MSc. Alireza Fasih

# Ultra fast CNN based Hardware Computing Platform Concepts for ADAS Visual Sensors and Evolutionary Mobile Robots

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der Technischen Wissenschaften

Alpen-Adria-Universität Klagenfurt

Fakultät für Technische Wissenschaften

**1. Begutachter:** Univ.-Prof. Dr.-Ing. Kyandoghere Kyamakya
Institut: Institut für Intelligente Systemtechnologien, Alpen-Adria-Universität Klagenfurt

**2. Begutachter:** Juniorprofessor Privat-Dozent Dr. habil. Zhong Li
Institut: Lehrgebiet INFORMATIONSTECHNIK, FernUniversität-Hagen in Hagen

May/2011

# EHRENWÖRTLICHE ERKLÄRUNG

Ich erkläre ehrenwörtlich, dass ich die vorliegende wissenschaftliche Arbeit selbständig angefertigt und die mit ihr unmittelbar verbundenen Tätigkeiten selbst erbracht habe. Ich erkläre weiters, dass ich keine anderen als die angegebenen Hilfsmittel benutzt habe. Alle aus gedruckten, ungedruckten oder dem Internet im Wortlaut oder im wesentlichen Inhalt übernommenen Formulierungen und Konzepte sind gemäß den Regeln für wissenschaftliche Arbeiten zitiert und durch Fußnoten bzw. durch andere genaue Quellenangaben gekennzeichnet.

Die während des Arbeitsvorganges gewährte Unterstützung einschließlich signifikanter Betreuungshinweise ist vollständig angegeben.

Die wissenschaftliche Arbeit ist noch keiner anderen Prüfungsbehörde vorgelegt worden. Diese Arbeit wurde in gedruckter und elektronischer Form abgegeben. Ich bestätige, dass der Inhalt der digitalen Version vollständig mit dem der gedruckten Version übereinstimmt.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Klagenfurt, am 25.5.2011

MSc. Alireza Fasih

## DEDICATION

This dissertation is dedicated to my parents *Mohammad Fasih* and *Mohtaram Mirkazemiha* and also to my brother *Mehrzad* and my sister *Mahnaz* who offered me unconditional support, encouragement and love.

## ACKNOWLEDGEMENT

It is my good fortune to have Prof. Kyamakya as my dissertation supervisor. He has been encouraging me by sharing with me his great ideas and his time. I am heartily thankful to him, whose support from the preliminary to the concluding level enabled me to develop an understanding of the subject.

# Contents

## List of Figures

## List of Abbreviations

| | | |
|---|---|---|
| ACC | .......................... | Adaptive Cruise Control |
| ADAS | .......................... | Advanced Driver Assistance System |
| ANN | .......................... | Artificial Neural Network |
| CNN | .......................... | Cellular Neural Networks |
| CPU | .......................... | Central Processing Unit |
| DDA | .......................... | Digital Differential Analyzer |
| DDR | .......................... | Double Data Rate – memory |
| DOF | .......................... | Degree of Freedom |
| DSP | .......................... | Digital Signal Processing |
| DT | .......................... | Discrete Time |
| FPGA | .......................... | Field-programmable Gate Array |
| GA | .......................... | Genetic Algorithm |
| GPU | .......................... | Graphics Processing Unit |
| HDL | .......................... | Hardware Description Language |
| ISE | .......................... | Integrated Software Environment |
| LDW | .......................... | Lane Departure Warning |
| MLP | .......................... | Multi-layer Perceptron Neural Network |
| ODE | .......................... | Ordinary Differential Equations |
| OpenCL | .......................... | Open Computing Language |
| OpenCV | .......................... | Open Source Computer Vision |
| PDE | .......................... | Partial Differential Equation |
| PLB | .......................... | Processor Local Bus |
| UM | .......................... | Universal Machine |

## ABSTRACT

Vehicle driving and traveling with car are part of our daily life. More than 80% of all personal travels are done by car and only 20% are done by public transportation. Alone in Europe every year we have more than 3 million additional private cars on roads and highways. Due to the congestion and capacity limitations of both highways and roads every year we have more than 7500 kilometers of blocked roads in Europe. This congestion and traffic has a negative direct impact on the economy and on total social costs. The estimation of traffic congestion and traffic safety on social costs for European people is more than 130 billion euro per year; see Ref. [1-3] . Monitoring and controlling traffic and improving road safety can reduce this cost; but still every year 40,000 people die because of car accidents in Europe [1], [4-6] . There are two solutions for overcoming the critical issue of road safety: improving the driver safety education programs and improving the vehicle safety using advance technology like ADAS (Advanced Driver Assistance System). One of the main factors in car accidents and traffic safety is the human factor. If the driver is tired or asleep the probability of accident will dramatically increase. A convenient way to avoid these types of accidents is using an assistance system.

This thesis answers the following eight research questions which are related to the potential performance improvement of ADAS technology with respect to the involved real-time image processing:

- Research question 1: What are the hard requirements of ADAS concerning real-time image processing and design flexibility? How far do traditional approaches fail to satisfy these requirements?
- Research question 2: What are the major limitations of traditional high performance computing approaches if used to ensure “real-time” image processing?
- Research question 3: What is the huge potential of neurocomputing involving either traditional neural networks (NN) or cellular neural networks (CNN) for high-speed and flexible image processing for ADAS? Are there any limitations and how can these eventually be addressed?
- Research question 4: What are the major template calculation schemes of relevance for CNN based image processing? How can these calculations be performed in a real-time high performance computing context?

- Research question 5: How far can the advantages of "analog computing" be used/gained through an emulation of analog computing on digital hardware platforms like FPGA (for the benefit of an ultrafast image processing)?
- Research question 6: How far can an efficient implementation of CNN on FPGA and GPU be designed and implemented?
- Research question 7: How far can CNN be used/involved in an evolutionary computing/control context example (for illustration)?

To cover the research question-1, we have conducted a survey concerning different ADAS concepts. High definition cameras are playing a very important role in ADAS concept and almost every ADAS concept includes one camera or some form of visual radar. We could formulate the overall requirements that ADAS systems set to the image processing based sensor functionality. And to finish we have shown as an example that there are many common modules for different ADAS systems.
Concerning research question-2, we have shown the limitations of traditional/sequential computing concepts for processing high quality images in the ADAS context and did propose a parallel processing model based on CNN.

Concerning research question-3 we have shown the advantages of neuro-computing especially of CNN as a high performance computing system in terms of flexibility in design and robustness.

Research question-4 is considering different methods for CNN template calculation. We did pass a review of the related state-of-the-art before proposing a genetic algorithm based approach for the calculation of complex templates. This concept can be implemented in hardware, for example of FPGA along with the CNN processor system. This should ensure a performance in real-time.

Concerning research question-5 we have demonstrated that the advantage of analog computing can be used in a real time solution for solving complex dynamic systems. Further we have shown the implementation an “analog computing” emulation on a digital platform (FPGA); the implementation was based on the *Digital Differential Analyzer* (DDA) method which has shown to be thousand time faster than a CPU.

The research question 6 does concern the implementation of CNN in a discrete-time version on both FPGA and GPU. We have shown the advantages and drawbacks of each of the implementations. Overall we could realize a CNN implementation on both platforms and the performance was very good.

Concerning the last research question 7, we have shown the efficient use of CNN and a genetic algorithm for controlling a legged-robot in the form of a neuro-evolutionary technique. The results did clearly demonstrate the effectiveness of the approach.

The evaluation of image sequences with the purpose of extracting useful information (about the environment, vehicle situation and traffic) is the main and challenging issue in ADAS visual sensors. This process does however involve a huge computational effort. Therefore, to have a real time processing platform we need appropriate hardware and software architectures. Today, there are many platform options for machine vision: DSP, FPGA and GPU. The requirements related to image quality, image size and frame rate per second for a processing in hardware are increasing. Overall we do face a tradeoff between processing time, power consumption on one hand, and video quality and precision on the other hand. This challenge does motivate research related to new architectures and software algorithms for video and signal processing. Designing and implementing a “single task” image processing functionality in hardware in not a big issue. However, the “multi tasking” case is much challenging.  Having a good model for image processing will reduce hardware resources. A CNN model has this potential as by dynamically changing the related templates values we can change the functionality of the system and thereby without any further hardware or software explicit reconfiguration. Taking advantage of the inherent parallel processing nature of CNN processors a strong integration of both hardware and software can be ensured. This thesis does address two main challenging issues in ADAS technology: a) a universal model and architecture for a real time visual processing; and b) the implementation of a prototype system on both GPU and FPGA.

# Chapter 1

## 1. Introduction

### 1.1 Motivation and general context

Two of the main factors in transportation (urban and suburban traffic) are the issues of safety and efficiency (including fuel consumption). According to recorded statistics human fault is the major cause of road accidents [7, 8]. Most of the dangerous cases are highly depending on the behavior of the drivers and not on technical problems. Fatigue, micro-sleep and alcohol can dramatically increase the accident probability [8]. To overcome these core problems that influence the safety of the driving process  a combination of different technologies has been developed during the last couple of decades [8, 9]. The integrating platform for them is known as *Advanced Driver Assistance System* (ADAS). Generally, these technologies involve the combination of different sensors, controllers and actuators to either give a warning or take part of the control in emergency situations.

ADAS systems are highly dependent on the sensory data information, fusion modules and control intelligence. There are various sensors involved such as laser scanner, LIDAR, radar, ultrasonic sensors  and different type of cameras [10]. Generally, we can classify DAS systems in two main classes: active and passive.  Active ADAS can have a control over the humane decision. This means that in a critical situation the system will react and change the driving parameters such as speed and steering angle. Contrary passive systems do only provide warnings and information the driver (for example about a dangerous situation) [8, 10].

The most effective sensor which used in almost every new car is the "camera". Cameras can provide visual information about the scene and the overall situation of the car on the road. Depending on the specific ADAS solution different types of cameras can be involved such as night camera, HD-camera and 'high frame rates' cameras.

A historical look at the evolution of ADAS shows a quick introduction of these technologies in vehicles. Increasing the reliability of ADAS, increasing computing capacity and decreasing the price are very important factors for both developers and system producers. Most of the ADAS solutions are camera-based and are using video processing for monitoring either the road or the driver to detect abnormal behaviors during driving. By fusing various sensors (cameras, radars, laser scanners, etc.) the field of view is enlarged, and the perception precision of objects in the relevant regions for the driving process is significantly increased [11]. Fusing data and information with different levels of quality and sampling rates is another challenging issue in ADAS technology. Darms *et al.* have proposed a modular system architecture for sensor data processing and combining short range sensor, long range sensor, video information, actuators feedbacks, and vehicle dynamics sensors in ADAS technology [12]. In general, ADAS has three data processing levels which are sensor level, fusion level and application level. Figure 1 does show the abstract architecture of an ADAS system with respect to the different processing levels.

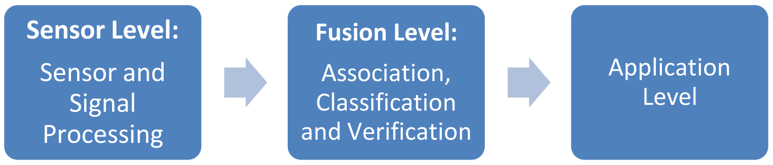

Sensor Level:
Sensor and Signal Processing
Fusion Level:
Association, Classification and Verification
Application Level

Figure 1-1: ADAS processing levels architecture

The following list does give a sample of functionalities provided by different ADAS solutions involving visual information and a digital camera:

- Lane departure warning system (LDWS)
- Traffic sign recognition
- Pedestrian detection
- Fatigue detection
- Adaptive cruise control

The large amount of data provided by cameras requires a huge processing effort. Speeding up the processing is therefore extremely important especially while facing real-time constraints [13]. Some algorithms such as stereo vision and depth estimation are particularly demanding in terms processing. Some algorithms have dependency and it is not trivial to speed up them by the using pipelining or parallel processing and it needs different model of processing [13].

## 1.2 Research questions and objectives of the thesis

In the following we do briefly describe and justify the key research questions of this thesis as well as the core of answers/solutions that have been obtained for each of them. Overall, we have formulated seven key questions concerning ADAS technology, importance of high performance computing and hardware implementation.

**Research question 1: What are the hard requirements of ADAS concerning real-time image processing and design flexibility? How far do traditional approaches fail to satisfy these requirements?**

One of the main issues to design an ADAS system is high processing speed and robustness of the environment perception (image processing based). The maximum interval for processing frames should not be greater than 20ms. In some cases such as lane departure warning and collision detection this level changes to maximum 10 ms processing time [14, 15]. The fundamental issue in all image processing systems is robustness and accuracy. In ADAS technology which is based on image processing and machine vision we have to guarantee the stability and robustness of detection, identification and recognition of features [16]. For example a lane departure warning system should work in any environmental condition and lighting [16, 17].

Concerning video- based ADAS solutions different image processing filters and appropriate hardware architectures are required. Some filters are very complex and very demanding in terms of processing run time. Hence, we are interested in heavy parallel processing and task concurrency. In most of different ADAS concepts we one is using the same processing modules with different connectivity and topology. Hence, if we have a reconfigurable architecture or a universal model we do not need to reserve hardware resources for each different concept.

While following traditional methodological ways of doing and design one does use/involve sequential processing architecture, time sharing and multi-threading algorithms/processing. The weakness of this traditional way of doing concerning processing performance is that it results in a 'too long' processing time and therefore making it very difficult to satisfy real-time constraints. The real-time constraints expect a completion of the processing of a frame within a time window that is less than the capturing time of that frame. Therefore, for a 60 FPS (frames per second) rate we do have a maximum of 15 milliseconds to finish all the processing. And since one does generally need about 6~10 different high definition (HD) image preprocessing modules whereby each of them takes around 5ms per frame we thus do reach a total ranging between 30 ms and 50ms of processing time if one tries to use the traditional processing schemes. This figure of 50 ms does fail to fulfill the real-time constraint of "maximum 15 ms" processing time per frame. It is therefore clear that traditional processing schemes are not capable of fulfilling the real-time constraints of visual sensors in/for ADAS.. Examples of ADAS solution of relevance for visual sensors are: the *Lane Departure Warning* (LDW), *Adaptive Cruise Control* (ACC), Emergency *Brake Assistant* (EBA) and *Blind Spot Detection* (BSD), etc. The different ADAS solutions should be able to re-use the same components/modules for the image processing. Another weakness of classical algorithms they do not enable an easy re-use of functional components. Therefore we do need and are looking for a special architecture that is reconfigurable by software; such a concept will enable an easy re-use of the same platform for different functionalities and algorithms.

**Research question 2: What are the major limitations of traditional high performance computing approaches destined to real-time image processing?**

The main limitations of traditional computing schemes are due to the sequential processing on Von Neumann architecture concepts. Further limitations are the following:

- They are very slow especially with regard to a processing in real-time for high definition quality images
- To reach a high performance one needs a very costly system
- They need too much resources especially while compared to the capacity of embedded systems
- Too high power consumption at high frame rates

- I Inflexibility in design and system modification (evolutive maintenance)

Therefore, we can formulate two different classes of limitations:

a) Limitations at the level of software and algorithms for manipulating pixels and extracting meaningful data
b) Limitations at the level of hardware especially with regard to the flexibility in design and re-configurability

We are looking for a specific design and appropriate architecture(s) to cover both of these limitations. *Cellular Neural Networks* (CNN) do offer parallel processing paradigm that is robust for image processing. It is a highly parallel architecture with local connectivity between cells [18, 19]. This makes it a very interesting platform for an implementation on hardware. In contrast to sequential computing architectures that are highly dependent on bus bandwidth and width, amount of memory, cache memory size, buffering, processor clock rate, and CPU [20, 21], CNN is highly scalable and flexible in design.

**Research question 3: What is the huge potential of neurocomputing involving either traditional NN or CNN for high-speed and flexible image processing for ADAS? Are there some limitations; how can these be addressed?**

Artificial neural networks (ANN) with their remarkable potential to derive meaningful data from complicated data and information is getting more popular in the field of image processing [22, 23]. Detecting complex patterns, classification and prediction are only few examples showing the potential of these networks and systems. The main advantages of using ANN in ADAS systems are flexibility, robustness, adaptivity in learning, real time operation after the learning phase and fault tolerance. In the case of machine vision conventional concepts are not robust enough and depending on the complexity of the task we cannot always easily formulate a mathematical definition of the problem to be solved.

ANN can process information in a similarly to the human brain. This means that we do not need any pre-defined model for either solving problems or extracting meaningful data. The network is composed of many connected nodes; after training they are capable of solving the specific problems [22, 24]. CNNs are a special type of ANN consisting of a grid of cells which are connected to eachother locally. The local connectivity makes CNNs more suitable for hardware implementation (while compared to other ANN which require a global connectivity) [25]. The main advantage of CNN is that by changing two templates matrices one can change the functionality of the CNN processor without any hardware reconfiguration. CNN cells do work in parallel and therefore ensure an ultrafast manipulation of pixels [26].

For processing information based on neuro-computing we have to consider parallel computation, learning method and adaptation [27]. Depending on the specific task, we can define a learning rule for training the network [28]. This method can play an important role for solving complex and time consuming problems in machine vision. Pattern recognition, optimization, classification and image enhancements are important tasks in ADAS concept for which we can use neuro-computing techniques [28]. In ADAS concepts, the key issues are real-time processing and robustness. Neuro-computing can provide very stable, accurate and ultra fast solutions for various problems. In most cases, the training phase is slow and time consuming but at the end, that is after the training phase, operating artificial neural networks it is very fast [24].

**Research question 4: What are the major template calculation schemes of relevance for CNN based image processing? How can these calculations be performed in a real-time high performance computing context?**

Cellular neural networks technology provides a very powerful analog computing architecture for a variety of array computation and image processing. From a theoretical point of view the CNN concept offers the capability of modeling various image processing filters and operators on a CNN processors' based "Universal Machine". A CNN processors array used in image processing has a feedback template, a feed-forward template and a bias. These three templates are matrices that can be used to reconfigure the CNN processors system without any hardware changes. The most challenging issue is however to find the appropriate and optimized templates for each application (filter, operator, etc.). Generally, there are three ways to calculate both feed-forward and feedback templates. One approach consists in the direct mapping of the mathematical model expressing the required processing to CNN templates; the mathematical models are often in the form of either *Ordinary Differential Equations* (ODEs) or *Partial Differential Equations* (PDE). The other approach consists in using heuristic search methods (i.e., genetic algorithm, iterative annealing, and etc). Most image processing operators are working around a central pixel such as calculating intensity gradient, finding edging or performing median or Gaussian on a pixel by involving the neighboring pixels in the processing of a central pixel. For each image processing operator there is a mathematical model and an approximation [29]. Using the central difference method we can approximate the original mathematical method and apply it on a digital image by using a convolution function. This enabled by the main characteristic of the central difference method, which is that we can easily clone/realize it by a 2D convolution operator. The forward template is a simple convolution that performs only one time in CNN. Therefore, a mathematical model can be used to extract the related forward templates of CNN. In case of multiple iterations we can also use the feedback templates in CNN. This template should perform on the output of all neighbor cells which are around the center cell. After the transient phase, all the values will converge to a stable level. Another template calculating approach is using heuristic search algorithm such as genetic algorithm and iterative annealing. The genetic algorithm does use a fitness function for evaluating the quality of partial results and minimizing the error gradient function. During the learning phase the partial training results of CNN will converge to a minimum error then one can store both feedback and feed-forward templates.

Template calculation is a time consuming process. Depending on a given problem we have to calculate the appropriate templates. Using genetic algorithms is very time consuming; therefore we have to calculate different templates for different scenarios and keep it them in a template bank. There are many static templates that we can pre-calculate such as image noise removing, laplacian operator/find edges, corner detection, skeleton of regions, morphological operators, etc. Another way is using direct mathematical mapping with dynamic parameters. This form of CNN template determination can also perform in real time. Image enhancement, local thresholding, active contours,   etc are some form of ‘dynamic mathematical mapping’ based templates.

**Research question 5: How far can the advantages of "analog computing" be used/gained through an emulation of analog computing on digital hardware platforms like FPGA?**

The main advantage of analog computing is that we can “nearly” simultaneously get the result of complex mathematical differential equations. All signals are generated in parallel and in real-time and the electronic components do compute simultaneously [30-32]. In the traditional analog computing method one does face a scaling problem for the dynamic range of the computing process. The dynamic range of all components is limited and one additionally faces issues related to noise and high voltage. Therefore, is not possible to compute any dynamic range and one therefore has to always rescale the ranges appropriately. Another disadvantage of classical analog computing is that due to the fact that the solutions appear in real-time one cannot easily record them for further analysis purposes.

We can cover/overcome all of these problems by using the advantages of digital systems like FPGA. Traditional analog computers are using operational amplifier based circuits to model “addition”,” subtraction”, “multiplication” and “integration” [32, 33]. These functions are the basis for analog computing. Therefore, since we can realize all of these functions by a digital circuit an emulation of “analog computing” on FPGA appears possible. We do use the functional block “Digital Differential Analyzer (DDA)” to compute the integral of a function over time. To get more speedup we can use fixed-point arithmetic calculation instead of a floating-point one. Therefore, we must thoroughly check the ranges of values and levels of accuracy for assigning reserve bits for both “integer” and “fraction” parts. By emulating analog computing on digital hardware platform like FPGA we become capable of getting the solution in real-time without any limitation (contrarily to traditional analog computing) related to scaling voltages. The reason is that instead of voltages we are dealing with registers and fixed-point operators on data. A further advantage is that we do not need any additional converter for/before storing data into the memory. We have full control on the clock rate and this also represents a great advantage of using FPGA and a digital architecture. Concerning setup time, debugging effort and configurability the digital emulation of “analog computing” is clearly very superior to the traditional “real electronics components” based analog computing systems.

**Research question 6: How far can an efficient implementation of CNN on either FPGA or GPU be designed and implemented?**

CNN is a complex design in terms of implementation and performance on traditional architectures of the von Neumann type (such as CPU and sequential processors). Therefore, we are looking for an appropriate architecture compatible with the parallel nature of CNN. Traditional ANNs display some form of global connectivity; this results in a difficult implementation in digital hardware. In contrast CNN has local connectivity of its cells; this results in a great potential for implementation of this architecture on either FPGA or GPU. FPGA macro cells and logical components can operate in a highly parallel manner. Further, due to a very flexible routing between them we can implement any complex digital circuit scheme or model on FPGA. To get more advantages from using FPGA one should use high level behavioral modeling languages such as VHDL, Verilog or SystemC. FPGA has a local and embedded memory; this is very important for storing the CNN states. Otherwise the memory access processes between the FPGA and and an external memory could be very time consuming and constitute a big bottleneck. Today most of FPGAs chips on the market do contain an internal dedicated standard CPU that has access to the hardware and to the logical field of the FPGA through the standard bus controller [34, 35]. There are many high level compilers based on ANSI-C standard for coding and debugging. This technology increases the system performance through integration of hardware and software. Therefore, the loading of initial states of the CNN cells, of the templates values, and setting 'time scales' and other parameters can be done/performed easily by the embedded CPU. The resources of FPGA are not endless and we have to consider this issue during designing the CNN architecture.

Another interesting platform for CNN is GPU, which is getting more popular every day. The highly parallel structure of GPU's makes them more efficient for image processing and for processing large blocks of data. Due to the high memory bandwidth between CPU and GPU, the integration of GPU and CPU through standard protocols/API and running multi kernels scripts on GPU, GPU appears today to be a very interesting technology for an efficient and cost-effective implementation of CNN. Since 2003, GPU technology has been experiencing a fast growth. Further, in terms of design flexibility we can now implement very complex models and systems by using flexible and robust high-level tools for appropriate software development. In OpenCL there is a standard API for communicating between CPU and GPU. This API does provide some essential commands for allocating memory, transferring memory content between host/CPUs RAM and device/GPUs RAM. There are some commands also for compiling GPU code which is a kernel file, and executing them. In OpenCL running multiple kernels is possible, and there is a direct and ultrafast channel for transferring memory and data between different kernels. These features make GPUs very flexible and reliable for designing complex architecture and models.

**Research question 7: How far can CNN be used/involved in an evolutionary computing/control context example?**

CNN has a great potential for signal processing tasks and it can generate very complex nonlinear waves and oscillation patterns at the output of CNN cells. Controlling both the kinematic and the inverse-kinematic of complex robots with high *Degree of Freedom* (DOF) is a very complex scenario whereby classical solutions fail to solve it easily. Complex ADAS solutions may also be seen as systems with a high degree of freedom, for example CACC (cooperative adaptive cruise control). Therefore, the evolution of CNN templates to generate the optimum wave for both the driving motor(s) and the robot actuators is a particularly interesting idea for research. In nature, organisms' systems are evolving as explained by the theory of natural selection. Overall, the art is to define a fitness function for evaluating the performance of the actual robot locomotion. Hence, by evolving the CNN template using a genetic algorithm and a fitness function, we can generate very complex waves for optimally controlling the robot hinges without directly involving the robot's kinematic equation in controller design. A two dimensional CNN should be sufficient for evolving a spatial wave over time for controlling the robot's actuators and hinges.

A fitness function is a particular type of objective function that quantifies the optimality of a solution [36]. Input data for the fitness function are based on measurements from robot parts' orientations, locations and displacements. In the fitness function we do not define any behavioral locomotion exactly. On the other hand, we define a function that satisfies the target or destination. With this method involving genetic algorithms an optimum template ensures that the robot can move or act according to our desires. The most important point in this learning method is that we do not need to (explicitly) predefine any robot's kinematics for movement in the fitness function.

## 1.3 Summary of the key contributions of the thesis

There are many different ADAS technologies offering various functionalities and levels of safety. According to literatures, cameras are playing a very important role in modern ADAS technology. Due to the large amount of visual data and the complexity of image processing algorithms as well as of algorithms for extracting meaningful data (i.e. image enhancement, noise removing, segmentation, classification, etc.) traditional methods fail to solve these challenges and are difficult to implement in hardware. This thesis mainly targets the development of concepts, hardware architecture(s) and related software concepts for ultra-fast image processing in ADAS. From a scientific point of view the goal is designing a fast and robust image processing system. CNN appears to be one of the best models that offers sufficient potential to solve these challenges. Most of the advanced filters and operators in image processing are mathematically based on PDEs. It is therefore motivating that CNN is very appropriate for solving various types of PDE's.

### 1.3.1 Scientific significance of the thesis

The core objective in this thesis is the development of an ultra fast and robust computer vision system based on cellular neural networks for ADAS. From a scientific perspective the aim is to develop a platform and architecture for performing very complex image processing filters and algorithms in dynamic environmental and different lighting conditions. The proposed system is a CNN platform which is based on an emulation of analog computing and can work in varying lighting conditions. This system opens the doors for performing different PDE based image processing models on videos. The major scientific contribution of this thesis is lying in the modeling of a CNN based image processing architecture through an emulation of the traditional "analog computing" on digital platforms (FPGA and GPU) to make a very robust and adaptive image processing system. Flexibility of design is another main major factor for developers that have been considered in this thesis. One can map most of the classical image processing models on this system that then does operates in a parallel mode, and thereby ensuring high speed, accuracy and robustness. This thesis did also focus on a template calculation concept involving genetic algorithms. We have shown how to find appropriate templates for image enhancement, obstacle detection and for controlling joints of an unstructured robot in different scenarios.

### 1.3.2 Practical significance of the thesis

Stability and real time processing are key issues in embedded systems and smart sensors. The concept of this thesis has the high potential of being easily implementable on a chip as a smart visual sensor. Visual computing on CNN can fulfill the real-time requirements of ADAS concepts. Processing visual data at sensor level does reduce the latency and the memory bottleneck, and at the same time it does increase the speed of processing while increasing the system's efficiency. The system does target the ADAS concepts in terms of real time processing, size, flexibility in design and configurability. The encapsulation of different ADAS concepts on a chip reduces the overall system cost. Integrating a system on chip with other peripheral is easier than designing a complex system which is not uniform. This design is particularly useful for developers that want to build different realtime ADAS concepts on the same platform.

## 1.4 List of publications in the frame of this thesis

The following eighteen (18) publications have been made in the frame of the research work of this thesis; seven (7) of them did appear in international scientific journals whereby the remaining did appear in the proceedings of international conferences:

1. Fasih A., Schwarzlmüller C., Kyamakya K., Al Machot F, "Video Enhancement for ADAS Systems based on FPGA and CNN Platform," *International Journal of Signal and Image Processing*, online: HyperSciences_Publisher, Vol 1, no. 3, pp. 169-176, May 2010.
2. M. R. Ghahrodi , A. Fasih, "A Hybrid Method in Driver and Multisensor Data Fusion, Using a Fuzzy Logic Supervisor for Vehicle Intelligence," *International Conference on Sensor Technologies and Applications SENSORCOMM-IEEE Computer Society*, 2007, pp. 393-398.

3. Fasih A., Chedjou J.C., Schwarzlmüller C., Kyamakya K, "New Computational Modelling for Solving Higher Order ODE Based On FPGA," *ISAST Transactions on Electronics and Signal Processing,* vol. 4, no. 1, pp. 58-61, October 2010.
4. Fasih A., Schwarzlmüller C., Chedjou J.C., Kyamakya K, "Framework for FPGA Based Real-time Machine Vision Direct Convolution Versus CNN," *ISAST Transactions on Electronics and Signal Processing,* vol. 4, no. 1, pp. 1-5, October 2010.
5. Schwarzlmüller C., Fasih A., Latif M.A., Kyamakya K, "Adaptive Contrast Enhancement Involving CNN-based Processing for Foggy Weather," *ISAST Transactions on Electronics and Signal Processing,* vol. 4, no. 1, pp. 24-35, October 2010.
6. Fasih A., Chedjou J.C., Kyamakya K, "Cellular Neural Network Trainer and Template Optimization for Advanced Robot Locomotion, Based on Genetic Algorithm," International *Journal of Intelligent Systems Technologies and Applications (IJISTA),* Geneva: Inderscience Enterprises Limited vol. 8, no. 1-4, pp. 36-45, 2010.
7. Fasih A., Chedjou C. J., Kyamakya K. "Cellular Neural Networks-Based Genetic Algorithm for Optimizing the Behavior of an Unstructured Robot," *International Journal of Computational Intelligence Systems (IJCIS),* Paris: Atlantis Press, 2009, pp. 124-133.
8. Fasih A., Schwarzlmüller C., Chedjou J.C., Kyamakya K, "An Ultra-fast and Adaptive Framework for FPGA-Based Real-Time Machine Vision for Advanced Driver Assistance Systems: a CNN-Based Processing Architecture," *Proceedings of Mallorca Workshop 2010 - Autonomous Systems.* Aachen: Shaker Verlag GmbH, vol. 6, pp. 28-34, October 2010.
9. Fasih A., Khan U., Chedjou J.C., Kyamakya K, "Efficient Control of the dynamic system 'Coordinated Urban Traffic Lights System' using a novel Reinforcement learning Concept," *ISTET International Symposium on Theoretical Electrical Engineering.* vol. 24, pp.195-198, June 2009.
10. Schwarzlmüller C., Fasih A., Kyamakya K, "Enhancement of Rainy Weather Degraded Images," *Proceedings of Mallorca Workshop 2010 - Autonomous Systems,* October 2010, pp.20-27.
11. Fasih A., Trong T. D., Chedjou J. C., Kyamakya K, "New Computational Modeling for Solving Higher Order ODE based on FPGA," *Proceedings of INDS´09. The Second International Workshop on nonlinear Dynamics and Synchronization,* vol. 2, pp. 49-53, June 2009.
12. Fasih A., Kyamakya K. , Chedjou J. C., Umair A. K, "Benchmarking of the Traditional Genetic Algorithm Method with a Novell Approach and a further novel Scheme, the 2-Point Crossover (F-Crossover)" *ISTET International Symposium on Theoretical Electrical Engineerin,* June 2009, pp. 195-198.
13. Schwarzlmüller C., Al Machot F., Fasih A., Kyamakya K, "A Novel Support Vector Machine Classification Approach Involving CNN for Raindrop Detection," *ISAST Transactions on Computers and Intelligent Systems,* vol. 2, pp. 52-65, November 2010.
14. Tuan D. T., Chedjou J. C., Fasih A., Kyamakya K, "A Novel Method for Computing the Minimum Spanning Tree and Solution Based on Cellular Neural Networks Implemented on Digital Platforms," *The Second International Workshop on nonlinear Dynamcis and Synchronization, INDS'09.* Vol. 2, pp 23-29, June 2009.
15. Khan U., Fasih A. Kyamakya K., Chedjou J.C. "Genetic Algorithm Based Template Optimization for a Vision System Used for Obstacle Detection," *ISTET International Symposium on Theoretical Electrical Engineering*, June 2009, pp. 164-168.
16. Schwarzlmüller C., Fasih A., Kyamakya K. "Partial Differential Equation based Image Inpainting by using Cellular Neural Networks," It will appear in ISAST Transactions on Computers and Intelligent Systems 2011.
17. Fasih A., Chedjou C. J., Kyamakya K., "Cellular Neural Network Trainer and Template Optimization for Advanced Robot Locomotion, Based on Genetic Algorithm," *15th International Conference on Mechatronics and Machine Vision in Practice,* December 2008, pp. 317-322.
18. Fasih A., Sasanca P., Kyamakya K, "CNN based High Performance Computing Platform for Real Time Image Processing on GPU," It will appear in 3rd International Workshop on nonlinear Dynamics and Synchronization 2011 – INDS'11.

# Chapter 2

## 2. Requirements of ADAS concerning real-time computing for the image processing based Sensors

### 2.1 Context and Motivation

In this chapter the focus lies on the following research question: *"What are the hard requirements of ADAS concerning real-time image processing and design flexibility? How far do traditional approaches fail to satisfy these requirements?"*

High-end vehicles are equipped with different technologies for driver assistance in order to ensure more safety [7, 9]. In ADAS there are different sensors with different sampling frequencies. One layer above the raw data layer there is an additional layer in which data fusion, pattern recognition and classification are performed [9]. These last mentioned tasks are computationally heavy; therefore this second layer is the most critical part of ADAS technology in terms of processing time and speed. The third layer is the application layer. In Table 1 selected examples of ADAS solutions are presented whereby the respective computational effort is roughly classified.

Table-1: Different types of ADAS solutions and processing power/effort needed

(Low: <10 GFLOPS; Medium: 10~50 GFLOPS; High: 50~150 GFLOPS; Very High: >150 GFLOPS)

| ADAS solution | Sensor types involved + Infrastructure | Processing power/effor needed –to ensure real-time processing |
|---|---|---|
| Night vision | Thermal or IR camera | Medium |
| Lane departure warning | HD camera | Medium |
| Near field collision warning | Radar-24 GHZ | High |
| Curve and speed limit info | Gyroscope and accelerometer | Low |
| Lane keeping assistance | HD camera | Medium~high |
| Adaptice cruise control | Lidar | High |
| Automatic parking | HD camera + ultrasonic sensors | Medium |
| Pre-crash collision system | Lidar | High |
| Obstacle warning | Stereo camera | Very high |
| Fatigue detection | HD camera + IR camera | Very high |
| Autonomous driving | Multiple sensors and multi sensor fusion | Very high |
| Traffic Sign Recognition | HD Camera | High~very high |

In many ADAS cases the driver will be warned if a potentially dangerous situation is detected. But in some other cases, depending on the type of assistance, the ADAS system can take over the control (partially or fully) over the car by sending appropriate commands to the actuators. In such cases of control takeover the signal/video processing must be ultrafast to ensure very hard real-time requirements.

Two main issues while designing an ADAS system are the processing speed and the robustness. A robust system should work in dynamic environmental condition, dynamic illumination and lighting [7]. Some distortion like sun in background and shadows can disturb the vision system and provide wrong information. To ensure that system is working in different condition we need a highly adaptable framework with dynamic coefficient.

Depending on the type of video-based ADAS applications we do need different appropriate software/hardware architectures as well as different combinations of image processing filters. Some filters are very complex in term of processing computational effort. A good example of a complex filters is the stereo vision for depth estimation and collision avoidance. For such complex processing tasks one does need a specific hardware and software architectures that amongst others enable and support parallel processing and tasks concurrency. .

Traditionally one has been using sequential processing architectures, time sharing and multi-threading algorithms [9, 37, 38]. The main weakness of this traditional way of processing with respect to performance and speed is that processing time is too slow for a real-time ADAS application [9]. Therefore, the traditional approach has only limited ways to reach a certain speeding-up: to extend the hardware, and use more powerful processors. To ensure a real-time processing of high quality images the system should able to complete the processing of a frame within less than therequired time for capturing a frame. Consequently, for a 60 FPS we do have a maximum 15 milliseconds for all processing. And if we have 6~10 different sequential high definition (HD) image preprocessing modules/function whereby each one takes around 5ms on traditional (embedded) processing platforms/architecture, it would take, overall, approximately 30 ms to50 ms. By those processing times one does clearly fail to satisfy the real-time requirement/deadline of 15 milliseconds.

## 2.2 State of the art of real-time ADAS platforms

A huge on-going research and several projects are and have been conducted in various aspects of ADAS technology [39-42]. The sensory system within ADAS does provide all necessary data and information which is needed to estimate the traffic situation and state of the vehicle in real-time [43]. For processing images captured by HD cameras [39] people have been using FPGA and DSP platforms with processing frames with 20~100 FPS. Examples of systems where this can be observed are [14, 44, 45], lane markings detection and obstacle detection. One has switched to hardware implementation (i.e. on FPGA or DSP) in order to reduce the CPU load; see reference [39]. In reference [43], Mr Wada has been using a normal PC for the development of advanced parking assistance systems. Further, in reference [46], authors Vitabile *et al* do propose a lane keeping system based on FPGA that can process up to 39 frames per second; this safety system has been capable of identifying a dangerous situation and react in real-time. A main drawback of all of traditional ADAS systems is that they are using different platforms for different tasks. Thereby a combination of all different tasks does need/consume too much hardware resources [12, 47].

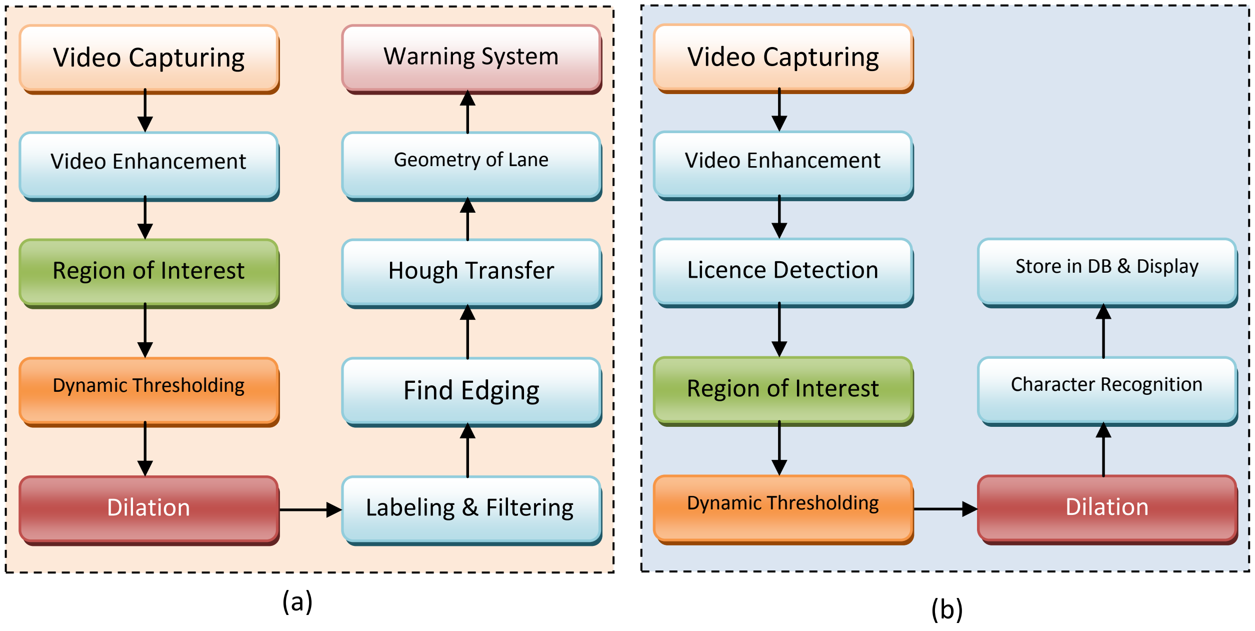


**Figure 2-1: Comparison of two different ADAS system. (a) Lane departure warning, (b) Licence plate recognition**

Figure 2-1 shows the comparison of two different types of ADAS concept. Both systems are using the same modules of video capturing, region of interest, dynamic thresholding and dilation. Therefore, without a dynamic architecture we should have to unnecessarily reserve resources for each of the systems.

## 2.3 Contribution for real-time ADAS

In this thesis, a new architecture for both hardware and software is proposed. In the proposed platform concept we can perform the ADAS algorithms related to the visual sensor processing within the fixed real-time constraint/deadline of 15 ms. Chapter 8 proposed a hardware and software model for implementing a robust ADAS system based on GPU. As described with more details in another chapter of this thesis chapter 7; a combination of FPGA and the “Cellular Neural Networks” paradigm does offer a powerful and robust platform concept that does ensure a real-time image processing for various ADAS solutions. FPGA is a field reconfigurable hardware that is generally specified by a hardware description languages such as VHDL, Verilog or SystemC [48]. The integration of hardware (hard core) and software (soft core) within FPGA does represent a great advantage of FPGA technology as it ensure a great flexibility and robustness. Due to the huge amount of logic cells, basic logical operators and a routing system that is dedicated in FPGA like a “switch matrix”, one is able to implement very complex functionalities for processing data through FPGA.

Developers are able to realize the integration of FPGA and external I/O (i.e. input/output) peripherals. Hence one can capture videos or data from the outside of the FPGA. In this thesis, the integration of a HD-camera with the FPGA through a small daughter-board is proposed. The FPGA logic cells and the internal CPU will have an access to the high-level data through this daughter board and can load frames of data for processing. It is known that FPGAs are perfectly suitable for performing parallel tasks for signal and image processing. One of the promising paradigms for ultrafast image processing is based on *Cellular Neural Network* (CNN) [26]. Many forms/architectures/versions of CNN do exist, that are discrete-time CNN (DT-CNN), non-linear CNN (NL-CNN), etc. But the DT-CNN appears to be more appropriate for image processing because it does require less hardware resources [25]. Each CNN cell consists of some basic mathematical operator such as addition, subtraction, integration module, and a sigmoid function. Beside the integration of CNN and FPGA as a target platform for ultrafast ADAS related image processing we do also propose and have developed an alternative concept that does integrate CNN and another also actually promising technology, the GPU. Details of this additional proposal (that combines CNN and GPU) are presented in two other chapters of this thesis; see chapters 7 and 8. The GPU technology does also offer a series of advantages ranging from design flexibility, availability, costs, the possibility of an easy integration with other framework such as *Open Computing Language* (OpenCL), and much more. The company AMD has released an embedded GPU to provide high performance in mobile and embedded systems. AMD Radeon E6760 is an embedded discrete graphics processor that supports OpenCL and it has a good performance for parallel processing. Hence, having a high-performance system in the scale of embedded system is possible.

# Chapter 3

## 3. Major limitations of traditional high performance computing concepts

In this chapter the focus lies on the following research question*: "What are the major limitations of traditional high performance computing approaches destined to real-time image processing?"*

### 3.1 Motivation and general context

The main goal for the use of ADAS solutions in cars is for increasing road safety [46]. ADAS do enable a better response to dangerous situations that may occur on the road in a very robust and fast manner [7, 46]. Traditional architectures for ADAS have been based on sequential processing on mainly *von Neumann* types of architecture. They are consequently not fast and flexible enough for processing huge amount of visual data [49, 50] within hard real-time constraints.

In fact, processing images on a *Central Processing Unit* (CPU) has many limitations. A computing problem (in this case, an image processing one) can generally be broken down into a discrete series of instructions where pixels will be manipulated individually. There are two major limitations of classical sequential algorithms performing sequentially on CPU hardware [26, 49].

First of all, in contrast to parallel processing systems, sequential techniques/algorithms are very slow for high definition quality images. Therefore, while using these techniques a real-time image processing can only be reached through a costly system having a relatively very high performance computing. Thereby the main drawback will be the size of system as well as the power consumption; both will result in increasing the total cost of the solution. One further limitation is the strong inflexibility with regard to design and modification potential of the system for different types of algorithms/processings. Thus one can formulate two different classes of limitations that are:  a) the one related to the software architecture and to the algorithms for manipulating pixels and afterwards extracting meaningful data; and b) hardware limitations and inflexibility of design. A much better solution for image processing is the use of a parallel processing architecture. *Multiple Instructions - Multiple Data* streams (MIMD) is the most common architecture for parallel processing; most modern computers fall into this category [51, 52]. In this model of processing every processing unit may access to different memory and data streams. For increasing the performance for reaching a sufficient speedup, dedicated hardware for each algorithm has been suggested.  Hence, designers implement each application on different platform, and this redundancy in hardware increases the price and complexity of the system [53, 54].

A direct mapping of algorithms on the hardware is often viewed as the best way of processing [53, 55, 56]. The only issue that should be considered in this case is the low flexibility of the system concerning design time. Therefore, the only drawbacks of this approach are: a) the complexity of mapping many image processing operators/functions onto the hardware, and b) the limitation of hardware resources.

Hence, we are looking for a reconfigurable model/concept/architecture that can change (or be reconfigured) into different functions and thereby significantly saving hardware resources. Another important factor of this model is parallel processing of pixels.

## 3.2 State of the art in traditional processing method

As the processing potential of processor is increasing, the computation requirements are also getting higher over time [57]. No matter how fast processors become the  technology of parallel processing is growing up to make them even faster as before [58]. Today, computer are highly complex and they are made up by complex components and architectures [59]. Most of them are using some kind of low level parallel computing at the level of instructions. They can load different instructions and perform different operations at the same time. This is generally called “instruction level parallelism” [58].

Recently computer architects have started to direct their attention onto other techniques for improving both the processing time and hardware performance. *Shared-Memory Parallel* (SMP) computing is very popular technique for speeding up the processing time [60-62]. Isaac G. *et al* do propose in reference [63] an heterogeneous computing concept which is a combination of a general purpose CPUs with an accelerator to improve the execution efficiency. This model is based on shared memory parallel technique.

J. Batlle *et al* proposed in reference [49] a dedicated parallel architecture based on FPGA and DSP for real-time image processing. This system has been designed to deal with pipeline procedures and operators. They have broken the image processing algorithm into three major steps: preprocessing, intermediate processing and, at the end, a post-processing. All low level functions are performed at the preprocessing level. In the intermediate level of processing we have some algorithms like segmentation, motion estimation and features extraction. In the post-processing level we involve statistical analysis and artificial intelligence [49].

D. Demigny *et al* proposed in reference [64] a high speed reconfigurable FPGA system for processing images in real-time. They have considered different architectures and models of processing architectures such as *Multiple Instruction - Single Data* (MISD) and *Single Instruction - Multiple Data* (SIMD). The SIMD architecture is very interesting for image processing because we have the same instruction for multiple data streams. The main drawbacks of all mentioned architectures are however: a) that there is no homogenous design that can cover different image processing algorithms, and b) that for any new design one has to reconfigure the hardware. It may be possible to design a real-time and robust image processing architecture for a specific task, but it not easy at all to reconfigure it for different tasks and procedures.

## 3.3 Contribution of Ideal ADAS architecture

Our main goal is developing a relatively universal processing architecture for image processing which should be running on hardware and work in a highly parallel manner. Hence, we have considered two different platforms based on FPGA and GPU. Developing highly parallel systems needs a proper platform which should be flexible and robust. In traditional parallel processing models many processors have access to a shared memory. *Message Passing Interface* (MPI) and directive-based interface are two important approaches in shared memory techniques. Figure 3-1 does illustrate the essentials of this model.

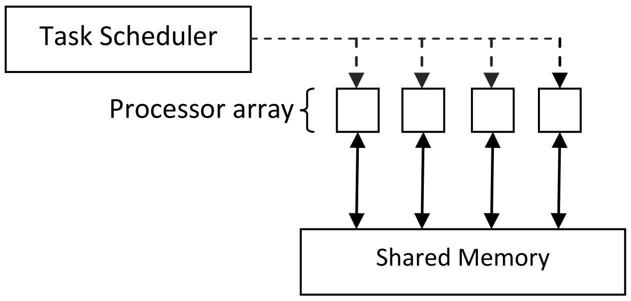


**Figure 3-1: Shared memory's parallel processing model**

The main advantage of this processing model/architecture is that one can calculate very complex algorithms on the stream of data with a shared memory. A major drawback of this model is however the latency and bottleneck in memory as well as the complexity of the task scheduling. Nevertheless, by coupling a simple distributed memory as the state variable of a single element and a simple processing unit in form of a 2-dimensional grid, to a nonlinear operation that is coupled also to the neighbors through local connections one can overcome too many problems of classical parallel image processing. Figure 3-2 has shown this type of architecture.

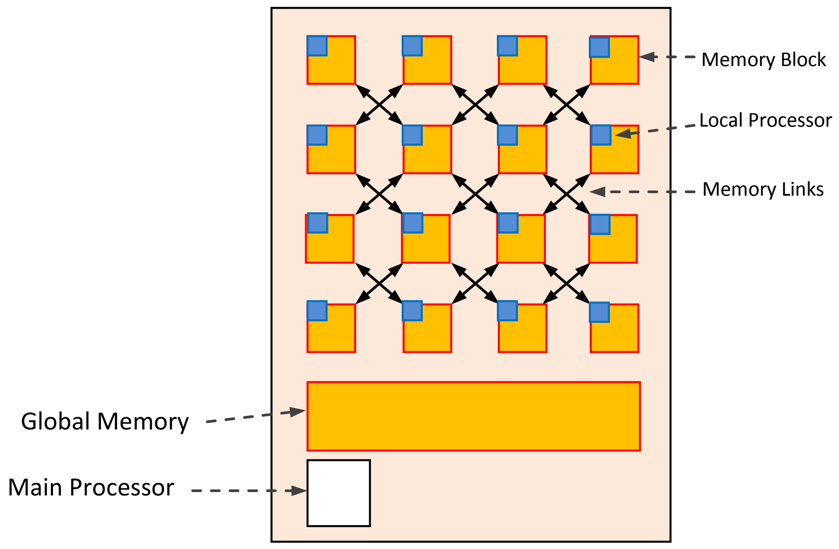


**Figure 3-2: General idea for distributed processing**

Worth a mentioning is that CNN is providing a similar model for processing data and images. By changing templates we can define a new model for different operations/functions. Coupling more than one layer of CNN can enable the designer to model very complex image processing operators [65, 66]. Due to the related robustness modeling image processing by PDE's is getting more popular [67, 68]. Some equations comes from minimizing energy function and some others are designed using geometrical arguments like mean curvature motion [69]. There are many application based on PDE such as inpainting for recovering corrupted regions in image [70], image segmentation [71], noise reduction edge preservation [72]. All of these examples and similar techniques are essential for video processing in ADAS. The procedure of solving PDEs in CNN is by transforming a PDE to set of ODEs. After transforming a continuous spatial PDE to an array of discrete interactive systems which are ODEs, we can map it on CNN cells. Because CNN is natural and flexible paradigm for modeling a simple locally interconnected dynamical system which are grid base.

The CNN architecture is very close to PDE's and even a direct mapping of PDE's into a CNN processor matrix is possible [73]. In the case of linear PDEs we can map each independent variable with related partial derivative of that to a CNN layer. If we have more than one independent variable we have to couple many CNN layer together to provide the solution. By defining the right templates one can model the behavior of PDEs through a CNN processor system that will generate the solution. T. Roska *et al* have shown in [73] a way of how to simulate a space invariant nonlinear PDE by CNN. They have described the dynamics of three different systems (i.e. 2D heat equation, Burgers' equation and Navier-Stokes equation) by sets of equations. Mapping a two-dimensional heat equation which is modeled by the Laplace operator has been solved in [73]. Equation 3-1 is showing this heat model.

(3-1)

$$\frac{\partial u(x,y,t)}{\partial t} = c\nabla^2 \mathrm{u(x,y,t)}$$

In this equation, $\nabla^2$ is the Laplace operator and it is applied to the intensity which is $u(x,y,t)$. After spatial discretization of this equation, the PDE is transformed into a system of ODEs. If we discrete the equation in space by steps of $\Delta x = \Delta y = h$, then we can map the $u(x,y,t)$ on a CNN layer. Before that we need a numerical solution of the equation based on Taylor-series. Equation 3-2 has shown this approximation.

(3-2)

$$\frac{\partial^2 u}{\partial x^2} \sim \frac{1}{h^2}\left[u(x+h,y) - u(x,y) - (u(x,y) - u(x-h,y))\right]$$

$$= \frac{1}{h^2}\left[u_{i+1,j} - 2u_{i,j} + u_{i-1,j}\right]$$

Using this approximation, it is easy to map this equation onto the CNN template.

$$A = \begin{bmatrix} 0 & \frac{1}{h^2} & 0 \\ \frac{1}{h^2} & \frac{-4}{h^2} & \frac{1}{h^2} \\ 0 & \frac{1}{h^2} & 0 \end{bmatrix}, B = 0, \qquad I = 0 \tag{3-3}$$

The time evolution of the CNN processors using this template will therefore give the solution of heat equation.

For image processing there is a discrete time version of CNN that is easy to implement on digital platforms like CPU and FPGA. Out first trial/exercise has consisted of two simple cells which are connected to each other for solving a 2nd order ODE. Each CNN cell has an internal integrator which fits for solving 1st order derivative equations. For solving higher order equation such as 2nd or 3rd and etc, we should couple them as a system of simple 1st order derivative equations. In chapter 6 we have shown how to solve a nonlinear Rössler equation by this technique. Later on, it is possible to model CNN by direct coupling of cells and integrators by local connections.

# Chapter 4

# 4. Potential of Neurocomputing including Cellular Neural Networks for ultrafast image processing

In this chapter the focus lies on the following research question: “*What is the huge potential of neurocomputing involving either traditional NN or CNN for high-speed and flexible image processing for ADAS? Are there some limitations; how can these be addressed?*”

## 4.1 Context and Motivation

Artificial Neural Networks (ANNs) are inspired by biological science; they mimic model and behavior of biological neurons and perceptrons. In a sense, ANNs are a kind of computing concept that are trained based on specific history and information in order to mimic the same as behavior as that of the brain neurons. They implicitly ensure some form of remembering of those historical information from the past [74].

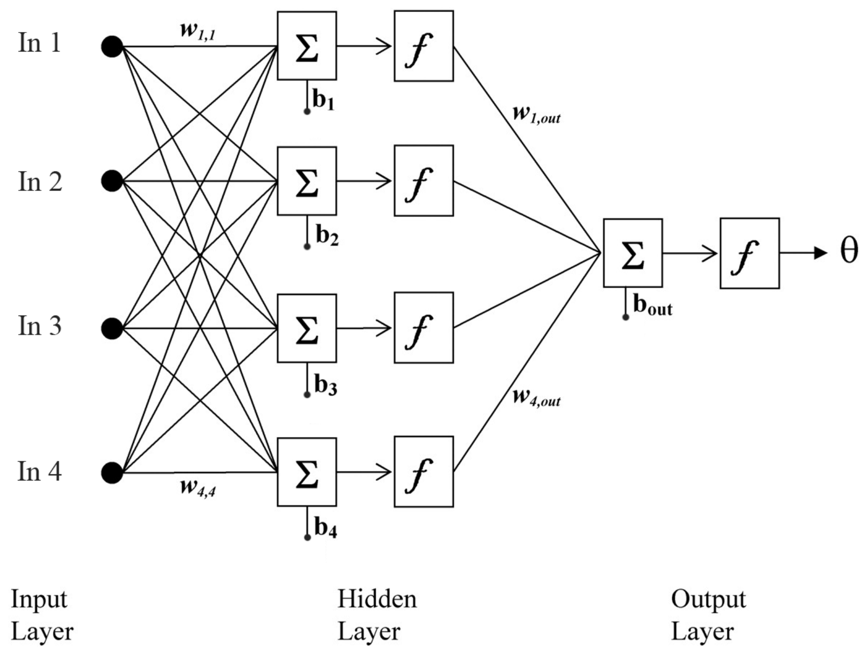


**Figure 4-1: Model of a feed-forward neural network with four inputs and one output**

Figure 4-1 shows a simple feed-forward neural network with four inputs and four cells as a hidden layer and one output. Information can flows from input layer to hidden layer and then to output layer through the connection which are generally called links or synapses. Each layer except the input layer has an activation function. Flow of information from input to output called feedforward. After training, the operation mode is feedforward mode. Means loading information in input layer and system calculate the proper output according to weight and topology of the network [75]. The most common learning method for teaching and training the MLP networks is backpropagation. It's a supervised learing method and it is derivation of delta rule method. For training we need to perform propagation phase and then updating weight. In propagation phase, we need to perform forward propagation of training pattern and get the result from output layer activation functions. After that, performing backpropagation of these output data by considering target in order to generate the delta of output and hidden neurons. For updating the weights we need to multiply output delts of each layer and input activation to calculate the gradient of the weight and then updating the direction of gradient by subtracting the ratio of synapse from the weight. This ratio is corresponding to the speed and quality of learning.

There are many applications in image processing and machine vision for which one can use this type of neural networks for processing. Traffic signs recognition could be a good example. Hereby, the main problem for traffic sign recognition is the design of an algorithm that is scale-variant and rotation-variant for recognition and classification of different traffic signs. This means system is not sensitive to the scale and angle of signs. During driving camera can see the sign from different angle of view and different scale. Therefore this is feature of ANN is important.

This type of neural network has potential to learn multi-scale and multi-variant traffic signs [76]. This means we can train ANN for different size and different traffic sign. The use of artificial neural networks with their remarkable potential to derive meaning data from complicated/complex data and information is getting more popular in the field of image processing [76]. Detecting complex pattern, classification, and prediction are only few examples illustrating the huge potential of neural networks for image processing.

The main advantages of using ANN in ADAS technology is flexibility, robustness, adaptivity in learning, real time operation after the learning phase and fault tolerance [77] [78]. It is known that many conventional algorithms for image processing and machine vision are not at all sufficiently fast in view the complexity of the tasks. It is furthermore relatively very difficult to define an appropriate mathematical model for solving those problems. ANNs however, can process the information in a similar way to the human brain. This means that a predefined model for solving the problems and extracting meaningful data is not necessary [79]. Artificial neural networks do offer many advantages. For example it requires less formal statistical training; another advantage is the ability of detecting highly complex and nonlinear relationships among independent and dependent variables. Finally, it has also the potential for multiple training algorithms [79]. Due to the massive parallelism, neural network are much faster than conventional methods.

If the modeling of a system is not trivial and thus complex and if it is hard to formulate an explicit algorithm as a solution/model one should use artificial neural networks[80]. There is a main difference between von Neumann models for computing which are based on memory/processors and artificial neural networks. In an artificial neural network we are using a parallel architecture similar the biological brains. We do not know details of the complex mechanisms within the human brain; but we do know the main principles of neurons operations either individually or globally [81]. Another main factor of artificial neural networks is scalability and adaptivity in the learning phase. The system can change its structure based on training information. We can model very complex models and relation between input and output.

Basically ANNs are appropriate for processing images [81]. However, they do have some weak points such as over-fitting during the training phase, their black-box nature and the complexity of a related hardware implementation [82].

Designing and implementing a computing system based on artificial neural networks in software for a high resolution image is very complex but can thereby even not be fast enough with regard to the real-time requirements of ADAS [80]. Therefore, we are interested in implementing it in hardware. There are two main problems faced by the hardware implementation of artificial neural networks. A first problem is the huge complexity of the network due to too many connections between cells. Another challenge is how to design a reconfigurable architecture that can  be reconfigured to execute different tasks [80]. To find a general purpose hardware platform and architecture concept capable of solving/computing different types of problems of image processing is not a trivial.

## 4.2 A survey of the related state-of- the art on image processing based on ANN.

An artificial neural network is a mathematical model or computational model of biological neural network. These are essentially a very simple mathematical model for mapping input to output. The things that make ANN attractive are learning possibility. Means during the learning phase, by giving a class of function system can find an optimum solution. We have many different type of neural network such as Feedforward *Multi-layer Perceptron Neural Network* (MLP), *Radial Basis function* (RBF), Kohonen *Self-Organizion Network* (SOM), Hopefield Network, *Bi-directional Neural Network* (BRNN), *Associate Neural Network* (ASNN) and etc that are useful for different application. The most common type of artificial neural network in the field of image processing is MLP. This model as it mentioned in Figure 4-1, has many input, hidden layer and output. Each neuron in one layer is connected the other neuron in another layer. Therefore we have fully connection between each layer to another layer. Depending to the type of data and information we have to reshape them for feeding them in our network. For example for loading an image into the network we have to reshape the 2D picture in form of vector. Figure 4-2 has shown this concept. We have to consider many issues before loading data to the network for training or in operational mode. For image processing, in many cases we need to perform a color conversion, normalization and quantization of values before feeding data into the network.

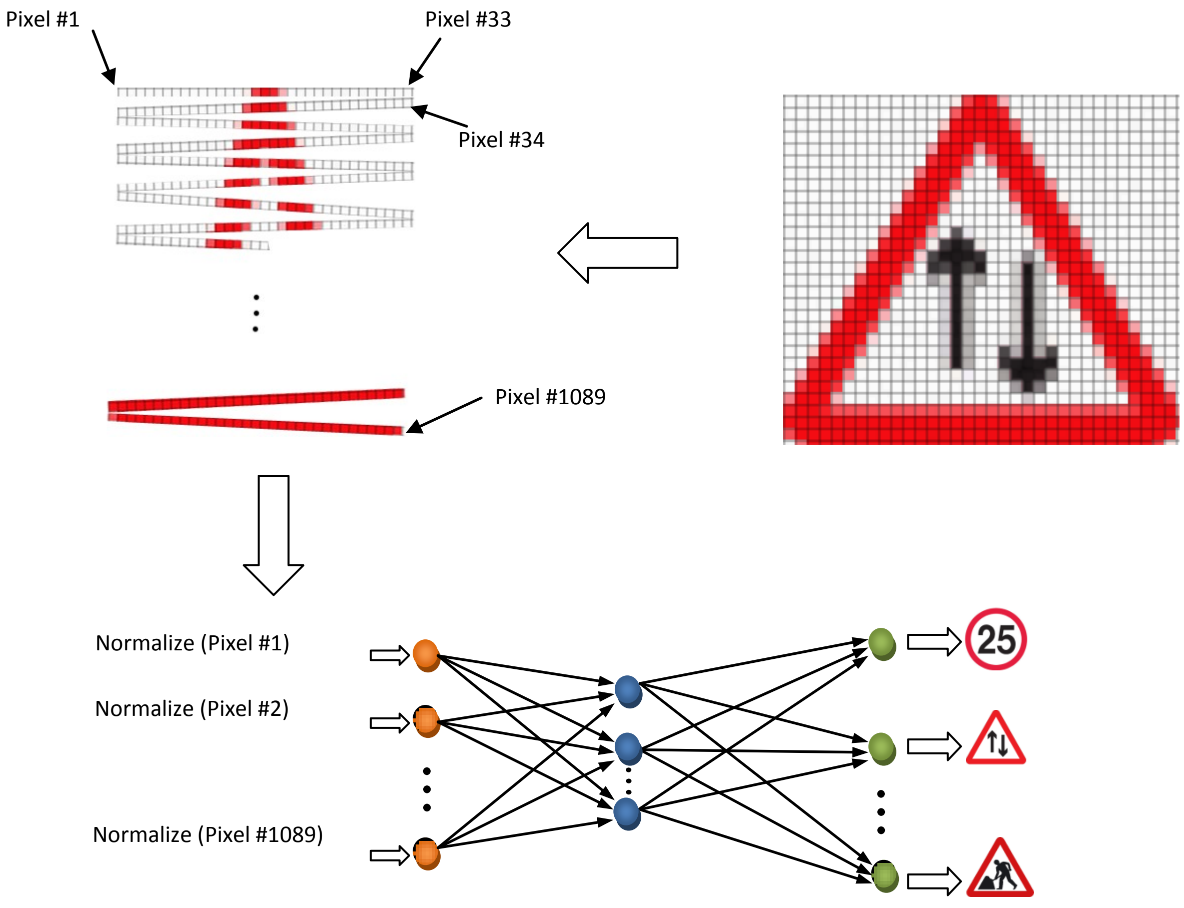


**Figure 4-2: Reshaping and normalizing 2D image data for loading in a MLP network**

Now let us look to the human brain potential; the human brain is a very complex processing unit. There are also some problems that are not easy for the brain to solve them within a proper time duration; an example is multiplication [80]. In recognition problems, the organic brain is more efficient than conventional digital computing systems at a factor of $10^8$ times [80]. This efficiency is not due to the processing speed; it is rather due to the processing paradigm. There are many problems in machine vision and image processing that we can use ANN. Feed-forward neural networks are one of the most popular ANN type for image processing especially for classification and recognition. For training such an ANN one does need a vector of solutions and input data. This is called supervised learning. There are two main types of learning for training ANN: supervised learning and unsupervised learning. Concerning supervised learning the aim is to train a network which should express a specific function [83]. One does have a set of data pairs which are combinations of inputs and targets values. More details on supervised learning can be found in literature [84-87]. In contrast to supervised learning there is the unsupervised learning. In this form of learning one has a set of values and a cost function that has to be minimized. The main areas which are targeted by this kind of learning are: estimation problems, clustering application, the estimation of statistical distributions, filtering and data compression. More details on unsupervised learning can be found in literature [85, 88, 89].

There is another type of artificial neural networks proposed by Chua *et al* and called cellular neural network (CNN). CNN is a combination of cellular automata and traditional artificial neural networks [90-92]. CNN consists of huge number of cells which are connected locally, that is, each cell is only connected to its neighbors. CNN is getting very popular in image processing [93]. One can train it by heuristic learning methods such as genetic algorithms and iterative annealing [94]. Another way to train the CNN is direct mapping of equation on the CNN. If we have PDE with two or more independent variable, it is possible to convert it to set of ODEs with one independent variable. We know that CNN is consisting of many integrators and some other active and passive elements which are coupled together with a specific topology. These coupled integrators are helpful for solving ODEs. Hence, if we solve these sets of ODE on a single CNN layer, the solution of complete system is solution of our PDE.

## 4.3 Contribution to an image processing platform for ADAS

In ADAS systems we need different types of filters and image processing components which are very complex and time consuming. We need a uniform platform as an image processing framework. Flexibility in design, the capability of reconfiguration of modules, the capability of redesigning the system architecture and a very short processing time along robustness are the main characteristic and properties of the ideal framework. In this thesis, a hardware architecture implementing a CNN processor matrix for performing different image processing filters and algorithms is provided. For implementing CNN on a digital platform we need an accurate approximation of the CNN equation in a discrete mode [95-97]. In this thesis the architecture of a CNN implementation based on GPU and FPGA are proposed. Figure 4-3 does show the abstract model of the GPU based system which has been proposed.

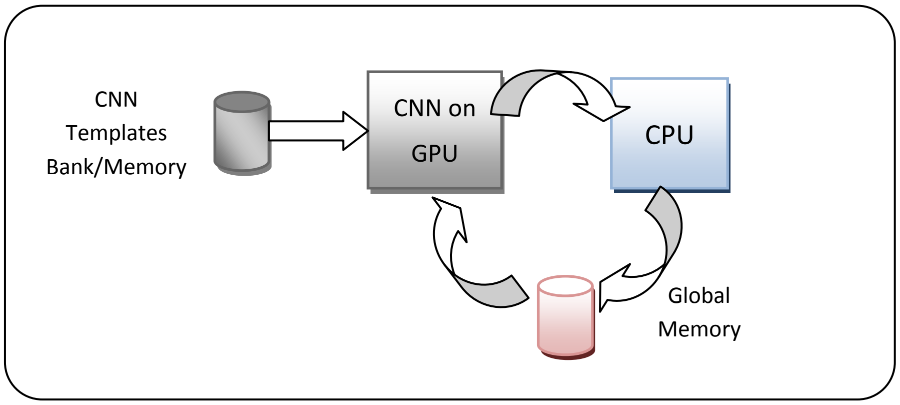


**Figure 4-3: Architecture of system for processing images based on CNN**

To have more flexibility in design and accuracy in result, software based implementation of CNN is a good option. The only drawback is that by increasing the CNN size, the CNN performance will be very poor. Therefore we proposed a parallel implementation of CNN on GPU. Instead of programming in pixel level by vertex engine and fragment engine we proposed an implementation on OpenCL platform. OpenCL which is a heterogeneous platform for high performance computing on GPU and CPU devices provided a sort of APIs for execution of kernels on computing devices and communication between them. Kernels are distributed in the form of one, two and three dimensional and they following hierarchical abstraction mode. In GPU device there is local, global and constant memory for computing and each computing unit has a local memory. OpenCL can manage easily local communication between these memories between different kernels. Figure 4-4 has shown the overview of the CNN GPU design, this part has been describe in details in chapter 8.

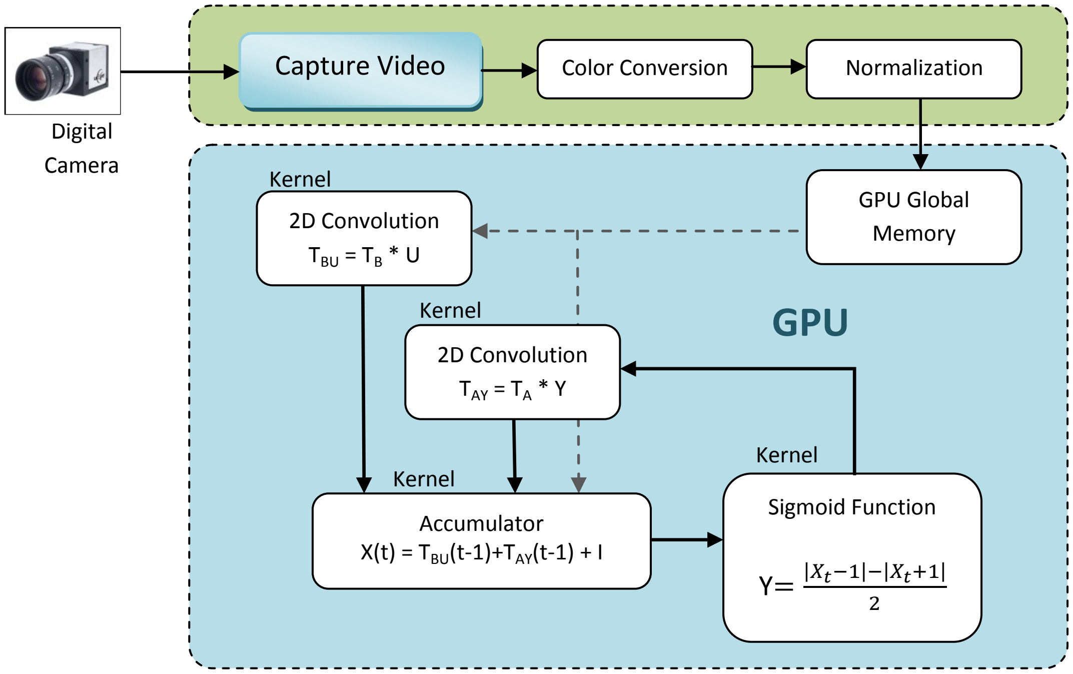


**Figure 4-4: CNN architecture based on GPU.**

In many cases because of power consumption, size and price using computer for processing is not a good idea; therefore we should choose another alternative platform like embedded system, system on chip or FPGA board. In chapter 7 we have proposed a CNN implementation on FPGA. In contrast to developing on GPU, traditional method of FPGA designing is not very flexible and easy. There are many obstacle and reasons behind this issue, such as low and middle level programming, hardware and software integration, low level programming for peripherals I/O and leak of high level debugging tool. But new technology in FPGA developers exist that called Impulse-C™. This package and software allows developers to write C-language for designing a digital system. Impulse-C directly can optimize for XILINX™ FPGAs family from C-language. This tool can also integrate the hardware and software as a mixed system. We used this system for capturing video from camera daughter board which is connected to the FPGA board. Another main advantage of this system is communication channels between software and hardware using *Fastest Simple Link* (FSL) and *Processor Local Bus* (PLB). We design complete architecture of CNN module in C in form of fixed-point. Impulse-C provided a very powerful debugger based on

Microsoft Visual Studio™ editor and Eclipse IDE. During the design and debugging phase we used this debugger for loading image into the CNN model and test the functionality of different modules and complete system. After debugging system we generate the optimized FPGA hardware and software interface and programmed on the FPGA by XILINX EDK ™. Details of complete system are described in chapter 7.

# Chapter 5

# 5. CNN template calculation schemes with a particular focus on the learning/training based approach through Genetic Algorithms

In this chapter the focus lies on the following research question: ***"****What are the major template calculation schemes of relevance for CNN based image processing? How can these calculations be performed in a real-time high performance computing context?"*

## 5.1 Introduction

Cellular Neural Networks technology provides a very powerful analog computing architecture for a variety of array computation and image processing tasks [94]. From a theoretical point of view CNN model offers a huge potential for modeling image processing filters and operators on a CNN Universal Machine.  Each CNN processor matrix used in image processing has a feedback template, a feed-forward template and a bias template. These three templates can reconfigure the CNN model without any changes in hardware. The most challenging issue is to find an optimum set of proper template values for each specific application[98]. Figure 5-1, shows this CNN architecture. Overall, there are three major ways to calculate the feed-forward and feedback templates:

**(a) The Intuitive method:**

This first method is needs intuitive thinking of the designer [99]. Depending on the designer's experience in either processing images or dynamics of arrays, we can have a template. There is no guarantee to find a template for all image processing operators and could be very difficult to find a template for complex solution. Experts are familiar with template of basic image processing operators and they can combine different templates or performing them on CNN individually one by one. For example, we know the template of Laplacian of Gaussian for finding edges and also template for smoothing image. If we combine these two templates on a control template and feedback template respectively, results will be an enhanced image with sharpen edges.

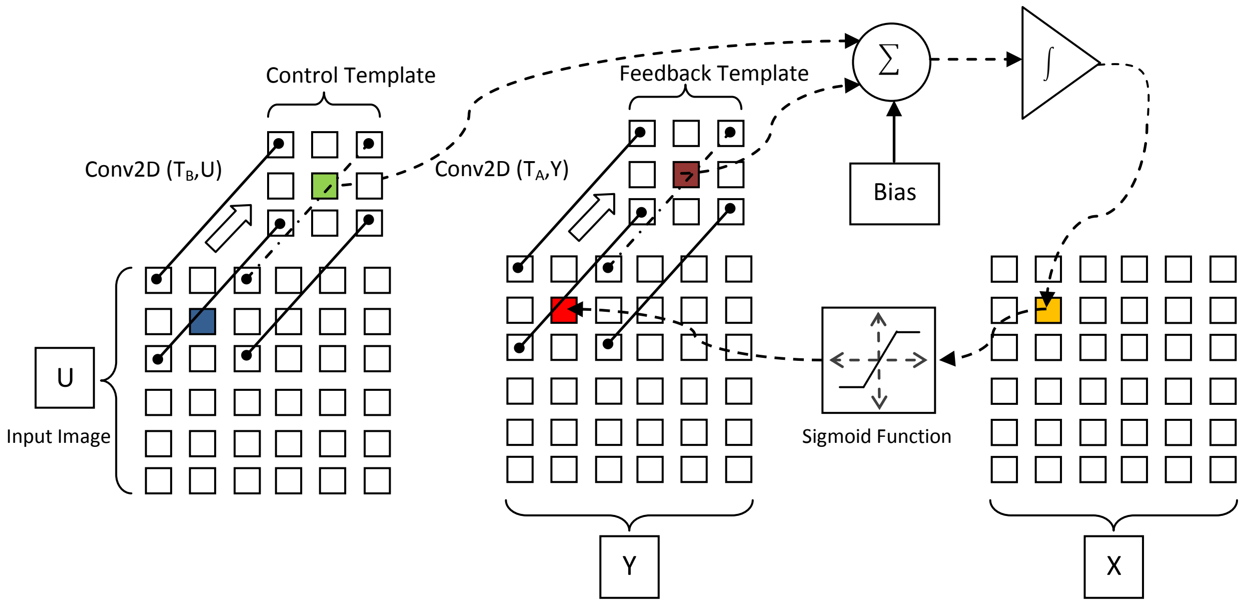


**Figure 5-1: CNN Architecture**

### (b) The heuristic based learning method:

Another method for template designing is the learning/training method [99]. This method is very popular in image processing. During the learning phase, there is a pair of input and target images that is supposed to be generated with a better and better becoming template. After every iteration step a fitness function evaluates the error between the input image and the target image. In some cases output of CNN is very sensitive to changing template. Hence, it's very difficult for GA to find a proper template and the learning algorithm never stosp because the error-level remains high and an appropriate template therefore does not exist. Another problem is that it could take too much time and the error will never converges to the minimum level [99],[94]. This does happen when the learning method/process is trapped in some form of local minimum.

**(iii) Direct template derivation method:**

The third method is the direct template design for those desired functions that are exactly explicit. This method is accurate but it is not always trivially possible to map any desired function onto the CNN system model. Depending on the function, enhancing the CNN is possible, such as adding a new layer or a specific nonlinear term [98]. We know that there are many application based on different PDE model, such as inpainting for recovering corrupted regions in image [70], image segmentation [71], noise reduction edge preservation [72]. The procedure of solving PDEs in CNN is by transforming a PDE to set of ODEs as a coupled system. After transforming a continuous spatial PDE to an array of discrete interactive systems which are ODEs, we can map it on CNN cells. Because CNN is natural and flexible paradigm for modeling a simple locally interconnected dynamical system which are grid base. Detail of this template modeling is already described in sub-chapter 3-3.

Our goal in this chapter is to give a practical introduction to template design. We however focus on the heuristic method based on genetic algorithms.

## 5.2 Genetic algorithm based template optimization for a vision system

A concept is developed for training and optimizing the templates of a cellular neural network involved in obstacle detection. The concept uses a genetic algorithm (GA) for training the cellular neural network. The traditional genetic algorithm method involves the creation of an initial population of random solutions (chromosomes) in binary format, the so called chromosomes encoding. But our genetic algorithm approach defines the chromosomes in the form of real numbers, thus eliminating the need of encoding and decoding of the chromosomes. The results do not differ, by no means, with those of the traditional methods. This method is used here for obstacle detection for autonomous vehicles giving two stereo images of a sequence as inputs. The output results for various different image processing tasks are also presented.

### 5.2.1 General background

The problem of obstacle detection for the vehicles driving with or without driver assistance is one of the major challenges in the field of robotics and machine vision. A robust mechanism inspired by the most complicated and accurate vision system, i.e., that of human beings, needs to be sorted out properly. The problem of identifying the changing environment of the roads, detecting the potential obstacles and avoiding them are tremendous tasks in the field of machine vision. The basic aim of obstacle detection is to extract/identify feature points/parts in images and removing all the other image contents. The most important factor which is always needed to be fine-tuned is the speed. This process needs to be accurate and should be carried out with a very fast speed. The common approaches in this context use analytical and statistical methods like motion estimation or the generation of maps. One of these methods involves features extraction, subsequent displacement vector estimation and a robust estimation of the motion parameters. Since this procedure is composed of several processing steps, the error propagation of the successive steps often leads to inaccurate results [100]. Through using CNN a direct obstacle detection can be performed which eliminates the above mentioned problems. The parallel computation paradigm of CNN provides a fast processing mechanism. Presenting two stereo images of a sequence to CNN, to highlight the 3D objects as potential obstacles in the image, provides a fast and robust mechanism for obstacle detection.

For obstacle detection using CNN, there is a need of training CNN for highlighting the 3D objects in the image. The training process includes parameter optimization for CNN. This approach has also been used in [100] which uses the so-called iterative annealing [101] method for parameters optimization. For carrying out this task, a CNN with 5-by-5 neighborhood and a polynomial cell coupling of degree 3 is used in this work. One of the drawbacks in iterative annealing is the possibility of trapping into local minima and ending with an incorrect solution. The approach presented above is adapted in this work by carrying out the same task by using a 3-by-3 CNN processor matrix. For template/parameter optimization, we do use a genetic algorithm. A genetic algorithm is a learning algorithm based on the mechanism of natural selection and genetics, which has proven to be effective in a number of applications [102]. Suitable selections of its operators enable the algorithm not to fall into local minima. The common approach of genetic algorithm involves creating an initial population of binary numbers that represent the possible solutions. The search evolves with these initial population members (called chromosomes) and manipulates them in order to achieve an accurate or optimal solution. The population of binary numbers needs repeated encoding and decoding process. Also the sizes of chromosomes are very large and vary proportionally to the problem variables. We do use a 'real coded' approach of genetic algorithm that exploits an initial population of real numbers rather than binary numbers. This eliminates the need of repeated encoding and decoding of chromosomes and improves both efficiency and speed of the algorithm. The approach was not only used for finding obstacles in the images but also for other image processing tasks: e.g. thresholding, noise removal, filling etc. Even in the lastly mentioned cases the concept was found to produce good results.

The next section discusses a brief introduction of CNN and genetic algorithm as a good candidate for CNN parameter optimization. The obstacle detection using CNN based on the real coded approach of a genetic algorithm is considered as well. Various output results for different image processing tasks are also presented.

### 5.2.2 Principles of Cellular Neural Network

The Cellular Neural Network (CNN) concept was introduced by Leon O. Chua and Ling Yang in 1988. It is a massive parallel processing paradigm which combines some of the features of Cellular Automata (Discrete states, concept of neighborhood) [103] and Artificial Neural Networks (simple processing elements, continuous states and parallel computation) [92]. CNN is an n-dimensional array of mainly identical systems, called cells [83]. What distinguishes CNN from traditional Artificial Neural Networks is the locality of connections. Unlike artificial neural networks, every cell in cellular neural network communicates directly to its nearest neighbors only. The locality of couplings contributes to the amazingly enhanced processing speed in cellular neural networks. Each cell is made up of a linear capacitor, a non-linear Voltage-controlled current source and a few resistive linear circuit elements, as shown in Figure 5-2 [104].

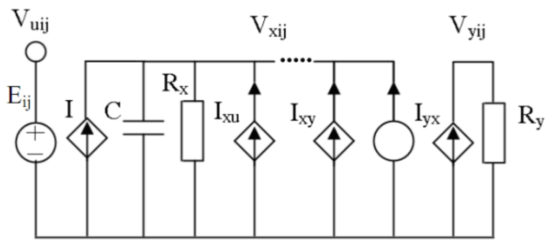


**Figure 5-2: Basic architecture of CNN cell: the equivalent electrical circuit**

In Figure 5-2, C is a linear capacitor; $\mathrm{R_x}$ and $\mathrm{R_y}$ are linear resistors; I is an independent voltage source; $\mathrm{I_{xu}}$ and $\mathrm{I_{xy}}$ are linear voltage controlled current sources with the characteristics $\mathrm{I_{xy}(i,j;k,l) = A(i,j;k,l)u_{ykl}}$ and $\mathrm{I_{xu}(i,j;k,l) = B(i,j;k,l)u_{ukl}}$ for all $\mathrm{C(i,j) \epsilon N(i,j)}$; $\mathrm{I_{xy}}$ is a piecewise-linear voltage-controlled current source. Applying *Kirchhoff's Current Law* (KCL) and *Kirchhoff's Voltage Law* (KVL), the following state equation of CNN can be derived.

(5-1)

$$\dot{x}_{i,j} = -x_{i,j} + \sum_{c(k,l)\in N(i,j)} A(i,j;k,l)y_{k,l} + \sum_{c(k,l)\in N(i,j)} B(i,j;k,l)u_{k,l} + I$$

Where $\mathrm{x_{ij}}$ is the state of the cell C (i,j); A(i,j;k,l) and B(i,j;k,l) are the feedback and control templates respectively for all cells C(k,l) in the neighborhood N(i,j) of cell C(i,j).

The output equation is given as:

$$y_{ij} = \frac{1}{2}(|x_{ij} + 1| - |x_{ij} - 1|) \quad (5\text{-}2)$$

The feedback template (A), the control template (B) and the Bias (I) are all core parts of a CNN processor concept and contribute to the determination its output for a given input.

### 5.2.3 Genetic algorithms

The 'genetic algorithm' concept was developed by John Holland in 1960s [105]. It is an effective method for determining the parameters for CNN. This method is inspired by the mechanism of natural selection and genetics. It has been effectively used for solving difficult search, optimization and machine-learning problems. It works by creating genotypes (set of chromosomes) that represent the possible randomly chosen solutions. The search evolves to improve the quality of chromosomes until the best chromosome is found that represents the optimal solution[94]. The process of evolution occurs in the form of generations and in each generation better chromosomes are sorted out. The parameters of genetic algorithm that play an important role in the process of evolution and of finding the best solution are the following: initial population, selection, reproduction, crossover, mutation, and fitness function [94].

### 5.2.4 Initial population for the genetic algorithm

An initial random population of the chromosomes is generated. Each chromosome represents a possible problem solution. All chromosomes are composed of a fixed number of genes. The approach we used in genetic algorithms represents the chromosomes in the form of real numbers instead of binary digits. This creates chromosomes of relatively smaller sizes and the repeated operations of encoding and decoding are eliminated. For a 3×3 CNN, the total numbers of genes contained in a chromosome are 19. Thus, 9 genes are for the control template; 9 for the feedback template and 1 for the bias. Similarly, if used for 5×5 CNN, the total number of genes in a chromosome would be 51.

### 5.2.5 Selection theory in the genetic algorithms

As the process of evolution creates new generations and since every new generation contains better chromosomes than the previous one, the process of selecting better chromosomes for a next generation is very important. The selection process does not consider the chromosomes merely on the basis of fitness. This process is rather random, through which the parents for the new generations are selected randomly based on their fitness. The method of selection used in our genetic algorithm approach is based on selecting parents from half of the population. After evaluating the fitness of all the chromosomes, the population is sorted in descending order. The best chromosomes occupy higher locations in the list. Every time the two chromosomes are selected randomly for crossover and mutation from half of the population. In order to increase the probability of selecting good parents, the parents can be selected from a specified fraction of the half of the population. For example, the first parent can be selected from half of the population and the second can be selected from 30% of the half of the population from the pool of much better parents.

### 5.2.6 Reproduction in the genetic algorithms

The reproduction phase involves generating a new population for the next generation from the selected parents by applying two genetic operators, crossover and mutation on the selected parents. These two processes result in the next generation population of chromosomes that is different from the previous generation. In our approach, after parents' selection, crossover, mutation and fitness evaluation, the two children can go to the next generation only when the fitness of each one of them is greater than the worst child. The first child is compared with the worst parent and if its fitness is greater than the worst parent, it is replaced by the worst parent and the population is sorted in descending order. The same is done with the second child.

### 5.2.7 Crossover and mutation in the genetic algorithms

After selecting parents from the population by any suitable mechanism, the genetic algorithm operator crossover is applied on the selected parents. Crossover breeds the selected parents to produce new children for the next generation. For the reproduction phase and to produce the next generation, breeding the parents to produce new children is necessary, otherwise the evolution process cannot proceed to better solutions. Crossover can (but not every time) produce children that have better fitness than the parents. We use a 2-point crossover in which two crossing sites are selected in the parent's chromosomes randomly and the genes between the crossing sites are interchanged between the two parents.  Mutation is a genetic operator that maintains genetic diversity from one generation of a population to the next. The purpose of mutation is to prevent trapping into local minima. In our approach of genetic algorithm that uses real number chromosomes, any arbitrary number (gene) in a randomly selected parent is changed by a randomly selected number that falls in the interval to which all the chromosomes genes belong.

### 5.2.8 Fitness function in the genetic algorithm

Fitness function plays an important role in determining the exact solution. It determines the fitness for every chromosome in every generation by comparing it with the original solution. The exact solution can be reached when an exact (or near to exact) match is found between a chromosome and the original solution. The search may also finish when a specified number of populations/iterations has been completed. The fitness function used to compare every output image with the target image in our genetic algorithm approach is based on Euclidean distance between the two images. It first calculates a cost which is a measure of to which extent two images differ from each other.

This cost value is then mapped to a fitness value which represents the fitness of the output image. Throughout the evolution process, the genetic process aims to minimize the cost function and increase the fitness value.

(5-3)

$$Cost\ (i,j) = \frac{\sum_{i=1}^{N}\sum_{j=1}^{M}(I(i,j) - T(i,j))^2}{4MN}$$

(5-4)

$$FitnessValue = 1 - Cost$$

In Equation 5-3, I represents an M×N input image into the CNN processor and $T$ does represent a M × N target or reference image. The Euclidean distance is useful to find the pixelwise difference between the input and the target image. The denominator 4×M×N is used for normalizing the cost between 0 and 1. The input image is normalized in the range [-1, +1] before being fed into the CNN.

### 5.2.9 Obstacle detection through the developed concept

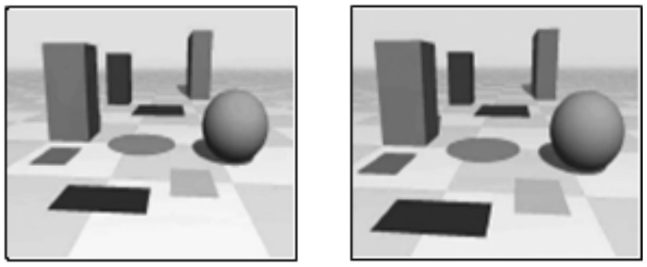
Collision prevention for autonomously navigating moving vehicles driving without driver assistance needs a robust prediction of potential objects. The basic aim of obstacle detection is to extract feature points in images.  Two images of a synthetical image sequence are presented in Figure 5-3 showing a ride over a textured plane on which three dimensional objects are located; the image has been recorded by a moving camera. As in real traffic scenes, the motion direction and the viewing direction are identical. The goal is to find the templates of a CNN processor that is able to extract the three dimensional objects by presenting two images of such a sequence [100]. The task can be performed by removing all the details inside the image except the 3D objects that represent the obstacles. The approach used in [100] performs the edge extraction of the two images and then thresholding as shown in Figure 5-4. For this, the two thresholded images are presented as input to CNN for training along with another target image. For the sake of comparison, we have used the same images. We found that using suitable CNN parameters, a direct thresholding of the image can also remove the background and highlight only the foreground objects. A direct thresholding for textured plane removal is shown in Figure 5-

5. The next step is to remove the objects present on the plane and to extract only the 3D objects in the image that represents the obstacles. This is done by presenting two thresholded images of this sequence to CNN along with a reference or target image.

Figure 5-6 shows the initial condition, the input and the target images applied to the CNN processor. A target image is constructed by removing all the plane objects and by leaving only those above the plane. The targeting is achieved just after 170 iterations and thereby producing the following CNN parameters:

$$A=\begin{pmatrix}0.5 & 1 & 3.2\\ -3 & 4.8 & 0.3\\ 1.4 & 5 & 1.7\end{pmatrix}, B=\begin{pmatrix}4.5 & 3 & -0.1\\ 4.4 & 4.5 & 1\\ -1 & 3.2 & 4.6\end{pmatrix}, I=-3.8 \tag{5-5}$$

Figure 5-7 shows the output of the concept's simulator. It can be seen that all the plane textures are removed and only the objects above the plane remain. The final output image does not contain the lower edges of the objects since there is no way to discriminate the lower edges of the objects and the edge pixel of texture on the plane.



(a) (b)

**Figure 5-3: Two images of a synthetically generated image sequence**

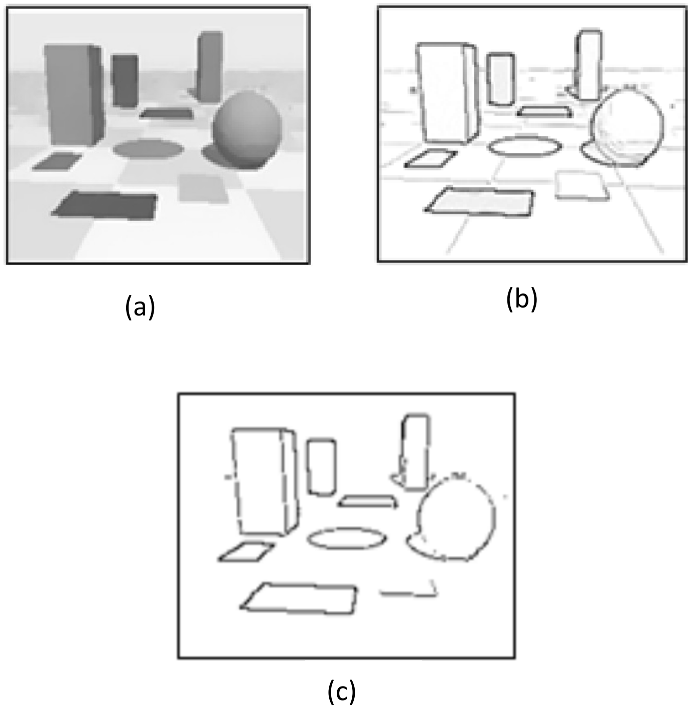


**Figure 5-4: (a), (b), (c): Input image, edge extracted image and threshold image respectively**

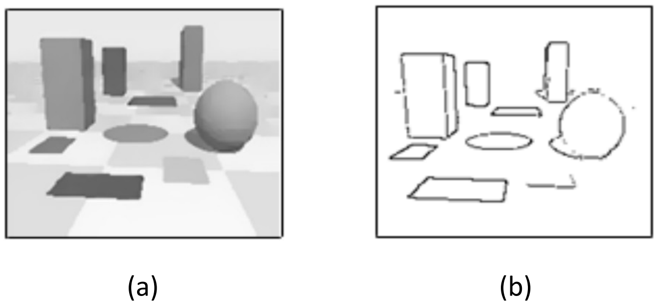
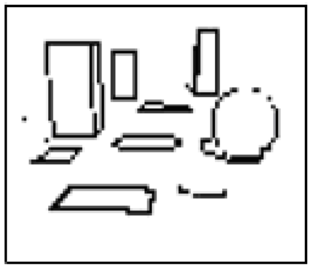
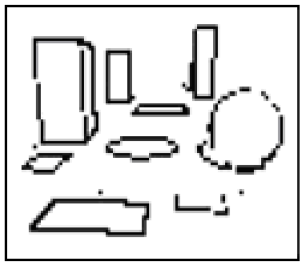


**Figure 5-5: (a) Input image and (b) thresholded image having no textured plane**


(a)


(b)

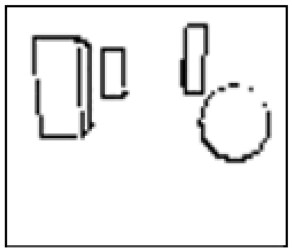

(c)

**Figure 5-6: (a) initial condition image (b) input image (c) target image**

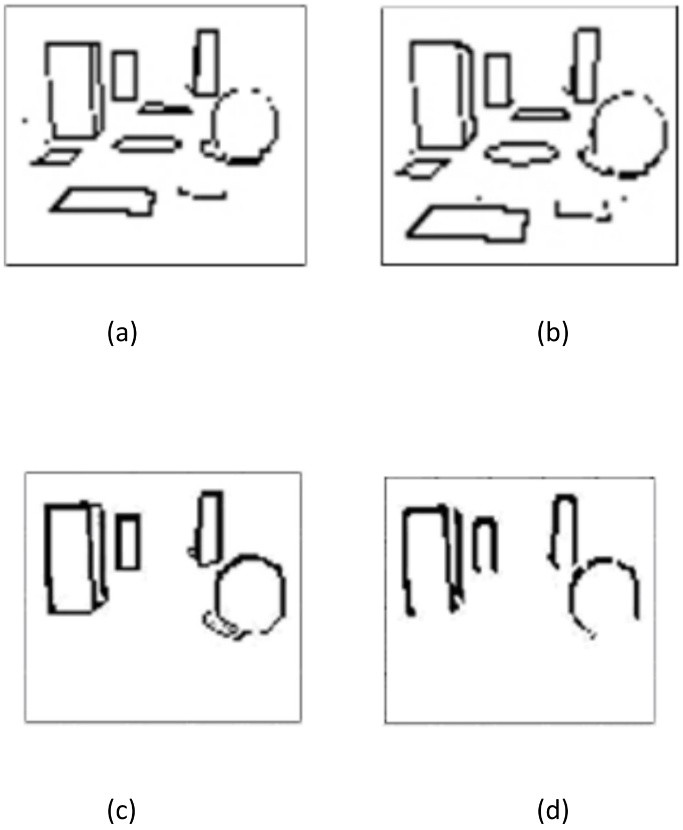


**Figure 5-7: (a), (b): Two input images of a sequence; (c): Target Image; (d): CNN generated output image.**

Other images of the sequence were also tested in the same way and the results are similar.

## 5.3 Experimental results

We tested the real coded approach of genetic algorithm for a number of images. Some sample results are shown in Figure 5-8 to 5-11. In these figures, (a) represents the input image, (b) the target image and (c) the CNN generated output. The CNN generated parameters are also given with corresponding results.

Figure 5-8 shows the separation of a rectangular part from a square as specified in the target image.

(5-6)

$$A=\begin{pmatrix}4.6 & 3.7 & 3.2\\ 3.3 & 4.5 & -0.8\\ 2.1 & 0.5 & -1.5\end{pmatrix}, B=\begin{pmatrix}3.3 & 2.5 & 3.6\\ -0.6 & 1.3 & 2.4\\ 4.2 & 0.8 & -2.7\end{pmatrix}, I=4.6$$

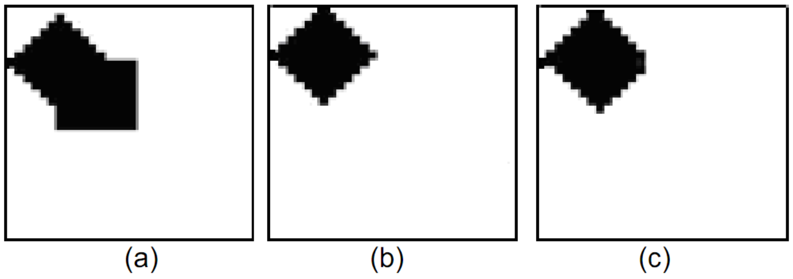


**Figure 5-8: Removing the rectangle part from the figure**

As it appears in Figure 5-9 (a), a noisy image is considered as input and a target is provided for noise removal. Figure 5-9 (c) shows the output without noise.

(5-7)

$$A=\begin{pmatrix}-1.92 & 3.28 & -1.68\\ 2.48 & 4 & 2.16\\ 0 & 3.44 & 0\end{pmatrix}, B=\begin{pmatrix}1.6 & 4 & 2.96\\ 1.92 & 1.68 & -0.08\\ 0.72 & 3.44 & -1.92\end{pmatrix}, I=3.36$$

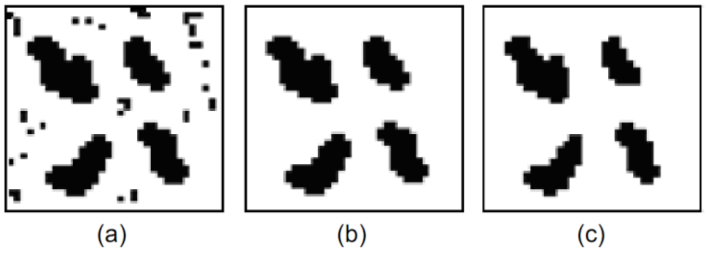


**Figure 5-9: Removing noise from the image**

An image having a tube is fed to the CNN, as shown in Figure 5-10(a). The aim is to fill the tube with a given dotted pixel. Figure 5-10(c) shows the output as specified by target.

$$A=\begin{pmatrix} 0.9 & 4.6 & -1.7 \\ 3.7 & 4.1 & 0 \\ 2.6 & 1.2 & -2 \end{pmatrix}, B=\begin{pmatrix} -2.1 & 1 & -0.7 \\ 1.4 & 1 & 4.7 \\ -2.7 & 3.4 & -3.7 \end{pmatrix}, I=-4.5 \tag{5-8}$$

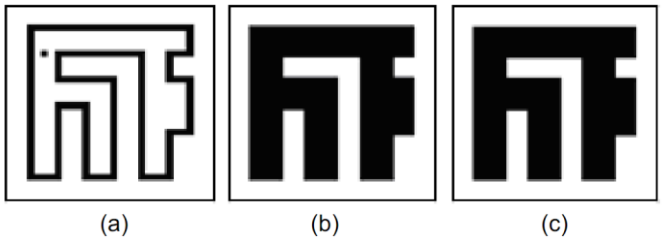


**Figure 5-10: Filling a long tube with a dotted pixel**

In Figure 5-11, a changing gradient image was provided as input. The aim is thresholding up to a specified limit as shown in the target image. Figure 5-11(c) shows the output image for the following set of parameters.

$$A=\begin{pmatrix} 2.7 & 2.6 & -2.8 \\ 4 & 4 & 1.2 \\ -3.3 & 3.9 & 1.2 \end{pmatrix}, B=\begin{pmatrix} 4.2 & -2.4 & -1.4 \\ -3.5 & 1.8 & 4.1 \\ -2 & 4.3 & 1.2 \end{pmatrix}, I=4.6 \tag{5-9}$$

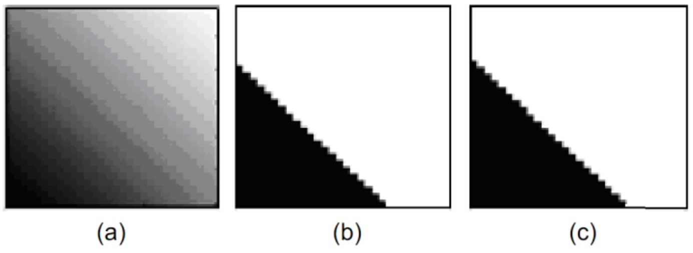


**Figure 5-11: Thresholding to a specified limit**

Obstacle detection was performed by a sequence of stereo images using Cellular Neural Networks. The CNN parameters were determined by a genetic algorithm based on real number chromosomes. Using real number chromosomes, repeated encoding and decoding of the chromosomes is not required. Unlike binary chromosomes, relatively smaller chromosomes are produced. This approach was successfully implemented for obstacle detection and also it was found to produce good results for many other image processing tasks. In future, we aim to implement this approach on hardware level for enhancing efficiency and speed.

## 5.4 Real-time computing issues for the genetic algorithm based CNN template's calculations

Adaptive image processing and image analysis is very important for ADAS concepts. Processing image under different and environmental visual conditions such as fog, rain, and sun in background is not trivial. To overcome problems in this form we need to involve adaptive image processing techniques. Our CNN platform can process images with a performance of 100 FPS. For dynamic processing, CNN needs dynamic templates or set of templates. To have a fast response, CNN needs to access different templates very fast. There are three ways for real-time template accessing. Template pre-calculation is one solution. There are many standard and classic templates like thresholding, contrast enhancement, dilation, opening, erosion, closing, find edging, median filter, etc that can be pre-calculated and stored in a list. And depending on the situation the procedure system can call different templates. Figure 5-12 shows this structure for pre-calculation of templates.

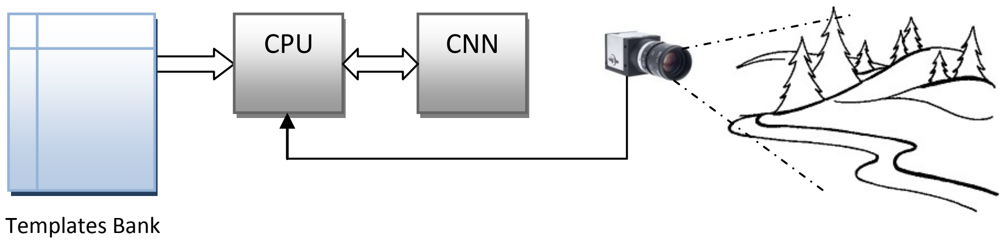


**Figure 5-12: Processing scenario by pre-calculated CNN templates**

Another way is using that type of template which is calculated by PDE's. If we map PDEs to templates in a parametric way then we are able to change the parameters in real-time for adaptivity. In [106], A Gacsádi *et al* proposed a PDE based template for contrast enhancement. They consider the energy function $E$ as indicated in Equation 5-10, and try to minimize this function.

(5-10)

$$E(\phi, G) = \iint \|\nabla\phi\|^2 dxdy + \lambda|G|$$

The first term of this equation is a smoothness constraint and the second part is an edge penalty. During the minimization, there is a tradeoff between image smoothness and image deblurring. The following Equation 5-11, is an approximation of the contrast enhancement equation in a single layer CNN.

(5-11)

$$A = \begin{bmatrix} 0 & 0.25 & 0 \\ 0.25 & 0 & 0.25 \\ 0 & 0.25 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & -\lambda & 0 \\ -\lambda & 4*\lambda & -\lambda \\ 0 & -\lambda & 0 \end{bmatrix}, \text{I=0}$$

In this template $\lambda$ is a scalar coefficient as a ratio between contrast enhancement and smoothness level. Mapping any PDEs with parametric coefficient is possible; therefore depending on the situation, a main controller can change these coefficients to adapt the CNN result. By this way we can pre-calculate templates for a wide range of problems and use them dynamically in a real-time situation.

There is a hardware based solution also for template calculation. Theoretically genetic algorithm is a time consuming optimization technique in any domain of science. There are two issues that make GA slow. The first issue is the nature of the selection process. This means that for optimizing a solution we have to perform the objective function to a huge amount of chromosomes until it converges to the global minimum. Another problem comes from the performance of the system. In our case (i.e. Finding template for CNN) for evaluating a quality of chromosomes as a solution, the system needs to decode the chromosomes and apply it on the CNN and check the results according to the objective function. In [107] D. Balya *et al* did analyse several papers to study techniques of template calculation based on genetic algorithms and proposed an analogic implementation of the genetic algorithm based template calculation on FPGA. In genetic algorithms the fitness function is a soft computing module and the rest of the system could be implement on any hybrid or homogenous machine. Figure 5-13 shows the schematic of ideal system for calculating the CNN templates.

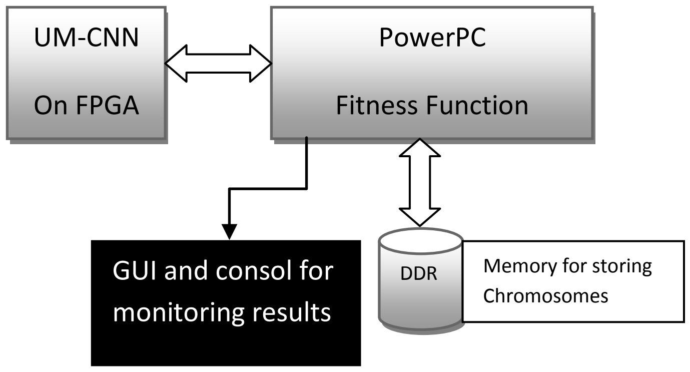

UM-CNN
On FPGA
PowerPC
Fitness Function
GUI and consol for monitoring results
DDR
Memory for storing Chromosomes

Figure 5-13: Integration of CNN on FPGA with PowerPC for speeding up of genetic algorithm

# Chapter 6

## 6. Emulation of analog computing on FPGA

In this chapter the focus lies on the following research question: *"How far can the advantages of analog computing be used/gained through an emulation of analog computing on digital hardware platforms like FPGA?"*

### 6.1 Introduction

The main advantage of analog computing is that we can simultaneously get the result of complex mathematic equations [108] [109]. All signals are running in parallel and in real-time and electrical elements can compute simultaneously. In the traditional analog computing approach/concepts we do face a scaling problem for the dynamic range of computing. All the components are limited in the range of their respective dynamics and one does also face problems related to noise and high voltage. Therefore, it is not possible to compute any dynamic range and one has to always rescale the ranges. Another disadvantage of classical analog computing is that the solutions appear immediately in real-time and eventually on cannot easily record or analyze them. We can alleviate all of these problems by using the advantages of digital systems like FPGA.

Traditionally, analog computers were using OP-AMP's (operation amplifiers) to model "adder", "subtraction", "multiplier", and "integrator". We can however model all of these functions by digital circuits in FPGA. The "Digital Differential Analyzer (DDA)" is a functional block to compute the integral of a function over time. To get more speedup we can use fix- point calculations instead of floating point one. Therefore, we must check the range of values and the level of accuracy for assigning sufficient reserve bits for both the integer and the fraction part. By emulating analog computing on a digital platform like FPGA we can get the solution in real-time without any limitation in scaling voltage since instead of voltages we are dealing (in digital circuits) with registers, fixed-point operators and data. Another advantage is that one does not need any extra converter for storing data in memory as they are already in digital form. Further, we have full control on the clock rate; and this also a great advantage of using FPGA and digital architectures. Setup time, debugging and reconfiguring the digital emulation of analog computing is are so fast and flexible that the digital emulation appears immensely much better than the traditional analog computing systems.

## 6.2 New computational modeling for solving higher order ODE's based on FPGA

In this research we propose a method for solving complex higher order ordinary differential equations (ODE) based on an emulation of the analog computing paradigm on digital hardware platforms. In this case, we mimic real analog system elements by digital discretized models. Due to the flexibility and reconfigurability of FPGA and also the possibility of system behavioral modeling through hardware description languages (HDL), we are able to create all fundamental elements that are necessary to both simulating complex systems and the modeling of any ordinary differential equations (ODE) or a system simulation based on ODEs. We therefore propose a novel methodology of solving systems and higher order ODEs. This technique is similar to the analog computing but with the key difference that we possess more flexibility and are able to control at will the precision level wanted/needed. Further features are values scaling of both the results and internal variables.

## 6.3 General background

Solving ordinary differential equations is essential in most scientific fields [110]. Around the 1950s, analog computing was the only solution/technology for solving the differential equations and simulating complex dynamic systems by using analog electronic components. In the last decades this method has been pushed away by the digital computing revolution. Almost all researchers have switched to numerical and algorithmic methods for solving ODEs. But the digital computing has also its limitations. Thus, recently some researchers are exploring ways to return to the use of the analog computing method again [111], especially in cases where ultrafast speed of the solving process is needed (see realtime simulation needs for example). There are many reasons behind this decision. The first and most important issue is the processing speed. Much of the physical phenomena in the real world are measured/expressed by calculus; therefore with analog computing we can simulate these models very fast. Due to the inherent process and computing parallelization, analog computers are capable of producing complex solutions in real time. In the early days of the analog computing age (more than 40 years ago) they were facing a series of limitations due amongst others to the use of discrete electronic components of this analog computing paradigm: imperfect connections between elements, limitation in the voltage scaling, variability of elements characteristics during the process due to the temperature, etc [32], [108]. Also the precision of the component characteristics values is a serious technological non-scalable limitation. In a real analog computer, there is a limitation of the voltage scaling. The voltage is limited between the noise level and the high voltage level. Because of this physical limitation one cannot reach solutions that are out of this boundary (interval). Since the 1980's up to now, many researchers have been trying to implement analog computation systems for specific problems in VLSI chips [112, 113]. In VLSI chips, one can rescale the voltages within CMOS or TTL ranges. One particular weakness of this approach (i.e., analog VLSI implementation) is that the circuit is not re-configurable and for solving a new problem we have to design another chip, what is a very expensive issue. Another problem is that we are not able to re-scale the time in this case. In some cases, for coupling system components, time synchronization becomes necessary. Due to the limitations of the VLSI approach, we have started thinking of an alternative that consists in the essence of an emulation concept of the analog computer on top of reconfigurable and scalable digital platforms, especially FPGA chips [111, 112, 114, 115]. This research shows that this emulation has been successful and that we are capable of modeling and solving any type of higher order even nonlinear ODE at an ultra-fast speed on this fully digital structure.

## 6.4 HDL Description and system architecture for the analog computing emulation concept of FPGA

Hardware description languages are similar to the normal programming languages. The main difference between sequential programming and HDL programming is that in HDL one describes the operations of digital circuits by code at either low level gates or at behavioral level [111]. But in sequential programming we have a flow chart and an algorithm that should sequentially run on a single execution CPU. In contrast to a software programming language, HDL syntax and semantics include explicit notations for expressing time and concurrency, which are the primary attributes of hardware. There are many types of HDL descriptions that can cover from low level gates up to behavioral modeling, such as Verilog, SystemC, VHDL, HandelC and so on. Verilog is the only language that can cover the whole of this domain. It means that for description of system operations there is the possibility to use a mix of gate level programming and behavioral programming. Also, Verilog is easy to understand, and one can describe the functionality of complex circuits by using this language.

To get good performance we have to execute operations such that as many operations as possible are in parallel mode. This is due to the low clock rate in FPGA if compared to a dedicated CPU. For implementing a system simulation or solving an ODE by a flow diagram, we need some basic and fundamental arithmetic components, such as summation, subtraction, multiplier, and integrator. For solving ODEs by analog computing, there is no general solution, as each differential equation requires its own unique circuit or bus connections. However, one could in theory create a series of reconfigurable matrix switches that would be rewired for obtaining circuits and bus structures to different arrangements depending on the problem (ODE) at hand. The implementation of these functions on a digital platform (here FPGA) is easy, however with the exception of the analog “Integrator”. For this, we use and adapt an old method called *Digital Differential Analyzer* (DDA) for computing time integrals. In the next section, this technique is presented.

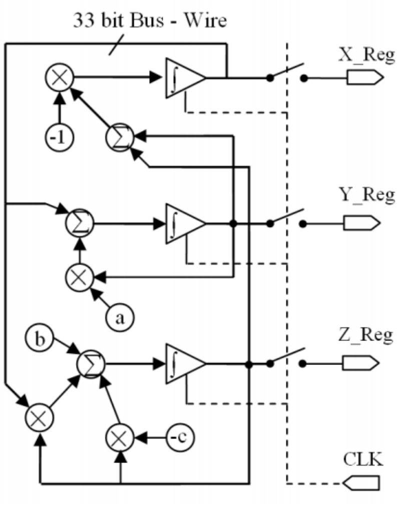


**Figure 6-1: Flow diagram for modelling the Rössler Equation (see Equation (6-2)) in the analog computing scheme emulated on FPGA**

Depending on the accuracy needed either for a given system simulation or for ODE solving, we can define a specific data type for modules and basic elements. For each basic element such as subtraction, addition and multiplier we define a fix-point data type with proper "integer" and "precision" ranges. Depending on the complexity of the equations, we can extend the number of bits used for storing both the "integer" and "precision" values. The *Most Significant Bit* (MSB) bit is for sign, and for subtraction we use the 2's complement method. The key advantage of a fix-point data coding in this case is the processing time and also the saving of FPGA resources. In a floating point representation alternative data coding we should have to use complex modules and architectures and it would need/require more resources on the FPGA when compared to the fix-point alternative. Still now, there is no general purpose architecture for solving any type of ODE equations [115], but we propose a method for designing a system and solving ODEs in a straight forward process (flow diagram). The flow diagram is consisting of many basic elements that are coupled together.

Figure 6-1 shows a flow diagram for solving the so-called "Rössler equation" (see Equation 6-2). In this diagram, outputs are registers, and internal connections are bus wires. The data bus width for connecting the components is the same in all parts of the diagram. We use 33 bits for sign registers, sign wires and storing the values and integrators. All components are synchronous and the system is operated by a common clock.

## 6.5 The "Digital Differential Analyzer" method

A digital differential analyzer (DDA), also sometimes called "digital integrating computer", is a digital implementation of the differential analyzer. The integrators in DDA are implemented as accumulators, whereby the numerical results are converted back to a pulse rate by the overflow of the accumulator. The main advantage of the digital integrator, when compared to an analog integrator, is the scalable precision. Also, in a digital integrator based on DDA, we don't have drift errors and noise [116] due to the imperfection of electronic components. By accumulation over time of values in a register we can calculate the integral of signals. The basic digital integrator is expressed by Equation 6-1.

$$X_{n+1} = X_n + K.S \tag{6-1}$$

In Equation 6-1, $\mathrm{X}_{\mathrm{n+1}}$ denotes the next state of the accumulator used for calculating the integral. The coefficient $K$ is a constant factor that is less than 1; it is used for time scaling. In this equation $S$ denotes the input signal for integration. We can map this technique on FPGA very easily by writing a behavioral code. After each rising clock pulse, the equation updates the integral value. In this integrator, rounding or truncation errors are only due to the limitation of registers. Therefore, by increasing the register sizes we have a way to control/reduce this error. This error is cumulative. Thus, for low precision registers a lack of accuracy will be observed after long time. The only way to overcoming to this problem is setting proper register sizes.

## 6.6 Integration of hardware and software

The hardware implementation of systems for solving ODE equations by this DDA method is one issue. Another issue is to ensure the communication of this module or system with the physical world for setting parameters, coefficients, and initial conditions. For making a System-on-chip (SoC) we need a central processor, a standard bus, and IO cores that are integrated together. The IBM Processor Local Bus (PLB) is a technology that can support and manage all of these facilities. PLBs are supported slave and master for controlling the IP (Intellectual Property) on the bus. There is a possibility to passing the parameters and values either through simple registers in PLB or by a first in first out method (FIFO) in PLB. In Xilinx Virtex-4 family we can design a system based on PLB and custom IPs. In this case, we integrated a PowerPC 405 (PPC) and Verilog modules as a custom IP by a 32bit PLB. PPC is a hardcore 32bit processor that can provide good performance as a central processor for controlling the custom IP and other peripherals. Through this bus also we are able to send interrupt signals, so that Verilog modules can send a signal to PPC 405 as an interrupt for doing some process on the output data. When the output of the module is valid, it sends an interrupt signal through PLB to PPC 405. PLB and everything that is connected to this bus is synchronous due to a common clock. By using System ACE (Advanced Configuration Environment) CF (Compact Flash cards) technology we can record results data in files. Then transferring or sending these data files to a computer (PC) through either the normal or high speed serial link is then feasible. On the PC the results will be visualized, saved or used for other purposes. In some Xilinx family there is a possibility of using a softcore CPU like MicroBlaze, instead of a hardware-core (PPC405). It is obvious that for using software-core CPU like MicroBlaze we need more logic slice resources. Therefore, for saving logic slice resources in software integration, we applied a hardware-core CPU (PPC 405).

## 6.7 Experimental results

For checking the accuracy of the developed system, we have implemented a Rössler equation. Equation 6-2 shows this Rössler equation, which is a set of three highly non-linear ordinary differential equations. This system can, depending on the set of parameters, generate chaotic signals at the output. Equation 6-3 shows the set of parameters we have used for generating chaotic waves. Traditional methods for solving the Rössler equation in digital computers are based on numerical methods, such as Euler or Runge-kutta algorithms. Depending on the initial conditions and the set of parameters, the system can converge to specific orbits, to a point or diverge to infinity. For modeling this equation by HDL we needed 6 multipliers, 3 integrators, and 4 adders. For realizing subtractions, we do apply additions by the 2's complement method. For coupling these components, we have to use wire bus connections. Also, for storing the values of the states we need registers. There is no exact rule for the definition/fixation of the number of bits for both integer and precision parts of the fix-point data coding. Depending on the complexity of the system at hand we can define a safe range for sign register and wires. For Rössler we have set in this work a 33bit bus for connections between modules and accumulating the values in integrators. Further, we have respectively set 1 bit for sign, 8 bit for the integer and 24 bit for the fraction parts. With this setting for the variables and connections, the solutions of the Rössler equation (x, y and z) will be limited within maximum range of ±255 (1-bit sign, 8-bit integer and 24-bit fraction). We have implemented this equation on a Xilinx ML405 evaluation board with a 100Mhz clock rate. This board is based on Virtex-4 FX20 family, with 20k logic cells and 32 DSP48 slices.

(6-2)

$$\frac{dx}{dt} = -y - z$$

$$\frac{dy}{dt} = x + ay$$

$$\frac{dz}{dt} = b + z(x - c)$$

$$a = 0.41 \qquad b = 1.15 \qquad c = 4.16 \tag{6-3}$$

At its most basic level, a DSP48 is a multiplier with a combination of adders and many optional operations. It has an 18-bit sign input signal and a 36-bit sign output result. This result is then sign extended to 48-bits, it can either be fed into the adder or connected directly to the output of a DSP48. Pipeline registers are a unique advantage of the DSP48 block compared to other FPGA DSP architectures [7]. Xilinx has dedicated many DSP48 slice in Virtex-4 and Virtex-5 family for speeding-up of calculations. The amount of resources used after synthesis for solving the Rössler equation in terms of DSP48 slices, Flip-Flop slices and 4-Input LUT Slices is shown in the Table 6-1.

Table 6-1: FPGA Resources used for Implementation of an Emulated Analog Computing Solver of the Rössler Equation on FPGA

| **# Components (available)** | | **18bit Rössler** | **33bit Rössler** |
|---|---|---|---|
| Flip Flops Slices | (17088) | 1% | 1% |
| 4 bit LUTs | (17088) | 1% | 2% |
| Bonded IOBs | (320) | 1% | 2% |
| GCLKs | (32) | 6% | 6% |
| DCM_ADVs | (4) | 25% | 25% |
| DSP48a | (32) | 25% | 62% |

*Integrated Software Environment* (ISE) is the Xilinx® design software suite that allows you to take your design from design entry through Xilinx device programming. By disabling the DSP48 slices in this program, the system tries to synthesize the code and design by normal logic slices. The FPGA synthesis report is in Table 6-2. For the 2 scenarios with and without DSP48 slices; the Table 6-2 shows what difference there is in terms of FPGA resources use/consumption between the two alternatives.

Table 6-2: Benchmarking of resources used for FPGA synthesis report for implementing a 33-bit ODE solver of the Rössler Equation

| # Components (available) | % of resources used when using DSP48 | % of resources used, without DSP48 |
|---|---|---|
| F-Flop Slices (17088) | 1% | 2% |
| 4 bit LUTs (17088) | 2% | 21% |
| Bonded IOBs (320) | 2% | 2% |
| GCLKs (32) | 6% | 6% |
| DCM_ADVs (4) | 25% | 25% |
| DSP48a (32) | 62% | ____ |

In some Xilinx family FPGA, there is no DSP48 slice for implementing integrators, multipliers and basic elements. Thus we need more logic cells to implementing these resources. According to Table 6-2, 62% DSP48 (20 slice DSP48 in Virtex-4, FX20 FPGA) is equal to 21% (3600 slice, 4-bit LUTs in Virtex-4 FX20 FPGA) of basic logic slices.

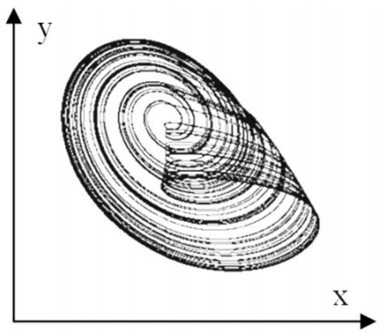


**Figure 6-2: Rössler equation output graph that is generated after a solving by the analog computing emulation on the FPGA board**

Solving the Rössler equation for 6000 iterations in Matlab (time step size selected is 10^-9, on a 3GHZ Dual Core CPU), takes 5400 ms, but in solving the same equation by hardware through the emulated analog computing on FPGA with 300 MHz clock  takes only 5 microseconds. The extremely high speed-up (compared to CPU) of many orders of magnitude (more than several ten thousands) is evident. Figure 6-2 shows the XY plot of Rössler equation output.

### 6.7.1 Future work

In the future, we are going to implement a system for generating the HDL code for solving equations by a flow diagram. For modeling an ODE the future program will be node-based. After coupling nodes by either code or a GUI, the program will be able to generate a gate level HDL code for direct programming on FPGA. By this technique we can speed up the design and implementation process of analog computing solvers on FPGA, which will be capable of solving complex ODE equations and simulating complex systems in real time on FPGA.

## 6.8 Concluding remarks

Each 18-bit integrator takes 1% of the FPGA resources, and each multiplier takes 2% resource. In Xilinx Virtex-5 family there is a powerful chip (XC5VSX240T), which does fit very well to the requirements of our analog computing emulation. This chip has 1k DSP48 slices, what is quite good for implementing integrators and multiplexers. The FPGA synthesis program can combine one integrator and one or two multipliers and adders in each DSP48. By this way we can implement many integrators, adder and multiplier on a single FPGA chip. The obtained results are very encouraging, a speed-up or more than a million has been reached when compared to Matlab performance on a normal PC.

# Chapter 7

## 7. Implementation of CNN on FPGA

In this chapter the focus lies on the following research question: *“How far can an efficient implementation of CNN on FPGA and on GPU be designed and realized? “*

### 7.1 Introduction

The implementation of CNN on hardware is very similar to the emulation of analog computing on FPGA. There are some bottlenecks for accessing the memory and updating state variables for each cell. In the emulation of analog computing in FPGA, we did model basic elements of system which are necessary for modeling the behavior of the system. The same method can lead to implementing CNN on FPGA. In contrast to ANN and because of the local connectivity of CNN cells, one does have the possibility to implement this architecture on FPGA and also on GPU [50]. FPGA macro cells and logical components can work in a highly parallel manner. And because of a very flexible routing between them, we can implement any complex digital circuit or model. To get more advantages from using FPGA we can use high level behavioral modeling languages such as VHDL, Verilog or SystemC. FPGA has local and embedded memory, which is very important for storing the CNN states, otherwise a transceiver of memory between FPGA and an external memory could be very time consuming and constitute a significant bottleneck. Today, most FPGAs have an internal dedicated standard CPU that has access to the hardware and logical field of FPGA through the standard bus controller [117, 118]. There are many high level compilers based on ANSI-C standard for coding and debugging. This technology increases the system performance by integrating of hardware and software. Therefore, loading CNN initial states, templates, and setting time scales and other parameters can be done easily by CPU. The resources of an FPGA are not endless; thus we have to consider this issue while designing the CNN architecture.

Concerning GPU, using GPU is getting more popular every day. The highly parallel structure of GPU makes it more efficient for image processing and for processing large blocks of data. The high memory bandwidth between CPU and GPU, the integration of GPU and CPU through the standard protocols and the running multi-kernels scripts on GPU make it a very efficient technology for the implementation of CNN [119, 120]. Since 2003 GPU technology is growing up dramatically and we can implement very complex models and systems by using flexible and robust tools and high-level software development instruments/tools.

## 7.2 A framework for FPGA based real-time machine vision: direct convolution versus CNN

In this Chapter we compare two different frameworks for real-time image processing, namely (a) a convolution based framework, and (b) a CNN (Cellular Neural Network) based framework. Hereby, a key focus is related to the main factor in image processing and machine vision that is the "processing time". For real-time applications this time must be the shortest possible. Due to the CPU structures (von Neuman architecture), in the classical image processing only a sequential processing of the pixels is possible. In such a context, convolution operations on images, which are very time consuming, will constitute a bottleneck for the whole sequential system. At the end, the experimental results of implementation of a hardware-based processing architecture for both the "CNN based image processing" and the "direct convolution method" on *Field Programming Gate Array* (FPGA) are presented and discussed. Thereby a systematic comparison of the performance achieved by each of the approaches is conducted.

## 7.3 Introduction to video processing platform

FPGA and dedicated video processing systems have been widely used since many years in video and image processing systems and machine vision application such as areal image processing, surveillance, medical imaging, vehicle automation and quality control in industrial systems [26, 49, 50]. In the classical video processing platforms we can use DSP or CPU core for manipulating/processing pixels. For process one frame of data, the system generally has to fetch both the data and the program to either CPU or DSP, perform required mathematical operations and then store the result(s) back into the memory. The system must handle the high priority interrupts at the same time. And all these extra cycles will add to the total number of cycles involved in the processing each pixel of image [121]. The main weakness of these traditional systems is clearly the low speed related to the high processing time. Due to the sequential architecture and the programs, the system cannot manipulate pixels in a real pipeline model. Therefore, we must design a suitable architecture with a pipelining potential. FPGA is one the best candidates for pipelining video processing. With newest FPGA technologies it is possible to design a multi-functional and high performance video processing system. New FPGA technologies have made them much faster and denser than before. XILINX Vertex technology provides a large two-dimensional array of logic and programmable block sets, which contain lot of dedicated memories and flip-flops. Having such facilities and infrastructures, one can easily map the image on this grid for further image processing[49]. This implementation presents a real-time video processing platform involving two concepts: a) direct convolution based image processing, and (b) CNN based image processing. Designing a proper image processing platform is extremely significant. Thus, we have to design a robust and flexible architecture. The main parts of a real-time platform are knowingly capturing video, buffering video streams, video stream processing and finally video output controller. All these parts are considered in the platform design of this implementation.

## 7.4 System Architecture

As already explained, standard architecture for a video processing system contained the essential modules such as capturing unit, processing unit, memory and video output controller. According to the Figure 7-1 we can connect these parts together. To capture video signals we have four possibilities that are using DVI, S-video input, composite input, and dedicated digital camera. The Digital Visual Interface (DVI) is a video interface standard designed to provide very high visual quality on digital video processing and displaying systems.

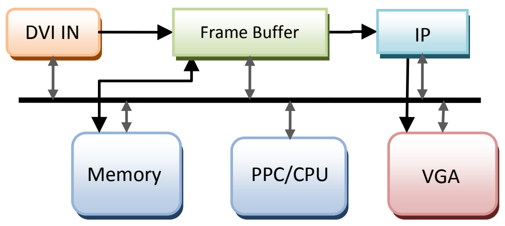


**Figure 7-1: General architecture for stream processing**

Digital DVI signals are partially compatible with *High Definition Multimedia Interface* (HDMI) signals. In this case, we have used an FMC video grabber daughter board as a capturing unit. This board has a central pico-processor to control both the signals and the synchronization with the FPGA board. The video stored in the buffer does contain the pixel values. After video the capturing phase, we must extract and split rows for the video processing module. The video processing module has access to the video memory for storing and retrieving video signals. The main property/function in the video processing is a convolution based filter. Depending on the convolution size, we must split the video stream into many rows and keep it in the internal FPGA memory.

## 7.5 Hardware Platform Specification

In the implementation of this work, we do use a platform comprising the Xilinx XtreamDSP 3400 and a Video FMC daughter board which has a Pico Microblaze for video signals capturing. It has a DVI input, a composite/S-video input and output, and a camera input. This board has two camera interfaces to allow the capturing of data from two cameras at the same time and simultaneously. The camera we use is a custom CMOS camera based on the Micron MT9V022 image sensor chip. This camera sends a high-speed LVDS (Low-Voltage Differential Voltage) data stream format to the board. It operates with a 26.6 MHZ clock rate that is generated by a reference clock which is provided via a fixed-frequency 27 MHZ oscillator on the board. Figure 7-2, shows the camera connection diagram to/in the FMC board. In the initial step we must define the voltage source level for the FMC board as well.

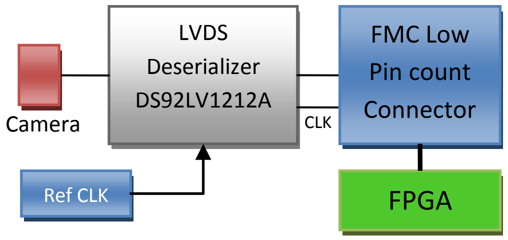


**Figure 7-2: Camera connection to the FMC board**

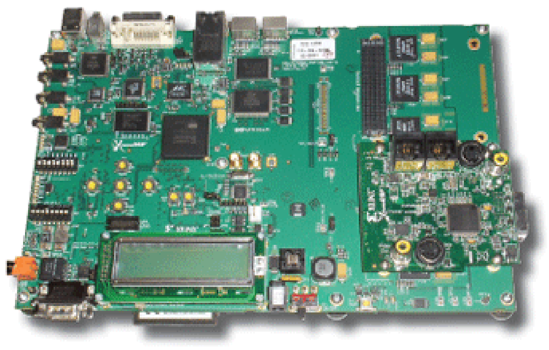


**Figure 7-3: Xilinx XtremeDSP Kit 3400**

Figure 7-3 shows the Xilinx XtremeDSP kit which is very appropriate for image processing and machine vision applications. This platform provides an embedded design framework that can be customized with user defined video accelerators implemented in the FPGA fabric. It has one Spartan FPGA with 3.4 million system gates and 126 DSP48 internal block sets. This board has 256MB DDR2 and 256MB Compact Flash. The clock rate for the FPGA is up to 200 MHZ.

## 7.6 Electronic System Level and High Level Direct Convolution Modeling

There are many possibilities to design a proper model for the video signal processing. As main part of an Image processing engine, we need a convolution operator. This basic and simple mathematic model is very useful for the implementation of many complex filters such as first derivative and second derivative ones, finding edge operations, median filters, noise cancelation, and etc. Due to the complexity of filters the best way to design a convolution module in FPGA is using HDL programming. There are a lot of techniques to implement and generate HDL code. One of the best techniques is using a high-end technology of Electronic System Level design (ESL) like Impulse CoDeveloper. Impulse CoDeveloper is a proper tool for developing custom IP modules such as convolution and image processing modules. This tool allows designers to quickly develop custom filter modules in standard c program. With this tool we can convert an untimed C program to synchronized HDL code. One of the best facilities of this tool is the debugging module for higher level C standard programs such as Microsoft visual studio and so on. CoDeveloper results are fully compatible with standard C and VHDL. Almost all the data interactions between modules and processes in CoDeveloper are based on stream passing and shared memory. For creating these convolution modules we have designed two parallel C-language processes named columns and convolution. The column process has access to the incoming video stream and it can store the pixels values of 3 rows of an image in the internal memory buffer for the convolution module. Figure 7-4 shows these two main modules for splitting rows and convolution. The designer has access to the multi-channel memory controller; by this way the program can use the pipelining technique. Depending on the hardware platform we can use the pipelining technique in our design. Using the Pipelining #Pragma switch, which is a pre-compiled command in C, we can synthesize "for-loops" commands and the internal contents of that in a concurrent mode. If a system has access to the dual-channel external memory, one can thereby synthesise some parts of the code in a pipelining mode which has an interaction with external memory and registers [122].

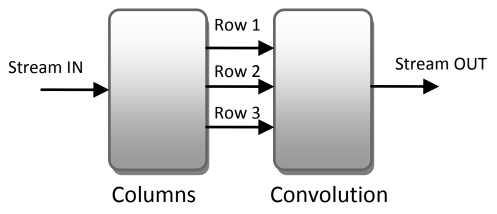


**Figure 7-4: Convolution and stream processing diagram**

These two processes will run concurrently. The first process read the data and split it in 3 separated rows, and the second process applies a convolution on these data. In each clock cycle, the columns module reads a pixel value from the live video stream source and it writes these pixels values on the output stream channels. These pixels values will then be used by the convolution process.

## 7.7 Modeling cellular neural network by DDA

Cellular Neural Network (CNN) is a paradigm for parallel processing that is similar to the neural network but with the difference that the connectivity between the cells is rather local and not global like in neural network. The main parts of a CNN module are the convolution module and the integrator [50]. For designing an integrator module we need a memory for keeping both data and cell values. According to the Equation 7-1, describing the CNN cell dynamics, we need three templates for each cell: a feedback template TA, a control template TB, and a bias template I.

$$\dot{x}_{ij} = -x_{ij} + T_B * u_{ij} + T_A * y_{ij} + I_{ij} \tag{7-1}$$

where, '$T_A$' is denoted feedback template (3x3), and '$T_B$' is denoted control template (3x3), 'u', 'x', 'y' and 'I' are the input, the state, the output and the bias term, respectively. The output signal is related to the Equation 7-2 below equation.

(7-2)

$$y_{ij}(t) = f(x_{ij}(t))$$

In the Equation 7-2, the function $f(x)$ is a nonlinear activation function defined in Equation 7-3.

(7-3)

$$f(x) = \frac{1}{2}\left(|x+1| - |x-1|\right)$$

We can model the solution of the Equation 7-1 on FPGA by implementing a simple approximation technique like the well-known digital differential analyzer (DDA). A DDA, also sometimes called “digital integrating computer”, is a digital implementation of the differential analyzer. The integrators in DDA are implemented as accumulators, whereby the numeric results are converted back to a pulse rate by the overflow of the accumulator. The main advantage of the digital integrator, when compared to an analog integrator, is the scalable precision. Also, in a digital integrator based on DDA, we don't have drift errors and noise due to the imperfection of electronic components. By accumulating over time of the values in a register, we can calculate the integral of input signals. The basic digital integrator is expressed by Equation 7-4.

(7-4)

$$X_{n+1} = X_n + K \cdot S$$

In Equation 7-4, $X_{n+1}$ denotes the next state of the accumulator used for calculating the integral. The coefficient of $K$ is a constant factor that is less than 1; it is used for time-scaling. In this equation, $S$ denotes the input signal for integration. We can map this technique on FPGA very easily by writing a behavioral code. After each rising clock pulse, the equation updates the integral value. In this integrator, rounding or truncation errors are only due to the limitation of registers. Therefore, by increasing the register sizes we have a way to control/reduce this error. This error is cumulative. Thus, for low precision registers a lack of accuracy will be observed after a long time. The only way for overcoming to this problem is setting proper register sizes.

By this way, we can compute directly the solution of differential equations. This simulation of analog computing is a fully parallel method for solving differential systems such as nonlinear equations and also to realize the integrator module within CNN. For integrator modules we must have access to the memory for storing the values and the cells output. The only critical term in CNN equation is the "Integrator", which we implement/realize through the DDA model. After approximating the basic CNN cell, we must cascade the cells together. All these steps are implemented in the CoDeveloper by using a Fixed-Point method. The result for each three rows will be stored in the memory. The CoDeveloper can handle the access to the external memory through the multi port memory controller (MPMC). This controller is a full feature memory controller that is compatible with standard DDR2 memory devices. This controller must be configured for at least one read and one write port. And for the many high-end video processing applications, there is no implicit limit on the number of read or writes ports in MPMC.

## 7.8 CNN Emulation Architecture

According to the original model of CNN, it consists of two parts, one is constant and the other part is variable over the time. A Convolution is realized between the control templates and the input. The bias value however is always constant during the CNN calculations $T_B * u_{ij} + I$. Therefore, the system doesn't need to recalculate it during the CNN solving process. The only part that must be recalculated in the DDA iterations is the term $T_A * y_{ij}$. The value of the "Template Convolution" added to the bias is constant for each pixel and it is therefore not recalculated again during the CNN's iterations; see Refs [18, 123].

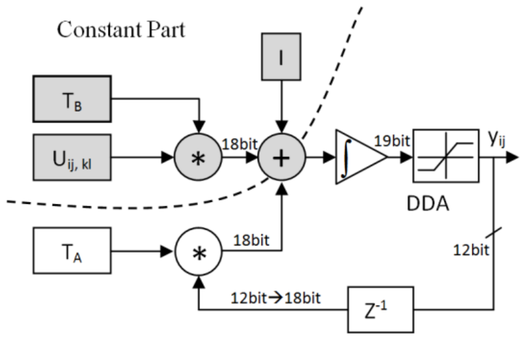


**Figure 7-5: Scheme of the DDA based model for CNN**

For storing the matrix template values and the convolution we have assigned a 14 bit signed register for each cell, 1bit sign, 3bit integer and 10bit precision. Similar to the standard analog CNN chip, we have limited the range of values to be between -5 and +5. We need an 18bit register to keep a convolution result. By 1bit sign, 7bit integer and 10bit precision it can store the values from -127 up to +127 with good accuracy. The maximum value for the convolution is in the range of (-125, +125]. If we define a Bias value in the range of (-3, +3], we can still save the value in same range register (18bit).

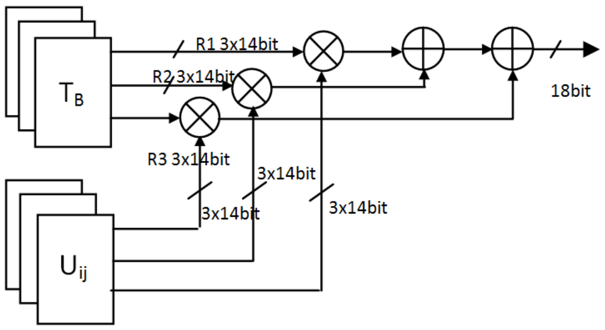


**Figure 7-6: Convolution diagram for the control template**

Figure 7-6, shows the Control Template and convolution diagram, which is the same also for the Feedback template convolution with the output signals. Signals are, after the saturation function, in the range of [-1, +1]; thus we must extend it from 12bit to 14bit range before getting convolution on feedback template.

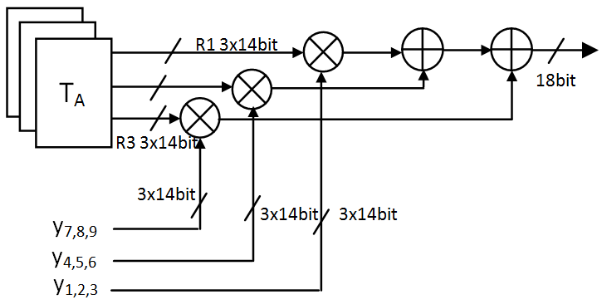


**Figure 7-7: Convolution diagram for feedback template**

According to the Equation 7-4, for implementing an integrator we need a register to accumulate the input value with the previous value. In Equation 7-4 we assign the value $2^{-9}$ to the 'K' factor.

$$x(n) = x(n-1) + (T_A * y + T_B * u + I)^{2^{-9}} \tag{7-5}$$

To apply the 'K' factor we can operate a 9 time shift to right. Depending on the 'K' factor we can increase the accuracy. There is a trade-off between accuracy and speed. For a small 'K' factor, we need more iteration to get the result. On the other hand by this way we can change the time scale.

## 7.9 Hardware and software integration

Hardware modules are significantly faster than software functions. But in some cases we need more flexibility in our system to access to the data and resources. The only way to get a high performance is hardware and software integration. Using a high bandwidth communication path between hardware modules and processors can be very helpful. For example, for changing the template in the convolution and as well in CNN we need a mechanism for loading data and storing it in the internal registers. For convolution based video pre-processing we can just load the matrix coefficient that can change the functionality of the system, and in the case of CNN we must change two different templates and bias values. We know that the Impulse C from CoDeveloper is based on the Communication Sequential Processor programming model. And it can generate the entire necessary internal signal for synchronization and communication between hardware modules. But for communication and exchanging values between hardware modules and processor, there are some methods such as Processor Local Bus (PLB) and Fast and Simplest Link (FSL). If we need only communication between two given modules, FSL is the best, but for communication between a master and many slave modules, PLB can be helpful. For Video and Image processing we do not need any high processing over the processor, and the only reason behind that is initializing, re-configuring and debugging the system; see Refs [124-126].

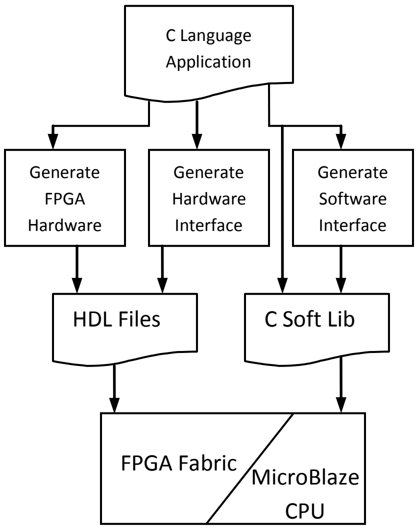


**Figure 7-8: Impulse CoDeveloper Design Process Diagram**

Imuplse C can generate hardware and software interfaces; depending on the architecture, it can be PLB based or FSL based. We have access to driver and low level code for communication between processor (MicroBlaze/PowerPC), convolution module and CNN module. We don’t need to store the pixels values of the whole image, and just after convolution calculations for every 3 rows, we can send it to the video output. In CNN, we have to keep those pixels values in a buffer for the next n iterations. The number of n is depending on the accuracy and on the DDA approximation. For this work we have used a 32bit MicroBlaze CPU for a combination of both hardware and software; hardware convolution modules must connect to the memory and data path according to the Figure 7-1 architecture. All the parameters and data can be transmitted between modules and memory, which is based on FIFO stream buffers and *Processor Local Bus* (PLB) technique; see Refs [127, 128].

## 7.10 Direct convolution vs. CNN – performance comparison

We have found by experience that in our design image smoothing operators, whether linear or non-linear, such as uniform filter, median filter, Gaussian filter and so on, which is easy to model by a direct convolution, are faster than a CNN based processing for the same. The results of first order and second order derivatives for finding edging in direct convolution was also faster than CNN implementation. The direct convolution method needed only two clock pulses for each pixel, while CNN needed 10 clocks (it depends on the number of DDA iterations).

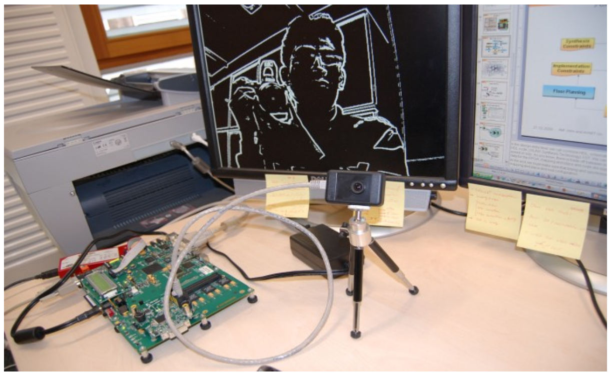


**Figure 7-9: Real-time output of the system on monitor**

In the case of morphological operations such as dilation, erosion, opening and closing operation, the CNN model has been found to be much faster, and it is not even possible to implement these operations only by cascading convolution operators and some Boolean arithmetic operations. For the complex algorithms like "skeleton", CNN has clearly a very good potential and it is easy to find a template to extract the image skeleton; see Refs [18, 129] [93, 96, 130].

## 7.11 Conclusion

In this chapter, a new method for CNN emulation on FPGA for real time machine vision applications has been proposed. The system is implemented on Xilinx XtremeDSP kit 3400, which is very flexible for video and image processing applications. The video is tested on DVI and Camera in connection with 1024×768 pixels and 60 fps monochrome for direct convolution and near 24 fps CNN emulation. The maximum frequency of the system is 200MHZ. The Computation of pixel operations for convolution method does only take two clock pulses. And for the CNN implementation the number of clock pulses needed for a give processing did depend on the accuracy needed to repeat the DDA iterations. The whole design has been made by ISE 11.2, EDK 11.2 and Impulse CoDeveloper 6.3. The main features of the convolution technique is that we don't need to access to the external memory and the frame rate is high. That's why it takes only two clock pulses. For implementing complex filters such as nonlinear image processing, CNN-based processing is clearly much better than convolution operations. To achieve the same result by cascading some convolution operators would result is a significantly slower process than using only one CNN operation processing.

# Chapter 8

## 8. Implementation of CNN on GPU

In this chapter the focus lies on the following research question: *"Can it be demonstrated by a series of concrete image processing examples of relevance for ADAS, that CNN based processing does really meet the hard requirements related to speed and robustness?"*

Every millisecond during driving is important. This could be a minimum interval for processing signals in vehicles. In some cases such as Lane Departure Warning (LDW), Adaptive Cruise Control (ACC), Emergency Brake Assist (EBA) and Blind Spot Detection (BSD) system should take a decision in a few milliseconds. Main bottleneck for video based ADAS is preprocessing image and preparation for extracting meaningful data from the image. DSP board could be a good alternative solution but due to complexity of image preprocessing, we do need a very robust and flexible platform which has potential of cascading function, sharing memory, soft reconfiguration and high speed processing. Hence, CNN architecture which implemented on FPGA or GPU could be more interesting for us. In ADAS the sequence of image preprocessing is clear for designer; therefore a flexible architecture which can reconfigure the system and change the functionality of the module by a small matrix template would be very interesting for them. High definition quality of image and performing complex filter in term of computation time, color space conversion, and extracting features in ADAS technology are hard requirements related to the speed and robustness.

## 8.1 CNN Based High Performance Computing for Real Time Image Processing on GPU

Many of the basic image processing tasks suffer from processing overhead to operate over the whole image. In real time applications the processing time is considered as a big obstacle for its implementations. A High Performance Computing (HPC) platform is necessary in order to solve this problem. The usage of hardware accelerator make the processing time low. In recent developments, the Graphics Processing Unit (GPU) is being used in many applications. Along with the hardware accelerator a proper choice of the computing algorithm makes it an added advantage for fast processing of images. The Cellular Neural Network (CNN) is a large-scale nonlinear analog circuit able to process signals in real time [12]. In this research, we develop a new design in evaluation of image processing algorithms on the massively parallel GPUs with CNN implementation using Open Computing Language (OpenCL) programming model. This implementation uses the Discrete Time CNN (DT-CNN) model which is derived from originally proposed CNN model. The inherent massive parallelism of CNN along with GPUs makes it an advantage for high performance computing platform [131]. The advantage of OpenCL makes the design to be portable on all the available graphics processing devices and multi core processors. Performance evaluation is done in terms of execution time with both device (i.e. GPU) and host (i.e. CPU).

## 8.2 Introduction

Image processing is an ever expanding and dynamic area with applications reaching out into everyday life such as in medicine, space exploration, surveillance, authentication, automated industry inspection and in many more areas [132]. Real time image processing using modern processors is limited [52]. Problems in computer vision are computationally intensive [133]. The tremendous amount of data required for image processing and computer vision applications present a significant problem for conventional microprocessors [52]. Consider a sequence of images at medium resolution (512×512 pixels) and standard frame rate (30 frames per second) in color (3 bytes per pixel). This represents a rate of almost 24 million bytes of data per second. A simple feature extraction algorithm may require thousands of basic operations per pixels, and a typical vision system requires significantly more complex computations.

As we can see, parallel computing is essential to solve such problems [133]. In fact, the need to speed up image processing computations brought parallel processing into computer vision domain. Most image processing algorithms are inherently parallel because they involve similar computations for all pixels in an image except in some special cases [133]. Conventional general-purpose machines cannot manage the distinctive I/O requirements of most image processing tasks; neither do they take advantage of the opportunity for parallel computation present in many vision related applications [121]. Many research efforts have shifted to *Commercial-Off-The-Shelf* (COTS) -based platforms in recent years, such as *Symmetric Multiprocessors* (SMP) or clusters of PCs. However, these approaches do not often deliver the highest level of performance due to many inherent disadvantages of the underlying sequential platforms and "the divergence problem". The recent advent of multi-million gate on the *Field Programmable Gate Array* (FPGAs) having richer embedded feature sets, such as plenty on –chip memory, DSP blocks and embedded hardware microprocessor IP cores, facilitates high performance, low power consumption and high density [134].

But, the development of dedicated processor is usually expensive and their limited availability restricts their widespread use and its complexity of design and implementation also makes the FPGA not preferable. However, in the last few years, the graphic cards with impressive performance are being introduced into the market for lower cost and flexibility of design makes it a better choice. Even though they have been initially released for the purpose of gaming, they also find the scientific applications where there is a great requirement of parallel processing. Along with the support of hardware platforms there are some software platforms available like *Compute Unified Device Architecture* (CUDA) and OpenCL for designing and developing parallel programs on GPU [135]. Out of these available software platforms OpenCL framework recently developed for writing programs can be executed across multicore heterogeneous platforms. For instance, it can be executed on multicore CPU's and GPU's and their combination. Usage of this framework also provides an advantage of the portability that is; the developed kernel is compatible with other devices. Along with the available hardware and software platforms we used the CNN parallel computing paradigm for some image processing applications.

The idea of CNN was taken from the architecture of artificial neural networks and cellular automata. In contrast to ordinary neural networks, CNN has the property of local connectivity. The weights of the cells are established by the parameters called the template. The functionality of the CNN is dependent on the template. So with a single common computing model, by calculating the templates we can achieve the desired functionality. The CNN has been successfully used for various high-speed parallel signal processing applications such as image processing, visual computing and pattern recognition as well as computer vision [91]. So we thought of implementing it on the hardware for the need of HPC in real time image processing. Also, the parallel processing capability of the CNN makes us to implement the CNN architecture on the hardware platform for its efficient visualization.

In this research, the effort is done to develop a DT-CNN model on the graphics processing units with the OpenCL framework. An effort is done to make the development of DTCNN entirely on the kernel which make it executable on every platform. But, it should be noticed that the GPU is a coprocessor which supports the processor in our system. Hence, the CPU still executes several tasks, like the transmission of the data to the local memory of the graphics card and retrieving back. Finally, GPU-based *Universal Machine - CNN* (UM-CNN) was implemented using the OpenCL framework on NVIDIA GPU. A benchmark is provided with the usage of GPU based CNN model for the image processing in comparison with CPU. The chapter is structured as follows: Section II gives a clear description about the theory involved in parallel computing. Section III introduces the concepts of CNN, the system diagram and its functionality and systems designed methodology which is done using OpenCL. Section IV concludes the section and says about the work going to be done in the future.

## 8.3 Theory of Parallel Computing

Traditionally, computer software has been written for the serial computation and time sharing computation. Therefore to solve a problem, an algorithm is constructed which produces a serial stream of instructions. These produced instructions are executed sequentially one after the other on the CPU of the computer.

Parallel computing on the other hand uses multiple processing elements simultaneously on a problem. The problem is broken into parts which are independent so that each processing element can execute its part of the algorithm simultaneously with others. The processing elements can be diverse and include resources such as a single computer with multiple processors, a number of networked computers, specialized hardware or any combination of any of the above. The software speedup was achieved by using a CPU with higher clock rates, which significantly increased each passing year. However, when clock speed reached 4GHz, the increase in power consumption and heat dissipation formed what is known as the “Power Wall” which effectively caused the CPU clock rate to level off [136]. Along with this, many applications today require more computing power than a traditional sequential computer can offer. All these things made the vendors search for an alternative to make increase of available cores with in a processor instead of increasing the clock rates. The increase of cores on the processor made the CPU clock rates to remain same or even reduced to economize the power usage. The old software design used for the sequential process will not get increased directly with the increase in the processor cores. To get the benefit form the current developed multi core processor, new software has to be designed to take the advantages of the new architecture. This makes the use of all available multi cores and performs process in parallel. Parallel computing is defined as “a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently (in parallel)".

Many different hardware architectures exist today to perform a single task using multiple processors. Some examples are: Grid computing – a combination of computer resources from multiple administrative domains applied to a common task. *Massively Parallel Processor* (MPP) systems – known as the supercomputer architecture Cluster server system – a network of general-purpose computers. *Symmetric Multiprocessing* (SMP) system – identical processors (in powers of 2) connected together to act as one unit. Multi-core processor – a single chip with numerous computing cores [136]. Heterogeneous computing systems also provide an opportunity to dramatically increase the performance of parallel and HPC applications on clusters with CPU and GPU architecture [137]. This concept can be achieved by combining the GPU and multicore CPUs.

## 8.4 System Design and Architecture of CNN

This section explains in detail about the CNN, its architecture and advantages. It is followed by the description of the system we have developed and about the OpenCL framework we have used and its advantages for programming on GPU. Analog circuits have played a very important role in the development of modern electronic technology. Even in our digital computer era, analog circuits still dominate such fields as communications, power, automatic control, audio and video electronics because of their real-time signal processing capabilities [104]. CNN technology is both a revolutionary concept and an experimentally proven new computing paradigm. Analogic cellular computers based on CNNs are set to change the way analog signals are processed and are paving the way to an entire new analog computing industry [138]. CNN was proposed by Chua and Yang in 1988 [104]. The CNN is defined as a n-dimensional array of cells that satisfies two properties: (i) most interactions are local within a finite radius r, and (ii) all state variables are continuous valued signals [94].The CNN has M by N processing unit circuit called cells C (i, j) located at site (i, j), i = 1, 2, . . ., M, j = 1, 2, . . ., N [98]. The array of CNN cell structure is as shown in Figure 8-1.

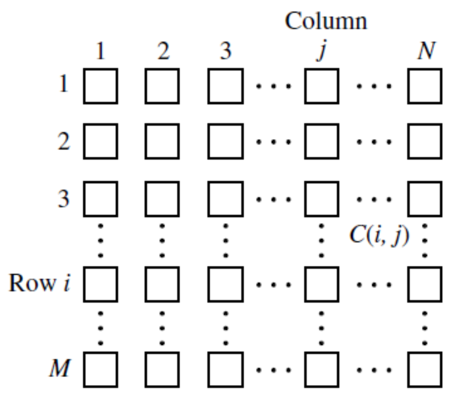


**Figure 8-1: A simple CNN array architecture**

Each cell of the CNN is made of a linear capacitor, a nonlinear voltage controlled current source and a few resistive linear circuit elements. The dynamic equation of a cell $C$ (i, j) in an M×N CNN, given by CHUA and Yang [139] is shown below

(8-1)

$$C\frac{dX_{ij}}{dt} = -\frac{1}{R}X_{ij} + \sum_{C(k,l)\in N_r(i,j)}(A_{ij,kl}\,Y_{kl} + B_{ij,kl}\,U_{kl}) + I$$

Where the output equation $Y_{ij}$ can be written as

(8-2)

$$Y_{ij} = f\big(X(i,j)\big) = \frac{1}{2}\,(|X+1| - |X-1|)$$

The mathematical equation mentioned in Equation 8-1 is representing the model of the Continuous Time CNN (CT-CNN). In the equation, $C$ is a linear capacitor and R is a resistor. $Y_{kl}$ is the output state of each cell. $U_{kl}$ is the input of each cell. $A_{ij}$ and $B_{ij}$ are the template elements. $X_{ij}$ represents the initial state and I represent the threshold or bias for each cell. The Equation 8-2 is the output equation of each iteration. This equation gives the functional model for the calculation of each pixel element to the output. This model in not very fast in the real time image processing. In order to overcome the drawbacks of CT-CNN, the concept of Discrete Time CNN (DT-CNN) is developed. The DT-CNN is defined by the difference equations instead of differential equations used in the CNN [140]. The model of DT-CNN is derived from the model of CT-CNN using the Euler's method. The DT-CNN can be described with the following equation [140].

(8-3)

$$X_{i,j}(t+1) \approx \sum_{c(k,l)\in N\,r(i,j)} A\,(i,j;k,l) f\left(X_{k,l}(t)\right) + \sum_{c(k,l)\in N\,r(i,j)} B\,(i,j;k,l)\;U_{k,l} + I$$

From Equation 8-3, we can see that $X_{i,j}$ is the state of the cell C(i, j) and $f(X_{k,l})$ is the output of cell C(k, l) within the neighborhood $N_{r(i,\,j)}$ of C(i, j). $U_{k,l}$ is the input of each cell C(k, l) within $N_{r(i,\,j)}$, and $I$ is the bias of cell. A and B are called the feed-back and feed-forward templates of the CNN respectively.

CNNs are widely used for real time image processing applications. Though the CNN, as a concept is characterized by a strict locality operation, the large scale digital implementation has been far from trivial [90].

## 8.5 System Diagram

The system diagram gives the clear understanding of the work we have done along with the flow of data and processing steps. The system diagram of the designed model of our research was show in Figure 8-2.

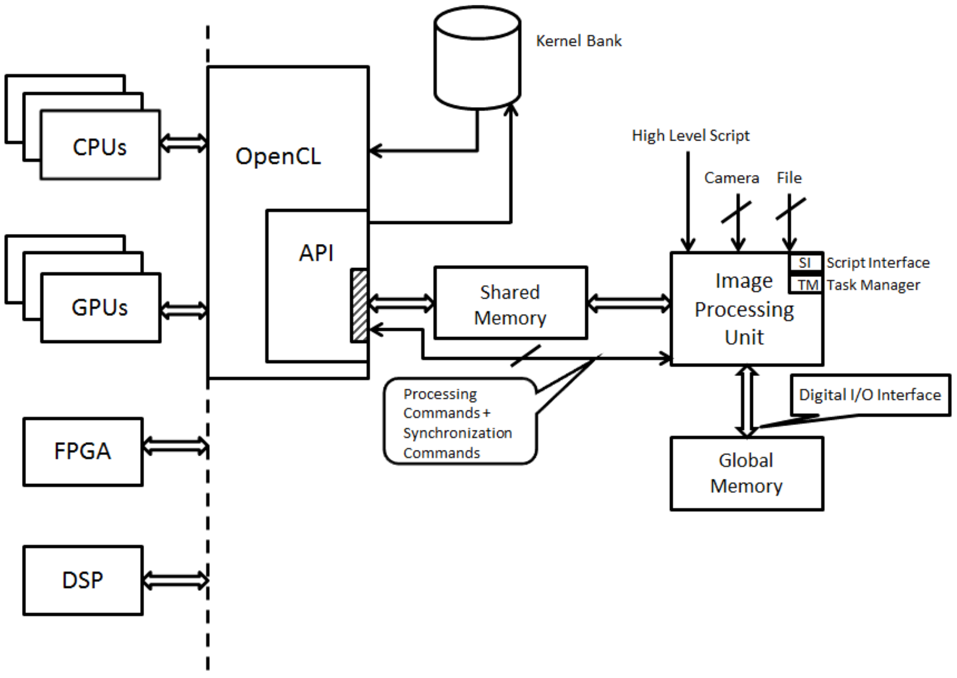


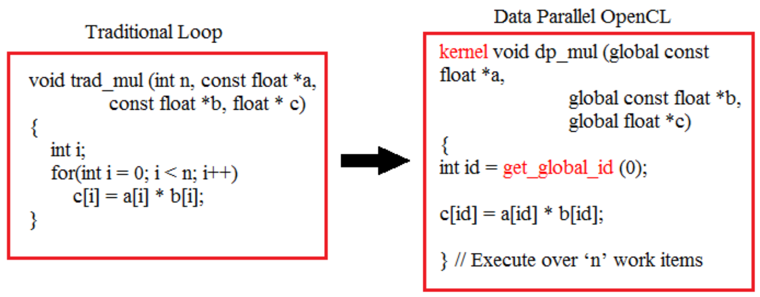
**Figure 8-2: System Design Architecture**

From the system diagram shown above we can understand that there are several available platforms which can help in the processing of image processing tasks. They are FPGAs, DSPs, CPUs and GPUs. Out of which the OpenCL framework is capable of developing these image processing algorithms on the multi core CPUs and on GPU or on cluster of GPUs. So we have chosen this OpenCL framework as a programming language for our task. This OpenCL framework has an *Application Programming Interface* (API) which helps the processors either CPU or GPU to communicate with the shared memory and image processing unit. This helps in acting as an interface to both of them. The Image processing unit has access to the global memory as a temporary buffer for the processing of images. There is a high level script interpreter for task management and accessing to I/O's and

digital channels such as files, Camera, etc.  The data which is to be processed is kept in the shared memory and is accessible by the OpenCL framework and the Image processing unit. After loading the instructions from the image processing unit, the API of the OpenCL will try to take the required kernel from the kernel bank, for the process of the instructions.
After getting the required kernels from the kernel bank the OpenCL make these kernels to be executed on the necessary processor and then the results are again stored in the shared memory which is collected by the image processing unit. The OpenCL API also checks the processing commands and the synchronization commands from the image processing unit in order to perform a correct operation on the proper image.

## 8.6 Methodology of system design using OpenCL

OpenCL is “a framework suited for parallel programming of heterogeneous systems”. The framework includes the OpenCL C language as well as the compiler and the runtime environment required to run the code written in OpenCL-C. OpenCL is standardized by the Khronos Group, who is known for their management of the OpenGL specification [136].
OpenCL is a framework which has been developed in order to program on any heterogeneous environment. Therefore we can say that the OpenCL code is capable of being executed on any computer. OpenCL provides standardized APIs that perform tasks such as vectorized Single *Instruction Multiple Data* (SIMD) operations, data parallel processing, task parallel processing and memory transfer. A simple functional execution block written in OpenCL is called as a kernel. The OpenCL kernel is written in its native language using its own API. The benefits of OpenCL are: it allows acceleration in parallel processing, it manages the computational resources, it views multi-core CPUs, GPUs etc. as computational units, allocates different levels of memory, it also supports cross-vendor software portability.  A simple kernel written in OpenCL in comparison with the traditional loop coding is seen the following Figure 8-3.



```
void trad_mul (int n, const float *a,
           const float *b, float * c)
{
  int i;
  for(int i = 0; i < n; i++)
     c[i] = a[i] * b[i];
}
```

```
kernel void dp_mul (global const
float *a,
                 global const float *b,
                 global float *c)
{
int id = get_global_id (0);

c[id] = a[id] * b[id];

} // Execute over 'n' work items
```

**Figure 8-3: A comparison traditional loop with the OpenCL data parallel kernel**

The Figure 8-3 gives the advantages of writing the code in OpenCL as well as the parallel computing capability of the OpenCL kernel. From Figure 8-4 we can clearly understand the steps involved in the OpenCL programming model. The steps show the structural design and execution of a kernel. Whenever the kernel is designed, these steps are followed in order to execute the kernel on the appropriate device.

Here in our work, we have used the OpenCL in order to develop the UM–CNN which is to be executed on the GPU. The required kernel which is necessary for the execution of the CNN is written using the OpenCL programming model. The interface of the data is done from the CPU which is to be loaded onto the device. For proper interface of the communication of data we have used the *Open Computer Vision Library* (OpenCV) for the reading of images, loading of the image data on to the memory elements and to display the retrieved image back from the GPU after its execution.

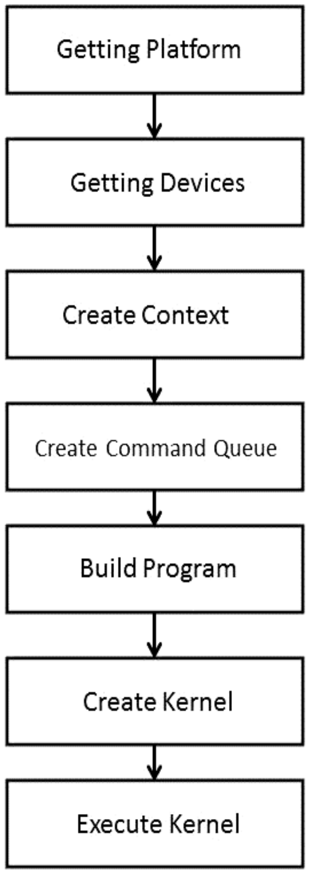


**Figure 8-4: OpenCL Programming Flow**

In our work we have used the combination of OpenCV for the appropriate user interface and for data acquisition, and OpenCL for the development of the desired UM-CNN on the heterogeneous platform.
For real time image processing applications the performance of CNN is evaluated for the contrast enhancement. The templates used for the contrast enhancement are given below

(8-7)

$$T_A = \begin{bmatrix} 0 & 0.25 & 0 \\ 0.25 & 0 & 0.25 \\ 0 & 0.25 & 0 \end{bmatrix};\ T_B = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}; I = 0$$

Figure 8-5 shows the given input image and output image of the CNN operation on GPU and Fig 8-6 shows the output image of the CNN operation on CPU.

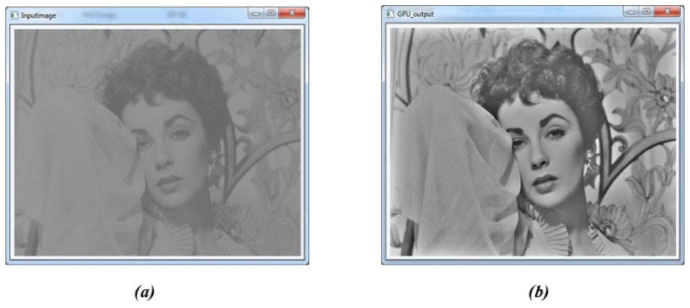


**Figure 8-5: (a) Input image for CNN (b) Output image of CNN on GPU**

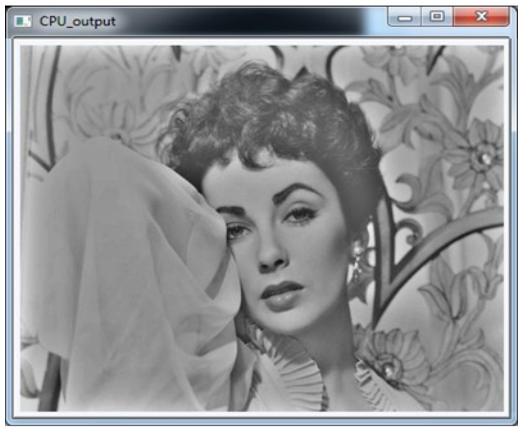


**Figure 8-6: Output of CNN on CPU after applying enhancement template**

## 8.7 Conclusion

In this work we propose a new model for image processing that can be executed in parallel and faster. We implement this model with OpenCL framework along with OpenCV. The concept of DT-CNN on the GPU makes the execution of the image processing task faster. The concept of high performance computing is achieved with the designed model. Till now the discussion goes on with the surrounding knowledge of the designed model. In the future work, a clear description of the entire model along with the benchmarks of comparing the performance of GPU with performance of CPU will be done.

Chapter 9

# 9. Cellular Neural Networks for Controlling an Unstructured Robot

In this chapter the focus lies on the following research question: *"How far can CNN be used/involved in an evolutionary computing/control context example?"*

## 9.1 Introduction

CNN has a great potential in signal processing, and it can generate very complex nonlinear wave and osculation pattern in the output of Cells. Controlling kinematic and inverse-kinematic of complex robots with *High Degree of Freedom* (DOF) could be very complex scenario and classical solutions are not able to solve it easily. Therefore evolution of CNN template to generate the optimum wave for driving motor and robot actuators could be an interesting idea for research. In the nature organisms system are evolving by the theory of natural selection. The art is to define a fitness function for evaluating of the performance of robot locomotion. Hence, by evolving the CNN template based on Genetic algorithm and a fitness function we can generate the very complex wave for optimal controlling the robot hinges without involving the robot kinematic equation in controller directly. A two dimensional CNN could be sufficient for evolving spatial wave over the time for controlling the actuators and hinges. Figure 9-1 has shown this connectivity between CNN and robot actuators.

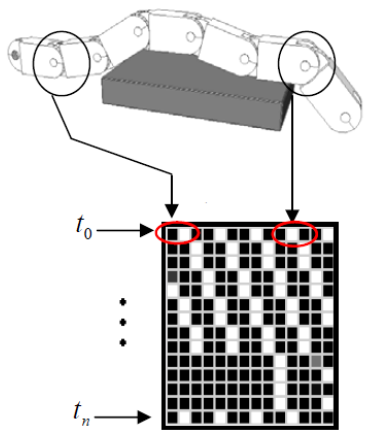


**Figure 9-1: Shows the spatial wave and time domain on a CNN, which connected to the robot actuators**

A fitness function is a particular type of objective function that quantifies the optimality of a solution. Input data for the fitness function is based on measurements of robot parts orientation, location and displacement. In fitness function we don't define any behavioral locomotion exactly. On the other hand, we define a function that satisfies the target or destination. With this method based on genetic algorithms, an optimum template ensures that the robot can move or act according to our desires. The most important point in this learning method is that we don't predefine any robot kinematics for movement in the fitness function.

## 9.2 Cellular Neural Networks-Based Genetic Algorithm for Optimizing the Behavior of an Unstructured Robot

A new learning algorithm for advanced robot locomotion is presented in this chapter. This method involves both *Cellular Neural Networks* (CNN) technology and an evolutionary process based on *Genetic Algorithm* (GA) for a learning process. Learning is formulated as an optimization problem. CNN Templates are derived by GA after an optimization process. Through these templates the CNN computation platform generates a specific wave leading to the best motion of a walker robot. It is demonstrated that due to the new method presented in this chapter an irregular and even a disjointed walker robot can successfully move with the highest performance.

## 9.3 Introduction

Nowadays, some of the main goals of robotics science, mechatronics and artificial intelligence lie in designing mechanisms close to or mimicking as good as possible some natural structures or animal behavioral models. According to this theory, the nature selects the powerful and stable genes for breed, and weak genes fall/disappear in the nature [141]. The good genes that can adapt the animal structure to the environment have higher chances for breed and evolution. The animal locomotion is trained and adapted according to the animal's body structure. One key issue in the training process is based on the energy saving. This justifies the striking interest devoted to the modeling and simulation of animal walking motion with the aim of optimizing the energy consumption [142-144]. It is well-known that the walking motion of animals is of a stereotype. In a large variety of animals a central neural controller does organize/coordinate the motion. A central neural controller (e.g. the *Central Pattern Generator* (CPG)) is a main unit for controlling limbs for walking [145]. The CPG unit does contain all the mechanisms needed to generate the rhythmic pattern of movement. This unit is suitable for designing walker, swimmer, or flyer robots which exhibit motion close to natural locomotion mechanisms. Due to recent advances in electronics and the ability of cellular neural networks to solve partial differential equations in real time, it is possible to simulate a Reaction-Diffusion model by a specific CNN architecture, the so-called *Reaction-Diffusion Cellular Neural Network* (RD-CNN).

A striking interest has been devoted to the robot control based on the RD-CNN technique [141, 145, 146]. In this technique, the mathematical model describing the robot behavior must be well-defined. This is a serious limitation as modeling the complex behavior of robots is challenging. In this chapter we introduce a robots control method which is not based on the mathematical modeling of the robots behavior. It is rather a general and effective method combining CNN with GA. This method can support and drive many types of structured and unstructured walker robots. The method/approach is based on both the natural modeling and the use of computational units close to biological models. A combination of both CNN (i.e. for computation) and GA (i.e. for optimizing the nature) is a good tool for modeling and controlling robots dynamics. The central parts of this scheme are made-up of a CNN processor and an evolutionary training unit. A *Cellular Neural Network* (CNN) is a parallel computing paradigm similar to the artificial neural networks computation platform, with the difference that in CNN the communication is allowed between neighboring units. This feature of the CNN processor makes it a good computation platform to analyze the dynamics of biological neurons. This research shows the possibility of directly driving a walker robot by an evolutionary training of a CNN processor. This method is further efficient to model widespread natural locomotion mechanisms of animals (e.g. worms, insects, quadrupeds, biped, etc) [104]. This locomotion is modeled in the 3D space describing the real environment and in very difficult situations (i.e. rough, bumpy, and/or scaly surfaces) as well. The challenging focus is finding the best signal for driving walker robot joints with minimum energy consumption and the best locomotion performance. This can be achieved by finding suitable CNN templates to generate an efficient wave for driving the walker robot joints. This chapter is organized as follows. Section 9.4 discusses the use of genetic algorithms for optimizing the CNN templates. Section 9.5 presents the training algorithm and some simulation results as well. Section 9.6 formulates some concluding remarks. Further, the quintessence of the results obtained is summarized, and some open research questions are outlined.

## 9.4 Using genetic algorithms for CNN template optimization

The concept of *Cellular Neural Networks* (CNN) was introduced by Leon O. Chua and Yang [104]. CNN is a computation platform which is mathematically modeled by Equation 9-1

$$\dot{x} = -x + T_B * u + T_A * y + I \tag{9-1}$$

where, *'TA'* denotes the 3×3 feedback template and *'TB'* stands for the 3×3 control template. *'I'* is a bias value and *'y'* is the nonlinear output sigmoid function of each cell. *'u'* denotes the input value and *'x'* is the state of each cell. The input value is discretized into pixels and is represented in a table of numbers called matrix. The size of this matrix depends upon the number of joints in the walker robot. In Equation9-1, the stars stand for convolution operations.

The genetic algorithm is a heuristic search technique used in computing to find either exact or approximate solutions for optimizing a given problem. The GA is an evolutionary algorithm that uses techniques inspired from biology such as inheritance, mutation, selection, and crossover. In this paper, this algorithm is used for finding the best templates for optimum robots locomotion.  The complete structure of the system used for the training process is shown in Figure 9-2. This structure consists of six main parts: (1) Initial Population; (2) Crossover; (3) Mutation; (4) Fitness Function; (5) Decoding; (6) Cellular Neural Network Simulator. In Figure 9-3 the connections between the robot actuators/hinges and the CNN outputs are shown. These connections are exploited in the control of both robot hinges and actuators. Wave rhythms are generated from the CNN processor outputs which can drive the walker robot on a specific path and/or direction depending on the high level task each of which consists of many low level tasks. After the learning phase, the output waves can drive the robot with a minimum energy and a good efficiency. This driving depends upon specific choices of templates values. Each template set is a solution for driving the robot by means of (or by performing) some specific low level tasks.

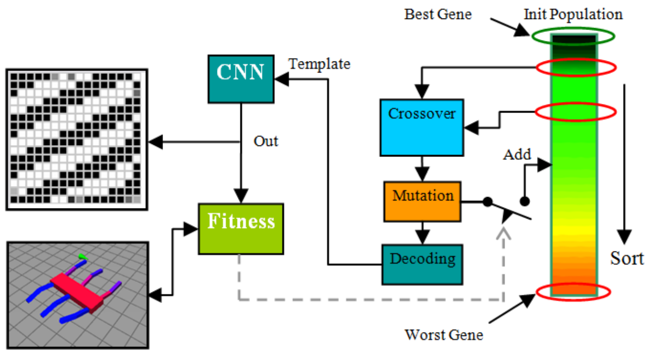


**Figure 9-2: System Architecture Diagram**

For optimizing these solutions, the templates must be coded as chromosomes as shown in Figure 9-4. In the initialization phase, Figure 9-4 generates many random chromosomes (in this case CNN templates), each being a solution for driving the robot. In fact, each chromosome is a CNN template that is reshaped in a one dimensional array. According to Figure 9-4, each chromosome does contain a feedback template, a control template and a bias value. Various methods exist (in genetic algorithms) for coding data as chromosomes. This paper implements two different methods for coding and generating chromosomes. The first method is based on the IEEE-754 scheme which is a floating point technique. In this technique, each value must be converted to binary format according to the IEEE-754 floating point technique. The IEEE floating point format consists of three main parts: the sign, the exponent, and the mantissa [147]. The number of bits for each field is shown in the table below.

Table 9-1: Single Precision - IEEE Floating Point Format Structure

| Sign | Exponent | Mantissa |
|---|---|---|
| 1 bit | 8 bit | 23 bit |

With the floating data types mentioned in Table 1, it is possible to store values between the ranges $\left[1.5\times10^{-45},\ 3.4\times10^{38}\right]$. The use of this method as a gene coder requires the definition of a mask for some bits. Otherwise, the random chromosome generator will generate values out of the range $\left[-5V,\ \ +5V\right]$. This condition is of high importance as a hardware implementation (using TTL devices) of this algorithm is under consideration. In the second method implemented, a “real” data type value is used as a chromosome coding. For this step, a random function generates a value in the acceptable range. The implementation of this method is easier than of the first method. The results from the two methods are compared and a very good similarity is obtained between them. Nevertheless, the convergence time in binary coding was 10 percent faster. One particular important part of this algorithm is the design of the fitness function. This function or cost function defines/fixes indirectly the robot behavior [148]. This function is a particular type of objective function that quantifies the optimality of a solution in Genetic Algorithms. The input data for the fitness function are based on measurements of robots’ parts orientation, location and displacement. In the fitness function we don't define any behavioral locomotion exactly, like a robot kinematics. On the other hand, we define a function that satisfies the target or destination without any details.

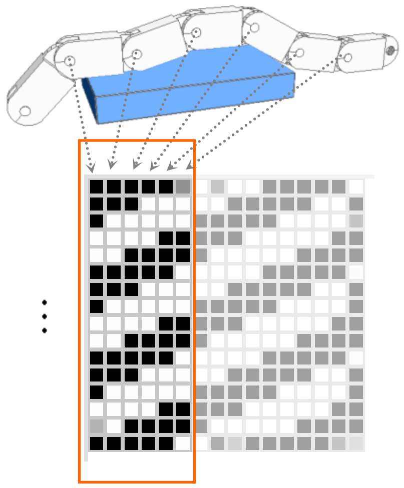


**Figure 9-3: Robot hinges connection to CNN array**

$$T_A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}, T_B = \begin{bmatrix} j & k & l \\ m & n & o \\ p & q & r \end{bmatrix}, I = s$$

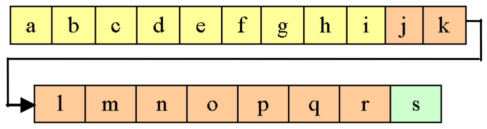


**Figure 9-4: Template Encoding in an Array List**

We need the robot to escape from a position without any specific direction. For this purpose, we must define a simple function for measuring the length between the central point (i.e. the gravity center) of the robot and the initial position. In this example, we didn't define any detail for the locomotion behavior. After generating new chromosomes, we obtain the corresponding fitness value by applying the fitness function. The main point on applying the fitness function is that this function is not a real time procedure and that the result from the fitness calculation will be only ready after a certain period of time beyond the time the wave effect will act on robot hinges/actuators. In fact, one cycle of time (i.e. one period of the wave acting on the actuator) is not sufficient for measuring with good accuracy the position in space of the robot. Many cycles of the wave generated are necessary to be applied to robot actuators. By measuring some robot parameters like the position of the robot central point, the robot angle (related to the global coordinates of the system) and so on, the fitness function is quantified. In the initial state of the training phase the algorithm selects some randomly generated chromosomes. There is no rule for evaluating how many chromosomes should be generated in the initial population. This number varies depending upon the complexity of the problem [149]. Some authors have defined 100 generations of chromosomes/genes for the initial state [149, 150]. After each generation, a fitness function is used to evaluate the cost of chromosomes in the simulator leading to maximum efficiency. During our computations, each evaluation took approx. 3 seconds and the program spent approx. 60 seconds to evaluate appropriated chromosomes. After this step, both chromosomes and fitness values will be sorted with the aim/goal of minimizing the fitness values in a link list. The next step concerns the crossover (i.e. both selection and breed) of chromosomes. Our experiments have shown that 50% of the best chromosomes are fitting for the crossover. It was found that this range has a good probability to generating the better chromosomes. In each step, we randomly select 2 chromosomes in this range for the crossover process. Many evaluations have shown that the use of the “two-point” technique for the crossover is the best solution. In this process we define two points (randomly) on the selected chromosomes; the contents of the chromosomes between these two points are exchanged (Figure 9-5). In Figure 9-5 P1 and P2 are two randomly selected points. S1 and S2 are two selected parents/chromosomes. The crossover leads to two new “children” (see Ch1 and Ch2 in Figure 9-5) with new properties. During the trial and error process, we obtained that the good probability for mutation is around 10%. This rate is essential for avoiding the local minimum trap. In the long term, this rate of the mutation increases the quality of chromosomes in the list [151].

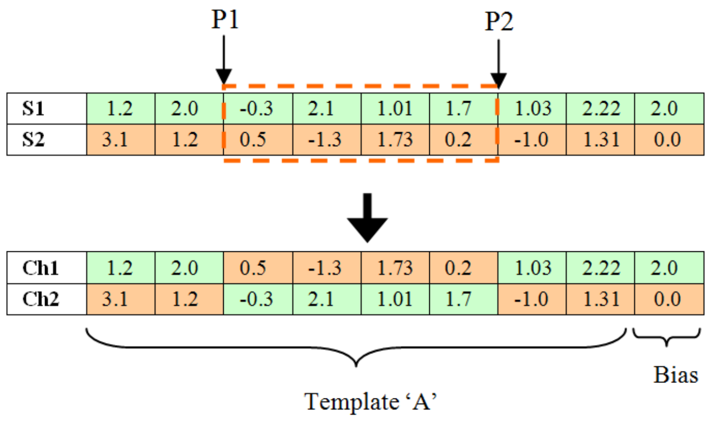


**Figure 9-5: Two-Point Crossover Method for Template 'A'**

## 9.2 Training algorithm and simulation results

One of the most important parts of this research is simulating both robot and environment. Some authors have implemented the robot and a virtual world by simulation of dynamic rigid bodies [152, 153]. The robot which depends on the physical parameters is implemented in a specific environment. Each part of the robot has a mass, a center of mass, an elasticity parameter, and both dynamic and static friction coefficients. Figure 9-6 shows the implementation of a "snake robot" made up of joints with 2 degrees of freedom. The hinges do not have any limitation in rotation. Nevertheless, applying limitation in the rotation range is possible for each joint separately. According to Figure 9-3, each column of the CNN processor is connected to robot actuators. Hence, each actuator of the robot should be connected to one of the columns separately. Since the robot actuators' response time is not equal for all of them we do assume/take the maximum delay for sending the wave on the robot actuators. This delay interval is essential for the robot locomotion/movement. The goal of the learning process is finding optimum templates for moving the robot according to our desires. Finding these templates for a specific movement mechanism/pattern is essential and suitable for the use in a multi layer tasks manager or controlling unit. We are able to use these templates for a low level robotics activity. When a high level controller sends commands to the robot for performing a specific task another controller needs to manage some low level skills like running, turning, jumping and so on, which are necessary to ensure the realization of the high level task [142, 143, 154, 155]. Therefore, by understanding some robot properties the high level task management is very simple in the high level controller. In the above referenced evaluation, authors have tried to find lateral undulation locomotion for a snake robot. Each hinge has two degree of freedom (2-DOF) and can turn in 2 directions. With the method based on genetic algorithms, an optimum template is obtained to make the robot moving or acting according to our desires. The most important point in this learning method is that we don't predefine any robot kinematics for movement/locomotion in the fitness function. The fitness function is a simple and important function which defines the robot behavior in the environment. Complicated rules and equations in the fitness function cannot improve the robot behavioral performance; a simple definition can result to a best robot behavior. Equation 9-2, define the fitness function used for the snake robot lateral undulation locomotion shown in Figure 9-6.

(9-2.a)

$$Fitness = RMS \times \frac{1}{AVG} \times DIST$$

(9-2.b)

$$RMS = \sqrt{\sum_{i=1}^{7} \left( AVG - L_i \right)^2}$$

(9-2.c)

$$AVG = \frac{\sum_{i=1}^{7} L_i}{7}$$

(9-2.d)

$$DIST = X_{L_1}$$

The term *'AVG'* denotes the mean distances between parts and the *'x'* axis. The term *'$L_i$'* denotes the distance between the *i'th* part of the snake robot and the *'x'* axis. RMS denotes the roots mean square error between the robot part's position and the *'x'* axis. $X_{L1}$ stands for the forward distance towards the *'x'* axis.

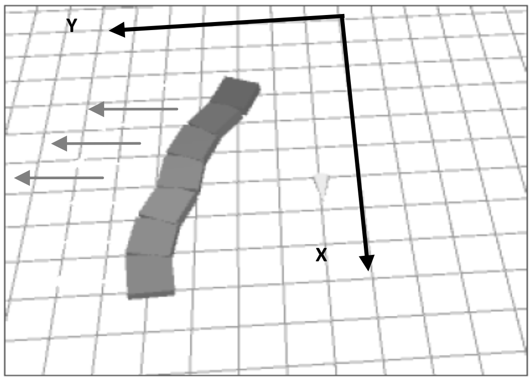


**Figure 9-6: Snake robot lateral undulation locomotion**

In the lateral undulation locomotion, this term of the fitness-function must be close to zero. This fitness-function defines the "snake robot" behavior for lateral undulation locomotion tasks. The first term *(RMS)* in the fitness function shows that the robot must keep itself in-line by moving parallel to the *'x'* axis. The second term *(AVG)* shows that the robot must escape from the *'x'* axis and the 3rd term *(DIST)* in the fitness function shows that the robot usually don't move in the frontal direction.

(9-3)

$$Ta = \begin{bmatrix} 0.8 & 0.01 & 1.79 \\ 3.44 & -2.85 & -4.89 \\ 4.33 & 2.18 & -4.17 \end{bmatrix}, Tb = \begin{bmatrix} -3.46 & -2.39 & -1.07 \\ 0.51 & 0.45 & -1.09 \\ 0.6 & -4.27 & -3.06 \end{bmatrix}, I = 3.29$$

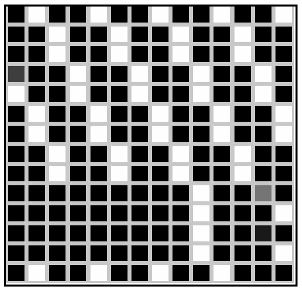


**Figure 9-7: Wave generated for lateral undulation locomotion**

After a first generation of 100 chromosomes, the robot learns to move in the lateral undulation with a corresponding set of CNN templates, which are obtained by the evolution method. These templates are shown in Equation 9-3. A task manager in the high level can select a best template for performing a specific task by the robot. On the other hand, each set of templates corresponds to a specific robot movement/locomotion. In another evaluation we define a fitness function according to Equation 9-4. This function is defined for robot rectilinear locomotion with a minimum sidle. According to this equation, each term must be close to zero. The first term *(RMS)* shows that the robot must have a minimum deviance to the *'x'* axis. The second term *(AVG)* shows that the robot should not be away from this axis. The last term *(DIST)* shows that the robot must crawl on the *'x'* axis. After each breed, a new chromosome is added to the chromosome population. After checking of new chromosomes by the fitness function, they will be sorted in a population list ordered by the best fitness. According to the evolution theory, after many generations, some chromosomes ("children") can inherit good properties from others ("parents") which are best and fit chromosomes.

After nearly 790 chromosome generations the robot would have learned to move with the highest speed. With Equation 9-5, the CNN processor can generate a hinge wave according to Figure 9-8. This wave is optimum for the robot rectilinear locomotion using an evolution algorithm. Figure 9-9 shows the robot during the simulation in rectilinear locomotion. Figure 9-10 is the plot of the time evolution of the fitness function obtained after 790 generation of chromosomes; the robot has learned the best movement and locomotion. The extension of this architecture or learning method to another kind of robot is possible. By connecting the CNN outputs to unknown/arbitrary robot actuators, the robot can learn any locomotion. Due to the high capacity of CNN, we can connect the CNN output to the robot hinges actuators by any arrangement and structure. The results are same although both learning and optimization times might change.

$$Fitness = RMS \times AVG \times \frac{1}{DIST} \tag{9-4}$$

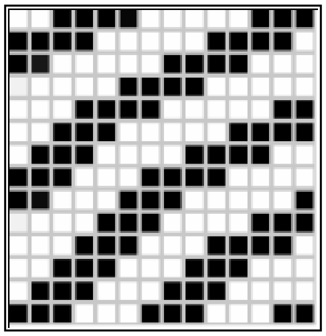


**Figure 9-8: Wave generated for rectilinear locomotion**

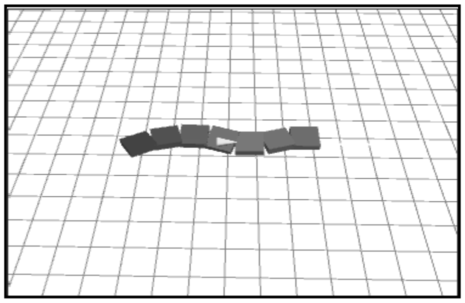


**Figure 9-9: Snake Robot rectilinear locomotion**

$$T_A = \begin{bmatrix} -2.67 & -2.74 & -2.01 \\ -4.62 & -4.15 & 1.69 \\ -0.92 & 2.1 & 0.58 \end{bmatrix}, T_B = \begin{bmatrix} 4.83 & 3.43 & -5 \\ -0.97 & -2.39 & 4.27 \\ -3.73 & 1.74 & -1.44 \end{bmatrix}, I = 3.05 \tag{9-5}$$

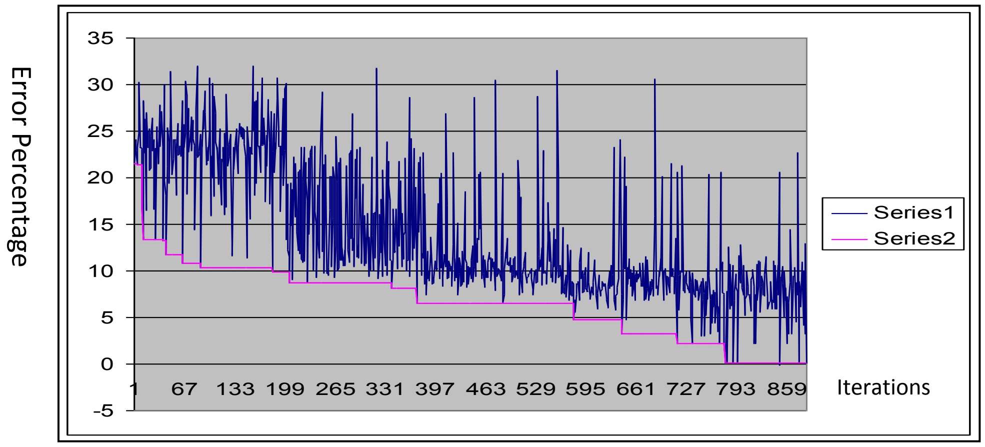


**Figure 9-10: Series 1 is fitness-function value; Series2 is fitness-function minimum value, during cycle of time in learning process. (Series1 is error rate; Series2 is number of itteration/time)**

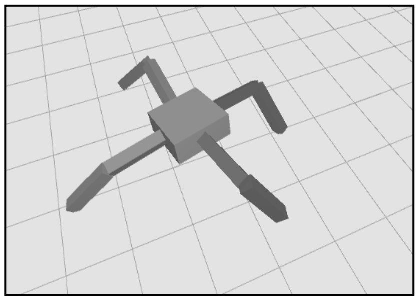


**Figure 9-11: Learning 4-legs semi-spider robot**

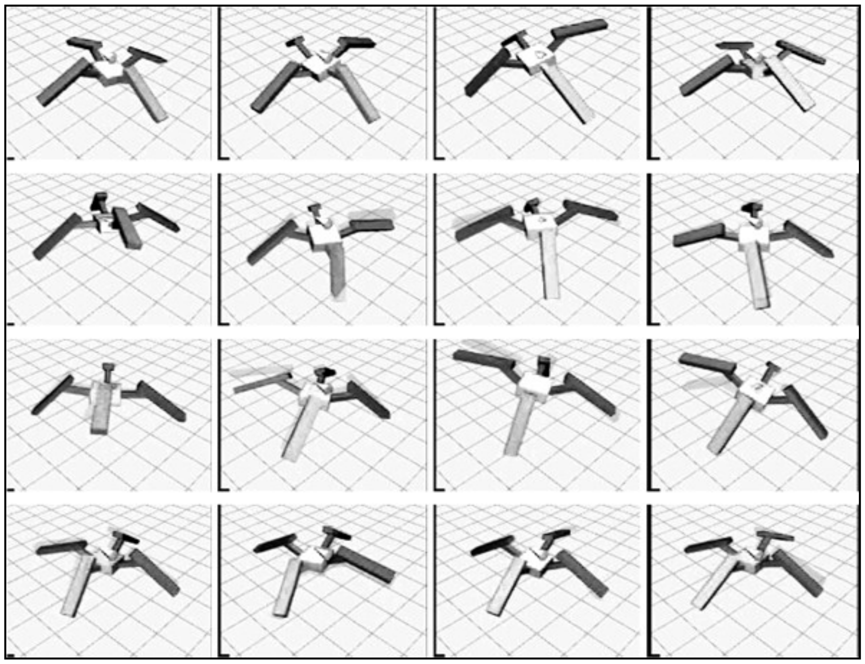


**Figure 9-12: 4-leg robot spider, turning skill**

Figure 9-11 shows a spider robot with 4 legs and 16 degrees of freedom. Each hinge has 2 degrees of freedom in rotation. A 16×16 CNN array can be used to drive this robot. Figure 9-12 shows the sequences of the robot locomotion after the learning process. In this test, the robot must turn around the ‘z’ axis. Another test in  Figure 9-11 shows the design of a 6 legs insect robot for locomotion learning. The robot has 12 degrees of freedom in hinges. In this case, the aim is moving around the circle with a given radius. After nearly 2500 iterations it was found that the result converged to zero. The CNN templates shown in Equation 9-6 are optimized for this purpose. The fitness function in Equation 9-7 is used to generate the CNN output wave shown in Figure 9-14.

(9-6)

$$T_A = \begin{bmatrix} 1.73 & -1.85 & 1.64 \\ 0.35 & -1.68 & -3.2 \\ -2.23 & -1.3 & 2.35 \end{bmatrix}, T_B = \begin{bmatrix} 4.03 & -3.05 & 3.52 \\ -3.2 & 2.61 & -2.49 \\ 4.88 & -3.02 & -0.95 \end{bmatrix}, I = -4.01$$

According to Equation 9-7, the fitness value has a direct relation with the distance between the initial position point (*Init_Center_Pos*) and the robot position (*Robot_Pos*) divided by (R). Further, this function has an inverse relation with the robot movement (Robot_ Movement).

$$\text{Fitness} = (1 - \text{Dist}(\text{Init\_Center\_Pos}, \text{Robot\_Pos}) / R) * (1/\text{Robot\_Movement}) \tag{9-7}$$

By optimizing the robot movement/locomotion, the fitness value will converge to zero. For the case of an unstructured robot, we have designed a broken-leg spider. In this test, the aim is learning the robot for the turning left and right skill as a complete and perfect spider. In Figure 9-15 is shown the representation of this type of robot.

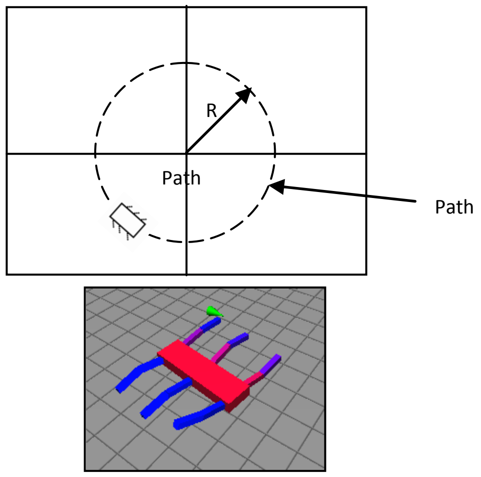


**Figure 9-13: Moving 6-Leg Robot, around the Circle**

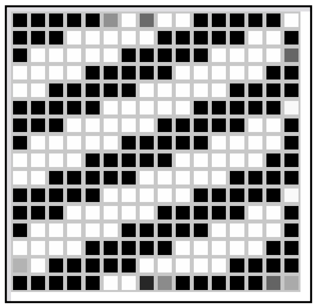


**Figure 9-14: Wave generated for circular locomotion**

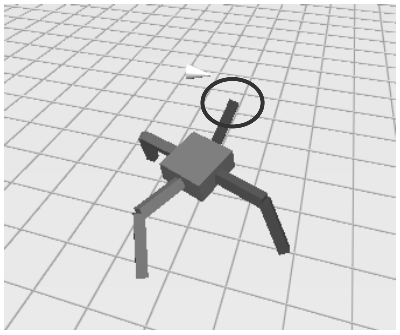


**Figure 9-15: Broken Leg Spider as an unstructured robot**

The definition of the fitness function is a little bit sensitive in this case. According to Equation 9-8, the fitness value has a direct relation with the distance between the initial position point (*Init_Center_Pos*) and the robot position (*Robot_Pos*). Further, this function has an inverse relation with the robot angle (*Robot_Angle*). During the optimization phase, the aim/goal is converging the fitness function to zero.

(9-8)

*Fitness = Dist (Init_Center_Pos, Robot_Pos)*(1/Robot_Angle)*

(9-9)

$$T_A = \begin{bmatrix} -0.61 & 1 & -3.85 \\ 4.61 & 0.21 & -4.77 \\ 0.97 & 3.25 & -4.17 \end{bmatrix}, T_B = \begin{bmatrix} 4.72 & 1.2 & -2.61 \\ -2.19 & 2.25 & 3.04 \\ -2.49 & 3.33 & 4.97 \end{bmatrix}, I = -3.92$$

After nearly 3300 chromosome generations and evolution, the robot is able to turn over its Yaw axis. Figure 9-16 shows the result of the wave pattern for unstructured spider turning. The usability of the templates in this paper can be summarized as follows. Templates are stored in a list/memory. By a high level task management templates are selected. This selection depends on the high level task management decision and the environment situation as well. Further, a factor of high importance is the behavioral architecture and behavioral programming. In fact, low level skills (e.g. Moving forward, turning left and right, jumping and etc.) For insuring the control of the robot are very important. The choice of templates is highly influenced by these factors.

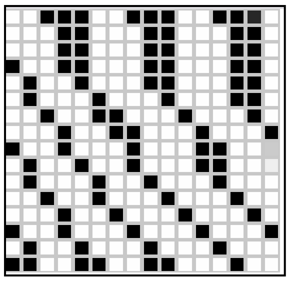


**Figure 9-16: Wave generated for Broken-Leg Spider; Turning Skill**

## 9.3 Conclusion

This paper has presented a concept based on an evolutionary technique for the robot locomotion learning. The technique proposed was a combination of both CNN and genetic algorithms. The motivation of this combination can be justified by the high accuracy of the CNN processors and their good computational speed as well. Further, the topology of CNN is flexible for designing neuro-evolutive systems. The genetic algorithm was exploited for the training process in order to determine the best genes according to the pre-defined requirements (i.e. dada requirements) for the design process. Two types of robots were considered (i.e. both structured and unstructured robots). For each of these types, algorithms were developed to derive the appropriate chromosomes from which corresponding templates were derived. The results in this paper have shown that combining the cellular neural networks (CNN) technology with an evolution scheme like genetic algorithm (GA) is very effective and suitable for learning the movement /locomotion of different types of robots (e.g. high DOF robots, symmetrical, unsymmetrical and defective robots). Due to the intrinsic characteristics of the CNN, this type of neural network is very close to natural processors and therefore is efficient for building robot controllers. During the training process, we found that the complexity of the environment (e.g. rough, bumpy, and/or scaly surfaces) was a key factor influencing the results. Basically, the technique developed in this paper provided interesting results with high accuracy in complex environments. Nevertheless, we found that the accuracy of the results decreases with the increasing complexity of the environment (e.g. ecosystem and robot environment). An interesting issue under investigation in subsequent and future works is implementing/developing methods of high accuracy and efficiency for robot control in very difficult environments.

# Chapter 10

## 10. Conclusion and Outlook

The Camera and the image processing unit belong the very important parts of machine vision-based ADAS concepts. Different weather and environmental conditions can significantly influence the image processing performance. The key challenging issue in ADAS is safety; hence all part of the system should work under any conditions without providing wrong information. To have a robust and real-time image processing module for ADAS, we do need a high performance processing system. Computing huge amount of visual information for extracting meaningful data, features, etc needs a special/appropriate hardware and processing architecture. In this research we did a survey about different hardware platforms for image processing and according to real-time related ADAS requirements we have proposed a robust and real-time architecture based on CNN, FPGA and GPU. The whole the system is a CNN based processing which is developed implemented on either GPU or FPGA. Before CNN implementation, we tried to understand the theory of analog computing and developed a direct emulation of that paradigm on the FPGA. Since there are many similarities between a CNN implementation and analog computing based on DDA, we could find a good way to implement CNN on FPGA. In this thesis, we have shown that CNN can solve a series of image processing tasks in real-time.

This thesis has formulated 7 key research questions and each main chapter did provide an answer to a respective research question:

- **Research question 1: What are the hard requirements of ADAS concerning real-time image processing and design flexibility? How far do traditional approaches fail to satisfy these requirements?**

  For answering to this question we performed a survey of all major ADAS concepts to check their respective architecture and requirements in terms of robustness and real-time processing. We have shown that in many cases the traditional approaches fail to satisfy the real-time processing requirements for complex scenarios. We did then propose a new concept that can satisfy all the cited requirements in ADAS systems including flexibility in design.

- **Research question 2: What are the major limitations of traditional high performance computing approaches if used to ensure "real-time" image processing in ADAS ?**

  For this research question we studied about high performance computing and real-time image processing. Also, we could show the limitation of traditional computing concepts/architectures based on the *Von Neumann* architecture. We have shown that manipulating and processing pixels in parallel does speed the image processing. For flexibility in design, hardware should be reconfigurabe by software in run-time mode and without any need for reprogramming the system.

- **Research question 3: What is the huge potential of neurocomputing involving either traditional neural networks (NN) or cellular neural networks (CNN) for high-speed and flexible image processing for ADAS? Are there any limitations and how can these eventually be addressed?**

  For this question we have shown that ANN has huge potential for image processing; examples of applications are pattern recognition, feature extraction, compression, etc. Also we have mentioned some drawbacks of ANN for hardware implementation and a comparison between CNN and ANN. Overall neurocomputing is a paradigm that promises to solve the tough requirements of ADAS concerning computing speed and flexibility.

- **Research question 4: What are the major template calculation schemes of relevance for CNN based image processing? How can these calculations be performed in a real-time high performance computing context?**

  Cellular neural networks technology provides a very powerful analog computing architecture for a variety of array computations and image processing tasks. From a theoretical point of view the CNN concept offers the capability of modeling various image processing filters and operators on a CNN processors' based "Universal Machine". For this research question we have analyzed three different methods for template calculation. We have then shown an example direct mapping of PDE's to templates for a selected image processing task. For a fast template calculation we have shown that there is a way to implement GA on FPGA and optimize any templates much faster than on computer/CPU.

- **Research question 5: How far can the advantages of "analog computing" be used/gained through an emulation of analog computing on digital hardware platforms like FPGA (for the benefit of an ultrafast image processing)?**

  The main advantage of analog computing is that we can "nearly" simultaneously get the result of complex mathematical differential equations. All signals are generated in parallel and in real-time and the electronic components do compute simultaneously. For this research question we have developed and implemented an emulation of analog computing on FPGA. The result is at least many thousand times faster than desktop computers. This result has been a good step for implementing complex models like CNN on FPGA.

- **Research question 6: How far can an efficient implementation of CNN on FPGA and GPU be designed and implemented?**

  CNN is a complex design in terms of implementation and performance on traditional architectures of the von Neumann type (such as CPU and sequential processors). Therefore, we did a survey about CNN implementations on hardware and later on we have proposed our implementation on FPGA. Due to the limitation of resources in FPGA, we have implemented a fix-point DT-CNN with high accuracy in results. Due to the offered flexibility of design by GPUs, we have also implemented a CNN universal machine on GPU image processing. We have used OpenCL which is a very strong framework for developing parallel algorithms on GPUs. The results obtained have shown to be at least 100 times faster than on normal computer/CPU for processing images.

- **Research question 7: How far can CNN be used/involved in an evolutionary computing/control context example (for illustration)?**

CNN has a great potential for signal processing tasks and it can generate very complex nonlinear waves and oscillation patterns at the output of CNN cells. Controlling both the kinematic and the inverse-kinematic of complex robots with high degree of freedom (DOF) is a very complex scenario whereby classical solutions fail to solve it easily. In the frame of this research question we have integrated a CNN processors system to the leg-robot for high level inverse kinematic controlling. For calculating templates we used GA and a very simple objective function without considering any inverse or direct kinematic. Different types of robots are able to move in an optimum way between two points or any kind of other scenarios and trajectories.

As outlook and future work we do see the integration of all functionalities developed in this thesis in a working prototype and improve it progressively according to test results in the real environment.

# References

1. Galizzi, M.M. *The Economics of Car-Pooling: A Survey for Europe*. 2004.
2. Agostinacchio, M., G. Albunia, and S. Olita, *Road Safety Evaluation: An Analytic Model Proposal For The Road Safety Audit And Review*. 2004, XIV Convegno Nazionale SIIV, Firenze, Italy.
3. Racioppi, F., *Preventing road traffic injury: a public health perspective for Europe*. 2004: World Health Organization Regional Office for Europe.
4. De Blaeij, A., et al., *The value of statistical life in road safety: a meta-analysis.* Accident Analysis & Prevention, 2003. 35(6): p. 973-986.
5. Hubaux, J.P., S. Capkun, and J. Luo, *The security and privacy of smart vehicles.* Security & Privacy, IEEE, 2004. 2(3): p. 49-55.
6. Vereeck, L. and K. Vrolix, *The social willingness to comply with the law: The effect of social attitudes on traffic fatalities.* International Review of Law and Economics, 2007. 27(4): p. 385-408.
7. Lu, M., K. Wevers, and R. Van Der Heijden, *Technical feasibility of advanced driver assistance systems (ADAS) for road traffic safety.* Transportation Planning and Technology, 2005. 28(3): p. 167-187.
8. May, J.F. and C.L. Baldwin, *Driver fatigue: The importance of identifying causal factors of fatigue when considering detection and countermeasure technologies.* Transportation Research Part F: Traffic Psychology and Behaviour, 2009. 12(3): p. 218-224.
9. Beekema, M. and H. Broeders, *Computer Architectures for Vision-Based Advanced Driver Assistance Systems.* Computer.
10. WALLNER, D., A. EICHBERGER, and W. HIRSCHBERG. *A Novel Control Algorithm for Integration of Active and Passive Vehicle Safety Systems in Frontal Collisions*. 2009: 2nd International Multi-Conference on Engineering and Technological Innovation.
11. Kaempchen, N., et al., *Sensor fusion for multiple automotive active safety and comfort applications.* Advanced microsystems for automotive applications 2004, 2004: p. 137-163.
12. Darms, M. and H. Winner. *A modular system architecture for sensor data processing of ADAS applications*. 2005: IEEE.
13. Bertozzi, M. and A. Broggi, *GOLD: A parallel real-time stereo vision system for generic obstacle and lane detection.* Image Processing, IEEE Transactions on, 1998. 7(1): p. 62-81.
14. Hsiao, P.Y. and C.W. Yeh. *A portable real-time lane departure warning system based on embedded calculating technique*. 2006: IEEE.
15. Arth, C., F. Limberger, and H. Bischof. *Real-time license plate recognition on an embedded DSP-platform*. 2007: IEEE.
16. Lee, J.W., *A machine vision system for lane-departure detection.* Computer vision and image understanding, 2002. 86(1): p. 52-78.
17. Bahlmann, C., et al. *A system for traffic sign detection, tracking, and recognition using color, shape, and motion information*. 2005: IEEE.
18. Chua, L.O. and L. Yang, *Cellular neural networks: Theory.* Circuits and Systems, IEEE Transactions on, 1988. 35(10): p. 1257-1272.

19. Cao, J. and D. Zhou, *Stability analysis of delayed cellular neural networks.* Neural Networks, 1998. 11(9): p. 1601-1605.
20. Lin, C.M., *Selective power-down for high performance CPU/system*. 1995, Google Patents.
21. Chhugani, J., et al., *Efficient implementation of sorting on multi-core SIMD CPU architecture.* Proceedings of the VLDB Endowment, 2008. 1(2): p. 1313-1324.
22. White, H., *Artificial neural networks: approximation and learning theory*. 1992: Blackwell Publishers, Inc. Cambridge, MA, USA.
23. Hagan, M.T., et al., *Neural network design*. 1996: PWS Boston, MA.
24. Agatonovic-Kustrin, S. and R. Beresford, *Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research.* Journal of pharmaceutical and biomedical analysis, 2000. 22(5): p. 717-727.
25. Albó-Canals, J., et al. *An efficient FPGA implementation of a DT-CNN for small image gray-scale pre-processing*. 2009: IEEE.
26. Toledo, F.J., et al., *Image processing with CNN in a FPGA-based augmented reality system for visually impaired people.* Computational Intelligence and Bioinspired Systems, 2005: p. 906-912.
27. Ruaro, M.E., P. Bonifazi, and V. Torre, *Toward the neurocomputer: Image processing and pattern recognition with neuronal cultures.* Biomedical Engineering, IEEE Transactions on, 2005. 52(3): p. 371-383.
28. Ham, F.M. and I. Kostanic, *Principles of neurocomputing for science and engineering*. 2000: McGraw-Hill Higher Education.
29. Gonzalez, R.C., R.E. Woods, and S.L. Eddins, *Digital image processing using MATLAB*. 2004: Pearson Education India.
30. Fasih, A., et al. *New computational modeling for solving higher order ODE based on FPGA*: IEEE.
31. Fasih, A., et al., *An Ultra-fast and Adaptive Framework for FPGA-Based Real-Time Machine Vision for Advanced Driver Assistance Systems: a CNN-Based Processing Architec-ture.*
32. Aspray, W., *Computing before computers*. 1990: Iowa State University Press.
33. Williams, J., *Analog circuit design: art, science and personalities*. 1991: Newnes.
34. Hubner, M., K. Paulsson, and J. Becker. *Parallel and flexible multiprocessor system-on-chip for adaptive automotive applications based on Xilinx microblaze soft-cores*: IEEE.
35. Nakamura, Y., et al. *A fast hardware/software co-verification method for system-on-a-chip by using a C/C++ simulator and FPGA emulator with shared register communication*. 2004: ACM.
36. Schmitt, L.M., *Theory of genetic algorithms.* Theoretical Computer Science, 2001. 259(1-2): p. 1-61.
37. Roser, M. and F. Moosmann. *Classification of weather situations on single color images*. 2008: IEEE.
38. Zheng, N.N., et al., *Toward intelligent driver-assistance and safety warning system.* Intelligent systems, IEEE, 2004. 19(2): p. 8-11.
39. Langheim, J., et al. *CARSENSE-New environment sensing for advanced driver assistance systems*. 2001.
40. Donecker, S.M., T.A. Lasky, and B. Ravani, *A mechatronic sensing system for vehicle guidance and control.* Mechatronics, IEEE/ASME Transactions on, 2003. 8(4): p. 500-510.

41. Kim, S.Y., et al., *An Intelligent and Integrated Driver Assistance System for Increased Safety and Convenience Based on All-around Sensing.* Journal of Intelligent and Robotic Systems, 2008. 51(3): p. 261-287.
42. Saad, J., A. Baghdadi, and F. Bodereau. *Fpga-based radar signal processing for automotive driver assistance system*: IEEE.
43. Wada, M., K.S. Yoon, and H. Hashimoto, *Development of advanced parking assistance system.* Industrial Electronics, IEEE Transactions on, 2003. 50(1): p. 4-17.
44. Wu, B.F., et al., *A DSP-based lane departure warning system.* Proc. of the 8th WSEAS Int. Conf. on Mathematical Methods and Computational Techniques in Electrical Engineering, 2006: p. 240-245.
45. Arth, C., F. Limberger, and H. Bischof, *Real-time license plate recognition on an embedded DSP-platform.* 2007: p. 1-8.
46. Vitabile, S., S. Bono, and F. Sorbello. *An embedded real-time automatic lane-keeping system*. 2007: Springer-Verlag.
47. Doshi, A. and M. Trivedi, *LACASA: A Layered Architecture for Cooperative Active Safety Applications.*
48. Azman, A., et al. *Optimizing resources of an fpga-based smart camera architecture*. 2007: IEEE Computer Society.
49. Batlle, J., et al., *A new FPGA/DSP-based parallel architecture for real-time image processing.* Real-Time Imaging, 2002. 8(5): p. 345-356.
50. Martínez-Alvarez, J., et al., *High Performance Implementation of an FPGA-Based Sequential DT-CNN.* Nature Inspired Problem-Solving Methods in Knowledge Engineering, 2007: p. 1-9.
51. Rudolph, L. and Z. Segall, *Dynamic decentralized cache schemes for MIMD parallel processors.* ACM SIGARCH Computer Architecture News, 1984. 12(3): p. 340-347.
52. Greco, J., *Parallel Image Processing and Computer Vision Architecture*. 2005, Citeseer.
53. Risack, R., N. Mohler, and W. Enkelmann, *A video-based lane keeping assistant.* 2000: p. 356-361.
54. VanderWerf, J., et al., *Modeling effects of driver control assistance systems on traffic.* Transportation Research Record: Journal of the Transportation Research Board, 2001. 1748(-1): p. 167-174.
55. Naab, K. and G. Reichart, *Driver assistance systems for lateral and longitudinal vehicle guidance-heading control and active cruise support.* JSAE Review, 1995. 16(2): p. 220-220.
56. McCall, J.C. and M.M. Trivedi, *Video-based lane estimation and tracking for driver assistance: survey, system, and evaluation.* Intelligent Transportation Systems, IEEE Transactions on, 2006. 7(1): p. 20-37.
57. Chavda, J., et al. *Performance Measure of Image Processing Algorithms on DSP Processor & FPGA Based Coprocessor*: IEEE Computer Society.
58. Chapman, B., G. Jost, and R. Van Der Pas, *Using OpenMP: portable shared memory parallel programming*. Vol. 10. 2007: The MIT Press.
59. Hwang, K., *Advanced computer architecture: parallelism, scalability, programmability*. Vol. 348. 1993: McGraw-Hill.
60. Hager, G. and G. Wellein, *Introduction to High Performance Computing for Scientists and Engineers*, CRC Press.
61. Dowd, K., *High performance computing*. 1993: O'Reilly & Associates, Inc.

62. Fan, Z., et al. *GPU cluster for high performance computing*. 2004: IEEE Computer Society.
63. Gelado, I., et al. *An asymmetric distributed shared memory model for heterogeneous parallel systems*: ACM.
64. Demigny, D., et al. *How to use high speed reconfigurable fpga for real time image processing?* 2000: IEEE.
65. Lee, C.C. and J.P. de Gyvez, *Color image processing in a cellular neural-network environment.* Neural Networks, IEEE Transactions on, 1996. 7(5): p. 1086-1098.
66. Zarandy, A., et al., *CNN-based models for color vision and visual illusions.* Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on, 1999. 46(2): p. 229-238.
67. Chan, T.F., J. Shen, and L. Vese, *Variational PDE models in image processing.* Notices of AMS, 2003. 50(1).
68. Chan, T.F. and J. Shen, *Image processing and analysis: variational, PDE, wavelet, and stochastic methods*. 2005: Society for Industrial Mathematics.
69. Kuijper, A., *Image processing with geometrical and variational pdes.* 2007: p. 89–96.
70. Chan, T.F. and J. Shen, *Nontexture inpainting by curvature-driven diffusions.* Journal of Visual Communication and Image Representation, 2001. 12(4): p. 436-449.
71. Song, B., *Topics in variational PDE image segmentation, inpainting and denoising*. 2003, Citeseer.
72. Lin, Z. and Q. Shi, *An anisotropic diffusion PDE for noise reduction and thin edge preservation.* 1999: p. 102-107.
73. Roska, T., et al., *Simulating nonlinear waves and partial differential equations via CNN. I. Basic techniques.* Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on, 1995. 42(10): p. 807-815.
74. Rajini, G. and G.R. Reddy. *Performance Evaluation of Neural Networks for Shape Identification in Image Processing*: IEEE.
75. Pinkus, A., *Approximation theory of the MLP model in neural networks.* Acta Numerica, 1999. 8(-1): p. 143-195.
76. Ghica, D., S.W. Lu, and X. Yuan. *Recognition of traffic signs by artificial neural network*. 1995: IEEE.
77. Bittencourt, J.R. and F.S. Osório. *Adaptive Filters for Image Processing based on Artificial Neural Networks*. 2000: Citeseer.
78. Egmont-Petersen, M., D. de Ridder, and H. Handels, *Image processing with neural networks--a review.* Pattern Recognition, 2002. 35(10): p. 2279-2301.
79. Belliustin, N. *Homogeneous ANN using in image processing and picture generation*: IEEE.
80. Hopfield, J.J., *Artificial neural networks.* Circuits and Devices Magazine, IEEE, 1988. 4(5): p. 3-10.
81. Fausett, L.V., *Fundamentals of neural networks: architectures, algorithms, and applications*. 1994: Prentice-Hall Englewood Cliffs, NJ.
82. Tu, J.V., *Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes* 1.* Journal of clinical epidemiology, 1996. 49(11): p. 1225-1231.
83. Hassoun, M.H., *Fundamentals of artificial neural networks*. 1995: the MIT Press.
84. Reed, R.D. and R.J. Marks, *Neural smithing: supervised learning in feedforward artificial neural networks*. 1998: MIT Press.

85. Tarassenko, I. and S. Roberts. *Supervised and unsupervised learning in radial basis function classifiers*. 1994: IET.
86. Hu, Y.H., J.N. Hwang, and S.W. Perry, *Handbook of neural network signal processing.* The Journal of the Acoustical Society of America, 2002. 111: p. 2525.
87. Moody, J. and C.J. Darken, *Fast learning in networks of locally-tuned processing units.* Neural computation, 1989. 1(2): p. 281-294.
88. Sanger, T.D., *Optimal unsupervised learning in a single-layer linear feedforward neural network.* Neural Networks, 1989. 2(6): p. 459-473.
89. Honkela, A. and H. Valpola, *Unsupervised variational Bayesian learning of nonlinear models.* Advances in neural information processing systems, 2005. 17: p. 593–600.
90. Malki, S., L. Spaanenburg, and N. Ray. *Image stream processing on a packet-switched discrete-time CNN*. 2004.
91. Yang, Z., Y. Nishio, and A. Ushida. *A Two Layer CNN in Image Processing Applications*.
92. Chua, L.O. and T. Roska, *The CNN paradigm.* Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on, 1993. 40(3): p. 147-156.
93. Matsumoto, T., et al. *Several image processing examples by CNN*: IEEE.
94. Kozek, T., T. Roska, and L.O. Chua, *Genetic algorithm for CNN template learning.* Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on, 1993. 40(6): p. 392-402.
95. Mohamad, S. and K. Gopalsamy, *Exponential stability of continuous-time and discrete-time cellular neural networks with delays.* Applied Mathematics and Computation, 2003. 135(1): p. 17-38.
96. Harrer, H. and J.A. Nossek, *Discrete time cellular neural networks.* International Journal of Circuit Theory and Applications, 1992. 20(5): p. 453-467.
97. Chen, H.C., et al., *Image-processing algorithms realized by discrete-time cellular neural networks and their circuit implementations.* Chaos, Solitons & Fractals, 2006. 29(5): p. 1100-1108.
98. Kawahara, M., T. Inoue, and Y. Nishio. *Image processing application using CNN with dynamic template*: IEEE.
99. Zarándy, Á., *The art of CNN template design.* International Journal of Circuit Theory and Applications, 1999. 27(1): p. 5-23.
100. Feiden, D. and R. Tetzlaff, *Cellular neural networks for motion estimation and obstacle detection.* Advances in Radio Science-Kleinheubacher Berichte. 1.
101. Feiden, D. and R. Tetzlaff. *Feature extraction in motion estimation with cellular neural networks using iterative annealing*.
102. UCAN, O.N., E. BILGILI, and R. COBAN, *Extraction Of Facial Features Using Genetic Cellular Neural Networks.* network. 1: p. 4.
103. Wolfram, S.E.W., S. (Ed.), *Theory and Application of Cellular Automata. Reading, MA: Addison-Wesley, 1986.*
104. Nossek, J.A., et al., *Cellular neural networks: Theory and circuit design.* International Journal of Circuit Theory and Applications, 1992. 20(5): p. 533-553.
105. Mitchell, M., *An introduction to genetic algorithms*. 1998: The MIT press.
106. Gacsádi, A., C. Grava, and A. Grava, *Medical image enhancement by using cellular neural networks.* 2005: p. 821-824.
107. Balya, D. and V. Gal, *Analogic Implementation of the Genetic Algorithm.* p. 1-6.
108. Small, J.S., *General-purpose electronic analog computing: 1945-1965.* Annals of the History of Computing, IEEE, 1993. 15(2): p. 8-18.

109. Bournez, O., et al., *The general purpose analog computer and computable analysis are two equivalent paradigms of analog computation.* Theory and Applications of Models of Computation, 2006: p. 631-643.
110. Mullins, B.F., *A Guide to Solving Simple Ordinary Differential Equations (ODE's).* Technical report, University of Toronto, Department of Physics. , 2009.
111. MacLennan, B.J., *Review of Analog Computing.* Technical Report, Department of Electrical Engineering & Computer Science University of Tennessee, Knoxville 2007.
112. Cowan, G.E.R., R.C. Melville, and Y.P. Tsividis, *A VLSI analog computer/digital computer accelerator.* Solid-State Circuits, IEEE Journal of, 2006. 41(1): p. 42-53.
113. Lee, J., B.J. Sheu, and R. Chellappa, *A VLSI neuroprocessor for image restoration using analog computing-based systolic architecture.* The Journal of VLSI Signal Processing, 1993. 5(2): p. 185-199.
114. Wittig, R.D. and P. Chow. *OneChip: An FPGA processor with reconfigurable logic*. 1997: IEEE.
115. Stoica, A., et al., *On-chip evolutionary synthesis of reconfigurable analog computing circuits.* Internet Source, 2003.
116. Sivaranjani, K., J. Venkatesh, and P. Janakiraman, *Realization of a digital differential analyzer using CPLDs.* International Journal of Modelling and Simulation, 2007. 27(3): p. 280.
117. Clark, C.R., R. Nathuji, and H.H.S. Lee. *Using an fpga as a prototyping platform for multi-core processor applications*. 2005: Citeseer.
118. Xu, S. and H. Pollitt-Smith. *A multi-microblaze based SOC system: from SystemC modeling to FPGA prototyping*. 2008: IEEE.
119. Ho, T.Y., P.M. Lam, and C.S. Leung, *Parallelization of cellular neural networks on GPU.* Pattern Recognition, 2008. 41(8): p. 2684-2692.
120. Dolan, R. and G. DeSouza, *GPU-based simulation of cellular neural networks for image processing.* 2009: p. 730-735.
121. Athanas, P.M. and A.L. Abbott, *Real-time image processing on a custom computing platform.* Computer, 1995. 28(2): p. 16-25.
122. Chu, P.P., *FPGA prototyping by VHDL examples: Xilinx Spartan-3 version*. 2008: LibreDigital.
123. miKayaer, K. and V. Tavsanoglu. *A new approach to emulate CNN on FPGAs for real time video processing*. 2008: IEEE.
124. Rosinger, H.P., *Connecting customized IP to the MicroBlaze soft processor using the Fast Simplex Link (FSL) channel.* Xilinx Application Note, 2004.
125. Pellerin, D. and M. Saini, *MicroBlaze and PowerPC Cores as Hardware Test Generators.* Xilinx Xcell journal, 2004. 51.
126. Pellerin, D., et al., *Accelerating PowerPC software applications.* Xilinx Xcell Embedded Magazine.
127. Aung, Y., et al., *C-based design methodology for fpga implementation of clustalw msa.* Pattern Recognition in Bioinformatics, 2007: p. 11-18.
128. Pellerin, D. and S. Thibault, *Practical fpga programming in c*. 2005: Prentice Hall Press Upper Saddle River, NJ, USA.
129. Roska, T., *Cellular neural networks.* John Wiley & Sons, Inc. New York, NY, USA, 1994.
130. Perez-Munuzuri, V., V. Perez-Villar, and L.O. Chua, *Autowaves for image processing on a two-dimensional CNN array of excitable nonlinear circuits: flat and wrinkled*

*labyrinths.* Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on, 1993. 40(3): p. 174-181.

131. Fernandez, A., et al. *Cellular neural networks simulation on a parallel graphics processing unit*: IEEE.

132. Kanwar, R., *Real-Time Edge Detection using Sundance Video and Image Processing System.* 2009.

133. Choudhary, A. and S. Ranka, *Guest editor's introduction: parallel processing for computer vision and image understanding.* Computer, 1992. 25(2): p. 7-10.

134. Wang, X. and S.G. Ziavras. *HERA: A reconfigurable and mixed-mode parallel computing engine on platform FPGAs*. 2004: Citeseer.

135. Tonde, C., *Hardware and software platforms for computer vision.*

136. Tsuchiyama, R., et al., *The OpenCL Programming Book.* Group, 2009.

137. Barak, A., et al. *A package for OpenCL based heterogeneous computing on clusters with many GPU devices*: IEEE.

138. Chua, L.O. and T. Roska, *Cellular neural networks and visual computing: foundation and applications*. 2002: Cambridge Univ Pr.

139. Montúfar-Chaveznava, R. and D. Guinea, *A Cellular Neural Network System for Real-Time Image Processing.*

140. Takenouchi, H., T. Watanabe, and A. Hideki, *Development of DT-CNN Emulator Based on GPGPU.*

141. Arena, P., L. Fortuna, and M. Branciforte, *Reaction-diffusion CNN algorithms to generate and control artificial locomotion.* Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on, 1999. 46(2): p. 253-260.

142. Arena, P., et al. *Bio-inspired robotics: application of a CNN-based CPG VLSI chip to control an autonomous mini-hexapod robot*. 2006: IEEE.

143. Arena, P., et al., *An adaptive, self-organizing dynamical system for hierarchical control of bio-inspired locomotion.* Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, 2004. 34(4): p. 1823-1837.

144. Fortuna, L. and L. Patane. *Hexapod locomotion control through a cnn based decentralized system*. 2002: IEEE.

145. Arena, P., et al. *An autonomous mini-hexapod robot controlled through a CNN-based CPG VLSI chip*. 2006: IEEE.

146. Branciforte, M., et al. *Reaction-Diffusion CNN design for a new class of biologically-inspired processors in artificial locomotion applications*. 1999: Published by the IEEE Computer Society.

147. Horn, J., N. Nafpliotis, and D.E. Goldberg. *A niched Pareto genetic algorithm for multiobjective optimization*. 1994: Ieee.

148. Cho, S.B. *Evolving multiple sensory-motor controllers based on cellular neural network*. 2001: IEEE.

149. Cigale, B. and D. Zazula, *Segmentation of ovarian ultrasound images using cellular neural networks.* Proceedings IWSSIP 2000, 2000: p. 33-36.

150. Beasley, J. and P.C. Chu, *A genetic algorithm for the set covering problem.* European Journal of Operational Research, 1996. 94(2): p. 392-404.

151. Bilotta, E., G. Cutrí, and P. Panano, *Evolving Robot's Behavior by Using CNNs.* From Animals to Animats 9, 2006: p. 631-639.

152. Wolff, K. and P. Nordin. *Learning biped locomotion from first principles on a simulated humanoid robot using linear genetic programming*. 2003: Springer.

153. Marbach, D. and A.J. Ijspeert. *Co-evolution of configuration and control for homogenous modular robots*. 2004.
154. Muthuswamy, B., *Implementing central pattern generators for bipedal walkers using cellular neural networks*. 2005, UNIVERSITY OF CALIFORNIA.
155. Arena, P., et al. *Realization of a CNN-driven cockroach-inspired robot*. 2006: IEEE.