A Real-time Image Feature Extraction Using Pulse-Coupled Neural Network

Trong-Thuc Hoang¹, Ngoc-Hung Nguyen², Xuan-Thuan Nguyen³ and Trong-Tu Bui⁴

¹²³⁴ The University of Science, VNU-HCM, Faculty of Electronics and Telecommunications,
227 Nguyen Van Cu St., Dist. 5, Ho Chi Minh City, Vietnam

Abstract: In this paper, we present two hardware architectures for an image feature vector extraction based on the Pulse-Coupled Neural Network (PCNN) algorithm. They are RAM-based model and pipelined model. Both models can generate a feature vector at the speed of more than 2000 vectors per second when using a clock frequency of 50 MHz and input image size of 128x128 pixels. Based on these architectures, a demonstration recognition system including a camera, a feature vector generator, a search engine and a DVI controller has been built and tested successfully on FPGA chips in order to verify the operation of the algorithm and the architectures.

Keywords: feature vector, hardware implementation, object recognition, PCNN, Pulse-Coupled Neural Network, real-time.

1. INTRODUCTION

Pulse-Coupled Neural Network (PCNN) is a biologically inspired neural network based on cat’s visual cortical neurons. The significant advantage of the PCNN model is that it can operate without any training needed. Since introduced by Eckhorn in 1990 [1], the model has proven its vital role in digital image processing, such as image segmentation [2], image thinning [3], motion detection [4], pattern recognition [5], face detection [6], etc. In the comparison with other image processing algorithms, the PCNN algorithm is anti-noise and robust against the translation, scale, and rotation of the input patterns [7]. The pattern-matching function plays an important role in various information processing systems. In order to implement such functions effectively, both proper data representation algorithms and powerful search engines are essential. Concerning the former, in this paper we present a design of a feature vector generator employing the PCNN algorithm. Many techniques and algorithms have been developed for image feature extraction using ICA [8], PCA [9], Haar transformation [10] etc. Those feature extraction methods are effective to extract features of simple images, such as simple shapes and letters. However, they are sensitive to noise and not invariant against the geometric changes. Feature extraction process is time consuming, thus software implementation is not suitable for real-time applications. Hardware implementation is a solution to overcome such an issue. In the literature, several analog VLSI circuits that implement the PCNN have been presented [11]. Although compact they suffer from accuracy as well as process variations and device mismatches which are disadvantages of analog circuits. Digital implementations have also proposed, Javier et al [12] presented an FPGA system that can operate at high speed, but it is only suitable for those applications that do not require the iterative operation of the PCNN. In an image feature extraction system, it requires the iterative processing of the PCNN model in order to generate a feature vector. Ranganath et al [13] proposed such a hardware design, but it needs multiple PCNN layers and dynamic parameters.

In this paper, we present two digital designs based on the PCNN algorithm which can be applied into a real-time object-recognition system. They are RAM-based model and pipelined model. The RAM-based model utilizes RAMs to store the network's information in order to reduce the LUTs and registers costs. In the other hand, the pipelined model uses only the registers, thereby giving rise to a convenient implement in ASIC. The pipelined architecture divides the operation of the PCNN algorithm into stages. Each stage generates one element of a feature vector. The resource costs of RAM-based model depend on the input image size and that of the pipelined model mainly depend on the wanted element number of an output feature vector. Both models can generate a feature vector at the speed of more than 2000 vectors per second when using a clock frequency of 50 MHz and image size of 128x128 pixels. Based on these architectures, a demonstration of object-recognition system has been built, including a camera, a feature vector generator, a search engine, and a DVI controller. The system can recognize an object in the input image based on the feature vector generated by the feature vector generator employing the PCNN algorithm.

The remainder of this paper is organized as follows. Section 2 briefly reviews the PCNN model and gives an implementation of a neuron. Section 3 describes a feature vector generation and its application in an object recognition system. Section 4 proposes two PCNN hardware architectures with their experimental results. Section 5 describes a completed object-recognition system and its performance. Section 6 gives the conclusion.
2. THE PCNN MODEL

2.1 Description of the Model

The PCNN is a two-dimensional neural network with a single layer. Each network neuron corresponds to an input image pixel. Because of this, the structure of the PCNN comes out from the structure of input image.

The PCNN's neuron structure is shown in Fig. 1 [14]. There are three parts that form a neuron: a feeding field (receptive field or dendritic tree in some references), a linking modulation, and a pulse generator. The neuron's operation is described as iteration by the following equations.

\[
F_y[n] = S_y + F_y[n-1]e^{-\alpha_T} + V_F \sum_{kl} W_{kl} Y_z[n-1] \tag{1}
\]

\[
L_y[n] = L_y[n-1]e^{-\alpha_L} - V_L \sum_{kl} W_{kl} Y_z[n-1] \tag{2}
\]

\[
U_y[n] = F_y[n] + \beta L_y[n] \tag{3}
\]

\[
T_y[n] = T_y[n-1]e^{-\alpha_T} - V_T Y_z[n-1] \tag{4}
\]

\[
Y_y[n] = \begin{cases} 
  1 & U_y[n] > T_y[n] \\
  0 & \text{otherwise} 
\end{cases} \tag{5}
\]

In these equations, \((i, j)\) is the position of a neuron in the network and a pixel in the input image. For example, if the input image size is 128x128 pixels, the range of indexes \((i, j)\) will be \(1, 1\) and \(128, 128\). \((k, l)\) represent the positions of the surrounding neurons; \(n\) is the iterative step number; and \(S_y\) is the gray level of the input image pixel. \(F_y[n], L_y[n], U_y[n],\) and \(T_y[n]\) is the feeding input, linking input, internal activities, and dynamic threshold, respectively. The pulse output of a neuron, \(Y_y[n]\), is the binary value which indicates the status of the neuron. \(Y_y[n]\) equals to 1 means the neuron is activated.

\(M\) and \(W\) are the constant synaptic weight matrices and depended on the field of the surrounding neurons, i.e. the linking field. For example, if each neuron is only connected by eight neurons surrounding it, the linking field is a 3x3 matrix. As a result, \(M\) and \(W\) are 3x3 matrices, too. \(M\) and \(W\) refer to the Gaussian weight functions with distance and they are usually chosen as the same. \(\beta\) is the linking coefficient constant; \(\alpha_T, \alpha_L,\) and \(\alpha_T\) are the attenuation time constants; and \(V_F, V_L,\) and \(V_T\) are the inherent voltage potential of the feeding signal, linking signal, and dynamic threshold, respectively.

In conclusion, the operation of a neuron at step \(n\) depended on its corresponding pixel value (i.e. \(S_y\)), the pulses output from the surrounding neurons at the previous step (i.e. \(Y_z[n-1]\)), and the internal signals of the neuron itself (i.e. \(F_y[n-1], L_y[n-1], U_y[n-1],\) and \(T_y[n-1]\)). All of the network pulses \(Y_y[n]\) form a binary image which contains important information such as regional information, edge information, and features of an object in the input image.

2.2A Neuron Implementation

Based on many references practical experiments, Tab. 1 shows the PCNN parameters which were selected for the feature extraction purpose. The top-view of hardware implementation for a neuron is shown in Fig. 2. \(iS\_data, iY\_data, iF\_data, iL\_data,\) and \(iT\_data\) in the figure are represent for \(S_y, Y_z[n-1], F_y[n-1], L_y[n-1],\) and \(T_y[n-1]\) in the theory, respectively. They are the required input for the neuron operation. \(oY\_data, oF\_data, oL\_data,\) and \(oT\_data\) are \(Y_y[n], F_y[n], L_y[n],\) and \(T_y[n]\) in the model, respectively. The input pixel value is 8-bit gray-scale and carried by the \(iS\_data\) signal. \(iY\_data\) signal has the width of nine because the linking field is chosen as a 3x3 matrix, \(oY\_data\) signal is the pulse output with binary value, thus it is a single net. With the PCNN parameters were fixed, the widths of the other signals are calculated logically.

![Figure 1: A PCNN neuron structure](image1)

![Figure 2: Top-view of a neuron implementation](image2)

![Figure 3: A PCNN neuron implementation](image3)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>(\alpha_T)</th>
<th>(\alpha_L)</th>
<th>(\alpha_T)</th>
<th>(V_F)</th>
<th>(V_L)</th>
<th>(V_T)</th>
<th>(\beta)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
<td>0.02</td>
<td>0.02</td>
<td>10</td>
<td>0.2</td>
</tr>
</tbody>
</table>
The detail of a PCNN neuron implementation is shown in Fig. 3. As mentioned above, the linking field is chosen as a 3x3 matrix leads to a synaptic weight matrix is selected as can be seen in the figure. With the chosen parameters, the complex factor in the model such as \( \exp(-\alpha P) \) can be considered as a simple factor \( \alpha G \). As a result, there is no real exponential functions are employed but the multiplication ones. The multiplier parameters \( \alpha G, \alpha L, \alpha R \) and \( \alpha T \) represent for \( \exp(-\alpha L), \exp(-\alpha R), \) and \( \exp(-\alpha T) \) in the equations, respectively. Furthermore, a CSD (i.e. canonical signed digit) multipliers method [15] was used to transform a fixed-point constant multiplying into an efficient "shift-and-add/subtract" algorithm.

3. Feature Extraction Using PCNN

3.1. Feature Vector

The feature vector is called as the time signature and proposed by Johnson [16]. As mentioned above, the output binary image which is formed by the network pulses \( Y_{ij}[n] \) contains much important information. In each iterative processing step \( n \), the PCNN produces a different output binary image. And a feature vector element \( G[n] \) is generated by summing the pixel values in that output binary image as shown in the following equation.

\[
G[n] = \sum_{i,j} Y_{ij}[n]
\]  

The length of the feature vector is defined as the total number of the PCNN iterative steps. For example, if PCNN iterates it’s processing N times, the feature vector will have the length of N elements. Thus, the range of \( n \) is between 1 and N values. The length of the vector \( G \) represents for the quality of the vector itself. In order to generate a high quality feature vector, the larger N is needed. As a consequence, it takes a long time for the operation to generate one feature vector. In this paper, the N number is chosen as 32 to implement the architectures.

3.2. Using Feature Vector for Object Recognition

Eq. 6 reduces the feature data from two-dimensional data (i.e. binary images) to one-dimensional data (i.e. feature vector), which is very suitable for a pattern recognition system such as object recognition. Different input images generate different feature vectors, with the result that a feature vector is unique for the input image. However, if two images contain similar objects, their feature vectors will be likely the same. Based on that concept, the feature vector which is generated by the PCNN algorithm can be used to recognize an object in the input image. Johnson [16] proves that the feature vector is anti-noise and invariant against the geometrical changes. For this reason, the PCNN model has strong advantages in the comparison with other object recognition algorithms. A typical object recognition system employing PCNN is shown in Fig. 4.

The feature vector generator employs the PCNN to extract features from the input image and produce a vector \( G \). Then, the search engine compares the vector with sample vectors in the templates in order to identify the object. The templates contain many sample feature vectors, and each vector represent for one object.

3.3. The Comparison between Feature Vectors

The MSE (i.e. mean square error) method is used to quantitative the differences between two feature vectors. The MSE value is calculated by Eq. 7. The small MSE value means two vectors are similar and the object can be recognized directly.

\[
MSE = \frac{1}{N} \sum_{n=1}^{N} [E_g(p) - E_y(p)]^2
\]  

where, \( n \) and \( N \) are the iterative step and the feature vector's length, respectively. \( E_g(p) \) and \( E_y(p) \) are two entropy vectors which are transformed from the feature vectors by the following equations.

\[
p_i[n] = \frac{G[n]}{P_{HV}}
\]  

\[
p_0[n] = 1 - \frac{G[n]}{P_{HV}}
\]  

\[
E_y(p) = -p_i[n] \log_2 p_i[n] - p_0[n] \log_2 p_0[n]
\]  

4. Proposed Hardware Architectures

Both RAM-based and pipelined models use the neuron implementation in sec. 2.2 as groundwork to build the architectures. The input image size is 128x128 pixels with 8-bit gray-scale level. The feature vector's length is chosen as 32 elements.

4.1. RAM-based PCNN Architecture

4.1.1. Idea

As mentioned above, the PCNN model is a two-dimensional network with a 1:1 corresponding between image pixels and network neurons. Hence, the original
PCNN implementation is formed by the \( P_{HV} \) neurons, where \( P_{HV} \) is the total pixels in the image. The idea of this architecture is to use a few neurons for the computation and RAMs for storing the network information such as input, output, and variables values. As a result, instead of implementing \( P_{HV} \) neurons, the RAM-based model can use one neuron to do the feature vector generation.

### 4.1.2. Implementation

The top-view of the RAM-based PCNN module is shown in Fig. 5. First of all, the module is allowed to run whenever the iStart signal is asserted for one clock cycle. Then, the input image is written into the module by two signals: iWrite and iWrite_data. Later, the module generates a feature vector by transmitting one vector's element at a time using oG_valid and oG_data signals. Finally, the oDone signal is asserted for one clock cycle whenever 32 elements of the feature vector have been transmitted.

![Figure 5 Top-view of the RAM-based PCNN module](image)

In the LP, the Block Neuron, Counter G, and the Controller do the PCNN computation, sum the output to generate a vector, and control the operation, respectively. The Controller controls three RAMs address while RAMs data are connected to the Block Neuron. The Counter G sum the output pulses \( Y_{ij}[n] \) from the Block Neuron to computes \( G[n] \). The Block Neuron is a group of a few neurons which its implementation is described in sec. 2.2. The signals' waveform of the operation is shown in Fig. 7. \( S_{ij} \), where \( i \) and \( j \) from 1 to 128, is the input pixel value at \( (i,j) \) position. \( A_i \) is the RAMs' addresses which store the information of the neuron \( (i,j) \). \( R_{ij} \) and \( W_{ij} \) are the neuron \( (i,j) \) information including pixel value \( S_{ij} \), output pulse \( Y_{ij} \), feeding \( F_{ij} \), linking \( L_{ij} \), and threshold \( T_{ij} \). \( R_{ij} \) is the read data from previous step (i.e. step \( n-1 \)), while \( W_{ij} \) is the read data in the current step (i.e. step \( n \)). \( G[n] \) is the feature vector element generated in step \( n \).

![Figure 6 RAM-based PCNN block diagram](image)

The block diagram of the architecture is shown in Fig. 6. The system is composed of two parts: the memory part (MP) and the logic part (LP). MP contains three RAMs to stores the input image \( S_{ij} \) (RAM_S), the internal variables such as \( F_{ij}[n], L_{ij}[n], T_{ij}[n] \) (RAM_FLT), and the output binary image \( Y_{ij}[n] \) (RAM_Y). Each RAM has the same memory cells (i.e. PHV) but their cell width because the widths of their stored signals are different. RAM_S, RAM_FLT, and RAM_Y have 8 bits, 28 bits and 1 bit per memory cell, respectively. Consequently, the total RAMs' capacity come from Eq. 11. It is clear that the RAM resources cost in the model depends on the input image size only.

\[
B_{RAM} = 37 P_{HV} 
\]  

(11)

### 4.1.3. Speed optimization

Firstly, the Controller waits for the assertion of the input iStart signal to begin the operation at (A). After this, between (B) and (C), the input image is written to RAM_S. At (C), when the final pixel value \( S_{128,128} \) has been stored, the Controller reads the image from RAMs by asserting the RAMs' read enables and RAMs' read addresses. Then, the read data \( R_{ij} \), also the Block Neuron's input, are transferred by the three RAMs. At (D), the Controller asserts three RAMs' write enables and RAMs' write addresses when the write data \( W_{ij} \), also the Block Neuron's output, is valid. Between (C) and (F), the previous step data (i.e. step \( n-1 \)) are read and processed in the Block Neuron, then the new data are written back into RAMs as the current step data (i.e. step \( n \)). After the Controller finishes the reading at (E), there are two more clock cycles to finish the writing at (F). The oG_valid signal is asserted for one clock cycle at (F) to give the feature vector element \( G[1] \) on the oG_data signal. The process of generating the \( G[1] \) value is done at (G). Simultaneously, the new process of generating the \( G[2] \) value is begun. The process between (C) and (G) is repeated for more 31 times in order to generate a full-length feature vector with 32 elements. As soon as the last feature vector element \( G[32] \) is valid at (H), the oDone signal is asserted for one clock cycle as same as the oG_valid signal. Finally, the operation is finished at (I), and a 32-element feature vector is already generated. Then, the RAM-based PCNN module returns to the initial state at (A).
The Block Neuron contains one neuron implementation, as a result of which there are $P_{iv} + 3$ clock cycles between (C) and (G) in Fig. 7 for an element generation. Because of this, the total clock cycles for 32-element feature vector generation are numerous. In order to process in real-time, the Block Neuron contains a group of neurons to speed up the operation. For example, the Block Neuron implements a block of 2x2 neurons, the LP do not process one neuron per clock cycle but a group of 4 neurons instead. Then, the number of clock cycles between (C) and (E) is a quarter of $P_{iv}$. Therefore, the operation time is reduced approximately four times. Because the LP processes a block of 2x2 neurons per clock cycle, each memory cell in RAMs has to store 4 neurons’ information. This leads to the wider cell width but fewer total memory cells. As a result, the total RAMs’ capacity in Eq. 11 remain constant. For faster implementation, the Block Neuron not only contains a block of 2x2 neurons but also a block of 4x4 neurons, 8x8 neurons, etc. In a word, there are multiple choices of implementing the RAM-based PCNN model and it depends on the speed requirement.

4.2 Pipelined PCNN Architecture

4.2.1 Idea

The pipelined model is formed by $N$ stages as shown in Fig. 8, where $N$ is the feature vector’s length. Each stage receives the data from the previous stage, generates one feature vector’s element, and transmits its data to the next stage. Only the registers are used in the model for buffering the data between stages.

4.2.2 Implementation

The top-view of the pipelined PCNN module is shown in Fig. 8. iData_valid and iData signals carry the 8-bit gray-scale input image to the first stage. oG_valid and oG_data signals transmit the feature vector. Generating the feature vector is done whenever the oDone signal is asserted.

The top-view of a stage is shown in Fig. 10. Whenever iValid signal is asserted, a stage receives one neuron information from the previous stage including pixel values $S_i$ (iData_S), feeding input $F_{ij}[n-1]$ (iData_F), linking input $L_{ij}[n-1]$ (iData_L), threshold $T_{ij}[n-1]$ (iData_T), and pulse $Y_{ij}[n-1]$ (iY_data). Correspondingly, a set of signals such as oValid signal and output data signals transfer the stage information to the next stage. As soon as a stage has finished its work, it asserts the oDone signal and transmits the feature vector element on the oG signal.

In order to process the neuron $(i,j)$ in the step $n$, it requires the corresponding pixel value (i.e. $S_i$), step $n$ internal activities (i.e. $F_{ij}[n-1]$, $L_{ij}[n-1]$, $T_{ij}[n-1]$), and a linking field of 3x3 neurons (i.e. $Y_{i+1,j}[n-1]$, $Y_{i+1,j}[n-1]$, $Y_{i+1,j}[n-1]$, $Y_{i+1,j}[n-1]$, $Y_{i+1,j}[n-1]$, $Y_{i+1,j}[n-1]$). A stage processing is started with the neuron $(1,1)$ and finished with the neuron $(H,V)$, where $H$ and $V$ are the total column number and row number, respectively. With the input image size of 128x128 pixels, both $H$ and $V$ are equal to 128. The neuron $(1,1)$ is at the corner of the network which has the linking field of 2x2 neurons (i.e. $Y_{1,1}[n-1]$, $Y_{2,1}[n-1]$, $Y_{1,2}[n-1]$, $Y_{2,2}[n-1]$), with the result that the stage $n$ has to wait for the neuron $(2,2)$ information from the stage $n-1$ in order to start. Consequently, the stage $n$ processing can be started only when the stage $n-1$ has processed H+2 neurons. Hence, $N$ stages’ processing times are alternated as shown in Fig. 11.

The stage 1 starts first. After the stage 1 has processed H+2 neurons, the stage 2 has enough information to start. Then, the stage 3 starts only when the stage 2 has processed H+2 neurons, and so on. The stage 1 finishes and produces $G[1]$ value when it processed the last neuron, $(H,V)$ neuron. This means that the stage 1 has processed $P_{iv}$ neurons in total, where $P_{iv}$ is the total neurons in the network. When the stage 1 is done, the stage 2 has the last H+2 neurons to process. For this reason, the stage 2 is done after the stage 1 H+2 neuron processing times. Then, stage 3 is done after the stage 2 H+2 neuron processing times, and so on. The operation is done when the last stage is done, and one 32-element feature vector is successfully generated.

It is clear that a stage needs buffers to store (H+2) neuron information. The stage design is shown in Fig. 12. There are five buffers in a stage to store pixel value $S_i$ (S buffer), feeding input $F_{ij}[n-1]$ (F buffer), linking input $L_{ij}[n-1]$ (L buffer), threshold $T_{ij}[n-1]$ (T buffer), and 3x3 matrix linking field $Y_{ij}[n-1]$ (Y buffer). As mentioned
above, buffers are constructed by registers only. The pixel values buffers and internal variables buffers utilize \((H+2)\) registers for each buffer. The linking field is a 3x3 matrix, with the result that three lines buffers are required as can be seen in the figure. The data widths of registers depend on the type of the signal which they carry. Hence, the total buffers capacity in one stage can be calculated in Eq. 12.

\[
B_{\text{Stage}} (\text{bit}) = 39h + 72
\]  

(12)

Besides the buffers, the stage contains one PCNN neuron, a Counter, and a Controller for the neuron computation, sum the output to produce an element value, and control the stage’s activities, respectively.

### 4.2.3 Area optimization

The first and the last stage are designed differently in the comparison with the design in Fig. 10. Owing to neuron’s initial values are zeros, stage 1 needs only the input pixel values to operate. Therefore, there are no buffers in the first stage. In addition, the input signals of the first stage are \(\text{iData}_S\) and \(\text{iValid}\) signals only. The last stage’s output signals are the \(\text{oY}_\text{data}\) only because it does not transmit the stage data. The result of stage 1 needs only the pixel value is to the buffers reduction in that stage. Therefore, The entire buffers capacity in the pipelined PCNN module can be calculated in Eq. 13.

\[
B_{\text{Pipelined}} (\text{bit}) = (N - 1)B_{\text{Stage}}
\]  

(13)

### 4.3 Experimental Results

Both models have approximate processing speed but the resources cost. The differences between two models' resources costs come from Eq. 11 and Eq. 13. As can be seen in two equations, the resources cost of the RAM-based model depends on the input image size, \(P_{HV}\), and that of the pipelined model mainly depends on the feature vector length, \(N\). Based on the specific requirement which requires the larger input image size or more feature vector elements, the better model can be selected.

Two models are verified by the Modelsim simulator and implemented on Altera Stratix III board with an EP3SL150F1152C2 FPGA chip. The input image size and the feature vector length are chosen as 128x128 pixels and 32 elements, respectively. Tab. 2 shows the resources utilization and the performance of two architectures. As mentioned above, the RAM-based model has multiple choices for implementation such as one neuron, block 2x2 neurons, block 4x4 neurons, and block 8x8 neurons with the different resources usage. The last row in the table shows the required processing time which is needed for one 32-element feature vector generation in each model. For example, as the pipelined model needs 289 ms to generate a feature vector when using a maximum clock frequency of 70.61 MHz, the model achieves the speed of approximate 3460 vectors per second. Tab. 2 shows that the RAM-based model with larger number of neurons produces vector faster but costs more ALUTs and registers. As mentioned above, the total RAM capacity in the RAM-based model depends on the input image size only, although the model implements more neurons in the Block Neuron. As a result, the
memory costs of the RAM-based model in various options remain constant as can be seen in the table. The pipelined model does not implement RAMs causes a zero value in the memory costs. However, the register costs are numerous.

Fig. 13 shows the comparison between the three feature vectors: the ideal vector is calculated by Matlab and others are obtained from the hardware designs. It is clear that both models generate feature vectors which are similar to the ideal one. As mentioned above, the feature vector is anti-noise and invariant against the geometrical changes. This is proved in Fig. 14. The Lena images are affected by the Gaussian noise ($\sigma^2 = 0.01$), the impulse noise (5%), and 300 angle rotation, but their feature vectors are similar as can be seen in the figure. The feature vectors which are used in the example are generated by the pipelined PCNN architecture. In conclusion, the proposed implementations generate the feature vectors which are similar to the ideal vector and strongly robust against noise and geometrical changes in the input image.

![Lena images and feature vectors](image)

**Figure 13** The variation of the Lena images and their feature vectors

5. AN OBJECT-RECOGNITION SYSTEM USING PCNN ALGORITHM

5.1. System Overview

In order to verify the operation of the models, a completed system has been built including a camera, an FPGA board, an off-chip DDR2 RAM, and a monitor. The block diagram of the object-recognition system is shown in Fig. 15. The Camera transmits the image which has the full HD resolution (i.e. 1920x1080 pixels) and 24-bit RGB color. The Camera Controller receives an image then transfers it to the Image Pre-processing and the off-chip DDR2 RAM. The Image Pre-processing reduces the input image to 128x128 pixels with 8-bit gray-scale level. On the other hand, the original image with full resolution is stored in the DDR2 RAM by the DDR2 Controller. The Feature Vector Generation which is formed by one of the proposed architectures processes on the reduced image in order to generate a 32-element feature vector. Then, the Search Engine identifies an object by comparing the received feature vector with the samples. Each of the feature vector samples which are stored in the Samples RAM represents for a specific object such as a computer mouse, a car, a chair, a clock, etc. As soon as the Search Engine had finished its work, it writes the result and the input feature vector into the Result RAM. Finally, the DVI controller reads the original image from the off-chip DDR2 RAM and the feature vector from the Result RAM then displays them both on the monitor.

![Object-recognition system block diagram](image)

**Figure 14** The object-recognition system block diagram

<table>
<thead>
<tr>
<th>Input Image</th>
<th>MSE = 0.0377</th>
<th>MSE = 0.0669</th>
<th>MSE = 0.0146</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MSE = 0.1254</td>
<td>MSE = 0.0542</td>
<td>MSE = 0.0197</td>
</tr>
<tr>
<td></td>
<td>MSE = 0.0042</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 15** The MSE values between the input image and the sample images

![Completed system](image)

**Figure 16** The completed system

5.2 Object-recognition System Experimental Results
As mentioned above, each input image has a unique feature vector which represents for the object in that image. However, images contain similar objects will generate similar vectors. The comparison between vectors is done by the MSE value which is presented in Sec. 3.3. With an input image from the camera, the MSE values can be calculated for each pair of the feature vectors. Fig. 16 shows an input image and templates with 7 sample images. A MSE value below a sample image is obtained by the comparison between the feature vector of that sample image and the feature vector generated from the input image. If the input image contains an object which is similar with any sample objects in the templates, their MSE value will smallest. Therefore, the system recognizes an object by finding the smallest MSE value via a winner-take-all circuit. It can be seen in the figure that the MSE value between the input image and the computer mouse sample image is the smallest one. Then, the system recognizes the input object as a computer mouse.

A completed system has been built and tested successfully on an Altera Stratix III board with an EP3SL150F1152C2 FPGA chip. As can be seen in Fig. 17, the object-recognition system is processing on a computer mouse image which is recorded by camera. The DVI port transmits the image and the feature vector from the board to the monitor. The feature vector which represents for the computer mouse is on the top-left corner of the monitor. With the speed of more than 2000 vectors per second, the system is suitable for real-time applications.

6. CONCLUSION

We have introduced two real-time hardware architectures for a feature vector generation based on the PCNN algorithm, and applied them to an object-recognition system in order to verify the operation the models. They are RAM-based model and pipelined model. As the input image size is 128x128 pixels and the feature vector's length is 32 elements, both models can generate a feature vector at the speed of more than 2000 vectors per second when using a clock frequency of 50 MHz. Additionally, the speed of the models is about 106 vectors per second with the image size of VGA resolution (i.e. 640x480 pixels) and the same feature vector's length. Two architectures have been implemented and tested successfully on the Altera Stratix III board with an EP3SL150F1152C2 FPGA chip. The resources costs of the RAM-based model and the pipelined model primary depend on the input image size and the feature vector's length, respectively. Based on the specific requirement which requires the larger input image size or more feature vector elements, the better model can be selected. The experimental results of two architectures show the significant advantages of the PCNN model such as anti-noise and invariant against the geometrical changes in the input image. Besides, the PCNN algorithm has the strong advantage which does not require any training in order to operate. The PCNN model has been proved that it can be used as a feature vector generator in such a pattern recognition system.

Acknowledgements

This work was supported by the University of Science, Vietnam National University – Ho Chi Minh City (VNU-HCMUS) under the fund No. B2011-18-04.

References


