Detection of objects in moving images and implementation of the purification algorithm on Analog CNN and DSP processors

Emel Arslan[1], Zeynep Orman[2], Sabri Arik[2]
[1] Research and Application Center for Computer Sciences,
[2] Department of Computer Engineering,
Istanbul University
Istanbul, TURKEY
{earslan, ormanz, ariks}@istanbul.edu.tr

Abstract: - CNN Universal Machines that contain two different processors working interactively with each other, have an important impact for image processing applications with their advanced computing features. These processors are named as ACE16k processor which is the hardware implementation of cellular neural networks and Digital Signal Processor (DSP). Bi-i Cellular Vision System is a CNN Universal Machine that also has these characteristics. In this study, certain objects in moving images are detected and their features are extracted. By using these features, a purification algorithm is implemented on the Bi-i Cellular Vision System. This algorithm is implemented with two different applications. In the first application, the algorithm is implemented only on a Digital Signal Processor. In the second one, it is run in coordination with both an ACE16k processor and a Digital Signal Processor. These applications are evaluated by comparing the obtained results in terms of run-time.

Key-Words: - Image processing, Cellular Neural Networks, Digital Signal Processor, ACE16k, Bi-i Cellular Vision System, CNN Universal Machine

1 Introduction

Image processing is one of the most important research topics in recent years. It is widely used in areas such as military, security, health, biology, astronomy, archeology and industry [1, 2]. For an image to be processed, it should be presented in a format that a computer can understand, this means it should be converted into its related digital form. In the digital form, each of its pixel is expressed by means of the corresponding element of a matrix.

Algorithms that are developed for digital image processing require fast systems due to their processing load. Although conventional computer systems are in an increasing trend in terms of their speed, they are insufficient when image dimensions are getting larger, because they can process transactions in a serial manner. These computers cannot satisfy the need for speed, when we especially consider the implementation of real-time moving image processing algorithms that require at least 15-25 frames to be processed in seconds.

Cellular Neural Network (CNN) theory that was proposed by Chua and Yang in 1988, is an analog, nonlinear and real-time processing neural network model [1]. CNNs also have advanced features for image processing applications. In 1993, Roska and Chua have presented the CNN Universal Machine [2]-[3]. This analogical array computer has cellular processors (ACE4k, ACE16k, etc.) which are the hardware implementation of CNNs and it is very suitable for image processing applications with its advanced computing capabilities. Bi-i Cellular Vision System is a CNN Universal Machine that can process high-speed and real time transactions and can be defined as a compact, independent and intelligent camera. This system has high-resolution sensors and two different processors named as CNN (ACE16k) and Digital Signal Processor (DSP) that can communicate with each other [4].

In this study, we will first discuss how to detect certain objects in a colored image and a purification algorithm that is used to purify the remains of each object. These remains are actually the parts of other objects that stay in the frame. Then, we will present an implementation of this algorithm on the Bi-i Cellular Vision System and evaluate the results that are obtained.

The remainder of this letter is organized as follows. Section II introduces fundamental concepts about CNN architecture, CNN Universal Machine, ACE16k processor, Bi-i Vision System and Bi-i

programming, respectively. In Section III, an algorithm that detects certain objects in moving images and purifies each object from its remains is proposed. In Section IV, a real implementation of the purification algorithm is provided to compare the obtained results and finally, Section V concludes the paper.

2 CNN Architecture and Bi-i Cellular Vision System

This section provides fundamental concepts about CNN architecture and Bi-i Cellular Vision System.

2.1 Architecture of the Cellular Neural Networks

Each cell of a 4x4 CNN is represented by a square and shown in Figure 1.

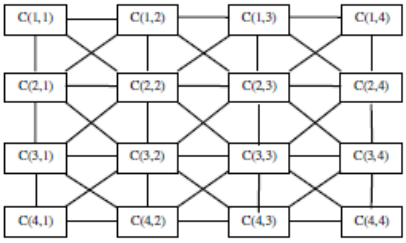

Fig. 1. A 4x4 cell two-dimensional CNN.

In this CNN architecture, each cell is linked only to its neighbors.

Let us assume a CNN with MxN cells arranged in M rows and N columns, the cell in row i column j is shown as C(i,j)[1]. r-neighborhood of a C(i,j)cell is defined as follows provided that r is a positive value:

$$N_r(i,j)=\left\{C(k,l)\left|\max_{1\le k\le M,1\le l\le N}\{|k-i|,|l-j|\}\le r\right.\right\} \quad (1)$$

2.2 CNN Universal Machine

The hardware implementation of CNN is easier compared to the Artifical Neural Networks as there is only connection between the neighbor cells and the cell structure. Analogical Cellular Engines (ACE4k, ACE16k etc. [5]) are based on CNN Universal Machine architecture. CNN Universal Machine architecture has been called by Roska and Chua as analogical computation since it can perform analog array operations and logical operations together [2].

2.3 ACE16k Processor

ACE16k, is a CNN based processor of CNN Universal Machine which can perform analog operations. ACE16k which is used to perform various image processing operations contain low resolution (128 x 128) CMOS gray level image sensor and analog processor arrays. This processor array is much faster (30000 frames per second) than the conventional processors in image processing applications since it can processes the whole image in parallel.

2.4 Bi-i Cellular Vision System

The Bi-i Cellular Vision System which contains two different processors, a CNN based ACE16k and a DSP that can be defined as a compact, standalone and intelligent camera capable of real time operations at very high speed [4]. The images are stored in local memories with the help of two different sensors as a (1280x1024) color CMOS sensor, and a (128x128) ACE16K sensor.

2.5 Bi-i Programming

CNN Universal Machine has two different programming methods. One of them is AMC (Analogical Macro Code) language which is a conventional Bi-i programming method. The codes written in AMC language are converted to binary basis and run on Bi-i. Another method is the Bi-i (Software Development Kit - SDK) which is used to develop more complex applications. Bi-i SDK, consists of the C++ programming library which is a group used to develop applications. These libraries can also used for the Digital Signal Processor (DSP) with the development unit Code Composer Studio and they contain many functions to control the whole ACE16k circuit [6].

3 An Algorithm That Detects Objects in Moving Images and Purifies the Remains of Each Object

An algorithm that detects objects in moving images and extracts their features is developed by using Bi-i Cellular Vision System. We named this algorithm as the purification algorithm because it is also used for purifying the remains of certain objects that are detected in moving images and saving each purified

object as a separate image file. These remains are actually the parts of other objects that stay within the frame of a certain object. This kind of remains problem is inevitable in image processing applications because each pixel of an object is expressed as an element of a matrix. The algorithm that we developed provides a solution to this problem for colored and moving images. A block diagram of this algorithm is shown in Figure 4.

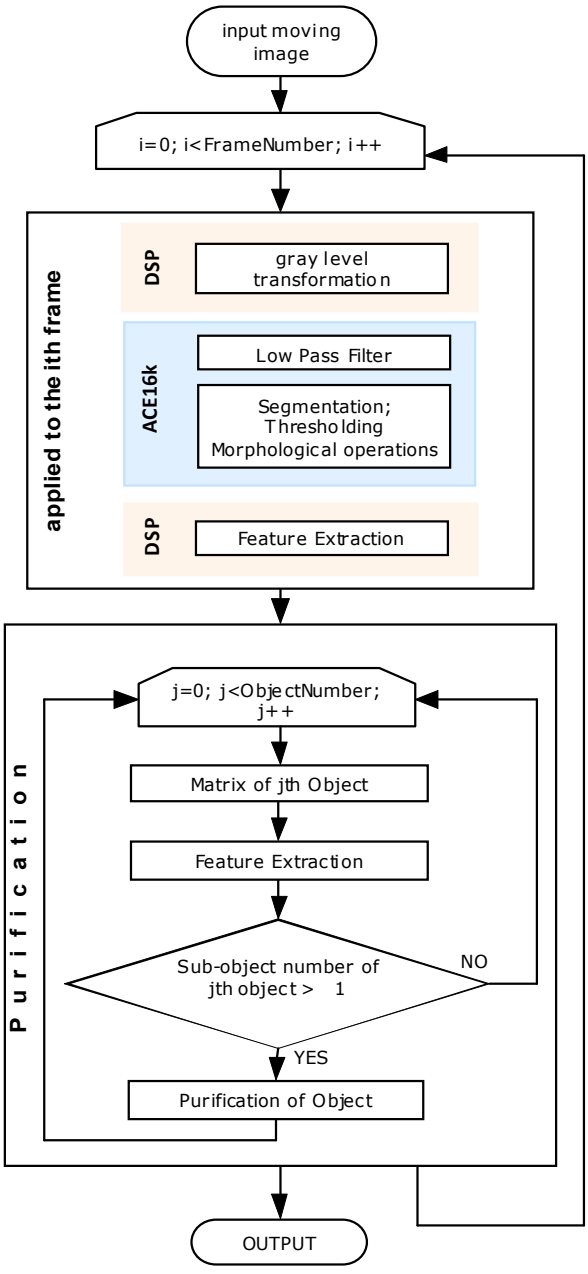

Fig.2. A block diagram of the purification algorithm

3.1 Preprocessing

Each frame of a moving object which is an input to the algorithm, is processed as a separate image. The first operation is to transform the frame to be processed into its relevant gray level image that runs on a DSP processor by using the RGB2ByteMatrix function. This function is located in Utils.h library under Instant Vision BaseData. The command line that is used for this transformation is given below:

```
RGB2ByteMatrix(sourceGRAY,sourceRGB,CH_RGB);
```

The moving image matrix that is transformed to a gray level image is then transferred to ACE16k processor for morphological operations. Here, the aim is to detect objects in a moving image as close as possible. The morphological operations are Low Pass Filtering [7], Thresholding, Negation, Point Removing, Hole Filling and Opening, respectively. For thresholding, we use the ConvLAMtoLLM function that is located in TACE.h library. For other operations we use the functions in TACE_IPL.h library. These are all ACE16k libraries that are under Instant Vision BaseData.

For morphological operations, the first step is to pass the moving image through a linear Low Pass Filter to remove the noise. This filter is used to get rid of some little details and fill the holes in lines before detecting the objects.

For filtering a moving image, we use the following command lines in our implementation:

```
ace << C_LAM1 << sourceGRAY;
ace.LowPassFilter (C_LAM2, C_LAM1,1, 0.05);
ace.ReadLAM(sourceGRAYFiltered, C_LAM2);
```

A gray level image that is sourceGray, is loaded to a local analogical memory (C_LAMI) that is on the ACE16k processor to be processed by using the first command line. The second command line applies Low Pass Filtering to an image matrix that is on C_LAM1 by using the LowPassFilter() function and passes the resulting image to another local analogical memory (C_LAM2). Finally, the last line assigns the resulting image to a variable named sourceGRAYFiltered.

By applying low pass filtering, we now have a smooth image. Afterwards, this image is converted to a binary image by thresholding. The command lines that are used for this conversion is given below:

```
ace.ConvLAMtoLLM(C_LLM2,C_LAM2,50);
ace.ReadLLM(sourceBit, C_LLM2);
```

In the first line, the image matrix that is on C_LAM2, is converted into a binary image and transferred to another local analogical memory

(C_LLM2) by using ConvLAMtoLLM() function. The second line assigns the resulting matrix to a variable named sourceBit.

To perform thresholding by using ByteMatrix2BitMatrix() function that runs on a DSP, we use the following command line:

```
ByteMatrix2BitMatrix(sourceBit, sourceGRAY,50);
```

After thresholding, the objects are represented as black and other parts are represented as white in the resulting image matrix. However, our algorithm detects groups of white pixels as objects and carries out its operations in this manner. To solve this problem, we need to negate the resulting image. This means, we should replace 0s with 1s and 1s with 0s in the image matrix by applying a negation process. This binary negation process is performed by the following command line:

```
ace.Not(sourceBitNegation,C_LAM2);
```

The same process can be run on a DSP processor with a Negation() function that is written in C++ programming language. This function is used with the following command line:

```
Negation(sourceBitOut, sourceBit);
```

After the negation process, the morphological operations that are given with the following command lines, are applied to the resulting image, respectively.

```
ace.SetIPLMode(IPL_MORPH);
ace.Calibrate();
ace.PointRemove();
ace.Calibrate();
ace.HoleFiller(1);
ace.Calibrate();
ace.Opening8(1);
ace.Calibrate();
ace.Dilate8(1);
```

Before going on with other morphological operations, the SetIPLMode() function should be used to set up ACE16k processor to carry out these procedures. Then, before each morphological operation, the Calibrate() function is used to calibrate ACE16k processor for morphological operations. It is very important to use this function and repeat it in every 10-20 milliseconds if the processor works in an IPL mode constantly because the ACE16k processor can forget this calibration process. This process is not important for gray level images.

After the calibration process, the PointRemove() function that cleans the minor pixels from the image, the HoleFiller() function that fills the holes of the image, the Opening8() function that performs the opening process and finally the Dilate8() function is applied to the image, respectively [8].

The command lines that perform these morphological operations on a DSP processor are as follows:

```
PointRemove(sourceBitOut,sourceBitOut);
HoleFiller(sourceBitOut, sourceBitOut, 1);
Opening8(sourceBitOut, sourceBitOut, 1);
Dilate8(sourceBitOut, sourceBitOut, 1);
```

After all these morphological operations, sample output images are shown in Figure 3 for different frames that belong to a specific moving image.

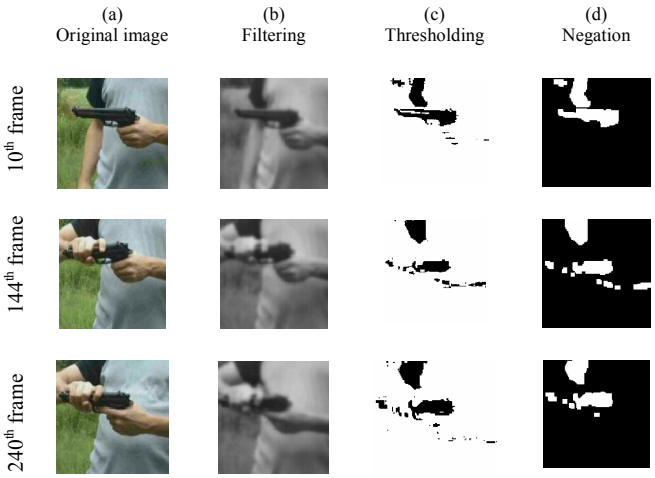

Fig.3. Sample output images after preprocessing

3.2 Extracting the Objects and their Features

The binary image that is preprocessed depending on the input moving image is now available for object detection and is an input parameter to the CalcFeatures() function. This function is used as follows:

```
CalcFeatures(ObjNum, Features, sourceBitOut, FEAT_ALL);
```

After this process, the objects in a moving image are detected and their features are extracted. These features are given as follows:

- Bounding Box: It draws a minimal rectangle around the object by using its upper-left and lower-right coordinates.
- Extremes: These are the extreme points on the bounding box of the object. These points are: Upper-Left, Upper-Right, Right-Upper, Right-Lower, Lower-Right, Lower-Left, Left-Lower and Left-Upper.
- Eccentricity: It is the proportion between the focus distance and the length of the longest axis of the current ellipse of the object. The eccentricity of a circle is 0 whereas the eccentricity of a line is 1.
- Diameter: It is the diameter of the circle which has the same area with the object.
- Orientation: It is the angle of the largest axis value of the current ellipse as the object to the positive direction of the horizontal axis.
- Extent: It is the proportion between the area of the object and the area of the bounding box.
- Center: It denotes the coordinates of the geometric coordinates of the object according to the upper left corner of the image.

After the objects and their features are extracted, the resulting image is saved to be purified from its remains.

3.3 Purifying Certain Objects from its remains

Each pixel of an image is expressed by means of the corresponding element of a matrix, so even no remains of other objects are detected, a part of the background will be still within the boundaries of the certain object that is detected in the image. That is why we should purify the detected objects from its remains. The input moving images that we are working on are colored and this makes the purification process easier. In our implementation, we will consider the moving image given in Figure 3 and try to purify the most distinct object in that certain image - which is the weapon.

For this implementation, we first obtain a matrix that represents the object by using the extreme points. This object is then saved as a separate image file with the help of the related matrix and purified from its remains that belong to other objects and from the background image by using the color codes. This function that performs this process is coded in C++ programming language.

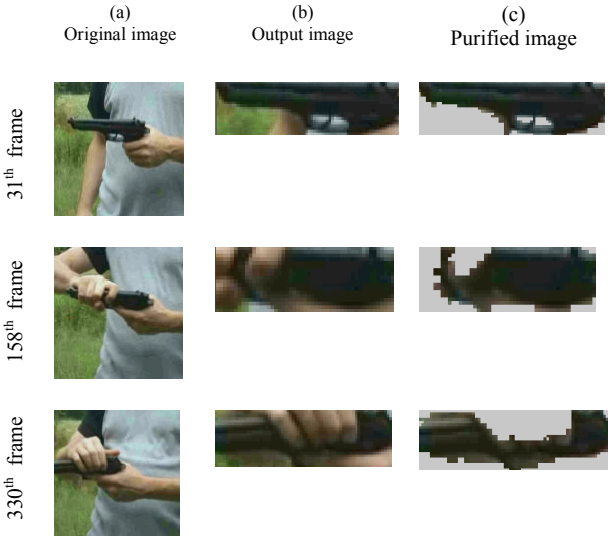

Fig. 4. Output images after the purification process

Because the image to be processed is colored, we can get the Red-Green-Blue values of each pixel of the weapon object and with these values, we call the ForMask() function. This function gets three input parameters. These are the matrix that represents the colored image, the [x,y] coordinate values of the pixel that is to be processed and the upper and lower values of the image that is to be purified. ForMask() function examines each element of the input matrix in accordance with the constraint generated by the color values. After the completion of this process, the extracted object is free from all colors that do not belong to it. Finally, the last step of this purification process is to resave the detected object with a pre-determined plain color background. Some example results for different frames from our implementation are shown in Figure 4.

4 Experimental Results

In this algorithm, the filtering and the segmentation processes that are applied to the image in the preprocessing phase, can be run on a ACE16k processor. For this implementation, we write two different program codes that one of them is just run on a DSP and the other one is run both on an ACE16k and a DSP interactively. The results obtained from the implementation are evaluated as run-time of the purification algorithm and shown in Table1.

Process	DSP	DSP+ACE16k
Filtering and Segmentation	65312 μs	28190 μs (ACE16k)
Feature Extraction	327703 μs	327703 μs (DSP)
Purification of object	42654 μs	42654 μs (DSP)
TOTAL	435669 μs	398547 μs

Table 1. Serial Run-Time Of Purification Algorithm

When we compare these results, one can easily notice that processing the algorithm on both processors is 37122 μs faster than processing it only on a DSP.

5 Conclusion

In this paper, we have studied on the detection of certain objects in moving images and the extraction of their features by using the Bi-i Cellular Vision System. We have also implemented an algorithm that purifies the remains of each object that we have detected by using the extracted features. The filtering and the segmentation processes of the algorithm are implemented on an ACE16k processor. The other processes like gray level transformation, feature extraction and purification of objects are implemented on a Digital Signal Processor (DSP). After the implementation of this algorithm, each purified object that is detected in a moving image, is saved as a separate image file.

The results given in Table 1 have shown that when the algorithm runs on both DSP and ACE16k processors, the run-time is faster than when it just runs on a DSP processor.

References:

[1] L. O. Chua ve L. Yang, " Cellular neural networks: Theory and applications", IEEE Trans. on CAS, cilt 35 no. 10, 1988, s.1257–1290

[2] T. Roska ve L. O. Chua, "The CNN universal machine: an analogic array computer", IEEE Trans. on CAS-I, cilt. 40 no.3, 1993, s. 163–173

[3] T. Roska ve A. Rodriguez-Vazquez, "Towards visual microprocessors", Proceedings of the IEEE, cilt 90 no.7, 2002, s. 1244–1257

[4] A. Zarandy ve C. Rekeczky, "Bi-i: a standalone ultra high speed cellular vision system.", IEEE Circuit and Systems Magazine, cilt 5, no.2, 2005, s. 36–45

[5] C. Gonzales ve R.E. Woods, "Digital Image Processing", Prentice Hall, New Jersey, 2002

[6] http://www.analogic-computers.com /Support

[7] T. Acharya ve A.K. Ray, "Image Processing: Principles and Applications", Wiley and Sons, 2005

[8] A.R. Vazquez, G. L. Cembrano, L. Carranza, E.R.Moreno, R.C. Galan, F.J. Garrido , R.D. Castro ve S. E. Meana, "ACE16k: the third generation of mixed-signal SIMDCNN ACE chips toward VSoCs", IEEE Trans. CAS-I, cilt 51,no.5, 2004, s. 851– 863