Using FPGA to Implement Artificial Neural Network to Drive a Vehicle Automatically

Behnam Ghiaseddin[1], Omid Rahmani Seryasat[1], Javad Haddadnia[2*]

[1,] Department of Electrical Engineering, Takestan branch, Islamic Azad University, Takestan, Iran
[2.] Associate Professor, Electrical and Computer Engineering Department, Hakim Sabzevari University & Center for Research of Advanced Medical Technologies, Sabzevar University of Medical Sciences, Sabzevar, Iran
[*]Corresponding author: Haddadnia@sttu.ac.ir

Abstract: Drive a Vehicle Automatically is consist of several concepts, like cruise control, automatic parking, and keep the vehicle in safe distance from other vehicle and etc. But to reaching full autonomous in vehicle driving the first issue is keeping the car in the road and controlling the speed base on the shape of the road. To acquire this capability we need an intelligence system that could think like a driver. One of the usual way to build such a system is achievable by using Artificial Neural Network (ANN). Using ANN, like most of other intelligent system, require lots of computation and using hardware implementation of this system could increase performance. We used Multilayer Perceptron as an ANN technique and Xilinx Spartan 3 FPGA as a hardware platform. We also use an open source simrace (Car Racing Simulator) application to simulate the functionality of the system. We tune the system to process 3 frames/sec but this is increasable. average error of steering angle accuracy is 1.2 degree which shows an acceptable result.

[Behnam Ghiaseddin, Omid Rahmani Seryasat, Javad Haddadnia. **Implementation of an Automatic Vehicle Driving System on a Single FPGA Chip**. *Life Sci J* 2013;10(5s):641-643] (ISSN:1097-8135). http://www.lifesciencesite.com. 112

Keywords: Automatic Vehicle Driving , FPGA, Neural Networks

1. Introduction

During the 1980s and early 1990s there was significant work in the design and implementation of hardware neurocomputers. Nevertheless, most of these efforts may be judged to have been unsuccessful: at no time have hardware neurocomputers been in wide use. On the other hand, gate-arrays of the period mentioned were never large enough nor fast enough for serious artificial-neural network (ANN) applications. But technology has now improved: the capacity and performance of current FPGAs are such that they present a much more realistic alternative [1]. Consequently neurocomputers based on FPGAs are now a much more practical proposition than they have been in the past. Generally, a pattern recognition system is implemented using software technology. However, the speed of software-based implementation is low, and software-based implementation relies on computer and is not suitable for using in the environments where high portability is required [2]. FPGAs are a form of programmable logic, which like software based systems, offer a good flexibility in design, but with performance speeds closer to Application Specific Integrated Circuits (ASICs). FPGAs can be reconfigured repeatedly, making them ideal prototyping tools for hardware designers [3]. Automatic vehicle driving is also a noticeable area in which many scientists has been work on[4,5].

In this paper we use an offline learning method. Offline training regards to learning procedure on a general-purpose computing platform before the trained system is implemented in hardware. Our proposed idea has two main parts: The first one is a Neural Network which decides for a suitable reaction of driver. The other one is the efficient implementation of the NN on a hardware platform. In continue the major parts of our proposed system are introduced in detail. The paper is organized as follows:

Section 2 explains the system design. Section 3 introduces the specifications of FPGA and Implementation Constrains. Section 4 presents the implementation of system and shows the result. Finally Section 5 concludes the paper and mention future works.

2. System Design

In a real driving process, the driver, see the scene and decide for pushing pedals and rotate steering wheel to drive as well. So the system should consist of three main parts: to get image as input, make a suitable decision and provide desired output.

Here we need a set of video frames that show the road as input. To simulate the driving condition, an open-source software called TORCS is used. It generates a road image in which a car can drive. Based on image get from input, our automated system should decide on how to drive in this road. Decisions are made using an artificial neural network which is implemented on FPGA. The output of FPGA will be sent to the software for driving in road. Steering wheel rotation and Desire Speed are the output parameters defined to drive the vehicle

automatically.

3. Preprocess of image

First of all, we need to gather data for training. To simulate the driving condition, an open source software called TORCS is used. It generates a road image in which a car can drive. This image is captured and saved while the value of pedals and car speed which were shown on image is also extracted from it. This is performed using a piece of code in MATLAB and finally extract the steer value, pedals and car speed as numbers. To decrease the processing cost, image size must be reduced to a 16x64 matrix. Then we used a segmentation method to extract the road from scene. For extracting the road, a 4-neighbor flood fill method was used. Finally, we would have a black-and-white image (or a matrix of 0 and 1s) as our network input which can really simplify the process.

4. Neural Network

As mentioned before, a Neural Network is designed for decision making. The most commonly used family of neural networks for pattern classification tasks is the feed-forward network. In this paper, we used a two-layer perceptron. Activation functions in hidden and output layer are chosen sigmoid and linear. Because of simplifications in input vector of network, the number of nodes in hidden layer is reduced. Nonetheless, due to number of input vector, weight matrix is still big and affects the computation and memory efficiency. Finally a 1024-10-2 multilayer Perceptron is designed to handle the main process of the system.

5. Network training.

Input data is used as a number of video frames and must be fed into network as a vector. So each 16x64 image frame has 1024 data which will be vectorized to input the NN. We used 2650 samples for test and Training algorithm is Levenberg-Marquart which has a high speed in process. After 5 epochs, network error is acceptable. Implementation Errors will be discussed in section 4 in detail.

6. Implementing NN on FPGA

To get a better result of neural networks specifications in parallel computation, we prefer to implement our algorithms on a hardware platform directly. For this purpose, we used our saved weight matrix of our trained network in FPGA. Obviously in this phase, an optimized implementation can strongly affect the result. For real-time use of system, first we should have an approximation for desired system speed. Consider this as a constraint, we can decide for optimized size of circuit. We try to minimize the number of bits for variables, to make them in a smaller area. Increase of bits can enlarge the circuit, not only in saving process, but also in computation phase. All the process is done in two phases:

Image segmentation, in which we extract road from image, and

Produce network output.

Each phase has its own process time shown with C_1 = 15360 clock and C_2 = 1079 clock. Therefore C_{total} becomes as follows:

$$C_{total} = C_1 + C_2 = 15360 + 1079 = 16439 \text{ clock} \quad (1)$$

The Master Clock is 50MHz, so total time for computation becomes 0.33 milliseconds.

$$\text{Latency} = \frac{C_{total}}{F_{MasterClock}} = \frac{16439 \text{ clock}}{50 \text{ MHz}} = 0.33 ms \quad (2)$$

So we can process 3041 process (image) per second:

$$PPS = \frac{1}{\text{Latency}} = 3041 \quad (3)$$

But there would be a delay for data transfer to FPGA and if it is considered, the PPS would decrease into 1354 process per second. The throttle, brake and steering wheel angles can be set by use of keyboard. So here we need to use a piece of code, to transform our NN outputs to the program and drive the car.

7. Hardware Constraints: Using Fixed Points

Although use of floating points has a better flexibility, their complexity can really cause the size of circuit to become large. In paper [2] authors have shown in their researches a great difference in implementation of a MLP with fixed and floating point variables. Their experiments showed that, fixed point implementation was 12 times faster and 13 times smaller than floating point implementation.

In first step, we need to calculate the inputs for activation functions:

$$WX + b \quad (4)$$

We can show it as below, in which, n is number of inputs:

$$\mathcal{W}_1\mathcal{X}_1 + \mathcal{W}_2\mathcal{X}_2 + \ldots + \mathcal{W}_n\mathcal{X}_n + b \quad (5)$$

As we are using the NN in offline mode, weight matrix is fixed; but input matrix (X) is changing over time. For implementing the activation function, we have used look up table. So, we can consider the constant values of the activation function in look up table. In Eq.6, it has been shown that each input should multiply in its related coefficient and be added to an accumulator.

$$\frac{\sum_{i=1}^{n} W_i' X_i' - W_b' \sum_{i=1}^{n} X_i' - \sum_{i=1}^{n} W_i' X_b + W_b' X_b}{b_a} + \frac{W_a' X_a \left(b' - b_b\right)}{W_a' X_a b_a} \quad (6)$$

Also, the inputs must be added to another accumulator without being multiplied. Finally the 2nd accumulator is multiplied to W_b and differentiates from the 1st accumulator. Although these can be done at the same time using a single accumulator, it can be decided, depends on implementing method of multipliers in hardware.

8. Test and Implementation Errors

For our network's first layer, error varies in [-1,+1] and in [-127,+127] for hardware neural network. Average of Absolute Error (E_a) and Relative Error(E_r) are gain from networks is:

$$Mean\left(Error\right) = E_a = 0.023 \quad (7)$$

$$E_r = \frac{E_a}{1} = 0.023 = 2.3\% \quad (8)$$

In Second layer also, these are same as first layer, but in 2nd layer, the absolute maximum is 15; and errors gained as:

$$Mean\left(Error\right) = E_{a2} = 0.4823 \quad (9)$$

$$E_{r2} = \frac{E_{a2}}{15} = \frac{0.4823}{15} = 0.0322 = 3.2\% \quad (10)$$

9. Conclusion & Future work

In This paper, we proposed a method for driving in a road. A simulator software was used for input and system reaction (steer & speed) as a virtual driver defined for system output. A multilayer perceptron were defined to decide as a virtual driver. The output result of network and errors has shown a good try for this purpose. For a better result in future, some key points are mentioned:

The road has been defined without any other vehicles on its path. We can train our network in a more complicated scene and get a more reliable result.

Two main parts of algorithm were image preprocesses and decision making based on neural networks. To get a better result, we implemented them on a FPGA chip. It is suggested to use RBF neural networks instead of MLP for a better result; as they are well defined for classification tasks.

Also we can use a fuzzy segmentation method to have a better extraction of road. This can help us to have a more accurate and precise map of road to decide.

Adding some other features and increase the input information of system is strongly advised. These features can be GPS (Global Positioning System), or vehicle acceleration in addition to its velocity that can make the system performance much more. As Automatic vehicle driving is one of the interesting topics of research these days, there is still lots of efforts must be done to establish an automated reliable system.

References

1. R. Omondi, J. C. Rajapakse, FPGA Implementations of Neural Networks, Springer, 2006.
2. Ahlander, and B. Svensson, *Floating point calculations in bit-serial SIMD computers*, Research Report, Centre for Computer Architecture, Halmstad University, 1992.
3. Almeida, L. D., *Backpropagation in perceptrons with feedback*, in NATO ASI Series: Neural Computers, Neuss, Federal Republic of Germany, 1987.
4. J.Windish. (October 2010). *Google's got an automated car,* themoderatevoice.com/88477/googles-got-an-automated-car/
5. The Official Google Blog, *What we're driving at,* googleblog.blogspot.com/2010/10/what-were-driving-at.html, Oct 2010

3/12/2013