11-1-2010

A Parallel Histogram-based Particle Filter for Object Tracking on SIMD-based Smart Cameras

Henry Medeiros
Marquette University, henry.medeiros@marquette.edu

Germán Holguín
Purdue University

Paul J. Shin
Purdue University

Johnny Park
Purdue University

A Parallel Histogram-based Particle Filter for Object Tracking on SIMD-based Smart Cameras

Henry Medeiros, Germán Holguín, Paul J. Shin, Johnny Park

School of Electrical and Computer Engineering, Purdue University

Abstract

We present a parallel implementation of a histogram-based particle filter for object tracking on smart cameras based on SIMD processors. We specifically focus on parallel computation of the particle weights and parallel construction of the feature histograms since these are the major bottlenecks in standard implementations of histogram-based particle filters. The proposed algorithm can be applied with any histogram-based feature sets — we show in detail how the parallel particle filter can employ simple color histograms as well as more complex histograms of oriented gradients (HOG). The algorithm was successfully implemented on an SIMD processor and performs robust object tracking at up to 30 frames per second — a performance difficult to achieve even on a modern desktop computer.

Key words: tracking, particle filter, smart cameras, SIMD processors

1. Introduction

As the demand for low-power, portable, and networked computing devices continues to increase, it is natural that the number and the complexity of services provided by such devices will only grow. The operating speed of these devices, however, is bound to be much lower than that of standard desktop computers due to power consumption and size constraints. In order to support these new, more complex applications, there has been a major effort to design alternative processing architectures for small computing devices. Since these architectures are fundamentally different from those of general purpose processors, existing algorithms often need to be redesigned in order to be implemented in these systems.

In the specific case of computer 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 approach is the color-based particle filter [1, 2, 3, 4, 5]. In this method, a reference histogram of the target is initially provided to the tracker, which then uses Bayesian estimation to search for the most likely new location of the target in each of the subsequent frames. The results reported in the literature 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 partial occlusion of the target object and variations in the color of the background.

Email address: {hmedeiro, gholguin, paulshin, jpark}@purdue.edu (Henry Medeiros, Germán Holguín, Paul J. Shin, Johnny Park)

Recently, new histogram-based particle filters that employ not only histograms of color but also histograms of edge orientations have been proposed [6, 7, 8]. The histogram of oriented gradients (HOG) [9] is one example of a feature set based on histograms of edge orientations. It has been reported that HOG is robust to photometric and background variations, and hence complements some weaknesses of histograms purely based on color.

Although the particle filter has been proved to be an effective method for object tracking, it is computationally expensive, thus not suitable for the current generation of wireless smart cameras based on low-power general-purpose microcontrollers (e.g. the Cyclops camera [10]). The computation of the weights of a large number of particles is beyond the capability of these smart cameras. On the other hand, the algorithm lends itself to effective parallel implementation since there are no data dependencies among particles. Therefore, we show in this paper that, by devising a parallel histogram-based particle filter, it is possible to achieve robust real-time object tracking on a smart camera based on an SIMD processor such as the WiCa camera [11].

There has been much work on the parallelization of the particle filter [12, 13, 14, 15, 16]. The main objective of these studies was to parallelize the resampling step. However, as we will show later, the resampling step is not the major source of computational burden of a typical histogram-based particle filter. Instead, the major computational bottleneck is the evaluation of the particle weights, more specifically the construction of the histograms of the regions surrounding each hypothesized target position.

This paper is an extension of our previous work [17, 18]

in which we proposed a method for the parallel computation of the particle weights in the color-based particle filter on an SIMD processor. In this paper, we show that our methodology is valid for any histogram-based feature set — we show in detail how the parallel particle filter can employ not only simple color histograms but also complex feature descriptors, specifically the histograms of oriented gradients (HOG). Furthermore, we have successfully implemented the algorithm in the WiCa camera, thus our experimental results show the performance of the algorithm running on a real embedded smart camera.

2. Parallel Implementation of the Particle Filter

In this section, we show that, as long as the processor architecture allows for efficient access to an external memory, it is possible to compute in parallel the multiple histograms required by a histogram-based particle filter and the corresponding particle weights, thereby overcoming its greatest bottleneck.

2.1. Hardware Architecture

We propose an algorithm for an SIMD linear processor array architecture [19, 20, 21, 22]. The Xetal family of SIMD processors, illustrated in Figure 1, is one example of such architecture. It is composed of a linear processor array (LPA) of P processing elements (PEs), each containing an arithmetic logic unit and a small amount of memory. Each PE has direct read and write access to the memory of its two nearest neighbors. The line memory, i.e., the overall memory of the PEs, can be directly accessed by a digital input processor and by a digital output processor, which are responsible for transferring information between the LPA and the external memory and video devices. A global control processor (GCP) controls the operation of the LPA and is also able to carry out global digital signal processing operations.

The architecture is designed based on the stream processing paradigm [23]. That is, data must be processed as soon as it becomes available. In the case of image processing, this implies that the digital input processor provides one video line to the LPA as soon as it becomes available from the image sensor. This video line either has to be processed or stored before the next video line becomes available. Moreover, the PEs have access only to the current video line or to video lines that have been previously stored in the memory. Therefore, if random access to elements of the image is required, they must be stored in the external memory in advance.

2.2. Reorganization of Particle Regions

The left side of Figure 2 illustrates a typical distribution of the particle regions over an image for one iteration of the histogram-based particle filter. The particles $\mathbf{x}^i$ correspond to the hypothesized positions of the target, and the measurements $\mathbf{z}^i$ are given by the histograms of the

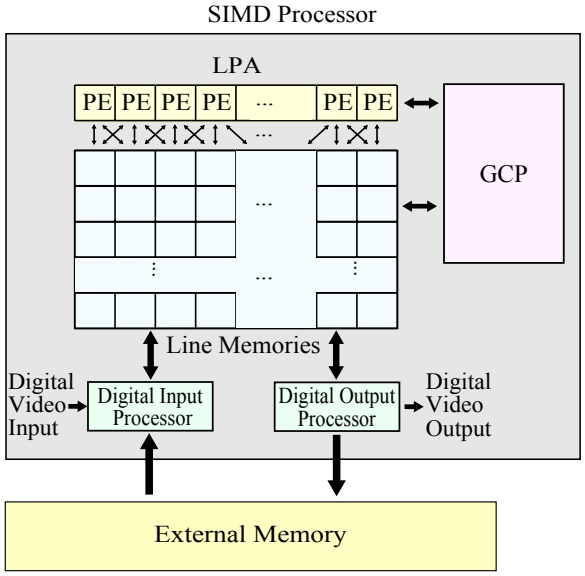


Figure 1: Hardware architecture of an SIMD linear processor array.

regions surrounding each particle. The particle likelihoods are computed based on the similarity between the measurements and the target reference histogram [3, 5, 18]. In order to process the particle regions in parallel, it is necessary to allocate a certain number of PEs to handle each region. Since the particle regions are randomly distributed in an image with many parts overlapping with one another, they must be reorganized so that each PE is provided the information about the region for which it is responsible.

Reorganization of the particle regions requires that the PEs have random access to the individual pixels of the image. Therefore, instead of processing the input video lines directly, we first extract the relevant image features and store them in the external memory so that they can be randomly accessed by the digital I/O processors, and consequently, by the PEs. Finally, for each particle region, the corresponding features stored in the external memory are reorganized into the line memory, as illustrated in the right side of Figure 2.

Let P denote the number of PEs and M the number of particle regions employed by the algorithm. We assume that all particle regions have the same size of $r_x \times r_y$ elements. The elements in each particle region are reorganized into an area of $s_x \times s_y$ elements in the line memory. Note that the value s_x is pre-assigned by the application depending on the values of M, P, r_x, and r_y (we will shortly discuss different cases of these values and a preferred value of s_x in each of these cases). Once the value of s_x is selected, then s_y is simply set to $\left\lceil \frac{r_x \cdot r_y}{s_x} \right\rceil$.

The reorganization proceeds as follows. In each processing step, s_x elements of each particle region are copied into the corresponding line memory space so that they can be processed in parallel. This is repeated s_y times until

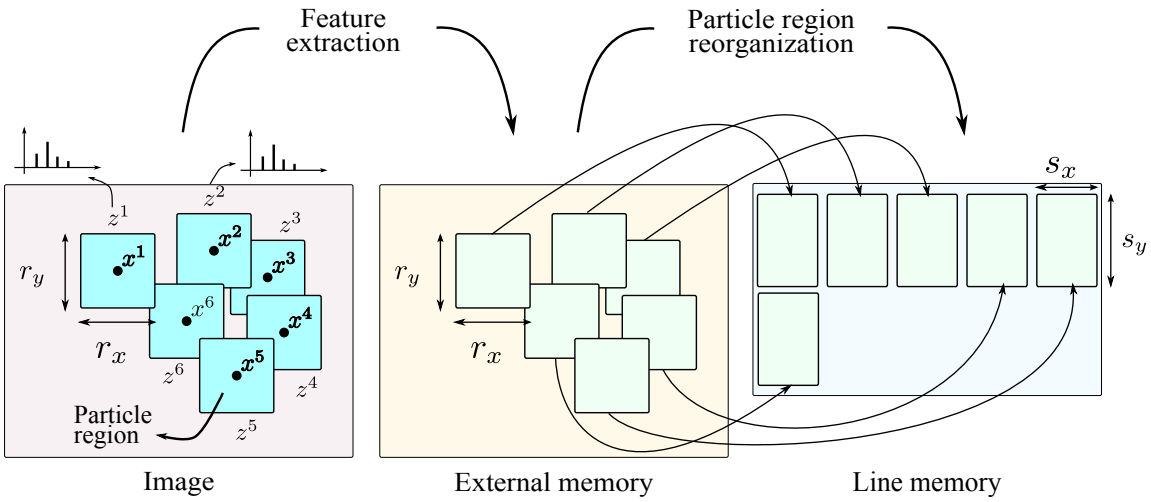


Figure 2: Extraction of features and reorganization of the particle regions in a parallel histogram-based particle filter.

all elements of the particle regions are processed. Since there are only P PEs, at most $\left\lfloor \frac{P}{s_x} \right\rfloor$ particle regions can be copied to the line memory simultaneously. In the example shown in Figure 2, $\left\lfloor \frac{P}{s_x} \right\rfloor = 5$. If $\left\lfloor \frac{P}{s_x} \right\rfloor < M$, there will be more than one row of particle regions in the line memory, as is the case in the above example where $M = 6$. In any case, at most $m = \left\lceil \frac{M \cdot s_x}{P} \right\rceil$ rows of particle regions are necessary in the line memory.

After the regions are reorganized, each row of particle regions can be processed in parallel in s_y steps. In the case of multiple rows ($m > 1$), $m \cdot s_y$ steps are required to process all the regions. Since s_x PEs are allocated to handle each particle region, additional $s_x - 1$ steps are then required for the PEs to share the results of their individual computations. The total time required to process all the particle regions is therefore given by $O(m \cdot (s_x + s_y))$. A more detailed explanation of particle region processing in the context of parallel histogram computation is given in Section 2.3.

When $P \geq M \cdot s_x$, the line memory can store all the particle regions side-by-side in a single row of particle regions, as illustrated in Figure 3(a). In that case, $m = 1$ and the time to process all the particle regions is $O(s_x + s_y)$. In the specific case of $s_x = r_x$, the processing time is $O(r_x + r_y)$, which is linear on the dimensions of the particle regions. As the number of particle regions M increases, m may become a large factor in the processing time. In that case, it is possible to reduce this factor by decreasing the value of s_x. Figure 3(b) shows the case for $s_x = 1$ where each particle region is organized as a column in the line memory. In that case, as long as $M \leq P$, the processing time is $O(r_x \cdot r_y)$, and each PE is responsible for processing one particle region so that up to P particles can be computed in parallel.

Table 1 summarizes the preferred reorganization approaches for different values of M with respect to P, r_x, and r_y. Note that for each case, a specific value for s_x is assigned.

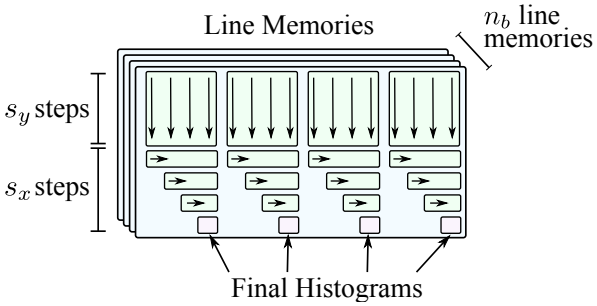


Figure 4: Parallel computation of the histograms.

2.3. Parallel Histogram Computation

One straightforward approach to compute the histograms of M rectangular image regions would be to employ integral histograms [24]. 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 requirement of integral histograms is generally too high for embedded systems.

Instead, in an SIMD linear processor array, it is preferable to compute the histograms of $\left\lfloor \frac{P}{s_x} \right\rfloor$ image regions in parallel as illustrated in Figure 4 — similar approaches were used in [22, 25]. This process is divided into two phases: vertical accumulation and horizontal accumulation. During vertical accumulation, the histograms of the columns of the regions are computed. Subsequently, the horizontal accumulation phase computes the total histograms of each image region by sequentially adding the histogram of a given column to that of its immediate neighbor. This procedure computes the histograms of all the image regions in $O(m \cdot (s_x + s_y))$ steps.

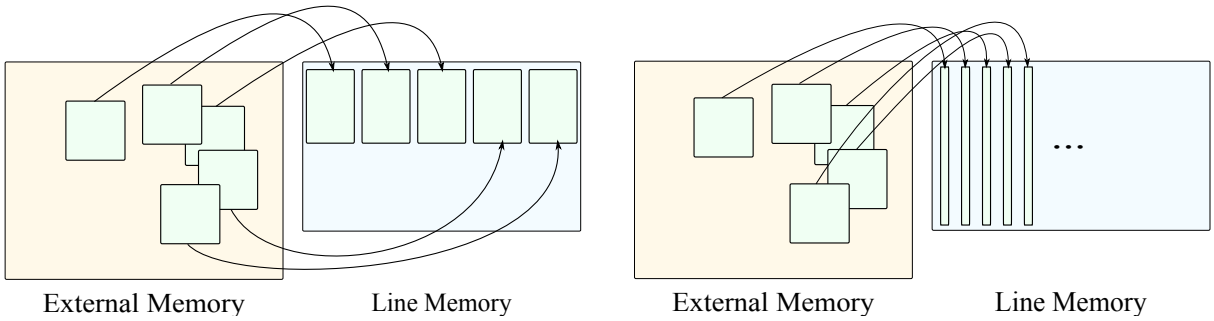


Figure 3: Organization of the particle regions in the line memory (a) as a single row of particle regions (i.e., when $\left\lfloor \frac{P}{s_x} \right\rfloor \geq M$) and (b) as columns (i.e., when $m \gg 1$).

Different values of M	Reorganization approach	s_x	Computation time
$M \leq \left\lfloor \frac{P}{r_x} \right\rfloor$	Single row of particle regions	r_x	$O(r_x + r_y)$
$\left\lfloor \frac{P}{r_x} \right\rfloor < M \leq \left\lfloor \frac{P \cdot r_y}{r_x + r_y} \right\rfloor$	Multiple rows of particle regions	r_x	$O(m \cdot (r_x + r_y))$
$M > \left\lfloor \frac{P \cdot r_y}{r_x + r_y} \right\rfloor$	Columns	1	$O(r_x \cdot r_y)$

Table 1: Preferred reorganization approaches for different values of M.

2.3.1. Parallel Computation of Color Histograms

In a color-based particle filter, each particle region consists of $r_x \times r_y$ elements, where each element has a corresponding bin number. Therefore, the feature extraction step (described earlier in Section 2.2) for the specific case of a color-based particle filter corresponds to computing the bin number of each pixel and storing it in the external memory. To determine the bin number, a function $h(\mathbf{u})$ discretizes the values of the color space of each pixel into n_b bins and assigns the corresponding bin number to the pixel. Using that approach, only $\log_2 n_b$ bits are required to store each pixel. Figure 5(a) illustrates the organization of the color information in the external memory.

Once the bin numbers are stored in the external memory, we can then proceed with the particle region reorganization process, as described in Section 2.2. Finally, the color histograms can be computed in parallel as described in Section 2.3.

2.3.2. Parallel Computation of Histograms of Oriented Gradients

In the feature extraction step for an HOG-based particle filter, instead of storing the gradient orientations of each individual pixel in the external memory, it is more effective to first compute the histograms of the HOG blocks in the entire image and then store them in the external memory, as illustrated in Figure 5(b). That is, as the video lines are received from the image sensor, n_b-bin block histograms are computed and stored in the external memory (to reduce the dimensionality of the HOG descriptor we compute one histogram per block instead of concatenating individual cell histograms). For the HOG features, one particle region corresponds to $b_x \times b_y$ blocks of histograms, each with n_b bins. To compute the particle weights in parallel, after the block histograms are stored in the external memory, those corresponding to each particle region are reorganized into the line memory as described in Section 2.2.

The block histograms are computed using the approach described in Section 2.3. However, since there is vertical overlap among adjacent HOG blocks (as shown in Fig. 5(b)), the horizontal accumulation step would overwrite column histograms of the neighboring blocks before their values were used in the computation of the block histogram. To overcome this problem, we allocate n_h additional line memories to accumulate the histogram bins in the positions corresponding to non-overlapping blocks. Histograms of vertically overlapping blocks are computed sequentially by temporarily storing the overlapping video lines.

2.4. Weight Computation

After the histogram distributions are computed, the likelihoods and corresponding unnormalized particle weights can be evaluated in parallel as long as the PEs have access to the common reference histogram. Weight normalization is then carried out by left- or right- shifting the weights and accumulating the total weight over all elements. The total weight is used by the GCP to compute a global scale factor which is multiplied by all the weights in parallel.

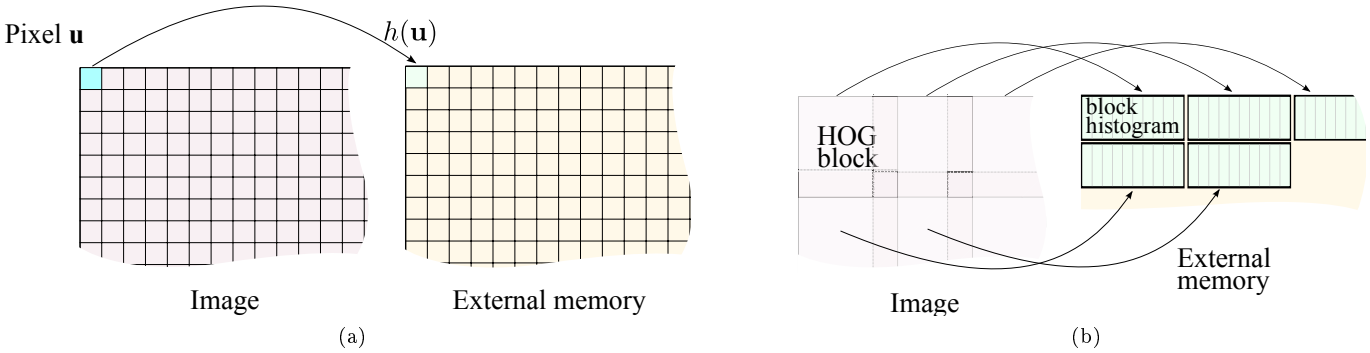


Figure 5: (a) Organization of the color information in the external memory. (b) Organization of the HOG block histograms in the external memory.

3. Performance Analysis

In this section, we provide a performance analysis of our algorithm. Since external memory access is a critical component of the platform that has a direct impact on the overall speed of the tracking algorithm, this section focuses on the constraints imposed by external memory access. With the purpose of creating a realistic evaluation scenario, the analysis is based on the hardware parameters of our experimental platform, which can be summarized on the following assumptions:

- The digital I/O processors are able to access up to $q_{max} = 640$ elements in the external memory during one video line interval. The maximum number of video line intervals per frame is $l_{max} = 480$.
- The relevant features are extracted and stored in the external memory in the first frame and the initial $q_o = q_{max} \cdot \frac{l_{max}}{2}$ external memory accesses take place during that period. Afterwards, $q_{max} \cdot l_{max}$ elements can be accessed per frame interval.
- l_c video line intervals — the time required to compute the particle weights after the particles are reorganized in the line memories — are negligible compared to the memory access time (this assumption will be justified in Section 4.2).

Let q be the total number of external memory accesses required by the tracking algorithm in each frame. If $q < q_o$, all the information can be read from the external memory during image acquisition, and only one frame is necessary to estimate the target position. Otherwise, the number of additional video line intervals required to process one video frame is $l = \left\lceil \frac{q-q_o}{q_{max}} \right\rceil$. The number of video frame intervals required to process one input video frame is then given by $f = \left\lceil \frac{l}{l_{max}} \right\rceil + 1$.

For the color features, the number of external memory accesses is given by the size of each particle region multiplied by the number of particles M, i.e. $q = M \cdot r_x \cdot r_y$. For the HOG features, the number of external memory accesses is given by the total number of histogram bins within a particle region multiplied by the number of particles, i.e. $q = M \cdot b_x \cdot b_y \cdot n_b$. In the case of HOG features, it is important to notice that the actual size of each particle region is typically larger than the number of histogram bins needed to represent it. Expressing the size of a particle region in terms of the size of HOG blocks, $e_x \times e_y$, and the number of non-overlapping pixels between adjacent HOG blocks, p_x and p_y, we have $r_x = e_x + (b_x - 1) \cdot p_x$ and $r_y = e_y + (b_y - 1) \cdot p_y$.

Figure 6(a) shows the number of frames required to estimate the target position as a function of the size of each particle region. Here 320 particles were used for both trackers, and the HOG features consisted of blocks of $e_x = e_y = 16$ pixels and $p_x = p_y = 4$ pixels. It can be seen that the I/O requirements of the color-based particle filter are much more stringent than those of the HOG-based tracker. The discontinuities shown on the graphs correspond to the times when the number of video line intervals required to read the elements from the external memory are exact multiples of l_{max}. Therefore, to make the optimal use of the available resources, it is highly desirable to make the region size a multiple of l_{max}.

Figure 6(b) shows the number of frames required to estimate the target position as a function of the number of particles with the particle region size fixed at 480 pixels. For up to 320 particles, both algorithms are able to track the target at $30fps$, but we can see again that the color-based particle filter requires much more frequent access to the external memory than the HOG version. Although the figure shows up to 4000 particles, this number is neither practical nor desirable. In practice, memory limitations would not allow thousands of particles to be processed since that would require each PE to store a large amount of data. Moreover, as will be shown in Section 4.1, in our application, the benefits of increasing the number of particles above 320 are negligible.

Figures 7(a)-(c) show the computation times of the HOG-based particle filter using 320 particles for several region sizes as a function of different parameters of the HOG descriptor. In Figure 7(a), the computation time is

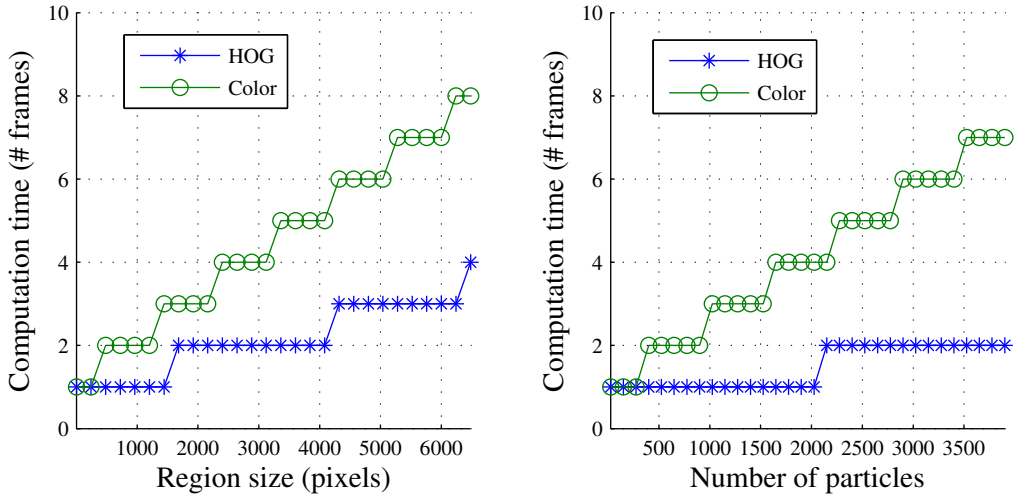


Figure 6: Analytical comparison of the particle filter computation time: (a) as a function of the size of each particle region and (b) as a function of the number of particles.

presented as a function of the number n_o of overlapping pixels among adjacent HOG blocks with a fixed block size of $e_x = e_y = 16$ and histograms of $n_b = 8$ bins. The figure shows that for regions of up to approximately 1500 pixels, it is possible to access all the required information in one frame interval for up to 12 overlapping pixels. Figure 7(b) shows the computation time as a function of the block size with $n_b = 8$, and $n_o = 12$. As the graph shows, for region sizes of 480 pixels, it is possible to access the information in one frame for block sizes as small as 196 pixels. Figure 7(c) shows the computation time as a function of the number of block histogram bins n_b with $e_x = e_y = 16$ and $n_o = 12$. As the figure shows, for regions of up to approximately 1500 pixels, it is possible to compute the histograms in one frame interval employing histograms of up to 9 bins. The benefits of using more than 9 bins are negligible however, as shown in [26].

3.1. Memory Access Requirements for Integral Histograms

For the purpose of comparison, we evaluated the memory access requirements of a particle filter in which the histograms of the particle regions are computed using integral histograms. Since this approach requires temporarily storing one histogram per image pixel, the total memory required is $q_w = n_b \cdot x \cdot y$, where n_b is the number of histogram bins and x and y are the horizontal and vertical image resolutions respectively (assuming each histogram bin can be represented by one byte using fixed-point notation).

In addition to storing the histograms in the external memory, in order to compute the histogram of each rectangular region, it is necessary to read the integral histograms of each of its four corners from the external memory. Therefore, if a particle region consists of n_r regions ($n_r = 1$ for the color-based particle filter and $n_r = b_x \cdot b_y$ for the HOG-based particle-filter), the number of elements that must be read from the external memory is $q_r = 4 \cdot n_b \cdot M \cdot n_r$. The total number of external memory accesses required by the particle filter using integral histograms is thus given by $q = q_w + q_r = n_b \cdot (x \cdot y + 4 \cdot M \cdot n_r)$. Figure 8(a) shows the number of frames required to estimate the target position as a function of the region size with the number of particles fixed at 320 pixels and the image resolution at 256×240. The parameters used to represent color histograms and HOG descriptors are the same as those presented in Figure 6(a). The graph shows that for both the color-based particle filter and the HOG-based particle filter, integral histograms require substantially larger numbers of frames to access the external memory than our proposed approach.

Even if we define a search window in the vicinity of the last known position of the target so that it is not necessary to compute the integral histograms over the entire image, our approach performs substantially better. Figure 8(b) shows the percentage of the image area covered by the search window of the integral histogram-based tracker so that its computation time is equivalent to that of the corresponding SIMD-based approach (i.e., color-based tracker or HOG-based tracker using different block sizes). The results are for an image resolution of 256×240 and 320 particles. For small particle region sizes, it is clear that the proposed algorithms greatly outperform the integral histograms-based approach. For example, for particle regions of 1600 pixels, the search window of the integral histogram-based tracker would be able to cover only 4.5% of the image area. As the particle region size increases, so does the area that can be covered using the integral histograms-based approach. In the extreme case of a particle region size of $10,000$ pixels, using a 16×16-block HOG, the integral histograms search window is able to cover approximately 40% of the image area, but this search region

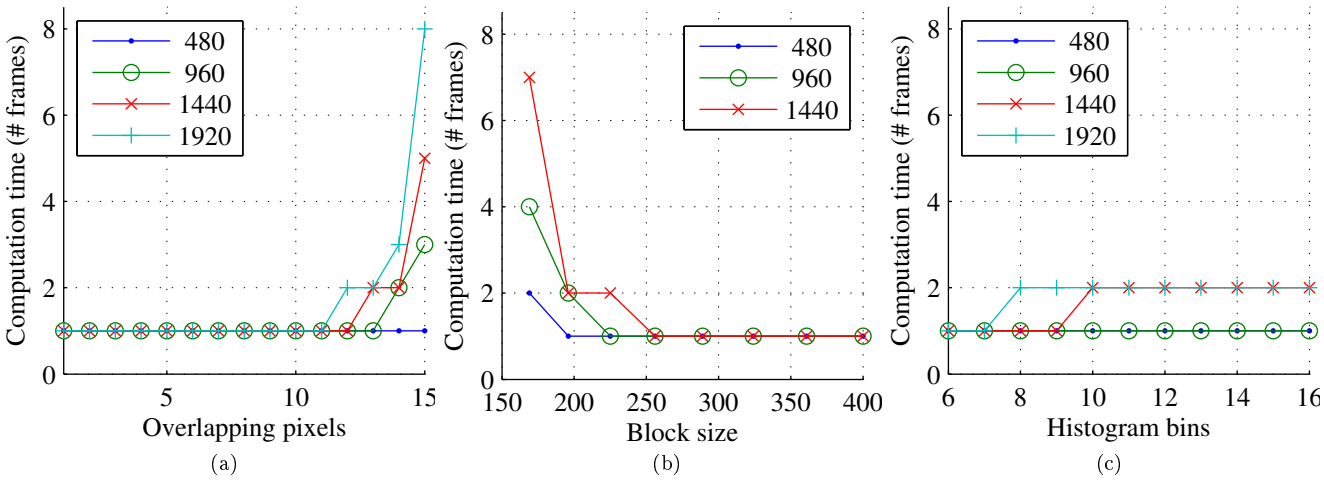


Figure 7: Computation time of the HOG-based tracker as a function of (a) the number of overlapping pixels, (b) the block size, and (c) the number of bins.

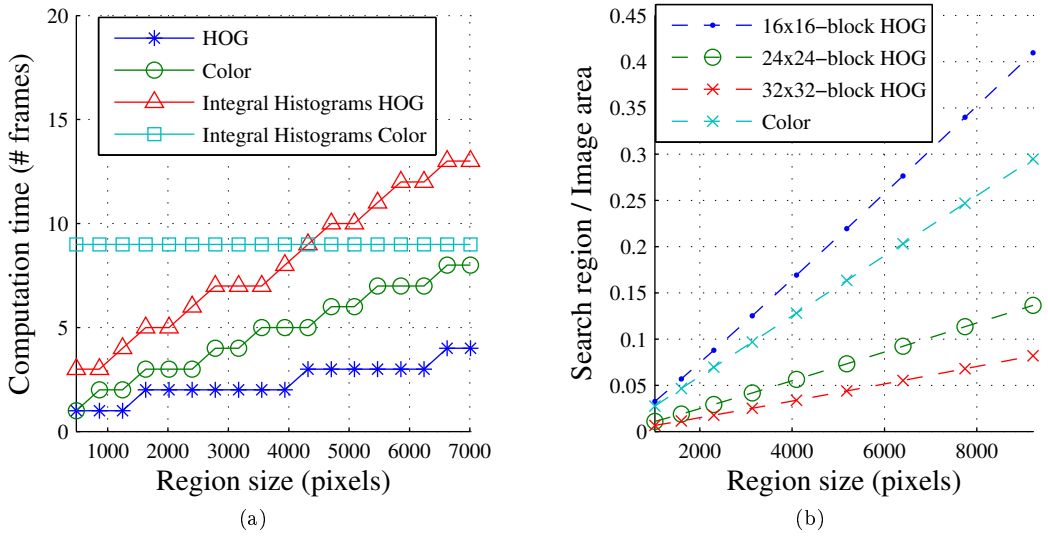


Figure 8: Performance comparison with integral histograms: (a) computation time with unrestricted search window, (b) integral histograms maximum search window size that allows performance equivalent to the proposed approach.

is only 50% larger than the size of the particle region itself.

4. Experimental Results

4.1. Accuracy and Speed

To evaluate the performance of our tracking algorithms and validate our choice of parameters, we implemented them on a standard desktop computer and performed experiments on five video sequences from the PETS 2001 data set.[1] In the HOG-based particle filter, we tracked regions of 2×4 blocks of 16×16 pixels with 12 pixels of overlap. In the color-based particle filter, we tracked regions of 12×25 pixels. In both filters, we used 320 particles.

Based on the ground-truth information, we computed the root mean squared (rms) error of the target position estimated by the HOG-based tracker. Figure 9(a) shows the accuracy improvement obtained as the number of particles is increased. The results correspond to the average rms error over five different targets. The experiment was repeated 40 times for each target. The results indicate that the accuracy gains obtained by increasing the number of particles above 320 are negligible.[2]

Figure 9(b) compares the average execution speed of the HOG-based and of the color-based particle filters implemented in a standard desktop computer to the expected execution speed of the SIMD versions of the algorithms. The results were obtained using an Intel Xeon Quad Core 2.83GHz.[3] In the case of the HOG-based tracker, it can be seen that our method outperforms the standard implementation when more than 175 particles are used. Although the standard implementation of the color-based tracker outperforms our approach for a small number of particles, it is important to notice that the performance of the parallel color-based particle filter is limited mainly by the fact that our platform can acquire at most 30 image frames per second.

4.2. Performance Evaluation of the Proposed Algorithms on an Embedded Platform

This section describes the implementation and performance evaluation of the proposed algorithms on an embedded platform, specifically on the WiCa smart camera. The WiCa camera consists of four main components: VGA color image sensor, IC3D/Xetal SIMD processor, general purpose processor, and Dual Port RAM (DPRAM) shared by the processors. The LPA of the IC3D/Xetal processor consists of $P = 320$ RISC PEs, each endowed with 64 words of memory (10 bits wide).

In our implementation, the target is tracked on images of resolution of 256×240 pixels, which are obtained by down-sampling the VGA frames provided by the image sensor. To track the target at $30fps$, we employ a pipelined processing approach in which extraction of features of the current frame is interleaved with histogram computation of the previous frame during the odd and even video line intervals. The distribution of the target state is approximated by 320 particles, and the observation likelihoods are computed based on the Bhattacharyya distances between the histograms of the particle regions and the target reference histogram. Since we are employing a large number of particles, they are organized into columns in the memory (as illustrated in Figure 3(b) of Section 2.2), so that each PE is responsible for one particle.

For the color-based particle filter, we track the target based on its hue histogram. Due to memory constraints, we use 40-bin histograms. Each particle region consists of a rectangular area of $r_x \times r_y = 21 \times 22$ pixels, which corresponds to the largest region that can be tracked at $30fps$ using 320 particles.

For the HOG-based particle filter, we compute the gradients using gray scale images and use simplified HOG histograms employing a binary voting scheme (i.e., each pixel counts as one vote for the corresponding bin regardless of the gradient magnitude). We chose $n_b = 8$ bins and $n_h = 2$ line memories, which makes the block overlap $n_o = 12$ pixels. Each particle region consists of $b_x \times b_y = 4 \times 9$ blocks of 16×16 pixels.

Table 2(a) shows the time required for each processing step of the color-based particle filter. Only the first step of the algorithm, i.e., conversion to HSV color space, is carried out during the odd video line intervals. As we can see in the table, this step takes $6.5\mu s$ per video line, or approximately 9% of one video line interval.

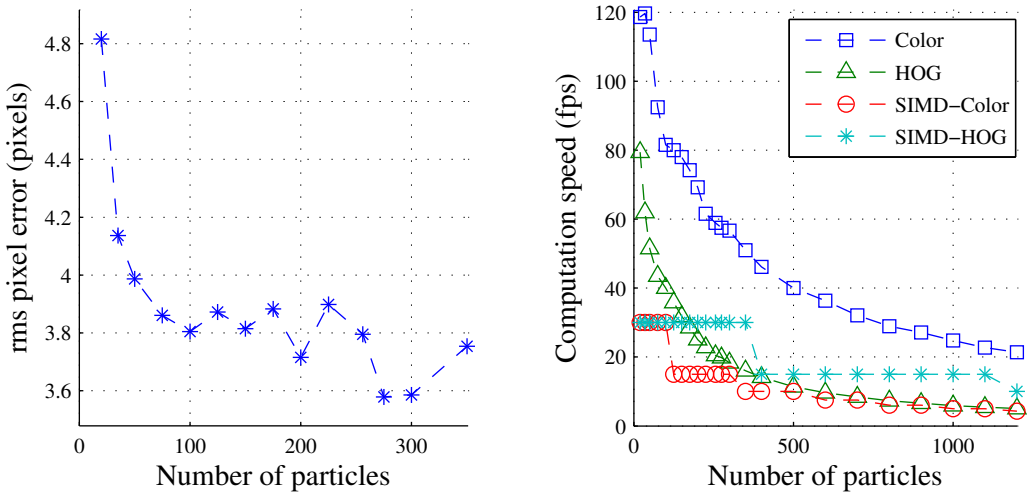
Table 2(b) shows the time required for each processing step of the HOG-based particle filter. During the odd video line intervals, the orientations of the gradients of each image pixel are computed and accumulated in the line memory to allow the computation of the histograms, which takes places at every $p_y = 4$ video lines. Gradient orientation computation takes $7.5\mu s$ per video line. Histogram computation takes $16.4\mu s$ and is executed $\frac{240}{4} = 60$ times. In the HOG-based particle filter, the LPA is busy, on average, for $11.6\mu s$ or 17% of the time during the odd video line intervals.

Since we chose to organize the particle regions as columns in the line memory, during the even video line intervals, every step of the algorithm is computed in parallel for all the particles. In the color-based particle filter, histogram computation — the first step executed during the even video lines — requires $\lceil \frac{r_x \cdot r_y}{2} \rceil = 231$ iterations of $2.4\mu s$ (since two elements per PE are read into the line memory at each

[1] http://www.cvg.cs.rdg.ac.uk/PETS2001/pets2001-dataset.html

[2] The minimum number of particles necessary depends on the complexity of the task that one wishes to accomplish. In our specific application 320 particles suffice, but that may not be the case in more complex applications such as tracking targets that undergo drastic appearance changes or that move among a highly dynamic background. In fact, under such conditions simple color histograms or HOG descriptors may not be powerful enough to perform robust tracking.

[3] Since we are interested in evaluating the performance of the algorithms on a sequential processor, our implementation was not optimized to use the four processing cores.



(a) (b)

Figure 9: (a) Accuracy of the HOG-based tracker as a function of the number of particles. (b) Comparison of the execution speeds of particle filter trackers.

video line interval) and is followed by the computation of the Bhattacharyya distances. In the HOG-based particle filter, since the block histograms are computed during the odd video line intervals (at every 4 video lines), the Bhattacharyya distances between each block histogram and the corresponding reference histogram can be computed during the even video line intervals in $b_x \cdot b_y = 36$ steps of $18\mu s$.

The remaining steps of the particle filter take place during the subsequent l_c video line intervals, and their execution times are shown in Table 2(c). The computation time for weight normalization is data-dependent due to the use of iterative routines for integer division. The table shows its average value and standard deviation over 20 iterations of the algorithm while tracking a target. The weighted average of the particles is computed using a shift/accumulate procedure similar to that used to normalize the particle weights, therefore its computation time is also data-dependent. The table shows its minimum and maximum values given the valid range of the weighted particles. The resampling time shown in the table corresponds to the time to copy the replicated particles to their corresponding PEs since the resampling factors are computed during weight normalization at every iteration of the filter. Therefore, after the histograms have been computed, between $271.5\mu s$ and $318.7\mu s$ are required to estimate the target position. Thus, $\left\lceil \frac{271.5\mu s}{69.5\mu s} \right\rceil \leq l_c \leq \left\lceil \frac{318.7\mu s}{69.5\mu s} \right\rceil$ or $4 \leq l_c \leq 5$. For both approaches, the total computation time during one frame is below $4ms$ or 12% of one frame interval.

Table 3(a) shows the line memory usage of the color-based particle filter. As we can see, more than 64% of the line memories are used to store the histograms. Regarding program memory usage, the current implementation of the algorithm consists of 1368 instructions out of the 2048 in-

Conversion to HSV	$240 \cdot 6.5\mu s$
Histogram computation	$231 \cdot 2.4\mu s$
Bhattacharyya distances	$79.6\mu s$
Total	$2.2ms$

(a)

Gradient orientation	$240 \cdot 7.5\mu s$
Histogram computation	$60 \cdot 16.4\mu s$
Bhattacharyya distances	$36 \cdot 18\mu s$
Total	$3.4ms$

(b)

Particle prediction	$10.6\mu s$
Likelihoods	$3.5\mu s$
Weight normalization	$61.4\mu s/4\mu s$
Weighted average of the particles	$2 \cdot 36\mu s$ to $55.6\mu s$
Resampling	$128\mu s$
Total	$271.5\mu s$ to $318.7\mu s$

(c)

Table 2: Processing times of each step of the algorithm on the WiCa camera. (a) Color and (b) HOG histogram extraction and distance computation and (c) particle filtering.

Target state	4
Particle weights	1
Histograms	40
Video line buffers	3
Temporary variables	14
Total	62

(a)

Target state	4
Particle weights	1
Histograms	16
Temporary gradient storage	16
Edge mask data	2
Video line buffers	3
Temporary variables	22
Total	64

(b)

Table 3: Line memory usage. (a) Color-based particle filter. (b) HOG-based particle filter.

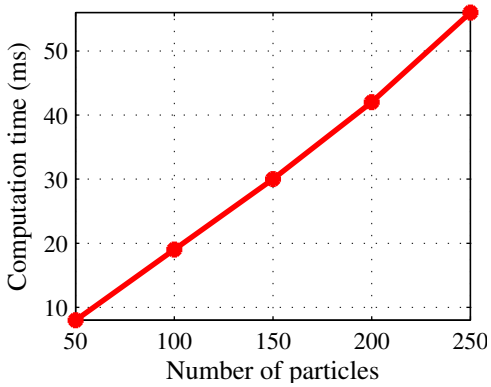


Figure 11: Resampling computation times in a low-power general-purpose processor.

structions of program memory available in the IC3D/Xetal processor. Table 3(b) shows the line memory usage of the HOG-based particle filter. In the HOG-based tracker, over 50% of the line memories are used for histogram computation. Regarding program memory usage, the algorithm consists of 1704 instructions.

Figure 10(a) shows snapshots of the tracking results of our implementation of the parallel color-based particle filter in the WiCa camera. Figure 10(b) shows snapshots of the tracking results of our implementation of the HOG-based particle filter.

4.3. Resampling

As we mentioned earlier, although resampling cannot be performed in parallel, it is not computationally expensive for a moderate number of particles. To validate this claim, we implemented systematic resampling on an Atmel AVR ATmega128 general-purpose processor running at 8MHz. Figure 11 shows the computation times for a varying number of particles. As the figure shows, it is possible to resample 200 particles in less than 40ms in a low-power general purpose processor. On the same platform, computing the weights of 200 particles using 32-bin color histograms of regions consisting of 16×16 elements previously stored in the internal memory of the processor takes 588ms. If the number of histogram bins is not a power of 2, due to the floating point operations involved in the computations, the processing time is much longer (1.7s for histograms of 33 bins). Therefore, it is clear that weight computation, rather than resampling, is the bottleneck of a histogram-based particle filter tracker.

5. Conclusions and Future Directions

Many previous studies have shown that it is possible to port complex computer vision algorithms to smart cameras based on SIMD processors [27, 28, 29, 30, 31]. As we have demonstrated, the histogram-based particle filter using color and HOG features is another such algorithm. In this paper, we have shown that not only is it possible to handle the major bottleneck of the algorithm — the computation of the feature histograms, and consequently, the particle weights — in a parallel manner suitable for an SIMD architecture, but also that the non-parallelizable steps can be implemented efficiently. We have analyzed the computational requirements of the algorithm and concluded that it is possible to track targets at $30fps$ and that this performance is scalable in terms of the size of the tracked regions and the number of particles. We have also demonstrated that it is possible to vary the HOG parameters without compromising the computational performance of the tracker. Our analysis also showed that the proposed algorithms clearly outperform alternative versions that use integral histograms to compute the observations. Finally, our experimental results showed that both the color-based and the HOG-based tracker are able to track targets at $30fps$ on a low-power embedded platform, a performance comparable to that obtained using a desktop computer. We believe that the real-time implementation of the histogram-based particle filter on an embedded camera will provide an invaluable building block for the design of practical applications using portable embedded devices and wireless camera networks.

One of the major limitations of our current approach is that the target models are not updated. Therefore, the algorithms are not robust to large variations in the appearance of the target. In particular, the color-based approach is sensitive to large illumination changes, whereas the HOG-based tracker cannot handle large scale variations. One extension of our work that should greatly increase the robustness of our algorithms would be to include a model update method. Some approaches have been proposed in the literature [6, 8, 32, 33], however, it is necessary to evaluate to what extent these approaches can be ported to an embedded system. It is also necessary to devise methods to implement these approaches in parallel.

Another immediate extension of our work would be to integrate both color- and HOG-based trackers into a single algorithm so that the target can be tracked based on both features simultaneously. Even though this inte-

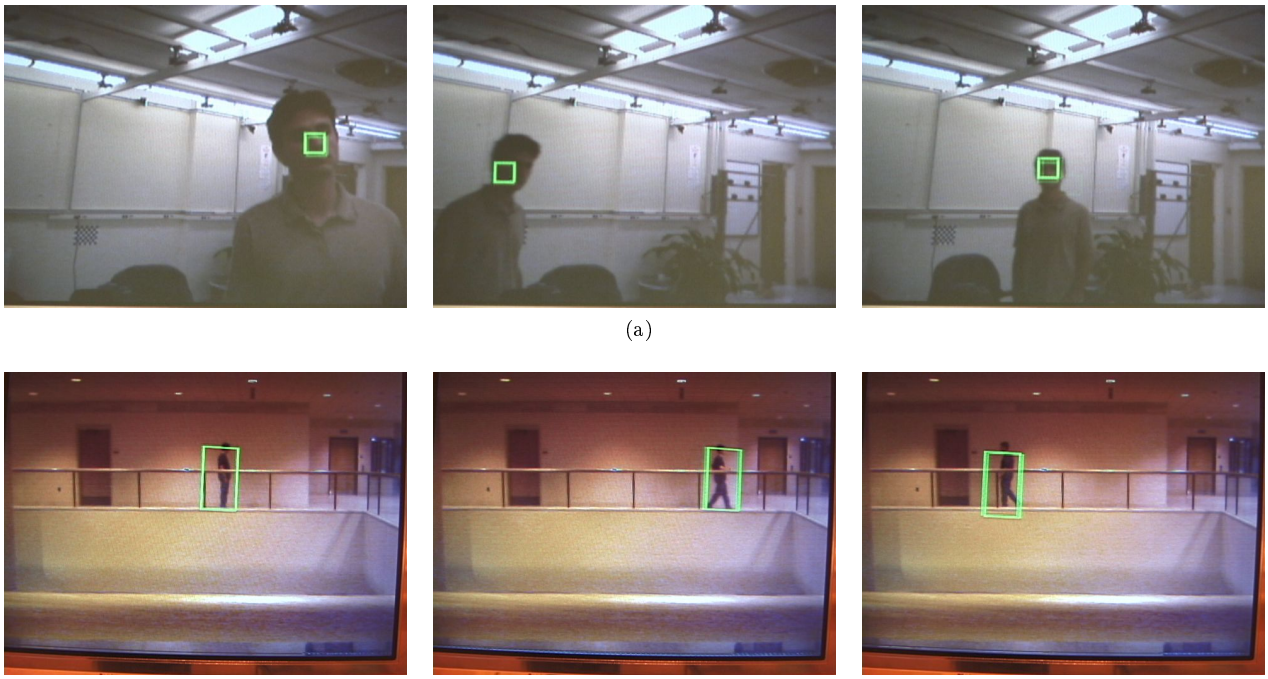


Figure 10: Tracking results. (a) Color-based particle filter. (b) HOG-based particle filter.

gration is conceptually straightforward, it is not trivial to overcome the hardware constraints of our current platform. One possible approach would be to employ multiple cameras, but several research issues must be addressed to allow multi-camera collaboration. We are currently investigating collaboration frameworks for multiple networked cameras tracking the same target in order to increase the robustness and accuracy of tracking. It should be possible, for example, to employ a cluster-based architecture [34] in which the cluster head is responsible for aggregating individual tracking results from the cluster members.

Acknowledgments

This work was supported by Olympus Corporation.

References

[1] J. Czyz, B. Ristic, 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), Vol. 2, 2005, pp. 217–220.

[2] S. Fleck, W. Strasser, Adaptive probabilistic tracking embedded in a smart camera, Computer Vision and Pattern Recognition, 2005 IEEE Computer Society Conference on 3 (2005) 134–134.

[3] K. Nummiaro, E. Koller-Meier, L. V. Gool, A Color-Based Particle Filter, in: First International Workshop on Generative-Model-Based Vision, 2002.

[4] K. Okuma, A. Taleghani, N. de Freitas, J. Little, D. Lowe, A boosted particle filter: Multitarget detection and tracking, in: Proc. ECCV, volume 3021 of LNCS„ Springer, 2004, pp. 28–39.

[5] P. Pérez, C. Hue, J. Vermaak, M. Gangnet, Color-based probabilistic tracking, in: ECCV '02: Proceedings of the 7th European Conference on Computer Vision-Part I, Springer-Verlag, London, UK, 2002, pp. 661–675.

[6] W.-L. Lu, J. Little, Simultaneous tracking and action recognition using PCA-HOG descriptor, in: The 3rd Canadian Conference on Computer and Robot Vision, 2006., 2006.

[7] K. K. Ng, E. J. Delp, New models for real-time tracking using particle filtering, in: M. Rabbani, R. L. Stevenson (Eds.), Visual Communications and Image Processing 2009, Vol. 7257, SPIE, 2009, p. 72570B.

[8] J. Zuo, C. Zhao, Y. Cheng, H. Zhang, Particle Filter Based Visual Tracking Using New Observation Model, in: Procedings of the IEEE International Conference on Automation and Logistics, 2007, pp. 436–440.

[9] N. Dalal, B. Triggs, Histograms of Oriented Gradients for Human Detection, in: Proceedings of the 2005 IEEE computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2005, 2005.

[10] M. Rahimi, R. Baer, O. I. Iroezi, J. C. Garcia, J. Warrior, D. Estrin, 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, pp. 192 – 204.

[11] R. Kleihorst, B. Schueler, A. Danilin, M. Heijligers, Smart Camera Mote With High Performance Vision System, in: Proceedings of the International Workshop on Distributed Smart Cameras (DSC-06), 2006.

[12] M. Bolic, P. Djuric, S. Hong, Resampling algorithms and architectures for distributed particle filters, IEEE Transactions on Signal Processing 53 (7) (2005) 2442–2450.

[13] M. Bolic, Architectures for Efficient Implementation of Particle Filters, Ph.D. thesis, Stony Brook University (Aug. 2004).

[14] S. Maskell, B. Alun-Jones, M. Macleod, A Single Instruction Multiple Data Particle Filter., in: Proceedings of Nonlinear Statistical Signal Processing Workshop, 2006.

[15] J. Kotecha, P. Djuric, Gaussian particle filtering, IEEE Transactions on Signal Processing 51 (10) (2003) 2592–2601.

[16] S. Sutharsan, A. Sinha, T. Kirubarajan, M. Farooq, An opti-

mization based parallel particle filter for multitarget tracking, in: Proceedings of the Spie„ Vol. 5913, 2005, pp. 87–98.
[17] H. Medeiros, X. Gao, J. Park, R. Kleihorst, A. Kak, A Parallel Implementation of the Color-Based Particle Filter for Object Tracking, in: Proceedings of the ACM Sensys Workshop on Applications, Systems and Algorithms for Image Sensing (ImageSense'08), 2008.
[18] H. Medeiros, J. Park, A. Kak, A parallel color-based particle filter for object tracking, in: IEEE computer society conference on computer vision and pattern recognition workshops, 2008. cvpr workshops 2008., 2008, pp. 1–8.
[19] R. Kleihorst, A. Abbo, A. van der Avoird, M. Op de Beeck, L. Sevat, P. Wielage, R. van Veen, H. van Herten, Xetal: a low-power high-performance smart camera processor, in: The 2001 IEEE International Symposium on Circuits and Systems, ISCAS 2001., Vol. 5, 2001, pp. 215–218.
[20] A. Abbo, R. Kleihorst, V. Choudhary, L. Sevat, P. Wielage, S. Mouy, B. Vermeulen, M. Heijligers, Xetal-II: a 107 GOPS, 600 mW massively parallel processor for video scene analysis, Solid-State Circuits, IEEE Journal of 43 (1) (Jan. 2008) 192–201.
[21] A. Lokhmotov, B. R. Gaster, A. Mycroft, N. Hickey, D. Stuttard, Revisiting simd programming, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 32–46.
[22] S. Kyo, S. Okazaki, T. Arai, An integrated memory array processor for embedded image recognition systems, IEEE Transactions on Computers 56 (5) (2007) 622–634.
[23] U. J. Kapasi, S. Rixner, W. J. Dally, B. Khailany, J. H. Ahn, P. Mattson, J. D. Owens, Programmable stream processors, Computer 36 (8) (2003) 54–62.
[24] 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 (2005) 829–836.
[25] S. Kyo, 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), 1996, pp. 160–165.
[26] N. Dalal, Finding People in Images and Videos, Ph.D. thesis, Institut National Polytechnique de Grenoble (17 Jul. 2006).
[27] I. Diaz, M. Heijligers, R. Kleihorst, 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), 2007, pp. 372–378.
[28] V. Jeanne, F.-X. Jegaden, R. Kleihorst, A. Danilin, 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), 2006.
[29] G. Kraft, 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, 2005, pp. 697–704.
[30] E. Simmons, E. Ljung, R. Kleihorst, Distributed vision with multiple uncalibrated smart cameras, in: International Workshop on Distributed Smart Cameras (DSC 06), 2006.
[31] C. Wu, H. Aghajan, R. Kleihorst, Mapping vision algorithms on simd architecture smart cameras, in: First ACM/IEEE International Conference on Distributed Smart Cameras, 2007. ICDSC '07, 2007, pp. 27–34.
[32] J. Vermaak, P. Pérez, M. Gangnet, 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, Springer-Verlag, London, UK, 2002, pp. 645–660.
[33] K. Nummiaro, E. Koller-Meier, L. V. Gool, An adaptive color-based particle filter, Image and Vision Computing 21 (1) (2003) 99 – 110.
[34] H. Medeiros, J. Park, A. Kak, Distributed Object Tracking Using a Cluster-Based Kalman Filter in Wireless Camera Networks, IEEE Journal of Selected Topics in Signal Processing 2 (2008) 448–463.