A Parallel Implementation of the Color-Based Particle Filter for Object Tracking

Henry Medeiros
hmedeiro@purdue.edu
Purdue University
West Lafayette, IN

Xinting Gao
xinting.gao@nxp.com
NXP Semiconductors
Eindhoven, The Netherlands

Johnny Park
jpark@purdue.edu
Purdue University
West Lafayette, IN

Richard Kleihorst
richard.kleihorst@nxp.com
NXP Semiconductors
Eindhoven, The Netherlands

Avinash Kak
kak@purdue.edu
Purdue University
West Lafayette, IN

Abstract

In this paper, we present an implementation on an SIMD parallel processor of an object tracker based on a color-based particle filter. The main focus of our work is on the parallel computation of the particle weights, which is the major bottleneck in standard implementations of the color-based particle filter since it requires the knowledge of the histograms of the regions surrounding each hypothesized target position. In addition to that, we also show that the remaining steps of the particle filter can be efficiently implemented on an SIMD processor. We have implemented the algorithm in a low-power SIMD-based smart camera, and the experiments show that it performs robust tracking at 30 fps.

Categories and Subject Descriptors

I.4.8 [**Image Processing and Computer Vision**]: Scene Analysis—*Color, Tracking*

General Terms

Algorithms

Keywords

Particle Filter, Smart Cameras, SIMD Processors, Tracking

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 bound 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, which has been employed over the past decade by many research groups to track non-rigid objects based on their color histograms [4, 15, 16, 17]. 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 [15, 17].

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 [19]). On the other hand, the algorithm lends itself to effective parallel implementation. Therefore, by devising a parallel implementation of the color-based particle filter, 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 [8].

This paper is an extension of our previous work [14] where we proposed a method for the parallel computation of the particle weights in the color-based particle filter in an SIMD processor. Here we show that not only is it possible to compute the particle weights in parallel, but it is also possible to achieve an efficient implementation of the remaining steps of the algorithm that are not immediately parallelizable. These steps include weight normalization, estimation of the target position, and resampling. Furthermore, we have successfully implemented the algorithm in the WiCa SIMD-based smart camera, and the experimental results show that the algorithm is capable of performing robust real-time tracking. In this paper, we assume the reader is familiar with the basic concepts of the particle filter as well as its usual color-based implementation for object tracking. For introductions to the particle filter, we refer the reader to [1, 14]. The color-based particle filter for object tracking is described in [14, 15, 17].

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. Section 3 presents our proposed parallel implementation of the algorithm. In section 4, we present experimental results obtained using an SIMD-based smart camera. Finally, section 5 concludes the paper.

2 Related Work

The particle filter was introduced to the computer vision community by Isard and Blake in their seminal work [6] 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.* [15] and by Pérez *et al.* [17] 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 Bhattacharyya 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.

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 [5], but their work does not consider parallel implementation issues.

Regarding parallel computation of the particle filter, many works have shown 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 [12], 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.* [2, 3] 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.

Sutharsan *et al.* [20] 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 [2]. 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 Parallel Implementation of the Particle Filter

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 of the algorithm [17]. 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.

3.1 Hardware Architecture

We propose an algorithm for an SIMD linear processor array such as the Xetal family of SIMD processors [7]. The architecture is composed of a linear processor array (LPA) of P processing elements, each consisting of an arithmetic logic unit and a small amount of memory. Each processing element has direct read and write access to the memory of its two nearest neighbors. The line memory, that is the overall memory of the PEs, can be directly accessed by a digital input processor and by a digital output processor. The digital input processor is responsible for parallelizing the data received from the image sensor or from the external memory and for storing them in the buffer. The digital output processor reads data from the line memory and serializes them to be stored in the external memory. The digital I/O processors can operate independently of the processing elements so that, while the processing elements are performing computations on the data, the digital I/O processors

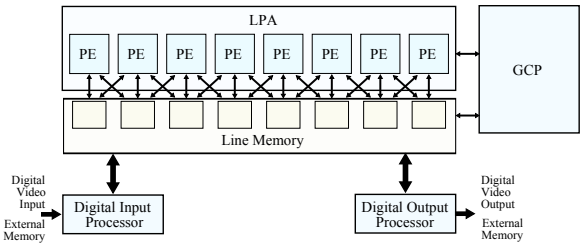

Figure 1. Hardware architecture.

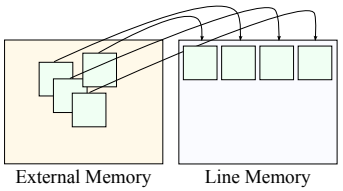

Figure 2. Organization of the particle regions in the line memory.

may store previously processed data or fetch new data from the image sensor or from the external memory. A global control processor (GCP) is responsible for controlling the operation of the LPA and is also able to carry out global DSP operations. This architecture is illustrated in Figure 1.

3.2 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 is:

$$R(\mathbf{x}) = R(x,y) = \{(u,v) : x \leq u \leq x+r_x, y \leq v \leq y+r_y\} \quad (1)$$

One straightforward approach to compute the histogram of region $R(\mathbf{x})$ would be to employ integral histograms [18]. 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.

In our approach, we compute the histograms of M image regions in parallel. To do so, we first store the image in the external memory so that its pixels can be randomly accessed by the digital I/O processors and, consequently, by the PEs. Once the PEs have random access to the image pixels, it is possible to read them back into the line memory of the LPA reorganized side-by-side, as illustrated in Figure 2, so that they can be later processed in parallel. To read the pixels from the external memory, each processing element has to know the initial coordinates, x_i and y_i, and the dimensions, r_x and r_y, of the region R_i.

In practice, not all the pixel information has to be stored in the external memory. Since the pixels are used only for the computation of the histograms of each region, we only need to store the bin number corresponding to the given pixel in the external memory. Using that approach, only $\log_2 m$ bits are required to store each pixel, where m is the number of histogram bins.

As the image regions are read, line-by-line, into the line memory, they can be processed in parallel using an approach somewhat similar to that used in [9, 10], which is illustrated in Figure 3. During the first r_y iterations, the histograms for each column of the regions are computed. 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 1. The algorithm runs in parallel in each processing element i. Using this procedure, it is possible to compute the histograms of all the image regions in $O(r_x+r_y)$ steps. The

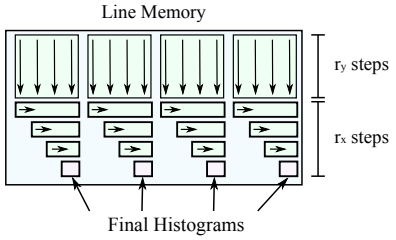

Figure 3. Parallel computation of the histograms.

Algorithm 1 Computing the histograms in parallel.

For $j = 0$ **To** r_y
  read position $(x_i + i \mod r_x, y_i + j)$ from the external memory into the line memory

  update the column histogram and store it in the line memory
End For

For $k = 0$ **To** r_x
  If $(i \mod r_x) = k$
    add the current column histogram to the histogram of the right neighbor
  End If
End For

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 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 processors 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 suggested in [14]. Using this arrangement, $\frac{M \times r_x}{n_x}$ extra steps are necessary to compute the histograms of all the regions. However, when the total width of the tracked regions $M \times r_x$ is much larger than the number of processing elements n_x, the number of extra steps required to compute the histograms may not be negligible. If the total time required for computing the histograms of the image regions in parallel exceeds the time required to compute the histogram of each region, that is, if $\frac{M \times r_x}{n_x}(r_x + r_y) > r_x r_y$, or $M > \frac{n_x r_y}{(r_x + r_y)}$, then it is more efficient to organize the regions as columns in the line memory. In that case, each PE is responsible for computing the histogram of one region so that up to n_x histograms can be computed in parallel. This approach is illustrated in Figure 4. In that case, the algorithm for computing the region histograms is given by Algorithm 2.

3.3 Parallel Weight Computation

Since there are no data dependencies among the particles during the computation of the likelihoods, 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. Since each PE has access to its immediate neighbors, weight normalization can be carried out by left (or right) shifting the weights and accumulating the total weight over all elements. The total weight can then be used by

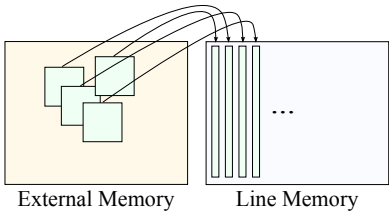

Figure 4. Organization of the particle regions into columns in the line memory.

Algorithm 2 Computing the histograms in parallel for a large number of particles.

For $j = 0$ **To** r_y
  For $k = 0$ **To** r_x
    read position $(x_i + k, y_i + j)$ from the external memory into the line memory

    update the histogram and store it in the line memory
  End For
End For

the GCP to compute a global scale factor which is then multiplied by all the weights in parallel. Regardless of the approach used to compute the histograms, weight normalization can be accomplished in at most $O(n_x)$ steps.

3.4 Resampling

As we already mentioned, resampling cannot, in general, be implemented in parallel due to data dependencies among the particles during this step of the algorithm. However, for a moderate number of particles, resampling is not computationally expensive. In particular, given the normalized weights, the replication factors r_i can be computed over a single iteration over the particles in most resampling algorithms such as Systematic Resampling and Residual Resampling [11]. Given the replication factors, the particles can be sequentially replicated in $O(n_x)$ steps by using the direct memory access among neighboring PEs.

3.5 Pipelined Implementation

In a straightforward implementation, two frame intervals would be necessary to estimate the target position since the pixel information of an entire frame must be stored in the external memory before it can be reorganized in the line memory. That is, during the first frame interval, the pixel information needs to be stored in the external memory. Then, during the second frame interval, the pixel information of the regions surrounding each particle is loaded into the line memory of the LPA as described in Section 3.2 in order to allow the parallel computation of the particle weights.

In our system, we use only half of the image resolution (i.e., only the odd video lines of an image) so that two half images can be processed in a single frame interval. To do so, during the odd video line intervals, we store the corresponding pixel information in an external memory space allocated for the current frame. Then, during the even video line intervals, we read the pixel information of the regions surrounding each particle from a different external memory space corresponding to the previous frame. As the pixel information of the previous frame is read into the line memory, it is reorganized as described in Section 3.2 so that the particle weights can be computed in parallel and the target position can be estimated. As a result, after one frame of latency, the filter provides one estimate of the target position at every frame, achieving a frame rate of 30 fps.

4 Experimental Results

We implemented our algorithm on the WiCa smart camera [8]. The WiCa basically consists of four main components, one VGA

color image sensor, one IC3D/Xetal SIMD processor, one general purpose processor, and a Dual Port RAM (DPRAM) which is shared between the SIMD processor and the general purpose processor. The IC3D/Xetal SIMD processor consists of a linear processor array (LPA) with 320 RISC processors, one digital input processor, one digital output processor, and one global control processor (GCP). Each processor in the LPA is endowed with 64 words (10 bits wide) of memory and can directly access the memory of its immediate neighbors. Data is streamed into the LPA memory by the digital input processor and out of the LPA memory by the digital output processor. The GCP, in addition to controlling the operation of the LPA, is also capable of performing global DSP operations. All the steps of the particle filter were implemented in the SIMD processor, and the general purpose processor was not used except for debug purposes.

In our current implementation, the target is tracked on images of resolution of 256×240 pixels, which are obtained by downsampling the VGA frames provided by the image sensor. The measurements are based on the hue histogram of the target. Due to memory constraints, 40-bin histograms were used. Each tracked region consists of a rectangular area of 15×15 pixels. The distribution of the target state is approximated by 240 particles. Since we are employing a large number of particles, the histograms are organized into columns in the memory (as described in Section 3.2), so that each PE is responsible for one particle. To efficiently use the I/O capabilities of the platform, only the rightmost 240 PEs are used in the computations. The remaining 80 PEs could be used (e.g., to keep track of more particles) at the cost of a more complex I/O management strategy.

The WiCa architecture provides access to up to 640 elements to/from the external memory at each video line interval. However, for simplicity of implementation, we only read 240 elements (one pixel per particle) from the external memory at each video line interval. Since the image sensor provides VGA frames, we have 480 video line intervals to access the external memory. However, half of these line intervals are used for storing the hue values of the pixels of the current frame into the external memory, hence, we have 240 remaining line intervals for reading the reorganized tracked regions into the line memory. Therefore, in our current implementation, one iteration of histogram computation is carried out at each even video line interval. It is possible to increase this by a factor of $640/240 = 2.67$ at the cost of a more complex I/O management strategy.

As the pixels are loaded into the line memory, the histograms of each particle region are computed in parallel. After the histograms are computed, the Bhattacharyya distances between each of the 240 histograms and the reference histogram are computed in parallel. The observation likelihoods and unnormalized particle weights are also computed in parallel based on the Bhattacharyya distances. Since each PE in the LPA has single-cycle read and write access to the memory of its immediate neighbors, weight normalization as described in Section 3.3 can be carried out efficiently.

The target state $\mathbf{x}_t = [x_t, y_t]^T$ consists of the current pixel coordinates of the target, and its dynamic behavior is modeled by a first order autoregressive process, that is:

$$\mathbf{x}_{t+1} = A\mathbf{x}_t + B\mathbf{x}_{t-1} + \mathbf{n} \quad (2)$$

where $\mathbf{x}_t$ is the target state at time t, A and B are the process parameter matrices and $\mathbf{n} \sim N(0, \sigma)$ is the process noise. The estimated target position is given by the weighted average of the particles. This step can be implemented efficiently using two additional shift/accumulate steps similar to that used to normalize the particle weights, one for the x coordinate and another one for the y coordinate of the target.

At every iteration of the filter, we perform a simplified residual resampling step. The replication factors are given by $r_i = \lfloor M \times w_i \rfloor + n_i$, where $n_i = M - \sum_i \lfloor M \times w_i \rfloor$ if $w_i = \max(w_i)$ and 0 otherwise. Given the normalized weights, the replication factors can be computed in parallel. Then the particles are replicated sequentially by shifting each particle i r_i times into its neighbors using the direct access to the immediate neighbors.

Conversion to HSV	$256 \times 12.4\mu s$
Histogram computation	$225 \times 2.4\mu s$
Particle prediction	$7.2\mu s$
Bhattacharyya distances	$79.6\mu s$
Likelihoods	$3.6\mu s$
Weight normalization	$31.9\mu s/5.2\mu s$
Weighted average of the particles	$2 \times 40.9\mu s/4.9\mu s$
Resampling	$102\mu s$
Total	$4ms$

Table 1. Processing times.

Target state (current and previous)	4
Particle weights	1
Histograms	40
Video line buffers	3
Temporary variables	14
Total	62

Table 2. Line memory usage.

Table 1 shows the time required for each processing step of the algorithm. As we already mentioned, 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 $12.4\mu s$ per image line. Since one image line period is approximately $69.5\mu s$, during the odd video line intervals, the LPA is busy approximately 17% of the time.

Since we choose to organize the particle regions as columns in the line memory, during the even video lines, every step of the algorithm is computed in parallel for all the particles. Histogram computation, the first step of the algorithm executed during the even video lines, requires $n_x \times n_y$ iterations of $2.4\mu s$. In our implementation, $n_x = n_y = 15$, and, since one element per PE is read into the line memory at each video line interval, 225 video lines are required for computing the histograms. The total processing time required for histogram computation is $225 \times 2.4\mu s = 540\mu s$.

Particle prediction using Eq. (2), which requires the generation of two Gaussian random numbers, takes $7.2\mu s$ for both the x and y coordinates of all of the 240 particles. Since weight normalization as described in Section 3.3 requires the use of iterative routines for integer division, the computation time is data-dependent. Table 1 shows the average computation time and the standard deviation over 20 iterations of the algorithm while tracking a target. The same thing happens with the computation of the weighted average of the particles. Since the resampling factors are computed during weight normalization, the resampling time shown in the table corresponds to the time to copy the replicated particles to their corresponding PEs. The total computation time during one frame is approximately $4ms$, or 12% of one frame interval.

Table 2 shows the line memory usage of the algorithm. As we can see, more than 64% of the line memory is used to store the histograms. Excluding the 3 line memories used for storing the RGB digital video from the image sensor and the 40 line memories used by the histograms, the entire algorithm requires only 19 line memories for storing the target state and temporary variables. Regarding program memory usage, the current implementation of the algorithm consists of 1362 instructions out the 2048 instructions of program memory available in the IC3D/Xetal processor.

Figure 5 shows snapshots of the tracking results of our implementation of the parallel color-based particle filter in the WiCa cam-

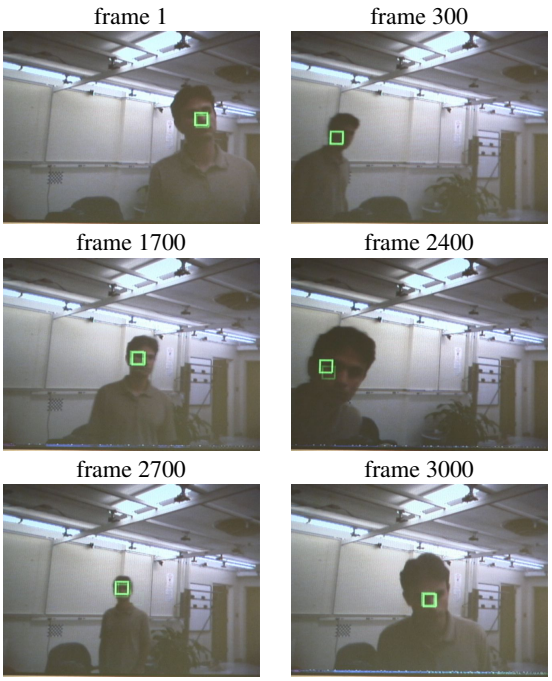

Figure 5. Tracking results.

era. In the images, it is possible to see that the algorithm is capable of keeping track of a human face over a large number of frames even in the presence of objects of similar colors and under large variations on the appearance of the target.

5 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 not only is it possible to implement the major bottleneck of the algorithm – the computation of the color 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. Our results show that it is possible to achieve real-time tracking even operating at relatively low clock frequencies.

The algorithm can be immediately extended to include spatial information in the tracked histograms. For example, as in [17, 15], pixels within a tracked region may be weighted so that pixels near the borders of the regions are less relevant in the histogram computation, or, as in [17], a coarse spatial layout of the tracked region can be incorporated into the tracker. In addition to that, since the processing elements have non-linear access to the external memory, it is straightforward to keep track of regions of different shapes such as ellipses or rectangles.

One of the major limitations of our current approach is the necessity to store all the histogram bins in the line memory. However, since the histograms are only needed to compute the Bhattacharyya distances, it should be possible to overcome this limitation by recursively computing the Bhattacharyya distance. That would not only allow us to employ histograms with more bins, increasing the overall robustness of the system, but also release most of the line memory for different uses. It should be possible, for example, to keep track of more complex target states or even track multiple objects.

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. It should be possible, for example, to employ a cluster-based architecture [13] in which the cluster head is responsible for robustly estimating the 3D coordinates of the target and automatically detecting and reinitializing cameras that lose track of the target. Since our ultimate goal is to port this method to wireless cameras, we are trying to achieve such collaboration while keeping the interaction among cameras to the minimum necessary.

6 References

[1] ARULAMPALAM, M., MASKELL, S., GORDON, N., AND CLAPP, T. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *Signal Processing, IEEE Transactions on 50*, 2 (Feb. 2002), 174–188.

[2] BOLIC, M. *Architectures for Efficient Implementation of Particle Filters*. PhD thesis, Stony Brook University, Aug. 2004.

[3] BOLIC, M., DJURIC, P., AND HONG, S. Resampling algorithms and architectures for distributed particle filters. *IEEE Transactions on Signal Processing 53* (2005), 2442–2450.

[4] CZYZ, J., RISTIC, B., AND MACQ, B. 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)* (Mar. 2005), vol. 2, pp. 217–220.

[5] FLECK, S., AND STRASSER, W. Adaptive probabilistic tracking embedded in a smart camera. *Computer Vision and Pattern Recognition, 2005 IEEE Computer Society Conference on 3* (2005), 134–134.

[6] ISARD, M., AND BLAKE, A. Condensation – conditional density propagation for visual tracking. *International Journal of Computer Vision 29*, 1 (1998), 5–28.

[7] KLEIHORST, R., ABBO, A., VAN DER AVOIRD, A., OP DE BEECK, M., SEVAT, L., WIELAGE, P., VAN VEEN, R., AND VAN HERTEN, H. Xetal: a low-power high-performance smart camera processor. In *The 2001 IEEE International Symposium on Circuits and Systems, ISCAS 2001.* (2001), vol. 5, pp. 215–218.

[8] KLEIHORST, R., SCHUELER, B., DANILIN, A., AND HEIJLIGERS, M. Smart Camera Mote With High Performance Vision System. In *Proceedings of the International Workshop on Distributed Smart Cameras (DSC-06)* (31 Oct. 2006).

[9] KYO, S., OKAZAKI, S., AND ARAI, T. An integrated memory array processor for embedded image recognition systems. *IEEE Transactions on Computers 56*, 5 (2007), 622–634.

[10] KYO, S., AND SATO, S. 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.

[11] LIU, J. S., AND CHEN, R. Sequential monte carlo methods for dynamic systems. *Journal of the American Statistical Association 93* (1998), 1032–1044.

[12] MASKELL, S., ALUN-JONES, B., AND MACLEOD, M. A Single Instruction Multiple Data Particle Filter. In *Proceedings of Nonlinear Statistical Signal Processing Workshop* (2006).

[13] MEDEIROS, H., PARK, J., AND KAK, A. A Light-Weight Event-Driven Protocol for Sensor Clustering in Wireless Camera Networks. In *First IEEE/ACM International Conference on Distributed Smart Cameras* (Sept. 2007).

[14] MEDEIROS, H., PARK, J., AND KAK, A. 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.* (June 2008), pp. 1–8.

[15] NUMMIARO, K., KOLLER-MEIER, E., AND GOOL, L. V. A Color-Based Particle Filter. In *First International Workshop on Generative-Model-Based Vision* (2002).

[16] OKUMA, K., TALEGHANI, A., DE FREITAS, N., LITTLE, J., AND LOWE, D. A boosted particle filter: Multitarget detection and tracking. In *Proc. ECCV, volume 3021 of LNCS,* (2004), Springer, pp. 28–39.

[17] PÉREZ, P., HUE, C., VERMAAK, J., AND GANGNET, M. Color-based probabilistic tracking. In *ECCV '02: Proceedings of the 7th European Conference on Computer Vision-Part I* (London, UK, 2002), Springer-Verlag, pp. 661–675.

[18] PORIKLI, F. 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.

[19] RAHIMI, M., BAER, R., IROEZI, O. I., GARCIA, J. C., WARRIOR, J., ESTRIN, D., AND SRIVASTAVA, M. Cyclops: in situ image sensing and interpretation in wireless sensor networks. In *Proceedings of the 3rd international conference on Embedded networked sensor systems* (2005).

[20] SUTHARSAN, S., SINHA, A., KIRUBARAJAN, T., AND FAROOQ, M. An optimization based parallel particle filter for multitarget tracking. In *Proceedings of the Spie,* (Jan. 2005), vol. 5913, pp. 87–98.