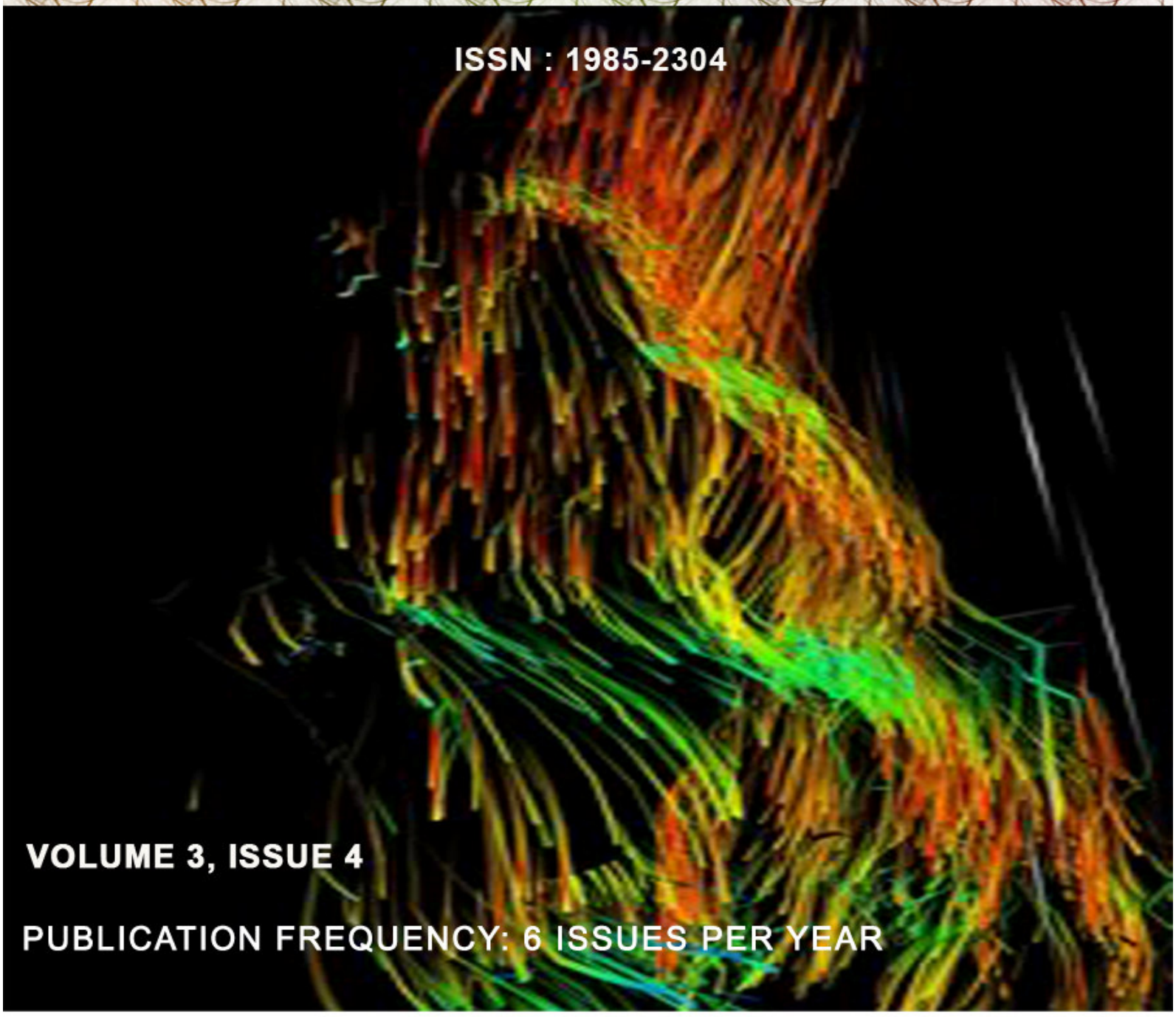
International Journal of Image Processing (IJIP)

ISSN : 1985-2304



VOLUME 3, ISSUE 4

PUBLICATION FREQUENCY: 6 ISSUES PER YEAR

Editor in Chief Professor Hu, Yu-Chen

International Journal of Image Processing (IJIP)

Book: 2009 Volume 3, Issue 4
Publishing Date: 31 - 08 - 2009
Proceedings
ISSN (Online): 1985 - 2304

IJIP Journal is a part of CSC Publishers
http://www.cscjournals.org

Published in Malaysia

Typesetting: Camera-ready by author, data conversation by CSC Publishing Services – CSC Journals, Malaysia

CSC Publishers

Table of Contents

Volume 3, Issue 4, August 2009.

Pages

A Framework for Soccer Video Processing and Analysis Based on Enhanced Algorithm for Dominant Color Extraction

Youness TABII youness.tabii@gmail.com
ENSIAS/GL
BP 713, Mohammed V University-Soussi
Rabat, 10000, Morocco

Rachid OULAD HAJ THAMI oulad@ensias.ma
ENSIAS/GL
BP 713, Mohammed V University-Souissi
Rabat, 10000, Morocco

Abstract

Video contents retrieval and semantics research attract a large number of researchers in video processing and analysis domain. The researchers try to propose structure or frameworks to extract the content of the video that's integrating many algorithms using low and high level features. To improve the efficiency, the system has to consider user behavior as well as develops a low complexity framework. In this paper we present a framework for automatic soccer video summaries and highlights extraction using audio/video features and an enhanced generic algorithm for dominant color extraction. Our framework consists of stages shown in Figure 1. Experimental results demonstrate the effectiveness and efficiency of the proposed framework.

Keywords: Video processing, Summary and Highlight, Finite state machine, Binarization, Text detection, OCR.

1. INTRODUCTION

Soccer video attracts a wide range of audiences and it is generally broadcasted for long hours. For most non-sport viewers and some sport fans, the most important requirements are to compress its long sequence into a more compact representation through a summarization process. This summarization has been popularly regarded as a good approach to the content-based representation of videos. It abstracts the entirety with the gist without losing the essential content of the original video and also facilitates efficient content-based access to the desired content.

In literature, many works have been proposed in the last decade concern sport videos processing and analysis. The sport videos are composed of play events, which refer to the times when the ball is in-play, and break events, which refer to intervals of stoppages in the game. In [1], the play-break was detected by thresholding the duration of the time interval between consecutive long shots; Xu et al. [2] segmented the soccer game into play-break by classifying the respective visual patterns, such as shot type and camera motion, during play, break and play/break transition.

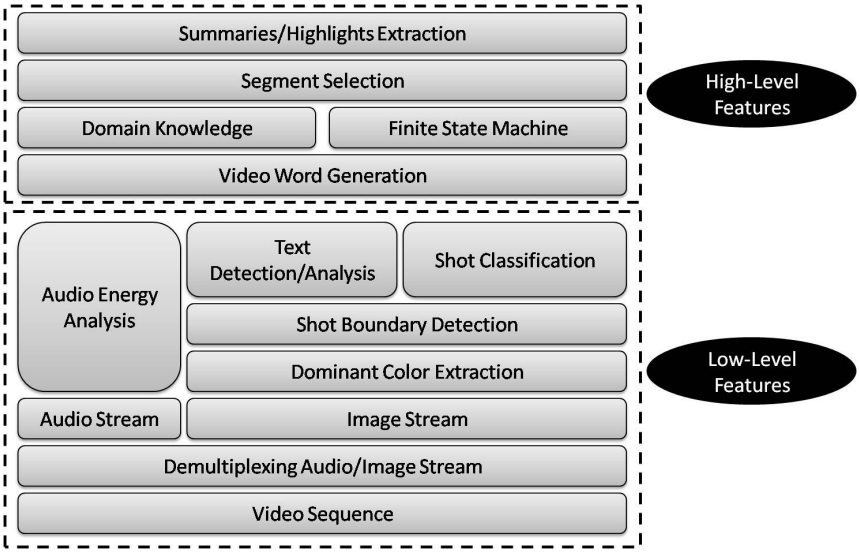


FIGURE 1: The proposed framework.

The highlight detection research aims to automatically extract the most important, interesting segments. In [3] introduced a generic game highlight detection method based on exciting segment detection. The exciting segment was detected using empirically selected cinematic features from different sports domains and weighted combine to obtain an excitement level indicator. Tjondronegoro et al. [4] detected the whistle sounds, crowd excitement, and text boxes to complement previous play-breaks and highlights localization method to generate more complete and generic sports video summarization.

Replay scenes in broadcast sports videos are excellent indicators of semantically mportant segments. Hence, replay scene detection is useful for many sports highlight generation applications. The characteristic of replay scenes are: 1) usually played in a slow-motion manner. Such slow-motion effect is produced or by repeating frames of the original video sequence. 2) Playing the sequence captured from a high- speed camera at the normal frame rate. Replay detection techniques have been proposed. In [5] used Bayesian Network together with six textual features extracted from Closed Caption to detect replay shot. Pan et al. [6] proposed to detect the transition effect, e.g. flying-logo, before and after the replay segments to detect replay. In [7] introduced a method to automatically discover the flying-logo for replay detection.

In the next of this paper, we will present each block/algorithm in our framework. We begin by representing the enhanced algorithm for dominant color extraction. Second, the shot boundary detection. Third, shots classification into defined classes. Fourth, the extraction of the audio descriptor. Fifth, the score box detection and text recognition, and as the final step, with domain knowledge we use finite state machine to extract the summaries and highlights.

2. DOMINANT COLOR EXTRACTION

The field region in many sports can be described by a single dominant color. This dominant color demonstrates variations from one sport to another, from one stadium to another, and even within one stadium during a sporting event.

In this section we will present our enhanced algorithm for dominant color extraction in soccer video. Adding that the algorithm can be used for other sports game like US football, golf and any sport games that have the play field color is green.

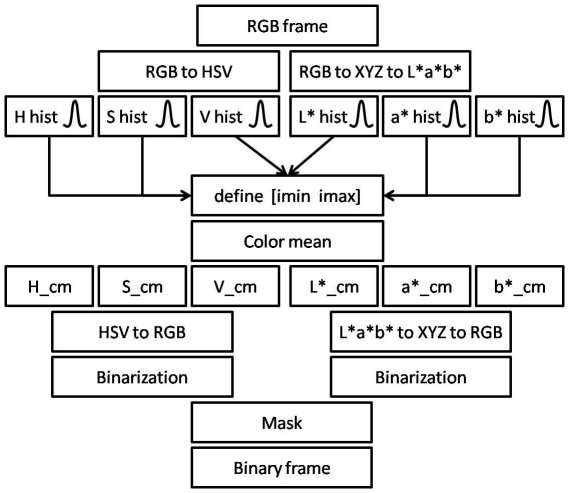


FIGURE 2: Dominant color stages.

The flowchart of the proposed algorithm is given in Figure 2. First, the system converts the **RGB** frames to **HSV** and to **L*a*b*** frames. After the conversion of frames, the system compute the histograms for each component (**H**, **S**, **V**, **L***, **a*** and **b***) using the following quantization factor: ***64*** bins for **H** and **L***, ***64*** bins for **S** and **b*** and ***128*** bins for **V** and **a***. Next, in order to remove noise effects that may result from using a single histogram index, the peak index ($\mathbf{i_{peak}}$), for each histogram is localized to estimate the mean value of dominant color for the corresponding color component. An interval about each histogram peak is defined, where the interval boundaries $\mathbf{[i_{min},i_{max}]}$ correspond to the dominant color. Which given in Equations (1).

$$\begin{aligned} &\sum_{i=i_{\min}}^{i_{peak}} H[i] <= 2H[i_{peak}] \quad and \quad \sum_{i=i_{\min}-1}^{i_{peak}} H[i] > 2H[i_{peak}] \\ &\sum_{i=i_{peak}}^{i_{\max}} H[i] <= 2H[i_{peak}] \quad and \quad \sum_{i=i_{peak}}^{i_{\max}+1} H[i] < 2H[i_{peak}] \end{aligned} \tag{1}$$

After the interval boundaries are determined, the mean color in the detected interval is computed by Equation (2) for each color component, and we get $\mathbf{H_{cm}}$, $\mathbf{S_{cm}}$, $\mathbf{V_{cm}}$, $\mathbf{L^*_{cm}}$, $\mathbf{a^*_{cm}}$ and $\mathbf{b^*_{cm}}$.

$$colormean = \frac{\sum_{i=i_{\min}}^{i_{\max}} H[i] * i}{\sum_{i=i_{\min}}^{i_{\max}} H[i]} \tag{2}$$

Next step, the system convert the mean color of each color component into ***RGB*** space to binarize the frame using the Equation (3). Where $\boldsymbol{I_R}$, $\boldsymbol{I_G}$, $\boldsymbol{I_B}$ are the matrix of *Red*, *Green*} and *Blue* components in the ***RGB*** frame. **K**, $\mathbf{R_t}$, $\mathbf{G_t}$, $\mathbf{B_t}$ and $\mathbf{G_{th}}$ are the thresholds for binarzation. ***G(x,y)*** is the binarized frame.

$$G(x,y)=\begin{cases}1 & if \quad \begin{cases} I_G(x,y) > I_R(x,y) + K(G_{cm} - R_{cm}) \\ I_G(x,y) > I_B(x,y) + K(G_{cm} - B_{cm}) \\ |I_R - R_{cm}| < R_t \\ |I_G - G_{cm}| < G_t \\ |I_B - B_{cm}| < B_t \\ I_G > G_{th} \end{cases} \\ 0 & otherwise \end{cases} \tag{3}$$

The last step in the system, is the obtaining the final binarized frame, by using the two binarized frame comes from HSV space and from L*a*b space. We use the binarized frame with HSV space as base and the binarized frame with L*a*b as mask, to improve the quality of the final frame.

$$\begin{cases} if \quad \sum Ibin_{hsv} > T_{th*} N^2 \quad then \quad Ibin^N_{hsvLab} = Ibin^N_{hsv} \\ else \quad Ibin = Ibin^N_{Lab} \end{cases} \tag{4}$$

The Equation (4), shows the application of binarized frame with L*a*b space as mask into binarized frame with HSV space. Where ***N*** is the block size, $\boldsymbol{T_{th}}$ the threshold, $\mathbf{Ibin_{hsv}}$ is the binarized frame using HSV space, $\mathbf{Ibin_{Lab}}$ the binarized frame using L*a*b* and $\mathbf{Ibin_{hsvLab}}$ is the resulted binarized frame using the two colors spaces.

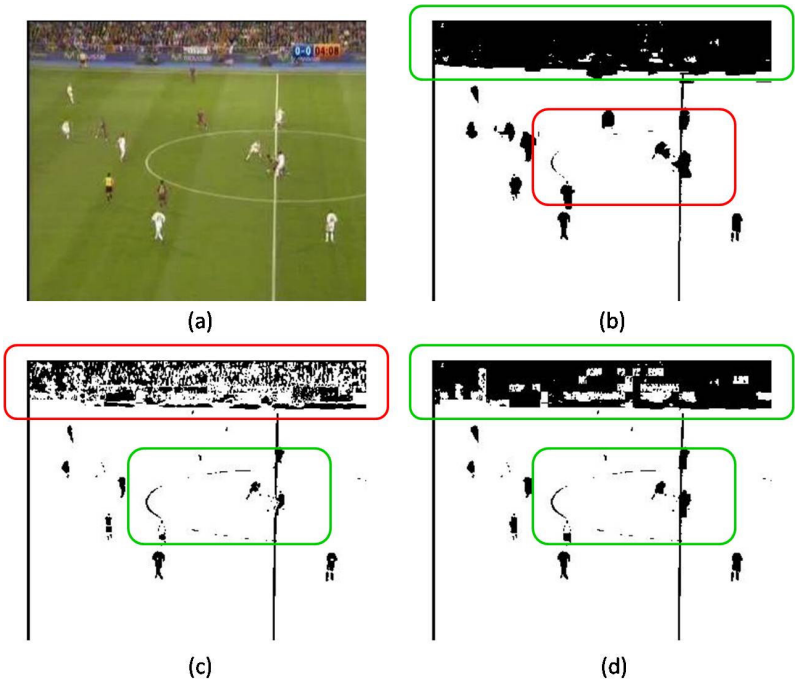


FIGURE 3: Dominant color extraction result.

Figure 3 show the result of dominant color extraction algorithm obtained in soccer video. Fig 3(a) it's the *RGB* frame, Fig 3(b) binarized frame comes from *HSV* space, Fig 3(c) binarized frame from L*a*b and Fig 3(d) it's the binarized frame using two color spaces and equation (4). This algorithm have generic behavior, we can apply it in US football and golf sport's game.

3. SHOT DETECTION

The shot is often used as a basic unit for video analysis and indexing. A shot may be defined as a sequence of frames captured by "a single camera in a single continuous action in time and space". The extraction of this unit (shot) still presents problems for sports video.

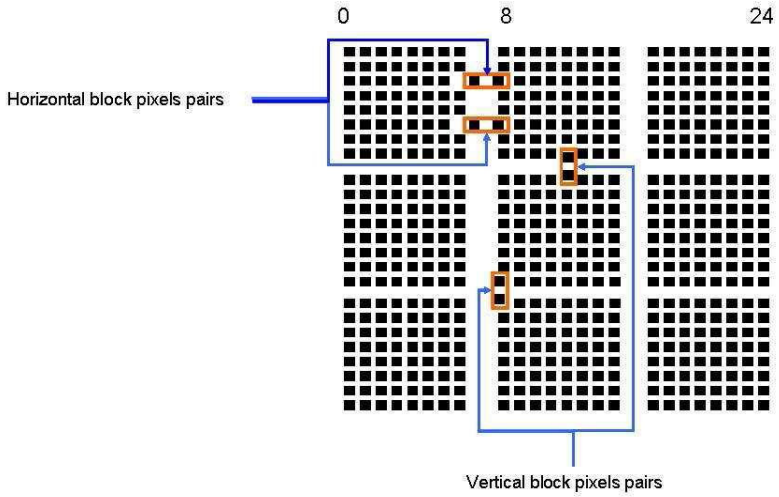


FIGURE 4: Adjacent pixels.

For shot detection, we use our algorithm based on Discrete Cosine Transform mutli-resolutions (DCT-MR) [8]. We binarize each I/P frames and we divide them into blocks of 2^R*2^R (**R** is the resolution) and for every block we calculate the DCT then we calculate the vertical distance ***distV*** (Eq (5)) and the horizontal distance ***distH*** (Eq (6))between adjacent pixels in blocks (Figure 4), then the means of both distances ***distHV*** is computed using equation (7).

$$distV = \sum_{i=1}^{\frac{w-R}{R}} \sum_{j=1}^{h} \frac{\left| pixel_{Rij} - pixel_{R(i+1)j} \right|}{h(w-R)/R} \tag{5}$$

$$distH = \sum_{i=1}^{w} \sum_{j=1}^{\frac{h-R}{R}} \frac{\left| pixel_{iRj} - pixel_{iR(j+1)} \right|}{w(h-R)/R} \tag{6}$$

$$distHV = \frac{distH + distV}{2} \tag{7}$$

Where ***w*** and ***h*** are the width and the height of frame and ***R*** is the resolution. We compare the distance ***distHV*** with the threshold ***0.12*** in order to decide if there is shot change or not [8].

4. SHOT CLASSIFICATION

The shots classification usually offers interesting information for the semantics of shots and cues. In this algorithm we used our algorithm in [9], which is a statistical method for shots classification. The method is based on spatial segmentation using 3:5:3 dividing format of the binarized key frames extracted previously.

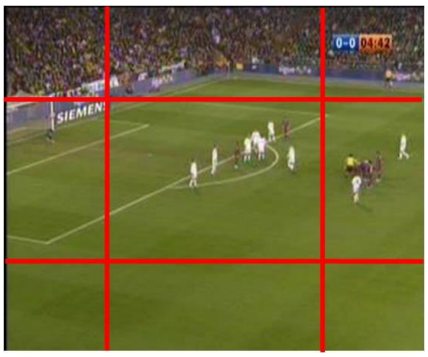


FIGURE 5: 3:5:3 division frame format.

The figure 5 shows the division 3:5:3 format of binarized frame. The shot classification algorithm in [9] gives very promising results in soccer video. We classify shot into four classes [9]: ***Long Shot***, ***Medium Shot***, ***Close-up Shot*** and ***Out Field Shot***.

5. AUDIO DESCRIPTOR

Adding to the color information in video track, the Audio track also bears important information that should not be ignored and, as well, gives the semantic to the video, especially in action and sport videos. In the case of soccer, when there is an important moment (goal, penalty, … etc), the voice intensity of the commentator's rises and falls proportionally with the action. This intensity of the audio track can be used as descriptor to more characterize the video.

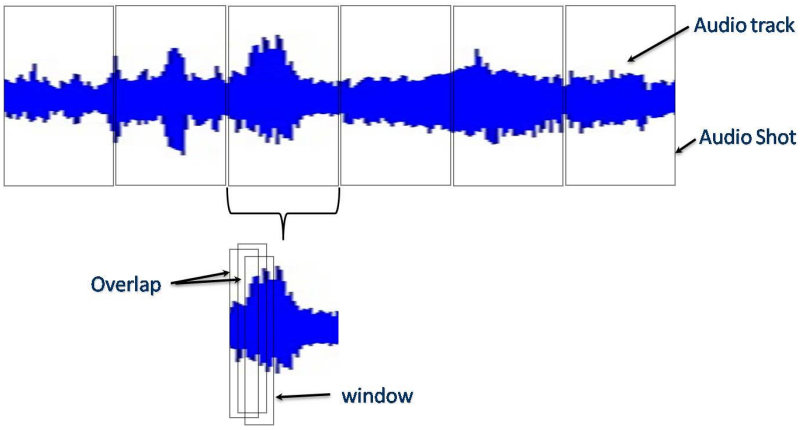


FIGURE 6: Windowing of the audio track.

In our method of soccer summaries/highlights extraction, we use the windowing algorithm with parameters: window size is ***10 seconds*** and the overlap parameter is ***2 seconds*** as shown in Figure 6; to compute the energy for each window in shot using Equation (8).

$$E_{shot,window} = \frac{1}{N}\sum_{i=1}^{N} S^2 \qquad (8)$$

In Equation (8), we compute the energy in each window of shot audio track, where **N** represents the number of samples in window of shot audio track and **S** is the set of samples in window of shot audio track.

After the energy computing, we extract the Maximum and the Minimum of energy of each shot; at the end of this step we get a vector of shot's Max and Min energy.

$$Max\_index = \arg Max(E_{shot}) \quad (9)$$

$$Min\_index = \arg Min(E_{shot}) \quad (10)$$

We make use this ***Max energy*** vector of shots to generate the candidate highlight of audio track. The *Max_index* and *Min_index* are used as indexes in XML file (section Audio Video XMLisation Block).

6. SCORE BOX DETECTION AND TEXT RECOGNITION

Text in video is normally generated in order to supplement or to summarize the visual content and thus is an important carrier of information that is highly relevant to the content of the video. As such, it is a potential ready-to-use source of semantic information.

Figure 7 show the stages of our algorithm for score box extraction and text recognition in soccer video. The first step consist of extraction a clip from the whole video, the length of the video clip extracted is **L**. The score box sub-block Extraction stage is the second step, that is based on soccer video editing. After, score box detection based on motion vector using Diamond Search (DS) algorithm. Next, the text detection in score box with binarization and number of morphology constraint (pre-processing). Finally, we use Optical Character Recognition (OCR) algorithm to generate an ASCII file contain the text detected in score box of soccer video.

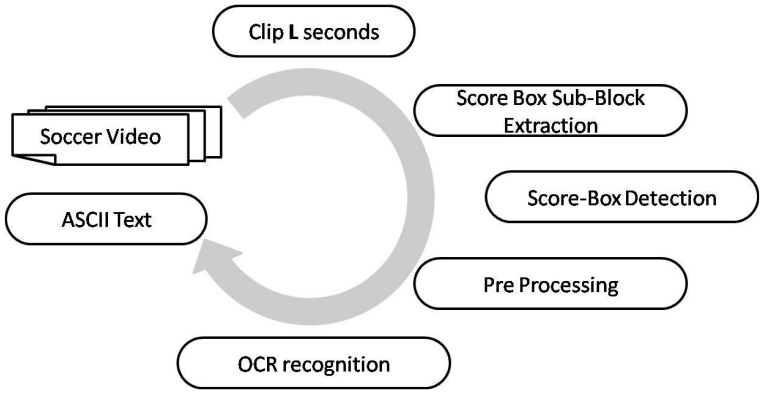


FIGURE 7: Score box detection flowchart.

In soccer video, the editors put the result of match and the names of the both teams on the top corners (left corner or right corner) (Figure 8). This box (teams' names and/or the result) remain superimposed in the video on the same place during the match.

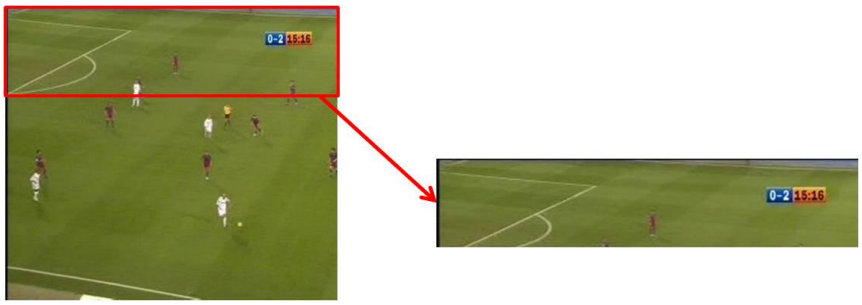


FIGURE 8: Sub-Block extraction.

With this style of video editing in soccer game, we extract the top block of frame which is about **30%** of screen video size (To optimize the search in DS algorithm), where we investigate the text superimposed in the video.

The property of stability of score box in the same place (corner) allows us to extract the score-box using motion vector (MV). To extract the score-box, we get ***L second*** (***L*** is the length of the clip in second for test) from the soccer video and we use the Diamond Search (DS) algorithm to extract motion vector of sub-block extracted previously (Figure 8). In our work we use DS algorithm with follows parameters: the Macro Block size is **8** pixels and the Search Parameter **p** size is **7** pixels.

To compute the motion vector of clip, we make a sampling of one from $\mathbf{S_{fr}}$ frames (we define $\mathbf{S_{fr}}$ in section of experiments). At the end, we get the motion vectors of all sub-blocks of all selected frames in sampling.

After getting the vector motion of all selected frames in clip of **L** seconds, we compute the 2D variance using Equation (11)

$$\sigma\_i_{t+1}^{\ t} = \sqrt{\frac{\sum_{l=1}^{M}\sum_{c=1}^{N}(MV_i(t+1)_{lc} - MV_i(t)_{lc})^2}{M*N-1}} \tag{11}$$

Where **M** and **N** are the height and the width of the matrix **MV** respectively, and **i** refers to the samples number **i**.

$$\sigma\_i_{mean} = \frac{1}{2*NS_{fr}*p}\sum_{t=1}^{K-1}\sigma\_i_{t+1}^{t} \tag{12}$$

In Equation (12), $\mathbf{NS_{fr}}$ represents the number of samples and **k** is the number of macro blocks in DS algorithm and **p** is the search parameter.

$$\begin{cases} if \quad \sigma^{i}_{mean} < T_{thd}\ then \quad i\_BlockStatic \\ else \quad i\_BlockNotStatic \end{cases} \tag{13}$$

The last step in searching of the static macro blocks that are candidate belong to the score box, we use the Equation (13) for this purposes, where $\mathbf{T_{thd}}$ is the threshold.

FIGURE 9: Score box detection result.

Figure 9 shows the result obtained for score box detection in clip video of **L = 3** second, sampling of **1** frame form **10** frames $\mathbf{S_{fr}=1/10}$, the DS algorithm with parameters: macro block of **8** pixels, search parameter of **7** pixels, and $\mathbf{T_{thd}=0.2}$ in Equation (13).

In order to extract the totality of the score box and eliminate the border effect, we delete one macro block from each side of sub-block and we add one macro block form each side of score-box detected. The Figure 10 shows the final result obtained after this procedure.

FIGURE 10: Score box extraction result.

After score-box extraction, we perform the binarization of the edge map in order to separate text-containing regions from the rest of the score-box using Otsu's global thresholding method as described in [10]. To reduce the noise and to connect loose characters form complete words, a number of morphological operations are performed as pre-processing. In Our case the morphological operations consist of the following steps:

- Step 1: 2x2 median filters.
- Step 2: 2x1 dilation.
- Step 3: 2x2 opening.
- Step 4: dilation in the horizontal direction using a 3x1.

The steps from 1 to 4 are repeated four times to avoid the problems mentioned above.

To complete our method and achieve the aims cited above, we used the freely optical recognition software know as ClaraOCR [11] to generate ASCII files.

7. AUDIO VIDEO XMLISATION

In this section we present a description of soccer video in XML file format. In sport video analysis, the Shot detection, shot classification, score extraction, key frames extraction and audio track analysis are very important steps for soccer video summaries and highlights extraction.

To give meaning to these steps, we present the video in a XML file format for future uses (section Summaries/Highlight Extraction). The structure adopted in our algorithm presented in figure bellow :

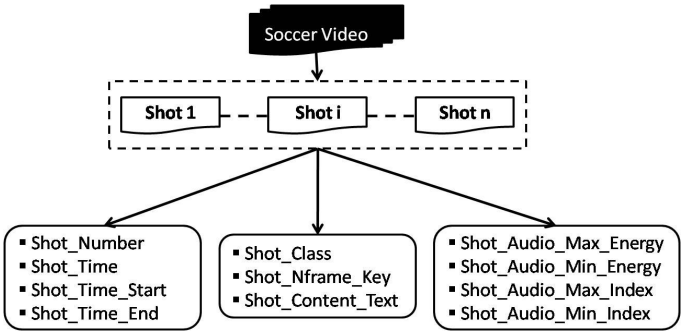


FIGURE 11: Adopted XML video file format.

In this structure we make used time as new factor, we shall use this factor in Finite State Machine (FSM) to generate the video summaries and highlights.

8. FINITE STATE MACHINE

In this section we present the finite state machine which modelizes soccer video. This finite state machine used with domain knowledge to generate dynamic summaries and highlights of soccer video (Figure 12).

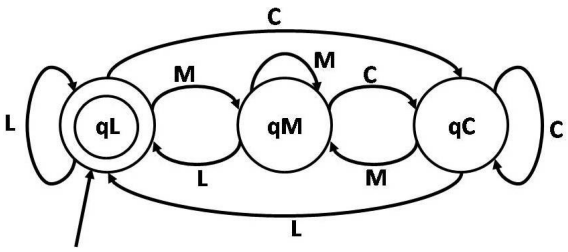


FIGURE 12: Adopted Finite State Machine for Soccer video (FSM-SC).

A Finite State Machine (FSM) is a 5-tuple $\mathbf{M = \{P; Q; \delta; q_0; F\}}$, where **P** is the alphabet, **Q** is a set of state, δ is a set of transition rules: $\delta \subseteq (Q^* \sum U \varepsilon^* Q)$, $q_0 \in Q$ is a set of initial (or starting) states and $F \in Q$ is a set of final states. In our case of soccer video, **P** is the alphabet which presents the summary, **Q = {qL;qM;qC}**, where **qL** is a Long shot, **qM** is a Medium shot and **qC** is a Close-up shot, δ is the domain knowledge, q_0 **= {qL}**, and **F = {qL}**.

Each video can be represented as a set of shots and scenes. The same, soccer video is a set of long, medium, close-up and out of field shots (for example: LLMMMCCMMLCMMMCCCLLLMLLCLMCCMLCM\dots).

Our Finite State Machine of soccer video (FSM-SC) will seek this **word** to find the **sub-words** that can be candidate as important moment (highlight) or can be as summary.

9. DOMAIN KNOWLEDGE

Domain knowledge is defined as the content of a particular field of knowledge, also the sum or range of what has been perceived, discovered, or learned.

Using the domain knowledge of soccer video which are the sum of the concepts and the steps followed by the most of editors of soccer match videos, we deduce a set of features specific to this domain, that allow as to more understand the structure of this type of videos.

- There is a limited number of shot views, Long view, Medium view, Close-up view and Out of field view.
- After every important moment in the match (goal, penalty,…), the editors make replays and/or slow motions.
- The cameraman always tries to follow all the actions: ball, goals, players, referee\dots etc.
- In order that TV viewers could see the action better, the cameraman puts the action in the middle of the screen.
- In case of goal, the cameraman keeps an eye on the striker player who reaches the goal.

10. SUMMARIES AND HIGHLIGHTS EXTRACTION

The last step of our algorithm of summaries and highlights extraction in soccer video is shown in Figure 13.

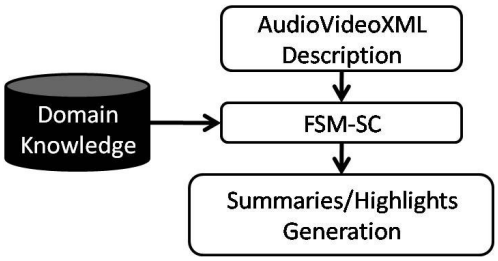


FIGURE 13: Last step in highlight extraction algorithm.

We make use the audio/video features: shot boundaries, shot classification, text in score box and Energy of audio presented in XML file and our FSM proposed to generate summaries and highlights of soccer video. Our framework use also the knowledge domain (set of rules) that helps the FSM to extract the video segments candidate to be summaries or highlights.

11. EXPERIMENTAL RESULTS

To show the effectiveness of our method that brings together the basic elements of the analysis of the video (shot detection and shot classification), we make use of five clips from five matches of soccer video. All the clips are in MPEG format, 352x288 size, 1150 kbps, 25 frames per second; dominant color ratio is computed on I- and P-frames.

Clip	Mpeg	Length	Goals	Fps	Total frames
clip soccer_1	2	10 min 22 sec	2	25	15551
clip soccer_2	2	13 min 06 sec	1	25	19652
clip soccer_3	2	16 min 55 sec	1	25	25380

TABLE 1: Clips information.

Table 1 shows a brief description of soccer video clips.
Where:

Clip Soccer_1: Match: FC Barcelona vs Real Betis.
Clip Soccer _2: Match: FC Barcelona vs Real Madrid.
Clip Soccer _3: Match: Cameroon vs Tunisia.

Clip	Summaries	Highlights	Detected Goals	Nothing	Missed
clip soccer_1	4	4	2	0	0
clip soccer_2	3	3	1	0	0
clip soccer_3	4	3	1	1	0

TABLE 1: Result of highlight and summaries extraction

.

Table 2 shows the obtained results of FSM, where **Summaries** presents the number of summaries generated by our framework, **Highlights** presents the number of important moment detected in clip, **Goals** is the number of goal detected in clip and **Nothing** presents summary of the clip in peaceful moments. Finally, **Missed** is the number of highlights missed by the framework in the clip.

12. CONSLUSION & FUTURE WORK

In this paper we present a full and efficient framework for highlights and summaries extraction in video soccer using audio/video features based on an enhanced and generic method for dominant color extraction.

The framework leaves much more for improvement and extension: there are other relevant low-level features that might provide complementary information and may help improve performance, such as camera motion, edge and higher-level object detectors.

13. REFERENCES

1. A. Ekin, A. M. Tekalp and R. Mehrotra. “Robust dominant color region detection with applications to sports video analysis”. In Proceedings of IEEE ICIP, vol. 1, pp. 21-24, 2003

2. P. Xu, L. Xie, S. Chang, A. Divakaran, A. Vetro and H. Sun. “Algorithms and system for segmentation and structure analysis in soccer video,” In Proceedings of IEEE ICME, pp. 928-931, 2001

3. A. Hanjalic. “Adaptive extraction of highlights from a sport video based on excitement modeling”. IEEE Trans. on MultiMedia, pp. 1114-1122, 2005

4. D. Tjondronegoro, Y.-P. Chen and B. Pham, “Highlights for more complete sports video summarization,” IEEE Trans. on Multimedia, pp. 22-37, 2004

5. N. Babaguchi and N. Nitta. “Intermodal collaboration: a strategy for semantic content analysis for broadcasted sports video”. IEEE ICIP, vol. 1, pp. 13-16, 2003

6. B. Pan, H. Li and M. I. Sezan; “Automatic detection of replay segments in broad-cast sports programs by detection of logos in scene transitions”. IEEE ICASSP, pp. 3385-3388, 2002

7. X. Tong, H. Lu, Q. Liu and H. Jin. “Replay detection in broadcasting sports videos”. ICIG, 2004

8. Y. Tabii and R. O. H. Thami. “A new method for soccer shot detection with multi-resolution dct”. COmpression et REprsentation des Signaux Audiovisuels (CORESA), 2007

9. Y. Tabii, M. O. Djibril, Y. Hadi and R. O. H. Thami. “A new method for video soccer shot classification,” 2nd International Conference on Computer Vision Theory and Applications(VISAPP), pp. 221-224, 2007

10. C. Wolf and J. M. Jolion. “Extraction and recognition of artificial text in multimedia documents”, Technical report (2002)

11. ClaraOCR download website : http://www.claraocr.org

A Comparison of SIFT, PCA-SIFT and SURF

Luo Juan qiuhehappy@hotmail.com
Computer Graphics Lab,
Chonbuk National University,
Jeonju 561-756, South Korea

Oubong Gwun obgwun@chonbuk.ac.kr
Computer Graphics Lab,
Chonbuk National University,
Jeonju 561-756, South Korea

Abstract

This paper summarizes the three robust feature detection methods: Scale Invariant Feature Transform (SIFT), Principal Component Analysis (PCA)–SIFT and Speeded Up Robust Features (SURF). This paper uses KNN (K-Nearest Neighbor) and Random Sample Consensus (RANSAC) to the three methods in order to analyze the results of the methods' application in recognition. KNN is used to find the matches, and RANSAC to reject inconsistent matches from which the inliers can take as correct matches. The performance of the robust feature detection methods are compared for scale changes, rotation, blur, illumination changes and affine transformations. All the experiments use repeatability measurement and the number of correct matches for the evaluation measurements. SIFT presents its stability in most situations although it's slow. SURF is the fastest one with good performance as the same as SIFT. PCA-SIFT show its advantages in rotation and illumination changes.

Keywords: SIFT, PCA-SIFT, SURF, KNN, RANSAC, robust detectors.

1. INTRODUCTION

Lowe (2004) presented SIFT for extracting distinctive invariant features from images that can be invariant to image scale and rotation. Then it was widely used in image mosaic, recognition, retrieval and etc. After Lowe, Ke and Sukthankar used PCA to normalize gradient patch instead of histograms [2]. They showed that PCA-based local descriptors were also distinctive and robust to image deformations. But the methods of extracting robust features were still very slow. Bay and Tuytelaars (2006) speeded up robust features and used integral images for image convolutions and Fast-Hessian detector [3]. Their experiments turned out that it was faster and it works well.

There are also many other feature detection methods; edge detection, corner detection and etc. Different method has its own advantages. This paper focuses on three robust feature detection methods which are invariant to image transformation or distortion. Furthermore, it applies the three methods in recognition and compares the recognition results by using KNN and RANSAC methods. To give an equality

comparison, use the same KNN and RANSAC to the three methods. In the experiment, we use repeatability measurement to evaluate the performance of detection for each method [4]; the higher repeatability score is better than the lower one. When a method gives a stable detector and matching numbers we can say that it is a stable method and if we want to know how correct the method is, we need to use correct matches number that can be get from the RANSAC method.

The related work is presented in Section 2 while Section 3 discusses the overview of the method. In section 4 we can see the experiments and results. Section 5 tells the conclusions and future work of the paper.

2. RELATED WORK

In [1], Lowe did not only presented SIFT but also discussed the keypoint matching which is also needed to find the nearest neighbor. He gave an effective measurement to choose the neighbor which is obtained by comparing the distance of the closest neighbor to the second-closest neighbor. In my experiment compromising of the cost and match performance, the neighbor will be chosen when the distance ratio is smaller than 0.5 [1]. All the three methods use the same RANSAC model and parameters, which will explain more in the following.
K. Mikolajczyk and C. Schmid [6], compared the performance of many local descriptors which used recall and precision as the evaluation criterion. They gave experiments of comparison for affine transformations, scale changes, rotation, blur, compression, and illumination changes. In [7], they showed how to compute the repeatability measurement of affine region detectors also in [4] the image was characterized by a set of scale invariant points for indexing.

Some researches focused on the application of algorithms such as automatic image mosaic technique based on SIFT [9][11], stitching application of SIFT [10][15][12] and Traffic sign recognition based on SIFT [12]. Y. Ke [2] gave some comparisons of SIFT and PCA-SIFT. PCA is well-suited to represents keypoint patches but observed to be sensitive to the registration error. In [3], the author used Fast-Hessian detector which is faster and better than Hessian detector. Section 3 will show more details of the three methods and their differences.

3. OVERVIEW OF THE THREE METHODS

3.1 SIFT detector

SIFT consists of four major stages: *scale-space extrema detection, keypoint localization, orientation assignment and keypoint descriptor.* The first stage used difference-of-Gaussian function to identify potential interest points [1], which were invariant to scale and orientation. DOG was used instead of Gaussian to improve the computation speed [1].

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) = L(x, y, k\sigma) - L(x, y, \sigma) \qquad (1)$$

In the keypoint localization step, they rejected the low contrast points and eliminated the edge response. Hessian matrix was used to compute the principal curvatures and eliminate the keypoints that have a ratio between the principal curvatures greater than the ratio. An orientation histogram was formed from the gradient orientations of sample points within a region around the keypoint in order to get an orientation assignment [1]. According to the paper's experiments, the best results were achieved with a 4 x 4 array of histograms with 8 orientation bins in each. So the descriptor of SIFT that was used is 4 x 4 x 8 = 128 dimensions.

3.2 PCA-SIFT detector

PCA is a standard technique for dimensionality reduction [2], which is well-suited to represent the keypoint patches and enables us to linearly-project high-dimensional samples into a low-dimensional feature space.

In other words, PCA-SIFT uses PCA instead of histogram to normalize gradient patch [2]. The feature vector is significantly smaller than the standard SIFT feature vector, and it can be used with the same matching algorithms. PCA-SIFT, like SIFT, also used Euclidean distance to determine whether the two vectors correspond to the same keypoint in different images. In PCA-SIFT, the input vector is created by concatenation of the horizontal and vertical gradient maps for the 41x41 patch centered to the keypoint, which has 2x39x39=3042 elements [2]. According to PCA-SIFT, fewer components requires less storage and will be resulting to a faster matching, they choose the dimensionality of the feature space, n = 20, which results to significant space benefits [2].

3.3 SURF detector

SIFT and SURF algorithms employ slightly different ways of detecting features [9]. SIFT builds an image pyramids, filtering each layer with Gaussians of increasing sigma values and taking the difference. On the other hand, SURF creates a "stack" without 2:1 down sampling for higher levels in the pyramid resulting in images of the same resolution [9]. Due to the use of integral images, SURF filters the stack using a box filter approximation of second-order Gaussian partial derivatives, since integral images allow the computation of rectangular box filters in near constant time [3].

In keypoint matching step, the nearest neighbor is defined as the keypoint with minimum Euclidean distance for the invariant descriptor vector. Lowe used a more effective measurement that obtained by comparing the distance of the closest neighbor to that second-closest neighbor [1] so the author of this paper decided to choose 0.5 as distance ratio like Lowe did in SIFT.

4. EXPERIMENTS & RESULTS

4.1 Evaluation measurement

The repeatability measurement is computed as a ratio between the number of point-to-point correspondences that can be established for detected points and the mean number of points detected in two images [4]:

$$r_{1,2} = \frac{C(I_1, I_2)}{mean(m_1, m_2)} \quad (2)$$

Where $C(I_1, I_2)$ denotes the number of corresponding couples, m_1 and m_2 means the numbers of the detector. This measurement represents the performance of finding matches.

Another evaluation measurement is RANSAC, which is used to reject inconsistent matches. The inlier is a point that has a correct match in the input image. Our goal is to obtain the inliers and reject outliers in the same time [4]. The probability that the algorithm never selects a set of m points which all are inliers is $1-p$:

$$1-p = (1-w^m)^k \quad (3)$$

Where m is the least number of points that needed for estimating a model, k is the number of samples required and w is the probability that the RANSAC algorithm selects inliers from the input data. The RANSAC repeatedly guess a set of mode of correspondences that are drawn randomly from the input set. We can think the inliers as the correct match numbers. In the following experiments, matches mean inliers.

FIGURE 1: Part of test images. A and H are the affine transformed images, B and C are the scale changed images, D are the rotation images, E and F are the blurred images, G are the illumination changed images.

In this paper, the three methods that were used are all based on opencv. We use the same image dataset, which includes the general deformations, such as scale changes, view changes, illumination changes and rotation. As shown in figure 1. All the experiments work on PC AMD 3200+, 2.0G, and 1.0 GB RAM, with Windows XP as an operating system.

4.2 Processing Time

Time evaluation is a relative result, which only shows the tendency of the three methods' time cost. There are factors that influenced on the results such as the size and quality of the image, image types (e.g. scenery or texture), and the parameters of the algorithm (e.g. the distance ratio) [1]. The first part of the experiment uses Graffiti dataset as shown in group A of figure 1, whose sizes are all 300 x 240 pixels. The parameters of the three algorithms are the same settings according to the original paper [1] [2] [3]. Time is counted for the complete processing which includes feature detecting and matching. Table 1 show that SURF is the fastest one, SIFT is the slowest but it finds most matches.

Items	SIFT	PCA-SIFT	SURF
total matches	271	18	186
total time (ms)	2.15378e+007	2.13969e+007	3362.86
10 matches' time(ms)	2.14806e+007	2.09696e+007	3304.97

TABLE 1: Processing time comparison. Using group A of figure 1, total time is the time of finding all the matches, 10 matches' time is counted until the first 10 matches.

4.3 Scale Changes

The second experiment shows the performance of scale invariant, which uses group B and C of figure 1. Figure 2 shows some of the matching images, table 2 shows the matching number of the three methods. In order to see the matches clearly, we just show the first 10 matches [2]. When the scale change gets larger, SIFT or SURF is much better than PCA-SIFT, therefore the results shows that PCA-SIFT is not stable as SIFT and SURF to scale invariant and PCA-SIFT detects few matches.

FIGURE 2: Scale changes comparison. Using Group B, C of figure 1. The first result is SIFT (10/66matches), second is SURF (10/26matches). The last two images show the results of PCA-SIFT in scale changes.

Data	SIFT	PCA-SIFT	SURF
1-2	41	1	10
3-4	35	0	36
5-6	495	19	298
7-8	303	85	418

TABLE 2: Scale changes comparison. Using group B, C of figure 1, data represents the total number of matches for each method.

4.4 Image Rotation

The third experiment shows the influence of rotation on the three methods. As shown in group D of figure 1, the image rotates 5 or 10 degrees. SIFT is represented by the blue line that detects the most matches and stable to rotation. SURF doesn't work well, it finds the least matches and gets the least repeatability which also shown in figure 3 and 4. In addition with, PCA-SIFT found only one correct match of the first ten matches. Although it has one correct match, we can still improve the PCA-SIFT and this is better than SURF.

4.5 Image Blur

This fourth experiment uses Gaussian blur like the images in group E and F of figure 1. The radius of the blur changes from 0.5 to 9.0. As shown in figure 5, SURF and PCA-SIFT shows good performance, when the radius gets larger, mention the matching results in figure 6, SURF detects few matches and PCA-SIFT finds more matches but smaller correct number. In this case, SIFT shows its best performance here.

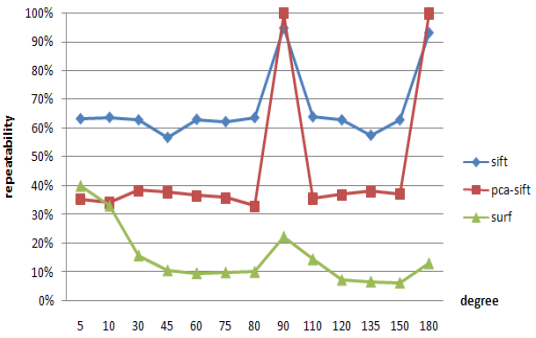


FIGURE 3: Rotation comparison. Using group D of figure 1, data represents the repeatability of rotation.

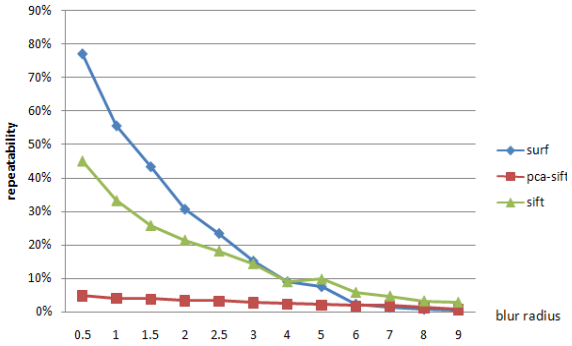


FIGURE 5: Blur comparison. Using group E of figure 1, data represents the repeatability of blur.

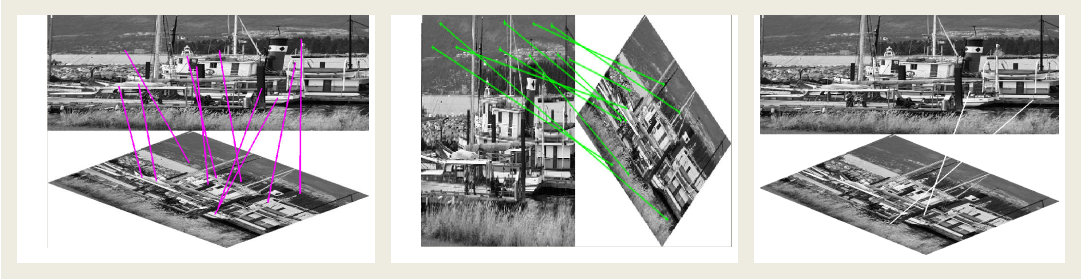


FIGURE 4: Rotation comparison. Using group D of figure 1, rotation degree is 45. The first two images show the first ten matches of SIFT (10 correct / 10) and PCA-SIFT (1 correct / 10), the result of SURF (2 correct / 2) is the total matches.

FIGURE 6: Blur comparison. Using group E and F of figure 1. From left to right, the results are: SIFT (10 correct / 10), SURF (3 correct / 3), PCA-SIFT (1 correct / 10), which blur radius is 9.

4.6 Illumination Changes

This fifth experiment shows the illumination effects of the methods. As shown in group G of figure 1, from data 1 to data 6, the brightness of the image gets lower and lower. Table 3 shows the repeatability of illumination changes. SURF has the largest repeatability, 31%, PCA-SIFT shows as good performance as SURF, which coincides with [2][3][6]. Look at one of the experiment result in figure 7, which shows the first 10 matches.

FIGURE 7: Illumination changes comparison. Using group G of figure 1, from left to right, SIFT (9 correct / 10), PCA-SIFT (6 correct / 10), SURF (10correct/10).

Data	SIFT	PCA-SIFT	SURF
1-2	43%	39%	70%
1-3	32%	34%	49%
1-4	18%	27%	25%
1-5	8%	18%	6%
1-6	2%	9%	5%
average	21%	25%	31%

TABLE 3: Illumination changes comparison. Using group G of figure 1, the data represents repeatability and the average of repeatability.

4.7 Affine Transformations

This sixth experiment evaluates the methods' stability of affine transformations (view transformation). Affine transformation is important in panorama stitching. This experiment uses images in group A and H of figure 1, the viewpoint change is approximately 50 degrees from data 1–6. Repeatability of affine transformation shows in table 4 and the matching results shown in figure 8. From the table and figure, SURF and SIFT have a good repeatability when the viewpoint change is small, but when the viewpoint change is larger than data 1–4, SURF detects 0 matches and PCA-SIFT shows better performance.

Data	SIFT	PCA-SIFT	SURF
1-2	47%	15%	54%
1-3	37%	12%	33%
1-4	22%	12%	12%
1-5	7%	10%	2%
1-6	0%	9%	0%

TABLE 4: Affine transformation comparison. Using group A, H of figure 1, the data represents the repeatability.

FIGURE 8: Affine transformations comparison. Using group A and H of figure 1, when overview change is large, the results shows from left to right: SIFT (5 correct / 10), SURF (0), PCA-SIFT (1), PCA-SIFT(2 / 10).

4.8 Discussion

Table 5 shows the results of all experiments. It also shows that there is no best method for all deformation. Hence, when choosing a feature detection method, make sure which most concerned performance is. The result of this experiment is not constant for all cases. Changes of an algorithm can get a new result, find the nearest neighbor instead of KNN or use an improved RANSAC.
SIFT's matching success attributes to that its feature representation has been carefully designed to be robust to localization error. As discussed by Y. Ke, PCA is known to be sensitive to registration error. Using a small number of dimensions provides significant benefits in storage space and matching speed [2]. SURF shows its stability and fast speed in the experiments. It is known that 'Fast-Hessian' detector that used in SURF is more than 3 times faster that DOG (which was used in SIFT) and 5 times faster than Hessian-Laplace [3]. From the following table, we know that PCA-SIFT need to improve blur and scale performances. SURF looks fast and good in most situations, but when the rotation is large, it also needs to improve this performance. SIFT shows its stability in all the experiments except for time, because it detects so many keypoints and finds so many matches, we can think about matching in some interest keypoints.

Method	Time	Scale	Rotation	Blur	Illumination	Affine
SIFT	common	best	best	best	common	good
PCA-SIFT	good	common	good	common	good	good
SURF	best	good	common	good	best	good

TABLE 5: Conclusion of all the experiments.

5 . CONCLUSIONS & FUTURE WORK

This paper has evaluated three feature detection methods for image deformation. SIFT is slow and not good at illumination changes, while it is invariant to rotation, scale changes and affine transformations.

SURF is fast and has good performance as the same as SIFT, but it is not stable to rotation and illumination changes. Choosing the method mainly depends on the application. Suppose the application concern is more than one performances compromising from the Table 5, choose a suitable algorithm and giving improvement according to the application. For example, PCA-SIFT is the best choice, but the application also concerns blur performance, so what needed is to give PCA-SIFT some improvement on blur performance. The future work is to improve the algorithm and apply these methods on single areas, such as image retrieval or stitching.

6. REFERENCES

1. **FOR JOURNALS**: D. Lowe.*"Distinctive Image Features from Scale-Invariant Keypoints", IJCV*, 60(2):91–110, 2004.
2. **FOR CONFERENCES**: Y. Ke and R. Sukthankar.PCA-SIFT: *"A More Distinctive Representation for Local Image Descriptors"* ,*Proc. Conf. Computer Vision and Pattern Recognition*, pp. 511-517, 2004.

3. **FOR CONFERENCES**: Bay,H,. Tuytelaars, T., &Van Gool, L.(2006). "*SURF: Speeded Up Robust Features*", *9th European Conference on Computer Vision*.

4. **FOR CONFERENCES:** K. Mikolajczyk and C. Schmid. "*Indexing Based on Scale Invariant Interest Points"*. *Proc. Eighth Int'l Conf. Computer Vision*, pp. 525-531, 2001.

5. **FOR JOURNALS:** K.Kanatani. "*Geometric information criterion for model selection*", *IJCV, 26(3):*171-189,1998.

6. **FOR JOURNALS**: K. Mikolajzyk and C. Schmid. "*A Perforance Evaluation of Local Descriptors*", *IEEE,Trans. Pattern Analysis and Machine Intelligence*, vol.27, no.10, pp 1615-1630, October 2005.

7. **FOR JOURNALS:** K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L.V. Gool*." A Comparison of Affine Region Detectors*", *IJCV,* 65(1/2):43-72, 2005.

8. Eric Chu,Erin Hsu, Sandy Yu. *"Image-Guided Tours: Fast-Approximated SIFT with U-SURF Features*", Stanford University.

9. **FOR CONFERENCES**: Yang zhan-long and Guo bao-long. "*Image Mosaic Based On SIFT"*, International Conference on Intelligent Information Hiding and Multimedia Signal Processing, pp:1422-1425,2008.

10. **FOR CONFERENCES:** M. Brown and D. Lowe." *Recognizing Panoramas*". *Proc. Ninth Int'l Conf. Computer Vision*, pp. 1218-1227, 2003.

11. **FOR CONFERENCES:** Salgian, A.S. "*Using Multiple Patches for 3D Object Recognition*", Computer Vision and Pattern Recognition, CVPR '07. pp:1-6, June 2007.

12. **FOR CONFERENCES:** Y. Heo, K. Lee, and S. Lee. "*Illumination and camera invariant stereo matching* ". In CVPR, pp:1–8, 2008.

13. **FOR SYMPOSIUM:** Kus, M.C.; Gokmen, M.; Etaner-Uyar, S. " *Traffic sign recognition using Scale Invariant Feature Transform and color classification*". ISCIS '08. pp: 1-6, Oct. 2008.

14. **FOR TRANSACTIONS**: Stokman, H; Gevers, T." *Selection and Fusion of Color Models for Image Feature Detection* ". Pattern Analysis and Machine Intelligence, IEEE Transactions on Volume 29, Issue 3, pp:371 – 381, March 2007.

15. **FOR CONFERENCES:** Cheng-Yuan Tang; Yi-Leh Wu; Maw-Kae Hor; Wen-Hung Wang. "Modified *sift descriptor for image matching under interference*"**.** Machine Learning and Cybernetics, 2008 International Conference on Volume 6, pp:3294 – 3300, July 2008.

Performance Improvement of Vector Quantization with Bit-parallelism Hardware

Pi-Chung Wang pcwang@cs.nchu.edu.tw
Institute of Networking and Multimedia
and Department of Computer Science and Engineering
National Chung Hsing University
Taichung, 402, Taiwan, R.O.C.

Abstract

Vector quantization is an elementary technique for image compression; however, searching for the nearest codeword in a codebook is time-consuming. In this work, we propose a hardware-based scheme by adopting bit-parallelism to prune unnecessary codewords. The new scheme uses a "Bit-mapped Look-up Table" to represent the positional information of the codewords. The lookup procedure can simply refer to the bitmaps to find the candidate codewords. Our simulation results further confirm the effectiveness of the proposed scheme.

Keywords: Image Compression, Nearest Neighbor Search, Vector Quantization, Look-up Tables.

1. INTRODUCTION

The use of images has become a common practice in computer communications. The sizes of images are usually huge and, therefore, need to be compressed for storage and transmission efficiency. Vector quantization (VQ) [1] is an important technique for image compression and has been proven to be simple and efficient [2]. VQ can be defined as a mapping from k-dimensional Euclidean space into a finite subset C. The finite set C is known as the *codebook* and $C=\{c_i|i=1,2,\ldots,N\}$, where c_i is a codeword and N is the codebook size.

To compress an image, VQ comprises two functions: an encoder and a decoder. The VQ encoder first divides the image into $N_w \times N_h$ blocks (or vectors). Let the block size be k ($k=w\times h$), and then each block is a k-dimensional vector. VQ selects a codeword $c_q=[c_{q(0)},c_{q(1)},\ldots,c_{q(k-1)}]$ for each image vector $x=[x_{(0)},x_{(1)},\ldots,x_{(k-1)}]$ such that the distance between x and c_q is the smallest, where c_q is the closest codeword of x and $c_{q(j)}$ denotes the j_{th}-dimensional value of the codeword c_q. The distortion between the image vector x and each codeword c_i is measured by their *squared Euclidean distance*, i.e.,

$$d(x,c_i)=\|x-c_i\|^2=\sum_{j=0}^{k-1}\left[x_{(j)}-c_{i(j)}\right]^2 \quad (1)$$

After the selection of the closest codeword, VQ replaces the vector x by the index q of c_q. The VQ decoder has the same codebook as that of the encoder. For each index, VQ decoder can easily fetch its corresponding codeword, and piece them together into the decoded image.

The codebook search is one of the major bottlenecks in VQ. From Equation (1), the calculation of the squared Euclidean distance needs k subtractions and k multiplications to derive $k[x_{(j)}-c_{i(j)}]^2$. Since the multiplication is a complex operation, it increases the total computational complexity of Equation (1). Therefore, speeding up the calculation of the squared Euclidean distance is a major hurdle.

Owing to the importance of vector quantization, a handful of methods have been proposed to shorten VQ encoding time [3,5–8,14,16,17]. The simplest one among them is the *look-up table* (LUT) method [6,17]. It suggests that the results of $[x_{(j)}\text{-}c_{i(j)}]^2$ for all possible x_j and y_{ij} should be pre-computed first and then stored into a huge matrix, the *LUT*. Suppose the values of $x_{(j)}$ and $c_{i(j)}$ are within [0,*m*-1]. Then the size of matrix *LUT* should be $m \times m$ and

$$LUT = \begin{bmatrix} 0 & 1^2 & \cdots & (m-1)^2 \\ 1^2 & 0 & \cdots & (m-2)^2 \\ \vdots & \vdots & \ddots & \vdots \\ (m-1)^2 & (m-2)^2 & \cdots & 0 \end{bmatrix}_{m \times m} \tag{2}$$

Given any $x_{(j)}$ and $c_{i(j)}$, we can get the square of their difference directly from $LUT[x_{(j)}, c_{i(j)}]$. Therefore, Equation (1) could be rewritten as follows:

$$d(x, c_i) = \sum_{j=0}^{k-1} \left[x_{(j)} - c_{i(j)} \right]^2 = \sum_{j=0}^{k-1} LUT \left[x_{(j)}, c_{i(j)} \right] \tag{3}$$

LUT can be employed to avoid the subtraction and the multiplication in Equation (1). Hence, it is an efficient method.

Rizvi et. al. proposed another LUT-based scheme in [8] to fasten the calculation of squared Euclidean distances which is called *truncated look-up table* (TLUT). In their method, Rizvi et. al. pre-computed the results of $(x_{(j)}\text{-}c_{i(j)})^2$ for all possible $|x_{(j)}\text{-}c_{i(j)}|$ and stored these results into the matrix *TLUT*, where

$$TLUT = \left[0, 1^2, 2^2, \ldots, (m-1)^2 \right]_{m \times 1} \tag{4}$$

Since the calculation complexity of the absolute subtraction $|x_{(j)}\text{-}c_{i(j)}|$ operation is much simpler and more efficient than multiplication operation, it could be derived before accessing the LUT. Hence, according to the matrix *TLUT*, Equation (1) can be expressed as follows:

$$d(x, c_i) = \sum_{j=0}^{k-1} \left[x_{(j)} - c_{i(j)} \right]^2 = \sum_{j=0}^{k-1} TLUT \left\| x_{(j)} - c_{i(j)} \right\| \tag{5}$$

The size of the matrix *TLUT* is *m*. Yet, it is still difficult for some special designs, such as VLSI implementations and systolic architectures [5,7], to be implemented.

In short, the design criteria of LUT-based schemes emphasize computation speed, table storage and image quality. However, the number of the calculated codewords has not been investigated since these schemes did not use the geometrical information implied in the codewords.

In this work, we propose a hardware-based scheme to represent the positional information. This scheme adopts bit-parallelism and constructs a “Bit-mapped Look-up Table” to prune feasible codewords. The lookup procedure simply involves the use of the bitmap information to prune candidate codewords. Moreover, the scheme is a plug-in, which can cooperate with other existing schemes to further tune up the search speed. An efficient hardware implementation is also presented in our simulation results.

The rest of this paper is organized as follows. The proposed scheme is presented in Section 2. Section 3 addresses the performance evaluation. Section 4 concludes the work.

2. BIT-PARALLEISM SCHEME

As discussed in previous section, VQ matches a test vector to the codeword with the smallest Euclidean distance. The selected codeword could be treated as the nearest point in the *k*-dimensional space. In the ideal case (with a well-trained codebook), the distance of each dimension between the test vector and the selected codeword should be very close. Hence, it is possible to filter out unfeasible codewords by referring to the positional information.

We use a codebook and five different images to show the effect of one-dimensional distances on andthe quality of the compressed images. The codebook is trained by using an image of "Lena" and five other images which are quantized by full search. The image quality is measured by *peak signal-to-noise ratio* (PSNR), which is defined as

$$PSNR = 10 \times \log_{10}(255^2 / MSE) \text{ dB}. \quad (6)$$

The *mean-square error* (MSE) is defined as

$$MSE = (1/H) \times (1/W) \sum_{i=0}^{H-1} \sum_{j=0}^{W-1} \left[\alpha_{(i,j)} - \beta_{(i,j)} \right]^2 \quad (7)$$

for an $H \times W$ image, where $\alpha_{(i,j)}$ and $\beta_{(i,j)}$ denote the original and quantized gray levels of pixel (i,j) in the image, respectively. A larger PSNR value usually preserves the original image quality better.

We shown the distribution of the maximal one-dimensional distance, $max|x_{(j)}-c_{M(i)}|$, where $0 \le j \le k-1$, between the test vectors and their matched codewords for each image in Fig. 1. For high-quality compressed images "Lena" and "Zelda", about 98% of their maximum one-dimensional distances are less than 32. However, the ratio is reduced to 91%~96% for other compressed images with lower quality.

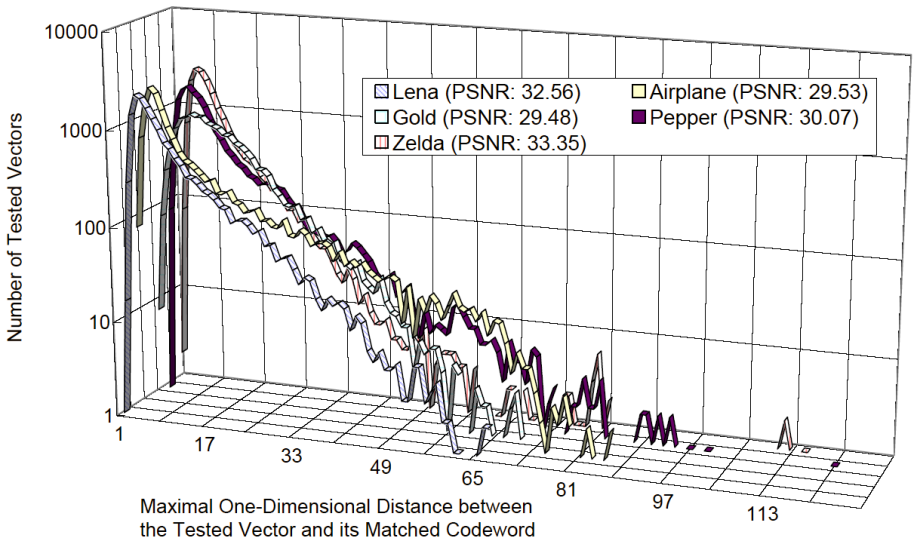


FIGURE 1: Distribution of the Maximum One-dimensional Distances for Different Images.

Figure 1 demonstrates that it is possible to prune feasible codewords by using only one-dimensional distances, especially for those images with high compression quality. We further demonstrate how one-dimensional distances can improve the performance of VQ with an example in Fig. 2. There are two codewords, C_1 (3, 1) and C_2 (2, 3) in this example. To calculate the nearest codeword for the test vector, V_1 (1, 2), the squared Euclidean distances to C_1 and C_2 are 5 and 2, respectively, and C_2 is chosen as the result. If we only consider the horizontal distances, only C_2 would be selected without performing the calculation for the Euclidean distance. However, both codewords are chosen by taking only the vertical distances into consideration, and the Euclidean distances to these two vectors must be calculated in the same way as in the full search to decide the closest codeword. In addition, using one dimension to select the candidate codewords might cause false matches. Let's consider a two-dimensional example in Fig. 2. In Fig. 2, C_2 is closer to V_2 than C_1 in the first dimension. Yet, the Euclidean distance between C_1 and V_2 is smaller than that between C_2 and V_2. Comparing multiple codewords, which are within a certain range in the dimension, could alleviate this problem.

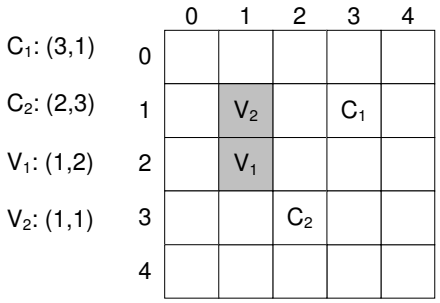


FIGURE 2: A Two-dimensional Example.

In sum, when the codeword is positioned closer to the test vector in the Euclidean space, the distance in each dimension is likely to be shortened as well. To exploit this property, we adopt bitmaps to represent the positional information. For each codeword, there is a uni-codeword bitmap with m bits for each selected dimension *j*, where $0 \le j \le k-1$. Each bit in the uni-codeword bitmap corresponds to a position in dimension *j*. Assume that the pre-defined distance is *D*. The bits from $c_{i(j)-D}$ to $c_{i(j)+D}$ are set to one for codeword *i*.

Figure 3 shows the resulting bitmaps for the example in Fig. 2. The distance *D* is defined as 1 for both dimensions. The set bits form a square in the two-dimensional case. If the test vector is located within the square of a certain codeword, the codeword would be one of the candidates in vector quantization. For example, V_1 is within the square of C_2, rather than C_1. Thus C_1 would not be considered in vector quantization. The bricks, (2,2) and (3,2), are shared by the squares of C_1 and C_2. Therefore, the test vectors positing in these two bricks select C_1 and C_2 as the candidates.

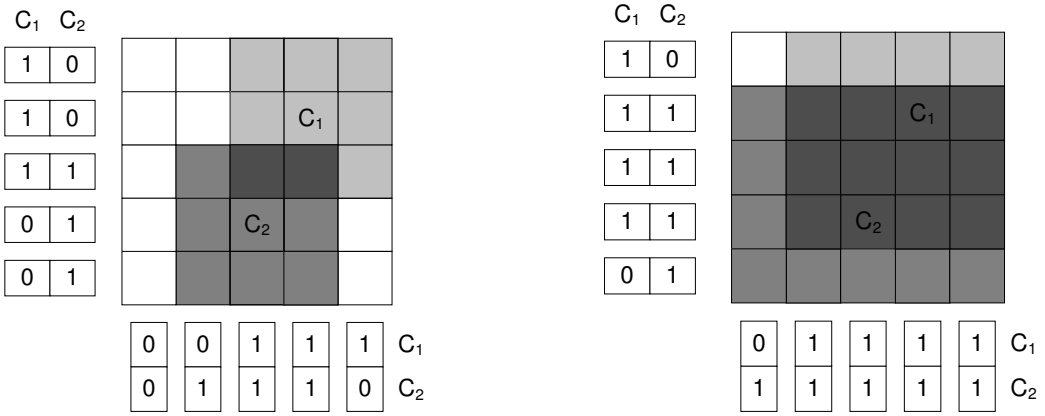


FIGURE 3: Two-dimensional Uni-codeword Bitmaps.

As seen in Fig. 3(a), there are several unoccupied bricks remaining. For the test vectors located within these bricks, e.g. V_2, the bitmaps are useless since there is no candidate could be derived. As a result, each codeword has to be calculated to nail down the one with the smallest Euclidean distance. To ease the problem, a longer distance could be adopted, as shown in Fig. 3(b), where the renewed bitmaps for *D*=2 are presented. With the new distance, most bricks are occupied by at least one codeword's square. However, the conjunct bricks are also increased due to the enlarged squares. A suitable distance is thus important to the performance of the proposed scheme since a wider range could increase the number of candidates, whereas a shorter distance might result in a null set. In our experiments, various distances are investigated to evaluate the performance and the image quality. Next, the construction/lookup procedure of the searchable data structure is introduced.

2.1 The Construction of the Searchable Data Structure - Positional Bitmaps

Although our uni-codeword bitmaps could present the positional information of each codeword, they are not searchable. It is because accessing uni-codeword bitmaps for each codeword would be inefficient. To facilitate the lookup speed, the uni-codeword bitmaps are transformed to uni-position bitmaps, which record the one-dimensional positions of related codewords. The relationships between the uni-codeword bitmaps and the uni-position bitmaps are illustrated in Fig. 4. The uni-position bitmap for position p at dimension j is defined as $B^D_{j,p}$, where D is the preset distance. The i_{th} bit is defined as $B^D_{j,p}(i)$ which is set to one if $p\text{-}D \le c_{i(j)} \le p+D$. The pseudo code is given in Fig. 5. For each bitmap, the required storage is $m \times N$ per dimension. Since there are k dimensions, each bitmap requires $m \times N \times$k bits. Therefore, each bitmap consumes 8 kilobytes for a typical 256-codeword codebook with 256 gray levels.

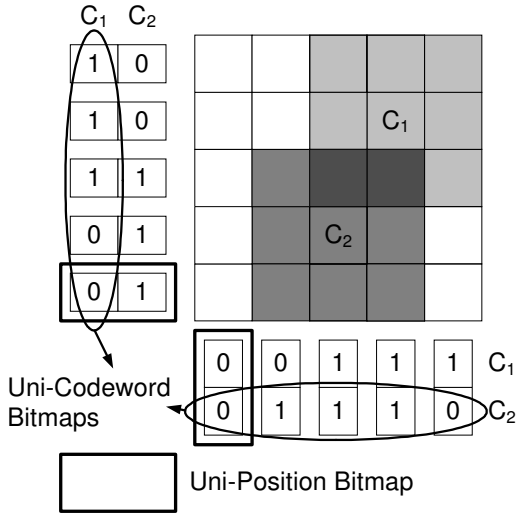


FIGURE 4: Relationships Between Uni-codeword Bitmaps and Uni-position Bitmaps. (D=1)

Bitmap-Filling Algorithm
For each uni-position bitmap $B^D_{j,p}$ **BEGIN**
  For each bit $B^D_{j,p}(i)$ **BEGIN**
    If $p\text{-}D \le c_{i(j)} \le p+D$, $B^D_{j,p}(i)=1$.
    Otherwise, $B^D_{j,p}(i)=0$.
  END
END

FIGURE 5: The Bitmap-Filling Algorithm

2.2 The Lookup Procedure

Our scheme, bit-mapped look-up table (BLUT), combines bitmap pruning and TLUT to achieve fast processing. For a test vector in the BLUT scheme, the j_{th} value x_j is used to access the bitmap B^D_{j,x_j}. Each set bit indicates that the corresponding codeword is within a distance D from the test vector at dimension j. Accordingly, the Euclidean distance is calculated by accessing TLUT. The pseudo code for the lookup procedure is listed in Fig. 6. To check whether the i_{th} bit is set, the fetched bitmap B^D_{j,x_j} is intersected to a pre-generated bitmap with only i_{th} bit set (00…010…0) by performing "**AND**" operation. If the value is larger than zero, then the codeword i is the candidate.

Vector Quantization by BLUT Algorithm
For each vector x **BEGIN**
  Fetch the B^D_{j,x_j}.
  For each set bit $B^D_{j,x_j}(i)$ **BEGIN**
    Calculate Euclidean distance $d(x,c_i)$ where
    $d(x,c_i) = \sum_{j=0}^{k-1} TLUT \left\| x_{(j)}, c_{i(j)} \right\|$.
    If $d(x,c_i) \le min\_distance$ **BEGIN**
      $min\_distance\_id = i$
      $min\_dietance = d(x,c_i)$
    END
  END
  min_distance_id is the quantized index for x
END

FIGURE 6: Vector Quantization by BLUT Algorithm

We use the previous example in Fig. 4 with D=1 to explain the lookup procedure. Assume that the horizontal dimension is used to prune the codebook. For the test vector V_1 (1,2), the second uni-position bitmap (0,1) is fetched. Therefore, C_1 is not considered in the vector quantization and C_2 is the result of vector quantization. However, for another vector V_2 (1,1), a *false match* is caused since only C_2 is selected as the candidate but C_1 is closer to V_2 than C_2. This is mainly due to bitmaps of only one dimension sometimes cover insufficient information.

The first variation is to generate bitmaps for multiple dimensions. With more bitmaps, the effect on codebook-pruning is increased. Furthermore, the multi-dimensional BLUT algorithm can improve the accuracy of image compression since it can cover more information as compared with the one-dimensional scheme. However, the required storage is also increased proportionately. The lookup procedure for multiple bitmaps is shown in Fig. 7. The set *dim* records the selected dimensions where $1 \le |dim| \le k$. Before the set bits are checked, the multiple bitmaps are intersected by performing "**AND**" operation to derive the representative bitmap R^D.

Again, we use the previous example to explain the procedure. For the test vector V_1 (1,2), the second uni-position bitmap (0,1) at the horizontal axis and third one (1,1) at the vertical axis are fetched. Consequently, the bitmap (0,1) is derived by intersecting two bitmaps, and C_2 is selected as the result. To quantize V_2, the resulting bitmap is (0,0), which means no candidates have been found. As a result, the Euclidean distance to C_1 and C_2 are calculated to decide the closest codeword. Although an extra step is required in the computation, *false matches* are eliminated.

Multi-dimensional BLUT Algorithm
For each vector x **BEGIN**
  Fetch the B^D_{j,x_j}, where $j \in dim$
  $R^D = \cap_{j \in dim} B^D_{j,x_j}$
  For each set bit $R^D(i)$ **BEGIN**
    Calculate Euclidean distance $d(x,c_i)$ where
    $d(x,c_i) = \sum_{j=0}^{k-1} TLUT \left\| x_{(j)}, c_{i(j)} \right\|$.
    If $d(x,c_i) \le min\_distance$ **BEGIN**
      $min\_distance\_id = i$
      $min\_dietance = d(x,c_i)$
    END
  END
  min_distance_id is the quantized index for x
END

FIGURE 7: Multi-dimensional BLUT Algorithm

BLUT could also adopt multiple distances to derive a minimum set of candidates. The basic operation is to test the bitmaps with D=1. If there is no candidate, then the bitmaps for D=2 is tested and so forth. In the basic scheme, the time complexity for bitmap operation is O(m). Simple binary search on distance could further improve the performance. That is, the bitmaps for D=m/2 are tested at first. If the candidate set is empty, then the bitmaps for D=m/4 are tested. Otherwise, the D=(m-1+m/2)/2 bitmaps are examined. The time complexity is thus reduced to O($\log_2 m$). However, such scheme requires huge memory space, thus further reduction of storage is necessary.

2.3 Implementations

In the one-dimensional version, BLUT is suitable for software implementation due to its simplicity. Nevertheless, hardware implementation is preferable for multi-dimensional BLUT since it requires memory bus with N-bit wide (typically N=256). In Fig. 8, we present a conceptual model for hardware implementation. This implementation includes independent RAM modules for uni-position bitmaps. Bitmaps of a specific dimension are located in a storage. While performing lookup, the uni-position bitmaps are fetched from RAM modules simultaneously and perform the "AND" operation. Accordingly, the resulting bitmap R^D enables the codewords in the candidate set for calculating the Euclidean distance in ALU.

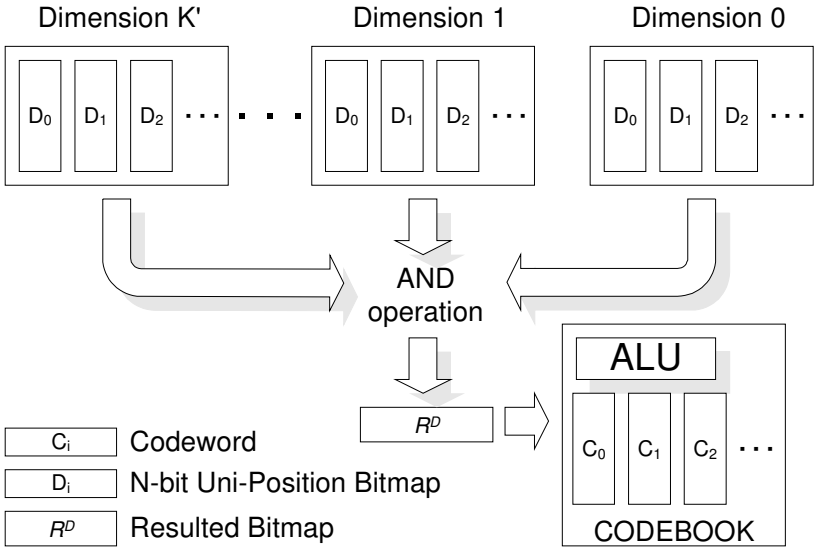


FIGURE 8: Hardware Implementation for Multi-dimensional BLUT.

3. SIMULATION RESULTS

We have conducted several simulations to show the efficiency of BLUT. All images used in these experiments were 512×512 monochrome still images, with each pixel of these images containing 256 gray levels. These images were then divided into 4×4 pixel blocks. Each block was a 16-dimensional vector. We used image "Lena" as our training set and applied the Lindo-Buzo-Gray (LBG) algorithm to generate our codebook C. In the previous literature [1,3], the quality of an image compression method was usually estimated by the following five criteria: compression ratio, image quality, execution time, extra memory size, and the number of mathematical operations. All of our experimental images had the same compression ratio. Thus only the latter four criteria are employed to evaluate the performance. The quality of the images is estimated by PSNR. The extra memory denotes the storage needed to record the projected space and the projected values from codewords onto the projected space. As for the mathematical operations, the number of calculated codewords is considered since the operations for each codeword are identical.

(a) *D*=16 (PSNR=31.329) (b) *D*=32 (PSNR=32.374) (c) *D*=64 (PSNR=32.554)

FIGURE 9: The Lena Images of BLUT.

In the first part, the performance of the software-based implementation with one bitmap was investigated. The experiments were performed on an IBM PC with a 500-MHz Pentium CPU. Table 1 shows the experimental results of BLUT based on image "Lena". VQ indicates the vector quantization without any speedup. The distances for BLUT vary from 16 to 128. With a shorter distance, the image quality is degraded since the occurrence of false matches is increased. Nevertheless, the calculated codewords are reduced by using uni-position bitmap as well as the execution time. Full search requires no extra storage while TLUT needs 256 bytes. For BLUT, the extra storage of a bitmap is 8 kilobytes and 256 bytes for TLUT. The decompressed images are shown in Fig. 9. While BLUT's distance is extended to 64, the image quality is almost identical to VQ and TLUT.

Metrics	Full Search	TLUT	BLUT Scheme				
			D=16	D=32	D=48	D=64	D=128
PSNR	32.56	32.56	31.329	32.374	32.53	32.554	32.56
Execution Time (sec.)	1.72	1.65	0.40	0.77	0.99	1.31	1.84
Calculated Codewords	256	256	50	93	131	165	243
Memory Size (Bytes)	0	256	8192 (Uni-position Bitmap) + 256 (TLUT)				

TABLE 1: The Performance of the Software-based BLUT. (Image: Lena)

Next, we adopt multiple bitmaps to evaluate the performance of the proposed BLUT schemes. Since software platform cannot support wide memory bus, the proposed scheme with multiple bitmaps is based on hardware implementation. The bitwise operation could be parallelized and the time for calculating Euclidean distance is proportional to the number of calculated codewords. In our experiments, we examine the performance with 1, 2 and 4 bitmaps. The distance varies from 16 to 128. In addition, we also present the performance of BLUT with different codebook sizes (128, 256, 512 and 1024) in Tables 2~5.

Table 2 lists the numerical results of vector quantization with 128 codewords. With distance D equals 16, the PSNR value degrades severely as the number of bitmaps increases to 4. This is because a smaller codebook would increase the resulted Euclidean distance of vector quantization as well as the maximal one-dimensional distance. Hence, the effect of BLUT is lessened since greater distance is required. If the distance extends to 32, the resulted image quality by using more than one bitmap is significantly improved. Moreover, only about one fifth of the codewords is calculated for Euclidean distance. BLUT with two bitmaps is usually superior to that with four bitmaps. However, as the number of codewords increases, the 4-bitmap BLUT could outperform the 2-bitmap BLUT by calculating fewer codewords while retaining similar image

quality, as demonstrated in Table 5.

Images		Airplane			Gold			Lena			Pepper			Zelda		
Metrics		Code-words	Time	PSNR	Code-words	Time	PSNR	Code-words	Time	PSNR	Code-words	Time	PSNR	Code-words	Time	PSNR
FS		128	0.54	28.97	128	0.54	28.54	128	0.53	31.46	128	0.55	28.89	128	0.54	32.31
TLUT		128	0.43	28.97	128	0.42	28.54	128	0.42	31.46	128	0.42	28.89	128	0.42	32.31
16	1	23	0.08	28.15	20	0.07	28.12	23	0.08	30.47	21	0.07	24.69	23	0.08	31.89
	2	7	0.02	26.02	5	0.02	27.09	7	0.02	28.30	7	0.02	19.58	7	0.02	29.85
	4	6	0.02	23.12	4	0.01	24.60	6	0.02	24.62	5	0.02	18.62	6	0.02	26.53
32	1	43	0.14	28.78	39	0.13	28.50	43	0.14	31.32	42	0.14	28.49	44	0.14	32.22
	2	22	0.07	28.36	19	0.06	28.45	22	0.07	30.91	22	0.07	27.78	23	0.08	31.91
	4	19	0.06	27.09	16	0.05	28.08	19	0.06	29.81	19	0.06	27.03	20	0.07	31.56
64	1	74	0.24	28.96	74	0.25	28.54	80	0.27	31.46	76	0.25	28.87	80	0.26	32.31
	2	51	0.17	28.96	55	0.18	28.54	59	0.20	31.45	56	0.18	28.85	61	0.20	32.30
	4	49	0.16	28.95	51	0.17	28.54	56	0.19	31.45	54	0.18	28.84	58	0.19	32.28
128	1	114	0.37	28.97	118	0.40	28.54	120	0.40	31.46	117	0.38	28.89	121	0.40	32.31
	2	102	0.33	28.97	112	0.38	28.54	114	0.38	31.46	111	0.36	28.89	117	0.38	32.31
	4	101	0.33	28.97	110	0.37	28.54	113	0.38	31.46	110	0.36	28.89	116	0.38	32.31

TABLE 2: The Performance of the Hardware-based BLUT. (*N*=128)

Images		Airplane			Gold			Lena			Pepper			Zelda		
Metrics		Code-words	Time	PSNR	Code-words	Time	PSNR	Code-words	Time	PSNR	Code-words	Time	PSNR	Code-words	Time	PSNR
FS		256	1.30	29.53	256	1.30	29.48	256	1.30	32.56	256	1.29	30.07	256	1.30	33.35
TLUT		256	1.11	29.53	256	1.10	29.48	256	1.09	32.56	256	1.10	30.07	256	1.09	33.35
16	1	36	0.15	28.62	46	0.20	29.12	50	0.22	31.33	42	0.18	25.95	51	0.22	32.98
	2	11	0.05	26.86	15	0.07	28.27	19	0.08	29.13	15	0.06	21.09	20	0.09	31.39
	4	9	0.04	23.81	10	0.04	25.64	15	0.07	25.66	12	0.05	20.07	16	0.07	29.46
32	1	66	0.28	29.30	88	0.38	29.45	93	0.40	32.37	84	0.36	29.73	97	0.42	33.27
	2	33	0.14	28.93	48	0.21	29.40	54	0.23	32.08	48	0.20	29.12	59	0.25	33.03
	4	29	0.12	28.03	40	0.17	29.21	47	0.20	31.04	42	0.18	28.16	52	0.22	32.59
64	1	134	0.57	29.53	161	0.70	29.48	165	0.72	32.55	160	0.68	30.04	172	0.74	33.34
	2	98	0.42	29.52	128	0.56	29.48	132	0.57	32.55	127	0.54	30.01	141	0.61	33.30
	4	92	0.39	29.52	119	0.52	29.48	125	0.54	32.55	121	0.52	30.00	135	0.58	33.29
128	1	224	0.95	29.53	240	1.04	29.48	243	1.05	32.56	239	1.02	30.07	247	1.06	33.35
	2	207	0.88	29.53	232	1.01	29.48	236	1.02	32.56	230	0.98	30.07	242	1.04	33.35
	4	206	0.88	29.53	230	1.00	29.48	234	1.02	32.56	227	0.97	30.07	240	1.03	33.35

TABLE 3: The Performance of the Hardware-based BLUT. (*N*=256)

Images		Airplane			Gold			Lena			Pepper			Zelda		
Metrics		Code-words	Time	PSNR	Code-words	Time	PSNR	Code-words	Time	PSNR	Code-words	Time	PSNR	Code-words	Time	PSNR
FS		512	2.39	30.00	512	2.41	30.18	512	2.39	33.63	512	2.42	30.57	512	2.41	34.08
TLUT		512	2.15	30.00	512	2.16	30.18	512	2.16	33.63	512	2.14	30.57	512	2.14	34.08
16	1	66	0.28	29.33	90	0.38	29.85	99	0.42	32.69	84	0.35	26.12	103	0.44	33.75
	2	25	0.11	27.70	34	0.14	29.22	42	0.18	30.61	34	0.14	21.53	46	0.19	32.19
	4	19	0.08	24.61	23	0.10	26.77	33	0.14	26.98	27	0.11	20.50	37	0.16	30.46
32	1	126	0.53	29.85	175	0.74	30.15	182	0.77	33.45	167	0.70	30.32	193	0.82	34.05
	2	70	0.30	29.45	103	0.43	30.12	113	0.48	33.03	101	0.42	29.87	124	0.52	33.73
	4	60	0.25	28.43	86	0.36	29.95	100	0.42	31.63	89	0.37	29.12	112	0.47	33.09
64	1	261	1.10	30.00	322	1.35	30.18	326	1.37	33.62	316	1.32	30.56	340	1.44	34.08
	2	197	0.83	29.99	262	1.10	30.18	266	1.12	33.62	258	1.08	30.51	285	1.20	34.08
	4	185	0.78	29.99	245	1.03	30.18	253	1.06	33.62	247	1.03	30.51	274	1.16	34.07
128	1	437	1.85	30.00	479	2.01	30.18	485	2.04	33.63	475	1.99	30.57	495	2.09	34.08
	2	407	1.72	30.00	465	1.96	30.18	472	1.98	33.63	458	1.92	30.57	486	2.05	34.08
	4	402	1.70	30.00	461	1.94	30.18	468	1.97	33.63	454	1.90	30.57	482	2.04	34.08

TABLE 4: The Performance of the Hardware-based BLUT. (*N*=512)

Images		Airplane			Gold			Lena			Pepper			Zelda		
Metrics		Code-words	Time	PSNR	Code-words	Time	PSNR	Code-words	Time	PSNR	Code-words	Time	PSNR	Code-words	Time	PSNR
FS		1,024	4.93	30.43	1,024	4.93	30.80	1,024	4.87	34.42	1,024	4.93	30.83	1,024	4.90	34.51
TLUT		1,024	4.39	30.43	1,024	4.39	30.80	1,024	4.40	34.42	1,024	4.39	30.83	1,024	4.40	34.51
16	1	146	0.63	29.79	178	0.76	30.55	199	0.85	33.59	171	0.73	26.63	205	0.88	34.26
	2	66	0.28	28.39	74	0.32	29.96	98	0.42	31.64	79	0.34	21.70	103	0.44	32.71
	4	51	0.22	25.28	50	0.21	27.56	78	0.33	27.80	62	0.27	20.74	83	0.36	31.17
32	1	269	1.15	30.30	346	1.49	30.78	369	1.58	34.33	337	1.45	30.62	386	1.65	34.47
	2	164	0.70	30.01	218	0.94	30.75	248	1.06	34.05	221	0.95	30.30	269	1.15	34.08
	4	143	0.61	29.50	184	0.79	30.61	221	0.95	33.52	197	0.85	29.43	244	1.05	33.69
64	1	551	2.37	30.43	638	2.74	30.80	658	2.82	34.42	637	2.73	30.80	682	2.92	34.51
	2	440	1.89	30.43	533	2.29	30.80	555	2.38	34.42	538	2.31	30.78	589	2.52	34.50
	4	415	1.78	30.43	500	2.15	30.80	530	2.27	34.42	516	2.22	30.77	567	2.43	34.49
128	1	896	3.85	30.43	959	4.12	30.80	971	4.16	34.42	953	4.09	30.83	987	4.23	34.51
	2	848	3.64	30.43	934	4.01	30.80	948	4.06	34.42	923	3.96	30.83	971	4.16	34.51
	4	839	3.60	30.43	926	3.98	30.80	941	4.03	34.42	914	3.92	30.83	965	4.13	34.51

TABLE 5: The Performance of the Hardware-based BLUT. (*N*=1K)

We also observed that images with a larger codebook are particularly suitable for BLUT. For example, Table 3 shows that the 2-bitmap BLUT scheme could derive good image quality with *D*=32. In Tables 4 and 5, the image quality is further improved while the number of calculated codewords is proportional to the size of codebooks.

From our experimental results, one can also notice that the images with better compression quality can benefit more from BLUT, since the maximum one-dimensional distance within these images could be smaller. For example, the compression quality of the images “Lena” and “Zelda” is better than other images. For these two images, the calculated codewords are more than those of other images at the same distance. Therefore, a well-trained codebook could improve the performance of the proposed scheme.

In summary, the performance of BLUT ties to the size of codebooks and its quality. Thus, BLUT is suitable for compressing high-definition images. We also designed a codebook training algorithm for BLUT by minimizing the deviation of the one-dimensional distance between the image blocks and codewords. As compared with the existing schemes, using a feasible hardware implementation without complex computation makes the proposed scheme suitable for digital home entertainment.

4. CONSLUSION & FUTURE WORK

In this study, we propose a new algorithm BLUT for codebook search. BLUT adopts bitwise data structure to represent the geometrical information. The bitwise representation is simple and suitable for hardware implementation. By setting a given range, BLUT could screen any unfeasible codewords. With the software implementation, only one bitmap is required, thereby minimizing the required storage. On the other hand, the hardware implementation in BLUT could adopt multiple bitmaps to ensure accuracy. Since BLUT relies on an accurate geometrical relation to prune eligible codewords, it is suitable for high-definition image compression. In the future, we will gear toward the design of codebook training algorithms to further improve the lookup performance of schemes like BLUT.

5. REFERENCES

1. R. M. Gray, “*Vector Quantization*”, IEEE ASSP Magazine, 1(2): 4-29, 1984

2. A. Gersho, R. M. Gray. "*Vector Quantization and Signal Compression*", Kluwer (1992)

3. T. S. Chen, C. C. Chang. "*An efficient computation of Euclidean distances using approximated look-up table*". IEEE Transactions on Circuits System Video Technology, 10(4): 594-599, 2000

4. C. C. Chang, Y. C. Hu. "*A fast LBG codebook training algorithm for vector quantization*". IEEE Transactions on Consumer Electronics, 44(4):1201-1208, 1988

5. G. A. Davidson, P. R. Cappello and A. Gersho. "*Systolic architectures for vector quantization*". IEEE Transactions on Acoust., Speech, Signal Processing, 36(10):1651-1664, 1988

6. H. Park, V. K. Prasana. "*Modular VLSI architectures for real-time full-search-based vector quantization*". IEEE Transactions on Circuits System Video Technology, 3(4):309-317, 1993

7. P. A. Ramamoorthy, B. Potu and T. Tran. "*Bit-serial VLSI implementation of vector quantizer for real-time image coding*". IEEE Transactions on Circuits System, 36(10):1281-1290, 1989

8. S. A. Rizvi, N. M. Nasrabadi. "*An efficient Euclidean distance computation for quantization using a truncated look-up table*". IEEE Transactions on Circuits System Video Technology, 5(4):370-371, 1995

9. P. Y. Chen, R. D. Chen. "*An index coding algorithm for image vector quantization*". IEEE Transactions on Consumer Electronics, vol. 49, no. 4, pp. 1513-1520, Nov. 2003

10. Singara singh , R. K. Sharma and M.K. Sharma. "*Use of Wavelet Transform Extension for Graphics Image Compression using JPEG2000 Framework*". International Journal of Image Processing, 3(1): 55-60, 2009.

11. W. S. Chen, F. C. Ou, L. C. Lin and C. Hsin. "*Image coding using vector quantization with a hierarchical codebook in wavelet domain*". IEEE Transactions on Consumer Electronics, 45(1):36-45, 1999

12. Y. C. Hu, C. C. Chang. "*Variable rate vector quantization scheme based on quadtree segmentation*". IEEE Trans. Consumer Electronics, 45(2):310-317, 1999

13. R. C. Chen, C. T. Chan, P. C. Wang, T. S. Chen and H. Y. Chang. "*Reducing computation for vector quantization by using bit-mapped look-up table*". In Proceedings of IEEE ICNSC'2004. Taipei, Taiwan, 2004

14. C. C. Chang, C. C. Chen. "*Full-Searching-Equivalent Vector Quantization Method Using Two-Bounds Triangle Inequality*". Fundamenta Informaticae, 76(1-2):25-37, 2007.

15. Rohmad Fakeh, Abdul Azim Abd Ghani. "*Empirical Evaluation of Decomposition Strategy for Wavelet Video Compression*". International Journal of Image Processing, 3(1): 31-54, 2009.

16. C. C. Chang, C. L. Kuo and C. C. Chen. "*Three Improved Codebook Searching Algorithms for Image Compression Using Vector Quantizer*". International Journal of Computers and Applications, 31(1):16-22, 2009.

17. C. C. Chang, W. C. Wu. "Fast Planar-Oriented Ripple Search Algorithm for Hyperspace VQ Codebook". IEEE Transactions on Image Processing, 16(6):1538-1547, 2007.

18. Hiremath P.S, Jagadeesh Pujari. "*Content Based Image Retrieval using Color Boosted Salient Points and Shape features of an image*". International Journal of Image Processing, 2(1): 10-17, 2008.

Conversion of Commercial Shoe Print to Reference and Recovery of Images

Prof S.Rathinavel rathinavel_s@hotmail.com
Research Scholar, Principal, Excel
College of Engg for Women,
Anna University
Komarapalayam,637303,(Tamilnadu),India

Dr S.Arumugam arumugamdote@yahoo.co.in
Chief Executive, Nandha Engineering College
Anna University
Erode, 638052,(Tamilnadu),India

Abstract

In every criminal investigation it is necessary to determine and prove that a particular person or persons may or may not have been present at the scene of a crime. For this reasons, the collection, preservation and analysis of physical evidence has become more frequent in the law enforcement community[5].

Since criminals must enter and exit crime scene areas, it should therefore, be reasonably assumed that they may leave traces of their own footwear. Criminals have become smarter and wiser by beginning to frequently wear protection over their hands to avoid leaving fingerprints, and masks over their faces to avoid eyewitness identification. However, they are rarely aware of, or make little attempt to conceal footwear. During an every day routine it is normal to see individual wearing gloves, but it's not so over their shoes. In shoe print biometrics one of the important tasks is the preparation of reference data from the available sources. But to use these images for feature extraction and recovery from data bases, it requires certain steps like conversion to gray scale, image enhancement before storage of reference image and image restoration. In this paper we have taken an example shoe image from a commercial web site and converted it to gray scale to reduce the size of storage and then enhanced the image using histogram technique and image is ready for analysis by various techniques. The simulation results are included to validate the method.
Keywords: Image processing, shoe mark, histogram, constrained least squares method, segmentation.

1. INTRODUCTION

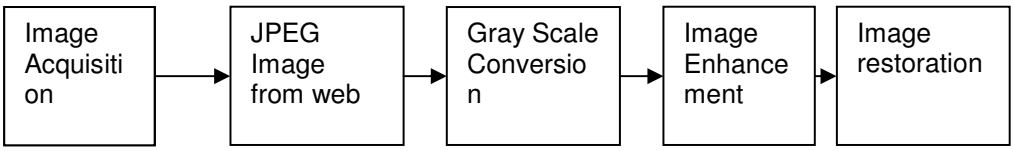
It is generally understood that marks left by a criminal's footwear at a crime scene will be helpful in the subsequent investigation of the crime. For example, shoeprints can be useful to likely link crime scenes that have been committed by the same person. They can also be used to target the most prolific criminals by allowing officers to be alerted to watch out for particular shoe marks. Also, shoeprints can provide useful intelligence during interviews so that other crimes can be brought to a criminal. Finally, they can provide evidence of a crime, if a shoeprint and suspect's shoe match[4][5][7].

Sole Mate, Foster & Freeman's database of shoeprints, proved its worth recently during the investigation of a murder of a woman in the kitchen of her Coventry home in U.K. Officers from West Midlands police station was confirmed by this database that the suspect's shoeprint has been produced by a Mountain Ridge™ shoe, unique to JJB Sports, a UK nationwide shoe retailer, and that there were two models using that particular sole, and this footwear was first available during the summer of 2002[5][7][8]. With this information, police officers were able to locate the store in which the footwear was purchased and to trace the shop's copy of the actual receipt issued to the suspect, later confirmed when an empty shoe box was found at the suspect's home.

Unfortunately, when crime scene is improperly secured or is disorganized, the search of the scene often results in this type of impression evidence being overlooked or destroyed. When this type of physical evidence is properly collected and preserved by the crime scene investigator, followed up by a detailed examination by a footwear expert, it can become an important part in proving or disproving a suspect was at the crime scene.

In shoe print biometrics, the major task is the preparation of reference data from the available sources. Normally reference shoe prints of branded commercial shoes are available in the web. But to use these images for feature extraction and recovery from data bases requires certain steps like conversion to gray scale, image enhancement before storage of reference image and image restoration[2][9]. So that many feature enhancement techniques can be used to index, identify the candidate shoe print. A shoe image from a commercial web site is taken and converted it to gray scale to reduce the size of storage, then enhanced the image using histogram technique[2]. Now image is ready for feature extraction by various techniques. The enhanced image is restored back using Wiener filter and constrained least square restoration methods so that it can be identified from the reference storage.

Section 2 gives the Block diagram. Section 3 deals with conversion from jpeg to bmp and gray scale conversion, section 4 with image enhancement using histogram method, section 5 with Image Restoration using Wiener filter and constrained least square restoration methods[1][2]. Relevant results are provided to illustrate the working of the algorithm.
The block diagram is as shown below:



The raw jpg image is converted to bmp format using image converter plus then gray scale conversion is carried out using MATLAB routines. Then image enhancement and restoration are carried out.

2. IMAGE ACQUISITION:

Footwear evidence can be found in two forms, impressions and prints[3][5][7]. The impression is normally described as three-dimensional, such as an impression in mud or a soft material; and the print is described as a print made on a solid surface by dust, powder, or a similar medium.

Footwear evidence, as well as latent fingerprint evidence, is classified into three categories of crime scene prints:

1. Visible prints
2. Plastic prints
3. Latent prints

The Visible Prints:

A visible print occurs when the footwear steps into a foreign substance and is contaminated by it, and then comes in contact with a clean surface and is pressed onto that surface. This print can be visibly seen by the naked eye without any other aids. The most common visible prints are prints left on a contrasting surface, such as kitchen floor, etc..

A variety of substances, such as blood, grease, oil or water will leave contrasting prints. This type of print must be photographed, prior to any other methods being used. An electrostatic dust lifter can also be utilized when the evidence is in dust.

The Plastic Prints:

Plastic prints are impressions that occur when the footwear steps into a soft surface, such as deep mud, snow, wet sand, or dirt creating a three-dimensional impression. This type of impressions should be photographed and then cast. These types of impressions are three-dimensional because they allow the examiner to see length, width, and depth[4,6].

The Latent Prints:

Latent prints are the most overlooked print that are generally found on smooth surfaces. They can be developed by the same way as latent fingerprints. This type of print needs a variety of powders, chemicals and even forensic light sources to make it visible in order to properly be collected. In most cases these prints should also be photographed prior to any recovery process.

However one of the sophisticated methods has been chosen for the investigation and is taken for the comparison with the known set of patterns that are collected from 50 shoes. These prints are scanned into images at a resolution of standard set of ***dpi*** values.

The commercial database gives shoe prints in JPEG/JPG format and to run our program we need it in *.bmp format and hence use image converter plus to convert from jpg to bmp.
The image used for the current study is shown in fig 1

FIGURE 1: Image considered for study

The gray scale conversion is carried out using the MATLAB routines *rgb2ind* and *ind2gra* [1][10]. The output of such a conversion is shown in figure 2

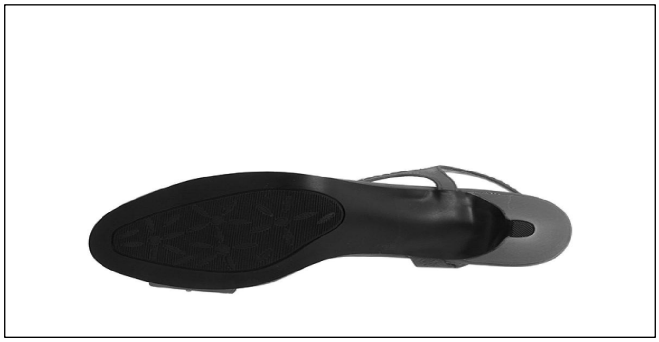


FIGURE 2: Gray scale image under study

3. IMAGE ENHANCEMENT:

Since various conversions are done on the image to carry out meaningful feature extraction it is essential we do image enhancement. In this study we used the histogram technique to carry out the image enhancement[2,6].

The following are the features of the histogram algorithm:

- x0: Converted JPEG input image into *.bmp as a gray level image
- y0: histogram equalized image, each pixel has 8 bits
- L: [0 to L-1] is the number of gray levels. Default L = 256

Algorithm:

1. Compute the histogram, and then equalized histogram of x0 which is the input image
2. Point-wise gray-level transform according to equalization

The following MATLAB routines were used *hist, round, reshape* in this algorithm

A MATLAB program was written and the output of this stage is given in figure 3 and figure 4.

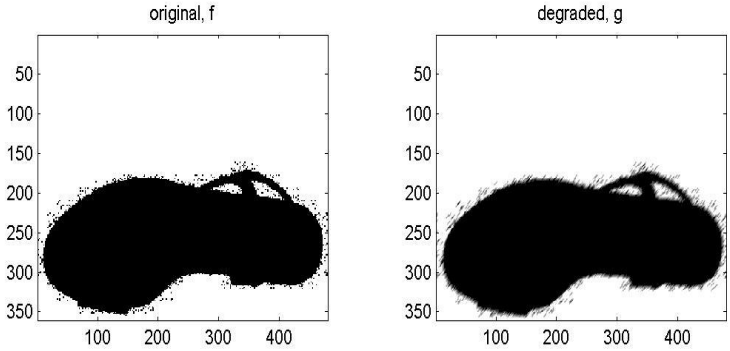


FIGURE 3 Enhanced Images

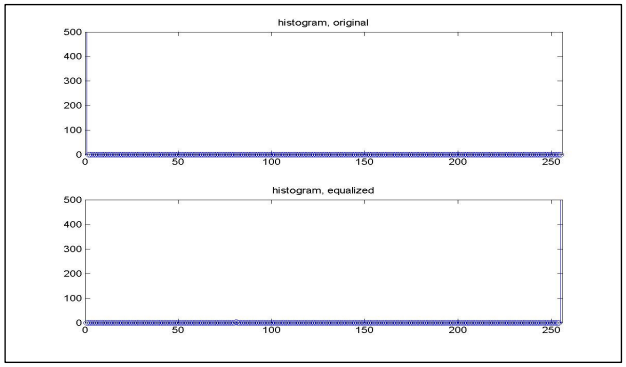


FIGURE 4: Histogram values

4. IMAGE RESTORATION:

We carried out the image restoration to understand whether the enhanced signal can be retrieved back from the data base also for carrying out meaningful Feature extraction method. We applied Wiener and Constrained Least Squares method.

Algorithm:

Step 1. Load the image enhanced image I

Step 2. Create -1*(x+y)mask

Step 3. Find FFT of I

Step 4. Note that fft does not divide the result by mn
Step 5. Create blurring mask
Step 6. Generate Gaussian noise with PSNR = 30db, 60db,etc as per need
Step 7. Fourier transform of the noiseless blurred image is obtained
Step 8. Calculate RMS error per pixel
Step 9. Calculate FFT of blurred+ noise image
Step 10. Do Inverse filtering
Step 11. Apply Wiener filtering
Step 12. compare both filtered image we can repeat with different values PSNR values

MATLAB code was written and comparisons of the all stages of images are shown in figure 5. *Motionblur* and *fft2* are two important MATLAB functions used in this calculation. PSNR of 60 db is used and restored images were obtained with different iterations.

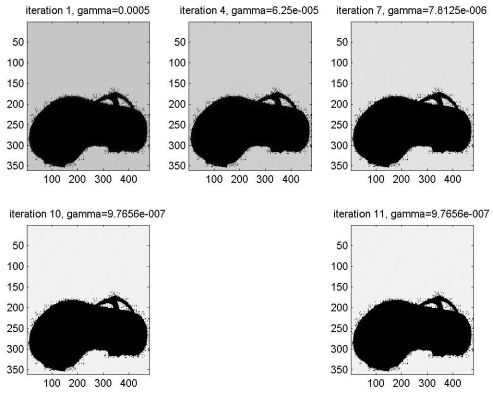


FIGURE 5: Comparison of all stages

5. CONCLUSION:

Thus we are able to demonstrate the successful preparation of raw images of the shoe print available in commercials databases. In these days of security threat it is an important activity so that security requirement of the country can be beefed up through preparation of such databases of shoe prints and these papers demonstrate one such method of preparation. The results show that our method is one of the candidates for such a database preparation along with we can use many feature extraction indexing techniques to store and retrieve from the database the identified shoe print.

6. REFRENCES:

[1] Raefel C.Gonzalez, Richard E. Woods, Steve L. Eddins, "*Digital Image Processing using MATLAB*",2nd Edition, May 2005.

[2] Anil .K.Jain, "*Fundamentals of Image Processing*", Prentice Hall, 2004.

[3] A. Alexander, A.Bouridane, Crookes, D. "*Automatic Classification and Recognition of Shoeprints*", Proc. of Seventh Int'l Conf. on (Conf. Publ. No. 465) Image Processing and Its Applications, vol. 2, 1999.

[4] W.Ashley, "*What Shoe Was That? The User Computerized Image Database to Assist Identification*, "Forensic science Int'l, vol.82, 1996

.

[5] W.Bodziac."*Footwear Impression Evidence: Detection, Recovery and Examination*". CRC Press, 2000.

[6] N.Otsu, "*A threshold selection method from gray level Histogram*", IEEE Transactions on Systems, Man and Cybernetics,9:62-66,1979.

[7] Hilderbrand, Dwayne. S , "*Footwear, The Missed Evidence*" Staggs Publishing, 1999

[8] "*Imaging for Forensic And Security*", Springer U.S Publisher, ISBN 978-0-387-09531(print)978-0-387-09532-5(online), www.springerlink.com.

[9] Chandra Sekhar Panda & Prof. (Dr.) Srikanta Patnaik, "*Filtering Corrupted Image and Edge Detection in Restored Grayscale Image Using Derivative Filters*, "International Journal of Image Processing, (IJIP) Volume (3) : Issue (3), 2009

[10] W.K.Pratt, "*Digital Image Processing*", John Wiley and Sons, 1992

[11] Ernest.L. Hall, "*Computer Image Processing and Recognition*", Academic Press, 1979

Parallelization Of The LBG Vector Quantization Algorithm For Shared Memory Systems

Rajashekar Annaji annaji.rajashekar@iiitb.ac.in
International Institute of Information Technology
Bangalore, 560 100
India

Shrisha Rao srao@iiitb.ac.in
International Institute of Information Technology
Bangalore, 560 100
India

Abstract

This paper proposes a parallel approach for the Vector Quantization (VQ) problem in image processing. VQ deals with codebook generation from the input training data set and replacement of any arbitrary data with the nearest codevector. Most of the efforts in VQ have been directed towards designing parallel search algorithms for the codebook, and little has hitherto been done in evolving a parallelized procedure to obtain an optimum codebook. This parallel algorithm addresses the problem of designing an optimum codebook using the traditional LBG type of vector quantization algorithm for shared memory systems and for the efficient usage of parallel processors. Using the codebook formed from a training set, any arbitrary input data is replaced with the nearest codevector from the codebook. The effectiveness of the proposed algorithm is indicated.

Keywords: Vector Quantization, Multi-core, Shared Memory, Clustering.

1 INTRODUCTION

Vector Quantization is an extension of the scalar quantization to multi-dimensional space [7], which is a widely used compression technique for speech and image coding systems [4]. It is similar to the clustering procedure known as K-means algorithm [10] which is used in pattern recognition. It is also useful in estimating the probability distributions of the given featured vectors over a higher dimension space.

For Data compression using Vector Quantization, the two main distinguishable parts are codebook generation and replacement of arbitrary data with the obtained codebook. This paper addresses the parallel implementation of the codebook generation part.

Vector quantization involves the comparison of vectors in the input training sequence with all the vectors in a codebook. Owing to the fact that all the source vectors are compared with the same codebook and

the source (training) vectors are mutually exclusive in operation, division of work can be clearly visualized and parallelization of the algorithm can be achieved.

The LBG algorithm used for Vector Quantization needs to be modified by exploiting the inherent parallelism, which should lead to the improvement in the usage of processors. And this will in turn reduce the time taken to generate the codebook for LBG algorithm.

This paper presents a parallel VQ algorithm based on the shared memory architecture and uses the master/slave paradigm to optimize two main bottlenecks of the sequential version:

- Time taken for the design of optimum codebook.
- Efficient distribution of work to the parallel processing units.

Taking the advantage of shared memory architecture by assigning equal chunks of input training data to all the processors and having a copy of codebook in the primary memory of each processor, the 'nearest codevector identification' which is the most time consuming part in the LBG algorithm is performed faster. The reduction in computation time does not result in the increase in distortion. Distributing the training samples over the local disks of slaves (processors) reduces the overhead associated with the communication process.

The 'cell table' which is formed after the integration by the master is stored in shared memory and is used in the Updating procedure. This is important for obtaining optimum codebook for the given input training data. Processors work parallel, updating a single codevector at any point. Assigning of the codevector to a processor is done randomly.

The paper is organized as follows. Section 2 describes the related work of vector quantization in the field of data compression and also provides some background about LBG and terms used in the algorithm. Section 3 describes the proposed Parallel implementation of Vector Quantization algorithm. Section 4 describes the performance issues of the above proposed algorithm and speedup of the system. Section 5 describes the results and simulations of the algorithm using OpenMP. Section 6 concludes the work done and describes some of the future work.

Related Work

Considerable work is being done in the fields of image compression, speech compression based on vector quantization and codebook generation. Some of the algorithms that have been used for sequential VQ codebook generation are LBG, pair wise nearest neighbor (PNN), simulated annealing, and fuzzy c-means clustering [10] analysis. The compression method2 used for this is based on a block encoding scheme known as vector quantization [1, 9, 10]. The last decade has seen much activity in exploring limits and applications for vector quantization. Currently, parallel processing is being explored as a way to provide computation speeds necessary for real time applications of image compression techniques [4].

Codebook design plays a fundamental role in the performance of signal compression systems based on VQ. The amount of compression is defined in terms of the rate, which will be measured in bits per sample. Suppose we have a codebook of size N , and the input vector is of dimension L, in order to inform the decoder of which code-vector was selected, we need to use $log2N$ bits. Thus the number of bits per vector is $log2N$. As each codevector contains the reconstruction values for L source output samples, the number of bits per sample would be $\frac{Log_2 N}{L}$. Thus, the rate for an L dimensional VQ with a codebook size of N is $\frac{Log_2 N}{L}$.

The main problem related to VQ is that the training process to create the codebook requires large computation time and memory, since at each new experiment to evaluate new feature sets or increase in

the database for training the HMM's(Hidden Markov Models), it is necessary to recreate the codebooks. Parallelism operates on the fact that large problems can almost always be divided into smaller ones, which may be carried out concurrently. Based on this principle, an algorithm for parallelizing of Vector Quantization (VQ) is proposed, which when applied on a Shared Memory system like a Multi-core system guarantees a better and faster initialization method for LBG codebook design.

Codebook consists of a set of vectors, called codevectors. Vector quantization algorithms use this codebook to map an input vector to the codevector closest to it. Data compression, the goal of vector quantization, can then be achieved by transmitting or storing only the index of the vector. Various algorithms have been developed to generate codebooks. The most commonly Known and used algorithm is the Linde-Buzo-Gray (LBG) algorithm.

Parallel versions of the LBG Vector Quantization algorithm have been proposed by many and most of them have been applied to Distributed systems where the overhead of process communication is present [4, 7, 6]. The speed and execution time of the algorithm depends on these communications mechanisms, which are minimal in case of shared memory architectures. Though some parts of these parallel implementations are similar, the speed and effectiveness depends upon the architecture used and efficient usage of system processors and memories.

Algorithms for Codebook Generation

Various algorithms have been developed for codebook generation. Some that have been used are the LBG, pair wise nearest neighbor (PNN), simulated annealing and the fuzzy c-means (FCM) clustering algorithms.

1. LBG Algorithm [1, 9]: This is an iterative clustering descent algorithm. Its basic function is to minimize the distance between the training vector and the code vector that is closest to it.

2. Pair wise Nearest Neighbor Algorithm [3]: This is a new alternative to the LBG algorithm and can be considered as an initialize for the LBG algorithm. For efficient execution of this algorithm, the two closest training vectors have to be found and clustered. The two closest clusters are then merged to form one cluster and so on.

3. Simulated Annealing Algorithm: This algorithm aims to find a globally optimum codebook as compared to the locally optimum codebook that is obtained using the LBG algorithm. Stochastic hill climbing is used to find a solution closer to the global minimum and escape the poor local minima.

4. Fuzzy C-Means Clustering Algorithm [10, 9]: The LBG and PNN partition the training vectors into disjoint sets. The FCM algorithm assigns each training vector a membership function which indicates the degree to which it will belong to each cluster rather than assigning it to one cluster as is done in the LBG and PNN algorithms.

Huang compares these algorithms [9]; the following are some of his conclusions.

- PNN is the fastest but has higher distortion than the LBG algorithm [3].

- Simulated Annealing produces the best distortion results but requires substantially greater CPU time and there is no significant improvement in the quality of the reproduced images.

- The FCM algorithm has worse results and is slower than the LBG Algorithm.

Background

Looking at the different codebook generations and their drawbacks, the algorithm chosen for parallelizing was the LBG algorithm. An attempt has been made to decrease the time for codebook generation while maintaining the distortion measures obtained from the sequential version of the LBG algorithm. The next section describes the LBG algorithm in more detail which is taken from the original Vector Quantization paper [1].

LBG Algorithm

- Initialization: Given N = number of levels, distortion threshold $\epsilon \geq 0$, an initial N - level reproduction alphabet $A0$ and a training sequence $\{xj : j = 0... n - 1\}$. Set $m = 0$ and $D-1 = \infty$.

- Given $A_m = \{yi : i = 1....N\}$ find the minimum distortion partition $P(A_m) = \{Si : i = 1...N\}$ of the training sequence: $xj \in Si$ if $d(xj, yi) \leq d(xj, yl)$ for all l. Compute the average distortion

$$Dm = D(\{A_m, P(A_m)\}) = n^{-1} \sum_{j=0}^{n-1} min_{y \in A_m} d(x_j, y).$$

- If $\frac{D_{m-1} - D_m}{D_m}$ halt with Am final reproduction alphabet. Otherwise continue.

- Find the optimal reproduction alphabet $x(P(Am)) = \{x(Si) : i = 1,...,N\}$ for $P(Am)$. Set $A_{m+1} = x(P(Am))$. Replace m by $m + 1$ and go to 1.

Terms used:

- Codebook: It is a collection of codevectors, and these codevectors can be stated as the quantization levels. In LBG algorithm number of codevectors can be only in powers of 2.

- Centroid: Centroid is nothing but the average of the input vectors in the particular region specified. The dimension of the centroid is same as Input training vector k

- Partial Cell Table: It is the part of the 'cell table' and it indicates the allocation of input vectors to the corresponding minimum distortion codebook. Against each index it stores the codevector number from which the corresponding input vector has the minimum distance (Euclidean distance). Each processor has its own partial cell table.

- Cell Table: After all the processors compute the minimum distortions and partial cell tables are formed, they are integrated to form the final 'cell table' and the values in the table are called as cell value.

- Distortion: The Euclidean distance between the input training vector and the codebook vector gives the distortion value. It helps indentifying the nearest codebook.

2 System Model for Parallel implementation of Vector Quantization

This parallel algorithm can be extended to shared memory architectures where in, all the processor cores have their primary cache memory, also called as $L1$ cache. And a single shared memory also called as $L2$ cache, which is accessible by all the cores by some means inter-process communication. The input training data is available in the shared memory and hence can be shared by all the processors.

Notations

We assume that the initial training data which is used to train the system is of the form of a $'M \times k'$ matrix where $'k'$ is the vector dimension. This can be any numerical data, which is obtained by processing an image or a speech signal.

- Number of Input Training vectors : M
- The dimension of Input data: $1 \times k$
- Codebook size: $N \times k$, where N is the number of codevectors
- Number of processors: P

The set of training vectors are
$T = \{X1, \ldots, XM\}$

And each input vector is denoted by,

$Xm = \{xm1, \ldots, xmk\}, m = 1, 2, \ldots, M$

$C = \{C1, \ldots, CN\}$

represents the codebook. Each codevector is k -dimensional, e.g.

$Cn = \{cn1, \ldots, cnk\}, n = 1, 2, \ldots, N$

Algorithm Implementation

The parallel algorithm is given as follows.

Functions used:

Centroid (Input) → The average of input.

CodebookIndex (MinDist) → this function gives the index of the codevector to which the input is nearest and the argument is “Minimum Euclidean distance”.

CommunicateToMaster () → All the processors communicate to the master to integrate the parallel sections.

IntegratePartialCellTables () → the partial cell tables formed by individual processors are integrated to form the final 'cell table'.

IntegrateDistortions () → the distortion value obtained by each processor for its set of input vectors allocated is communicated to the master, which does the integration of all those distortions.

ExtractVectors(CellTable, index) → the values in the 'CellTable' are the indexes of codevectors. From the 'CellTable' extract all the input vectors which belong to a particular codevector denoted by its index.

NewCodevector (extracted vectors) → the centroid of the extracted vectors gives the updated codevector.

Input: Any data of dimension $M \times k$

Output: Codebook of dimension $N \times k$

```
1    MasterSelection ();
2        → compute centroid (input);
3        → Distribute inputs to Processors
4     Codebook Splitting:
5        foreach centroid do [centroid + δ], [centroid − δ]

6    Cell Allocation:
7        → parallel execution at each processor with processor ID ρ = 0, 1, ..., P − 1;
8        {
9        for z ← [ρ × (M)/P + 1] to [(ρ + 1) × M/(P)] do
10           MinDist ← minimum(E.Dist[input(z), codebook(0)], E.Dist[input(z), codebook(1)]...);
11           index ← CodebookIndex(MinDist);
12           D_ρ ← D_ρ + MinDist ;
13           PartialCellTable (z) → index ;
14        end
15       }

16   CommunicateToMaster ();
17    celltable () → IntegratePartialCellTables ();
18   TD_ρ ← Integrate Distortions ();
          (TD_ρ − TD_ρ−1) / TD_ρ ≤ €
19   if                              then
20       if Vectors == N (required number of codevectors) then
                TERMINATE;
21        else go to CodebookSplitting step
22   end
23   else Updation step:
24   → parallel execution at each processor;
25   {
26   for j ← 1 to M do
27       Extract Vectors (CellTable, index );
28       NewCodevector (index ) ← centroid (Extracted Vectors );
29   end
30    }
```

31 go to *Cell Allocation* step;

Algorithm 1: Parallelized Vector Quantization Algorithm

The following are the important steps in the Algorithm:

1. Initialization
2. Codebook Splitting
3. Cell Allocation
4. Updation

Initialization:

- One of the processors is chosen as the master either by leader election or randomly.
- The master computes the centroid of the initial training data and it is assigned as the initial codebook.

Input Training Set				
1	a_{11}	a_{12}	.....	a_{1k}
2	a_{21}	a_{22}	.....	a_{2k}
:				
:	a_{m1}	a_{m2}	.....	a_{mk}
M	*Centroid*			

Table 1: Centroid

In Table $\mathbf{1}$, the centroid $(1 \times k)$ is the average of all the input training vectors of the dimension $M \times k$, and forms the initial codebook.

- The master allocates the training vectors equally among all the slaves. The number of vectors to each slave are $\left\lfloor \frac{M}{P} \right\rfloor$
- $D - 1$, which is the distortion value, is initialized to a very high positive value.
- The threshold € decides the termination condition of the algorithm.
- The splitting parameter $'\delta'$ is initialized to a constant value.

Codebook Splitting:

The master doubles the size of the initial codebook. The increase in the number of codevectors is done by getting two new values, by adding and subtracting δ (which may be considered a variance, and is constant throughout the algorithm), to each centroid value. Therefore, the codebook splitting step generates a new codebook which is twice the size of the initial codebook.

$$[Centroid + \delta], [Centroid - \delta]$$

The codebook so formed in this splitting step is duplicated into the primary memory of all the slave processors.

Cell Allocation:

Each slave calculates the Euclidean distance between the input vectors allocated to it and the codebook vectors. Based on the Euclidean distance, it finds the nearest codebook vector to each input vector and allocates the particular input vector to the codebook vector.

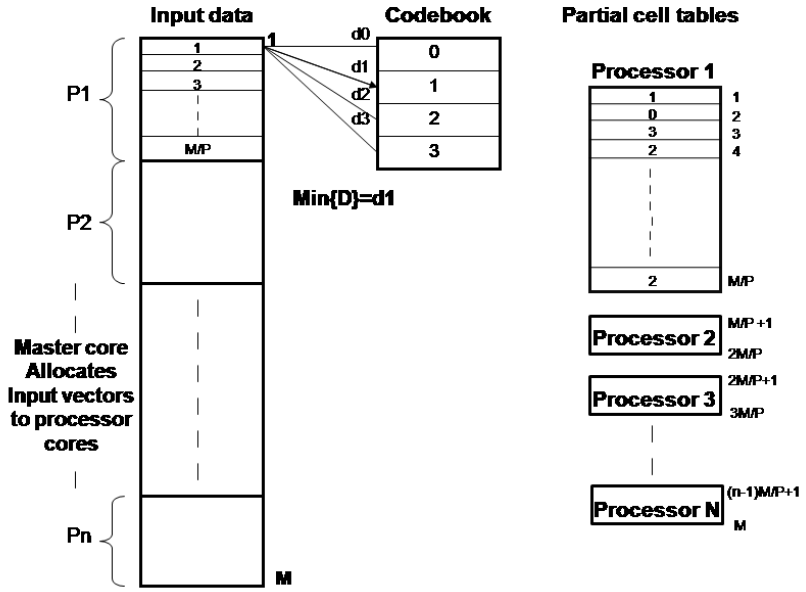


Figure 1: Input data allocation to processors and nearest codebook, cell table

In Figure 1, the initial training data is divided equally among all the processors

$$P = \{P1 \ldots Pn\}.$$

$$processor1 \leftarrow \left[1 \; to \; \frac{M}{P}\right], \quad processor2 \leftarrow \left[\left(\frac{M}{P} + 1\right) to \; 2M\right], \ldots processorN \leftarrow \left[\left(\frac{(n-1)M}{P} + 1\right) to \; M\right].$$

Whenever a new codebook is formed, codevectors are duplicated into primary cache $(L1)$ of all slave processors. The Euclidean distance between each input vector allocated to the processor and all the

codevectors is computed, minimum of all is taken and added to $'DPi(Distortion)'$. And the index of the codevector nearest to the input is placed in the corresponding location in the 'partial cell table' $\left(\frac{M}{P} \times k\right)$. Finally when all the allocated training vectors are computed, the 'partial cell table' and 'distortion' have to be communicated to the master, and this process is done by every other processor executing in parallel.

For each slave processor, the distortion value is $Di = \sum min\{D\}$ where $D = \{d1, \ldots, dr\}$ is the set of Euclidean distortions for each input vector with respect to the codevectors. And the corresponding 'distortion' values and 'partial cell tables' will be communicated to the master by all the slaves, and the master computes the 'total distortion' TDi and also integrates the individual cell table to form a 'final cell table' which is used for updating codebook.

The total distortion computed by the master is

$$TDi = \sum\{Di\}$$

The threshold value € which is initialized previously to a constant value is used to identify the termination of the algorithm. The termination condition for the algorithm is,

$$\frac{(TD_i - 1 - TD_i)}{(TD_i - 1)} \leq €$$

If the condition is satisfied, it implies that the optimum codebook for that level is formed for the given input training data set. And if the number of codevectors formed in that state is equal to the size of the codebook specified, terminate the program. Else go back to the Codebook Splitting step and proceed with the same.

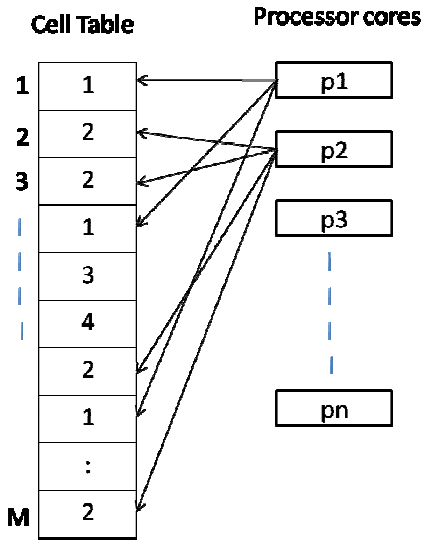


Figure 2: Parallel updation of codebook by slave processors

Codebook Updation:

If the Threshold condition is not satisfied then it implies that the codebook so formed is not the optimum one and needs to be updated. The codebook is updated by replacing each codevector by the centroid of all the input vectors allocated to it. The new centroids are calculated in parallel wherein with set of input vectors corresponding to one codevector are computed by a single processor. This procedure is explained from Figure 2.

In Figure 2, the cell table contains the indexes of codevectors to which the corresponding input vector is the nearest. A single processor updates single codevector at any point. The process of updation is:

- Extract all the input vectors which have the value 0 in the cell table. These are the set of input vectors which are nearest to codevector1.
- Compute the centroid of the vectors which have been extracted. This forms the new codevector.

Hence, if we have P slave processors available, then at any point of time P codevectors will be updated in parallel, and then the next round of updation proceeds and so on up to the number of codevectors, which executes in a round robin fashion. And the assigning of P codevectors to the P processors can be done randomly or serially. In the case of serial updation, if the size of the codebook in that stage is $'S'$, then S number of codevectors must be updated. If $CodevectorIndex \% P == \varphi$, implies processor φ performs the updation of that codevector. Once all the codevectors are updated go back to Cell Allocation step and continue the iterative procedure until the required numbers of codevectors are generated.

Performance

From Algorithm 1 described in the Section 3 it can be observed that the parallel processing tasks are identified separately. And from our experimental analysis, in the sequential version of Vector quantization using LBG algorithm, these parts of the program which can be parallelized were extracted out as separate process blocks (functions). And using a GNU gprof profiler, which helps in knowing where the program has spent most of its time and which process is called by which other process while it is executing, it is observed that parallel blocks consume 80 *to* 85% of the overall time and also provides the time consumed by individual parallel blocks. This parallelization can be accounted to two of the main processes in the algorithm.

1. Generation of 'Cell Table' also called as 'Allocation table,' i.e., allocating the input training vector to the nearest codebook vector based on Euclidean distance, which is a highly time consuming computation.

2. Calculation of Centroids in the Updation of codebook step.

The first step is the most time-consuming part and takes about 70 *to* 75% time of the total sequential part. In this step the entire input training data is divided into number of chunks equal to number of processors and allocated to them. As each processor has its own duplicated version of the codebook, further computations are done in parallel by the processors. Hence, the theoretical efficiency of the multi-core system or to say processor utilization' would be 100%. The second step, which is calculation of centroids in the updation step, takes about 10 *to* 15% of the sequential part. According to Amdahl's law:

$$speedup = \frac{1}{\left(S + \frac{(1-S)}{n}\right)}$$

Where S is time spent in executing the serial part of the parallelized version and n is the number of parallel processors. In the proposed algorithm, 85% of it can be parallelized, hence the time spent in executing serial part is 15% and assuming $n = 4$, the speedup of the parallel version is

$$\frac{1}{0.15 + \frac{1 - 0.15}{4}} = 2.76$$

i.e., a quad-core system with has 4 cores would work 2.76 times as fast as the single processor system.

Results and Simulation

The proposed algorithm has been simulated using OpenMP with the training input size of 2000 vectors with the dimension varying from 2 to 10 .These results have been compared with the standard sequential execution and the results have been plotted. The graph clearly indicates the expected reduction in time consumed to create the codebooks.

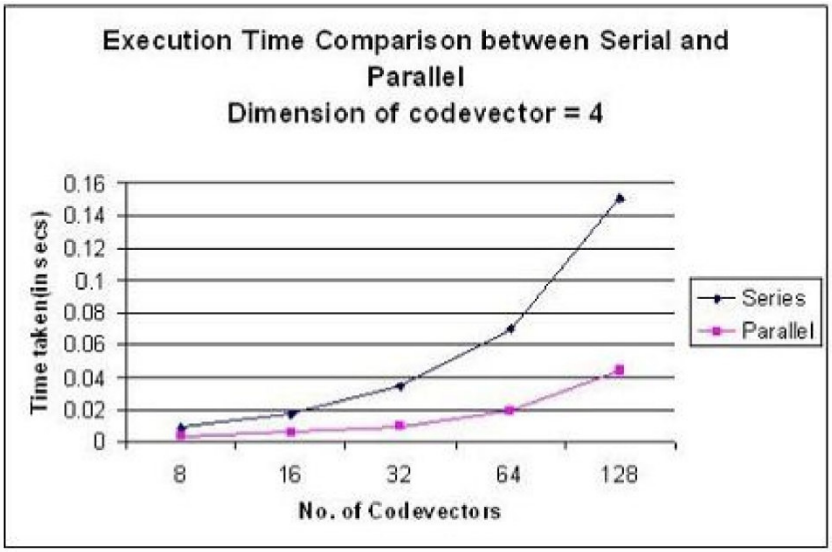


Figure 3: Execution Time Comparison

Figure 3 is the graph of the number of codevectors vs time taken to create the codebook. The number of processors is fixed at 4 and the number of codevectors is varied from 8 to 128 . The results are plotted both for sequential as well as parallel algorithm. The difference is clear when the size of the codebook increases.

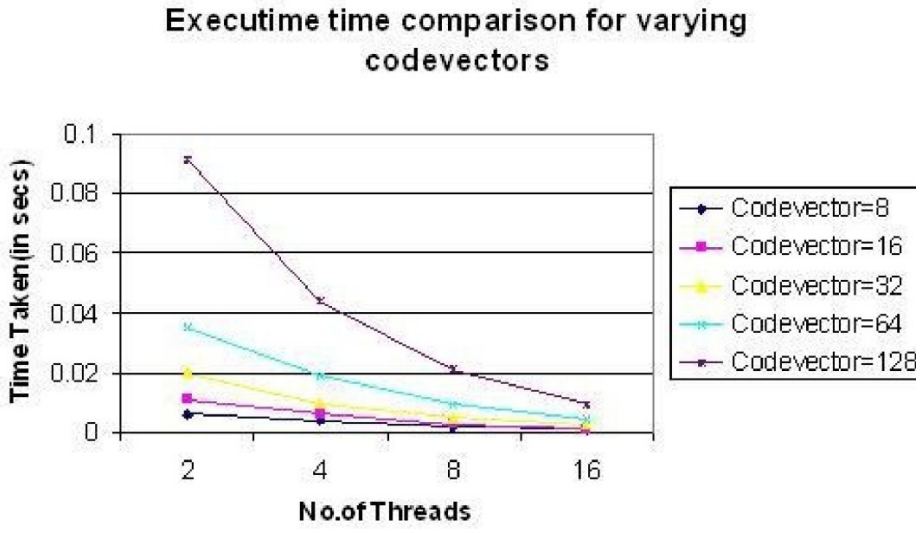


Figure 4: Plot of Time Taken vs. Number of threads

3 CONCLUSION & FUTURE WORK

In this paper a parallel implementation of LBG Vector quantization is demonstrated. The algorithm proposed here is general in the sense that it can be applied on any shared memory architecture irrespective of the underlying inter process communication. The performance of the proposed algorithm is analyzed and also experimentally simulated using the OpenMP shared programming. When compared with the sequential version of the LBG, the proposed algorithm has better performance in terms of speed of execution, and the study of execution time is done for varying number of parallel processing units.

The results obtained are the output of simulations of the sequential version of VQ using OpenMP. Implementation of the same using a multi-core system would provide accurate results regarding aspects like 'memory used' and 'core utilization' which were difficult to obtain in the present scenario. In a typical low-bandwidth network, consider a server which provides videos on demand. If the number of requests to the server is very high then the server sends a compressed version of the original high quality video. In this kind of scenario, the algorithm proposed here would greatly enhance the server response time by decreasing the time taken for video compression techniques which make use of LBG algorithm.

4 REFERENCES

[1] Y. Linde, A. Buzo, and R. M. Gray: *An algorithm for vector quantizer design*, IEEE Trans. Commun., vol. COM-28, pp. 84-95, Jan. 1980

[2] Toshiyuki Nozawa, Makoto Imai, Masanori Fujibayashi,and Tadahiro Ohmi: *A Parallel Vector Quantization Processor Featuring an Efficient Search Algorithm for Real-time Motion Picture Compression* ASP-DAC 2001: 25-26

[3] Akiyoshi Wakatani: *A VQ compression algorithm for a multiprocessor system with a global sort collective function*, data compression Conference .DCC 2006 proceedings.

[4] Troy Maurice Thomas: *Vector Quantization of Color Images Using Distributed Multiprocessors*, T.R. #USUEE-88-15, Utah State University, Logan, Ut., 1988.

[5] *Parallel codebook design for vector quantization on a message passing MIMD architecture,* Hazem M. Abbas, Mohamed M. Bayoumi 2002.

[6] Lee, H.J. Liu, J.C. Chan, A.K. Chui, C.K: *A parallel vector quantization algorithm for SIMD multi-processor systems,* Data Compression Conference, 1995. DCC '95 Proceedings

[7] Vandana S.Rungta: *parallel vector quantization codebook generation,* Utah State University, Logan, Utah 1991.

[8] Edward A. Fox: *DVI Parallel Image Compression*, Communications of the ACM, Vol 32, Number 7, July 1989, pp 844-851.

[9] C. Huang: *Large vector quantization codebook generation analysis and design*, Ph.D. dissertation, Utah State Univ., 1990

[10] W. H. Equitz: *A vector quantization clustering algorithm*, IEEE. Trans. ASSP. vol. 37, no. 10, pp. 1568-1575, Oct. 1989.