FPGA-based System for Real-time Video Texture Analysis

Dimitris Maroulis, Dimitris K. Iakovidis, Dimitris Bariamis

Real Time Systems and Image Analysis Laboratory, Department of Informatics and Telecommunications, University of Athens, Greece

{d.maroulis, d.iakovidis, d.bariamis}@di.uoa.gr

Abstract. This paper describes a novel system for real-time video texture analysis. The system utilizes hardware to extract 2nd-order statistical features from video frames. These features are based on the Gray Level Co-occurrence Matrix (GLCM) and describe the textural content of the video frames. They can be used in a variety of video analysis and pattern recognition applications, such as remote sensing, industrial and medical. The hardware is implemented on a Virtex-XCV2000E-6 FPGA programmed in VHDL. It is based on an architecture that exploits the symmetry and the sparseness of the GLCM and calculates the features using integer and fixed point arithmetic. Moreover, it integrates an efficient algorithm for fast and accurate logarithm approximation, required in feature calculations. The software handles the video frame transfers from/to the hardware and executes only complementary floating point operations. The performance of the proposed system was experimentally evaluated using standard test video clips. The system was implemented and tested and its performance reached 133 fps and 532 fps for the analysis of CIF and QCIF video frames respectively. Compared to the state of the art GLCM feature extraction systems, the proposed system provides more efficient use of the memory bandwidth and the FPGA resources, in addition to higher processing throughput, that results in real time operation. Furthermore, its fundamental units can be used in any hardware application that requires sparse matrix representation or accurate and efficient logarithm estimation.

Keywords: Field Programmable Gate Arrays, Parallel Architectures, Pattern Recognition, Video Signal Processing, Real-Time System.

1. Introduction

Texture is an innate property of the natural objects, and it is widely used for video content description. The utility of texture feature extraction from video extends to a wide range of advanced modern applications, including segmentation of objects in image sequences [1,2], object recognition [3], tracking of moving objects [4,5], video transcoding for video content adaptation [6] and adaptive intra refreshment schemes for improved error resilience in object-based video coding [7].

The Gray Level Cooccurrence Matrix (GLCM) features [8] describe the textural image content by encoding the second order statistical properties of texture. These properties are mostly related to the human perception and discrimination of textures [9]. The GLCM features have been successfully utilized in a number of applications including medical [10], remote sensing [11] and industrial visual inspection applications [12,13].

A major drawback of the GLCM feature extraction method is its high computational complexity, which is prohibiting for real-time video texture analysis in software. A system capable of performing video texture analysis in real-time would be useful for a variety of applications including temporal analysis of video frame sequences [14], and analysis of medical video streams, such as endoscopic [10] or ultrasound screening [15]. In such cases, the software implementations are usually incapable of achieving real-time performance when full resolution video streams are used rather than downscaled videos. To overcome the limitations imposed by the software, we considered the utilization of dedicated hardware based on Field Programmable Gate Arrays (FPGAs). FPGAs are low cost and high density gate arrays capable of performing many complex computations in parallel while hosted by conventional computer hardware. They have been the choice for the implementation of computationally intense feature extraction tasks, including fingerprint feature extraction [16], facial feature extraction [17], the computation of Zernike moments [18], etc.

An FPGA-based system for the computation of two GLCM features has been proposed by Heikkinen and Vuorimaa [19]. This system approximates only two simple

features, namely mean and contrast, without actually computing the GLCMs. Tahir et al. [20] has presented another architecture that calculates the GLCM of multispectral images. The calculation of the GLCM is performed by one FPGA core whereas the computation of the GLCM features is performed by a second core that is subsequently programmed onto the FPGA. However, the use of this second core results in a time overhead for reprogramming the FPGA, affecting the overall feature extraction performance.

A similar system was proposed by our research group [21]. This system calculates both the GLCMs and features in hardware, but relies on software for a significant part of the computations. It performs well when the input image is divided into overlapping blocks, but in the case of non-overlapping blocks, it is inefficient as several of the units are unused for extended periods of time. Another FPGA-based system for GLCM calculation has been proposed by our research group [22], which provides more efficient calculation of GLCMs, however it does not calculate any GLCM features in hardware. Furthermore, the transfer of GLCMs over the PCI bus incurs a significant performance overhead, which can be prohibiting for real-time video texture analysis. The system presented in [23] was capable of GLCM features calculation in hardware, but employed data redundancy in order to achieve high processing throughput. However, the redundancy led to high memory capacity requirements and redundant transfers of data over the PCI bus.

In this paper we propose a novel FPGA-based system for real-time extraction of GLCM texture features from video frames. The motivation for the development of this system was to cover the need for real time extraction of texture features from uncompressed video streams, such as the input of the Colorectal Lesion Detection (CoLD) software [24]. The proposed system is capable of calculating a total of 64 GLCM features in parallel, namely angular second moment, correlation, inverse difference moment and entropy, at four different directions in a video frame, for four video frame blocks. It is implemented on a single FPGA core that performs the calculation of both the GLCMs and the features, exploiting the symmetry and sparseness of the GLCM and using integer and fixed point arithmetic. The proposed system also incorporates an algorithm for efficient approximation of the logarithm

in the entropy feature, and an effective buffering scheme, which only occupies a small fraction on the FPGA area and reduces the external memory requirements, while retaining a high processing throughput.

The rest of this paper is organized in five sections. The methodology used for the extraction of the GLCM texture features is described in Section 2. The architecture of the proposed system presented in Section 3 is followed by a complexity analysis in Section 4. The results obtained from the experimental evaluation on standard video clips are apposed in Section 5. Finally, Section 6 summarizes the conclusions derived from this study.

2. Texture Features Extraction

GLCMs encode the gray level spatial dependence based on the estimation of the 2nd order joint-conditional probability density function, which is computed by counting all pairs of pixels of a video frame block at distance d having gray levels i and j at a given direction θ. The cooccurrence matrix can be regarded symmetric if the distribution between opposite directions is ignored, so the angular displacement is usually included in the range of the values {0°, 45°, 90°, 135°} [25]. Among the 14 statistical GLCM features, originally proposed by Haralick et al [12], we consider four (Eqs. 1 to 4), namely, angular second moment (f_1), correlation (f_2), inverse difference moment (f_3) and entropy (f_4). These four selected features are widely used in the literature because they provide high discrimination accuracy, which can be only marginally increased by adding more features in the feature vector [12,26].

$$f_1 = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p_{ij}^2 \tag{1}$$

$$f_2 = \frac{\sum_{i=1}^{N_g} \sum_{j=1}^{N_g} i \cdot j \cdot p_{ij} - \mu_x \cdot \mu_y}{\sigma_x \cdot \sigma_y} \tag{2}$$

$$f_3 = \sum_{i=1}^{N_g}\sum_{j=1}^{N_g} \frac{1}{1+(i-j)^2} p_{ij} \tag{3}$$

$$f_4 = -\sum_{i=1}^{N_g}\sum_{j=1}^{N_g} p_{ij} \cdot \log_2 p_{ij} \tag{4}$$

where p_{ij} is the ijth entry of the normalized cooccurrence matrix, N_g is the number of gray-levels of the video frame, μ_x, μ_y, σ_x, and σ_y are the means and standard deviations of the marginal probabilities $P_x(i)$ and $P_y(j)$ obtained by summing up the rows or the columns of matrix p_{ij} respectively.

The calculation of the Eqs. 1-4 requires floating point operations that would result in high FPGA area utilization and low operating frequencies. To implement the calculation of the features efficiently in hardware, we have reformulated the equations by extracting five expressions V_1 to V_5 as follows:

$$f_1 = \frac{V_1}{r^2} \tag{5}$$

$$f_2 = \frac{\left(N_g - 1\right)\cdot\left(r \cdot N_g^2 \cdot V_2 - r^2\right)}{N_g^2 \cdot V_5 - N_g r^2} \tag{6}$$

$$f_3 = \frac{1}{r} \cdot V_3 \tag{7}$$

$$f_4 = r \cdot \log_2 r - \frac{1}{r} \cdot V_4 \tag{8}$$

where

$$V_1 = \sum_{i=1}^{N_g}\sum_{j=1}^{N_g} c_{ij}^2 \tag{9}$$

$$V_2 = \sum_{i=1}^{N_g}\sum_{j=1}^{N_g} i \cdot j \cdot c_{ij} \tag{10}$$

$$V_3 = \sum_{i=1}^{N_g}\sum_{j=1}^{N_g} c_{ij} \cdot IDMLUT[|i-j|] \tag{11}$$

$$V_4 = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} c_{ij} \cdot \log_2 c_{ij} \tag{12}$$

$$V_5 = \sum_{i=1}^{N_g} C_x^2(i) \tag{13}$$

$$c_{ij} = r \cdot p_{ij} \tag{14}$$

$$C_x(i) = r \cdot P_x(i) \tag{15}$$

$$IDMLUT[|i-j|] = \frac{1}{1+(i-j)^2},\ 0 \le |i-j| < N_g \tag{16}$$

In the above equations, the operations needed to calculate V_1 to V_5 (Eqs. 9-13) are performed using integer or fixed point arithmetic. These values are computed in hardware and subsequently used for the calculation of the four features f_1 to f_4 in software, requiring only 16 floating-point operations that incur a negligible time overhead. In contrast to this approach, the implementation of a floating-point unit capable of performing the division operation on FPGA would significantly diminish the performance of the system, by reducing the achieved frequency and throughput of the hardware architecture.

The parameter r (Eqs. 14 and 15) represents the number of pixel pairs of a block, for a specific direction and distance. $IDMLUT[|i\text{-}j|]$ is an N_g×32-bit lookup table, which contains a 32-bit fixed point representation of the function $1/(1+(i\text{-}j)^2)$, $0 \le |i\text{-}j| < N_g$. The calculation of the logarithm $\log_2 c_{ij}$ is implemented using a fast approximation method in hardware. The implementation returns a 32-bit fixed point value $\log_2 c_{ij}$ for each integer c_{ij}.

3. System Description

The proposed system is based on a Xilinx XCV2000E-6 FPGA, programmed in VHDL [27]. The FPGA features 19,200 slices, includes 160 256×16-bit Block RAMs and can support up to 600kbit of distributed RAM. It is packaged in a 560-pin ball grid array (BGA560) that provides 404 user I/O pins. It is hosted by the Celoxica RC-1000 board that includes four 2MB static RAM banks [28]. These RAM banks can be accessed by the FPGA

or the host computer independently, whereas simultaneous access is prohibited. This is ensured by the board's arbitration circuit which assigns the ownership of each bank to either the host or the FPGA using isolator circuits. The FPGA and the host processor can communicate through the board in two ways: by bulk PCI transfers from/to the memory banks, and by control and status byte transfers. Bulk transfers are commonly used for large data transfers, but can only be initiated by the host. Control and status byte transfers are mostly used for synchronization and can be initiated by either the FPGA or the host.

The architecture of the implemented hardware is illustrated in Fig. 1. The software iteratively feeds the FPGA board with four video frame blocks per iteration. Each pixel is represented by 6 bits (N_g=64) and the pixels of the four blocks are interleaved, allowing the utilization of the memory bank width and the retrieval of one pixel from each block in one clock cycle, for a total of four pixels. The FPGA reads each block's pixels, calculates the GLCM of each block and their respective feature vectors for the θ=0°, 45°, 90° and 135° directions and d=1 distance, and stores them into memory bank 1 and 2.

The FPGA architecture consists of:

- A control unit
- Three memory controllers (for memory banks 0, 1 and 2)
- A circular buffers unit
- Sixteen GLCM calculation units (GCUs)
- Four vector calculation units (VCUs)