Video Compression and Retrieval of Moving Object Location Applied to Surveillance

William Robson Schwartz[1], Helio Pedrini[2], and Larry S. Davis[1]

[1] University of Maryland, Department of Computer Science
College Park, MD, USA, 20742
[2] Institute of Computing, University of Campinas
Campinas, SP, Brazil, 13084-971

Abstract. A major problem in surveillance systems is the storage requirements for video archival; videos are recorded continuously for long periods of time, resulting in large amounts of data. Therefore, it is essential to apply efficient compression techniques. Additionally, it is useful to be able to index the archived videos based on events. In general, such events are defined by the interaction among moving objects in the scene. Consequently, besides data compression, efficient ways of storing moving objects should be considered. We present a method that exploits both temporal and spatial redundancy of videos captured from static cameras to perform compression and subsequently allows fast retrieval of moving object locations directly from the compressed data. Experimental results show that the approach achieves high compression ratios compared to other existing video compression techniques without significant quality degradation and is fast due to the simplicity of the operations required for compression and decompression.

1 Introduction

Surveillance videos are widely used in domains such as access control in airports, traffic monitoring and human identification. In many of these applications, cameras capture information over long periods of time, resulting in a large amount of data. Such large amounts of data require compression techniques that are not only efficient but also suitable for the domain of surveillance.

Compression techniques take advantage of both spatial and temporal redundancies present in the data. In contrast to spatial redundancy, which has been extensively studied in the image compression field, temporal redundancy has received less consideration. Most of video compression techniques are intended for general purpose videos, for instance when no assumptions about camera motion are made. However, in surveillance applications, videos are usually collected from stationary cameras, resulting in a large amount of temporal redundancy due to high inter-frame correlation. Therefore, suitable techniques can be applied to achieve high compression ratios without losing important information.

Block-based and object-based coding are the main approaches applied to video compression. Within the first group are the video compression techniques such as H.261, H.263, MPEG-1, MPEG-2 and MPEG-4 [1–5]. Most block-based compression approaches reduce temporal and spatial redundancy by considering motion compensation prediction and then applying transform coding to the difference of predicted and actual frames. As block-based techniques do not make assumptions regarding camera motion, temporal redundancy is not fully exploited. Furthermore, it is not a trivial task to retrieve regions of interest from data compressed with such techniques. On the other hand, object-based techniques achieve data compression by separating moving objects from the stationary background and obtaining representations for their shape and motion. Such techniques are more suitable for compressing surveillance videos since they assume that the camera is static.

An object-based video compression system using foreground motion compensation for transmission of surveillance videos was proposed by Babu and Makur [6]. The moving objects are segmented from the background, assumed to be known before hand, by using an edge-based technique. Then, the objects in the current frame are motion compensated according to the previous frame and the resulting error is encoded by a shape adaptive discrete cosine transform. A drawback of such an approach is that it is not robust to incorrect foreground segmentation; therefore, information regarding moving objects might be lost.

Another object-based approach is proposed by Hakeem et at. [7], where object models are learned while compression is performed. Each object is modeled by a few principal components obtained by principal component analysis (PCA). As well as in [6], they assume that the foreground segmentation is given.

Instead of performing compression based on whole objects, Nishi and Fujiyoshi [8] propose a method based on pixel state analysis. Although it is possible to restore the intensity of pixels belonging to moving objects, the location of the whole object is not directly extracted from the compressed data. Additionally, key frames need to be saved every few seconds to adapt to ambient illumination changes. Despite this method takes advantage of the temporal redundancy in both background regions and moving objects by looking at variations over time, the reduction in spatial redundancy is not considered since each pixel is separately encoded.

While Nishi and Fujiyoshi [8] consider variations on pixels, the method proposed by Iglesias et al. [9] represents an entire frame using its projection on the eigenspace computed from a reference frame. In the case of small variations, only a few coefficients need to be stored. However, when major changes take place in the scene, the eigenspace for large regions needs to be updated. They try to overcome this problem by dividing the video into groups of frames and creating an eigenspace for each, assuming small variations within a group of frames.

This paper proposes a method for handling the problems incurring in techniques described previously. Under the assumption that the video is acquired from a static camera, the method is developed based on two key ideas. First, we consider eigenspace representations for non-overlapping blocks in the image so that the method can be robust to random variations and exploit spatial and temporal redundancy. Second, regions containing moving objects are encoded differently from background regions, providing information used to allow efficient retrieval of regions of interest directly from the compressed data.

Two methods are combined for the data compression. For a given frame, we first attempt to model blocks using eigenspaces since they can achieve high compression ratios. However, in case of nonlinear changes, where eigenspaces fail, we apply a second method that groups blocks poorly modeled by the eigenspaces and compress them by using MPEG-4 method.

Jinzenji et al. [10] also attempt to encode stationary and moving regions differently. However, in their approach the video is divided into segments composed of a small fixed number of frames. Then, a model of the stationary regions is built and stored for each one of these segments containing significant amount of stationary regions. This usually leads to lower compression ratios once the background may not be changed during several consecutive segments.

The application of a two-stage technique distinguishes our method from other encoding techniques providing significant reduction in the amount of data needed to reconstruct each frame. Additionally, our method provides the moving object locations directly in the compressed data, which reduces the search space for locating regions of interest, supporting fast computer vision analyses, such as object tracking, object detection and event detection/recognition.

This paper is organized as follows. Section 2 describes the eigenspace representation. In Section 3 we describe the proposed method. Experimental results are shown and discussed in Section 4. Section 5 concludes with some final remarks.

2 Eigenspace Representation

Although pixel-based techniques attempt to reduce temporal redundancy due to correlation between frames, they tend to ignore the correlation of neighboring pixels. Ignoring this neighborhood information results in instability due to random variations present in pixels and poor use of the spatial redundancy. To overcome such problems, region-based approaches may be considered.

Principal component analysis (PCA) is a well-known technique that reduces spatial redundancy of the input data by projecting the data onto a proper basis, called an eigenspace [11]. One of the advantages of PCA over other transforms such as discrete cosine transform (DCT) and wavelets is that the basis depends on the data, which allows a more accurate reconstruction of the original information with fewer coefficients.

Similarly to [7, 9], other works have exploited eigenspaces to perform video coding [12–14]. In general, they either compute eigenspaces for the entire frame and update the model as the image changes, try to model objects in the scene, such as human faces, or code the error in motion prediction. In contrast, we compute eigenspaces for small blocks of the image by sampling a set of frames of the video so that different conditions can be captured. By considering blocks, we reduce the effect of nonlinear changes that take place when the scene is considered as a whole, for example, an illumination change may lead to global nonlinearities, but be locally linear almost everywhere.

A number of methods have been proposed for computing PCA by performing eigenvalue decomposition of the covariance matrix of $\boldsymbol{X}$ [15–17]. Instead, we use an iterative algorithm called NIPALS (Non-linear Iterative PArtial Least Squares) which computes the eigenspace directly from the data matrix $\boldsymbol{X}$ [18].

NIPALS algorithm avoids estimating the covariance matrix, which may be expensive depending on the number of pixels in the block. Computing the eigenspace directly from $\boldsymbol{X}$ can be a good approximation to PCA when only a few principal components are used [19]. According to our experiments, NIPALS is on average ten times faster than PCA to extract the number of components used in our method. This allows us to update the eigenspaces when needed without significant reduction in speed.

3 Proposed Method

This section describes the method proposed for compressing surveillance videos by reducing both spatial and temporal redundancy present in videos and allows the retrieval of moving object locations directly from the compressed data.

We combined two encoding methods for compressing the data. The first method uses eigenspaces to model non-overlapping rectangular blocks of the image. Initially, the eigenspaces are learned from a subset of sampled frames. Then, for each frame, pixels within a block are projected to the corresponding eigenspace and the reprojection error is measured. If the reprojection error is high, meaning that the block is not modeled properly by the eigenspace, a second method is used to encode the block. The second method uses MPEG-4 to compress a set of blocks poorly modeled by the eigenspaces.

Since one of the main goals of surveillance systems is to analyze events taking place over time, in general characterized by the interaction of moving objects, we take advantage of the compression algorithm and encode the location of blocks containing moving objects. Such locations are obtained from the reprojection error of the eigenspaces.

3.1 Learning the Eigenspaces

Before performing data compression, eigenspaces for non-overlapping blocks of the image need to be estimated. Although this step incurs nontrivial computational cost, the contribution of this preprocessing step to the overall time is negligible compared to compressing the entire video, since the proposed method is applied to long duration surveillance videos.

To learn an eigenspace, a set of frames is sampled so that PCA can capture the variation of pixels within the block. However, to obtain a robust estimate of the variations in a block, the sample needs to be free of nonlinear changes such as moving objects. Therefore, the first step to learn the eigenspaces is the removal of undesired frames for a given block.

Assuming that in any large subset of frames there is a certain number of samples free of nonlinear changes and moving objects, we proceed as follows.

First, an eigenspace is computed based on the entire subset and the reconstruction error Δ is measured for each frame. This eigenspace fits the mode of the data, composed of the desired frames, which have small Δ. Then, to estimate which frames need to be removed, we compute the median of the error, m. We also estimate the standard deviation, σ, for those frames with $\Delta < m$. Finally, frames for which $\Delta > m + c\sigma$ are removed, where c is a constant.

Using the median and standard deviation, the estimation of the threshold to define frame removal is robust to large errors from frames containing nonlinear variation and prevents good samples with relatively low Δ from being discarded. Figure 1 shows the reconstruction error for a subset of frames. After applying the described procedure, samples marked in red are removed, most of them containing moving objects inside the block.

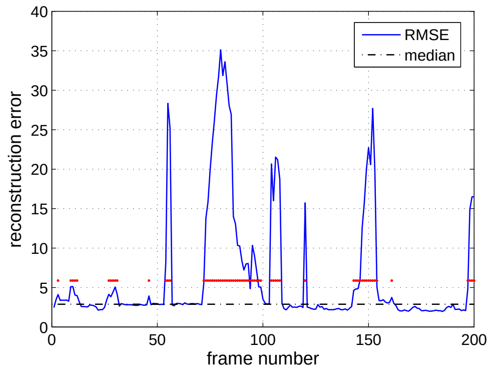


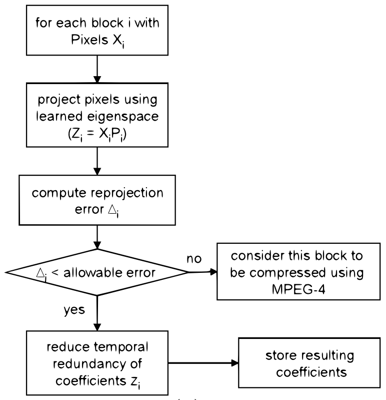
Fig. 1: Reconstruction error for a block. Removed frames are marked with red.

An eigenspace $\boldsymbol{P_i} = \{\boldsymbol{p_1}, \boldsymbol{p_2}, \ldots, \boldsymbol{p_k}\}$ is computed for each block i, where k is defined according to the number of principal components needed, which leads to a trade-off between quality and amount of compression. Additionally, considering the reconstruction error for every pixel p within a block, an error distribution δ_p is estimated, where δ_p is assumed to be normally distributed.

Considering that some changes may take place in the scene over time, we allow for new eigenspaces to be recomputed in order to adapt to the new conditions. For each block, we use Δ and σ to create an error model ω_i, so that we can evaluate if an eigenspace is becoming obsolete for the compression, as will be discussed in the next section.

3.2 Compression Algorithm

As discussed in Section 2, the use of eigenspaces can achieve high compression since only a few principal components are required for each block. However, eigenspaces cannot model nonlinear changes. For this reason, we consider a second method, using MPEG-4 compression, to compress blocks poorly modeled by eigenspaces. The outline of the proposed compression algorithm is shown in Figure 2(a) and its details are described as follows.


for each block i with Pixels X_i
project pixels using learned eigenspace ($Z_i = X_iP_i$)
compute reprojection error Δ_i
Δ_i < allowable error
no
consider this block to be compressed using MPEG-4
yes
reduce temporal redundancy of coefficients Z_i
store resulting coefficients
(a)

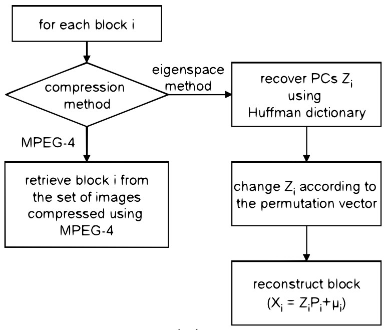

for each block i
compression method
eigenspace method
recover PCs Z_i using Huffman dictionary
MPEG-4
retrieve block i from the set of images compressed using MPEG-4
change Z_i according to the permutation vector
reconstruct block ($X_i = Z_iP_i+\mu_i$)
(b)

Fig. 2: Proposed method. (a) compression algorithm; (b) decompression algorithm.

To compress a video, each frame is decomposed into the same set of non-overlapping blocks for which the eigenspaces have been learned. Then, pixels within a block i are stored in a vector $\boldsymbol{X}_i$ and projected into the eigenspace $\boldsymbol{P_i}$ by $\boldsymbol{Z_i} = \boldsymbol{X_i}\boldsymbol{P_i}$. Finally, using principal components $\boldsymbol{Z_i}$, the mean squared error (MSE), the reprojection error, is computed by Δ_i=$((\boldsymbol{X_i} - \boldsymbol{Z_i}\boldsymbol{P_i}^T)^T(\boldsymbol{X_i} - \boldsymbol{Z_i}\boldsymbol{P_i}^T))/n$, where n denotes the number of pixels in the block.

Blocks resulting in Δ_i smaller than an allowable error are considered well modeled by the eigenspace $\boldsymbol{P_i}$; therefore, we only need to store the coefficients $\boldsymbol{Z_i}$. On the other hand, we need to consider a second method for compressing blocks with Δ_i higher than the allowable error. For each frame, an image is formed by the blocks poorly modeled by the eigenspaces. All the other blocks of this image are assigned to black (to allow high compression ratios). After a number of such images have been processed, the compression method MPEG-4 is applied.

When an eigenspace is not able to model a block for a certain period of time, this may indicate that some change occurred in the region, therefore, a new eigenspace should be estimated, so that the second method, using MPEG-4 compression, could be avoided. A new eigenspace is computed for a block i in case of the reprojection error associated with eigenspace $\boldsymbol{P}_i$ does not satisfy the error model ω_i for a certain number of consecutive frames.

3.3 Exploiting Temporal Redundancy

In this section, we focus only on blocks well modeled by the eigenspaces. As the eigenspaces are not computed often, we are able to reduce both spatial and temporal redundancy once only a few coefficients of the principal components (PCs) need to be stored for a block per frame. In the case of small linear variations within a period of time, the coefficients assume similar values; therefore, further compression may be achieved. First, we convert the principal component coefficients into integers by rounding. According to experimental results, we have seen that this conversion does not increase the reconstruction error significantly.

To achieve extra compression we apply Huffman coding to reduce the number of bits necessary to encode principal component coefficients that appear more frequently. The range and the frequency distribution for the values of each principal component coefficient are estimated during the computation of the eigenspaces. Also, instead of using one Huffman dictionary for each principal component, which would increase significantly the size of the header of the compressed file, we use the same dictionary for all PCs coefficients. To do that, we sort the frequencies in descending order so that the values with highest frequencies are coded using the smallest codewords. Thus, for each PC coefficient, we only need to store a permutation vector that allows us to recover the original ordering and its minimum value.

3.4 Locating Blocks Containing Moving Objects

As discussed earlier, another goal of this work is to support efficient retrieval of the location of moving objects directly from the compressed data. Characteristics of the proposed compression algorithm allow the extraction of object locations without significant overhead.

First, if a block can be modeled by the eigenspace, then we assume that there are no moving objects within that block, otherwise the reprojection error would be high due to nonlinearity of the changes, and thereby that block would be compressed by the second method. Therefore, we need to look for moving objects only in blocks compressed by the MPEG-4 method.

Second, once the projection error for each block is computed by the compression algorithm and we have the error distribution for each pixel, δ_p (estimated during the computation of the eigenspaces), it is possible to determine if the error is consistent with the distribution for δ_p (in such a case it is due to inherent variations in the pixel), or it is due to unexpected changes. In the latter case, the changes may be caused either by noise not captured in δ_p or by moving objects. Looking at the pixel's neighborhood helps to determine which is the case since the spatial distribution of noise tends to be spread and the spatial distribution of an object tends to be more compact.

We now present the algorithm used to locate blocks containing moving objects. For a given frame, consider only blocks compressed by the second method. For each block, find pixels p such that the reconstruction error have low probability of belonging to error distribution δ_p. Create a binary matrix the same size of the block and set the location of all such pixels p to 1. Apply the median filter to this matrix, then if the number of 1 entries has not been significantly reduced, mark the block as containing moving objects. The number of pixels presenting value 1 reduces when values of 1 are spread, which does not characterize the presence of an object.

3.5 Data Storage and Decompression Algorithm

We divide the data storage into two categories: a header section that stores data resulting from one time computation and a data section that stores data obtained from the compression of each frame.

For each block i, the header section keeps the block size, the eigenspace $\boldsymbol{P_i}$, the mean vector $\boldsymbol{\mu_i}$ of its pixels (required since PCA is computed from a mean centered matrix), and the permutation vectors. Besides that, the header also stores the dictionaries used by Huffman coding. The size of this section does not change over time.

For each block contained in a frame, the data section stores either the coefficient encoded with PCs or the result from the MPEG-based method according to the encoding method used for compressing the block, and two Boolean variables, the first to indicate which encoding method was used and the second to indicate if the block contains moving objects. Additionally, the data section stores new eigenspaces created over time.

Figure 2(b) shows the steps of the decompression algorithm for each frame. Locations of moving objects can be extracted efficiently by looking directly at the second Boolean variable associated with each block.

4 Experimental Results

In this section, we show results and comparisons among the proposed approach and three standard techniques used for video compression. The results were obtained using an Intel Core 2 T200 with 2 Gbytes of RAM memory and running Windows XP operating system.

To use the proposed method for compressing surveillance videos, we first convert the color space to YCbCr, format commonly used for data compression, where Y is the luma and Cb and Cr contain blue and red chroma components. Then, eigenspaces are computed for blocks of 16×16 pixels for each color band, considering 200 frames sampled from the video. The number of coefficients kept after performing PCA is estimated for each block according to either a target reconstruction error or a maximum number, if the target error cannot be reached.

We have chosen blocks of 16×16 pixels because we noticed that for larger blocks, such as 64×64 and 128×128, the number of times we needed to use MPEG4 due to changes in small areas of the region is higher resulting in smaller compression ratios. As a result, we can see that blockwise approach works better than whole-frame PCA solutions because when a small region of the image changes, in the whole-frame approach, the eigenspace for the entire image needs to be updated or rebuilt, reducing the compression ratio significantly. However, in the blockwise approach, only changing regions need to be tackled either by using MPEG4 for compression or updating the eigenspace.

The measurement of reconstruction quality is the peak signal-to-noise ratio (PSNR) between the original frames and the reconstructed ones in RGB space for each color band separately, such that the average is used as the resulting PSNR. A commonly used approach to comparing different compression methods is either to fix the PSNR and assess the compression ratio (CR) or, on the opposite, having a fixed compression ratio, and then measuring the reconstruction quality using PSNR.

We compared our results on several video sequences with different standard video compression techniques. Due to the lack of standard datasets to compare video compression methods that consider static cameras, most of our videos are well-known sequences widely used by the surveillance community. One reconstructed frame of each sequence is shown in Figure 3. We compared our method to MPEG-2, MPEG-4 and H.263 using the default set of parameters provided by MEncoder [20], program used to compress the data. In addition, we considered the same set of parameters used by MPEG-4 in the second stage of our method.

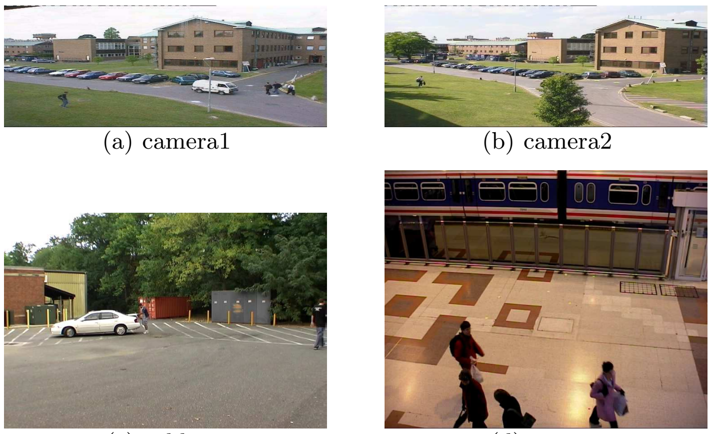

(a) camera1 (b) camera2 (c) robbery (d) station

Fig. 3: Reconstructed frames from the video sequences used in the experiments.

Table 1 shows the results obtained by using the proposed method and other video compression techniques, where a target PSNR was fixed to compare the compression ratios. As can be observed in the table, our method achieves high compression ratios. This is due to the fact that the use of eigenspaces provides higher compression ratios since only few PC coefficients need to be stored for each block. The results obtained for MPEG-2 are not shown in the table since it did not meet the target PSNR.

video sequence	frame size (pixels)	frames	PSNR (dB)	MPEG-4	H.263	proposed
camera1	768×288	2695	39.00	33.78	34.33	**37.27**
camera2	768×288	5333	38.00	40.44	40.27	**45.06**
robbery	720×480	3320	38.00	34.70	34.66	**61.14**
station	720×576	2370	39.50	51.27	49.94	**105.64**

Table 1: Compression ratios obtained using the proposed method and other video compression techniques for a given PSNR.

We observed that the application of Huffman coding over PC coefficients reduces, on average, to one byte per coefficient, instead of its original size of four bytes, since it is a floating-point number.

Our unoptimized MATLAB code can compress video sequences listed in Table 1 at 5.2 frames/second, on average. This running time can be substantially improved since most of operations required during the processing are vector multiplications and the MPEG-4 used to compress blocks poorly modeled by the eigenspaces runs at high frame rates.

In addition to the compression, the approach locates regions containing moving objects, which is not available in standard compression techniques. For each frame, blocks with moving objects are located as described in Section 3.4 and encoded in a bitmap. Each entry of the bitmap contains the value of the second Boolean variable presented in Section 3.5. This way, the retrieval of the object location can be done quickly by indexing such bitmaps, saving processing time of subsequent processing stages, such as object recognition or object tracking. Figure 4 shows a frame of the *camera2* sequence where blocks containing moving objects are marked.

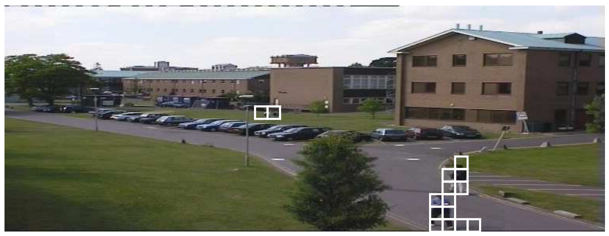


Fig. 4: Location of moving objects.

To evaluate if object locations were correctly encoded, handed-adjusted results from background subtraction are used as ground truth of moving object location for sequence *camera2*. We evaluate the false positive rate (FPr) and the false negative rate (FNr), considering that a block was correctly added to the bitmap if it contains pixels belonging to objects. As results we have obtained FPr=0.025 and FNr=0.051. This means that 5.1% of moving object locations have not been encoded in the bitmap, mainly due to objects having only few pixels in a block. Also, 2.5% of the background regions were added into the bitmap, mainly due to a waving tree present in the scene.

5 Conclusions

In this work, we have described a compression technique applied to surveillance videos. Besides achieving high compression ratios as shown in the experimental results, this technique provides a useful feature that can be used in further video processing, a mapping that locates moving objects in the scene is readily available in the compressed data.

Acknowledgements

W.R. Schwartz acknowledges “Coordenação de Aperfeiçoamento de Pessoal de Nível Superior” (CAPES - Brazil, grant BEX1673/04-1). The authors are also grateful to CNPq and FAPESP for their financial support.

References

1. ISO/IEC 11172-2 Information Technology: Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to 1.5 Mbits/s. Part 2 (1993)
2. ISO/IEC 13818-2 Information Technology: Generic Coding of Moving Pictures and Associated Audio Information. Part 2: Video (2000)
3. ISO/IEC 14496-2 Information Technology: Coding of Audio-Visual Objects. Part 2: Visual (2001)
4. ITU-T Recommendation H.261: Video Codec for Audiovisual Services at px64 kbit/s (1990) Geneve.
5. ITU-T Recommendation H.263: Video Coding for Low Bitrate Communication. Version 2 (1998) Geneve.
6. Babu, R., Makur, A.: Object-based Surveillance Video Compression using Foreground Motion Compensation. In: 9th International Conference on Control, Automation, Robotics and Vision. (2006) 1–6
7. Hakeem, A., Shafique, K., Shah, M.: An Object-based Video Coding Framework for Video Sequences Obtained from Static Cameras. In: Proceedings of the 13th Annual ACM International Conference on Multimedia, New York, NY, USA, ACM (2005) 608–617
8. Nishi, T., Fujiyoshi, H.: Object-based Video Coding using Pixel State Analysis. In: Proceedings of the 17th International Conference on Pattern Recognition. Volume 3. (2004) 306–309
9. Perez-Iglesias, H., Dapena, A., Castedo, L.: A Novel Video Coding Scheme based on Principal Component Analysis. IEEE Workshop on Machine Learning for Signal Processing (2005) 361–366
10. Jinzenji, K., Okada, S., Kobayashi, N., Watanabe, H.: MPEG-4 Very Low Bitrate Video Compression by Adaptively Utilizing Sprite to Short Sequences. In: Multimedia and Expo, 2002. ICME '02. Proceedings. 2002 IEEE International Conference on. Volume 1. (2002) 653–656
11. Jolliffe, I.: Principal Component Analysis. Springer, New York, NY, USA (2002)
12. Liu, J., Wu, F., Yao, L., Zhuang, Y.: A Prediction Error Compression Method with Tensor-PCA in Video Coding. In: Multimedia Content Analysis and Mining. Volume 4577. Springer (2007) 493–500
13. Torres, L., Prado, D.: A Proposal for High Compression of Faces in Video Sequences using Adaptive Eigenspaces. In: Proceedings of International Conference on Image Processing. Volume 1. (2002) I–189–I–192
14. Yao, L., Liu, J., Wu, J.: An Approach to the Compression of Residual Data with GPCA in Video Coding. In: Advances in Multimedia Information Processing - PCM 2006. Volume 4261. Springer (2006) 252–261
15. Golub, G.H., Loan, C.F.V.: Matrix Computations. third edn. Johns Hopkins Press, Baltimore, MD, USA (1996)
16. Roweis, S.: EM algorithms for PCA and SPCA. In: Advances in Neural Information Processing Systems. Volume 10., Cambridge, MA, USA, MIT Press (1998) 626–632
17. Sharma, A., Paliwal, K.K.: Fast principal component analysis using fixed-point algorithm. Pattern Recognition Letters **28** (2007) 1151–1155
18. Wold, H.: Estimation of Principal Components and Related Models by Iterative Least Squares. In Krishnaiah, P.R., ed.: Multivariate Analysis. Academic Press (1966)
19. Martens, H., Naes, T.: Multivatiate Calibration. John Wiley, Chichester, Great Britain (1989)
20. MPlayer: The Movie Player (2009) http://www.mplayerhq.hu/.