Available online at www.sciencedirect.com

SCIENCE DIRECT

Real-Time Imaging 10 (2004) 145–159

www.elsevier.com/locate/rti

A robust hardware algorithm for real-time object tracking in video sequences

Mahmoud Meribout[a,*], Lazher Khriji[a], Mamoru Nakanishi[b]

[a] *ECE Department, College of Engineering, Sultan Qaboos University, Muscat, Oman*
[b] *Integrated Information and Energy Laboratories. NTT corporation, Kanagawa 243-0192, Japan*

Abstract

Most of the emerging content-based multimedia technologies are based on efficient methods to solve machine early vision tasks. Among other tasks, object segmentation is perhaps the most important problem in single image processing. The solution of this problem is the key technology of the development of the majority of leading-edge interactive video communication technology and telepresence systems. The aim of this paper is to present a robust framework for real-time object segmentation and tracking in video sequences taken simultaneously from different perspectives. The other contribution of the paper is to present a new dedicated parallel hardware architecture. It is composed of a mixture of Digital Signal Processing and Field Programmable Gate Array technologies and uses the Content Addressable Memory as a main processing unit. Experimental results indicate that small amount of hardware can deliver real-time performance and high accuracy. This is an improvement over previous systems, where execution time of the second-order using a greater amount of hardware has been proposed.
© 2004 Elsevier Ltd. All rights reserved.

1. Introduction

Real-time Object segmentation and tracking in video sequences is among the important leading-edge technologies, which substantially support multimedia application. As example, to provide new functionalities and increase the compression rate, the MPEG-4 video standard should encode content-based video data with the aim of using and presenting it in highly flexible way [1,2]. A second example concerns the World Wide Web, which is being increasingly populated. Hence, there is growing need for effective indexing and retrieval of large amounts of such media data.

The aim of this paper is to describe a new framework for object segmentation/tracking in video sequences. Object segmentation and tracking for complex problems have been plagued by both poor speed and a high incidence of false alarms [1,2]. The primary source of these problems is clutter in the image. Such image clutter not only requires considerable computations to process, but also yields false positive instances of the objects that are sought. Solutions to these problems have been only partially successful. In this project an approach to image object tracking on image sequences is given. Hence, a semantic object is modeled as a set of regions with corresponding spatial and visual features and we suppose that semantic knowledge of the first frame of the scene is already performed. This model directly links the semantic object to its underlying feature regions of the segmentation of the frame. The same algorithm could be extended to content-based video retrieval.

Besides the algorithm's accuracy, another not less important feature in the field of multimedia is the time complexity. Hence, most of the works on object segmentation and tracking algorithms are still unrealistic to implement in real-time because of the irregular and huge data structure of the algorithms. Therefore, several approaches have been proposed to map these algorithms in highly parallel hardware machines [3–5]. In [6], we have shown how a real-time line extraction can be implemented in a compact, low cost, and highly parallel board namely, Highly Parallel Integrated Circuit system (HiPIC). Recently, the board has been designed and completely implemented. Thus, in this paper we are particularly interested to address the performance of

*Corresponding author. Department of Electronics, College of Engineering, Sultan Qaboos University, P.O. Box 33, El-Khod, Oman. Fax: +501-6941-418.
E-mail address: kassantina@hotmail.com (M. Meribout).

1077-2014/$ - see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.rti.2004.05.002

that machine for our new object video segmentation/tracking algorithm. The other contribution of the paper is then to show how a parallel implementation of these algorithms has been achieved in the HiPIC board. A top–down software/hardware design methodology has been followed for this purpose. Comparison with other previous implementations indicates clearly the superiority of our HiPIC concept.

This paper is organized as follows. The core algorithm is briefly reviewed in Section 2. More details of the algorithm is given in Section 3. In Section 4, comprehensive complexity analysis for these algorithms is presented. Based on this analysis, we describe the design space exploration and optimization for real-time implementations. Finally, we summarize our conclusion in Section 5.

2. Overview of the algorithms and some related previous works

Automatic object tracking and segmentation of semantic objects is difficult expect for specific application domains. The rich set of features and spatio-temporal structural information at the boundary of objects has also proved to be effective in video indexing [7]. As shown in Fig. 1(a), the integrated framework extracts and tracks 2D objects. This is suitable for object-base coding such as MPEG4. Key research issues in this paper lie in (1) extract accurate boundaries on which it performs tracking, and (2) demonstrate how this algorithm can be amenable to a parallel framework.

Object-based video processing generally belongs to two principal classes, namely region- and shape-based techniques. The successfulness of one or the other class depends primarily on the degree of texture in the scene [8,9]. Shape-based techniques have been shown to be appropriate for lowly textured scenes. In this paper we are particularly interested to address such kind of scenes.

2.1. *Some previous works on active video object segmentation and tracking algorithm*

One of the major challenges facing multimedia-based algorithms is to achieve very low bit rate coding. An integrated spatio-temporal framework, which uses block-based motion estimation and DCT have been widely adopted for this purpose. However, in case of zoom or rotation of objects in the video scene the performance of these motion algorithms degrades dramatically, and the encoder provides a low compression ratio. Many approaches have then been developed using the motion field or optical flow. Wang and Andelson in [10] presented an affine clustering-based algorithm. In [11], instead of using optical flow, Ayer and Sawhney propose a method to estimate motion models and their layer support simultaneously. In [12], Meyer and Bouthemy developed a pursuit algorithm to track an object based on the multiresolution estimation of the affine model from the motion field within the object. In general, the above methods concentrate on segmenting objects and cannot track static objects within intermittent motions (e.g. objects stop and move between frames). Furthermore, due to the accuracy limitation, motion segmentation may not give clear object boundaries.

Recently, with the demand for object tracking in videos and the requirement of more accurate segmentation boundaries, region based methods, which combine common image segmentation techniques with motion estimation methods have been reported in [13,14]. In [13], Dubuisson and Jain presented an approach to combine motion segmentation using image subtraction with static color segmentation using the split and merge paradigm. The motion field of the object is

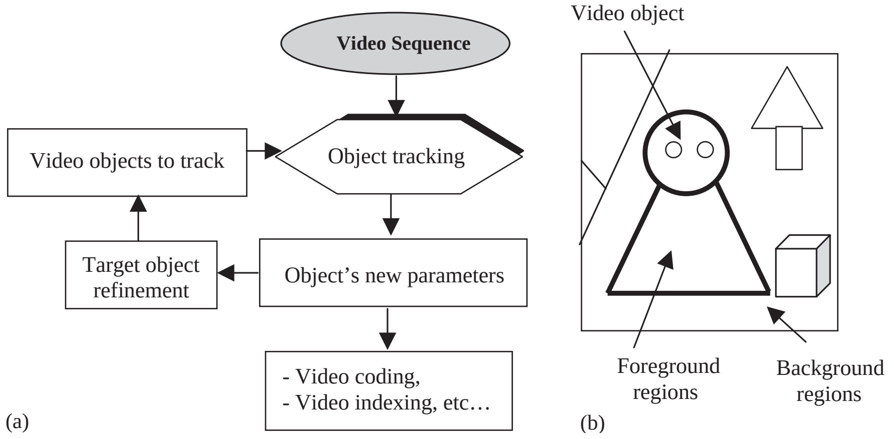


Fig. 1. General framework of the system for: (a) 2D video segmentation and (b) Model of input video sequence.

simultaneously performed and the combination of the two segmentation results is done only at the final stage using certain heuristic rules. In [14], Gu and Lee proposed a semantic object tracking system using mathematical morphology and perspective motion.

Satisfactory results for these works were reported for a certain type of video content e.g. with simple motions, constant illumination (since the region growing and matching methods use basically the color information), and no occlusion. In addition, due to the noisy nature of their motion field in real-world scenes, tracking results may be error-prone. Also, as there are no constraints being applied between motion and static segmentation, when the two results are different from each other, its hard to align them to generate the final object mask. Furthermore, these techniques tend to ignore the background content process. This may cause problems in tracking regions near the boundary of the object. Finally, most of these algorithms are not parallelizable which lead to high execution time. This makes hard to meet one of the most important aspect in video processing: real-time performance.

Other approaches use the shape boundary as main information for object tracking. Mathematicians typically define shape as an equivalence class under a group of transformations. This definition is incomplete in the context of visual analysis. This only tells us when two shapes are exactly the same. We need more than that or a theory of shape similarity or shape distance. An extensive survey of shape matching in computer vision can be found in [1,2]. In [15], a real-time gradient-based segmentation hardware algorithm was developed to match edge detection and line approximation results with motion segmentation. Hence, neighboring pixels having a similar gradient direction are grouped in a same line. However, the weak performance of this algorithm in a noisy or cluttered scene results in fragmented and spurious line segments, which enhances dramatically the complexity of post-processing tasks. Additional procedures such as line merging and vertices extraction [16] become then necessary. In addition, the edge detection operator must be strong enough to avoid noisy and thick boundaries [17]. Overall, the method is not well accurate in case the object's boundary is not smooth. Another method, the Generalized HT (GHT), which explores the gradient of the luminosity function, has been widely adopted for 2D video-based segmentation [8,18]. Its robustness against noise constitutes one of its major advantages. Its difficulty, however, is the large hardware amount required in terms of memory and processing power since a large four-dimensional (4D) accumulator array needs to be processed for each video frame. Thus, this algorithm has been mainly applied to static images. Several variants of the GHT [8,9,18] have been reported in order to reduce its complexity. Most of them primarily proceed by grouping some similar pixel points that are likely to belong to the same object. This yields improvements for both the speed and accuracy of object segmentation. However their clustering criteria made these algorithms limited to extract only particular features of the image such as edge corners.

To solve the above problems for general video sources, we developed an active system that uses an innovative method for combining object segmentation and tracking methods. It is based on the Generalized HT (GHT) algorithm and uses a new grouping technique, which clusters more features than other previous grouping methods. Furthermore, it is amenable to a parallel framework yielding to a significant improvement of the computation time and memory complexity.

3. New video-based GHT algorithm for 2D object segmentation and tracking

In our approach, we treat an object as a point set and we assume that the shape of an object is essentially captured by a finite subset of its points. More practically, a shape is represented by a discrete set of points sampled from the contours on the object. These can be obtained as locations of edge pixels as found by an edge detector, giving us a set $P = \{p_1, p_2, \ldots, p_n\}$, of n points. These points need not correspond to key points such as maxima of curvature or inflection points. In the following, we first review the concept of GHT, which uses feature points to recognize 2D objects. The algorithm is then extended for tracking in a sequence of images.

3.1. Overview of the GHT algorithm

The key idea of the GHT algorithm is that when many sets of matches between image features and object model features are mapped into the space of object positions that bring them into alignment, a cluster forms at the correct position of the object, if it appears in the image. Thus, in the GHT, the search pattern is parameterized as a set of vectors from feature points (edges) in the pattern to a fixed reference point, referred to as *RP*. To locate the pattern in an image, each of these features is considered and the corresponding locations of the reference point are calculated. Assume that the shape, scale S, and rotation τ of the desired region are known. A reference point, $O(x_R, y_R)$, is chosen at any location inside the sample region. Then an arbitrary line can be constructed at this reference point aiming in the direction of the region border (Fig. 2(a)). This line intercepts the boundary template at the point $M(x, y)$. The gradient direction ϕ, the distances of the reference point to the region border, r, and the angle γ for the

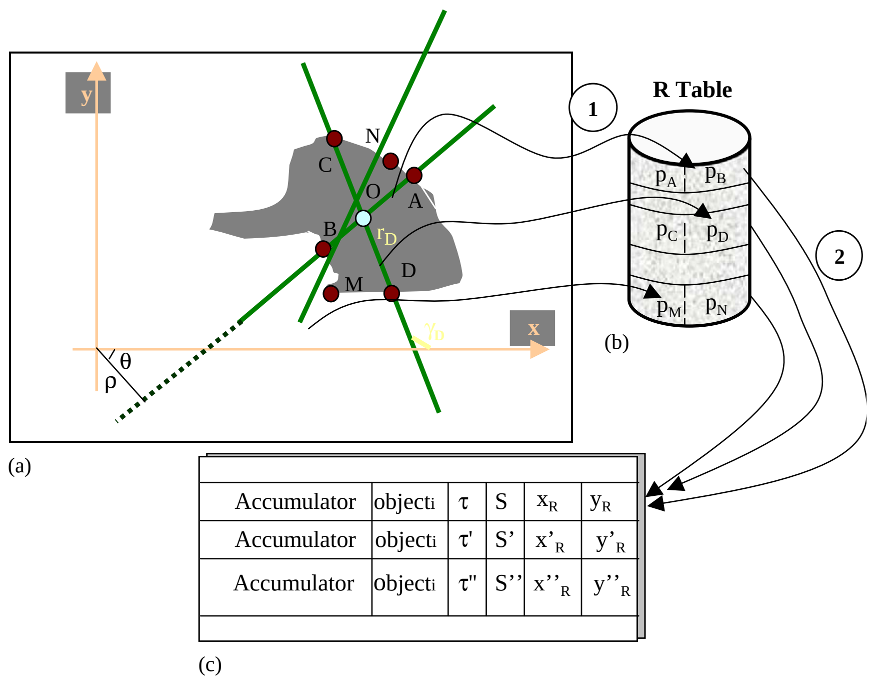

y
x
C
N
O
A
B
r_D
D
M
γ_D
θ
ρ
(a)
R Table
1
2
p_A p_B
p_C p_D
p_M p_N
(b)
Accumulator objecti τ S x_R y_R
Accumulator objecti τ' S' x'_R y'_R
Accumulator objecti τ'' S'' x''_R y''_R
(c)

Fig. 2. Principle of the new object segmentation and tracking algorithm: (a) image space; (b) Π_1 parameter space; (c) Π_2 parameter space (5D).

segment OM are then determined as a geometrical template. The first process consists then to construct a reference table using ϕ as a key and (r, γ) as data. This process is repeated for several other boundary points of the template and the reference table progressively filled for each of them. In the matching phase, the 4D space (S, τ, x_R, y_R) is investigated. When the figure with the scale S_u and rotation τ_v is the object for extracting, the edge direction ϕ_D is determined for an edge point $D(x_D, y_D)$ in an image and the nearest value to $\phi_D - \tau_u$, ϕ_i, is searched in keys of the reference table. By using the values of r_D and γ_D, which are indexed by ϕ_i, the candidate coordinates (x_R, y_R) of the RP, are determined using the following equations:

$$x_R = x_D + r_D \times S_u \times \cos(\gamma_D + \tau_v), \tag{1}$$

$$y_R = y_D + r_D \times S_u \times \sin(\gamma_D + \tau_v). \tag{2}$$

Then a voting process for increasing the value of the coordinates (S_u, τ_v, x_R, y_R) of the 4D parameter space is done. Calculating candidate coordinates of the RP and voting process to coordinates are executed for several values of S and τ. After all video frame features have been processed, the 4D accumulator array will contain high value (peak) for locations where many image features coincide with many pattern features. High peaks (relative to the number of features in the pattern) correspond to reference point locations where instances of the pattern occur in the image.

One major advantage of the above algorithm is its high robustness against noise. However, its main difficulty is the huge 4D parameter space, which can be increased to 5D if different target objects have to be detected. Even though some parts of the algorithm feature a parallel data structure, the huge parameter space makes difficult its real-time implementation.

3.2. *Our segmentation framework: new GHT for video-based 2D object extraction and tracking*

To reduce the time complexity of the GHT and improve its accuracy, our idea is to decompose the segmentation framework into several sub-problems that must be examined iteratively. It can be formulated as a data-clustering problem, where data exhibiting similar features are grouped together by pairwise data clustering under additional topological considerations. Special attention has to be given to the aspect of real time optimization. This is accomplished by parallelizing the algorithm. Hence, an additional 2D parameter space from which the two end-points, A and B, belonging to a same line support, are extracted. Their invariant features to scale, rotation, and translation, p_A and p_B, are then compared to the whole list of its reference table (referred in Fig. 2(b) as an R-table). The 5D parameter space is then voted (Fig. 2(c)). As it will be shown in the next section, the structure of the R-table is different from the reference table used in the classical GHT. The improvement here is that a less number of edge pixels, though meaningful ones contribute to the 5D parameter space. This is because the probability of error for a pair of two edge points to fit one element pair in the R-table is lower than that of a single edge point. Therefore, it is possible for our technique to not only distinguish sets of points that are likely to belong to the same object, but also to reduce the amount of search that is necessary in matching points between an object model and points in the image that are known to be from the object, by producing only certain subsets of the points among all possibilities. This makes the detection more accurate against noise with a faster convergence. In addition, the 5D parameter space is not considered as a whole but progressively filled in a hash table. Another advantage of this method is that the range of the rotation and scale is not initially fixed, but progressively calculated using different features of the candidate pairs of edge points. As a result, the processing power and memory usage are improved.

The detection of the target objects, labeled *object$_i$*, is preceded by an off-line model preprocessing, in which model data is statistically compiled into an R-table for fast access in the segmentation stage. This is followed by the segmentation and tracking phase.

3.2.1. *Segmentation of the initial frame and design of the R-table*

The frame is supposed to be already segmented, either manually or automatically, using any segmentation technique [1,2]. In our experiments, the user identifies a semantic object of the frame by using a tracing interface (e.g. mouse. The input is a polygon, whose vertices and edges exist roughly along the desired object boundary). Next, an edge detection algorithm is applied to the image. The gradient is usually calculated on the luminance component. However, sometimes luminance variations are very small along the borders of the adjacent objects. In such cases, we may get false boundaries due to the resulting small gradients. To overcome this problem, we incorporate color information into gradient computation. Let $g_Y(x,y)$ denote the gradient of the pixel located at the address (x,y) of the image space, which is obtained from luminance information and $g_C(x,y)$ denote the gradient obtained from the color information, then the incorporated gradient is given by:

$$g(x,y) = \max(g_Y(x,y), g_C(x,y)) \tag{3}$$

$g_Y(x,y)$ is calculated on the Y component in the YcbCr color space using (3), and is calculated as follows:

$$g_C(x,y) = \sqrt{(g_L(x,y))^2 + (g_a(x,y))^2 + (g_b(x,y))^2}, \tag{4}$$

where, $g_L(x,y)$, $g_a(x,y)$, and $g_b(x,y)$, are calculated on the L, a, b component in the (L,a,b) color space. Hence, in order to get the gradient from color information, we use a nonlinear color space conversion, the (L,a,b) color space [14], which is related to the CIE XYZ standard observer through a nonlinear transformation. The (L,a,b) color space is a suitable choice for this purpose because it is a perceptually equalized color space, i.e., the numerical distance in this space is proportional to perceived color difference. The transformation of YCbCr (with ITU-R Rec. 624-4 specs) to RGB can be represented as follows [19]:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.4022 \\ 1 & -0.3456 & -0.7145 \\ 1 & 1.7710 & 0 \end{bmatrix} \begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix}. \tag{5}$$

L, a, and b values can be obtained as following:

$$L^* = \begin{cases} 116^*\left(\dfrac{Y}{Y_n}\right) & \text{for } \dfrac{Y}{Y_n} > 0.00885 \\ 903.3^*\dfrac{Y}{Y_n} & \text{otherwise} \end{cases}$$

and,

$$\begin{cases} a^* = 500^*\left(f\left(\dfrac{X}{X_n}\right) - f\left(\dfrac{Y}{Y_n}\right)\right), \\ b^* = 200\left(f\left(\dfrac{Y}{Y_n}\right) - f\left(\dfrac{Z}{Zn}\right)\right), \end{cases}$$

where (X, Y, and Z are defined as), and

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.4124 & 0.3575 & 0.1804 \\ 0.2126 & 0.7151 & 0.0721 \\ 0.0193 & 0.1191 & 0.9502 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

and

$$f(t) = \begin{cases} t^{1/3} & \text{for } t > 0.00885, \\ 7.787^*t + \dfrac{16}{116} & \text{otherwise.} \end{cases}$$

Fig. 3 shows another video scene, where the target object is the shoulder. It can be clearly seen how the

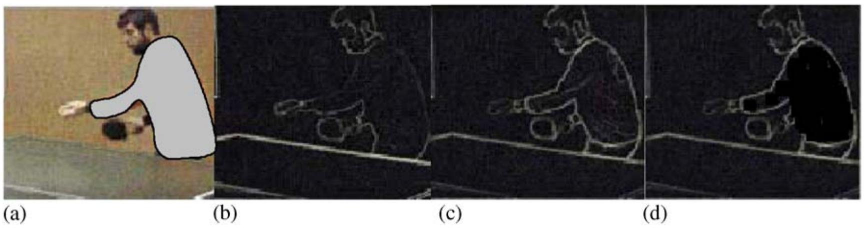


Fig 3. Example of extraction of object boundary: (a) Original image. (b) Gradient from luminance information. (c) Gradient from color information. (d) Boundary of the object after filtering.

object's contour are better defined with color information (Fig. 3(c)) than using the luminance information (Fig. 3(b)).

After the gradient information is obtained, the final step consists to find the exact boundary of the target object. Hence, only the edges, which are close to the defined boundary, are considered, whereas, all its other internal edges are rejected. Fig. 4 illustrates the principle of the algorithm. Hence, the point $M(x,y)$ is considered to belong to the object boundary (e.g. true edge) if there is no pair of edge points which are simultaneously between (M and P) and (M and Q).

Fig. 3(d) shows the final results of the boundary extraction after applying this algorithm. It can be observed how the inner edges of the target object (shoulder) could be rejected.

Fig. 5 shows the edge detection results of the frames 24, 30, and 36, respectively, of the same video scene.

The next step consists to design the R-table, which will be used during the matching process. Hence, the matching approach should be not only invariant under scaling, rotation, and translation but robust under small geometrical distortions, occlusion, and presence of outliers as well. Thus, our experience suggests that matching is easier and less ambiguous if rich descriptors, e.g. better than single edges, are used. As a key contribution, we propose a novel descriptor, the line feature that could play such role in shape matching.

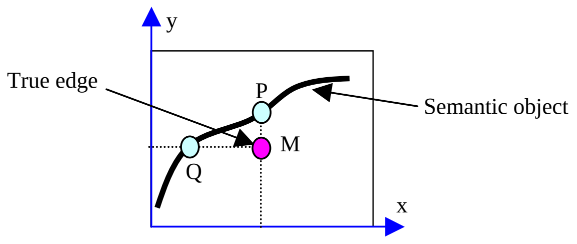


Fig. 4. Selection of true edges in the image space.

Consider the set of vectors originating from a point to all other sample points on a shape. These vectors express the configuration of the entire shape relative to the reference point. Next, to each of these vectors is applied a robust perceptual grouping mechanism. Many cues have already been used to perform efficiently grouping of image features. Some examples include parallelism, proximity, and colinearity [13,14]. In this paper, the features, p, are the angle formed by the gradient orientation and the line joining the edge pair features, and the color information of the pixels corresponding to the edge pixels as well. The selection of these features is efficient in case the target object is smooth enough (as will be demonstrated in next sections). However, in case it is highly textured, other properties using other low level features, such as the fractal analysis, can be applied [6].

Initially, the RP, $O(x_R, y_R)$, and the invariant feature p, to be used, are selected. Next, an edge point, $M(\rho_M, \alpha_M = \phi_M - \theta_M, L_M, a_M, b_M)$, which belongs to the object boundary is selected. Here, ϕ_M is the gradient direction of the edge point M, and θ_M the slope and ρ_M the distance to the origin of the line passing through M and O. L_M, a_M, b_M are the L, a, and b components, respectively, of the pixel M. The intersection of this line with the object boundary forms a second point $N(\rho_N, \alpha_N = \phi_N - \theta_N, L_N, a_N, b_N)$, where ϕ_N represents its gradient direction. The obtained angles α_M and α_N, which are invariant to rotation, along with the vector ratio $k : \overrightarrow{OM} = k \times \overrightarrow{MN}$, which is invariant to scale, the label $object_i$ of the target object being processed, the segment length [MN], and the color information are then stored into the R-table, at the address indexed by the angle α_M. This process is repeated for a predefined sampling rate and range of the slope θ. The R-table elements corresponding to another target object, $object_{i+1}$, are determined with the same manner. Fig. 2(c) shows the configuration of the resulting R-table.

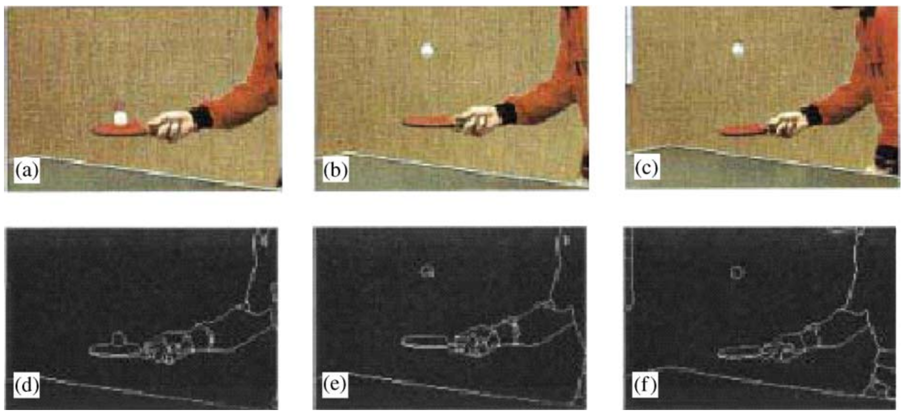


Fig. 5. Edge detection results of the frames of the table tennis sequence: (a) frame 024; (b) frame 030; (c) frame 036; (d) frame 024 after edge detection; (e) frame 030 after edge detection; (f) frame 036 after edge detection.

During the R-table construction, one other constraint being used is that the pair of edge points belonging to the same line descriptor must be some minimum distance apart in the image to form a group. This is because edge points that are close together produce unstable results in the pose estimation process, and thus less likely to be useful.

3.2.2. *Object segmentation in the subsequent video frames*

Two parameter spaces, denoted, respectively, by Π_1 (Fig. 2(b)) and Π_2 (Fig. 2(c)), are simultaneously processed. Π_1 is a 2D parameter space which processes the line parameters; whereas Π_2 is a 5D parameter space which handle the rotation τ, the scale S, the object class, and the position of the RP, $O(x_R, y_R)$. If within the video frame, a pair of edge points $A(x_A, y_A, \phi_A, L_A, a_A, b_A)$ and $B(x_B, y_B, \phi_B, L_B, a_B, b_B)$ belonging to a line support (ρ, θ) is found (process 1), then the algorithm checks if in the R-table exists a pair of edge points $P(x_P, y_P, \phi_P)$ and $Q(x_Q, y_Q, \phi_Q)$, which verify the following equations:

$$\alpha_P = \phi_A - \theta, \quad \alpha_Q = \phi_B - \theta, \alpha v \delta$$
$$\Delta s = w_L(\Delta L) + w_u(\Delta a) + w_v(\Delta b) < th, \tag{6}$$

where the terms ΔL, Δa, and Δb are the color differences for L, a, and b components, respectively, between pair of points (A, P) and (B, Q). In this paper, the color of the pixel corresponding to the two edge point are said to belong to the same object if the color distance, Δs, between them is smaller than a given threshold, th. The weights w_L, w_u, and w_v are defined by the user, however, experimental results show that generally better results are generated by setting higher weights on chrominance channels (e.g. two times higher than that of the luminance channel).

In case if the above equations are verified, then, the candidate reference point, $O(x,y)$, scale S, and rotation τ which correspond to the pair (A, B) can be calculated using the following relationships:

$$\overrightarrow{OA} = k \times \overrightarrow{AB}, \quad S = PQ/AB, \quad \text{and} \quad \tau = \theta_0 - \theta. \tag{7}$$

The parameter cell of Π_2, located at the address (x, y, S, τ) is then incremented in case it already exists. Otherwise, it is added as a new candidate. A naïve implementation of the Π_2 accumulator array uses a full 5D space, making its implementation impractical. Our approach to this problem is to use a fixed (small) hash table store to accumulate votes. It accepts one element at a time and the garbage collection of flushing happens when its finite length is exhausted. This operation is valid in our method because during the R-table construction, a maximum gap of few scan lines exists between the boundary points of the R-table.

When the whole video frame is scanned, the cell with maximum vote on Π_2 space is found and the corresponding scale, rotation, and coordinates of the object deduced.

Besides its various advantages cited above, Section 4 will show how an efficient parallel implementation of the algorithm could improve the performance of this algorithm. Compared to region merging-based algorithms [13,14], and provided with the same video clips (Figs. 3 and 10), our algorithm is better and good for parallelism and does not depend on the luminosity variation, even thought the other methods they use more weights to chrominance component of the color.

3.3. *Extendibility of the algorithm for other multimedia applications: very low bit rate coding*

Besides object segmentation, the above algorithm can be applied for very low bit rate encoding by performing multi-objects tracking in video image sequences. Hence, after all parameters (e.g. translation, zoom, and rotation) of different objects in the actual frame are known, new foreground regions are projected back to the image space using these parameters. These foreground regions are then subtracted from the objects of the actual frame to form an error component, ξ. To this last element, ξ is applied a Discrete Cosine Transform (DCT) and the resulting DCT coefficients, along with the parameters of different objects of the actual frame, are sent to the decoder. In case the error for a given object is greater than a predefined threshold (this may occur principally in case of 3D motions and rigid objects), then a new segmentation of that object is required. As will be shown in the next section, one positive contribution of our approach, compared to the classical motion estimation algorithms is its capability to extract zoom and rotation motions of more than one object in a cluttered background. In addition, the extensive block-based calculations existing for classical motions estimation algorithms are removed.

3.4. *Video object detection, tracking and indexing evaluation*

The proposed algorithm has shown very good tracking, and indexing results on general video sources. Our first experiments were applied on the Tennis scene (Fig. 2(a)). Four objects of the initial frame (tennis, racket, ball, left hand, and shoulder) were initially segmented by the user. The boundaries of these target objects were then isolated and an edge detection algorithm was applied to each of these regions, according to the rules addressed in the previous section. Both the edge magnitude and direction were simultaneously computed for each edge pixel. A reference point, RP, was then selected for each of these regions. From this point, a set of straight lines is manually traced (see Fig. 2(a) for the case of shoulder). The pair of points that intercepts simultaneously the object boundary and these lines are then stored in the R-table. Using this

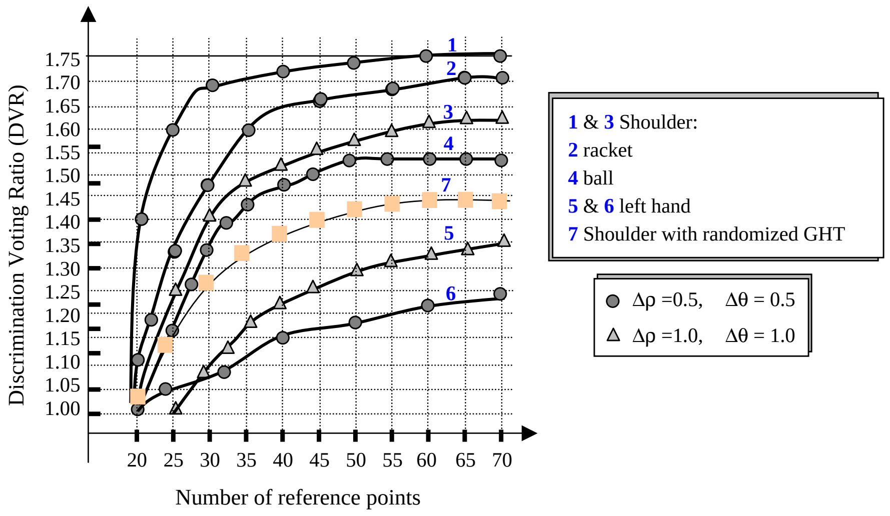


Fig. 6. Accuracy performance for different objects of Fig. 2(a).

R-table, the object segmentation and tracking algorithm then proceeds for the subsequent frames. Fig. 6 illustrates the performance of the algorithm on detecting accurately the exact parameters of the object in the second frame of the same scene, versus number of reference points used in the object's boundary. Hence, we define a new variable, namely the Discriminating Voting Ratio (DVR), as:

$$DVR = \textit{the highest vote}/\textit{the next highest vote}, \tag{8}$$

where the highest vote corresponds to the maximal value carried out from the accumulator array of the Π_2 parameter space. Different values of $\Delta\rho$ and $\Delta\theta$, representing the sampling factors of the slope and angle to the origin of straight lines during object tracking, were also experienced. Fig. 6 highlights how the algorithm can easily perform maximal peak extraction, function of the number of reference points in the object boundary.

Hence, the best accuracy was obtained with the shoulder because the distance between pair of pixels belonging to the same parameter line was higher than in the case of the three other objects. It can be also observed that the DVR ratio increases sharply to 1.73 when 40 reference points in the R-table are used. From 50 reference points, DVR tends to an asymptotic value of 1.55. On the other hand, the worst performance has been observed for the left hand because of the disappearance of the edges between the last two fingers, due to a 3D motion, and hence several reference points could not be detected. Overall, the accuracy of the line extraction (e.g. small quantization values of $\Delta\rho$ and $\Delta\theta$) and the number of reference had clearly a positive effect on object tracking. However, in all cases, our algorithm outperformed the randomized GHT algorithm [18].

Also, with the same video sequence, it has been observed that after 35 frames, only the ball and the racket could still be tracked with good accuracy. The most variable parameters were the zoom, translation, and rotation parameters, respectively. However, the shoulder and left hand could not be tracked after the frames 25th and 35th, respectively, because of a brusque 3D motion of these two objects.

On the other hand, four video sequences (bear, Akiyo, plane, and hand) with different types of motion and background are used to subjective and objective evaluation of our system (Fig. 7). After the first user input at the starting frame, the object tracking algorithm starts to process subsequent frames. As can be seen in Fig. 7, subjective object tracking of these three testing sequences gave us acceptable results.

In the objective evaluation, we manually extracted semantic objects in each frame over 40 successive frames and considered these as the ground truth. We then computed the average numbers of missing pixels, false pixels, and maximum boundary deviation between the ground truth and the segmentation results. The number of missing and false pixels is simply computed by comparing a segmented object mask with its related ground truth. While there are different ways to define the boundary deviation for a missing pixel, we define the deviation for a missing pixel as the distance between this pixel and its nearest foreground pixel in segmented object mask; and for a false pixel as the distance between this pixel and its nearest foreground pixel in the ground truth mask (Fig. 8). The maximum boundary derivation is the maximum value of the deviations of all missing or false pixels.

The performance results are shown with different number of points for clustering in Fig. 9. Hence, in the experiments, when the error results are above a predefined threshold, then a new segmentation is performed on the image. Main tracking errors are usually caused by new or uncovered background or

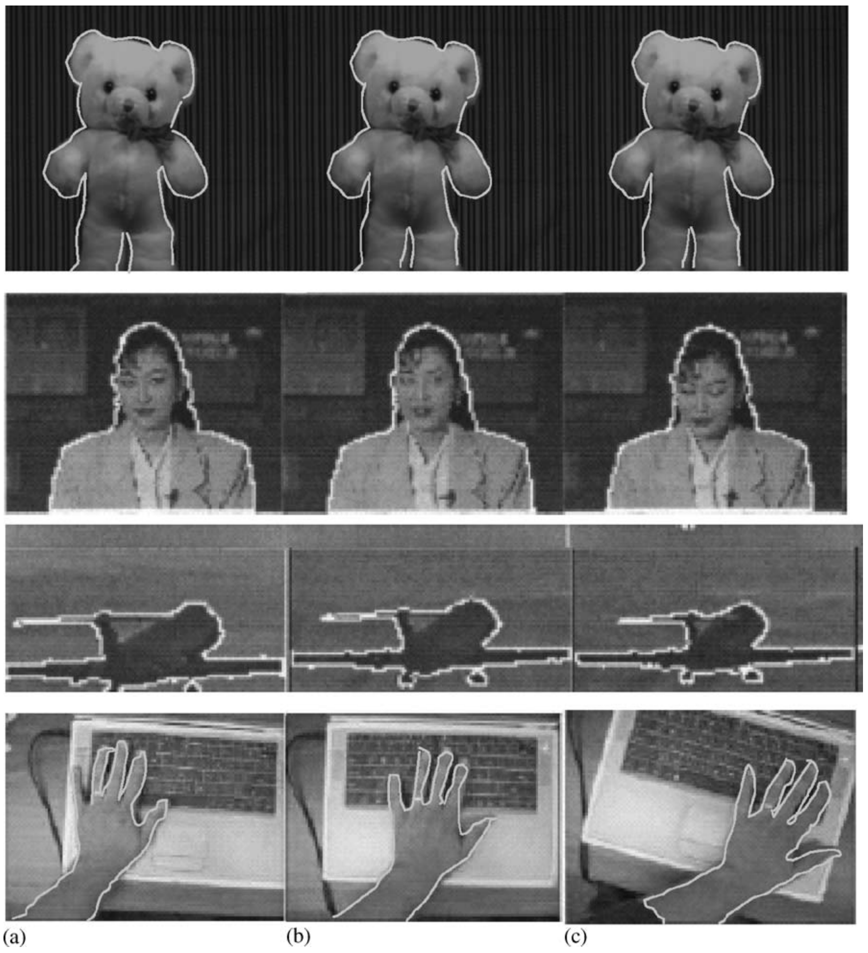


Fig. 7. Object tracking results of three sequences after one user input: frame (a) 1, (b) 3, (c) 7.

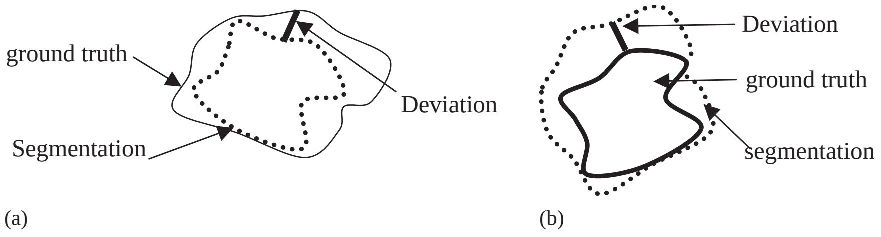
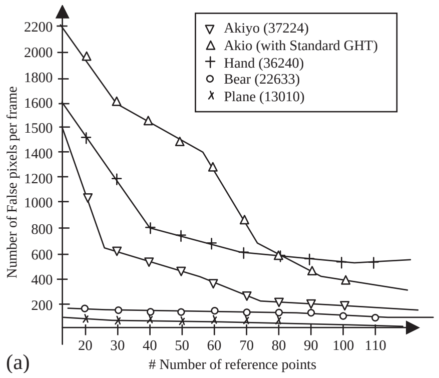


Fig. 8. The boundary deviations: (a) Missing pixels; (b) false pixels.

foreground objects or occlusion of the foreground object. The remaining errors may be considered as an accuracy limitation of our detection and tracking algorithm. Considering inherent errors caused by boundary blur (especially for the MPEG sequences with fast motion) and manual segmentation, our system generates very good tracking results. The worst results were obtained with the "hand" sequence since some fingers were rigid (3D motion).

The algorithm has also been applied for video indexing on image database (Fig. 10). To each image class, five different rotations (20°, 30°, 80°, 130°, and 170°) and five scales (1.5, 1.2, 0.9, 0.7, 0.5) were applied, which creates a database of 500 objects. The R-table was obtained only from the 15 image classes in Fig. 10(a). The performance is measured in terms of the average retrieval rate that is defined as the average percentage number of patterns belonging to the same image (Fig. 10(b)). A high rate of 0.92 was obtained using only 40 reference points.

Note that in our algorithm, the object is detected as having a translation and rotation (S and τ, respectively) from its initial position only if the number of votes in the (S, τ) cell exceeds a predefined threshold, *th*,



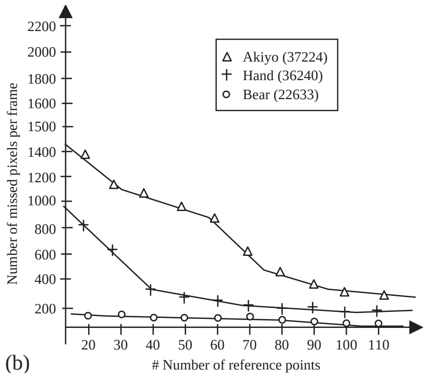


Fig. 9. Objective evaluation over 100 frames (numbers in the legends are the average region sizes). (a) Average number of false pixels. (b) Average number of missed pixels.

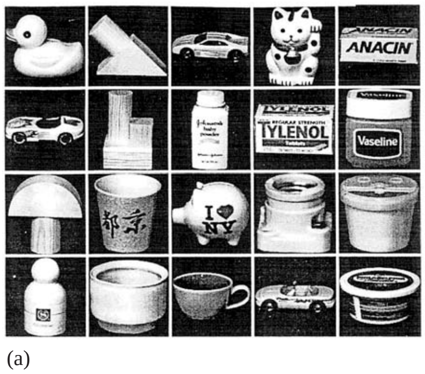
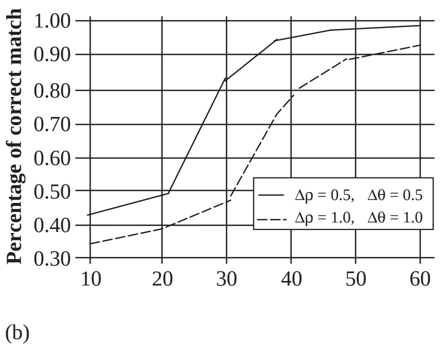



Fig. 10. Video indexing performance: (a) Image database. (b) Accuracy performance.

proportional to the number of the reference points for that object in the R-table*: i.e.*: *th* = *Number of reference points*/2. For all the experiments done in Figs. 3, 7, and 10, the number of votes where greater than the above threshold.

4. Complexity and the need for dedicated hardware design

Along with the algorithm's accuracy, real-time performance is mandatory for multimedia-based applications such as MPEG4 coding. This section highlights the analysis of the computational complexity of our object segmentation and tracking algorithm, which is measured by dynamic run-time count. Basically, the analysis is performed on the basis of realistic program execution on a Sparc 20 Workstation. Table 1 lists the average runtime simulation for object tracking (bear, Akiyo, hand, and plane). The voting in the $\Pi 2$ space is constant and negligible since for each scene, only one object model is considered in the database. The total time includes false peaks elimination and maximal extraction in the $\Pi 2$ space. The results reveal the heavy demand of computation power for this no-optimized implementation of the algorithms. The performance depends heavily on the number of edge points in the image and is much less than one frame per seconds, which is very far from real-time performance. However, by means of dedicated hardware architecture, the computation of can be efficiently parallelized and pipelined.

Table 1
Object tracking performance (in ms)

Image	Voting in $\Pi 1$ space (ms)	Voting in $\Pi 2$ space (ms)	Total (ms)
Akiyo	676	0.013	690.42
Hand	523	0.013	543.56
Bear	385	0.013	392.39
Plane	175	0.013	181.23

4.1. Previous works on parallel video processing machines

In the recent years, several parallel hardware architectures for video processing have been developed. Application Specific Integrated Circuit (ASIC)-based implementations have been extensively used [1,2] because of their high performance in terms of computation power, small die area, and low power consumption. However, they are not flexible enough to be adapted to even fine modifications of the algorithms. A more attractive solution is to have a flexible, compact, and robust hardware platform capable to sustain real-time performance and software enhancement. SIMD-based architectures still constitute the technology of choice for high complexity video processing. The Symphony [5] and CNAPS [20] are some examples. The fine-grained IMAP VLSI [21] video chip is among the most recent ones. It contains hundreds of Processing Elements, PEs. Each PE contains an Arithmetic Logic Unit (ALU) and a local memory. In these architectures, each video section is conducted by a group of PEs in a spatial parallel mode, e.g. each frame in the video sequence is spatially divided into pieces that are processed in parallel by the PEs in the group.

In [3] a High performance Multimedia coarse-grained SIMD-based Digital Signal Processor (DSP) has been designed and built. It contains sixteen data paths, centrally controlled by a RISC control unit, executing three instructions at a time. Data exchange between data paths is provided by a shared memory, while external data can be transferred concurrently via an SDRAM interface at bit rates of more than 6 Gigabits/s. The processors operate at a speed of 100 MHz and achieve sustainable performance of 2 GOPS.

Another recent fixed point Very Long Instruction Word (VLIW)-based DSP processor, the TMSc64x, from Texas Instruments has been designed for video processing applications [4]. It can operate at a clock of up to 600 MHz to perform nearly five billion instructions per second. Hence, its eight functional units, including two multipliers and six arithmetic units, are highly orthogonal, providing the compiler and Assembly Optimizer with many execution resources. Within each cycle, eight 32-bit RISC-like instructions are fetched. These latest can then be executed either in parallel, serial, or in parallel/serial combinations. This optimized scheme enables significant reductions in code size, number of program fetches and power consumption.

A common feature of these VLSI chips is that they have been shown to be particularly suitable for low level and window-based video processing. However, the performance starts to decrease in case of irregular data structures such as video-based segmentation. This is because sequential processing and inter-PE communications becomes dominant, as the number of PEs is much lower than the image size.

More PEs, and therefore higher degree of parallelism, can be provided by another type of SIMD-like VLSI architecture, namely Associative Memory, because of the simplicity of the structure of its individual PEs. VLSI architectures such as [22] have been successfully applied for low-level, window-based, video processing algorithms. In [6], an intermediate-level video processing application, which consists to extract the line from video sequence has been achieved. However, no 2D or 3D Associative Memory-based video segmentation has been yet reported. One reason is the need for an appropriate hardware–software design methodology, simultaneously at the chip and system levels.

4.2. *Our concept: content addressable memory*

Our algorithm was coded to run on memory process-based technology, CAM. A common feature of CAM chips is that each PE has only two types of instructions set, namely parallel-maskable write and parallel-maskable search. Thus, the overall complexity in terms of number of interconnects and transistors can be dramatically reduced, providing several thousands of PEs per chip. With an appropriate combination of these two types of instructions, various arithmetic operations can be performed in parallel by all PEs. Thus, the successfulness of the operation is satisfied when the total number of search operations becomes dominant and much larger than single arithmetic operations. In the case of our video processing applications, the CAM LSI circuit is used in two different ways. Either all PEs receive the pixels of input video frames in order to process them, or a set of functionality is implemented in the PEs, which acts as parameter space.

4.3. *Video processing in the parameter space*

Generally, the CAM-based hardware is appropriate for the algorithms, which perform most of the calculations not in arithmetic but in parallel search mode. One way to achieve this is to leave all the calculations at specific long-intervals period of the video frame. This idea can be efficiently used for our algorithm addressed in this paper. Its parallel implementation requires that each PE, $PE(u,v)$, is composed of three different fields, namely the decision field, $DF(u,v)$ which contains the main operation to be performed by each PE, the data field, $DT(u,v)$, which stores the data to be processed and read, and the flag field, $F(u,v)$ which stores the intermediary results (Fig. 11). Fig. 12 shows the general structure of the parallel algorithm. The algorithm has been designed so that the I/O transfer does not constitute a bottleneck since the PEs are loaded only once during the initialization and then automatically initialized at the end of the frame. The content of each field depends on the application. Thus, the algorithm reads the actual video stream and processes the data of each pixel, (k,y), which satisfies a predetermined property (line 6). For such pixel, a function $f(k,y)$ is calculated. The Single Hit Flag Register, SHFR, is then set for all PEs, whose decision field, DF, is equal to $f(k,y)$. At the end of the scan line, k, the Flag field,

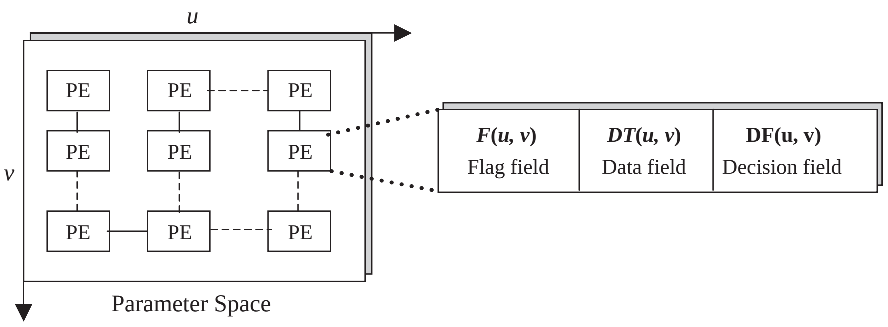


Fig. 11. General structure of the PEs during parameter space processing.

```
1   for each video frame do{
2      Initialize the parameter space;
3      Reset all the single hit flag registers;
4      for each scan line k, of a video frame do{
5         for each pixel of the image do{
6            if (property (pixel (k, y)) is true){
7               - Calculate the function f(k ,y) = 0
8               - Parallel search of the function f in the whole parameter space;
9               - The SHFR of all PEs, which satisfy the above condition are set;
10           }// end if;
11        }// end for
12        - for each PE(u, v), whose SHFR is set, the Flag field, F(u, v) is set;
13        - for ach PE(u, v), whose F(u, v) is set, the function g(u, v) is calculated
14           in the accumulator field;
15        - Each PE(u, v) updates its decision field: h(u, v, k+1) = 0;
16        - Reset all the SHFR of all PEs;
17     }//end for
18     Carry out the solution, (e.g. the maximal peak) from the CAM
19     Each PE updates the parameter space, which corresponds to the scan line 0;
20  }//end for
```

Fig. 12. General structure of the algorithm for parameter space processing.

$F(u,v)$, is set for every $PE(u,v)$, whose SHFR is set (line 12). Their accumulator fields are then modified according to the equation $g(u,v)=0$. Next, before handling the next scan line, $k+1$, of the actual video frame, the decision field, DF, of all elements is calculated using the recursive function: $h(u,v,k+1)=0$ (line 15). At the end, when the whole video frame is scanned, the decision field of all PEs is updated according to the function, $h(u,v,0)$ of the scan line 0. The output result, which corresponds to the maximal accumulated value in the DF register, is stored in the output video frame.

The timing diagram of task modules can be illustrated in Fig. 13. A controlling unit operates in parallel with the CAM array in order to calculate the function $f(k,y)$ and to read or write pixels from the video frames. Two video frames are required: One for actual processing (video frame 1) and the other for grabbing the actual video frame of the camera (video frame 2). In this way, the underlined hardware can process a continuous stream of video frames. We can observe that the long serial arithmetic operations (i.e. $F(u,v)$, $g(u,v)$, and $h(u,v,k+1)$) are calculated either at the end of the scan line, or at the end of the video frame, but not for each pixel. This leads the algorithm to be executed in a constant time, which primarily depends on the size of the video frame.

For the example of the 2D video-based segmentation algorithm, its parallel implementation requires that the $\square_1$ parameter space is partitioned so that each PE of the CAM corresponds to one cell, located at the address ($\square_{..}$ $\square$), and split into the three fields as follows: (i) The decision field, $DF=x_k$, (ii) the data field, DT, which contains the angle field of the two reference points (α_M,α_N) and the y-coordinates field of the edge pixel, which has contributed to the actual PE. In addition, the DT register includes a temporary field, used to store the y-coordinates, k, and the gradient angle, ϕ, of the edge pixel (x,k) when the line k is scanned, and (iii) the flag field, f_0 and A, to indicate how many reference points are already stored in one CAM cell.

4.4. Designing and implementing the hardware architecture: HiPIC concept

A primary goal of our HiPIC concept [6] is to provide simultaneously high performance and good flexibility to allow fast time to market. In terms of performance, the underline hardware should allow execution of each instruction of the above algorithms in one single clock cycle. According to the algorithms requirements discussed in the above section, a new 0.25 μm CAM LSI has been designed to run at 40 MHz clock speed. More details of this chip can be found in [23]. Fig. 14 shows the overall hardware components of our dedicated HiPIC architecture. It is compact and does not require any high bandwidth memory. The DSP processor [24] is directly responsible for CAM sequence and serial data processing and can provide a large internal memory that holds the video frames (1 and 2) and instruction sequences. The advantage of using DSP processor rather than FPGA for this task is that the complexity of the state machine generating the appropriate sequence of instructions heavily depends on the application. Our goal was to generate one instruction per clock cycle, independently of the complexity of the target application. This is difficult to achieve with actual FPGA devices. In addition, implementing the state machine on the DSP rather than FPGA avoids to the designer using hardware languages, such as Verilog

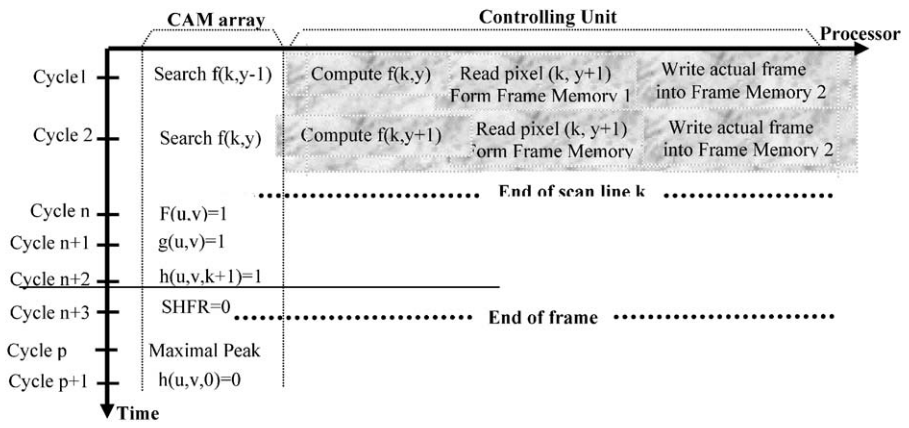


Fig. 13. Timing diagram of task modules of the parallel video-processing module.

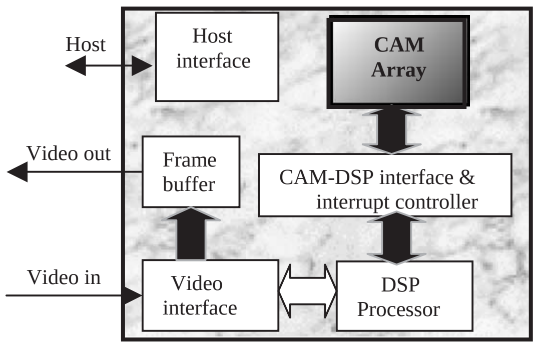


Fig. 14. Overall architecture of the CAM-based system.

HDL, which usually generate unpredictable results of hardware synthesis in case of complex applications. Another motivation of using the DSP is because it features a high-speed serial links, suitable for video frame transfer.

4.5. Algorithm's performance using HiPIC board

Both synthetic and real world video sequences were acquired by a CCD camera, which is fed to our system. Table 2 indicates both the execution time and amount of hardware when using our HiPIC board. The performance is dependant on the image size, $N \times N$, the quantization steps ($\Delta\rho$ and $\Delta\theta$) during the line extraction phase, and on the number of reference elements, q, stored in the R-table. Here, $\Delta\rho = 0.5$, $\Delta\theta = 0.5^\circ$ and $q = 40$ have been used. We can deduce from the table that for an image 256×256, and using an R-table composed of 46 different patterns, a set of 4 CAM LSIs are enough to sustain the whole algorithm under the frame rate interval.

4.6. Comparison with other previous works

As previously mentioned, several attempts to design highly parallel hardware architectures, based on either Advanced DSP processors [3,4], or dedicated SIMD processors [5,21] have been done to implement the HT.

Thus, in all these architectures, the input video frame is subdivided into several blocks that are assigned, separately, to one PE. One common limitation of these architectures concerns the I/O communication overhead: In case of spatio-parallelism, each PE needs to process a block of pixels. Therefore, if the size of each block is large, then the communication overhead becomes prevalent and the sequential processing is dominant. In addition, each PE needs to read different frames, which result in congestion since pixel processing and storing tasks cannot be simultaneously done by the PEs. In [3], a high speed SDRAM interface was used to overcome this limitation. However, this engenders a sharp increase of power consumption. The I/O contention is then found to be a serious drawback for these systems. In addition, a dedicated real-time kernel is required in order to ease the programmer's task by automatically handling the reconfiguration network between the PEs. The unsuccessfulness of many DSPs and other RISC processors, such as the Intel's i860 Processor, has been witnessed because of the lack of such software design tools. Table 3 illustrates the performance of these architectures to implement line extraction-based algorithm. In the experiment, the VLSI chips run with their highest clock speed, which is 100, 40, 600, and 40 MHz for Multimedia DSP [3], IMAP [5], TMSc64x [4], and our CAM LSI chip [23], respectively. Also, the hardware implementation in our CAM LSI has been arranged so that one single PE can handle 4 cells of the parameter space [6]. We can observe that the IMAP device achieves the lowest performance because its PEs do not contain multipliers required by the HT algorithm. In addition, the PEs cannot simultaneously store and process the pixels. Therefore, an additional time of video storage is added. The DSP chips however have an enhanced DMA controller, which allows simultaneously transfer, storage, and processing. Among the two advanced DSP, the TMSc64x performs

Table 2
Computation time performance

Image size	64 × 64		128 × 128		256 × 256	
Number of Hough points	4 K		16 K		64 K	
$\Delta\rho$	0.5		0.5		0.5	
$\Delta\theta$	0.5		0.5		0.5	
Edge detection	Frame rate		Frame rate		Frame rate	
	△	○	△	○	△	○
R-table voting ($q = 40$)	0.13	40	0.26	44	0.51	48
Line parameters	3.45	57	5.28	59	6.35	61
Max number of classes	204		95		46	
Total	3.58	97	5.54	103	6.86	109

△ Execution time (ms), ○ Number of bits/PE.

Table 3
Performance of the HT algorithm on different hardware architectures (ms)

Image size	Number of Hough points ($\rho \times \theta$)	Number of edge points/line (%)	Multi Media DSP [3] (100 MHz)	IMAP [5] (40 MHz)	C64x [4] (600 MHz)	Our solution (40 MHz)
64 × 64	64 × 64	30	4	12(1.6)[a]	1.33(0)[a]	1.6
		50	6	15(1.6)[a]	1.80(15)[a]	1.64
256 × 256	256 × 256	30	17	48(6.5)[a]	5.7(6.5)[a]	6
		50	23	59.7(6.5)[a]	7.6(6.5)[a]	6.5
512 × 512	512 × 512	30	67	195(11.2)[a]	23(3.2)[a]	25
		50	80	235(11.2)[a]	28(3.2)[a]	26.5
Hardware complexity			1–100-MHz SDRAM: 2 × $N \times N$ bytes 2-FPGA for edge detection & video timing	1–10 MHz VRAM 2-FPGA for edge detection & video timing	1–1600-MHz SDRAM: 2 × $N \times N$ bytes 2-FPGA for edge detection & video timing	1–10 MHz VRAM. 2-FPGA for edge detection & video timing
Total number of transistors (Millions)			7.4	5.2	9.2	15.5

[a] A(B): A is the processing time, whereas B is the required time to distribute the video frame on the PEs.

slightly better than our CAM chip in case of small amount of black pixels in the edged video frame. However, in case of higher number of edge points, the performance of the DSP chips decreases, whereas our algorithm is achieved in almost constant time. The extraction of the end-points requires even a higher processing time for the DSP and IMAP because of the additional sequential structure of the hardware algorithm, especially when the two constraints (quantization and proximity) are added. Thus, our algorithm, which uses end-point extraction algorithm as main preprocessing task, is better achieved with our solution. In the future, better results (almost double the performance) for line extraction can be expected if the same technology scale of the TMSc64x LSI (0.12 μm) will be applied to our CAM chip VLSI (which is actually 0.25 μm). In terms of hardware complexity, unlike the Multimedia DSP and the TI's c64, our hardware does not require two $N \times N$ high-speed SDRAM memory banks (for storing the edge image and parameter space), but only one $N \times N$ VRAM memory bank of 10 Mbits/s bandwidth (for edge image). In terms of hardware complexity, our CAM LSI has the largest number of transistors (15.5 versus 9.2 for the TI's C64). However, its (*Performance*/*Hardware complexity*) factor is the highest for the segment line extraction. The difference becomes even higher for the object tracking and stereovision algorithms because of the highly fine parallel granularity of their data structure.

5. Conclusion

In this paper, we presented an integrated scheme for real-time object segmentation and tracking. Experimental results have shown promising results and great potential for developing advanced video search tools for semantic video representation such as MPEG4. A parallel hardware architecture using CAM as a core processor has been demonstrated robust enough to sustain real-time performance of both the algorithms. For the sake of high efficiency, a hardware/software design methodology at the chip and system levels has been followed.

As first a new HT algorithm for object segmentation of unknown shape was proposed. Its key point is to use a pair of edge features which are more likely to correspond to a meaningful structure in the video frame than low level features such as single points. More than two points can be used in order to reduce the occlusion problem. Experimental results indicate that the execution time is faster, while achieving good segmentation rate. Furthermore, the method can be used to detect any kind of object and is not limited only to polyhedra objects like in [9]. Other variations and adaptations of this geometric approach should emerge to tackle other machine vision tasks, such as texture classification by using some features more appropriate than edge direction and edge magnitude, such as texture estimators [8]. Experimental results on our HiPIC-based hardware indicate that 4 CAM LSI are enough to sustain this application in real-time.

References

[1] Pereira F. MPEG-4: Why, what, how and when. Journal of Signal Processing: Image Communication 2000;15:271–9.

[2] Brady N. MPEG-4 Standardized methods for the compression of arbitrarily shaped video objects. IEEE Transactions on Circuits and Systems for Video Technology 1999;9:1170–89.

[3] Laboratory for Microelectronics and Systems, University of Hannover, “The HiPar Multi-Media DSP Processor”, Data Sheet, 2001.

[4] Texas Instruments, “A new 0.12 μm Highly Parallel DSP Processor: TMS320c64x”, Data Sheet, 2002.

[5] Fujita Y, Kyo S, Yamashita N, Okazaki S. A 10 GIPS SIMD processor for PC-based real-time vision applications: architecture, algorithm implementation and language support. Computer Architecture for Machine Perception (CAMP) 1997;22–32.

[6] Meribout M, Ogura T, Nakanishi M. On using the CAM concept for parametric curve extraction. IEEE Transactions on Image Processing 2000;9(12):2126.

[7] Chang S, Chen W, Zhing D. VideoQ: An automated content-based video search system using visual cues. In: Proceedings of ACM Fifth Multimedia Conference, Seatle, WA, November 1997. p. 313–24.

[8] Grimson WEL. The combinatorics of object segmentation in cluttered environments using constrained search. Artificial Intelligence 1990;44(1–2):121–65.

[9] Silberberg T, Davis L, Harwood DA. Object segmentation using oriented model points. Computer Vision, Graphics and Image Processing 1996;35:47–71.

[10] Wang JY, Adelson E. Representing moving images with layers. IEEE Transactions on Image Processing 1994;3:625–38.

[11] Ayer A, Sawhney HS. Layered representation of motion video using robust maximum likelihood estimation of mixture models and MDL encoding. In: Proceedings of the Fifth International Conference Computer Vision, 1995. p. 777–84.

[12] Meyer F, Boutheney P. Region based tracking using affine motion models in long image sequences. CVGIP 1994;60(2):119–40.

[13] Dubuisson MP, Jain A. Contour extraction of moving objects in complex outdoor scene. International Journal of Computer Vision 1995;14:83–105.

[14] Gu C, Lee M. Semantic video object segmentation and tracking using mathematical morphology and perspective motion model. In: Proceedings of ICIP’97, Santa Barbara, CA, October 1997. p. 514–7.

[15] Hou KM, Belloum A, Meribout M. A perception sensor for mobile robot. Real-time Imaging 1997;3:379–90.

[16] Grimsson WEL. Object segmentation by computer: the role of geometric constraints. Cambridge, MA: MIT Press; 1986.

[17] Freeman H. On the encoding of arbitrary geometric configuration. IRE Transactions on Electronic Computing 1981;EC-10(2):260–8.

[18] Olson CF. Improving the generalized hough transform through imperfect grouping. Internal report, Jet Propulsion Laboratory, California Institute of Technology, 1997.

[19] Berns RS. Principles of color technology, 3rd ed. New York: Wiley; 2000.

[20] Hammerstrom D, Strain D. CNAPS—An experiment in desktop super-computing. Internal Research Report, Department of Electrical and Computer Engineering, Oregon Graduate Institute of Science and Technology, 1999.

[21] Juvin D, Gamrat Ch, Larue, Collette JF. SYMPHONIE: Highly Parallel Calculator: Modelization and Realization. Journées Adéquation Algorithmes Architectures, Toulouse, 1996.

[22] Hariyama M, Sasaki K, Kameyama M. Collision detection VLSI processor for intelligent vehicles using a hierarchically-content-addressable memory. IEICE Transactions on Electronics E82-C 1999;9:1722–9.

[23] Ogura T, Ikenaga T. A fully parallel 1 Mb CAM LSI for real-time pixel parallel image processing. Proceedings International Solid State Circuits Conference, 1999. p. 15.5.

[24] “ADSP, Shark 21061 DSP Processor”, user guide, Analog Device, 1996.