International Journal of Modern Engineering Research (IJMER)
www.ijmer.com Vol.2, Issue.4, July-Aug 2012 pp-2821-2824 ISSN: 2249-6645

Design of a Neural Network Based Image Recognition System Using Configurable VLSI

Vidya G. S [1] , Sarath Raj S[2], Manu T S[3]
[1, 2, 3](Dept.of Electronics and Communication, TKM Institute of Technology, Affiliated to CUSAT, India.)

Abstract: *This paper describes the initial steps in the development of an object detection system for manipulation purposes to be embedded in a mobile robot. The goal is to design a neural network based recognition module. The neural network module and additional image processing algorithms which are used to convert the image into useful information for the neural network and the control of the whole system is designed using the soft core processor in the FPGA. The neural network implementation can be performed using the VHDL coding and processor can be designed using the Xilinx EDK tool.*

Keywords: *EDK, embedded systems, FPGA, image processing, Neural Network, SoPC, visual servoing*

I. INTRODUCTION

Image recognition systems are widely used in different industries such as production plants to detect faulty components, to select a piece on a conveyor or as surveillance systems that are capable of detecting intrusion, differentiating people or observing their motion. Object positions and environmental conditions have to be acquired in real-time. The term Visual Servoing refers to a useful capability for both manipulator arms and mobile robots [1]. Visual Servoing involves moving a robot or some part of a robot to a desired position using visual feedback [2].

However, fast and computation intensive tasks are difficult to implement in small and low power consumption electronic systems required in robot-like systems. The goal of this research is to develop an efficient hardware/software implementation of an object recognition system for an autonomous robot. This recognition system is based on an artificial neural network. In addition, some image processing modules to provide the network with useful data have been designed. Some other works that use neural network based systems for Visual Servoing can be found in [3], [4], [5]. However, in general most of such implementations are PC-based architectures.

The implementation presented here is carried out in a FPGA (Field Programmable Gate Array). The very high integration of present FPGAs enable the accommodation of all the components of a typical embedded system (processor core, memory blocks, peripherals, specific hardware,...) on a single chip, commonly referred to as system-on-a-programmable chip (SoPC). The design described here is based on such a SoPC. In particular the neural network module together with the control of the whole system are implemented as software in the embedded processor core

II. SYSTEM ARCHITECTURE

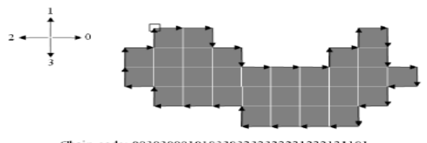
The initial approach to afford the recognition problem has been limited to the recognition of simple shapes, but in such a way that it could be extrapolated to any shape, for example those of hand tools. For the experiments performed up to now, some different colour wood pieces (cubes, cylinders, rectangular prisms and triangular prisms) have been used. Hence, four possible shapes have to be recognized: square, circle, rectangle and triangle.

Usually, image processing algorithms are implemented in software and run on a PC. However, in applications with high restrictions in response time or low consumption requirements (like the system described here), hardware specific implementations are needed . The main objection of the image recognition techniques for its realization in hardware is the high complexity of the existent algorithms. For this reason, in this paper a method optimized for its hardware implementation is presented.

2.1. IMAGE PRE-PROCESSING

The pre-processing stage converts the images into useful information for the neural classification system. Once a binary image is obtained, the amount of information contained in it is reduced to preserve only the information considered more relevant for the recognition. An edge extraction technique grounded on the chain-code algorithm [6] has been chosen. The bases of the chain-code algorithm were introduced in 1961 by H. Freeman [7], who described a method which permits the encoding of arbitrary geometric configurations, as a way to make it easier for a digital computer to manipulate them. It is a lossless compression algorithm for binary images, which provides a useful way to depict an object and to derive its features for later applications in pattern recognition.

Chain-codes are used to represent the contour of an object by means of a sequence of small vectors of unit length, each one representing the direction of the contour at that point. The number of possible directions is determined, being the 8-connected neighborhood and the 4-connected neighborhood configurations the most commonly used. The 4-connected set of directions, also referred to as external chain-code or crack code in some sources, is the one employed in this work (Figure 1).



Chain-code: 003030001010330323232222122212101

Figure1: Example of the external chain code of a binary image

First of all, the region of the image where the object lays must be determined, in terms of the density of white pixels. Then, the origin of the object in the image must be fixed. In the algorithm presented in this paper, the origin is considered to be the left-most white pixel of the first line in the object region. Once the origin is fixed, the object is outlined in clockwise manner and the directions of the boundary are stored until the algorithm reaches back the initial point.

Using this algorithm, each object is represented by a sequence of numbers, which length is different for each case, depending on the size of the object in the image and its shape.

In order to make the lengths equal and to reduce even more the codification, in such a way that it can be used as the input for the neural network, the sequence is normalized by dividing it into a fixed number of smaller sequences. Each of them is processed to obtain the slope between its end points. Thus, each object is represented by a fixed length sequence which contains the slopes of the contour. In turn, these slopes can only take a definite number of values, so that the translation to a digital system is more direct.

2.2. NEURAL CLASSIFICATION

The classification module consists of an artificial neural network where the inputs are the values provided by the image pre-processing stage. Based on these data the neural network classifies the shape of the target object. The neural network has multi-layer perceptron architecture [8], consisting of an input layer, a hidden layer and an output layer.

The number of input neurons has been set to 16, which forces the sequence obtained from the image processing stage to be of this length. The hidden layer has 32 neurons with a tan-sigmoid activation function. Lastly, the output layer consists of 4 output neurons, one for each possible shape, and no activation function is applied. The architecture of the neural network is presented in Figure 2.

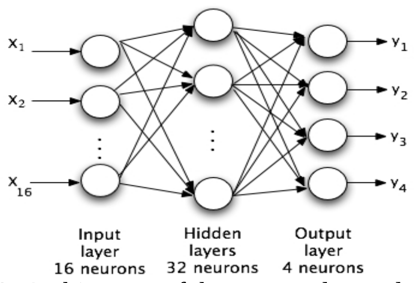


Figure 2: Architecture of the proposed neural network

The network is trained by means of the back-propagation algorithm, using the gradient-descendent method. The training set is made up of 16 sequences obtained from different images of different object shapes, along with the corresponding target output for each sequence. This set is divided into three different subsets: the 60% of the samples are used for training, the 20% for validation (useful for early-stopping of the training process) and the remaining 20% for test (for estimating the network s ability to generalize). The training algorithm of the network is performed in Matlab, by means of the Neural Network Toolbox [9].

III. HARDWARE/SOFTWARE PARTITION

Nowadays, the so called SoPC (system-on-a-programmable chip) take advantage from the flexibility of software and the high performance of hardware. Their proliferation has been possible thanks to the high integration levels achieved in the microelectronic industry, which allow the inclusion of a small microprocessor inside the programmable chip. This fact allows the designing of efficient heterogeneous hardware/software architectures on a single chip. Historically, the most common way for the implementation of neural networks has been a program running on a personal computer or a workstation. This is due to the fact that software implementations offer a high flexibility and give the users the possibility of modifying the topology of the network, the type of the processing elements or the learning rules, according to the requirements of their application. However, biological neural networks, in which artificial neural networks are inspired, operate highly in parallel. Hence, implementing them on a sequential computer does not seem the most efficient way to do it.

Dedicated hardware implementations, on the other hand, offer a number of important advantages, because they exploit the inherent parallelism of neural networks and also are much faster and robust if compared to software solutions.

Furthermore, they provide a physically reduced and low-power solution, useful for applications where including a personal computer or a workstation might not be feasible (such as the case of autonomous robots). These are the main reasons why it has been decided to implement the recognition algorithms on an embedded system and, more specifically, the neural network on the hardware partition of the system.

The architecture is as shown in Figure3..

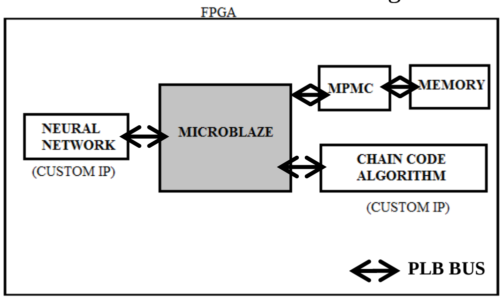


Figure 3: Internal Architecture of SoPC

The software partition is built on a MicroBlaze (the softcore processor from Xilinx) [10] and includes the control of the complete system and, also the image pre-processing algorithms.

3.1. INTERFACE BETWEEN MODULES

The interface between both partitions is based on the PLB (Processor Local Bus) bus [11], that provides a fast and efficient communication mechanism. The Processor Local Bus (PLB) consists of a bus control unit, a watchdog timer, and separate address, write and read data path units with a three-cycle-only arbitration feature. The PLB supports read and write data transfers between master and slave devices equipped with a PLB bus interface and connected through PLB signals. Bus architecture supports multiple master and slave devices. Each PLB master is attached to the PLB through separate address, read-data, and write-data buses. PLB slaves are attached to the PLB through shared, but decoupled, address, read-data, and write-data buses and a plurality of transfer control and status signals for each data bus.

The Processor Local Bus (PLB) consists of a bus control unit, a watchdog timer, and separate address, write and read data path units with a three-cycle-only arbitration feature. The PLB supports read and write data transfers between master and slave devices equipped with a PLB bus interface and connected through PLB signals. Bus architecture supports multiple master and slave devices. Each PLB master is attached to the PLB through separate address, read-data, and write-data buses. PLB slaves are attached to the PLB through shared, but decoupled, address, read-data, and write-data buses and a plurality of transfer control and status signals for each data bus.

Access to the PLB is granted through a central arbitration mechanism that allows masters to compete for bus ownership. This arbitration mechanism is flexible enough to provide for the implementation of various priority schemes. Additionally, an arbitration locking mechanism is provided to support master-driven atomic operations. PLB arbiters can be implemented on the FPGA fabric and are available as soft IP cores. The PLB is a fully synchronous bus.

The PLB arbiter multiplexes signals from masters onto a shared bus to which all the inputs of the slaves are connected. The PLB arbiter handles bus arbitration and the movement of data and control signals between masters and slaves. The PLB-to-PLB bridge is required when two PLB segments are connected. The bridge translates PLB transactions on one side into the PLB transactions of the other side. The bridge functions as a slave on one PLB side and a master on the other**.** For a typical system with two PLB segments, one bridge is necessary for transactions originating from the processor. A second bridge is required if a peripheral on the other side is master capable and wants to address a peripheral on the processor side. Figure4 provides an example of the PLB connections for a system with three masters and three slaves.

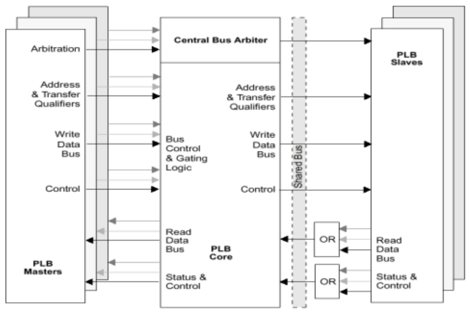


Figure 4: PLB Interconnection Diagram

3.2 ARCHITECTURE OF THE NEURAL NETWORK

The architecture of the network is the one presented in Figure 2. In a hardware implemented neural network, the processing elements (i.e., neurons) have to be independent and operate in parallel. They should be designed in such a way that their internal calculations are optimized, while they should be so simple that the chip area occupied by them is the minimum possible. Following these requirements, a very small but high performance system can be achieved.

The architecture proposed in this paper comprises the following modules.

- A two-layer processing module: the hidden layer and the output layer (the input layer merely transmits the inputs)
- Three ROM modules, which store the network parameters (weights) for the hidden layer, the output layer and the sigmoid function, respectively.
- Additional components, such as a multiplexer and a block that calculates the maximum of its inputs.
- A circuit controller that governs the whole operation of the system.

The main component of the processing module is the neuron, which is just a MAC (multiply-accumulate) block. The MAC is loaded with an initial value (offset or bias) and then multiplies each input with its corresponding weight and accumulates these values to obtain the sum of all them. It is a two-cycle synchronous component (see Figure 5). The total number of these MAC blocks is 36 (32 for the hidden layer and 4 for the output layer).

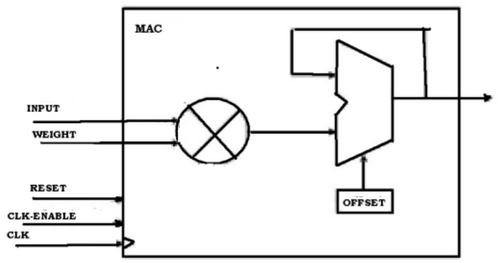


Figure 5: MAC schematic. .

As for the ROM modules, the one that contains the weights of the first layer (ROM1) has a size of 512 weights with a word length of 12 bits, whilst the one corresponding to the second layer (ROM2) contains 128 weights of a word length of 8 bits. ROM1 is organized in 16 blocks of 32 weights, so that for each of the inputs the corresponding block of 32 weights is addressed and sent to the first layer of neurons. In the same way, ROM2 is divided into 32 blocks of 4 weights each. Finally, ROM3 is the memory that stores the pre-computed sigmoid function and contains 256 values with a word length of 8 bits.

The system controller, whose main component is a six-bit counter, provides the control signals for the whole system. Such signals are the reset signals for all the modules, the signals to enable each block, the address signals for ROM1and ROM2 and the selection signal for the multiplexer.

The detailed operation of the whole system is described next. The 16 input data come serially through the PLB bus. Each neuron (MAC block) receives the inputs serially, multiplies each of them with the corresponding weight (stored in the ROM1 memory) and adds them up. The neuron needs only one clock cycle per input to process the MAC operation, because while the accumulate operation is being done, the next data are already being multiplied, creating a pipeline. Furthermore, the 32 neurons of the layer work in parallel. Hence, only 17 clock cycles (one for each input and one more for the first data, before starting the pipeline operation) are needed to perform the calculations of the first layer, in spite of the fact that the inputs enter the system serially.

Once the outputs of the first layer are available, they are used to address ROM3, which contains the activation function. This memory is the same for all the neurons of the first layer. The outputs of this block are the sigmoid functions of their inputs. All the accesses to the ROM3 are made in parallel, so just a clock cycle is required. This ROM module provides 32 outputs that act as the inputs to a 32 to 1 multiplexer. The multiplexer makes it possible for the inputs to the following layer of neurons to arrive serially, in such a way that this layer would work like the first one. Thus, each of the 4 neurons of the last layer receives the 32 incoming data serially and performs the MAC operation, needing 33 clock cycles to finish this task (one for each input and an additional one, as in the previous layer).

Finally, the result of these MAC operations are carried to a module that calculates which of them has the maximum value, needing only one cycle to do so. The output of this module represents the shape recognized by the network, codified in 2 bits. This data is sent back to the software partition through the PLB bus.

The final output showing the recognition of four basic shapes is as shown in Figure 6, which is obtained using the Chipscope Pro tool.

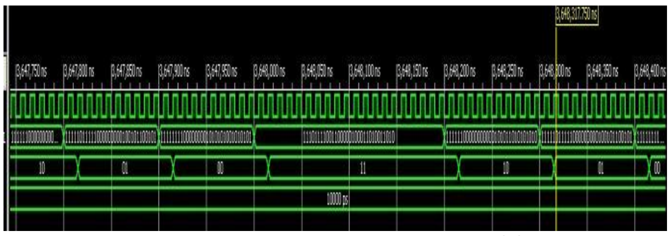


Figure 6: Results showing the recognition of four basic shapes

IV. CONCLUSION AND FUTURE SCOPE

In this paper a prototype of a vision system for a robotic platform to assist in manipulation activities has been presented. An FPGA module for embedding the object recognition module within a robotic mobile platform is being designed. The FPGA module includes the neural network and the control and the image processing modules are built on a Microblaze. More work will be done to strengthen the overall performance of this system, taking into account more variability in object shape and colour (real objects). Up to now, the FPGA module includes the implementation of the neural network. As further work, the rest of the image processing algorithms should also be implemented on the chip. They would be included preferably on the hardware partition of the SoPC for performance reasons, but to do so, a previous analysis has to be made in order to study the feasibility of this option. In addition, the whole system has to be integrated with the robotic platform in order to perform the manipulation activities.

ACKNOWLEDGEMENT

We would like to express my sincere gratitude to the Principal, Head of the department, all teaching and non teaching staffs of Electronics and Comunication department of TKMIT for their guidance and valuable suggestions for the successful completion of this paper.

REFERENCES

[1] Maria Isabel de la Fuente1, Javier Echanobe2, Inés del Campo2, LoretoSusperregui1, Iñaki Maurtua, Hardware implementation of a Neural-Network Recognition module for Visual Servoing in a Mobile Robot. *Proceedings of the 6th International Conference on Hybrid Intelligent Systems* (HISá10), Auckland, New Zealand, Pages 226-232, November 2010.

[2] Wells G., Venailleb C., Torrasa C., *Vision-based robot positioning using Neural Networks, Image and Vision Computing*, Elsevier B.V., Volume 14, Issue 10, December 1996, Pages 715-732.

[3] Mutlu Avcý, Tulay Yýldýrým, "Generation of Tangent Hyperbolic Sigmoid Function for Microcontroller Based Digital Implementation of Neural Networks", *International XII. Turkish Symposium on Artificial Intelligence and Neural Networks, 2003.*

[4] Nasri Sulaiman, Zeyad Assi Obaid, M. H. Marhaban and M. N. Hamidon Design and Implementation of FPGA-Based Systems - A Review *Malaysia, 43400 UPM Serdang,* Selangor Darul Ehsan, Malaysia.

[5] Haitham Kareem Ali and Esraa Zeki Mohammed, Design Artificial Neural Network Using FPGA, *IJCSNS International Journal of Computer Science and Network Security*, VOL.10 No.8, August 2010.

[6] Walid Shahab, Hazem Al-Otum, and Farouq Al-Ghoul, A Modified 2D Chain Code Algorithm for Object Segmentation and Contour Tracing,*The International Arab Journal of Information Technology,* Vol. 6, No. 3, July 2009.

[7] Freeman H, On the encoding of arbitrary geometric configurations. *IRE transactions on Electronic computers.*

[8] Omondi A. R., Rajpakse J. C. (Eds.*), FPGA Implementations of NeuralNetworks,* (Springer, 2006.)

[9] Demuth H., Beale M., Hagan M., *Neural Network Toolbox 6 User's Guide*, 2010, The MathWorks, Inc.

[10] *MicroBlaze™ RISC 32-Bit Soft Processor User Guide*, Product Brief, August 21,2002 Xilinx

[11] *Processor Local Bus User Guide*, Product Brief, August 21,2002 Xilinx

.