is suitable for tracking non-rigid objects since the color histogram is relatively independent of the target deformation and is robust to occlusion and to variations in the color of the background [19, 23].

However, the particle filter is computationally expensive and, therefore, is not suitable for the current generation of wireless smart cameras based on low-power general purpose microcontrollers (e.g. the Cyclops camera [26]). On the other hand, the algorithm lends itself to effective parallel implementation. Therefore, by devising a parallel implementation of the color-based particle filter, we believe that it is possible to achieve robust, real-time object tracking on low-power smart cameras based on an SIMD processor such as the Wica camera [13].

This paper is organized as follows. In section 2, we present some of the works on color-based particle filters and on methods to implement the general particle filter in parallel. In section 3, we describe the basic idea of the particle filter as well as its color-based version for object tracking in image sequences. Section 4 presents our proposed parallel implementation of the algorithm. In section 5, we provide a brief analysis of the potential gains of the algorithm. In section 6, we present a proof-of-concept implementation of our algorithm in a standard desktop computer. Finally, section 7 concludes the paper.

## 2. Related Work

The particle filter was introduced to the computer vision community by Isard and Blake in their semi-

nal work [9] in which they presented the CONDENSATION algorithm, which tracks objects based on their contours. The idea of tracking objects based on their color histograms using the particle filter was suggested by Nummiaro et al. [19] and by Pérez et al. [23] around the same time. In spite of a few minor differences, both works present essentially the same algorithm wherein the measurement likelihood is based on the Battacharyya distance between the current color histograms and the reference color histogram, and the target dynamics are represented by a constant velocity model perturbed by Gaussian noise. Both works present different strategies for initializing the filter. Pérez et al. also presents some extensions such as tracking objects using multiple histograms and introducing a model of the background into the tracker. The general form of the color-based particle filter presented by these works is widely accepted today.

Nummiaro et al. later extended their work in several ways. In [20] they used an adaptive target model wherein the color histogram of the target is slowly updated as the object moves. To update the target histogram, they first threshold the observed state probability to eliminate outliers and then employ a forgetting process so that the contribution of previously estimated states decreases as the state becomes older. They also presented a multi-camera tracker [21] which uses multiple reference histograms corresponding to different views of the target and reinitializes cameras that lose track of the target based on the epipolar geometry.

Pérez et al. also presented many extensions to their initial work. In [24] they included sound and motion cues into the color-based particle filter to increase the robustness of the tracker. They also presented an adaptive target model using both color and movement information in [30].

Currently the use of the color-based particle filter is widespread and a comprehensive survey is beyond the scope of this work. Nonetheless, it is important to mention that an embedded implementation of the color-based particle filter has already been presented in [8], but their work does not consider parallel implementation issues.

Regarding parallel computation of the particle filter, many works have claimed that the particle filter is immediately parallelizable since there are no data dependencies among particles. That is the case indeed for most steps of the particle filter except for resampling. Therefore, most of the works on parallel particle filters focus on designing a resampling step suitable for parallel implementation.

In [18], for example, the authors showed how each of the building blocks of a particle filter, including many known resampling techniques, can be implemented in a

fine-grained parallel architecture in which each processing element is responsible for processing one particle.

Bolic et al. [3, 4] presented techniques to improve the resampling step. After showing that a particle filter with  $K$  particles can be computed in an SIMD machine with  $M$  processing units in  $K/M+L$  steps, where  $L$  is the latency for the first particle to be available, they presented different parallel resampling methods and proposed architectures for effective implementation of these methods.

Kotecha and Djurić devised the Gaussian particle filter [14] with the specific goal of avoiding resampling and, as a consequence, providing a fully parallelizable algorithm. The Gaussian particle filter approximates the posterior distribution by a Gaussian distribution and uses the principle of importance sampling to propagate the estimated mean and covariance of the distribution. As opposed to the extended Kalman filter [31] or the unscented Kalman filter [11], the Gaussian particle filter does not require the process and observation noise distributions to be Gaussian.

Sutharsan et al. [29] proposed an SIMD particle filter for multi-target tracking. Their system uses a distributed resampling method which requires exchange of fewer particles among processors. Considering the communication overhead of transmitting particles among processors, they devised an algorithm that minimizes the computation time by balancing the load (i.e., the number of particles) processed by each processing element.

The main objective of most of the aforementioned works is to parallelize the resampling step. However, for a moderate number of particles, resampling itself is not computationally expensive [3]. The main focus of our work is, therefore, on the computation of the particle weights for the specific case of color-based particle filters. This step is the major bottleneck in the implementation of the filter since it requires the computation of the histograms of the regions surrounding each hypothesized target position.

### 3. The Particle Filter

In a Bayesian framework, object tracking is carried out by modeling the evolution of the state of the target as well as its measurement process by a set of (possibly non-linear) equations perturbed by (possibly non-Gaussian) i.i.d. noise. That is:

$$\mathbf{x}_{k+1} = f_k(\mathbf{x}_k, \mathbf{v}_k) \quad (1)$$

$$\mathbf{z}_k = h_k(\mathbf{x}_k, \mathbf{n}_k) \quad (2)$$

where  $\mathbf{x}_k$  is the state of the target at discrete time  $k$ ,  $f_k(\cdot)$  is the dynamic equation of the target state,  $\mathbf{v}_k$  is the process noise vector,  $\mathbf{z}_k$  is the measurement

vector,  $h_k(\cdot)$  is the measurement function, and  $\mathbf{n}_k$  is the measurement noise vector. In effect, Eqs. (1) and (2) provide respectively the conditional distribution of the state of the target given the previous state and the process noise, and the likelihood of the measurement given the target state and the measurement noise. Our ultimate goal is to estimate the distribution of the state of the target given all the previous measurements, that is,  $p(\mathbf{x}_k | \mathbf{z}_{1:k})$ , where  $\mathbf{z}_{1:k} = \{\mathbf{z}_1, \dots, \mathbf{z}_k\}$ . If the initial distribution of the target is known, it is possible, in theory, to recursively predict the state of the target using:

$$p(\mathbf{x}_{k+1} | \mathbf{z}_{1:k}) = \int p(\mathbf{x}_{k+1} | \mathbf{x}_k) p(\mathbf{x}_k | \mathbf{z}_{1:k}) d\mathbf{x}_k \quad (3)$$

and, as a new measurement becomes available, the state can be updated using Bayes' rule:

$$p(\mathbf{x}_{k+1} | \mathbf{z}_{1:k+1}) = \frac{p(\mathbf{z}_{k+1} | \mathbf{x}_{k+1}) p(\mathbf{x}_{k+1} | \mathbf{z}_{1:k})}{p(\mathbf{z}_{k+1} | \mathbf{z}_{1:k})} \quad (4)$$

When the assumptions of linearity and Gaussian distribution hold, Eqs. (3) and (4) can be solved analytically, yielding the Kalman filter. However, in many practical applications such assumptions do not hold. Therefore, it is necessary to find alternative approaches to solve the equations. The particle filter provides one way of approximately solving the equations in the general case.

In the particle filter, the distribution  $p(\mathbf{x}_k | \mathbf{z}_{1:k})$  is approximated by a set of  $M$  discrete samples  $\{\mathbf{x}_k^i\}_{i=1\dots M}$  and associated weights  $\{w_k^i\}_{i=1\dots M}$ . Then, we have:

$$p(\mathbf{x}_k | \mathbf{z}_{1:k}) \approx \sum w_k^i \delta(\mathbf{x}_k - \mathbf{x}_k^i) \quad (5)$$

where

$$w_{k+1}^i \propto w_k^i \frac{p(\mathbf{z}_{k+1} | \mathbf{x}_{k+1}^i) p(\mathbf{x}_{k+1}^i | \mathbf{x}_k^i)}{q(\mathbf{x}_{k+1}^i | \mathbf{x}_k^i, \mathbf{z}_{k+1})} \quad (6)$$

$$\sum w_k^i = 1 \quad (7)$$

and  $\delta(\cdot)$  is the Kronecker delta function. The term  $q(\mathbf{x}_{k+1}^i | \mathbf{x}_k^i, \mathbf{z}_{k+1})$  in Eq. (6) is an importance density, which is generally obtained by approximating  $p(\mathbf{x}_{k+1} | \mathbf{x}_k, \mathbf{z}_{k+1})$  with a Gaussian distribution, or by simply using  $p(\mathbf{x}_{k+1} | \mathbf{x}_k)$ . By approximating the importance density, it is possible to use Eqs. (5) and (6) to recursively approximate the distribution of the target state.

However, particle filters are subject to a phenomenon known as sample degeneration in which the number of particles with significant weights decreases in each iteration of the filter, and, eventually, only a very small number of particles represents the posterior distribution. To mitigate this problem, particle filters employ, in general, a technique called resampling. This

technique consists basically in generating a new set of particles by resampling the current particles based on their weights. Resampling can be carried out at every filter iteration or only when a significant amount of degeneracy is observed.

### 3.1. Color-Based Particle Filter

In the color-based particle filter, the measurements are the color histograms of the target object (usually in the HSV space to reduce sensitivity to illumination changes). These histograms, when normalized, provide the color distribution of a region in the image around the target. Generally, the color distribution is computed so that pixels farther from the center of the region, which are more prone to occlusion, are given less weight than pixels near the center. Assuming  $m$ -bins histograms, and letting the color distribution of a region centered at a pixel location  $\mathbf{y}$  be  $p(\mathbf{y}) = \{p(\mathbf{y})^i\}_{i=1\dots m}$ , one common approach to compute  $p(\mathbf{y})$  is [5]:

$$p(\mathbf{y})^i = C \sum_{\mathbf{u} \in R(\mathbf{y})} w(|\mathbf{y} - \mathbf{u}|) \delta[h(\mathbf{u}) - i] \quad (8)$$

where  $C$  is a normalization constant so that  $\sum p(\mathbf{y})^i = 1$ ,  $R(\mathbf{y})$  defines the region where the histogram is being computed,  $w(\cdot)$  is a monotonically decreasing function, and  $h(\mathbf{u})$  is a function that assigns a histogram bin to the color vector at pixel location  $\mathbf{u}$ .

The measurement likelihood is computed based on the distance between the measured target histogram and the reference histogram of the target. One common approach to compute the distance between the color distributions is to use the Battacharyya distance [19, 23]:

$$d[p(\mathbf{y}), \mathbf{p}_0] = [1 - \rho(p(\mathbf{y}), \mathbf{p}_0)]^{\frac{1}{2}} \quad (9)$$

where

$$\rho(p(\mathbf{y}), \mathbf{p}_0) = \sum_{i=1}^m \sqrt{p(\mathbf{y})^i p_0^i} \quad (10)$$

is the Battacharyya coefficient between the measured color distribution  $p(\mathbf{y})$  and the reference color distribution  $\mathbf{p}_0$ . The measurement likelihood is then given by a function of the Battacharyya distance, such as [23]:

$$p(\mathbf{z}_k | \mathbf{x}_k) \propto \exp - \lambda d^2[p(\mathbf{y}), \mathbf{p}_0] \quad (11)$$

Since we have a model of the target movement, the likelihood function, given by Eq. (11), and an approximate importance density, we are able to perform the particle filter. Algorithm 1 summarizes this process.

## 4. Parallel Implementation

It has been reported that the bottleneck in the implementation of the color-based particle filter is the computation of the  $M$  color distributions at each step

---

**Algorithm 1** Color-based particle filter.

---

Current particle set:  $\{\mathbf{x}_k^m\}_{m=1\dots M}$

Predict the new particle set by sampling  $M$  particles from the dynamic model – Eq. (1)

**For** each predicted sample  
    compute the color distribution – Eq. (8), the likelihoods  
    Eq. (11), and the corresponding weights – Eq. (6)

**End For**

Sample  $M$  new particles,  $\{\mathbf{x}_{k+1}^m\}_{m=1\dots M}$ , according to their weights

---

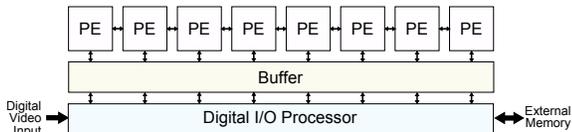


Figure 1. Hardware architecture.

of the algorithm [23]. This bottleneck is due to the fact that, in a general purpose processor, each of the  $M$  histograms has to be computed sequentially. In this paper, we show that, as long as the processor architecture allows for efficient access to external memory, it is possible to compute the histograms in parallel.

#### 4.1. Hardware Architecture

We propose an algorithm for an SIMD linear processor array similar to the Xetal family of SIMD processors [12]. The architecture is composed of  $P$  processing elements each of which with an arithmetic logic unit and a small amount of memory. Each processing element is able to communicate directly with its two nearest neighbors. The processing elements also have read and write access to a buffer which is able to store multiple image lines. A digital I/O processor is responsible for parallelizing data received from the image sensor or from the external memory and store them in the buffer. The digital I/O processor also reads data from the buffer and serializes them to store in the external memory. This architecture is illustrated in Figure 1. The digital I/O processor can operate independently of the processing elements so that, while the processing elements are performing computations on the data, the digital I/O processor may store previously processed data or fetch new data from the image sensor or from the external memory.

#### 4.2. Parallel Histogram Computation

Suppose we want to compute the histogram of a rectangular image region  $R(\mathbf{x})$  of dimensions  $r_x \times r_y$ , that

---

**Algorithm 2** Parallel computation of the integral histograms.

---

**For** each image line  
    compute  $c^i(x, y)$  in parallel for all  $x$  using Eq. (13)

    store  $c^i(x, y)$  in the external memory

**End For**

**For** each column of  $c^i(x, y)$   
    read the  $i^{th}$  column of  $c^i(x, y)$  transposed

    compute the value of  $l^i(x, y)$  in parallel for all  $y$  using Eq. (14)

**End For**

---

is:

$$R(\mathbf{x}) = R(x, y) = \{(u, v) : x \leq u \leq x + r_x, \\ y \leq v \leq y + r_y\} \quad (12)$$

One straightforward approach to compute the histogram of region  $R(\mathbf{x})$  would be to employ integral histograms [25] as follows. Let  $c^i(x, y)$  represent the total count of the  $i^{th}$  histogram bin for pixel  $x$  at column  $y$ . This count can be computed recursively by:

$$c^i(x, y) = \delta[h(x, y) - i] + c^i(x, y - 1) \quad (13)$$

In a linear processor array, Eq. (13) can be computed in parallel for all the values of  $x$ . Letting  $l^i(x, y)$  represent the total count of the  $i^{th}$  histogram bin then we can compute the total count of the  $i^{th}$  histogram bin using:

$$l^i(x, y) = l^i(x - 1, y) + c^i(x, y) \quad (14)$$

If the values of  $c^i(x, y)$  are stored in an external memory, Eq. (14) can be computed in parallel by reading each column of  $c^i(x, y)$  transposed into the memory of the SIMD processor. This procedure is illustrated in Algorithm 2.

Therefore, if the value  $l^i(x, y)$  is stored for each pixel of the image, each bin  $i$  of the histogram of the region  $R(\mathbf{x})$  can be computed by:

$$l^i(\mathbf{x}_4) - l^i(\mathbf{x}_3) - l^i(\mathbf{x}_2) + l^i(\mathbf{x}_1) \quad (15)$$

where  $\mathbf{x}_1 = (x, y)$ ,  $\mathbf{x}_2 = (x + r_x, y)$ ,  $\mathbf{x}_3 = (x, y + r_y)$ ,  $\mathbf{x}_4 = (x + r_x, y + r_y)$ .

The main drawback of this approach, however, is that we need to store one histogram per pixel. Since each histogram consists of a relatively large data structure, the memory requirements of integral histograms are generally too high for embedded systems. For instance, approximately 42 megabits of memory are needed to store the integral histograms of a  $320 \times 240$  pixels image using histograms of 32 bins. This limitation has motivated us to propose a novel approach to compute the histograms which requires temporary storage of only one histogram per image column.

In our approach, we compute the histograms of  $M$  image regions in parallel. To do so, we reorganize the

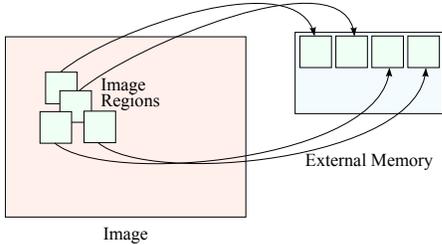


Figure 2. Organization of the histograms in the external memory.

---

**Algorithm 3** Organization of the histograms in the external memory.

---

```

For each region  $R_i$ 
  If  $(x_i < x < x_i + r_x) \ \&\& \ (y_i < y < y_i + r_y)$ 
    store pixel value in the external memory position
    corresponding to  $R_i$ 
  End If
End For

```

---

image regions for which we want to compute the histograms into the external memory side-by-side, as illustrated in Figure 2, so that they can be later processed in parallel. To store the pixels in the external memory, each processing element has to know the initial coordinates,  $y_i$  and  $x_i$ , the dimensions,  $r_x$  and  $r_y$ , and the corresponding initial position in the external memory of each region  $R_i$ . This procedure is illustrated in Algorithm 3, which is executed in parallel for all the elements in one image line.

It is important to note that, although Algorithm 3 requires  $O(M)$  iterations per pixel, these iterations are done during a line read from the image sensor. Therefore, the time required to store the image regions should be negligible.

After the image regions are stored in the external memory, they are read, line-by-line, into the internal memory to be processed in parallel using an approach somewhat similar to that used in [16, 17], which is illustrated in Figure 3. During the first  $r_y$  iterations, the histograms for each column of the regions are computed. This step can be carried out efficiently by employing embedded histogramming functions available in processors such as the Xetal II [1]. After the column histograms are computed, we need  $r_x$  steps to compute the total histograms of each image region. This is done by sequentially adding the histogram of a given column to that of its immediate neighbor.

The procedure to compute the histograms in parallel is illustrated in Algorithm 4. Using this procedure, it is possible to compute the histograms of all the image regions in  $O(r_x + r_y)$  steps. The main bottleneck in this procedure is reading the data from the external memory. Since the external memory has to be read sequentially, we need  $O(n_x)$  operations to read each line of data, where  $n_x$  is the number of elements in

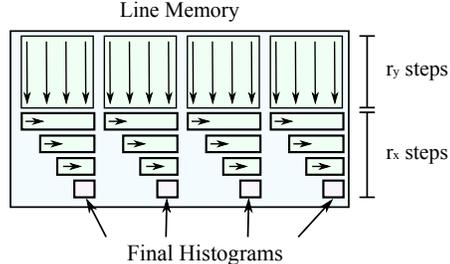


Figure 3. Parallel computation of the histograms.

---

**Algorithm 4** Computing the histograms in parallel.

---

```

For line_counter = 0 To  $r_y$ 
  read the pixel values of the current line into the line memory
  compute the column histogram and store in the line memory
End For
For column_counter = 0 To  $r_x$ 
  If  $(x \bmod r_x) = \text{column\_counter}$ 
    add the current column histogram to the histogram of the
    right neighbor
  End If
End For

```

---

one row of the line memory (which is the same as the number of processing elements since the line memory is basically the memory within the processing elements). This problem can be mitigated if the external memory can be accessed in a pipelined manner. That is, if we allow the digital I/O processor responsible for reading the external memory to read an entire line and store it in a temporary buffer while the linear processor array processes the previous line.

Depending on the size of each line of the line memory and the number of regions to be stored, it may be the case that the line memory cannot store all the image regions side-by-side, i.e.,  $n_x < M \times r_x$ . In that case, it is possible to store the elements in an array as illustrated in Figure 4. Using this arrangement,  $\frac{M \times r_x}{n_x}$  extra steps are necessary to compute the histograms of all the regions. However, since in practice we expect  $r_x \ll n_x$ , there should not be too many extra steps. As an example, if we let  $n_x = 320$ ,  $r_x = 16$ , and  $M = 60$ , then only three steps of 32 iterations are necessary to compute all the histograms.

Evidently, the function  $w(|\mathbf{y} - \mathbf{u}|)$  used in Eq. (8) can also be computed in parallel for each pixel of the image. Since there are no data dependencies among the particles, after the histogram distributions are computed, each likelihood can be computed in parallel as long as the processing elements have access to the common reference histogram. After the likelihoods are computed, the (unnormalized) weights can also be computed in parallel.

As the weights are computed, the remaining steps of the color-based particle filter are very simple and it should be possible to implement them effectively even

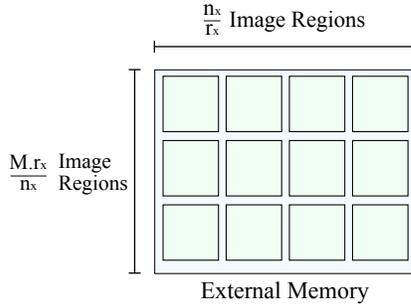


Figure 4. Organization of the image regions in the external memory when  $n_x < M \times r_x$

in microprocessors with relatively limited processing capabilities. Evidently, careful implementation is always a paramount concern in wireless smart cameras. However, fine tuning the parameters of the algorithm such as the histogram resolution and the number of particles should provide a point of balance between its robustness and performance suitable for embedded smart cameras.

## 5. Performance Analysis

In this section, we provide an analysis of the potential performance gains of our approach as compared to a sequential implementation. The analysis is based on the following assumptions:

- Reorganization of the histograms can be done in parallel with image acquisition. Thus, the time required for this step is not considered.
- Reading a line from the external memory can be done in parallel with computations in the linear processing array. Thus, the time required for this step is also considered negligible.
- The sequential processor can compute 1 histogram element in 1 unit of processing time whereas the SIMD processor can perform the same computation in 50 units of processing time. This assumption should reflect the usually much lower clock speeds of low power processors.

Figure 5(a) shows the time required to compute the histograms as a function of the number of particles for the sequential processor and the SIMD processor with the line memory width fixed to 320 elements and the region size fixed to  $16 \times 16$  pixels. As the figure illustrates, gains of up to 4 times are possible, and this gain increases with the number of particles. The discontinuities in the computation time of the SIMD processor occur when the total width of the regions (i.e. the number of particles times the width of each region) is a multiple of the line width. This is due to the

fact that when the total width is not a multiple of the line memory width, the computations performed by the processing elements to the right of the last region are not used. Therefore, for an SIMD implementation, it is highly desirable to make the total width of the regions a multiple of the line memory width.

Figure 5(b) shows the same comparison with the number of particles fixed to 60, line memory width fixed to 320 elements and varying tracked region size. In that case, performance gains of up to 14 times are possible.

Figure 5(c) shows the effects of varying the line memory width when the number of particles is fixed to 60 and the region size to  $16 \times 16$  pixels. The computation time required by the sequential processor is constant in that case since it obviously does not depend on the line memory width. As the figure illustrates, the wider the line memory the faster the SIMD processor can compute the histograms. This is an expected result since wider line memory implies a larger number of processing elements. As the line memory width becomes wider than the total width of the image regions, no further gains are achieved since the extra processing elements are not involved in any computations. It is important to note, however, that although increasing the line memory width provides great performance gains, the increased hardware complexity will lead to more energy consumption. Therefore, there is a trade-off between performance and energy consumption, which must be carefully evaluated.

To validate the claim that resampling is not computationally expensive, we have implemented systematic resampling in an Atmel AVR ATmega128 processor running at 8MHz. Figure 6 shows the computation times for a varying number of particles. In our application, we have empirically verified that 100 particles are enough to obtain good tracking results. Hence, it is possible to perform resampling in less than 20ms in a very low power processor. On the same platform, computing the weights of 100 particles using 32 bins histograms of regions consisting of  $16 \times 16$  elements previously stored in the internal memory of the processor takes 294ms. If the number of bins of the histogram is not a power of 2, due to the floating point operations involved in computing the histogram, the processing time is much longer (855ms for histograms of 33 bins). Therefore, it is clear that weight computation, rather than resampling, is the bottleneck of the algorithm.

## 6. Proof-of-Concept Implementation

As a proof-of-concept, we have implemented a color-based particle filter using our histogram computation algorithm in a standard desktop computer. The filter

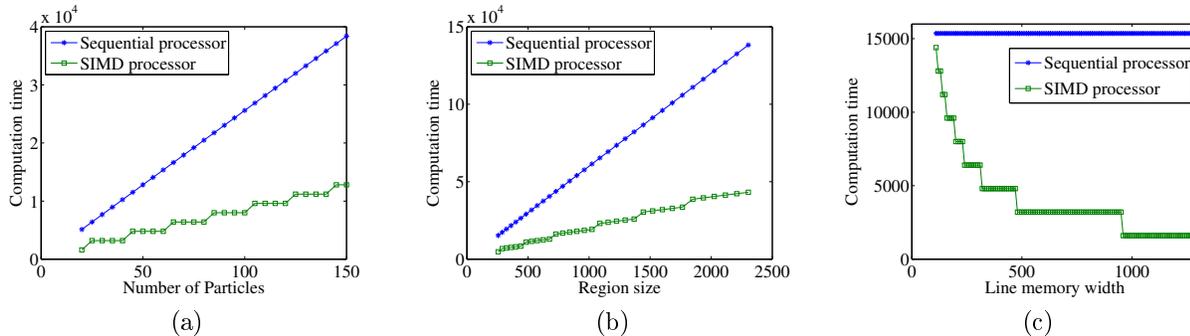


Figure 5. Analytical comparison of the computation times as a function of: (a) the number of particles, (b) the tracked region size, and (c) the width of the SIMD line memory.

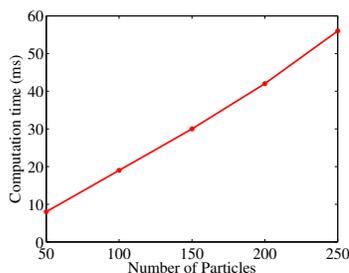


Figure 6. Resampling computation times.

uses the hue histograms of 100 particles, and systematic resampling is carried out at every iteration of the filter. To keep the implementation simple so that it could be ported to an embedded system, we restricted the tracked region to a fixed size of  $16 \times 16$  pixels. Figure 7 shows the tracking results obtained using the traditional method (left side), and our method (right side). As expected, since the histograms computed using our method are identical to the histograms computed using the traditional method, the results obtained by both systems are essentially the same.

## 7. Conclusions and Future Directions

The color-based particle filter is an effective algorithm for tracking non-rigid objects, however, at the cost of high computational expense. In this paper we have shown that it is possible to implement the major bottleneck of the algorithm, the computation of the color histograms, in a parallel manner suitable for an SIMD architecture. Our analysis of the algorithm shows that it should be possible to achieve substantial performance improvement in comparison to standard desktop computers while operating at much lower clock frequencies.

Many previous works have shown that it is possible to port complex computer vision algorithms to smart cameras based on SIMD processors [7, 10, 15, 27, 32]. As we have shown, the color-based particle filter is an

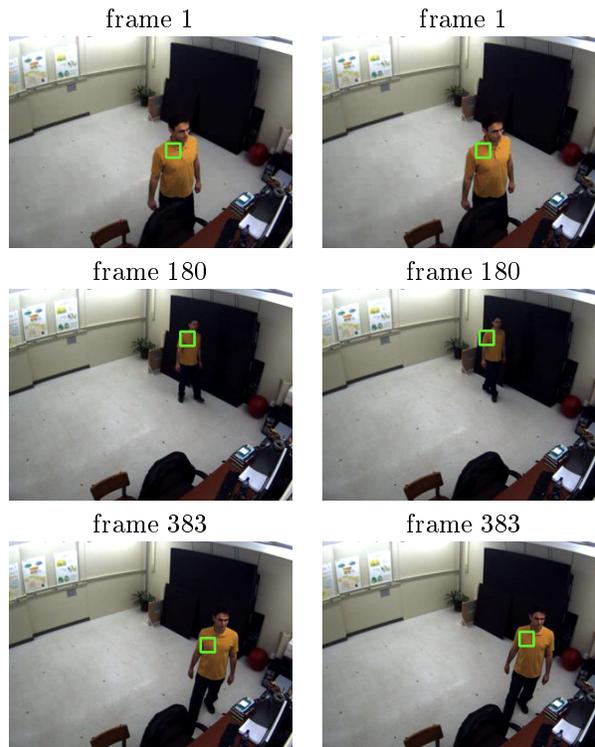


Figure 7. Tracking results.

other such algorithm. We believe that its real time implementation in an embedded camera will provide an invaluable building block for the design of applications of practical interest in portable embedded devices and wireless camera networks.

Our next goal is to implement the algorithm in a real smart camera based on an SIMD linear processor array in order to measure the actual performance gains. We are also currently investigating possible ways to allow multiple cameras tracking the same target using a color-based particle filter to collaborate in order to increase the robustness and accuracy of the algorithm. Since our ultimate goal is to be able to port this method to wireless cameras, we are trying to achieve

such collaboration while keeping the interaction among cameras to the minimum necessary.

## References

- [1] A. Abbo, R. Kleihorst, V. Choudhary, L. Sevat, P. Wielage, S. Mouy, B. Vermeulen, and M. Heijligers. Xetal-II: a 107 GOPS, 600 mW massively parallel processor for video scene analysis. *Solid-State Circuits, IEEE Journal of*, 43(1):192–201, Jan. 2008.
- [2] M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *Signal Processing, IEEE Transactions on*, 50(2):174–188, Feb. 2002.
- [3] M. Bolic. *Architectures for Efficient Implementation of Particle Filters*. PhD thesis, Stony Brook University, Aug. 2004.
- [4] M. Bolic, P. Djuric, and S. Hong. Resampling algorithms and architectures for distributed particle filters. *IEEE Transactions on Signal Processing*, 53(7):2442–2450, 2005.
- [5] D. Comaniciu, V. Ramesh, and P. Meer. Real-time tracking of non-rigid objects using mean shift. In *IEEE Conference on Computer Vision and Pattern Recognition, 2000. Proceedings*, volume 2, pages 142–149, 2000.
- [6] J. Czyz, B. Ristic, and B. Macq. A color-based particle filter for joint detection and tracking of multiple objects. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005. (ICASSP '05)*, volume 2, pages 217–220, Mar. 2005.
- [7] I. Diaz, M. Heijligers, R. Kleihorst, and A. Danilin. An embedded low power high efficient object tracker for surveillance systems. In *First ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC '07)*, pages 372–378, Sept. 2007.
- [8] S. Fleck and W. Strasser. Adaptive probabilistic tracking embedded in a smart camera. *Computer Vision and Pattern Recognition, 2005 IEEE Computer Society Conference on*, 3:134–134, 2005.
- [9] M. Isard and A. Blake. Condensation – conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29(1):5–28, 1998.
- [10] V. Jeanne, F.-X. Jegaden, R. Kleihorst, A. Danilin, and B. Schueler. Real-time face detection on a dual-sensor smart camera using smooth-edges technique. In *International Workshop on Distributed Smart Cameras (DSC 06)*, 31 Oct. 2006.
- [11] S. J. Julier and J. K. Uhlmann. A New Extension of the Kalman Filter to Nonlinear Systems. In *Proc. of AeroSense: The 11th Int. Symp. on Aerospace/Defence Sensing, Simulation and Controls*, 1997.
- [12] R. Kleihorst, A. Abbo, A. van der Avoird, M. Op de Beeck, L. Sevat, P. Wielage, R. van Veen, and H. van Herten. Xetal: a low-power high-performance smart camera processor. In *The 2001 IEEE International Symposium on Circuits and Systems, ISCAS 2001.*, volume 5, pages 215–218, 2001.
- [13] R. Kleihorst, B. Schueler, A. Danilin, and M. Heijligers. Smart Camera Mote With High Performance Vision System. In *Proceedings of the International Workshop on Distributed Smart Cameras (DSC-06)*, 31 Oct. 2006.
- [14] J. Kotecha and P. Djuric. Gaussian particle filtering. *IEEE Transactions on Signal Processing*, 51(10):2592–2601, 2003.
- [15] G. Kraft and R. Kleihorst. Computing Stereo-Vision in Video Real-Time with Low-Cost SIMD-Hardware. In *7th International Conference Advanced Concepts for Intelligent Vision Systems, ACIVS 2005*, pages 697–704, 5 Oct. 2005.
- [16] S. Kyo, S. Okazaki, and T. Arai. An integrated memory array processor for embedded image recognition systems. *IEEE Transactions on Computers*, 56(5):622–634, 2007.
- [17] S. Kyo and S. Sato. Efficient Implementation of Image Processing Algorithms on Linear Processor Arrays using the Data Parallel Language 1DC. In *Proc. of IAPR Workshop on Machine Vision Applications (MVA '96)*, pages 160–165, 1996.
- [18] S. Maskell, B. Alun-Jones, and M. Macleod. A Single Instruction Multiple Data Particle Filter. In *Proceedings of Nonlinear Statistical Signal Processing Workshop*, 2006.
- [19] K. Nummiaro, E. Koller-Meier, and L. V. Gool. A Color-Based Particle Filter. In *First International Workshop on Generative-Model-Based Vision*, 2002.
- [20] K. Nummiaro, E. Koller-Meier, and L. V. Gool. Object tracking with an adaptive color-based particle filter. In *Symposium for Pattern Recognition of the DAGM*, pages 353 – 360, 2002.
- [21] K. Nummiaro, E. Koller-Meier, T. Svoboda, D. Roth, and L. J. Van Gool. Color-Based Object Tracking in Multi-camera Environments. In *DAGM-Symposium 2003*, pages 591–599, 2003.
- [22] K. Okuma, A. Taleghani, N. de Freitas, J. Little, and D. Lowe. A boosted particle filter: Multitarget detection and tracking. In *Proc. ECCV, volume 3021 of LNCS*, pages 28–39. Springer, 2004.
- [23] P. Pérez, C. Hue, J. Vermaak, and M. Gangnet. Color-based probabilistic tracking. In *ECCV '02: Proceedings of the 7th European Conference on Computer Vision-Part I*, pages 661–675, London, UK, 2002. Springer-Verlag.
- [24] P. Perez, J. Vermaak, and A. Blake. Data fusion for visual tracking with particles. *Proceedings of the IEEE*, 92(3):495–513, 2004.
- [25] F. Porikli. Integral histogram: A fast way to extract histograms in cartesian spaces. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 1:829–836, 2005.
- [26] M. Rahimi, R. Baer, O. I. Iroezzi, J. C. Garcia, J. Warrior, D. Estrin, and M. Srivastava. Cyclops: in situ image sensing and interpretation in wireless sensor networks. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, 2005.
- [27] E. Simmons, E. Ljung, and R. Kleihorst. Distributed vision with multiple uncalibrated smart cameras. In *International Workshop on Distributed Smart Cameras (DSC 06)*, 31 Oct. 2006.
- [28] H. Sugano and R. Miyamoto. A Real-Time Object Recognition System on Cell Broadband Engine. In *Second Pacific Rim Symposium in Advances in Image and Video Technology (PSIVT)*, pages 932–943, Dec. 2007.
- [29] S. Sutharsan, A. Sinha, T. Kirubarajan, and M. Farooq. An optimization based parallel particle filter for multitarget tracking. In *Proceedings of the Spie.*, volume 5913, pages 87–98, Jan. 2005.
- [30] J. Vermaak, P. Pérez, M. Gangnet, and A. Blake. Towards improved observation models for visual tracking: Selective adaptation. In *ECCV '02: Proceedings of the 7th European Conference on Computer Vision-Part I*, pages 645–660, London, UK, 2002. Springer-Verlag.
- [31] G. Welch and G. Bishop. An Introduction to the Kalman Filter. Technical report, University of North Carolina at Chapel Hill, 1995.
- [32] C. Wu, H. Aghajan, and R. Kleihorst. Mapping vision algorithms on simd architecture smart cameras. In *First ACM/IEEE International Conference on Distributed Smart Cameras, 2007. ICDSC '07*, pages 27–34, 2007.