Hindawi Publishing Corporation
Advances in Artificial Intelligence
Volume 2016, Article ID 6361237, 34 pages
http://dx.doi.org/10.1155/2016/6361237

Research Article

Automatic Representation and Segmentation of Video Sequences via a Novel Framework Based on the *n*D-EVM and Kohonen Networks

José-Yovany Luis-García and Ricardo Pérez-Aguila

Group of Multidisciplinary Research Applied to Education and Engineering (GIMAEI), The Technological University of the Mixteca (UTM), Carretera Huajuapan-Acatlima Km 2.5, 69004 Huajuapan de León, OAX, Mexico

Correspondence should be addressed to Ricardo Pérez-Aguila; ricardo.perez.aguila@gmail.com

Received 28 September 2015; Revised 19 January 2016; Accepted 20 January 2016

Academic Editor: Francesco Buccafurri

Recently in the Computer Vision field, a subject of interest, at least in almost every video application based on scene content, is video segmentation. Some of these applications are indexing, surveillance, medical imaging, event analysis, and computer-guided surgery, for naming some of them. To achieve their goals, these applications need meaningful information about a video sequence, in order to understand the events in its corresponding scene. Therefore, we need semantic information which can be obtained from objects of interest that are present in the scene. In order to recognize objects we need to compute features which aid the finding of similarities and dissimilarities, among other characteristics. For this reason, one of the most important tasks for video and image processing is segmentation. The segmentation process consists in separating data into groups that share similar features. Based on this, in this work we propose a novel framework for video representation and segmentation. The main workflow of this framework is given by the processing of an input frame sequence in order to obtain, as output, a segmented version. For video representation we use the Extreme Vertices Model in the *n*-Dimensional Space while we use the Discrete Compactness descriptor as feature and Kohonen Self-Organizing Maps for segmentation purposes.

1. Introduction

Video segmentation is an open problem in the Computer Vision and Pattern Recognition fields. In recent years, this problem is of high interest because of the applications that can be derived from it [1, 2], for example, video indexing in databases, video summary, surveillance, medical imaging, event analysis, and computer surgery. The applications based on video content usually analyze the scene of the sequence and they require semantic information about the events that are taking place in it. The events' information is obtained from the objects that are relevant to the scene.

Applications for video content analysis require that the object information is provided in a suitable manner. For example, a very common approach is to compute features for describing objects, such that it is possible to distinguish them among others. For this reason, one of the most important tasks involved in object extraction is the process of segmentation, because the information that will be computed from the objects depends largely on the quality of this task.

Some the most common video applications that include a segmentation stage are the following:

(i) *Action recognition*: the objective is to determine the actions taking place in the scene, for example, if the people are walking, running, or jumping. Some related works for this applications are [3, 4].

(ii) *Object recognition*: it aims to object detection, where the main tasks involve feature extraction and measuring similarities with respect to an ideal model. So, the performance of these systems depends largely on the segmentation stage [5, 6].

(iii) *Surveillance and traffic control*: it is performed by dividing the scene in a background and the significant

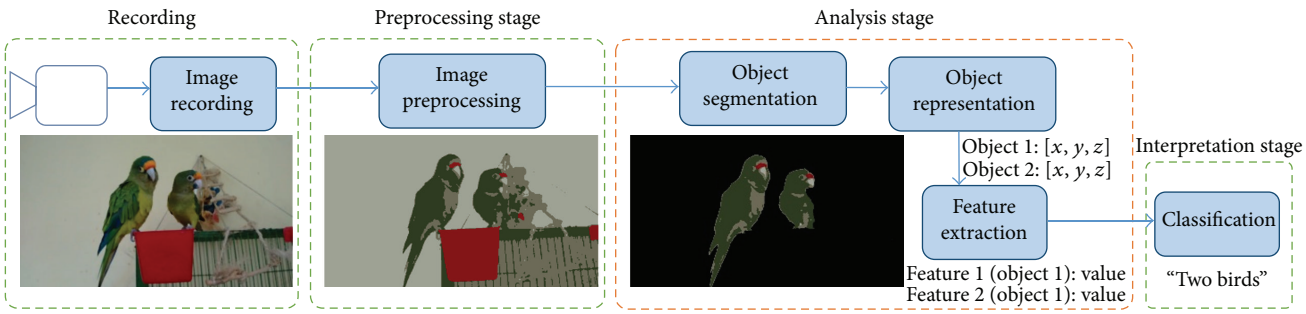


Figure 1: A general video or image processing framework.

elements. Therefore, it is executed as object tracking and by this way it is possible to detect movement [7–9].

(iv) *Video indexing*: where the approach is to describe the scene content in form of text [10, 11]. By this way, it is possible to process a search query based on the content of the video. A survey of the state of the art for these applications is presented in [12]. One approach was implemented in [13], where a video can be indexed based on its Discrete Compactness.

(v) *Background compression*: the objects of interest, or at least the most prominent ones, are coded with more information than the background [14]. For example, when having a video-call we want to see more details about the person we are talking to, rather than the background [15]. A complete reference for video compression, under the well-known coding standards H.264 and MPEG-4, can be found in [16].

(vi) *3D object reconstruction*: these applications aim to reconstruct 3D objects from 2D frames which were captured from different angles [17]. These applications are commonly applied in order to generate a 3D view of an urban area [18, 19], as well as for obtaining 3D models of human body parts or faces [20].

(vii) *Digital entertainment*: such as changing the background of a scene, where the main characters are placed in a virtual location [21].

Video and image segmentation can be defined as the process that forms groups of information that share same features [22–24]. A formal definition for image segmentation was given by Fu and Mui [22] by considering an image R that can be divided in R_i, $i = 1, 2, \ldots, m$, disjoint nonempty regions. These regions should satisfy the following conditions:

(i) The union of all the regions R_i results in the original image.

(ii) The intersection of any two regions is the empty set.

(iii) The elements in a region share similar properties while elements in different regions have different properties.

These conditions can be easily extended to video segmentation, taking into account that a 2D image can be represented as a function $f(x_1, x_2)$, such that we would have, for example, an additional third dimension. So, we can express an image that changes in time as a function $f(x_1, x_2, t)$ which represents the color changes of a pixel in position (x_1, x_2) and along the temporal dimension t.

The segmentation process is a key stage for a framework for image or video processing, such as the one shown in Figure 1. As described in [23, 25, 26], the segmentation stage is very important for the subsequent stages because the computation of descriptors for the segmented objects varies for different segmentation results, and therefore this will impact the recognition results.

1.1. A New Approach to the Problem of Video Segmentation. In the literature there have been proposed several approaches for video segmentation based on unsupervised methods. For example, in [27] the *Multiple Hypothesis Video Segmentation* (MHVS) method is defined. It generates hypothesis for grouping and labeling a set of corresponding pixels in consecutive frames. In this way, a *superpixel* is defined which is a collection of pixels in a contiguous region. An extension of this approach is given in [28], where the motion is considered for superpixels' formation. Another important work in this line of research is presented in [29]. This work proposes the use of *Must-Link Constraints* to reduce the amount of superpixels by a merging-like technique, consequently reducing the use of computing resources and improving the segmentation performance.

This work boards the problem of video segmentation with a new approach. We propose using the Extreme Vertices Model in the n-Dimensional Space (nD-EVM) for video sequences' representation. Moreover, it is considered the clustering approach using the *Kohonen Self-Organizing Maps* (SOM) as a segmentation method which in turn makes use of the *Discrete Compactness* (DC) descriptor as a feature.

The nD-EVM model has been used for efficient representation of *n-Dimensional Orthogonal Pseudo Polytopes* (nD-OPPs) [30, 31]. One of the main advantages of this model is that it allows representing an nD-OPP using only a subset of its vertices: the *extreme vertices*.

In the context of a 2D image, a frame can be represented through a higher dimensional polytope. We have a 2D base which is defined by the width and length of the image, and additional dimensions are required depending on the

considered color scheme. Some well-known color schemes are grayscale, RGB, and RGBA, to mention a few ones. Now, we take a step further: we start from the idea that the information in a video sequence can be seen as an nD-OPP since we have two geometrical dimensions for width and length of the frames, one temporal dimension, and a number of additional dimensions depending on the considered color scheme [26, 32]. In [32] the foundations of color video representation are presented in the nD-EVM model, where a color frame can be seen as a 3D-OPP and therefore it can be represented in a 3D-EVM. A drawback of this approach is because the RGB scheme is considered as only one dimension: the RGB24 scheme is a 24-bit color representation which in turn is extruded in only one dimension as an integer. This means that small variations in higher order bits produce large variations in the whole integer value. On the other hand, in [26], the same idea for representing a video sequence is taken, but instead of taking only one-third dimension for RGB frames, the color is split in three dimensions so that a color frame is represented through a 5D-EVM: a 2D base and three dimensions for the RGB scheme, one for each color component. Therefore, in order to represent a video sequence this can be done through a 6D-EVM. This approach is more reliable in the sense that we capture more information about the color of a single frame pixel, and its corresponding extrusion has geometrical and topological characteristics that allows us to distinguish it among others in a more efficient manner. Furthermore, using shape descriptors as features, such as DC, we can perform useful classification tasks.

The DC descriptor is a shape descriptor that can be applied to an nD-OPP expressed in the nD-EVM model. In this work we use DC to describe a small region inside the 6D-EVM of an animation. This region is a sixth dimensional *mask*. Generally, a mask (sometimes called *template*) is used in image processing for querying or modifying the information in a certain area [33, 34]. In our context, for video sequences represented through the EVM model, a mask is only applied for obtaining a subanimation. The lengths of the mask in every axis of the 6D space are arbitrarily established. For example, we can take a small region in the 2D space, let us say a 5×5 square. But we can also establish the time length of the mask, 5 frames, for example. This can be seen as retrieving the changes in a specific time range of a certain region of the video sequence.

With the above reasoning, we can obtain all the possible subanimations from a video sequence and compute the DC descriptor to characterize them. This process is similar to a convolution in the image processing field. These DC values form a training set, which will be used for the training process of the SOM. Once the SOM has been trained, we segment the video sequence into m classes, where m is the number of neurons or clusters that we want to obtain. Considering the clustering process only labels the cluster's content with the cluster number, the subanimations in a cluster only have information of position in the 2D space and position in the temporal axis. Therefore, for a cluster, we generate a 3D-EVM which only contains the union of all the subanimations associated with it.

2. Self-Organizing Maps Foundations

As abovementioned, the segmentation process consists in a clustering process. *Clustering* is an unsupervised learning approach for categorizing a set of patterns. In this case, the labels (classes) of the patterns are not available, so the objective is to reveal the organization of them into clusters of similar patterns [35]. The clusters provide a structure to determine similarities and differences among the patterns. By this way it is possible to derive conclusions from such organization.

As usual, the patterns are represented in terms of features. Therefore, there are some basic steps to follow in order to properly perform dataset clustering [35]:

(i) *Feature selection*: features play an important role because they must be properly selected for providing as much information as possible with respect to the pattern.

(ii) *Proximity measure*: they quantify the similarity or dissimilarity between two patterns.

(iii) *Clustering criterion*: it is a function of the proximity measure and it must be sensitive to the nature of the patterns. An expert on the field provides clues on the type of clusters expected.

(iv) *Clustering algorithms*: they are the algorithmic approaches for the clustering problem.

(v) *Validation of results*: This step allows validating the correctness of the clusters.

(vi) *Interpretation*: in this step conclusions are derived from the clustering results. Generally this step is performed by the expert in the application field.

In this work the training dataset is formed by DC descriptors as features and the proximity measure is the Euclidean distance between two patterns. The *Self-Organizing Map* (SOM) is the selected algorithmic approach, specifically the *Kohonen* model where the clustering criterion is based on competitive learning [36–38]. The neurons in the SOM compete to be activated and updated for every input pattern; therefore the neurons are tuned to different regions in the feature space. This way a meaningful structure representation is formed, where the spatial locations of the neurons represent intrinsic features of the input patterns, hence the name of map.

There are three main stages for a SOM to be considered: competition, cooperation, and synaptic adaptation. Considering a generic case, these stages are described in the following subsections.

2.1. Competition Stage. Let us consider the case of a feature nD Euclidean space; then feature vectors are denoted as

$$X = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}^T. \tag{1}$$

On the other hand, let us consider a total of m neurons; in consequence the synaptic weight vectors are denoted as follows:

$$W_j = \begin{bmatrix} w_{j,1} & w_{j,2} & \cdots & w_{j,n} \end{bmatrix}^T \quad j = 1, 2, \ldots, m. \tag{2}$$

For a given input pattern X, in order to find the *winning neuron* the proximity measure between the feature vector and the synaptic weight of each neuron is computed. As mentioned before, the Euclidean distance is our proximity measure. Since the Euclidean distance is also a dissimilarity measure, in order to find the winning neuron we want to minimize the distance between X and W_j, $j = 1, 2, \ldots, m$:

$$i(X) = \arg\min_j \left\| X - W_j \right\|. \tag{3}$$

The winning neuron is the one that minimizes the Euclidean distance. This means that the weight updates will be centered on that neuron.

2.2. Cooperation Stage. Given a winning neuron, in this stage a neighborhood is defined having this neuron as its center. In simpler terms, nearby neurons process similar patterns and distant neurons process totally different patterns. This scheme was taken from the neurobiological nature of the human brain to process sensory information, this in the basis of many research works [39, 40].

A neighborhood relationship is stronger in the closest neurons, and it decreases as the synaptic weight distance increases. For a given input pattern, when a weight update is performed in the winning neuron there are also weight updates in the neighbor neurons. But the weights of neighbor neurons are updated in function of the neighborhood distance. This means the weights of closest neurons are more affected than the farthest ones. This behavior is achieved with a *Gaussian* function, and it has its maximum for a zero radius, which corresponds to the coordinates of the winning neuron, and it decreases as the distance of neighbor neurons increases.

Since the SOM's neurons are becoming more specialized on each dataset presentation, it is desired that the width of the neighborhood reduces when the number of presentations increases. Therefore, at the final presentations we will have a radius close to zero. The neighborhood function is then denoted as follows:

$$h\left(j, i(X), d_{j,i}, t\right) = \exp\left(-\frac{d_{j,i}^2}{2\sigma(t)^2}\right), \tag{4}$$

where $i(X)$ is the winning neuron for the current input pattern X, j is the neuron that we want to compute the neighborhood, $d_{j,i}$ is the Euclidean distance between neuron j and the winning neuron i, t is the current iteration of the dataset presentation, and finally $\sigma(t)$ is the effective distance that indicates the degree in which the neighbor neurons are activated.

2.3. Adaptation Stage. This stage consists of synaptic weight updates. Given an input pattern X, the weights are updated as follows:

$$W_j(t+1) = W_j(t) + \alpha \cdot h\left(j, i(X), d_{j,i}, t\right) \cdot \left(X - W_j\right), \tag{5}$$

where $W_j(t+1)$ is the new and updated weight and $W_j(t)$ is the current weight of neuron j. As can be noted, the updates are a function of the winning neuron i, as previously mentioned. Another parameter is introduced: the *learning rate* α. It is a function of the current iteration of the dataset presentation.

2.4. Summary of the Training Algorithm. With the previously described stages the basic training algorithm is defined for the formation of the SOM. Summarizing, this algorithm consists of the following steps:

(1) Initialization: all synaptic weights are initialized with a random number.

(2) Iterations: the training dataset is presented to the SOM network for weight adjustments:

  (a) for each pattern X in the dataset, there exist

    (i) winning neuron: it is identified as the winning neuron, whose weight vector minimizes the Euclidean distance to the input pattern, as stated in (3),

    (ii) weight updates: the weight updates are performed for the winning neuron and the neurons in the neighborhood. This is performed according to (5).

3. The Extreme Vertices Model in the n-Dimensional Space (nD-EVM)

The nD-EVM model is a scheme for representing n-Dimensional Orthogonal Pseudo Polytopes (nD-OPPs). The theoretical foundations of the nD-EVM were originally established in [30, 31]. Some research work related to this model considers representation and visualization of video sequences [32], modeling of 3D and 4D datasets [41, 42], and Discrete Compactness computation [13]. In order to understand the nD-EVM, we enunciate some fundamental propositions but for the sake of brevity their corresponding formal proofs are presented in the abovementioned references.

3.1. nD-OPPs Fundamentals

Definition 1 (see [43]). A singular n-Dimensional hyperbox in $\mathbb{R}^n$ is given by the continuous function:

$$I^n : [0,1]^n \longrightarrow [0,1]^n, \quad x \sim I^n(x) = x. \tag{6}$$

Definition 2 (see [43]). A general singular k-Dimensional hyperbox in the closed set $A \subset \mathbb{R}^n$ is the continuous function $c : [0,1]^k \to A$.

In the above definitions, the function I^n defines a unit hyperbox $[0,1]^n$ in the nD Euclidean space. More specifically, and as an example, in the 2D case it defines a unit square, while in the 3D space this function defines a unit cube.

Definition 3 (see [43]). For all $i = 1, 2, \ldots, n$, the two singular $(n-1)$D hyperboxes $I^n_{(i,0)}$ and $I^n_{(i,1)}$ are defined as follows: if $x \in [0,1]^{n-1}$, then

$$\begin{aligned} I^n_{(i,0)}(x) &= I^n(x_1, \ldots, x_{i-1}, 0, x_i, \ldots, x_{n-1}) \\ &= (x_1, \ldots, x_{i-1}, 0, x_i, \ldots, x_{n-1}), \\ I^n_{(i,1)}(x) &= I^n(x_1, \ldots, x_{i-1}, 1, x_i, \ldots, x_{n-1}) \\ &= (x_1, \ldots, x_{i-1}, 1, x_i, \ldots, x_{n-1}). \end{aligned} \tag{7}$$

Definition 4 (see [43]). In a general singular nD hyperbox c one defines the (i,α) cell as $c(i,\alpha) = c \circ I^n_{(i,\alpha)}$.

Definition 5 (see [43]). The orientation of a $(n-1)$D cell $c \circ I^n_{(i,\alpha)}$ is given by $(-1)^{i+\alpha}$.

Definition 6 (see [43]). A $(n-1)$D oriented cell is given by the scalar-function product $(-1)^{i+\alpha} \cdot c \circ I^n_{(i,\alpha)}$.

Definition 7 (see [43]). A formal linear combination of singular general kD hyperboxes, $1 \le k \le n$, for a closed set A is called a k-chain.

Definition 8 (see [43]). Given a singular general nD hyperbox c we define the $(n-1)$ chain, called the boundary of c, by

$$\partial(c) = \sum_{i=1}^{n} \left(\sum_{\alpha=0,1} (-1)^{i+\alpha} \cdot c \circ I^n_{(i,\alpha)} \right). \tag{8}$$

Definition 9 (see [43]). The boundary of an n-chain $\sum c_i$, where each c_i is a singular general nD hyperbox, is given by

$$\partial\left(\sum c_i\right) = \sum \partial(c_i). \tag{9}$$

The above definitions share with us the basis for handling collections of hyperboxes in the nD space. By this way, when the boundary of a collection of hyperboxes is computed, shared cells between hyperboxes are automatically deleted because of their sign's orientation.

Definition 10. An n-Dimensional Orthogonal Pseudo Polytope, or just an nD-OPP, is a n-chain composed by nD hyperboxes arranged in such way that, by selecting a vertex in any of these hyperboxes, it describes a combination of nD hyperboxes composed up to 2^n hyperboxes.

3.2. nD-EVM Fundamentals

Definition 11. Let c be a combination of hyperboxes in the nD space. An Odd Adjacency Edge of c, or just an Odd Edge, is an edge with an odd number of incident hyperboxes of c. Conversely, if an edge has an even number of incident hyperboxes of c, then it is called an Even Adjacency Edge, or just an Even Edge.

Definition 12. A Brink or Extended Edge is the maximal uninterrupted segment built out of a sequence of collinear and contiguous odd edges of an nD-OPP.

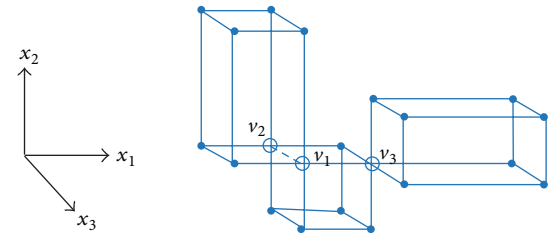


Figure 2: Example of a 3D-OPP and its extreme vertices.

Considering that a brink is formed by m odd edges, $m \ge 1$, it contains $m+1$ vertices: two of them are extreme vertices and the remaining $m-1$ are interior vertices. We will see that the interior vertices are not considered in the EVM model which is going to be defined.

Definition 13. The ending vertices of all the brinks in an nD-OPP p will be called the extreme vertices of p. EV(p) will denote the set of extreme vertices of p.

Definition 14. Let p be an nD-OPP. One defines the Extreme Vertices Model of p, denoted by $\text{EVM}_n(p)$, as the model that only stores all the extreme vertices of p.

Theorem 15. *A vertex of an nD-OPP p, $n \ge 1$, when it is locally described by a set of surrounding nD hyperboxes, is an extreme vertex if and only if it is surrounded by an odd number of such nD hyperboxes.*

Theorem 16. *Any extreme vertex of an nD-OPP, $n \ge 1$, when it is locally described by a set of surrounding nD hyperboxes, has exactly n incident linearly independent odd edges.*

Let us take the example from Figure 2. A 3D-OPP and its extreme vertices are presented. As can be observed, the continuous lines represent odd edges; on the other hand, the dotted line represents an Even Edge. Black dots are extreme vertices and white dots are internal vertices. In this case we can see that at the end of every brink an extreme vertex is found. On the other hand, vertices v_1, v_2, and v_3 are nonextreme vertices. The remaining vertices satisfy the conditions established by Theorems 15 and 16: only extreme vertices have exactly three incident odd edges. As we can see, vertices v_1, v_2, and v_3 all have at least four incident odd edges.

Definition 17. Let p be an nD-OPP. A kD couplet of p, $1 < k < n$, denoted by $\Phi(p)$, is the maximal set of kD cells of p that lies in a kD space, such that a kD cell e_0 belongs to a kD couplet if and only if e_0 belongs to an $(n-1)$D cell present in $\partial(p)$; that is,

$$\begin{aligned} (e_0 \in \Phi(p)) \Longleftrightarrow (&\exists c,\ c \text{ belongs to } \partial(p)) \\ &\cdot \left(e_0\left([0,1]^k\right) \subseteq c\left([0,1]^{n-1}\right)\right). \end{aligned} \tag{10}$$

Definition 18. Consider an nD-OPP p:

(i) Let np_i be the number of distinct coordinates present in the vertices of p along X_i-axis, $1 \le i \le n$.

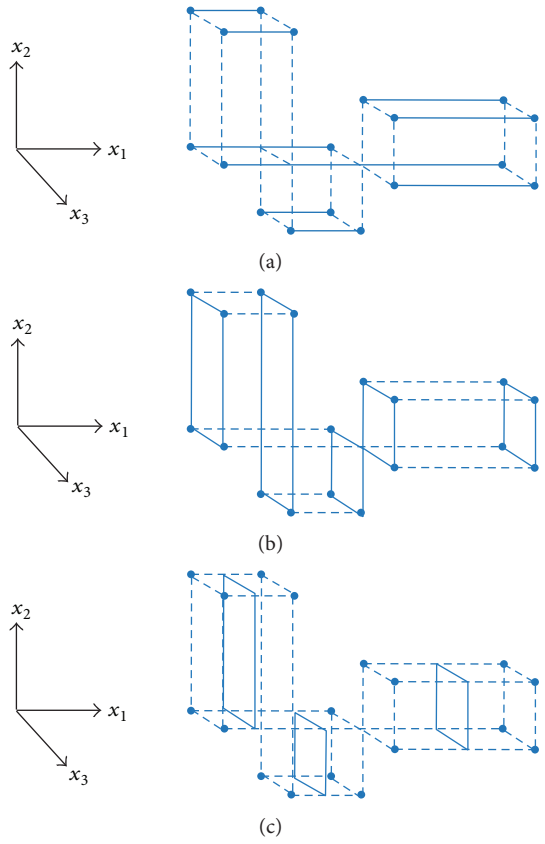


FIGURE 3: Some brinks, couplets, and sections for a 3D-OPP.

(ii) Let $\Phi_k^i(p)$ be the kth $(n-1)$D couplet of p which is perpendicular to X_i-axis, $1 \le i \le n$.

In Figure 3 we illustrate an example for brinks and couplets for the 3D-OPP previously shown in Figure 2. As we can see, the brinks can be classified with respect to the coordinated axis they are aligned to. All the brinks shown in Figure 3(a) are aligned with respect to X_1-axis. In a similar fashion, couplets can be classified according to which axis they are perpendicular to. In Figure 3(b) the couplets are shown that are perpendicular to X_1-axis.

3.2.1. Sections and Slices

Definition 19. A slice is the region contained in an nD-OPP p between two consecutive couplets of p. $\text{Slice}_k^i(p)$ will denote the kth slice of p which is bounded by $\Phi_k^i(p)$ and $\Phi_{k+1}^i(p)$, $1 \le k \le np_i$.

Definition 20. A section is the $(n-1)$D-OPP, $n > 1$, resulting from the intersection between an nD-OPP p and a $(n-1)$D hyperplane perpendicular to the coordinate axis X_i, $1 < i < n$, not coinciding with any $(n-1)$D couplet of p. A section will be called external or internal section of p if it is empty or not, respectively. $S_k^i(p)$ will refer to the kth section of p between $\Phi_k^i(p)$ and $\Phi_{k+1}^i(p)$, $1 < k < np_i$.

In Figure 3(c), sections are shown perpendicular to X_1-axis for the 3D-OPP originally presented in Figure 2. Sections can be also classified considering their perpendicularity with respect to X_2- and X_3-axis.

3.2.2. Computing Couplets and Sections

Definition 21. The Projection Operator for $(n-1)$D cells, points, and sets of points is, respectively, defined as follows:

(i) Let $c(I_{(i,\alpha)}^n(x)) = (x_1, \ldots, x_n)$ be an $(n-1)$D cell embedded in the nD space. Let $\pi_j(c(I_{(i,\alpha)}^n(x)))$ denote the projection of the cell $c(I_{(i,\alpha)}^n(x))$ onto an $(n-1)$D space embedded in nD space whose supporting hyperplane is perpendicular to X_j-axis:

$$\pi_j\left(c\left(I_{(i,\alpha)}^n(x)\right)\right) = \left(x_1, \ldots, \widehat{x}_j, \ldots, x_n\right). \tag{11}$$

(ii) Let $v = (x_1, \ldots, x_n)$ a point in $\mathbb{R}^n$. The projection of that point in the $(n-1)$D space, denoted by $\pi_j(v)$, is given by

$$\pi_j(v) = \left(x_1, \ldots, \widehat{x}_j, \ldots, x_n\right). \tag{12}$$

(iii) Let Q be a set of points in $\mathbb{R}^n$. The projection of the points in Q, denoted by $\pi_j(Q)$, is defined as the set of points in $\mathbb{R}^{n-1}$ such that

$$\pi_j(Q) = \left\{p \in \mathbb{R}^{n-1} : p = \pi_j(x),\ x \in Q \subset \mathbb{R}^n\right\}. \tag{13}$$

In all cases given by (11), (12), and (13), $\widehat{x}_j$ is the coordinate corresponding to X_j-axis to be suppressed.

Theorem 22. *The projection of the set of $(n-1)$D couplets, $\pi_i(\Phi_k^i(p))$, of an nD-OPP p, can be obtained by computing the regularized XOR ($\otimes$) between the projections of its previous $\pi_i(S_{k-1}^i(p))$ and next $\pi_i(S_k^i(p))$ sections; that is, $\pi_i(\Phi_k^i(p)) = \pi_i(S_{k-1}^i(p)) \otimes^* \pi_i(S_k^i(p))$.*

Theorem 23. *The projection of any section, $\pi_i(S_k^i(p))$, of an nD-OPP p, can be obtained by computing the regularized XOR between the projection of its previous section $\pi_i(S_{k-1}^i(p))$ and the projection of its previous couplet $\pi_i(\Phi_k^i(p))$.*

Theorem 24. *Let p and q be the two nD-OPPs having $EVM_n(p)$ and $EVM_n(q)$ as their respective EVMs in the nD space; then $EVM_n(p \otimes^* q) = EVM_n(p) \otimes EVM_n(q)$.*

In the previous theorems the foundations were given for obtaining couplets and sections sequences. This is useful for the specification of algorithms for performing modifications or operations between two nD-OPPs expressed in the nD-EVM.

3.3. Algorithms in the nD-EVM. In this subsection we present the basic algorithms for the nD-EVM model. The basic algorithms were introduced in [31] and in research works cited at the beginning of the section. The following list describes some primitive operations for the nD-EVM model:

(i) *initEVM*: initialize a nD-EVM object, which contains the empty set.

(ii) *isEmpty*: it returns *true* if a given nD-EVM p is empty; otherwise it returns *false*.

(iii) *endEVM*: it returns *true* if the end of a given EVM p along the X_1-axis has been reached. This is useful when an operation is performed or a procedure requires to explore the couplets sequence. This way we can know if we already explored all couplets in p.

(iv) *readCouplet*: it extracts the next $(n-1)$D couplet that is perpendicular to X_1-axis from a given nD-EVM p. This reading is performed sequentially and orderly, which means that every time we call this procedure it will return a different couplet (the next one), and when we reach the end of p it will return an empty object.

(v) *putCouplet*: given a nD-EVM p and a $(n-1)$D couplet c, and considering that c is perpendicular to the X_1-axis, it appends c to p.

(vi) *getCurrentCoord*: for a given nD-EVM p, it returns the x_1 coordinate of the next $(n-1)$D couplet to be extracted from p.

(vii) *setCoord*: it sets the x_1 coordinate value on every vertex of a $(n-1)$D couplet c.

(viii) *mergeXor*: it applies the exclusive XOR operation to the vertices of the given input nD-EVMs p and q and returns the resulting set.

Now, from the above primitive operations it is possible to compute sections and couplets projections. This is based on Theorems 22, 23, and 24, where the *mergeXOR* algorithm is used:

(i) *getCouplet*: it computes the projection of a couplet between two consecutive sections. This is achieved by calling *mergeXOR* algorithm.

(ii) *getSection*: this algorithm computes the projection of a section from its previous section and following couplet. This is also achieved by using *mergeXOR* algorithm.

From the previous algorithms, Algorithm 1 computes the sections sequence for an nD-OPP p expressed in the nD-EVM. As can be seen, this procedure is performed in a sequential manner: it reads the couplet projections from the given polytope and then computes the section projection considering the algorithm *getSection*. Once a section is obtained, it is possible to perform operations or some processing to it, as can be observed in line 10. In practical terms, Algorithm 1 serves as layout for performing operations between two nD-EVMs as it establishes the main workflow of some algorithms for the nD-EVM.

3.3.1. The Boolean Operations Algorithm. The Boolean Operations Algorithm under the nD-EVM is a very important procedure in the sense that it applies a Regularized Boolean Operation $\text{op}^* \in \{\cup^*, \cap^*, -^*, \otimes^*\}$ to a pair of given nD-OPPs p and q expressed in the nD-EVM. See Algorithm 2. The algorithm computes the resulting nD-OPP $r = p\text{op}^* q$. The main elements of the algorithm are described [31]:

```
    Input: An nD-EVM p.
(1) Procedure sectionSequence (nD-EVM p)
(2)         EVM couplet;
(3)         EVM Si, Sj; // Previous and current section
(4)         Couplet ← initEVM ( );
(5)         Si ← initEVM ( );
(6)         Sj ← initEVM ( );
(7)         couplet ← readCouplet (p);
(8)         while !endEVM (p) do
(9)             Sj ← getSection (Si, couplet);
(10)            Process the current section Sj;
(11)            Si ← Sj;
(12)            couplet ← readCouplet (p);
```

Algorithm 1: Computing the section sequence for a given nD-EVM p.

(i) For both nD-OPPs p and q their sections sequence is obtained as can be observed in lines 17 and 20. Because the algorithm performs the operations at the level of sections, it considers only the sections perpendicular to X_1-axis.

(ii) The order of appearance of couplets of both p and q is taken into account. This is performed in the function *nextObject* (line 13). It returns the value of the *coordinate* for the next couplet to process; this coordinate is common to both operands. It also sets two flags *fromP* and *fromQ*: they indicate from which operand the next couplet is taken. These flags are evaluated at the conditions of lines 15 and 18.

(iii) The resulting sections of r are computed as follows [30, 31]: $S_k^i(r) = S_k^i(p)\text{op}^* S_k^i(q)$. By this way a recursive pattern of execution for this algorithm is established, and it can be observed in line 22.

(iv) The sections for p and q are processed recursively until we reach the basic case: the Regularized Boolean Operations for 1D-OPPs which are managed by *booleanOperation1D* function in line 8. By this way, we can build the result r from its sections' projections, and the algorithm keeps track of the previous and current sections in order to compute the projection of the corresponding couplet, which is appended to the result in line 25.

(v) There are two repetition structures after the main *while* structure, at lines 27 and 30. They manage the case when one of the nD-OPPs has reached the end with respect to X_1-axis, but the other operand still has couplets to be operated. Therefore, in those *while* structures it is considered the Regularized Boolean Operation that is being applied, because it depends if the remaining couplets are taken into account for the resulting final nD-EVM.

```
Input: Two nD-EVMs p and q, the Boolean Operation op, and the dimensionality n.
Output: A resulting nD-EVM.
(1)  Procedure booleanOperation (nD-EVM p, nD-EVM q, booleanOperator op, integer n)
(2)        EVM pSection, qSection;
(3)        EVM couplet;
(4)        boolean fromQ, fromP;
(5)        real coord;
(6)        EVM result, rPrevSection, rCurrSection;
(7)        if n = 1 then
(8)             return booleanOperation1D (p, q, op);
(9)        else
(10)            pSection ← initEVM ( );
(11)            qSection ← initEVM ( );
(12)            rCurrSection ← initEVM ( );
(13)            nextObject (p, q, fromP, fromQ);
(14)            while !(endEVM (p)) and !(endEVM (q)) do
(15)                 if fromP then
(16)                      couplet ← readCouplet (p);
(17)                      pSection ← getSection (pSection, couplet);
(18)                 if fromQ then
(19)                      couplet ← readCouplet (q);
(20)                      qSection ← getSection (qSection, couplet);
(21)                 rPrevSection ← rCurrSection
(22)                 rCurrSection ← booleanOperation (pSection, qSection, op, n − 1);
(23)                 couplet ← getCouplet (rPrevSection, rCurrSection);
(24)                 setCoord (couplet, coord);
(25)                 putCouplet (couplet, result);
(26)                 nextObject (p, q, coord, fromP, fromQ);
(27)            while !endEVM (p) do
(28)                 couplet ← readCouplet (p);
(29)                 putBool (couplet, result, op);
(30)            while !endEVM (q) do
(31)                 couplet ← readCouplet (q);
(32)                 putBool (couplet, result, op);
(33)       return result
```

Algorithm 2: Computing the Regularized Boolean Operations for two nD-EVMs.

3.3.2. Computing the Content of an nD-OPP. As described in [31], we aim to compute the nD content enclosed by an nD-OPP by considering its slices. As we know, a slice is a set of one or more disjoints nD hyperprisms; it has a $(n-1)$D section as its base and two boundary $(n-1)$D couplets. Let us take a 3D-OPP p as an example. The content of p can be obtained from the sum of the volumes of its 3D slices. Now, the volume of a $\text{Slice}_k^i(p)$ is given by the product between the area of the 2D section $S_k^i(p)$ (i.e., the 2D base of the 3D slice) and the distance between its two boundary 2D couplets $\Phi_k^i(p)$ and $\Phi_{k+1}^i(p)$ (it is the height of the 3D slice). Now suppose that we have a 2D-OPP $q = S_k^i(p)$ which in fact is a 2D section of p and the base of $\text{Slice}_k^i(p)$. The content of q, its area, is the sum of the areas of the 2D slices that form q. The area of a 2D is given by the product of the length of the 1D section $S_k^i(q)$ as its base and the distance between 1D couplets $\Phi_k^i(q)$ and $\Phi_{k+1}^i(q)$. In the basic case, for an 1D-OPP r, the content of r is computed as the sum of the lengths of its brinks. Now, the recursive formula for computing the content of a nD-OPP p is the following [31]:

$$\text{Content}_n(p) = \begin{cases} \text{Length}(p) & n = 1 \\ \sum_{k=1}^{np_i - 1} \text{Content}_{n-1}\left(S_k^i(p)\right) \cdot \text{Dist}\left(\Phi_k^i(p), \Phi_{k+1}^i(p)\right) & n > 1. \end{cases} \tag{14}$$

Algorithm 3 is the implementation for (14). It computes the nD content enclosed by a given nD-OPP that is represented through the nD-EVM.

3.3.3. Computing the Discrete Compactness Descriptor. The *Discrete Compactness* (DC) descriptor is very useful for characterizing objects considering their similarity with some ideal objects. Moreover, Discrete Compactness improves the classical *Shape Compactness* descriptor [44, 45].

It is important to mention the definition of a hypervoxelization [30, 31], which is the basis for properly establishing

```
     Input: A nD-EVM p and the number of dimensions n.
     Output: The content of the nD space enclosed by p.
(1)  Procedure Content (nD-EVM p, n)
(2)          contSum ← 0;
             /* Objects initialization */
(3)          EVM couplet 1, couplet 2;
(4)          EVM section;
(5)          if n == 1 then
(6)                  return Length (p);
(7)          couplet 1 ← InitEVM ();
(8)          couplet 2 ← InitEVM ();
(9)          section ← InitEVM ();
(10)         couplet 1 ← readCouplet (p);
(11)         while !endEVM (p) do
(12)                 couplet 2 ← readCouplet (p);
(13)                 section ← getSection (section, couplet 1);
(14)                 contSum ← contSum + Content (section, n − 1) ∗ dist (couplet 1, couplet 2);
(15)                 couplet 1 ← couplet 2;
(16)         return contSum
```

Algorithm 3: Computing the nD content enclosed by a given nD-EVM p.

the DC descriptor. A hypervoxelization is a Hyperspatial Occupancy Enumeration scheme for polytopes. A hypervoxelization is a list of cells in the Hyperspace occupied by a polytope. These cells are called hypervoxels and they are hyperboxes of a fixed size in the nD space. In this sense, a hypervoxel in the 2D space is a pixel, a 3D hypervoxel is a voxel, and for the 4D space the term *rexel* is recommended. A collection of hypervoxels is codified as an array $C_{x_1,x_2,\ldots,x_n}$ of dimensionality n, in which a black (occupied) hypervoxel and a white (unoccupied) hypervoxel are specified when $C_{x_1,x_2,\ldots,x_n} = 1$ and $C_{x_1,x_2,\ldots,x_n} = 0$, respectively. A polytope is then defined as a collection of occupied hypervoxels. It is important to note that hypervoxels are in fact disjoint or quasi-disjoint, which means that they do not overlap. The result of the union of all black hypervoxels in a given hypervoxelization describes in fact a nD-OPP p:

$$p = \bigcup_{\lambda} \text{BlackHypervoxel}_{\lambda}. \tag{15}$$

Considering that from a black nD hypervoxel 2^n extreme vertices can be generated, the corresponding nD-EVM representation of p can be obtained as follows:

$$\begin{aligned} \text{EVM}_n(p) &= \text{EVM}_n\left(\bigcup_{\lambda} \text{BlackHypervoxel}_{\lambda}\right) \\ &= \bigotimes_{\lambda} \text{EVM}_n\left(\text{BlackHypervoxel}_{\lambda}\right). \end{aligned} \tag{16}$$

For computing the DC descriptor it is important to consider that the nD-OPPs were originally expressed as hypervoxelizations where the hypervoxels are hypercubes and their coordinates are integers. In [46, 47] the foundations for computing DC for 2D and 3D objects are presented. In [13] computation for objects in the nD space $n \geq 3$ was presented. In any case, formula for DC descriptor is the following:

$$\text{DC}(p) = \frac{L_c(p) - L_{\text{Min}}}{L_{\text{Max}} - L_{\text{Min}}}. \tag{17}$$

$L_c(p)$ is the number of internal contacts for the nD-OPP p. L_{Min} is the minimal number of internal contacts for an ideal nD object with the same content as p. In similar way, L_{Max} is the maximal number of internal contacts for an ideal nD object with the same content as p. The internal contacts for an nD-OPP are the $(n-1)$D cells that are shared between the nD hypervoxels that originally defined p. L_{Min} and L_{Max} must be specified according to the desired application. For an nD-OPP represented through the nD-EVM, the procedure for computing the internal contacts perpendicular to X_1-axis is shown in Algorithm 4. This procedure is described as follows:

(i) We process the slices aligned with respect to X_1-axis.

(ii) The number of internal x_1-coordinates are obtained for the current slice. This is achieved by obtaining the x_1-coordinates c_1 and c_2 for the bounding couplets of the slice (lines 12 and 13):

$$n\text{Coords} = c_2 - c_1 - 1. \tag{18}$$

(iii) For every slice the $(n-1)$D content of the corresponding section S_j is obtained. The internal contacts perpendicular to X_1-axis are computed as follows (line 14):

$$\text{IC} = n\text{Coords} \cdot \text{Content}\left(S_j\right). \tag{19}$$

(iv) It is considered that there may be contiguous and isolated slices; for this reason we need to obtain the internal contacts between boundaries of consecutive

```
Input: A nD-EVM p and the number of dimensions n.
Output: The number of internal contacts perpendicular to X_1-axis.
(1)  Procedure InternalContacts (nD-EVM p, n)
(2)        EVM couplet, currentSection, previousSection;
(3)        EVM sectionInt; // Intersection result
(4)        integer c1, c2; // Couplets coordinates
(5)        integer nCoords;
(6)        integer ic ← 0; // Total internal contacts
(7)        previousSection ← InitEVM ( );
(8)        c1 ← getCoord (p);
(9)        couplet ← readCouplet (p);
(10)       while !endEVM (p) do
                 // Internal contacts for the current slice
(11)             currentSection ← getSection (previousSection, couplet);
(12)             c2 ← getCoord (p);
(13)             nCoords ← c2 − c1 − 1;
(14)             ic ← ic + nCoords ∗ Content (currentSection, n − 1);
                 // Internal contacts between consecutive slices
(15)             sectionInt ← BooleanOperation (previousSection, currentSection, ∩*, n − 1);
(16)             ic ← ic + Content (sectionInt, n − 1);
(17)             previousSection ← currentSection;
(18)             c1 ← c2;
(19)             couplet ← readCouplet (p);
(20)       return ic
```

Algorithm 4: Computing the $(n-1)$D adjacencies, shared $(n-1)$D cells, between the hypervoxels that originally defined to a given nD-OPP p.

```
Input: A nD-EVM p and the number of dimensions n.
Output: The total number of internal contacts for the hyper-boxes of a given nD-EVM
(1)  Procedure TotalInternalContacts (nD-EVM p, n)
(2)        integer ic ← 0; // Total internal contacts
(3)        EVM SortedP;
(4)        for each sorting ∈ AxisSorting do
(5)              SortedP ← sortEVM (p, n, sorting);
(6)              ic ← ic + InternalContacts (SortedP, n);
(7)        return ic
```

Algorithm 5: Computing the total internal contacts, $L_c(p)$, for a given nD-OPP p.

slices. This is achieved by computing the regularized intersection between consecutive sections and then computing the content of the result (lines 15 and 16).

Since Algorithm 4 considers its processes in terms of the first coordinate, it will be required to apply it for every x_i-coordinate, $1 \leq i \leq n$. At the implementation level, this is easy by reordering the coordinates of the EVM representation as follows:

$$\begin{aligned}\text{AxisSorting} = \{&\{x_1, x_2, \ldots, x_{n-1}, x_n\},\\ &\{x_2, x_3, \ldots, x_n, x_1\}, \{x_3, x_4, \ldots, x_1, x_2\}, \ldots,\\ &\{x_n, x_1, \ldots, x_{n-2}, x_{n-1}\}\}.\end{aligned} \tag{20}$$

In order to compute the total internal contacts it is required to process all the possible axis orderings. In Algorithm 5 the procedure for computing the total internal contacts is shown, in which a new nD-EVM is obtained from a different axis ordering in line 5. In line 6 the cumulative sum of the total internal contacts is performed.

Before computing the DC descriptor for a given nD-OPP p it is important to define the application context. So, it is required to provide the two nD-OPPs p_l and p_u which represent the ideal objects against all the pertinent nD-OPPs that will be compared. Once those bounds have been defined, we compute via Algorithm 5 the minimum and maximum internal contacts as follows:

$$\begin{aligned}L_{\text{Min}} &= \text{TotalInternalContacts}\left(p_l, n\right),\\ L_{\text{Max}} &= \text{TotalInternalContacts}\left(p_u, n\right).\end{aligned} \tag{21}$$

$L_c(p)$ is the number of total internal contacts for p, and finally the DC descriptor is computed using (17).

```
Input: The path of the frames and time positions for initial and final frame.
(1)  Procedure generateAnimation (String workingDirectory, integer initFrame, integer endFrame)
(2)        EVM currentFrame, prevFrame, diffFrame;
(3)        string framePath;
(4)        prevFrame ← initEVM ( );
(5)        currentFrame ← initEVM ( );
(6)        for i = initFrame to endFrame do
(7)              currentFrame ← initEVM ( );
(8)              framePath ← workingDirectory + "frame" + toString (i) + ".image";
(9)              currentFrame ← loadImage (framePath);
(10)             diffFrame ← getCouplet (prevFrame, currentFrame);
(11)             saveEVM (diffFrame, i);
(12)             prevFrame ← currentFrame;
(13)       saveEVM (prevFrame, endFrame + 1);
```

Algorithm 6: Generating an EVM-animation in the nD-EVM from a frame sequence.

4. Video Processing with the nD-EVM Model and Self-Organizing Maps

In this section we present the foundations for our proposed framework. Summarizing, the framework is composed by four stages.

Stage 1. Representation of a frame sequence: it is defined as the workflow for representing a given frame sequence in the nD-EVM.

Stage 2. Subanimations extraction: it aims to obtain nD-EVMs that contain small portions of the animation.

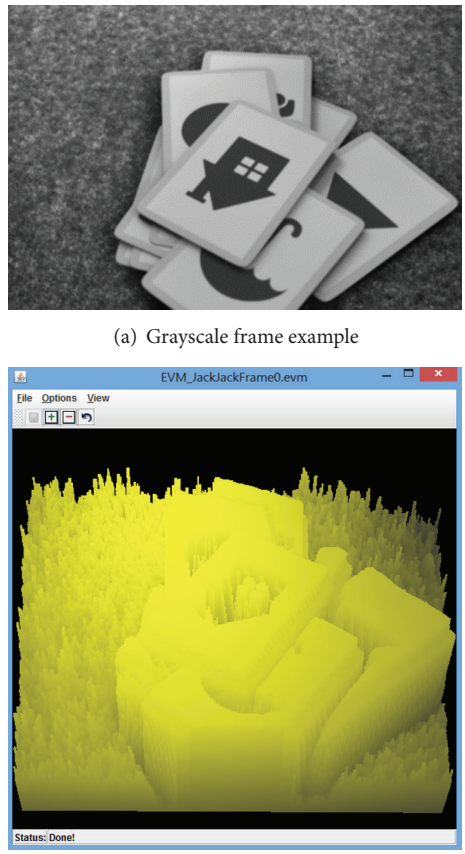
Stage 3. Computing the DC descriptor for every subanimation: a set that contains all the DC values is formed.

Stage 4. Subanimations clustering using a SOM approach: the subanimations are grouped into m clusters. As a result we obtain m 3D-EVMs P_c, $1 \leq c \leq m$, one for each cluster.

4.1. Representation of Frame Sequences. For this stage it is considered that the frames are given in a plain image format, in this particular case the BMP image format. At first, each image defines a 2D rectangular region, the lengths are specified as the *width* and *height* of the frame, and for the color extrusion the color scheme plays an important role. The amount of color components in the color scheme establishes the additional dimensions for the polytopes' dimensionality.

There are many color schemes, for example, grayscale, RGB, RGBA, or CMYK. The grayscale scheme only specifies one color component, which means that to represent a frame this would be achieved with a 3D-OPP. On the other hand, a frame in a RGB color scheme is represented through a 5D-OPP.

Figure 4 is an example of a grayscale frame 3D representation; the original frame is shown in Figure 4(a). Figure 4(b) illustrates its color extrusion towards the third dimension through a software designed specifically for 3D-EVM edition and visualization [48, 49].



(a) Grayscale frame example

(b) 3D extrusion

Figure 4: An example of color component extrusion.

The representation of a frame sequence in the nD-EVM model is called an *EVM-animation*. For an EVM-animation it is considered one additional dimension for the time in which the frames are sorted. Therefore, a RGB frame sequence is represented through a 6D-EVM. Algorithm 6 provides a generic procedure for generating an EVM-animation from a frame sequence.

```
     Input: The frame file path.
     Output: An EVM that is an extrusion for the input frame.
(1)  Procedure loadImage (string imagePath)
(2)        EVM p // The resulting EVM.
(3)        ImageData imageData;
(4)        integer colorCount;
(5)        integer array hyperPrismBase, hyperPrismLengths;
(6)        p ← InitEVM ( );
(7)        imageData ← readImageData (imagePath);
(8)        colorCount ← imageData.colors;
           /* Initialize the hyper-prism base and lengths */
(9)        for k = 0 to (colorCount − 1) do
(10)             hyperPrismBase [2 + k] ← 0;
(11)       hyperPrismLengths [0] ← 1;
(12)       hyperPrismLengths [1] ← 1;
(13)       for i = 0 to (imageData.height − 1) do
(14)             hyperPrismBase [1] ← i;
(15)             for j = 0 to (imageData.width − 1) do
(16)                  hyperPrismBase [0] ← j;
                     /* Get every color component for the current pixel. */
(17)                 for k = 0 to (colorCount − 1) do
(18)                      hyperPrismLengths [2 + k] ← imageData.pixelData (i, j).color (k);
(19)                 populateHyperPrism (p, hyperPrismBase, 2 + colorCount, 0, hyperPrismLengths);
(20)       return p
```

Algorithm 7: Algorithm for converting a frame to a EVM object.

The workflow for Algorithm 6 is described as follows:

(i) For every frame its EVM representation is obtained.

(ii) Later, the differences are computed between consecutive frames. These differences are by instance expressed as a couplet. By this way the couplets' projections are obtained.

Algorithm 6 indicates that only the couplets will be stored. The reason for this is because they have less information than sections (the frames themselves), and the original frames can be retrieved using Algorithm 1.

As we can see in Algorithm 6, the function *loadImage* is required which builds the extrusion of the frame and returns its EVM representation. This function performs a double iteration for retrieving information from every pixel from the frame. Algorithm 7 shows the generic procedure for obtaining the extrusion of a frame.

In the context of an extrusion, for every pixel of a frame a hyperprism whose base is built on the 2D coordinates of the pixel and the color components to zero is obtained (or the lowest value allowed). Therefore, as shown in Algorithm 7, two arrays are defined, *hyperPrismBase* and *hyperPrismLengths* (line 5). Their purpose is to store the hyperprism initial coordinate (lines 9, 10, 14, and 16) and lengths (lines 11, 12, and 18), respectively. The lengths for the color components are equal to the pixel color information. These arrays are later sent as input to the *populateHyperPrism* function (line 19) which performs the hyperprism population based in the pixel position and color information. For each hyperprism in the nD space it is possible to generate 2^n vertices, which are indeed extreme vertices. In this sense the final EVM for the given frame consists of the union of all the hyperprisms obtained from it. In this sense, it is important to note that all the hyperprisms are disjoint or quasi-disjoint; this means that they do not overlap. As described in [30, 31], when having two nD-OPPs p and q with those properties, considering their respective EVM representation, $\text{EVM}_n(p \cup q) = \text{EVM}_n(p) \otimes \text{EVM}_n(q)$.

In Algorithm 8 the procedure for obtaining a nD hyperprism from *hyperPrismBase* and *hyperPrismLengths* arrays is shown. The function *insertVertex* tries to insert a vertex from the current hyperprism to EVM p. In case the vertex does not exist, then it is inserted to p; otherwise it deletes the vertex from p. This functionality provides the same behavior as the XOR operator, because it removes common vertices.

Considering an example of representing a video with a grayscale color scheme, in Figure 5 the resulting 3D couplet computed from the difference of two consecutive frames is shown.

4.2. Extraction of Subanimations. We know now an EVM-animation is a frame sequence represented through the EVM model. Moreover, we have stated a subanimation is a small portion from the original EVM-animation. In order to extract a subanimation we are going to use a *mask*. In the image processing field, a mask is a small 2D region of fixed size, which is used for obtaining or changing information of an image [34]. This is achieved by exploring, with the mask, all possible regions in the image and obtaining the pixels' information or performing the corresponding operations. Image filters are well-known examples of operators that make use of masks.

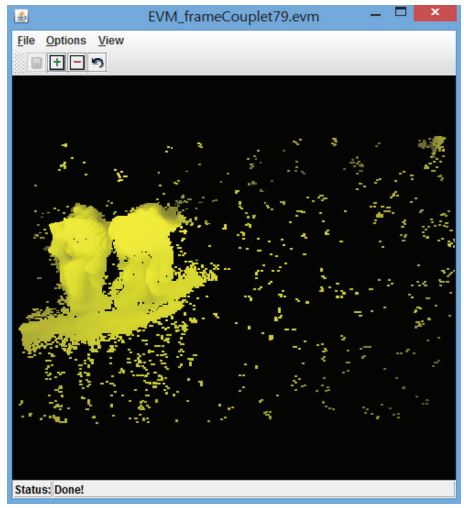
Input: The nD-EVM p, the hyperprism base and lengths, current the number of dimensions, and the number of dimensions.

(1) **Procedure *populateHyperPrism*** (nD-EVM p, integer array *hyperPrismBase*, integer *dimensions*, integer *currentDim*, integer array *hyperPrismLengths*)
(2) **if** !(*currentDim* < *dimensions*) **then**
(3) insertVertex (*p, hyperPrismBase, dimensions*);
(4) **return**;
(5) populateHyperPrism (*p, hyperPrismBase, dimensions, currentDim* + 1, *hyperPrismLengths*);
(6) *hyperPrismBase* [*currentDim*] ← *hyperPrismBase* [*currentDim*] + *hyperPrismLengths* [*currentDim*];
(7) populateHyperPrism (*p, hyperPrismBase, dimensions, currentDim* + 1, *hyperPrismLengths*);
(8) hyperPrismBase [currentDim] ← hyperPrismBase [currentDim] – hyperPrismLengths [currentDim];

Algorithm 8: Algorithm to populate the hyperprism from the *hyperPrismBase* and *hyperPrismLengths* arrays.

(a) First frame

(b) Second frame



(c) Couplet

Figure 5: Computing a couplet from two consecutive frames.

In this work, we understand a mask is a small EVM-animation that has a fixed 2D base, a set of dimensions depending on the color scale, and a temporal dimension. That is, dimensionality of the mask is the same as the original EVM-animation. More specifically, due to the operations to be performed, color values for the mask are set to the maximum level. In the RGB context, this configuration corresponds to an EVM-animation where all frames have white color. For generating a mask's EVM representation Algorithm 8 is applied by providing it the initial coordinate and the mask's corresponding lengths for each dimension. Once the mask has been configured and generated, for extracting a subanimation from the original one, the Regularized Intersection Operation between both EVM representations is performed: the EVM-animation and the mask.

We exemplify the above procedure by assuming a given EVM-animation can be represented through an 3D-OPP; hence, its frames and couplets are represented through 2D-EVMs. Figure 6 presents an EVM-animation with these properties. A mask placed in some location inside the EVM-animation is also shown. Under this scenario we can see a subanimation can be considered as an OPP that contains

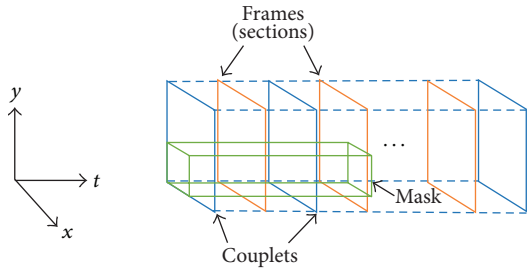


Figure 6: Example of an EVM-animation and a mask for subanimation extraction.

the changes through time of a small 2D region of some consecutive frames. For obtaining all possible subanimations, there are performed shifts on the EVM-animation's possible coordinates x_1, x_2, and t.

4.3. Computing the DC Descriptor for Subanimations. In order to differentiate the subanimations the DC descriptor is computed for each one. As explained in Section 3.3.3, it is required to provide a pair of bounds for DC descriptor (see (17)). These bounds are set as follows:

(i) L_{Max} is obtained directly from the mask to be utilized, where all color components are set to their maximum values.

(ii) L_{Min} is computed from a mask with a similar configuration as the previous one. The difference lies in the color components' values. They are set to the minimum possible values. For example, under the RGB color scheme we assign color values such that the mask refers to an animation with black frames.

Now, we proceed to generate the training set for the SOM clustering. That set contains all DC values for every subanimation. This set is called *DCValues* and its generation is based on the following steps:

(i) An iteration is performed over all the x_1 and x_2 coordinates covered by the original EVM-animation. This is analogous to the exploration of all pixels in an image. In our specific case, the exploration is performed by applying unitary shifts on the original mask.

(ii) Once the previous iteration has finished, a unit shift on the temporal dimension is applied, and the previous step is performed again. Time shifts are performed until all temporal coordinates covered by the EVM-animation have been processed.

(iii) For every subanimation obtained in the previous steps, its corresponding DC descriptor is computed.

The elements of set *DCValues* are organized as follows:

$$DCValues = \begin{Bmatrix} maskDC_{0,1} & maskDC_{0,2} & \cdots & maskDC_{0,i} \\ maskDC_{1,1} & maskDC_{1,2} & \cdots & maskDC_{1,i} \\ & \vdots & & \\ maskDC_{\tau,1} & maskDC_{\tau,2} & \cdots & maskDC_{\tau,i} \end{Bmatrix}, \quad (22)$$

where

(i) τ denotes temporal position of subanimation $maskDC_{\tau,i}$. The maximum temporal position depends on the original EVM-animation's frames count and the mask's temporal length. Hence $\tau = 0, 1, \ldots, (endFrame - mask.timeLength + 1)$,

(ii) index i is assigned to every subanimation such that when a time shift is applied then i is set to zero in order to count subanimations associated with the new temporal coordinate. Therefore, index i is bounded as follows:

$$i = 0, 1, \ldots, (maxXShift \cdot maxYShift), \quad (23)$$

where

$$maxXShift = frame.width - mask.xLength + 1,$$
$$maxYShift = frame.length - mask.yLength + 1. \quad (24)$$

As previously commented, this process is similar to a convolution because we iterate over every shift that can be applied to the mask for subanimation extraction. Algorithm 9 presents the generic for performing our proposed convolution process in order to build set *DCValues* from a given EVM-animation. The main flow of Algorithm 9 is described as follows:

(i) In lines 2–9 the variables initialization is performed.

(ii) The function *maskAnimSections* (lines 10 and 30) obtains an EVM-animation that contains only the frames (sections) that are currently intersected by the mask in its current position. As can be observed in lines 10 and 30, these frames are stored in variable *frameSeq*. By this way, the Regularized Intersection Operation is improved because it will not be required to compute all previous, and unused, sections just like the classical Boolean Operations Algorithm (see Algorithm 2).

(iii) The exploration of the EVM-animation, over X_1-, X_2-, and t-axis, is performed by means of the iterative while structures are specified in lines 15, 16, and 17.

(iv) In line 18 the Regularized Intersection Operation is performed between the current mask and the EVM-animation stored in the *frameSeq* object. As can be noted, the intersection operation is performed by using the function *maskIntersection*, which is a modified version of Algorithm 2. As mentioned

```
     Input: The nD-EVM for the mask, temporal position of the final frame, and the width and height of the frame.
     Output: The DCValues set that contains the values of DC descriptor for every possible sub-animation.
(1)  Procedure AnimConv (nD-EVM mask, integer endFrame, integer width, integer length)
(2)        EVM currentResult; // Current intersection result
(3)        EVM frameSeq;
(4)        integer τ ← 0, maxTimeShift;
(5)        integer x ← 0, maxXShift;
(6)        integer y ← 0, maxYShift;
(7)        real maskDC;
(8)        Set of reals DCValues;
(9)        integer i ← 0; // Counter for every time shift.
           /* Obtaining the frame sequence required for the intersection operation. */
(10)       frameSeq ← maskAnimSections (mask, endFrame);
           /* Obtaining the maximum shift for the time, X and Y dimensions. */
(11)       maxTimeShift ← endFrame − mask.timeLength + 1;
(12)       maxXShift ← width − mask.xLength + 1;
(13)       maxYShift ← length − mask.yLength + 1;
(14)       DCValues ← ∅;
(15)       while τ ≤ maxTimeShift do
(16)            while y ≤ maxYShift do
(17)                 while x ≤ maxXShift do
(18)                      currentResult ← maskIntersection (mask, frameSeq);
(19)                      maskDC_{τ,i} ← discreteCompactness (currentResult, mask.LcMin, mask.LcMax);
(20)                      DCValues.addDCValue (maskDC_{τ,i});
(21)                      mask.EVMTraslation (1, 1);
(22)                      i ← i + 1;
(23)                      x ← x + 1;
(24)                 mask.dimReset (1);
(25)                 mask.EVMTraslation (2, 1);
(26)                 y ← y + 1;
(27)                 x ← 0;
(28)            mask.dimReset (2);
(29)            mask.EVMTraslation (0, 1);
(30)            frameSeq ← maskAnimSections (mask, endFrame);
(31)            τ ← τ + 1;
(32)            y ← 0;
(33)            i ← 0;
(34)       return DCValues
```

Algorithm 9: Algorithm for the convolution process to generate set *DCValues*.

before, with this function the intersection operation is improved in temporal terms because it does not require to compute all the previous sections in the original EVM-animation.

(v) In line 19 the DC descriptor for the current subanimation is computed. By this way set *DCValues* is being formed which contains DC values for each subanimation (line 20).

(vi) At lines 23, 26, and 31 the mask shifts for each axis are performed.

(vii) Finally, the final set *DCValues* is returned as output.

4.4. Subanimations Clustering Using the SOM Approach. According to the SOM's training stage described in Section 2, in first place the training set's presentations are performed to the network. Later, when the training stage has been achieved, another presentation is performed for obtaining the corresponding winning neuron for every subanimation. For a subanimation, the winning neuron gives the *cluster* number which it belongs to. Let us suppose that we have m neurons; therefore we obtain m clusters $cluster_j$, $j = 1, 2, \ldots, m$. A given $cluster_j$ will contain the indexes of the subanimations for which j is the corresponding winning neuron:

$$cluster_j = \{(\tau, i) : maskDC_{\tau,i} \in DCValues, \text{minDistanceNeuron}\left(maskDC_{\tau,i}\right) = j\}. \quad (25)$$

As can be seen, the clusters only contain indexes for their associated subanimations. In fact, in this stage such indexes are important because they are required for identifying all the subanimations that are going to be used in order to get the whole subanimation of a given cluster. By the way, a 3D-OPP is obtained for every cluster, which contains only the regions,

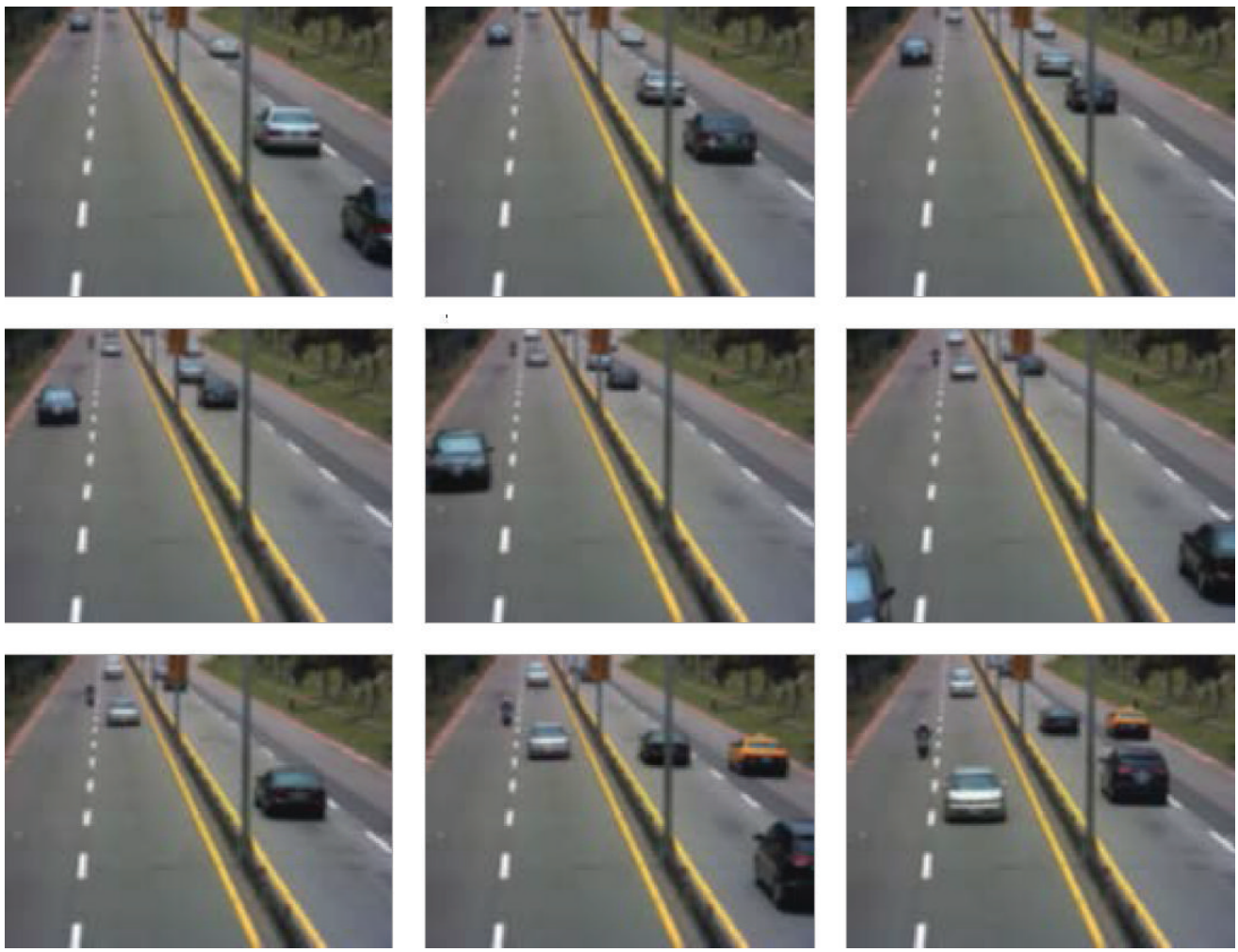


Figure 7: Some frames of the traffic control video sequence.

with no color information, that are grouped in a given cluster. These 3D-OPPs consider only original X_1- and X_2-axis and also the temporal dimension. Nevertheless it is possible to query the original EVM-animation for obtaining the original color information for those regions.

5. Segmentation Results

In this section we are going to verify that our proposed framework achieves its main objective: to obtain segmented and concise versions from the original EVM-animation. For this purpose we have run some tests and we present a description of the results. In all our tests the configuration of the mask consists in a 2D region of size 3×3 and 3 frames as the time length.

5.1. Tests for a Traffic Control Video Sequence. We analyze segmentation results for a traffic control video sequence taken from [50]. It consists in a static camera pointing to the center of a highway. Figure 7 shows some selected frames from this video sequence. As can be observed, the scene contains the movement of cars along the highway while at left and right sides green zones are found. The frames of this video sequence have a size of 160×120 pixels, and we have 60 consecutive frames at a rate of 20 frames per second. For this case, the frame sequence was represented in an EVM-animation that consists of a 6D-EVM with a total of 17,058,774 extreme vertices. The SOM's tested configurations were 5 and 15 neurons. In both cases the training stage executed 200 iterations. The results are analyzed in the following subsections.

5.1.1. Results for 5 Neurons. In Table 1 the sizes of every 3D-EVM are shown associated with every cluster. It can be observed that we have a total of 80,128 extreme vertices. By comparing against the 6D-EVM of the original EVM-animation we obtained a compression rate of 212.89. In Figure 8 the quantity of subanimations grouped in each cluster is presented.

Now we describe some information contained in clusters 0, 2, and 4:

(i) For cluster 0 its frame sequence is shown in Figure 9. It can be observed that it grouped information related

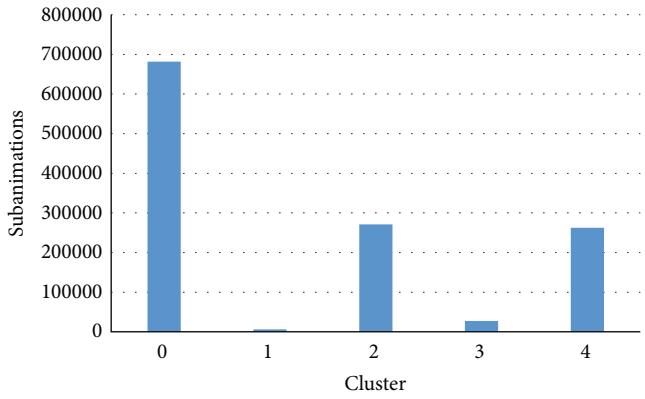


Figure 8: Number of subanimations for each cluster associated with the traffic control video sequence under a SOM with 5 neurons.

Table 1: Size of the 3D-EVMs generated for every cluster associated with the traffic control video sequence under a SOM with 5 neurons.

Class	EVM size
0	28,968
1	1,528
2	24,674
3	6,308
4	18,650
Total	**80,128**

to dark regions of the highway and also contains movement of the cars. In Figure 10 the 3D-OPP can be observed generated for cluster 0. In this 3D-OPP the temporal evolution of the highway's grouped regions can be appreciated.

(ii) Figure 11 presents the frame sequence obtained for cluster 2. As can be observed, the brighter regions of the highway are grouped as well as the movement of the cars that take place in those regions. In Figure 12 the 3D-OPP is shown generated for cluster 2. In it, the regions can be seen grouped for the highway, the movement of the cars, and the evolution of those regions across time.

(iii) For cluster 4 we present the frame sequence shown in Figure 13. This cluster contains important information because we can observe only the movement of the cars on the highway. This implies the SOM captured movement patterns and tried to separate them in a single cluster. Nevertheless, it also groups information of the highway's green zones. This is because they have some similarities with the cars' shadows in the sense that they are dark regions. In Figure 14 the 3D-OPP associated with cluster 4 is shown. In this OPP clearly we can observe regions correspondent to the movement of the cars across the time axis.

5.1.2. Results for 15 Neurons. In Table 2 the sizes of every 3D-EVM associated with every cluster are shown. We have

Table 2: Size of the 3D-EVMs generated for every cluster associated with the traffic control video sequence under a SOM with 15 neurons.

Class	EVM size
0	176
1	38,826
2	8,560
3	15,116
4	3,070
5	23,346
6	1,408
7	8,352
8	33,272
9	45,096
10	496
11	0
12	0
13	4,796
14	23,626
Total	**206,140**

a total of 206,140 extreme vertices. By comparing against the 6D-EVM of the original EVM-animation we obtained a compression rate of 82.75. In Figure 15 the quantity of subanimations grouped in each cluster is presented.

We present a brief description of information contained in clusters 1, 5, and 9:

(i) For cluster 1, in Figure 16, its frame sequence is shown. It is observed that this cluster groups darker regions of the highway and also the movements of cars which take place in such regions. Now, the corresponding 3D-OPP is shown in Figure 17. It is possible to observe the movement of the cars, the abovementioned darker regions, and their evolution across the temporal dimension.

(ii) In the case of cluster 5 see Figure 18 for its frame sequence. As can be observed, for this cluster the SOM has provided a proper separation of the movement of the cars. In Figure 19 the 3D-OPP is shown formed with all the subanimations grouped. Clearly, we can observe the movement of cars in the center of the highway. There are also grouped regions from green zones because they are also dark zones.

(iii) The last cluster to be described is number 9. Its frame sequence is shown in Figure 20, where we can observe that it groups the brighter regions of the highway and also grouped the movement of the cars that takes place in those zones. In Figure 21 the corresponding 3D-OPP is shown.

5.2. Tests for a Billiard Game Video Sequence. Now we proceed to analyze segmentation results for a video sequence that consists in a game of a professional billiard player [51]. This video is a capture of a single play. Figure 22 shows some selected frames from this video sequence. The frames

Figure 9: Traffic control video sequence under SOM with 5 neurons: segmentation results for cluster 0.

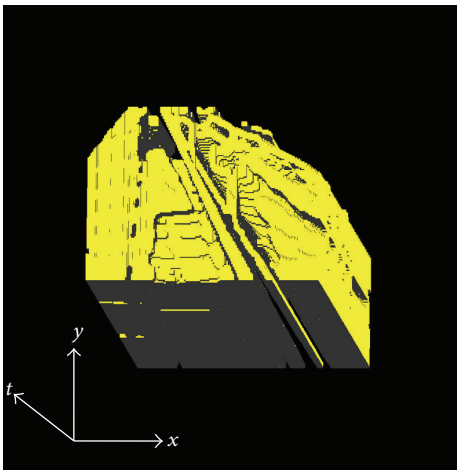


Figure 10: 3D-OPP associated with cluster 0: traffic control video sequence and 5 neurons.

of this video sequence have a size of 240 × 160 pixels, and we have 48 consecutive frames under RGB color model. For this case, the frame sequence was represented as an EVM-animation that consists of a 6D-EVM with a total of 22,076,702 extreme vertices. The SOM's tested configurations were 10 and 15 neurons. In both cases the training stage executed 200 iterations. The results are analyzed in the following subsections.

5.2.1. Results for 10 Neurons. In Table 3 the sizes of every 3D-EVM are shown associated with each cluster. It can be observed that we have a total of 76,754 extreme vertices. By comparing against the 6D-EVM of the original EVM-animation we obtained a compression rate of 287.62. In Figure 23 the quantity of subanimations grouped in each cluster is presented.

Now we provide a brief description of the information contained in clusters 1, 4, and 6:

(i) Cluster 1 has the frame sequence shown in Figure 24. It contains regions corresponding to the borders of the elements present in the scene. In Figure 25 the 3D-OPP can be seen associated with the current cluster. In such OPP we can clearly observe the borders

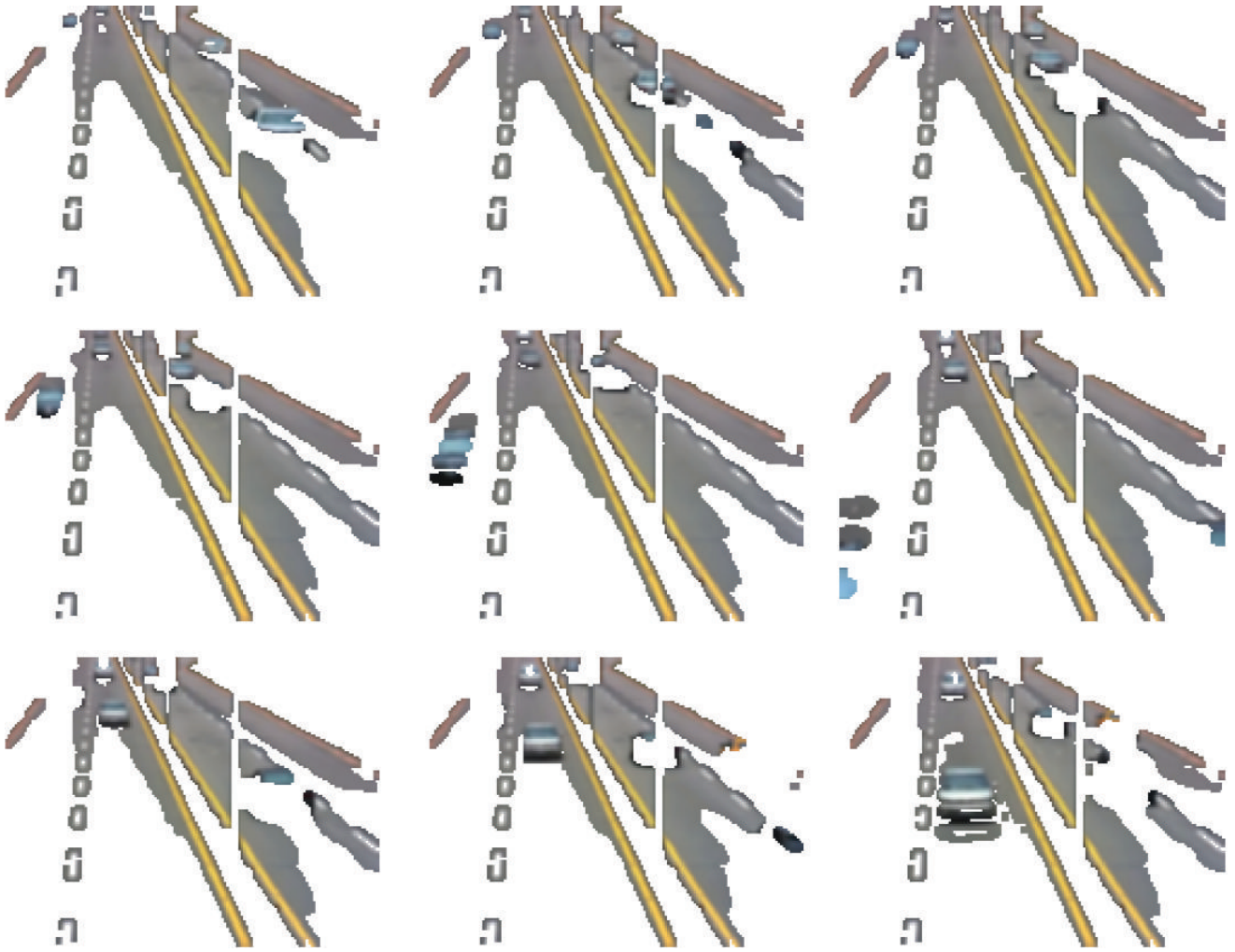


Figure 11: Traffic control video sequence under SOM with 5 neurons: segmentation results for cluster 2.

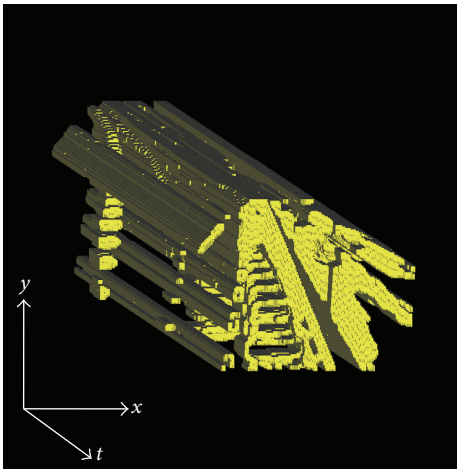


Figure 12: 3D-OPP associated with cluster 2: traffic control video sequence and 5 neurons.

previously mentioned and also their evolution along the temporal dimension can be seen.

Table 3: Size of the 3D-EVMs generated for every cluster associated with the billiard video sequence under a SOM with 10 neurons.

Class	EVM size
0	2,352
1	10,364
2	13,604
3	866
4	4,434
5	4,552
6	6,614
7	18,520
8	14,194
9	1,254
Total	**76,754**

(ii) Cluster 4 contains the frame sequence that is shown in Figure 26. It can be observed that it groups the regions corresponding to the movement of the billiard balls. In this case, the SOM has been able to distinguish

Figure 13: Traffic control video sequence under SOM with 5 neurons: segmentation results for cluster 4.

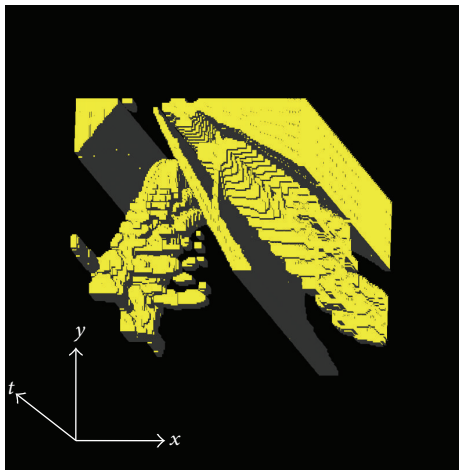


Figure 14: 3D-OPP associated with cluster 4: traffic control video sequence and 5 neurons.

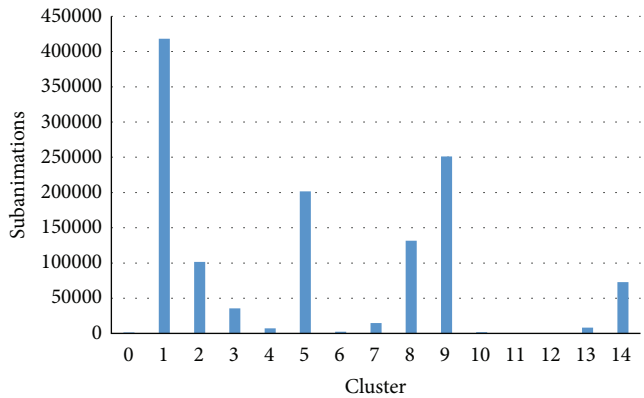


Figure 15: Number of subanimations for each cluster associated with the traffic control video sequence under a SOM with 15 neurons.

movement patterns. In Figure 27 the 3D-OPP for cluster 4 can be observed. In it, we can clearly note the regions of the movement of the billiard balls.

(iii) Cluster 6 has the frame sequence shown in Figure 28. We can observe it also groups regions related to the billiard balls movements, but in this case it also groups other small bright regions from the scene.

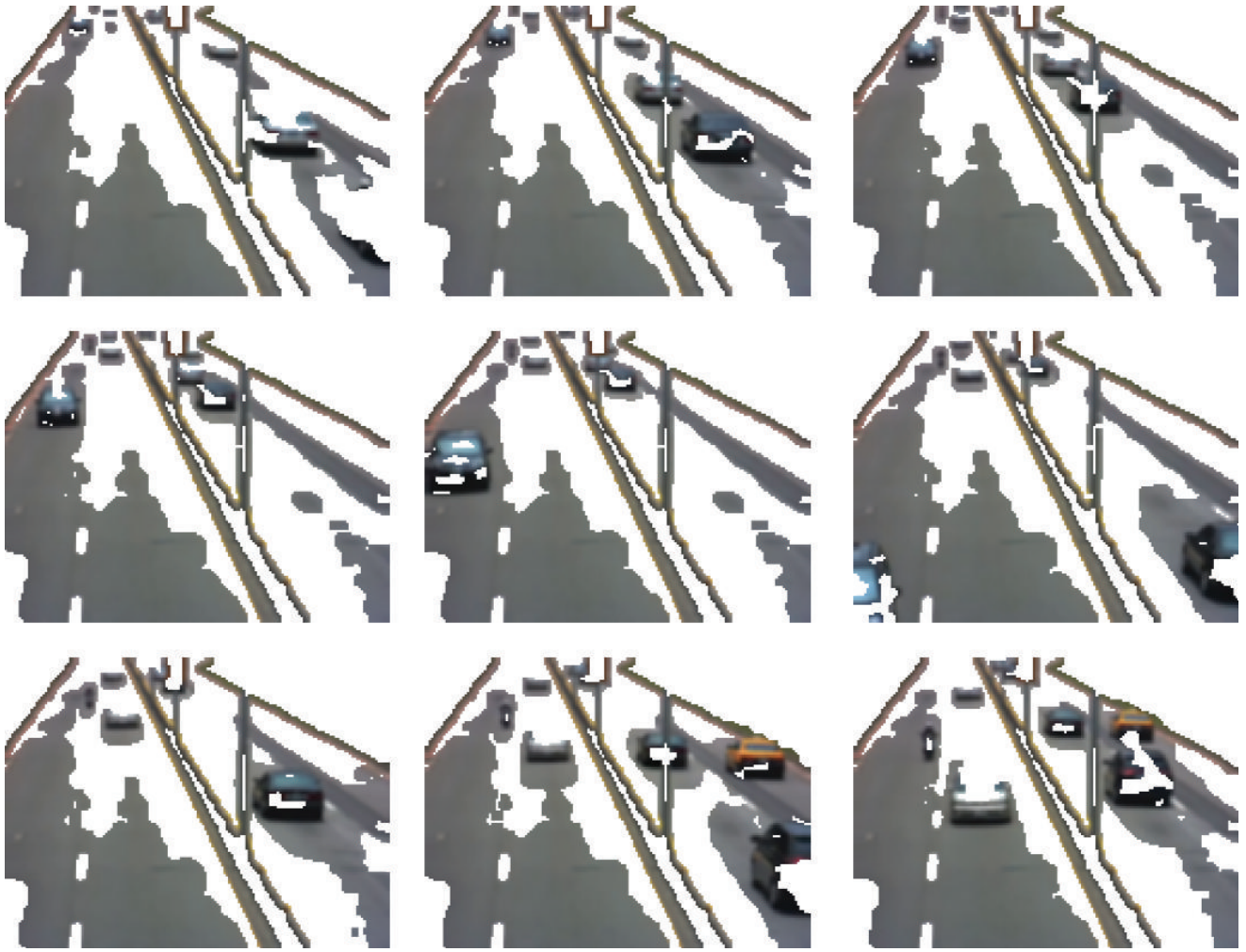


Figure 16: Traffic control video sequence under SOM with 15 neurons: segmentation results for cluster 1.

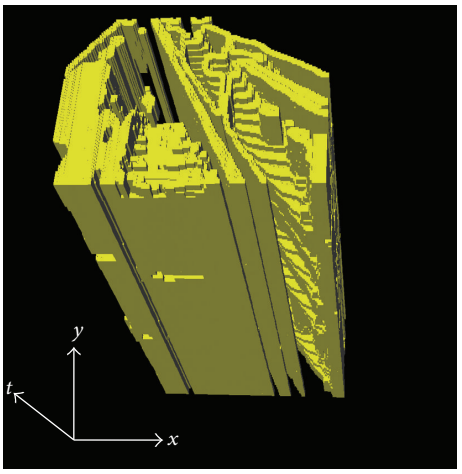


Figure 17: 3D-OPP associated with cluster 1: traffic control video sequence and 15 neurons.

Nevertheless, this cluster also captures the dynamic of the movement of the balls. In Figure 29 the 3D-OPP can be observed for the current cluster. Clearly, we can note the regions of the movement of the billiard balls and the bright regions mentioned before.

5.2.2. Results for 15 Neurons. In Table 4 the sizes of every 3D-EVM are shown associated with each cluster. It can be observed that we have a total of 121,204 extreme vertices. By comparing against the 6D-EVM of the original EVM-animation we obtained a compression rate of 182.14. In Figure 30 the quantity of subanimations is presented grouped in each cluster.

Now we provide a description of the information contained in clusters 1, 2, and 9:

(i) Cluster 1, which has the frame sequence shown in Figure 31, contains regions related to the borders of the elements of the scene. A similar situation has been presented in the previous test (Section 5.2.1). The corresponding 3D-OPP for this cluster is shown in Figure 32, where we can observe the borders of the elements and their evolution across time.

Figure 18: Traffic control video sequence under SOM with 15 neurons: segmentation results for cluster 5.

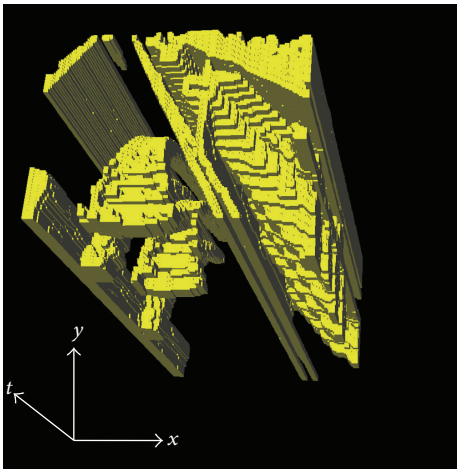


Figure 19: 3D-OPP associated with cluster 5: traffic control video sequence and 15 neurons.

(ii) Clusters 2 and 9 present the case of the patterns of movement of the billiard balls. It is important to

Table 4: Size of the 3D-EVMs generated for every cluster associated with the billiard video sequence under a SOM with 15 neurons.

Class	EVM size
0	17,960
1	10,666
2	6,072
3	4,700
4	1,706
5	7,782
6	634
7	13,838
8	1,338
9	4,064
10	15,570
11	2,798
12	966
13	12,054
14	21,056
Total	**121,204**

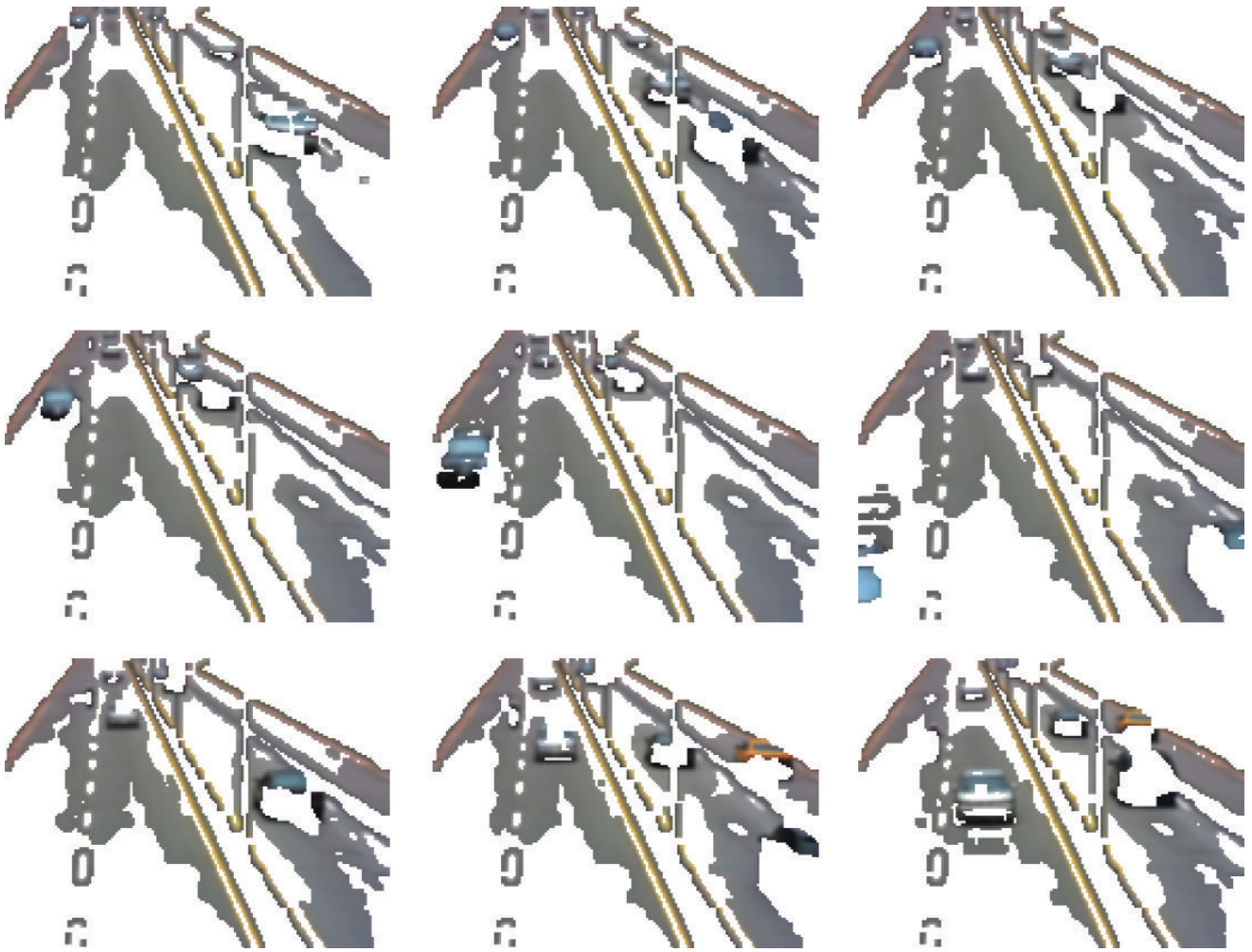


Figure 20: Traffic control video sequence under SOM with 15 neurons: segmentation results for cluster 9.

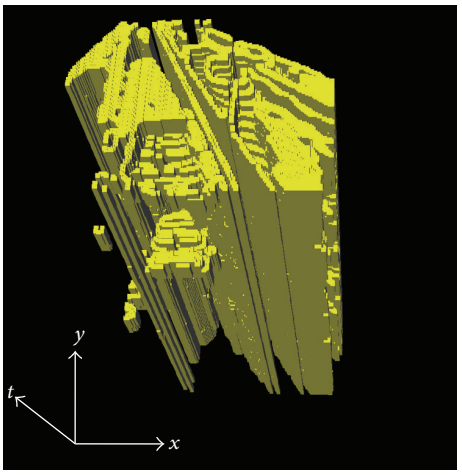


Figure 21: 3D-OPP associated with cluster 9: traffic control video sequence and 15 neurons.

consider that both clusters contain similar information, but cluster 2 contains information about the environment of the scene, that is, elements such as the billiard table. The frame sequences of both clusters are shown in Figures 33 and 35, respectively. Their corresponding 3D-OPPs are shown in Figures 34 and 36. In these OPPs how they grouped the movement of the balls can be seen clearly, but cluster 9 obtains a better isolation of the event. This result has been observed in previous tests, and it is very promising in the sense that the SOM is able to recognize movement patterns across time.

6. Comparisons with Alternative Masks Definitions

In the previous sections we have described the elements that compose our proposed framework and some experimental results obtained when it was applied to two video sequences. As we have previously stated, one of the framework's main elements is the one related to the representation of a video sequence as a higher dimensional OPP. There were considered two geometrical dimensions, one temporal dimension, and one dimension for each color component under the

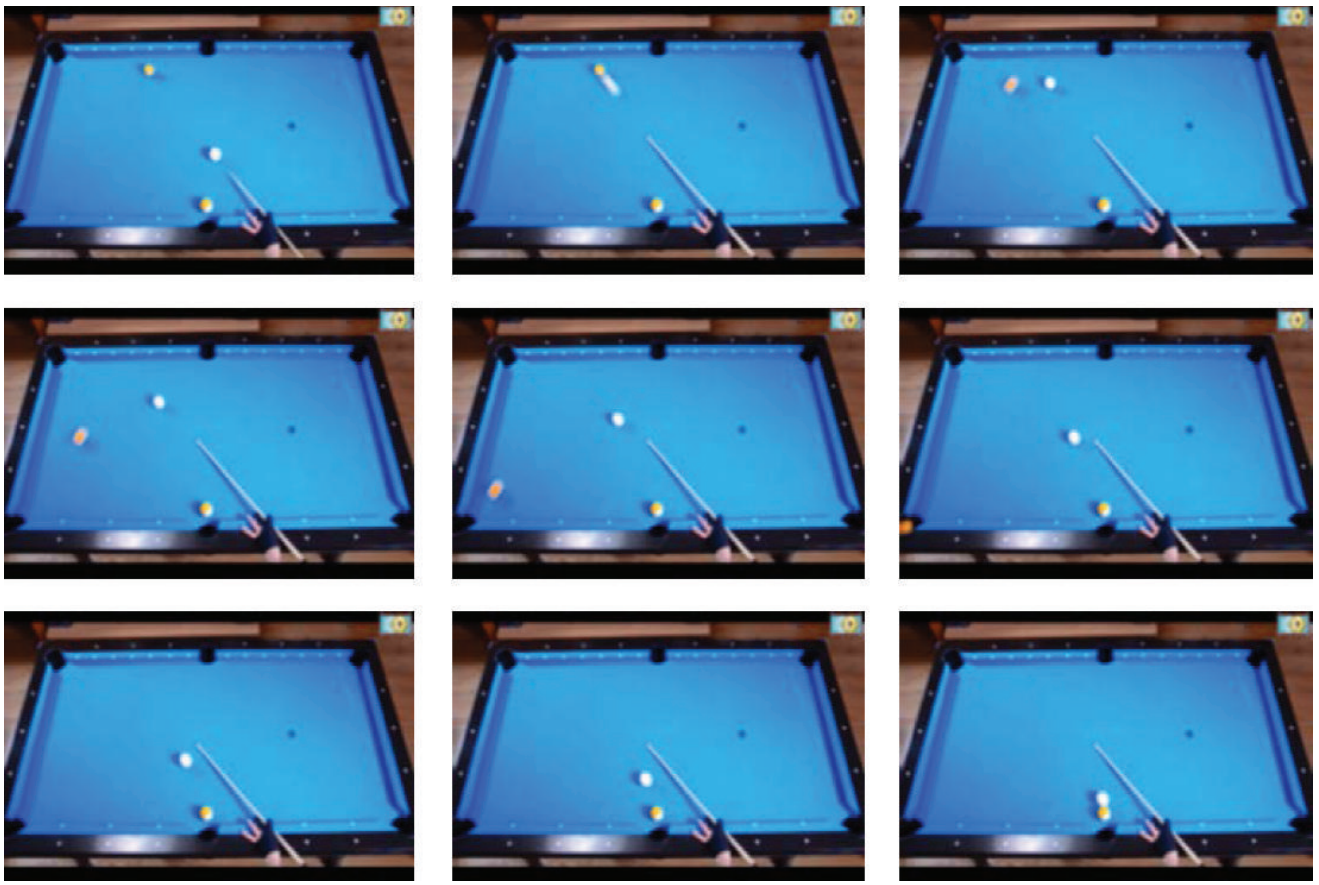


Figure 22: Some frames for the Billiard Game Video Sequence.

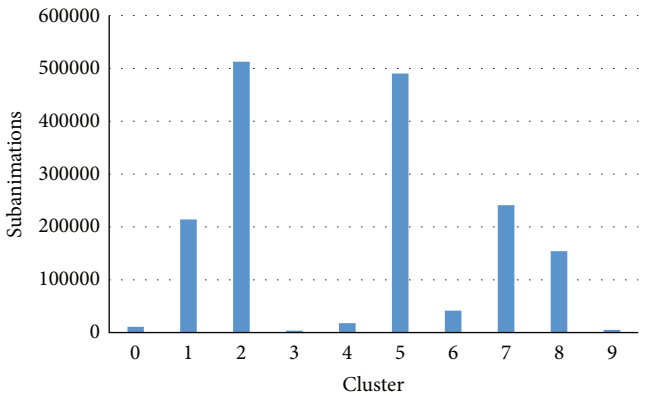


Figure 23: Number of subanimations for each cluster associated with the billiard video sequence under a SOM with 10 neurons.

respective color model. As seen before, our presented study cases were expressed as 6D-OPPs because of the use of the RGB color model.

In this section we are going to present some additional experiments whose objective is the one related to the study of two scenarios.

Scenario 1. Pixels in a frame can be distinguished from pixels from another frame because we have information related to the temporal position, in the video sequence, for these frames. Hence, we could define masks as stated in Section 4 but omitting the temporal component. This implies a mask describes only a region from only one frame, instead of being a subanimation that contains information from several consecutive frames. For this new type of masks its respective DC value can be obtained which in turn is used for training a SOM. Then, a 3D-OPP associated with a cluster, containing a temporal dimension, can also be built by simply recurring to the frame membership of each mask. Taking into account the RGB color model, we have 5D masks (instead of the 6D masks defined in Section 4): two geometrical dimensions and three dimensions associated with each color component. What differences arise, with respect to our proposed framework, when training a SOM without temporal information?

Scenario 2. In Section 1.1 a representation of RGB video sequences was mentioned where the three components' values were integrated as a single integer. Therefore only one dimension is assigned for managing color in a frame's extrusion. One effect that is well identified is the fact that small variations in the integer's higher order bits produce large variations in the whole integer value. Now, we can define masks where the temporal dimension is omitted and only one dimension is assigned to color scale. Again, just as commented above, these masks describe only a region from

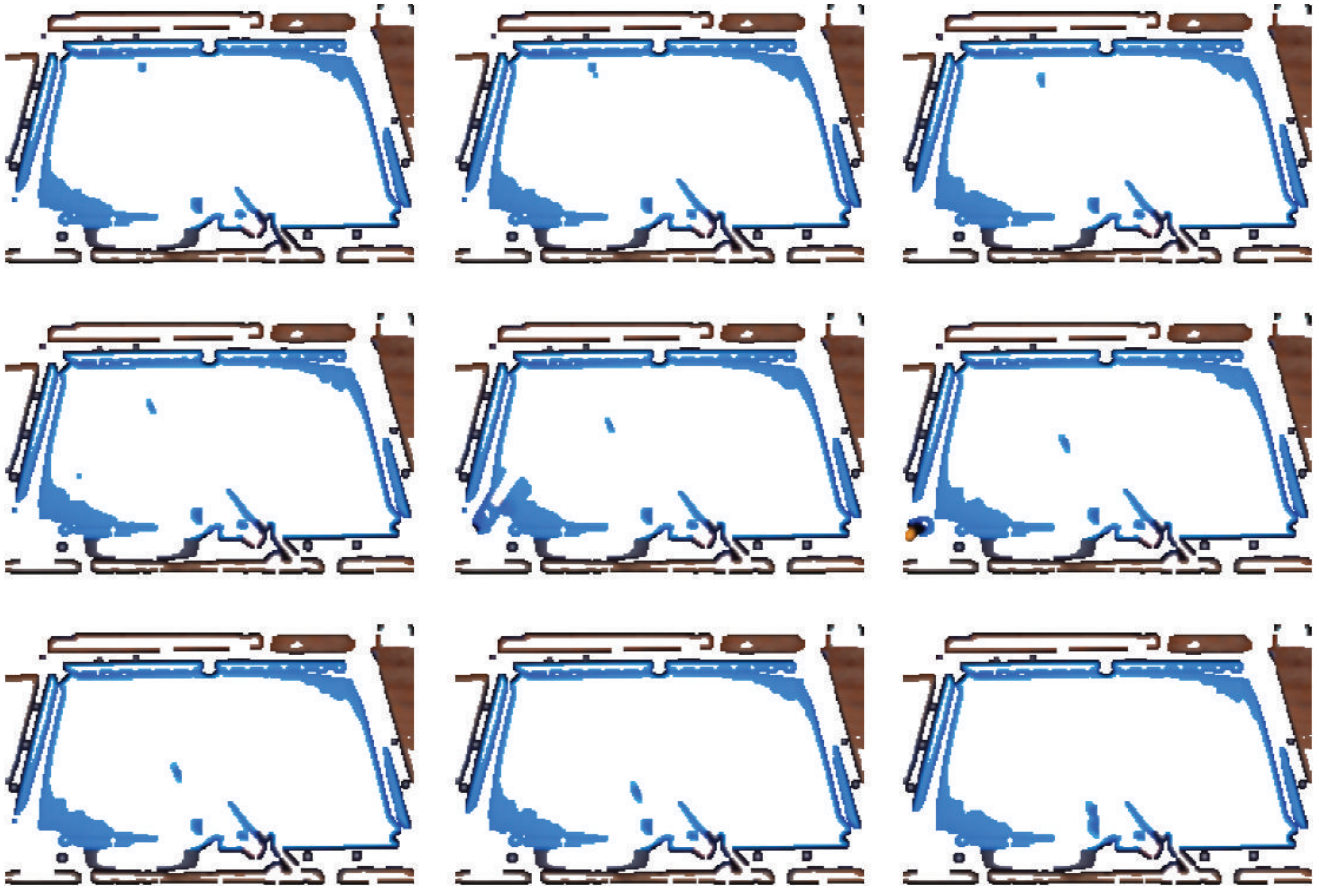


Figure 24: Grouping results for cluster 1: billiard video sequence and 10 neurons.

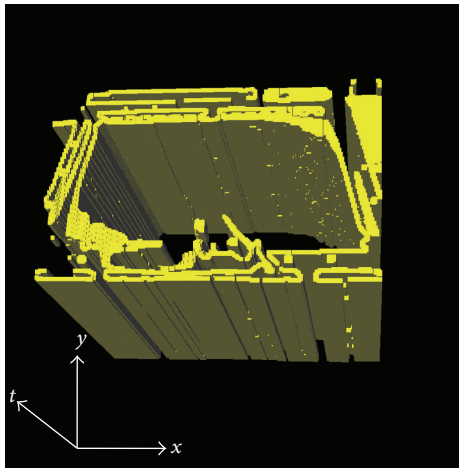


Figure 25: 3D-OPP for cluster 1 of the billiard video sequence and 10 neurons.

only one frame but their respective DC values can be obtained and used for training a SOM. Then, a 3D-OPP associated with a cluster, containing a temporal dimension, is built by recurring to the frame membership of each mask. In this scenario we are dealing with 3D masks: two geometrical dimensions and one dimension related to color scale. What differences arise, with respect to our proposed framework, when training a SOM not only without temporal information but by comprising color components in just one dimension?

The results obtained from the abovementioned pair of scenarios provide us with experimental evidence related to the reason for considering a 6D representation (by using the RGB color scale) for our video sequences, and, for instance, the definition of masks as subanimations taken from this 6D polytope, as stated in Section 4. As seen, these 6D masks provide geometrical, temporal, and color information, with each color component expressed in individual way, to the SOM. For the sake of brevity, we will concentrate only on comparing sizes of the 3D-OPPs obtained in the new experiments with the 3D-OPPs described in Section 5. That is, we are going to compare conciseness of the obtained video segmentations. We will use the same pair of video sequences considered in previous section.

6.1. Scenario 1: Removal of the Temporal Component. Just as presented in Section 5, the two video sequences under consideration are a traffic control video sequence and a Billiard Game Video Sequence. In both cases the configuration of the 5D mask consists in a 2D region of size 3×3, three dimensions for red, green, and blue components, without temporal information. The SOM's training stage executed 200 iterations.

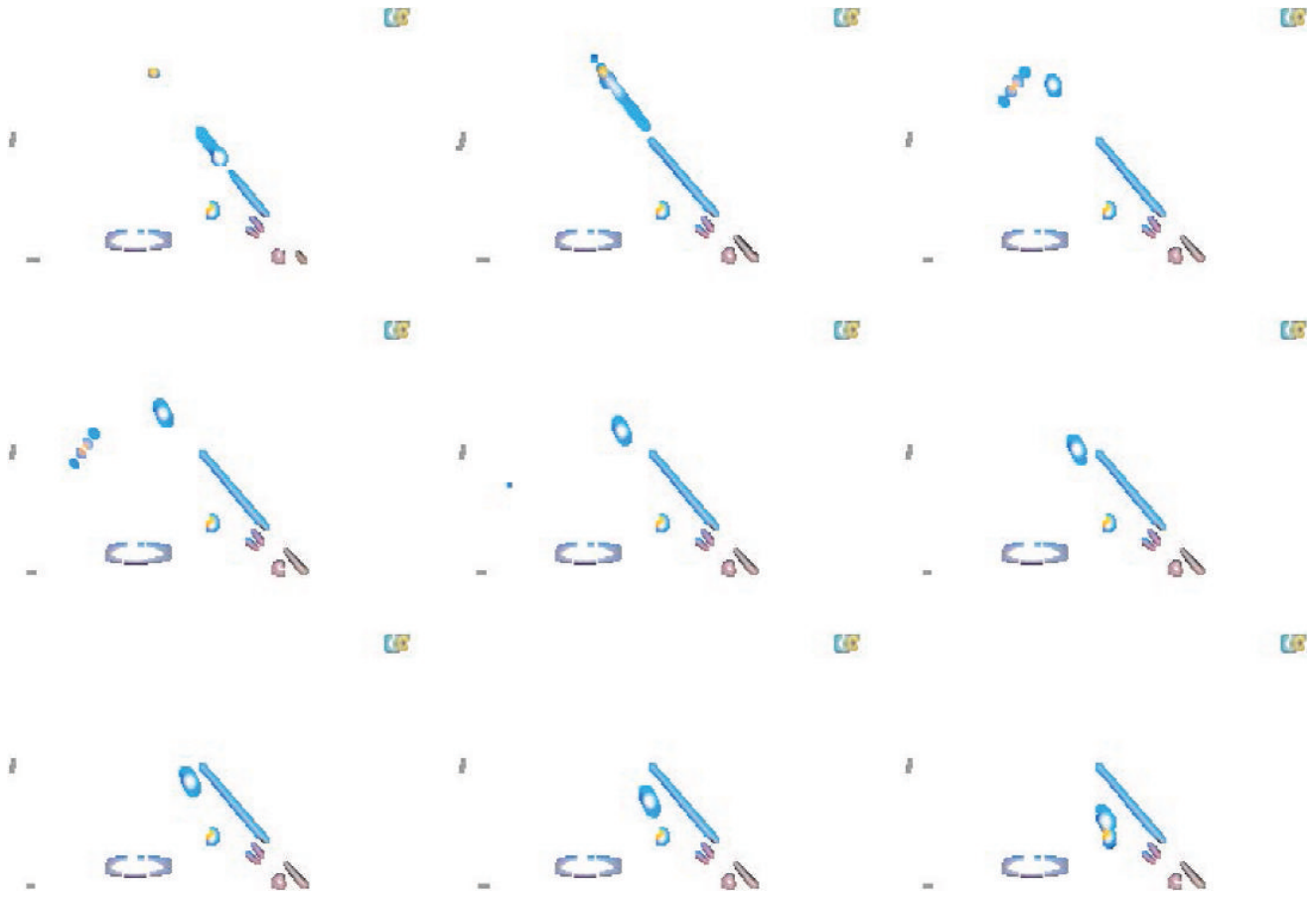


Figure 26: Grouping results for cluster 4: billiard video sequence and 10 neurons.

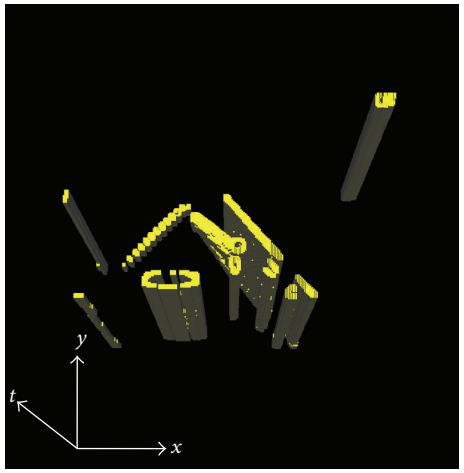


Figure 27: 3D-OPP for cluster 4 of the billiard video sequence and 10 neurons.

6.1.1. Tests for a Traffic Control Video Sequence. The SOM's tested configurations, for this video sequence, were 5 and 15 neurons. Tables 5 and 6 present the obtained classification results. Concerning SOM's configuration with 5 neurons

Table 5: Size of the 3D-EVMs generated for every cluster associated with the traffic control video sequence under a SOM with 5 neurons and removal of the temporal component.

Class	Members	EVM size
0	369,462	28,872
1	211,473	34,846
2	707,381	56,644
3	11,028	5,490
4	5,736	1,410
Total	**1,305,080**	**127,262**

a total of 127,262 extreme vertices were obtained for 3D-EVM representation (Table 5). In Section 5.1.1, for the same SOM configuration and considering 6D masks, a 3D-EVM representation was reported with 80,128 extreme vertices. Table 6 reports results for SOM configuration with 15 neurons where 396,794 extreme vertices are required for 3D-EVM representation. For this same SOM configuration, and by using 6D mask, in Section 5.1.2 we reported that 206,140 extreme vertices were required for 3D-EVM representation. Therefore the removal of the temporal component impacts on

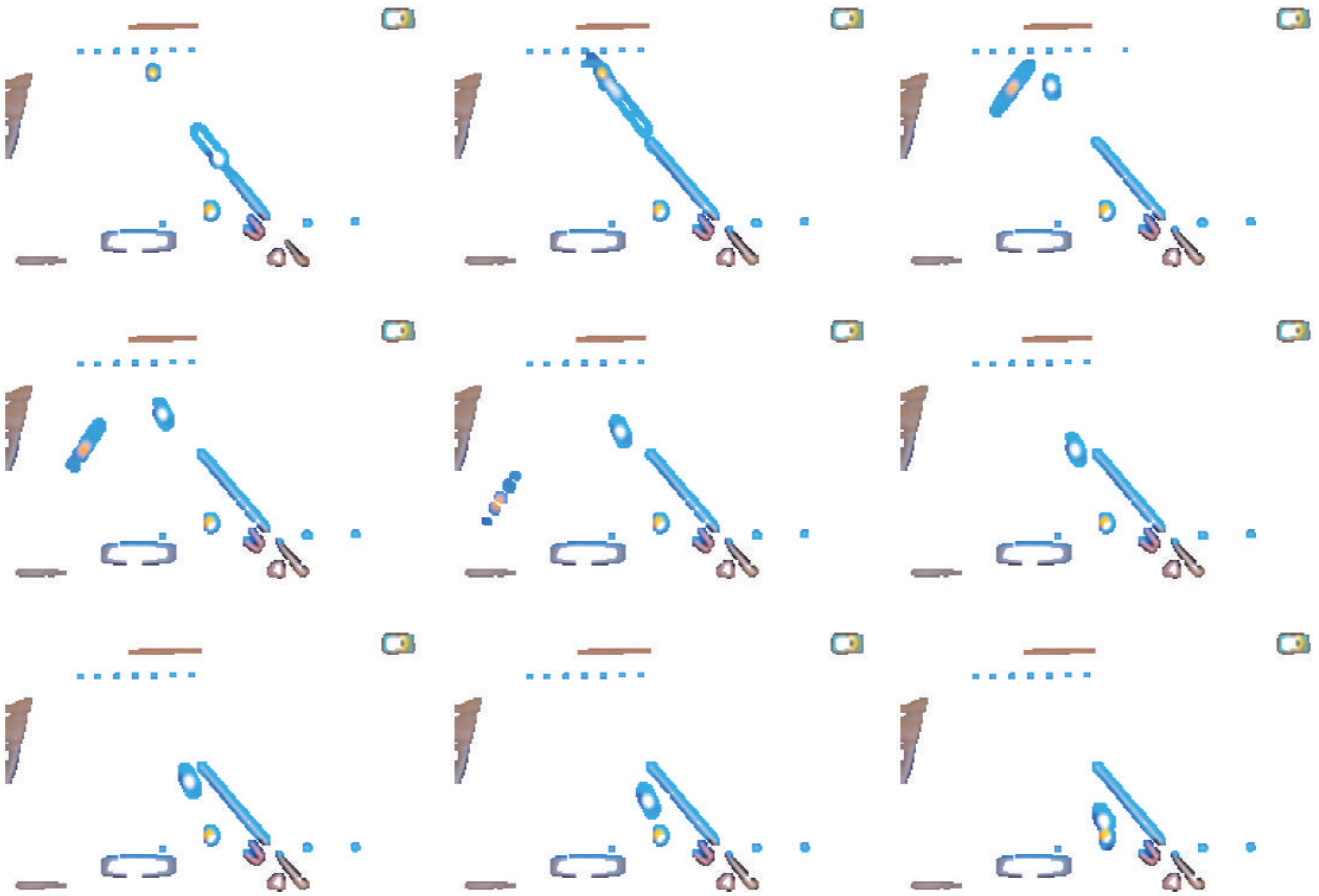


Figure 28: Grouping results for cluster 6: billiard video sequence and 10 neurons.

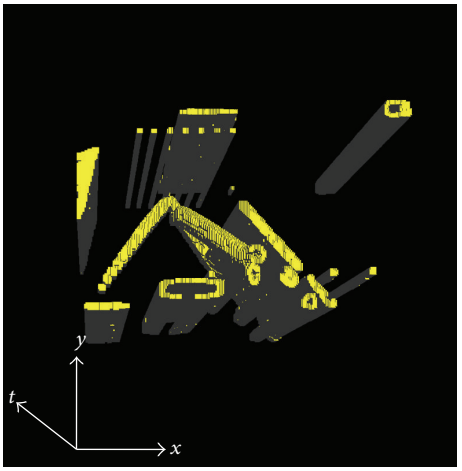


Figure 29: 3D-OPP for cluster 6 of the billiard video sequence and 10 neurons.

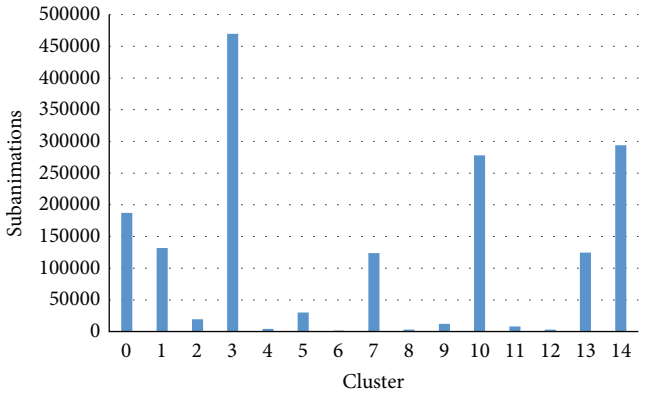


Figure 30: Number of subanimations for each cluster associated with the billiard video sequence under a SOM with 15 neurons.

having EVM representations 1.58 and 1.92 times greater than those presentations achieved by means of using 6D masks.

6.1.2. Tests for a Billiard Game Video Sequence. The SOM's tested configurations, for this case, were 10 and 15 neurons. Tables 7 and 8 present the obtained classification results. Concerning SOM's configuration with 10 neurons 119,420 extreme vertices were required for 3D-EVM representation (Table 7). In Section 5.2.1 a 3D-EVM representation was reported with 76,754 extreme vertices. Table 8 reports results for SOM configuration with 15 neurons where 231,660 extreme vertices are required for 3D-EVM representation. For this same SOM configuration, and by using 6D masks, in Section 5.2.2 we

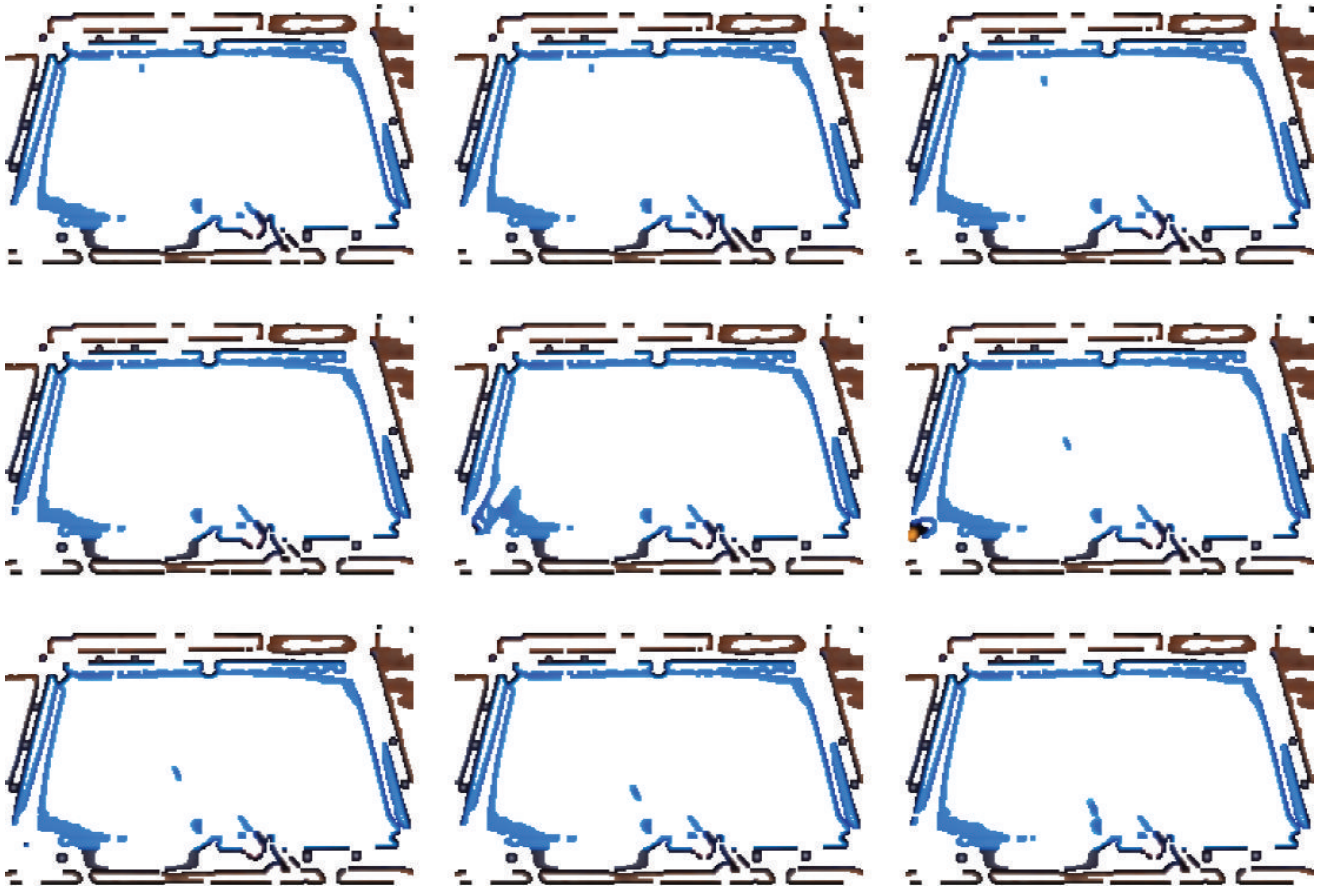


Figure 31: Grouping results for cluster 1: billiard video sequence and 15 neurons.

Table 6: Size of the 3D-EVMs generated for every cluster associated with the traffic control video sequence under a SOM with 15 neurons and removal of the temporal component.

Class	Members	EVM size
0	9,111	8,310
1	5,063	4,658
2	147,723	61,178
3	2,839	1,324
4	83,716	30,748
5	3,131	2,930
6	249,764	62,728
7	143,663	55,420
8	141,288	36,546
9	139,151	18,918
10	2,404	2,192
11	100,026	45,394
12	250,749	50,160
13	924	400
14	25,528	15,888
Total	**1,305,080**	**396,794**

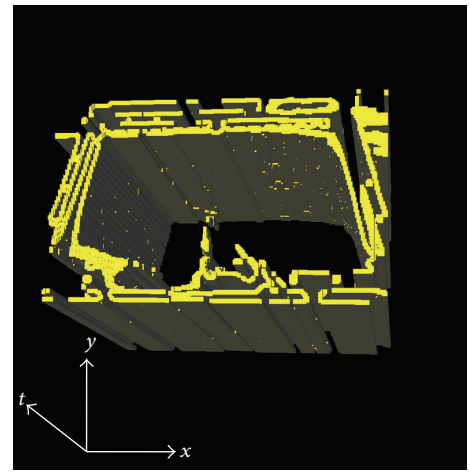


Figure 32: 3D-OPP for cluster 1 of the billiard video sequence and 15 neurons.

reported that 121,204 extreme vertices were required for 3D-EVM representation. Therefore the removal of the temporal component impacts on having alternative representations 1.55 and 1.91 times greater than those presentations achieved by means of using 6D masks.

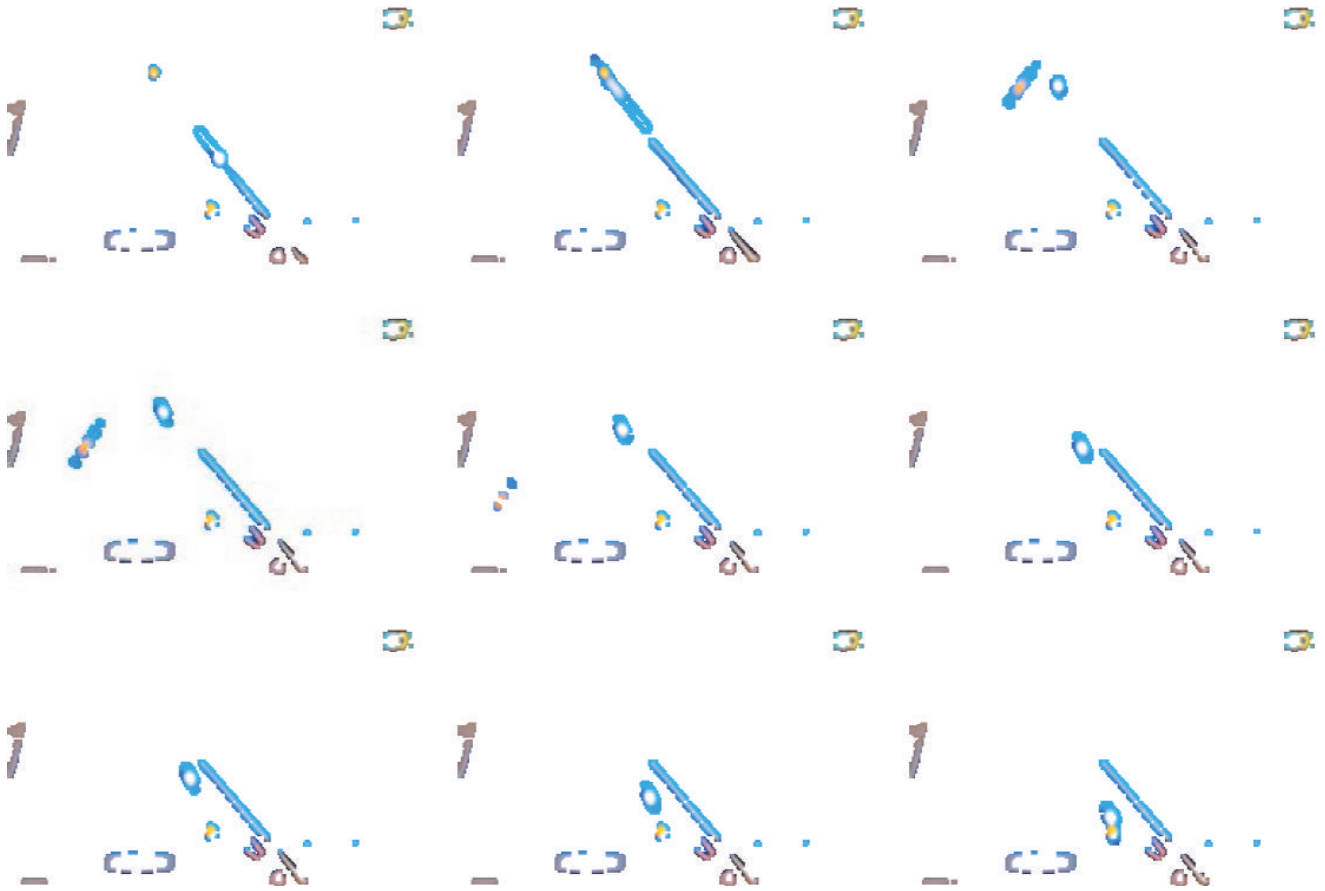


Figure 33: Grouping results for cluster 2: billiard video sequence and 15 neurons.

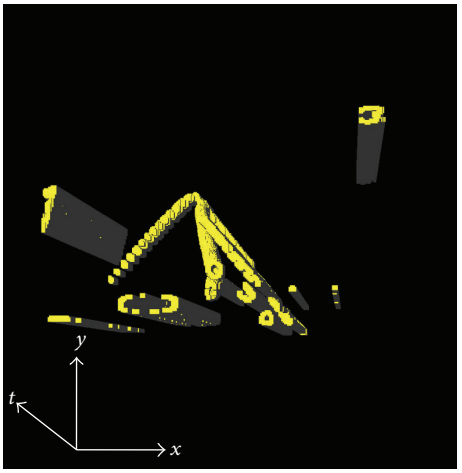


Figure 34: 3D-OPP for cluster 2 of the billiard video sequence and 15 neurons.

Table 7: Size of the 3D-EVMs generated for every cluster associated with the Billiard Game Video Sequence under a SOM with 10 neurons and removal of the temporal component.

Class	Members	EVM size
0	12,795	4,766
1	271,960	23,592
2	43,324	9,662
3	3,144	1,158
4	568,858	25,418
5	588,160	11,790
6	4,142	2,010
7	287,241	31,304
8	6,414	2,832
9	18,954	6,888
Total	**1,804,992**	**119,420**

6.2. Scenario 2: Removal of the Temporal Component and Unification of RGB Components. In both considered video sequences the configuration of the 3D mask consists in a 2D region of size 3 × 3, one dimension integrating as a single integer the red, green, and blue components, without temporal information. The SOM's training stage executed 200 iterations.

6.2.1. Tests for a Traffic Control Video Sequence. The SOM's tested configurations were 5 and 15 neurons. Tables 9 and 10 present the obtained classification results. Concerning SOM's configuration with 5 neurons 168,862 extreme vertices

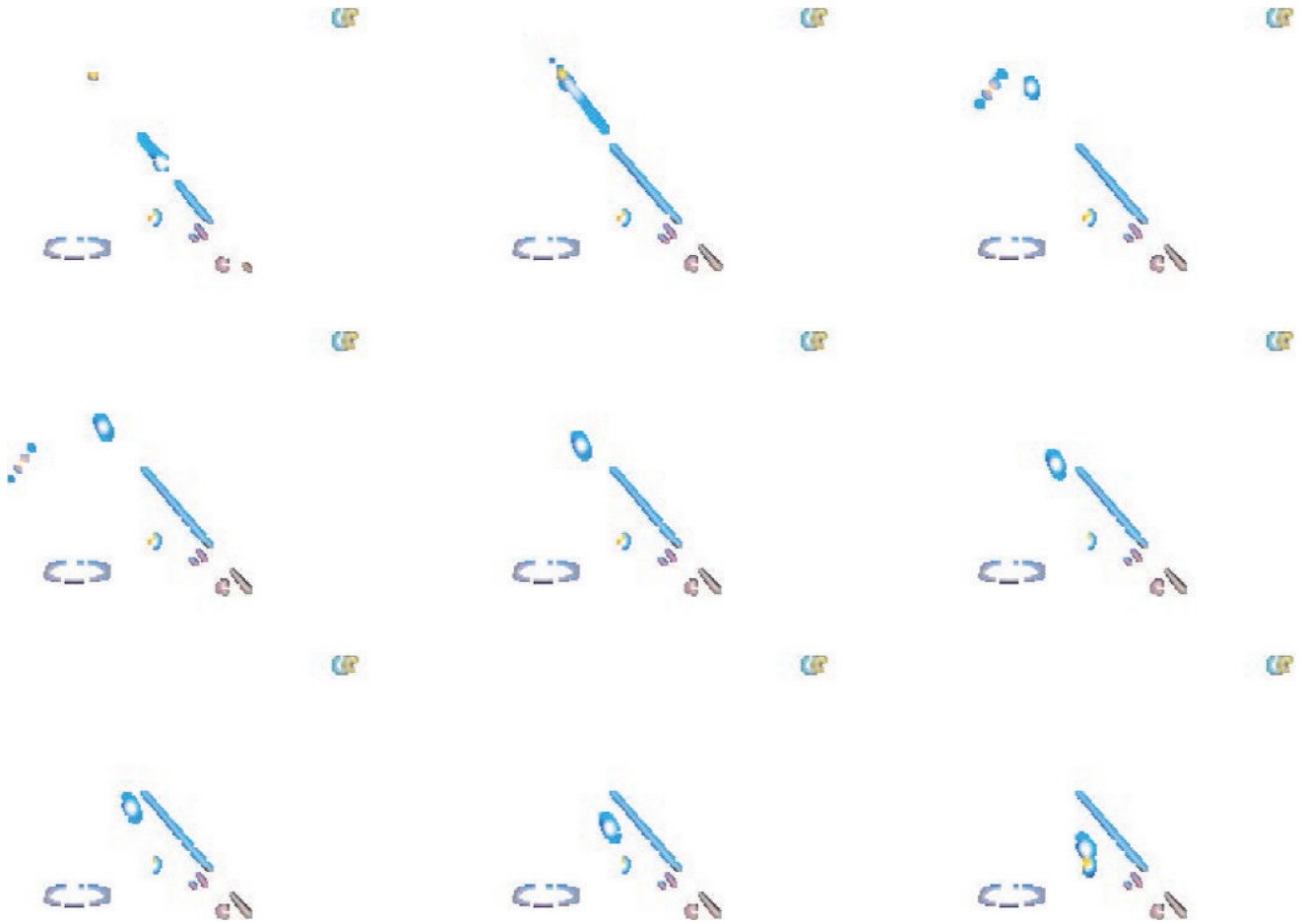


Figure 35: Grouping results for cluster 9: billiard video sequence and 15 neurons.

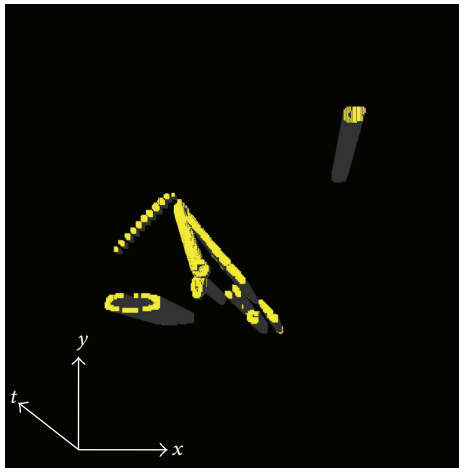


Figure 36: 3D-OPP for cluster 9 of the billiard video sequence and 15 neurons.

were required for 3D-EVM representation (Table 9). In Section 5.1.1 a 3D-EVM representation was reported with 80,128 extreme vertices. Table 10 reports results for SOM configuration with 15 neurons where 459,974 extreme vertices

Table 8: Size of the 3D-EVMs generated for clusters associated with Billiard Game Video Sequence under a SOM with 15 neurons and removal of temporal component.

Class	Members	EVM size
0	244,714	40,280
1	187,126	22,722
2	2,048	930
3	201,427	30,670
4	3,428	2,062
5	210,437	42,532
6	189,185	25,880
7	8,776	3,966
8	11,881	5,808
9	548,273	10,294
10	143,238	25,930
11	29,990	8,946
12	16,003	7,350
13	4,944	2,732
14	3,522	1,558
Total	**1,804,992**	**231,660**

Table 9: Size of the 3D-EVMs generated for every cluster associated with the traffic control video sequence under a SOM with 5 neurons, removal of the temporal component and unification of RGB components.

Class	Members	EVM size
0	236,870	44,890
1	60,893	6,106
2	273,111	39,546
3	628,234	60,900
4	105,972	17,420
Total	**1,305,080**	**168,862**

Table 10: Size of the 3D-EVMs generated for every cluster associated with the traffic control video sequence under a SOM with 15 neurons, removal of the temporal component and unification of RGB components.

Class	Members	EVM size
0	101,659	41,200
1	8,090	1,544
2	75,412	27,546
3	20,303	4,132
4	39,248	19,346
5	100,466	44,756
6	141,671	54,886
7	50,004	26,708
8	32,365	13,120
9	237,961	62,432
10	29,493	8,780
11	25,558	7,496
12	190,047	62,248
13	186,832	52,542
14	65,971	33,238
Total	**1,305,080**	**459,974**

are required for 3D-EVM representation. For this same SOM configuration, and by using 6D masks, in Section 5.1.2 we reported 206,140 extreme vertices were required for 3D-EVM representation. Therefore the removal of the temporal component and the integration of color components in a single dimension impacts in having alternative representations 2.1 and 2.23 times greater than those presentations achieved by means of using 6D masks.

6.2.2. Tests for a Billiard Game Video Sequence. The SOM's tested configurations were 10 and 15 neurons. Tables 11 and 12 present the obtained classification results. Concerning SOM's configuration with 10 neurons there were required 172,376 extreme vertices for 3D-EVM representation (Table 11). In Section 5.2.1 a 3D-EVM representation was reported with 76,754 extreme vertices. Table 12 reports results for SOM configuration with 15 neurons where 280,928 extreme vertices are required for 3D-EVM representation. For this same SOM configuration, and by using 6D masks, in Section 5.2.2 we reported 121,204 extreme vertices were required for 3D-EVM representation. Therefore the removal of the temporal component and the integration of color components in a single dimension impacts on having alternative representations 2.24 and 2.31 times greater than those presentations achieved by means of using 6D masks.

Table 11: Size of the 3D-EVMs generated for every cluster associated with the Billiard Game Video Sequence under a SOM with 10 neurons, removal of the temporal component and unification of RGB components.

Class	Members	EVM size
0	82,064	13,552
1	340,045	45,358
2	250,389	9,012
3	180,390	26,104
4	66,439	8,896
5	49,017	6,072
6	15,538	2,726
7	672,032	37,044
8	73,421	11,524
9	75,657	12,088
Total	**1,804,992**	**172,376**

Table 12: Size of the 3D-EVMs generated for every cluster associated with the Billiard Game Video Sequence under a SOM with 15 neurons, removal of the temporal component and unification of RGB components.

Class	Members	EVM size
0	206,717	8,288
1	44,117	6,288
2	11,638	4,048
3	8,596	2,282
4	47,503	10,082
5	47,525	11,804
6	332,658	58,154
7	381,654	39,536
8	46,984	11,182
9	53,709	8,540
10	116,704	19,198
11	41,066	11,092
12	60,888	13,880
13	249,839	46,318
14	155,394	30,236
Total	**1,804,992**	**280,928**

7. Concluding Remarks and Future Work

According to the results presented in Section 5, many potential applications can be proposed for our proposed framework. At first, it was possible to represent video sequences as a 6D-OPP considering the RGB color scheme, but as we mentioned in Section 4, it is possible to represent videos considering other color schemes, for example, grayscale. This feature of the framework provides versatility in the sense that

it can be adjusted to the application we want to develop and the video sources available.

The video sequences' segmentation was performed by means of extraction of subanimations. For each one of these subanimations its DC descriptor was computed which is a feature that considers the shape and topology of the subanimations. Moreover, upper and lower bounds were proposed in order to compute the DC descriptor: the upper bound is obtained from a subanimation which contains only white frames, while the lower bound is determined from a subanimation that contains only black frames.

To form the training set *DCValues* for the SOM, a convolution was performed between the 6D representation of the original animation and a mask. By this way the convolution is given by the mask's unit shifts over the X_1-, X_2-, and t-axis. An important characteristic of the convolution process' implementation lies in the fact that only some sections from the original animation are required for performing the regularized intersection between the mask and the animation. In that sense, this enhancement represents an improvement with respect to the original Boolean Operations Algorithm under the *n*D-EVM because it is not required to manage all sections from the original animation. Finally, for every cluster a 3D-OPP was formed expressed under the 3D-EVM, which contains only the regions belonging to all the subanimations grouped precisely in the cluster.

In general terms, as seen in experiments from Section 5, the framework we have developed separates the regions in an animation in such way that there are grouped regions that share similar patterns. From the segmentation tests we comment on some observations:

(i) Some clusters group a considerable quantity of information. This is because some video sequences present temporal redundancy. On the other hand, there are some groups that contain a small quantity of information because they tend to group small regions that are different to the rest of the scene.

(ii) There are clusters that contain regions where borders of the elements of the scene are only present.

(iii) In some clusters we observed that they group regions related to the shadows of the objects in the scene. This behavior is very interesting because this could be useful to identify the quantity of objects in the scene; see for example, [52, 53].

(iv) Another interesting behavior that was observed in the performed tests is that the SOM is able to group regions of moving objects and isolate them from the rest of the scene. This result is important because it represents a starting point for identifying movement patterns, and therefore it could be used for other applications.

The experiments performed in Section 6 have provided us with experimental evidence related to the importance for considering the temporal dimension as well as to explicitly separate color components. Although it is possible to define alternative masks whose DC descriptor can be also computed, we have seen how, in first instance, by omitting time as a dimension (Section 6.1) we obtained 3D-OPPs' representations whose sizes range from 1.55 to 1.92 times greater than those presented in Section 5 for the same video sequences. In a second instance, we have observed that by integrating available color components in a single dimension (Section 6.2), and omitting again temporal dimension, the differences increase: we have obtained 3D-OPPs' representations whose sizes range from 2.1 and 2.31 times greater than those obtained by means of our original proposal using 6D subanimations. Summarizing this point, these experiments share us elements for justifying the managing of two geometrical dimensions, one dimension for each color component, and finally, the time axis.

The general conclusion of this work lies in the fact that it was possible to develop a framework for representing and segmenting frame sequences using the *n*D-EVM model and the SOM approach. This fact encourages us to the development of various future research works. One of them considers the generation of segmentation results for video sequences with different color schemes than that used in this work. Experiments are going to be made in order to determine if there is a color scheme that is more suitable for segmentation tasks.

The segmentation results are promising and encouraging by using the DC descriptor. For this reason, we consider that it is important to develop algorithms for computing other shape descriptors for *n*D-OPPs expressed in the EVM. By this way it would be possible to form feature vectors for each subanimation. This would provide us with more elements for better subanimations' comparison. Therefore, it would be possible to improve the performance of the clustering stage. Nevertheless, the challenge here remains in the development of algorithms that can be applied for EVMs for any dimensionality or by defining specific cases.

The segmentation stage using the SOM approach was developed in a modular fashion, which means that other clustering approaches can be used. This provides an opportunity for developing a study and comparison of the clustering performance using different clustering approaches such as [35] *k-means*, *fuzzy c-means*, and *vector quantization*, just to mention a few ones.

Considering that some clusters group regions of the borders of the objects in the scene, we propose developing a classification system using the supervised learning approach. In this approach the goals are to form a training set that contains all the regions of objects borders. Then the training of the classification system will be performed, and at the end of the training stage this system would be able to identify all the regions containing borders for a completely new animation. This idea can also be applied for image processing.

Another line of future research that can be derived from this work consists in using the same approach of supervised learning. But this time we will provide, to the classification system, a training set that contains movement patterns. By this way the classification system will learn how to identify movement in a video sequence. In this sense, it would be possible to train the classification system for recognizing the movement of humans, cars, animals, and so forth and separate them from the rest of the scene.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] K. Ngan and H. Li, *Video Segmentation and Its Applications*, Springer, New York, NY, USA, 2011.

[2] K. N. Ngan and H. Li, "Semantic object segmentation," *IEEE Communications Society Multimedia Communications Technical Committee E-Letter*, vol. 4, no. 6, pp. 6–8, 2009.

[3] H. Wang and C. Schmid, "Action recognition with improved trajectories," in *Proceedings of the 14th IEEE International Conference on Computer Vision (ICCV '13)*, pp. 3551–3558, Sydney, Australia, December 2013.

[4] H. Wang, A. Kläser, C. Schmid, and C.-L. Liu, "Dense trajectories and motion boundary descriptors for action recognition," *International Journal of Computer Vision*, vol. 103, no. 1, pp. 60–79, 2013.

[5] L. Li, W. Huang, I. Y. Gu, and Q. Tian, "Foreground object detection from videos containing complex background," in *Proceedings of the 11th ACM International Conference on Multimedia*, pp. 2–10, ACM, Berkeley, Calif, USA, November 2003.

[6] I. Kokkinos and P. Maragos, "Synergy between object recognition and image segmentation using the expectation-maximization algorithm," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 8, pp. 1486–1501, 2009.

[7] C. Kim and J.-N. Hwang, "Fast and automatic video object segmentation and tracking for content-based applications," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 2, pp. 122–129, 2002.

[8] C. Gentile, O. Camps, and M. Sznaier, "Segmentation for robust tracking in the presence of severe occlusion," *IEEE Transactions on Image Processing*, vol. 13, no. 2, pp. 166–178, 2004.

[9] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: a survey," *ACM Computing Surveys*, vol. 38, no. 4, article 13, 2006.

[10] B. Ko and H. Byun, "FRIP: a region-based image retrieval tool using automatic image segmentation and stepwise boolean and matching," *IEEE Transactions on Multimedia*, vol. 7, no. 1, pp. 105–113, 2005.

[11] V. Mezaris, N. V. Boulgouris, I. Kompatsiaris, and M. G. Strintzis, "Real-time compressed-domain spatiotemporal segmentation and ontologies for video indexing and retrieval," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, no. 5, pp. 606–621, 2004.

[12] C. G. M. Snoek and M. Worring, "Multimodal video indexing: a review of the state-of-the-art," *Multimedia Tools and Applications*, vol. 25, no. 1, pp. 5–35, 2005.

[13] R. Pérez-Aguila, "Computing the discrete compactness of orthogonal pseudo-polytopes via their *n*D-EVM representation," *Mathematical Problems in Engineering*, vol. 2010, Article ID 598910, 28 pages, 2010.

[14] T. Meier and K. N. Ngan, "Video segmentation for content-based coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 9, no. 8, pp. 1190–1203, 1999.

[15] D. Chai and K. N. Ngan, "Face segmentation using skin-color map in videophone applications," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 9, no. 4, pp. 551–564, 1999.

[16] E. Richardson, *H.264 and MPEG-4 Video Compression: Video Coding for Next-Generation Multimedia*, John Wiley & Sons, Hoboken, NJ, USA, 2004.

[17] C. L. Zitnick and S. B. Kang, "Stereo for image-based rendering using image over-segmentation," *International Journal of Computer Vision*, vol. 75, no. 1, pp. 49–65, 2007.

[18] A. Akbarzadeh, J.-M. Frahm, P. Mordohai et al., "Towards urban 3D reconstruction from video," in *Proceedings of the 3rd International Symposium on 3D Data Processing, Visualization, and Transmission*, pp. 1–8, IEEE, Chapel Hill, NC, USA, June 2006.

[19] M. Pollefeys, D. Nistér, J.-M. Frahm et al., "Detailed real-time urban 3D reconstruction from video," *International Journal of Computer Vision*, vol. 78, no. 2-3, pp. 143–167, 2008.

[20] C. Bregler, A. Hertzmann, and H. Biermann, "Recovering non-rigid 3D shape from image streams," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. 690–696, IEEE, Hilton Head Island, SC, USA, 2000.

[21] L.-K. Liu, "Model-based video segmentation for vision-augmented interactive games," in *Image and Video Communications and Processing*, vol. 3974 of *Proceedings of SPIE*, pp. 432–439, International Society for Optics and Photonics, San Jose, Calif, USA, April 2000.

[22] K. S. Fu and J. K. Mui, "A survey on image segmentation," *Pattern Recognition*, vol. 13, no. 1, pp. 3–16, 1981.

[23] Y. Zhang, *Advances in Image and Video Segmentation*, IGI Global Research Collection, IGI Global, 2006.

[24] S. Bhattacharyya and U. Maulik, *Soft Computing for Image and Multimedia Data Processing*, Springer, Berlin, Germany, 2013.

[25] T. B. Moeslund, *Introduction to Video and Image Processing: Building Real Systems and Applications*, Undergraduate Topics in Computer Science, Springer, 2012.

[26] J.-Y. Luis-García, *Creación del framework nd-evm/kohonen para la representación, segmentación y compactación de secuencias de video [Master's Thesis]*, Technological University of the Mixteca (UTM), Huajuapan de León, Mexico, 2015 (Spanish).

[27] S. Vazquez-Reina, S. Avidan, H. Pfister, and E. Miller, "Multiple hypothesis video segmentation from superpixel flows," in *Computer Visio—ECCV 2010*, vol. 6315 of *Lecture Notes in Computer Science*, pp. 268–281, Springer, Berlin, Germany, 2010.

[28] F. Galasso, R. Cipolla, and B. Schiele, "Video segmentation with superpixels," in *Computer Vision—ACCV 2012: 11th Asian Conference on Computer Vision, Daejeon, Korea, November 5–9, 2012, Revised Selected Papers, Part I*, vol. 7724 of *Lecture Notes in Computer Science*, pp. 760–774, Springer, Berlin, Germany, 2012.

[29] A. Khoreva, F. Galasso, M. Hein, and B. Schiele, "Learning must-link constraints for video segmentation based on spectral clustering," in *Pattern Recognition*, X. Jiang, J. Hornegger, and R. Koch, Eds., vol. 8753 of *Lecture Notes in Computer Science*, pp. 701–712, Springer, 2014.

[30] A. Aguilera, *Orthogonal polyhedra: study and application [Ph.D. Dissertation]*, Universitat Politècnica de Catalunya, Barcelona, Spain, 1998.

[31] R. Pérez-Aguila, *Orthogonal polytopes: study and application [Ph.D. thesis]*, Universidad de las Américas, Puebla UDLAP, 2006.

[32] R. Pérez-Aguila, "Representing and visualizing vectorized videos through the extreme vertices model in the n-dimensional space nd-evm," *Journal Research in Computer Science*, vol. 29, pp. 65–80, 2007.

[33] J. R. Parker, *Algorithms for Image Processing and Computer Vision*, John Wiley & Sons, 2010.

[34] M. Nixon, *Feature Extraction and Image Processing*, Elsevier Science, Amsterdam, The Netherlands, 2013.

[35] S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, Elsevier Science, Amsterdam, The Netherlands, 2008.

[36] T. Kohonen, *Self-Organizing Maps*, Physics and Astronomy Online Library, Springer, Berlin, Germany, 2001.

[37] S. Haykin, *Neural Networks and Learning Machines*, Prentice-Hall, Upper Saddle River, NJ, USA, 3rd edition, 2009.

[38] S. Samarasinghe, *Neural Networks for Applied Sciences and Engineering: From Fundamentals to Complex Pattern Recognition*, Taylor & Francis, 2006.

[39] S. American, *Mind and Brain: Readings from Scientific American Magazine*, W. H. Freeman, New York, NY, USA, 1993.

[40] R. Colom, S. Karama, R. E. Jung, and R. J. Haier, "Human intelligence and brain networks," *Dialogues in Clinical Neuroscience*, vol. 12, no. 4, pp. 489–501, 2010.

[41] R. Pérez-Aguila, "Modeling and manipulating 3D datasets through the extreme vertices model in the n-dimensional space (nD-EVM)," *Research in Computer Science*, vol. 31, pp. 15–24, 2007.

[42] R. Pérez-Aguila, "Towards a new approach for modeling volume datasets based on orthogonal polytopes in four-dimensional color space," *Engineering Letters*, vol. 18, no. 4, p. 326, 2010.

[43] M. Spivak, *Calculus on Manifolds: A Modern Approach to Classical Theorems of Advanced Calculus*, Westview Press, Boulder, Colo, USA, 1971.

[44] R. S. Montero and E. Bribiesca, "State of the art of compactness and circularity measures," *International Mathematical Forum*, vol. 4, no. 25–28, pp. 1305–1335, 2009.

[45] S. Marchand-Maillet and Y. M. Sharaiha, *Binary Digital Image Processing: A Discrete Approach*, Academic Press, 1999.

[46] E. Bribiesca, "Measuring 2-D shape compactness using the contact perimeter," *Computers & Mathematics with Applications*, vol. 33, no. 11, pp. 1–9, 1997.

[47] E. Bribiesca, "Measure of compactness for 3D shapes," *Computers& Mathematics with Applications*, vol. 40, no. 10, pp. 1275–1284, 2000.

[48] R. Ruiz-Rodríguez, *Implementación del EVM (extreme vertices model) en java [M.S. thesis]*, Universidad de las Américas, Puebla UDLAP, 2002.

[49] R. R. Rodríguez, "A 3D editor for orthogonal polyhedra based on the extreme vertices model," in *Décimo Congreso Internacional de Investigación en Ciencias Computacionales (CIICC '03)*, vol. 3, Oaxtepec, Mexico, October 2003.

[50] MCC Video, "Nione security megapixel cctv camera nvnd752m-e [traffic]," 2010, https://www.youtube.com/watch?v=ukMFR0IQ3Yc.

[51] PoolShot.org, "Trickshots for beginners #3—bilyar—pool trick shot & artistic billiard training lesson," 2014, https://www.youtube.com/watch?v=vkrfc65vanY.

[52] R. Cucchiara, C. Grana, M. Piccardi, and A. Prati, "Detecting moving objects, ghosts, and shadows in video streams," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 10, pp. 1337–1342, 2003.

[53] A. Prati, I. Mikic, M. M. Trivedi, and R. Cucchiara, "Detecting moving shadows: algorithms and evaluation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 7, pp. 918–923, 2003.