

Applied High-Speed Analogic Algorithms for Multitarget Tracking and Offline Handwriting Segmentation

Ph.D. dissertation

Gergely Tímár

Analogical and Neural Computing Laboratory
Computer and Automation Institute
Hungarian Academy of Sciences

Scientific Adviser:

Dr. Csaba Rekeczky, Ph. D

Budapest, 2006

“We are at the very beginning of time for the human race. It is not unreasonable that we grapple with problems. But there are tens of thousands of years in the future. Our responsibility is to do what we can, learn what we can, improve the solutions, and pass them on.”

Richard Feynman

Acknowledgements

I would like to thank *prof. Tamás Roska* for his guidance, help and support throughout my research and daily work. I am also grateful to *Dr. Csaba Rekeczky* for introducing me to the world of CNNs, for the many thoughtful discussions, the many ideas that grew into some of the algorithms in this dissertation and the adventurous trips. To *Dr. Dávid Bálya* for his thorough and constructive comments on a prior version of the manuscript, and together with *György Cserey* their thought provoking and entertaining discussions could bridge multiple time zones and continents. To *Dr. István Szatmári* for his insights and practical advice and to *dr. Viktor Gál* for his help with biology and his offbeat jokes, which cracked me up when there was the greatest need for it.

I would also like to thank my family: *my parents*, who encouraged me to enter graduate school, and supported me throughout the years and *Orsi, my sister*, who was always there for me, when I needed it, and *Tamás Deli* who helped with the proofreading of the manuscript and provided valuable comments.

In addition, I would like to thank my colleagues, with whom I discussed ideas, questions and worked together on different projects: *Dr. Ákos Zarándy, Dr. Péter Szolgay, Dr. Tamás Szirányi, Viktor Binzberger, István Horváth, Tibor Gyimesi, Dr. Levente Török, Kristóf Karacs, Dr. László Kék, Dr. Péter Földesy, Dr. László Orzó, Zoltán Szlávik, Péter Jónás.*

A heartfelt “Thank You” goes out to *Katalin Keserű* and *Gabriella Kék* for their kind help with the intricacies of the bureaucracy.

Finally, I am sorry if I inadvertently omitted somebody. It was not intentional.

Table of Contents

Acknowledgements.....	5
1 Introduction.....	11
1.1 The CNN Model.....	13
1.1.1 State equation of a single layer CNN with first order cell model	14
1.2 The Wave Computing Model	16
1.3 The CNN Universal Machine (CNN-UM).....	17
1.4 CNN-UM Implementations, Utilized Hardware	19
1.5 Application Examples	22
2 Multitarget Tracking Algorithm for Video Flows.....	25
2.1 Block-level overview of the MTT algorithm	26
2.2 The CNN-UM algorithms.....	27
2.2.1 Enhancement Methods and Spatio-Temporal Channel Processing.....	27
2.2.2 Remarks on the Ace4k and Ace16k Chip Implementation of the Multi-channel CNN Algorithms	31
2.2.3 Channel Interaction and Detection Strategies	32
2.2.4 Prediction Methods.....	33
2.2.5 Feature Extraction and Target Filtering	34
2.2.6 The Full Multichannel Algorithm.....	35
2.3 The DSP-based MTT Algorithms.....	37
2.3.1 Data Association	37
2.3.2 2-D Assignment Algorithms.....	39
2.3.2.1 The Utilized Distance Measures	39
2.3.3 Track Maintenance.....	40
2.3.4 State Estimation	41
2.3.4.1 Fixed-gain State Estimation Filters (α - β and α - β - γ Filters)	41
2.3.4.2 Multiple Model Filter with Fixed Multiple Models	43
2.4 Automatic Parameter Tuning.....	46
2.4.1 Response to Lighting Changes.....	46
2.4.2 Response to Target Motion Changes.....	46
2.4.3 Response to Channel Output Corruption.....	48
2.5 Experiments and Results	49

2.5.1	Algorithm Accuracy Measurements	50
2.5.2	Algorithm Performance Measurements.....	54
2.6	Discussion.....	55
2.7	Case Study: The Multitarget Framework in a Real-time Control Environment.....	57
3	Analogic Preprocessing and Segmentation Algorithms for Offline Handwriting Recognition	61
3.1	The Basic Structure of an Offline Handwriting Recognition System.....	62
3.2	Description of the Test Handwriting Database	64
3.3	The Preprocessing Tasks And Their Solutions	64
3.3.1	Locating The Lines	64
3.3.2	Correcting the Line Skew.....	66
3.3.3	Segmentation of Lines into Words.....	67
3.4	Segmentation Of Words Into Letters.....	68
3.4.1	Localization of the Upper and Lower Baselines, Skew Correction.....	69
3.4.2	Images Aiding in Segmentation	70
3.4.3	Skeletons of the Background.....	71
3.4.3.1	Locating the Endpoints of the Background Skeleton	73
3.4.4	Skeletons of the Foreground	73
3.5	Segmenting Words into Letters	73
3.5.1	Post processing the Endpoints of Skeletons – Parallel Distance Approximation....	74
3.5.2	The Letter Segmentation Algorithm	76
3.6	Discussion of the Results	77
3.7	Conclusions	79
4	Summary	81
4.1	Methods of Investigation.....	82
4.2	New Scientific Results.....	82
4.3	Application Areas of the Results	84
5	Bibliography.....	87
5.1	Publications Related to CNNs and CNN Technology	87
5.2	Publications Related to Multitarget Tracking	88
5.3	Publications Related to Offline Handwriting Recognition.....	89
5.4	The Author’s Publications.....	92
5.4.1	Journal Papers.....	92
5.4.2	International Conference Papers	93

6	Appendix: CNN Templates, Operators and Subroutines.....	95
6.1	Basic Operators.....	95
6.2	Subroutines.....	98
6.3	Linear, Isotropic CNN Templates.....	99
6.4	Linear, Non-Isotropic Templates.....	100

1 Introduction

When I started working in the research group at the Analogic and Neural Computing Laboratory, one could literally feel an air of excitement lingering in the offices. A new chip was in the final stages of the design process and was to be sent out to the foundry for fabrication. This new chip was one of the crown jewels of mixed-signal VLSI hardware and contained a relatively new type of processor, called a Cellular Neural Network Universal Machine (CNN-UM). The new chip was christened Ace16k, and is one of the processors used in implementing and testing most algorithms of this dissertation.

CNNs were first described by prof. L. Chua and L. Yang [1] and then further developed into universal machines (in the Turing sense) with prof. Tamás Roska [2]-[5]. The new processing paradigm promised breakthrough solutions to previously untractable problems, and the race was on to find application areas where the CNN-UM chips would really excel. During the last ten years, more and more scientists in the research community have started to use CNNs and especially CNN-UMs to solve diverse problems and to test the applicability of the paradigm in varied problem domains. The formal model has become ever more precise and rich (Cellular Wave Computer) while the physical implementations are becoming more advanced (optical, emulated digital, FPGA-based etc.), and the “Bi-i” computer which combines a classic digital processor and sensor with a CNN-UM has also been introduced. VLSI silicon-based implementations of the CNN-UM paradigm have advanced as well, from the initial 20x22 resolution to 128x128 and the soon to be available 176x144. Some versions are also capable of executing multilayer CNNs.

It was clear to me at the start of my research, that the discipline and the available tools have reached a maturity level where new problems may be tackled that were thought to be impossible,

or impractical to attempt before. Many papers were written to solve important problems related to image processing using CNN-UMs (and their implementations), but there have been very few attempts to use CNN-UMs in complex systems in any application area. I felt that solutions, which embrace and build on the different virtues of CNN-based and classical DSP-based algorithms, could allow the creation of better and more efficient algorithms than what would be possible with either architecture alone.

I started my research work with the goal of designing two systems: one used for tracking multiple targets in real time using optical input, the other for preprocessing the handwritten page images for offline handwriting recognition. Both tasks seem comparatively easy to a human being, but previous algorithms were only able to show modest results.

In multitarget tracking the task is to track many rapidly moving objects in a plane, so that the system is able to determine the kinematic properties of the individual targets (position, speed and acceleration) while robustly handling errors from occlusions and illumination changes that may occur. During the development of my algorithm, I relied heavily on our group's accumulated knowledge gained from research into modeling the mammalian retina. In essence, in the retina, the input image is filtered and transformed in various different ways and the image streams are processed in parallel, and only very sparse and compactly coded information is sent toward the higher areas of the brain involved in vision. I use this principle in the algorithm described in Chapter 2 of the dissertation to ensure that the measurements from a given input image are the best possible, increasing the accuracy of the whole tracking system. The application of the same principle also made it possible for the system to adapt to changes in the environment while running, ensuring robust tracking.

Offline handwriting recognition is the recognition of a handwritten text after its writing was completed (usually on paper, but other forms of media may also be used). In contrast to *online* handwriting recognition, where dynamic information about the writing is available (frequently used in PDAs, for example), *offline* handwriting recognition is more difficult, because no temporal data on the dynamics of the writing is available, which usually increases the ambiguity of the written text. The successful commercial systems in use, which rely on offline handwriting recognition, are systems that operate in a constrained environment, where the range of possible handwritten input strings is limited and may be easily enumerated, such as the addressing on envelopes, or the writing on prescriptions, for example. Unconstrained offline handwriting recognition is still an unsolved problem, largely because the shortcuts that are possible due to the limited number of input strings are not applicable. I designed algorithms that would perform the tasks at the image preprocessing stage of an offline handwriting recognition system, which

consists of the segmentation of an image into handwritten lines, the segmentation of the lines into words, and the words into letters. In order to improve their speed, the utilized CNN algorithms use dynamic, wave front propagation based methods instead of relying on morphologic operators embedded into iterative algorithms (whenever possible).

1.1 The CNN Model

This section introduces the theoretical principles of CNNs. A Cellular Neural/Nonlinear Network (CNN), as an operator, is defined by the following constraints:

- A spatially discrete collection of continuous nonlinear dynamical systems called cells where information can be encrypted into each cell via three independent variables called input (u), threshold (z), and initial state ($x(0)$).
- A coupling law relating one or more relevant variables of each cell to all local neighboring cells located within a prescribed sphere of influence $S_r(ij)$ of radius r centered at i,j .

Figure 1.1 shows a 2D rectangular CNN composed of cells that are connected to their nearest neighbors. Due to its symmetry, the regular structure and simplicity of this type of arrangement (a rectangular grid) are of primary importance in all implementations.

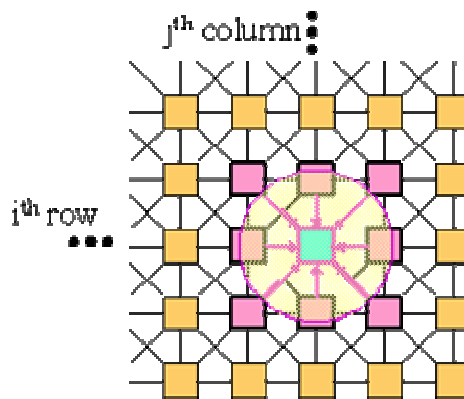


Figure 1.1 A two-dimensional CNN defined on a square grid. The i,j -th cell of the array and cells that fall within the sphere of influence of neighborhood radius $r = 1$ (the nearest neighbors) are highlighted

The CNN paradigm does not specify the properties of a cell. The implemented cell models are in Figure 1.2. As the basic framework throughout this dissertation, let us consider a two-dimensional ($M \times N$) CNN array in which the cell dynamics is described by nonlinear ordinary differential equations with linear and nonlinear terms. The extension to three dimensions is straightforward allowing similar interlayer interactions.

The bias (also referred to as the “bias map”) of a CNN layer is a grayscale image. The bias map can be viewed as the space-variant part of the cell threshold. By using pre-calculated bias maps, “linear” spatial adaptivity can be added to the templates in CNN algorithms. If the bias map is not specified, it is assumed to be zero.

CELL MODEL NAME	PLACE OF DESIGN	NUMBER OF PARAMETERS	REMARKS
DTCNN Discrete time CNN	Munich, 1993		Only the value of the state is analog, the time and space are discrete.
First order Chua-Yang model	Berkeley, 1996	19	Standard first-order CNN cell
PHS positive high-gain sigmoid	Helsinki, 1997	11	Hard-limited output for binary image processing,
FSR Full state range	Seville, 1998	20	The state and output are the same and the voltage swing of the transient is limited [-1;1]
vBJP – pseudo Bipolar Junction Transistor	Taiwan, 2000	12	4-connected, 2-neighborhood
FSR Complex-kernel	Seville, 2002	25	2 nd -order FSR-based model
R-Unit		21	3 rd -order spatially isotropic templates

Figure 1.2 The different CNN cell models

The mask (also referred to as the “fixed-state map”) of a CNN layer is a binary image specifying whether the corresponding CNN cell is in active or inactive state in the actual operation. Using the binary mask is one of the simplest ways to incorporate “nonlinear” spatial adaptivity to the templates in CNN algorithms. If the mask is not specified, it is assumed that all CNN cells are in active state, that is, the initial state is not fixed.

In order to specify fully the dynamics of the array, the boundary conditions have to be defined. Cells along the edges of the array may see the value of cells on the opposite side of the array (circular boundary), a fixed value (Dirichlet-boundary) or the value of mirrored cells (zero-flux boundary).

1.1.1 State equation of a single layer CNN with first order cell model

The standard first order CNN array dynamics is described by the following equations, which are equivalent to the simplest wave instruction. The $\mathbf{C}:\{\mathbf{A}, \mathbf{B}, \mathbf{z}\}$ is the cloning template.

$$\text{Cell dynamics: } \tau \dot{x}_{ij}(t) = -x_{ij}(t) + z_{ij} \quad (1.1)$$

$$\text{Local cell interactions: } A * y_{ij} + B * u_{ij} \quad (1.2)$$

$$\text{Output equations: } y_{ij}(t) = f(x_{ij}(t)) = \frac{|x+1| + |x-1|}{2} = \begin{cases} 1 & \text{if } x_{ij}(t) \geq 1 \\ x_{ij}(t) & \text{if } -1 \leq x_{ij}(t) \leq 1 \\ -1 & \text{if } x_{ij}(t) \leq -1 \end{cases} \quad (1.3)$$

where

- x_{ij} , y_{ij} , u_{ij} are the state, the output, and the input voltage of the specified CNN cell, respectively. The state and output vary in time, the input is static (time independent), ij refers to a grid point associated with a cell on the 2D grid.
- z_{ij} is the cell bias (also referred to as threshold) which could be space and time variant.
- τ is the cell time-constant
- Term A represents the linear coupling, B the linear control.
- Term $f(\cdot)$ is the output nonlinearity, in our case a unity gain sigmoid.
- t is the continuous time variable.

The first part of Eq. (1.1) is called cell dynamics; the following additive terms represent the synaptic linear and nonlinear interactions. Though the threshold z_{ij} may be space-variant, usually it is added to the template (space-invariant case). Eq. (1.3) is the output equation. A CNN cloning template, the program of the CNN array, is given by the linear and nonlinear terms completed by the cell current.

Many times, the CNN cell dynamics are implemented via an electronic circuit and the interactions are added as shown in Figure 1.3. The input, state and output variables are voltages, the A and B templates are VCCS-s (voltage controlled current sources) and the time constant comes from the capacitance (C) and resistance (R) of the cells as $\tau = RC$.

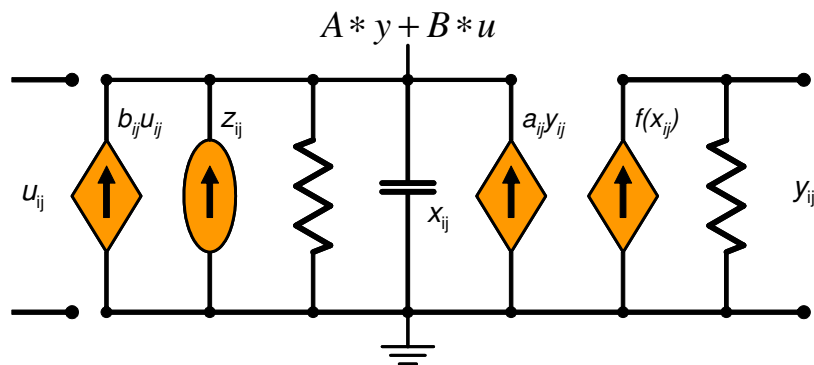


Figure 1.3 A CNN base cell corresponding to equations (1.1), (1.2). The linear control and coupling terms are represented by voltage controlled current sources (B and A).

1.2 The Wave Computing Model

Since the CNN network is an analog construct, its computing dynamics are very different from regular digital processors. The computational operators can be considered “waves” and prompted the definition of a new model of computation for CNNs. The definitions provided here are based on [7].

1. Data is defined as a continuous image flow $\Phi(t)$

$$\Phi(t): \{\varphi_{ij}(t), i = 1, 2, \dots, n; j = 1, 2, \dots, m\} \in \mathbf{R} \times \mathbf{R} \quad t \in T = [0, t^*] \quad (1.1)$$

A frame is obtained by setting the time variable in a given finite time instance t^* , i.e.

$P = \Phi(t^*)$. Without loss of generality, we may assume that in a gray scale image +1 and -1 represent the black and white levels, respectively, and the gray levels are in between.

2. Elementary instructions Ψ are the basic wave instructions:

$$\Phi_{\text{output}}(t) = \Psi(\Phi_{\text{input}})$$

Input: $U(t) \equiv \Phi_{\text{input}}: u_{ij}(t), t \in T$

State: $X(t) \equiv \Phi(t): x_{ij}(t), t \in T$ Initial state: $X(0)$

Output: $Y(t) \equiv \Phi_{\text{output}}: y_{ij}(t), t \in T$

Operator: the solution of the two-dimensional spatial-temporal state equation/output equation not necessarily only at the equilibrium point(s)

$$\tau \frac{dx_{ij}(t)}{dt} = -x_{ij} + z_{ij} + \sum_{kl \in S_r(ij)} A_{ij,kl} y_{kl}(t) + \sum_{kl \in S_r(ij)} B_{ij,kl} u_{kl}(t) \quad (1.4)$$

$S_r(\cdot)$: sphere of influences: $S_r(ij) = \{C(kl): \max\{|k-i|, |l-j|\} \leq r\}$

$y_{ij}(t) = \sigma(x_{ij}(t))$; σ : a nonlinear, usually sigmoid function

Useful operator notations: $\dot{x} \equiv \frac{dx_{ij}(t)}{dt}$; $A * y \equiv \sum_{kl \in S_r(ij)} A_{ij,kl} y_{kl}(t)$

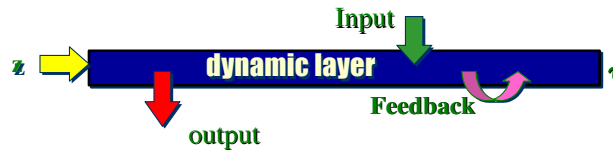


Figure 1.4 The graphical representation of the elementary wave processing. The horizontal bar represents the processing structure as a two-dimensional layer and the arrows show the external connections to each cell. The color of the arrow refers to the sign of the connection. The individual $\varphi_{ij}(t)$ base units and their connections are shown on Figure 1.1.

1.3 The CNN Universal Machine (CNN-UM)

All early neural network chip realizations had a common problem: they implemented a single instruction only, thus the weight matrix was fixed when processing some input. Reprogramming (i.e. changing the weight matrix) was possible for some devices but took an order of magnitudes longer time than the computation itself.

This observation motivated the design of the CNN Universal Machine (CNN-UM, [2]), a stored program nonlinear array computer. This new architecture is able to combine analog array operations with local logic efficiently. Since the reprogramming time is approximately equal to the settling time of a non-propagating analog operation, it is capable of executing complex analogic algorithms. To ensure programmability, a global programming unit was added to the array, and to ensure an efficient reuse of intermediate results, each computing cell was extended by local memories. In addition to local storage, every cell might be equipped with local sensors and additional circuitry to perform cell-wise analog and logical operations. The architecture of the CNN-UM is shown in Figure 1.5.

As illustrated in Figure 1.5, the CNN-UM is built around the dynamic computing core of a simple CNN. An image can be acquired through the sensory input (e.g. **OPT**: Optical Sensor). Local memories store analog (**LAM**: Local Analog Memory) and logic (**LLM**: Local Logical Memory) values in each cell. A Local Analog Output Unit (**LAOU**) and a Local Logic Unit (**LLU**) perform cell-wise analog and logic operations on the stored values. The output is always transferred to one of the local memories. The Local Communication and Control Unit (**LCCU**) provides for communication between the extended cell and the central programming unit of the machine, the Global Analogic Programming Unit (**GAPU**). The GAPU has four functional blocks. The Analog Program Register (**APR**) stores the analog program instructions, the CNN templates. In case of linear templates, for a connectivity $r = 1$ a set of 19 real numbers has to be stored. If spatial symmetry and isotropy is assumed, this is even less. All other units within the GAPU are registers containing the control codes for operating the cell array. The Local Program Register (**LPR**) contains control sequences for the individual cell's LLU, the Switch Configuration Register (**SCR**) stores the codes to initiate the different switch configurations when accessing the different functional units (e.g. whether to run a linear or nonlinear template). The Global Analogic Control Unit (**GACU**) stores the instruction sequence of the main (analogic) program. The GACU also controls timing, sequence of instructions and data transfers on the chip and synchronizes the communication with any external controlling device. It has its own global analog and logic memories (GAM and GLM, respectively) and global Arithmetic Logic Unit (**ALU**). As a special case, the GACU can be implemented by a digital signal processor

(DSP) or a microcontroller.

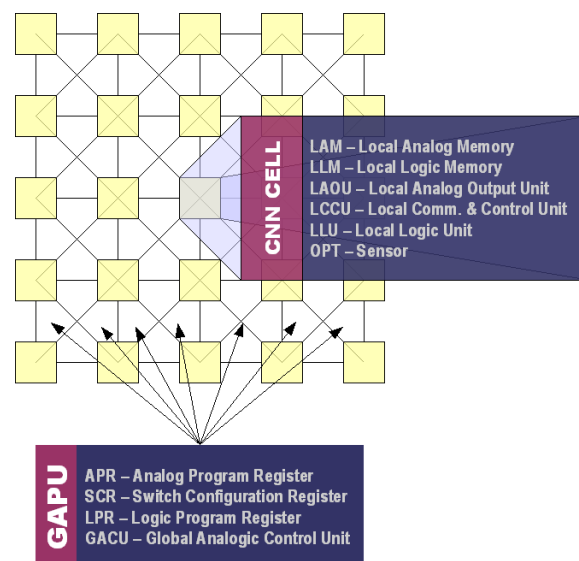


Figure 1.5 The architecture of the CNN Universal Machine

Different cell models are used in the CNN-UM implementations. The simplest one is the discrete time CNN (DT-CNN), when the state evolution is not continuous but discrete in time. The so-called *full signal range* model (FSR) is a VLSI-friendly non-linear cell model, where the voltage of the state variable is always the same as the output [10].

Synthesizing an analogic algorithm running on the CNN-UM the designer should decompose the solution in a sequence of analog and logical operations. A limited number of intermediate results can be locally stored and combined. Some of these outputs can be used as a bias map (space-variant current) or fixed-state map (space-variant mask) in the next operation adding spatial adaptivity to the algorithms without introducing complicated inter-cell couplings. Either linear or nonlinear templates define analog operations. The output can be defined in both the fixed and the non-fixed state of the network (equilibrium and non-equilibrium computing) depending on the control of the transient length. It can be assumed that elementary logical (**NOT**, **AND**, **OR**, *etc.*) and arithmetical (**ADD**, **SUB**, **MUL**) operations are implemented and can be used on the cell level between LLM and LAM locations, respectively. Certain operators (e.g. arithmetical) might have two input values, denoted by P and Q . In addition, data transfer and conversion can be performed between LAMs and LLMs.

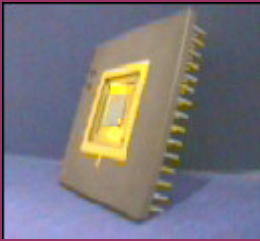
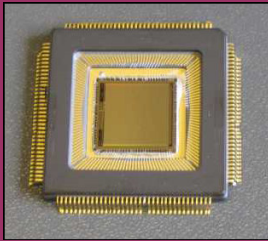
In most image processing tasks, the input and state of the CNN array are loaded with image data, and the result of CNN computation is generally defined as the steady state after the transient of the network. If each cell of the CNN array is equipped with analog and logic memory units, logic operations can be defined between these logic memory units, and these logic

functions along with the templates become programmable, we arrive at the concept of the CNN Universal Machine (CNN-UM) [2]. This (re)programmability makes the CNN a real computer, the first algorithmically programmable *analogic* (i.e., both analog and logic) computer. In this framework, each particular CNN operation (analog transient computation or local logic operation) can be thought of as an *analogic instruction* of this computer. This allows to store intermediate results of the processing, and to build up and run complex image processing algorithms on a CNN chip using some control hardware.

In the course of CNN template and algorithm design, many useful specialized templates and simple template combinations have been found, many of which are compiled in a CNN Software Library [16]. These provide basic components of several standard image-processing techniques. Beyond that, analogic CNN algorithms may utilize a number of spatio-temporal effects in their basic operations, which can hardly be applied if conventional image processing technology is used. These may be mathematical morphology or different PDE-based techniques.

1.4 CNN-UM Implementations, Utilized Hardware

When I started this research in 2001, the best available VLSI CNN-UM implementation in our lab was the Ace4k chip designed at IMSE-CNM, in Seville [8]. After about 2 years, the next generation Ace16k became available, with 4 times as many cells and new capabilities. Figure 1.6 shows a brief comparison of the two chips' capabilities.

<i>Property</i>	<i>ACE4K</i>	<i>ACE16K</i>
		
<i>Technology</i>	CMOS 0.5 μ m 3M-1P	CMOS 0.35 μ m 5M-1P
<i>Die size</i>	9.145 x 9.534 mm ²	11.885 x 12.230 mm ²
<i>Number of transistors</i>	~1,000,000	3,748,170
<i>Number of cells</i>	64 x 64	128 x 128
<i>Number of template memories</i>	32	32
<i>Number of instruction memories</i>	64 configurations	64 x 64 configurations
<i>Image I/O</i>	Analog and binary	Digital (8 bit), internal DAC-s and ADC-s
<i>Image I/O control</i>	Address buses and external timer based control	Internal addressing handshake protocols, no timing constraints

<i>Property</i>	<i>ACE4K</i>	<i>ACE16K</i>
<i>Max. Image I/O rate</i>	1MHz (analog), 10MHz (binary)	32 MHz (digital)
<i>Cell density</i>	82 cell/mm ²	180 cell/mm ²
<i>Optical sensor</i>	Parasitic	Logarithmic and linear integration
<i>Address event detection</i>	-	Available
<i>Analog memory writing mask</i>	-	Available
<i>Number of LAMs</i>	4	8
<i>Number of LLMs</i>	4	-
<i>Transistors per cell</i>	172	198
<i>Time constant (t)</i>	~ 1.2 μ s (maximum feedback)	~ 0.8 μ s (linear convolution: 160 ns)
<i>Power dissipation</i>	1.5 W (worst case)	< 4 W

Figure 1.6 Comparison of the VLSI CNN-UM implementations used in the experiments and algorithm validation. Both chips were designed at the IMSE-CNM, in Seville, Spain

A new commercial focal plane processor designed by AnaFocus Ltd. in Spain will also appear in 2006 that will be significantly more advanced than the Ace16k chip. It has an even larger resolution (QCIF: 176x144) with monochrome and RGB color sensors at each cell. The sensor's dynamic range is 68 dB and the maximum speed of operation is 10000 frames/sec (Fps), with power dissipation at 30 mW @ 25Fps and 100mW @ 10000 Fps.

Of course, the chips by themselves are not very useful in real applications, they must be embedded in a suitable hardware (usually a camera) which also contains additional supporting hardware such as memory, control circuits etc. Since the field of VLSI CNN-UM chips is changing rapidly, our development hardware was constantly evolving during this research. We used several platforms supplied by Analogic Computers Ltd [19].

The Ace4k chip was enclosed in the ACE-BOX platform, which is a PC104 form factor expansion card for Intel compatible PCs. It contains a Texas Instruments TMS320C6202 DSP running at 250Mhz and 16MB of SD-RAM. This platform did not have optical input capabilities, so we had to rely on external input sources such as video files or high-speed cameras. Figure 1.7 shows the ACE-BOX.

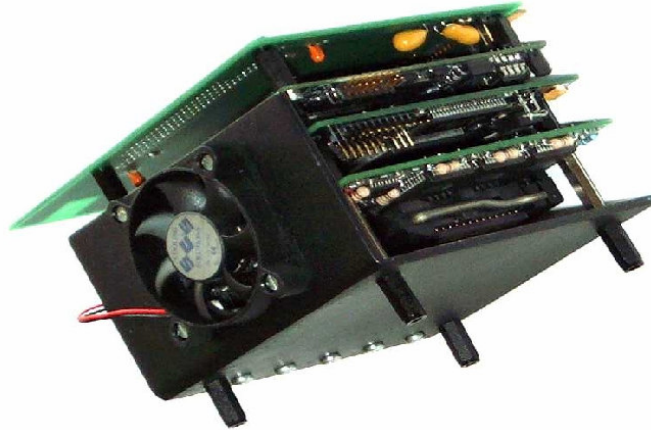


Figure 1.7 The ACE-BOX platform (a PCI extension card) hosting the Ace4k chip

The new Ace16k chip was hosted in a completely stand-alone hardware, which grew into an entire family of devices, called the Bi-i platform. The Bi-i has the unique capability of supporting the simultaneous image acquisition and processing from two sensors at the same time. These sensors can be Ace16k CNN-UMs, or regular CMOS image sensors in any configuration. The initial versions of the Bi-i cameras (Bi-i v1) contained the same TI DSP as the ACE-BOX which controlled the Ace16k chip. Since the Bi-i camera is stand-alone, it contains a number of IO interfaces to communicate with the outside world: serial ports, 100Mbit Ethernet, USB 1.1 and programmable digital IO. We used these capabilities to demonstrate the control possibilities using the MTT algorithms. Current versions of the Bi-i camera (Bi-i v2) contain a more powerful DSP, the TI TMS3206415 running 600Mhz and there have been some cosmetic changes to the housing as well. Figure 1.8 shows the two versions of the Bi-i cameras.



Figure 1.8 The Bi-i Smart Cameras. The left picture shows the Bi-i v1, with 2 sensors, while the right picture shows the Bi-i v2 with 2 sensors and the laser scanner attached

1.5 Application Examples

Practical applications of the designed systems abound. The multitarget tracking system may form the foundation of an advanced, “intelligent” surveillance and monitoring system. These systems are the more sophisticated siblings of the ubiquitous CCTV (closed circuit TV) and security camera networks installed in many places today. These security cameras are currently manned 24 hours a day, 7 days a week by human guards, who typically have to monitor the images of many cameras at the same time. This job is very tiring and very boring, which leads to security breaches and slow reaction to critical events. In an “intelligent” surveillance system, the system can detect, analyze and act upon certain events. Most of these events have to do with the motion of something: a vehicle, a human (or a group of humans) or an animal. If the system can

- isolate and track moving objects in a scene
- reliably measure the properties of the movement
- distinguish between objects based on its features

then it is possible to set up rules to trigger alarms and other intervention responses. The multitarget tracking algorithms described in the thesis can provide much of the needed functionality. The MTT algorithms calculate the location, speed and acceleration of the moving targets. They also analyze the target features, which enable the differentiation of different classes of targets. The MTT algorithm also provides a way to follow certain moving objects across cameras, if the location of the cameras is known a priori (which is nearly always the case).

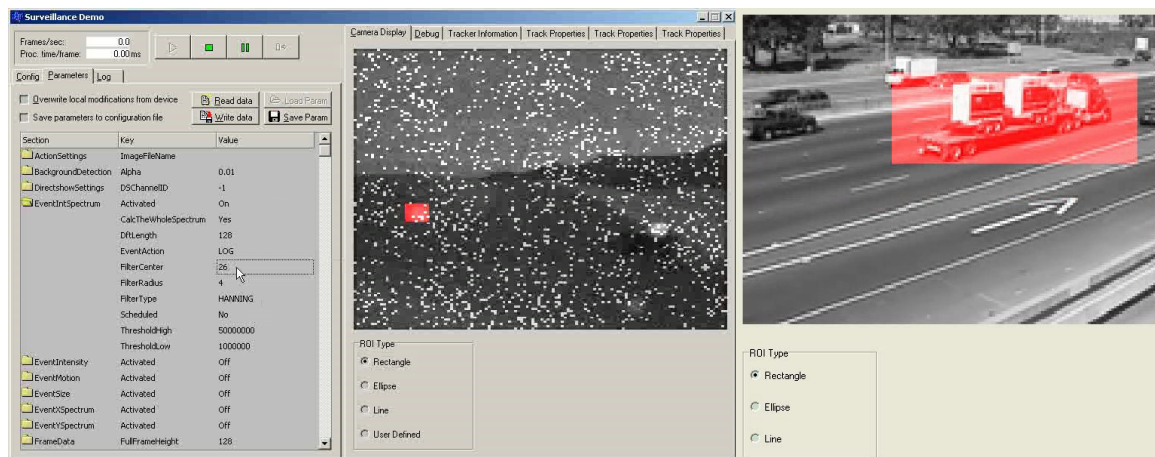


Figure 1.9 Screenshots from a prototype surveillance application utilizing the MTT algorithms (the targets generating an alert are highlighted in red). In the left screenshot, target tracking is combined with image intensity analysis to generate an alert. The target is engulfed in background noise, but its distinctive intensity change profile enables successful target localization. In the screenshot on the right, target size is the alert trigger.

The MTT algorithm could also be utilized in certain unmanned aerial vehicle (UAV)

applications, where the optical flow must be calculated from the visual input. Optical flow is usually determined based on local correlation based methods, which determine local displacement based on local similarities in texture. These methods break down, if:

- the texture is periodic, because there will be many matches during the local search
- the vehicle speed is too large, because the images changes will be too great
- the vehicle speed is too small, because the images changes will be too small

An MTT algorithm can help by identifying distinct salient features as “targets” that may be tracked across individual frames. In contrast to conventional correlation-based approaches, these salient “targets” do not have to be located in a specified neighborhood between individual frames. The optical flow can be calculated from the kinematic properties of these targets even if the sampling of the movement is very sparse. MTT-based optical flow can also help if certain areas of the frame are occluded for some reason (clouds, for example).

2 Multitarget Tracking Algorithm for Video Flows

Recognizing and interpreting the motion of objects in video flows is an essential task in a number of applications, such as security, surveillance, online quality control, vision-based control etc. In many instances, the individual objects to be tracked have no known distinguishing features (such as a distinct color, for example) which would allow feature (or token) tracking [20],[21], optical flow or motion estimation [22],[23]. Therefore, the targets must be identified and tracked largely based on their measured positions and derived motion parameters. These applications also demand real-time performance, i.e. the tracking algorithms must run fast enough to keep up with the input image flow; this requirement puts a severe limit on the types of algorithms that may be used.

Target tracking algorithms developed for tracking targets based on sonar and radar measurements are widely known and can be used for tracking based on visual input (also known as motion correspondence). However, the requirement that the system should operate at least at video frame-rate (possibly even higher) limits the choices between the well-established statistical and non-statistical tracking algorithms. The real-time requirements motivated the use of a unique image sensing and processing device, the CNN-UM [1]-[9] and its VLSI implementations, which provide several advantages over conventional CMOS or CCD sensors:

- Possibility of focal plane processing, which means that the acquired image does not have to be moved from the sensor to the processor
- Very fast parallel image processing operators
- Unique trigger-wave and diffusion based operators

While planning the algorithm, we realized that the decreased running time of image

processing algorithms could provide some headroom within the real-time constraints that would allow for the use of more complex state estimation and data assignment algorithms and sensor adaptation possibilities. We decided to explore these possibilities in detail, and present the results in section 2.2. In the next section, we give a high-level overview of the system, and then we present the algorithms running on the CNN-UM. In section 2.3, we give an overview of the algorithms used in estimating the state of the targets, and creating and maintaining the target tracks and the adaptation possibilities that the tight coupling of the track maintenance system (TMS) and the sensor can provide. Finally, we present experimental results obtained by running the algorithms on actual hardware in section 2.5.

2.1 Block-level overview of the MTT algorithm

The MTT system contains two main architectural levels (Figure 2.1): the CNN-UM level, where all of the image processing takes place (and possibly image acquisition) and the DSP level where track management functions are performed. After image acquisition, the CNN-UM can perform image enhancement to compensate for ambient lighting changes, motion extraction and related image processing tasks and feature extraction for some types of features. The DSP runs the rest of the feature extraction routines, and the motion correspondence algorithms such as distance calculation, gating, data assignment and target state estimation. It also calculates new values for some CNN-UM algorithm parameters thus adapting the processing to the current environment.

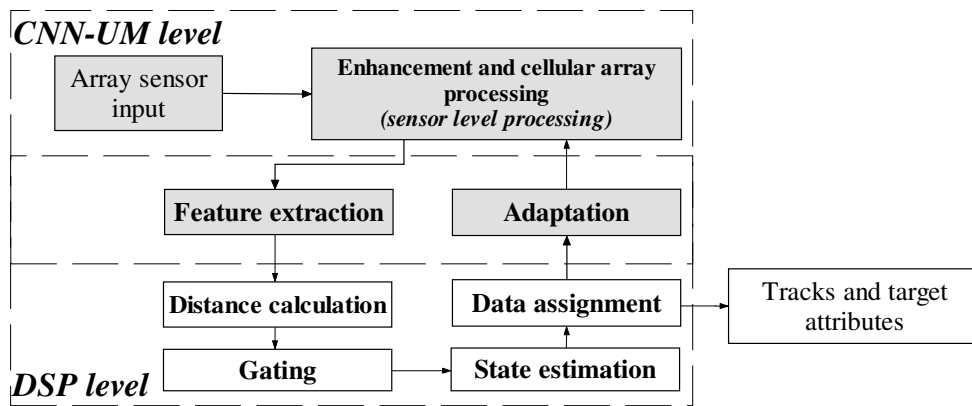


Figure 2.1 CNN-UM/DSP hybrid system architecture for multi-target tracking. The main processing blocks are divided into three categories: those that are best performed on the CNN-UM processor, those that are especially suitable for the DSP, and those that have to be performed using both processors.

2.2 The CNN-UM algorithms

Knowledge gained from the study of the mammalian visual system, especially the retina heavily influenced the algorithms presented here. Recent studies [26] uncovered that the retina processes visual information in parallel spatio-temporal channels and transmits only a sparse encoding of the information on these channels via the optic nerve to higher visual areas in the brain. Additionally, there is context and content sensitive interaction between these parallel channels via enhancement and suppression which results in remarkable adaptivity. These are highly desirable characteristics for all image-processing systems, however they are essential for visual tracking tasks where degradation of input measurements can not always be compensated for at the later stages of processing (by domain knowledge, for example).

In the following subsections, we will describe a conceptual framework for such a complex CNN-UM based front-end algorithm. First, we will discuss the computing blocks in general and then specify the characteristics of the test implementation on the Ace4k CNN-UM chip and Ace16k CNN-UM chips operating in the ACE-BOX and Bi-i systems, respectively.

2.2.1 Enhancement Methods and Spatio-Temporal Channel Processing

We tried to capture the main ideas from the natural system by defining three “change enhancing” channels on the input image flow: a spatial, a temporal and a spatio-temporal channel (see Figure 2.2A). The spatial channel contains the response of filters that detect spatial i.e. brightness changes, revealing the edges in a frame. The temporal channel contains the result of computing the difference between two consecutive frames, thereby giving a response to changes, while the spatio-temporal channel contains the non-linear combination of the spatial and temporal filter responses.

To eliminate additive Gaussian noise, we use a linear low-pass filter on the input grayscale image before any further processing is attempted (both linear and constrained linear diffusion approximations can be used). The pseudo code for the filter (*PreProc*) is:

$$\begin{aligned}\Phi_{pp} &= \text{PreProc}_D(\Phi, \sigma) \\ \Phi_{pp} &= \text{Diffus}(\Phi, \sigma) \\ \Phi_{pp} &= \text{PreProc}_{CD}(\Phi_1, \Phi_2, \lambda, \sigma_1, \sigma_2) \\ \Phi_{pp} &= \text{CDiffus}(\Phi_1, \Phi_2, \lambda, \sigma_1, \sigma_2)\end{aligned}$$

The filtering on the parallel channels can be defined as causal recursive difference-type filtering using some linear or nonlinear filters as prototypes (typically difference of Gaussian (DoG) filters implemented using constrained linear diffusion [27], or difference of morphology (DoM) filters implemented by min-max statistical filters [10]). These filters can be thought of as

band-pass filters tuned to a specific spatial/temporal frequency (or a very narrow band of frequencies), thus enabling highly selective filtering of grayscale images and sequences.

The three main parameters of these change-enhancing filters are:

- Spatial scale (σ): the spatial frequency(s) (basically the object size) the filter is tuned to (in pixels)
- Temporal rate (λ): the rate of change in an image sequence the filter is tuned to (in pixels/frame)
- Orientation (ϕ): the direction in which the filter is sensitive (in radians)

In our current framework, the orientation parameter is not used, since we are relying on isotropic Gaussian kernels (or the approximations thereof) to construct our filters, but we are including it here because the framework does not inherently preclude the use of it. It is possible to tune the spatial channel's response to objects of a specific size (in pixels) using the σ parameter. Similarly, the λ parameter allows the filtering out of all image changes except those occurring at a certain rate (in pixels per frame). This enables the multi-channel framework to specifically detect targets with certain characteristics.

The output of these channels is filtered through a sigmoid function:

$$y = \frac{1}{1 + e^{-\beta(x-\vartheta)}} \quad (2.1)$$

The parameters of this function are the threshold (ϑ) and slope (β). For every $x > \vartheta$, the output of the function is positive, hence the threshold name. The slope parameter specifies the steepness of the transition from 0 to 1 and as it becomes larger, the sigmoid approximates the traditional threshold step function more closely.

The following paragraphs present the pseudo-code for the image channels. The *temporal filter* (**TeFilt**) calculates the convex sum between the grayscale input image and an internal state (which is the result of the previous operation) and subtracts the resulting image from the grayscale input image (this corresponds to a causal recursive temporal motion sensitive filtering if the images are subsequent frames of a grayscale image flow):

$$\begin{aligned} \Phi_{TF}(k) &= \text{TeFilt}(\Phi(k), \lambda) \\ \Phi_{TF}(k) &= \Phi(k) - \Phi_H(k-1) \\ \Phi_H(k) &= (1 - \lambda)\Phi_H(k-1) + \lambda\Phi(k) \end{aligned}$$

The spatial filter (**SpFilt**) calculates the spatial local difference based enhancement (a Laplacian of a Gaussian, Difference of Gaussians or Sobel of Gaussians) of a grayscale input image:

$$\begin{aligned}
 \Phi_{SPF} &= SpFilt_LoG(\Phi, \sigma_1, \sigma_2) \\
 \Phi_{SPF} &= (\Phi * G(\sigma_1)) * L(\sigma_2) \\
 \Phi_{SPF} &= SpFilt_DoG(\Phi, \sigma_1, \sigma_2) \\
 \Phi_{SPF} &= \Phi * G(\sigma_1) - \Phi * G(\sigma_2) \\
 \Phi_{SPF} &= SpFilt_SoG(\Phi, \sigma_1, \sigma_2) \\
 \Phi_G &= \Phi * G(\sigma_1) \\
 \Phi_{SPF} &= \Phi * S_H(\sigma_2) + \Phi * S_V(\sigma_2)
 \end{aligned}$$

The *spatio-temporal filter* (**SPTFilt**) calculates the convex sum between the low-pass filtered grayscale input image and an internal state (which is the result of the previous operation). It subtracts the resulting image from the low-pass filtered grayscale input image (this corresponds to a causal recursive spatio-temporal motion sensitive filtering if the images are subsequent frames of a grayscale image flow):

$$\begin{aligned}
 \Phi_{SPTF}(k+1) &= SPTFilt(\Phi(k), \sigma, \lambda) \\
 \Phi_D(k) &= \Phi(k) * G(\sigma) \\
 \Phi_{SPTF}(k) &= \Phi_D(k) - \Phi_H(k-1) \\
 \Phi_H(k) &= (1 - \lambda)\Phi_H(k-1) + \lambda\Phi_D(k)
 \end{aligned}$$

The output of the best performing individual channel could be used by itself as the output of the image processing front-end, if the conditions where the system is deployed are static and well controlled. If the conditions are dynamic or unknown a priori, then there is no way to predict the best performing channel in advance. To circumvent this problem, we decided to combine the output of the individual channels through a so-called interaction matrix, and use the combined output for further processing. The inclusion of the interaction matrix enables the flexible runtime combination of the images on these parallel channels and the prediction map while also specifying a framework that can be incorporated into the system at design time. Our experimental results and measurements indicate that the combined output is on average more accurate than each single channel for different image sequences. Figure 2.2A shows the conceptual block diagram of the multi-channel spatio-temporal algorithm with all computing blocks to be discussed in the following section.

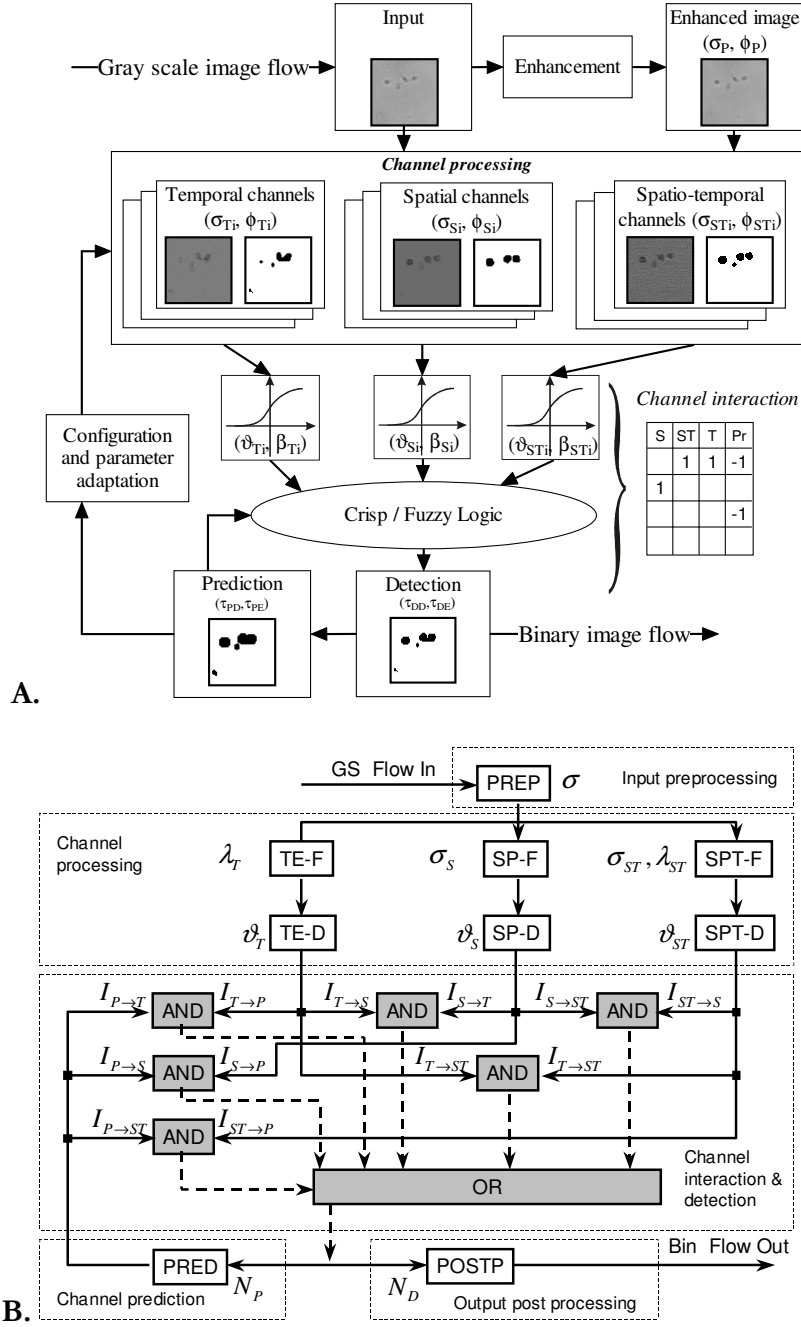


Figure 2.2 A) Block overview of the channel-based image processing algorithm for change detection. B) Ace4k/16k implementation of the same algorithm. The input image is first enhanced (histogram modified), and then it is processed in three parallel change-enhancing channels. These channels and the prediction image are combined through the interaction matrix and thresholded to form the final detection image. Observe, that the framework allows the entire processing to be grayscale (using fuzzy logic); the only constraint is that the detection image must be binary. In the Ace4k implementation, the results of the channel processing are thresholded to arrive at binary images, which are then combined using Boolean logic functions as specified by the interaction matrix. The parameters for the Ace4k algorithm are: λ – the temporal rate of change, σ – the scale (on the spatial (SP) and spatio-temporal (SPT) channels), ϑ – per channel threshold values, L – logical inversion (-1) or simple transfer (+1), N – the number of morphological opening ($N > 0$) or closing ($N < 0$) operations

2.2.2 Remarks on the Ace4k and Ace16k Chip Implementation of the Multi-channel CNN Algorithms

The change enhancing channels are actually computed serially (time multiplexed) in the current implementation, but this is not a problem due to the high speed of the CNN-UM chips used. We performed the first round of experiments using the ACE-BOX hardware containing the Ace4k chip, since the next generation Ace16k was not yet available. As preparation for this dissertation progressed, the Ace16k chips became available for experimentation in the Bi-i system. We modified the original Ace4k algorithms to take advantage of the capabilities of the new chip: we substituted isotropic diffusion wherever iterative convolutions were used to achieve the same effect. This makes the algorithm much faster, since the iterative steps are eliminated.

In the first stage of the on-going experiments, only isotropic ($\phi \rightarrow 0$) spatio-temporal processing has been considered followed by crisp thresholding through a hard nonlinearity (essentially a step function acting as a threshold). Thus, the three types of general parameters used to derive and control the associated CNN templates (or algorithmic blocks) are the scale and rate parameters (σ and λ) and the threshold parameter ϑ . Figure 2.2B shows the functional building blocks of the Ace4k implementation of the algorithm (a hardware-oriented simplification of the conceptual model) with all associated parameters.

The enhancement (smoothing) techniques have been implemented in the form of nearest neighbor convolution filters (circular positive B template with entries normalized to 1) and applied to the actual frame (σ determines the scale of the prefiltering in pixels, i.e. the number of convolution steps performed). The spatio-temporal channel filtering (including the temporal filtering solution) was implemented as a fading memory nearest neighbor convolution filter applied to the actual and previous frames on the Ace4k, while it was possible to realize directly on the Ace16k via its resistive grid. In temporal filtering configuration (no spatial smoothing), λ represents the fading rate (in temporal steps), thereby specifying the temporal scale of the difference enhancement. In spatio-temporal filtering configuration (the fading rate is set to a fixed value), σ represents the spatial scale (in pixels) at which the changes are to be enhanced (the number of convolution operations on the current and the previous frame are calculated implicitly from this information).

The pure spatial filtering is based on Sobel-type spatial processing of the actual frame along horizontal-vertical directions and combining the outputs into a single “isotropic” solution (here σ_s represents the spatial support in pixels in the Sobel-type difference calculation).

2.2.3 Channel Interaction and Detection Strategies

The interaction between the channels may be Boolean logic based for binary images or fuzzy logic based for grayscale images, specified via the so-called channel interaction matrix. Its role is to facilitate some kind of cross-channel interaction to further enhance relevant image characteristics and generate the so-called detection image, which is treated as the result of image processing steps in the whole tracking system. The interaction matrix is a matrix where each column stands for a single image. These images are the outputs of the parallel channels (SP, T, SPT) and the prediction (see next section, Pr). The values within the matrix specify the interaction “weight” (\mathbf{w}) of a given image (the image selected by the column of the matrix element). If using binary images, the non-zero weights are treated as follows: if $\mathbf{w} > 0$, then the input image is used, if $\mathbf{w} < 0$, then its inverse is used.

The interaction takes place in a row-wise fashion, with the row-wise results aggregated. The interactions themselves are given globally as a function pair, and must be Boolean or real valued functions (when using binary or grayscale images, respectively). The first function in the pair is the row-wise interaction function (\mathbf{R}); the second is the aggregation function (\mathbf{A}). \mathbf{R} is used to generate an intermediate result (\mathbf{I}_r) for each row. These intermediate results are the arguments of \mathbf{A} , which the aggregation function uses to generate the detection image. The number of rows in the interaction matrix must be at least one, but can be arbitrarily large, which allows the construction of sophisticated filters. A sample interaction matrix with the calculated detection result is shown on Figure 2.3. Figure 2.3 A sample interaction matrix, and the calculated result. $\text{Detection} = \mathbf{A}(\mathbf{R}(\text{SP}, \text{T}, \text{Pr}), \mathbf{R}(-\text{T}, \text{SPT}))$

<i>SP</i>	<i>T</i>	<i>SPT</i>	<i>Pr</i>
1	1		1
	-1	1	

Figure 2.3 A sample interaction matrix, and the calculated result. $\text{Detection} = \mathbf{A}(\mathbf{R}(\text{SP}, \text{T}, \text{Pr}), \mathbf{R}(-\text{T}, \text{SPT}))$

If using a fuzzy methodology, the detection image is thresholded, so the result of the channel interaction is always a binary map (the detection map) that will be the basis for further processing. Ideally, this only contains black blobs where the moving targets are located.

In our current experiments, we used only Boolean logic based method. In the binary case, the channels are thresholded depending on the ϑ parameters of the channel detection modules

then combined pair wise through AND logic and all outputs are summarized through a global OR gate (which corresponds to $R:=AND$ and $A:=OR$).

We post process the output of the interaction matrix using N_D -step morphological processing (one of the following, depending on the application scenario: erosion, dilation, opening or closing) to correct the target image perimeters and fill any internal errors. The following pseudo code shows post processing algorithm (**PostProc**):

$$\begin{aligned} \Phi_{PO} &= PostProc_E(\Phi, B, p_n) \\ \Phi_{PO} &= \Phi \\ &for\ i = 1\ to\ p_n \\ &\quad \Phi_{PO} = Erode(\Phi_{PO}, B) \\ &end \end{aligned}$$

$$\begin{aligned} \Phi_{PO} &= PostProc_D(\Phi, B, p_n) \\ \Phi_{PO} &= \Phi \\ &for\ i = 1\ to\ p_n \\ &\quad \Phi_{PO} = Dilate(\Phi_{PO}, B) \\ &end \end{aligned}$$

$$\begin{aligned} \Phi_{PO} &= PostProc_O(\Phi, B, p_n) \\ \Phi_{PO} &= Open(\Phi, B, p_n) \end{aligned}$$

$$\begin{aligned} \Phi_{PO} &= PostProc_C(\Phi, B, p_n) \\ \Phi_{PO} &= Close(\Phi, B, p_n) \end{aligned}$$

2.2.4 Prediction Methods

We also compute a prediction map that specifies the likely location of the targets in the image solely based on the current detection map and the previous prediction. This can then be used (via the interaction matrix) as a mask to filter out spurious signals. It is hard to include efficiently kinematic assumptions at the cellular level of processing – other than the maximum speed of the moving targets – given the real-time constraints, since this would require the generation of a binary image based on the current detection and the kinematic state parameters. Therefore, the algorithms only use isotropic maximum displacement estimation implemented by spatial logic and trigger-wave computing. However, the experiments indicate that even rudimentary input masking can be very helpful in obtaining better MTT results.

The algorithm (**Pred**) calculates an N -step isotropic expansion of the objects identified by

spatial logic from two binary input images (this corresponds to a “prediction map” given that one of the input images is a “detection map” containing the target objects in form of binary patches and the other input image is the “prediction map” from the previous operation). Only those objects in the prediction map that are also part of the detection map are expanded:

```

 $\Phi_{PredOut} = Pred(\Phi_{Det}, \Phi_{Pred}, B, N_D)$ 
  if isempty( $\Phi_{Det}$ )
     $\Phi_{PredOut} = \Phi_{Pred}$ 
  else
     $\Phi_0 = \Phi_{Det}$  and  $\Phi_{Pred}$ 
    if isempty( $\Phi_0$ )
       $\Phi_{PredOut} = \Phi_{Det}$ 
    else
       $\Phi_{DP} = Rec(\Phi_{Pred}, B, \Phi_0)$ 
       $\Phi_{CompDP} = \Phi_{DP} XOR \Phi_{Pred}$ 
       $\Phi_{PredOut} = \Phi_{Det}$  or  $\Phi_{CompDP}$ 
    end
  for  $i = 1$  to  $N_D$ 
     $\Phi_{PredOut} = Dilate(\Phi_{PredOut}, B)$ 
  end
end

```

2.2.5 Feature Extraction and Target Filtering

The DSP state-estimation and data assignment algorithms operate on position measurements of the detected targets, therefore these have to be extracted from the detection map. During data extraction, it is also possible to filter targets according to certain criteria based on easily (i.e. rapidly) obtainable features. The set of features we are currently using is: area, centroid, bounding box, equivalent diameter (diameter of a circle with same area), extent (the proportion of pixels in the bounding box that are also in the object), major and minor axis length (the length of the major axis of the ellipse that has the same second-moments as the object), eccentricity (eccentricity of the ellipse that has the same second-moments as the object), orientation (the angle between the x -axis and the major axis of the ellipse that has the same second-moments as the object) and the extremal points. Filtering makes it possible to concentrate on only a certain class of targets while ignoring others.

The calculation of all of these features can be implemented on the DSP but some of the features (centroid, horizontal or vertical CCD etc.) can be efficiently computed on the CNN-UM as well. Since the detection map is already present on the CNN-UM, calculation of these features can be extremely fast. It is also possible to calculate a set of features in parallel on the DSP and the CNN-UM, further speeding up this processing step. The location of the center of

gravity (centroid) of each target is usually considered the position of the target, unless special circumstances dictate otherwise.

Analogic chip implementation: Morphological filtering (structure and skeleton extraction) is implemented on the Ace4k chip. Feature extraction is performed exclusively on the DSP in the first test implementation.

2.2.6 The Full Multichannel Algorithm

In this section, we present pseudo code which implements the multichannel algorithm described above for MTT. It uses subroutines presented in the previous sections, and relies on operators described in the Appendix.

```

if k = 0
     $\Phi_{Pred}(k) = empty()$ 
end
 $\Phi_{PP}(k) = PreProc(\Phi(k), \sigma)$ 
 $\Phi_{CH_1}(k) = TeFilt(\Phi_{PP}(k), \lambda_T)$ 
 $\Phi_{CT_1}(k) = Thr(\Phi_{CH_1}(k), \vartheta_T)$ 
 $\Phi_{CH_2}(k) = SpFilt(\Phi_{PP}(k), 0, \sigma_{SP})$ 
 $\Phi_{CT_2}(k) = Thr(\Phi_{CH_2}(k), \vartheta_{SP})$ 
 $\Phi_{CH_3}(k) = SPTFilt(\Phi_{PP}(k), \sigma_{SPT}, \lambda_{SPT})$ 
 $\Phi_{CT_3}(k) = Thr(\Phi_{CH_3}(k), \vartheta_{SPT})$ 
 $\Phi_{CT}^*(k) = [\Phi_{CT_1}(k), \Phi_{CT_2}(k), \Phi_{CT_3}(k), \Phi_{Pred}(k-1)]$ 
 $\Phi_{Det}(k) = empty()$ 
for i = 1 to 4
    for j = i+1 to 4
         $\Phi_{CI}(k) = I(i, j) \Phi_{CT_i}^*(k) \text{ and } I(j, i) \Phi_{CT_j}^*(k)$ 
         $\Phi_{Det}(k) = \Phi_{Det}(k) \text{ or } \Phi_{CI}(k)$ 
    end
end
 $\Phi_{Pred}(k) = Pred(\Phi_{Det}(k), B, N_{DP})$ 
 $\Phi_{PO}(k) = PostProc(\Phi_{Det}(k), B, N_{PO})$ 

```

CNN algorithms also have a common formal description, somewhat similar to flowcharts, but more adapted to the unique capabilities of CNNs, called the UMF diagram [18]. Figure 2.4 shows the CNN implementation of the MTT multichannel front-end algorithm expressed in the UMF framework.

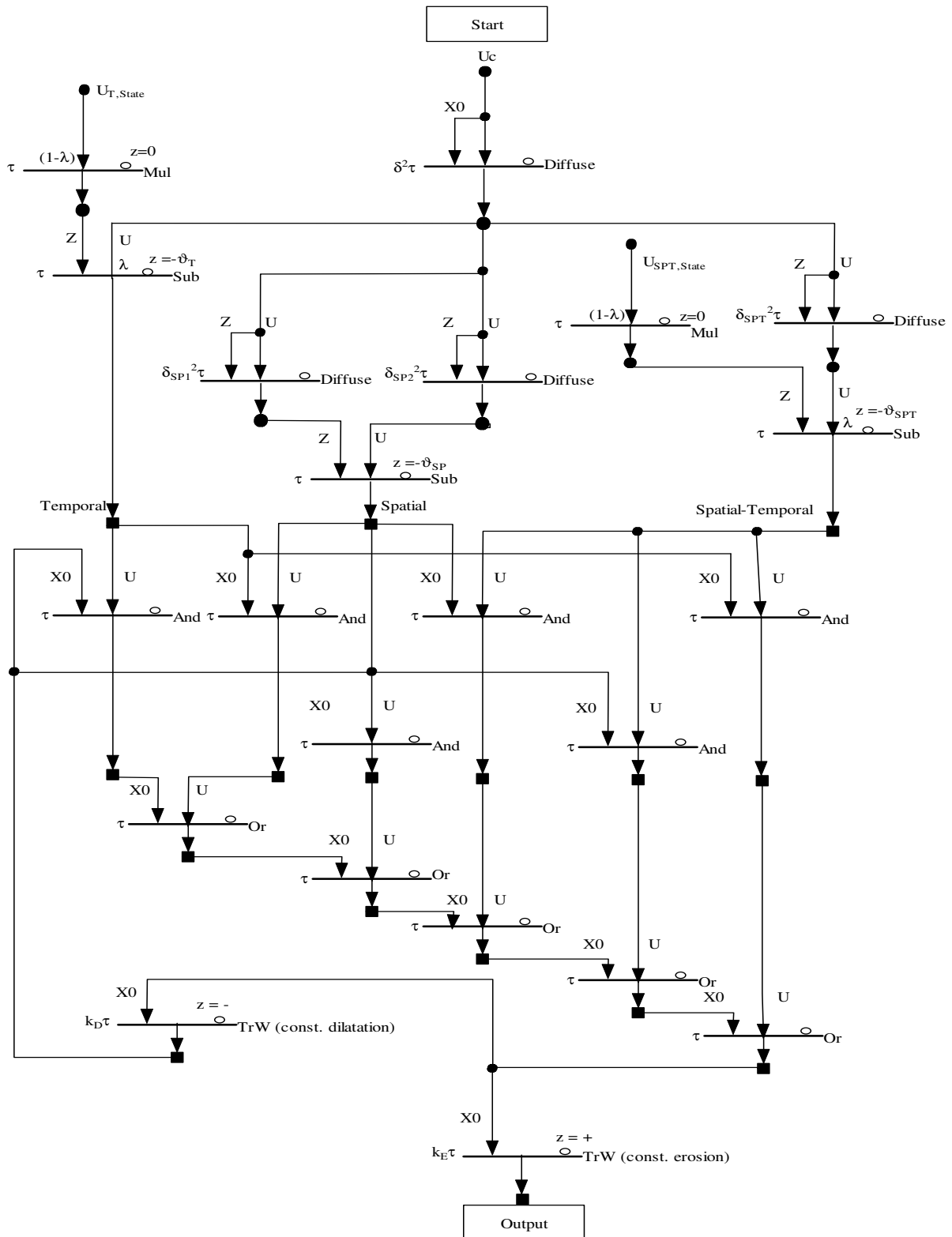


Figure 2.4 UMF diagram of the multichannel MTT algorithm front-end. This diagram shows the implementation on an idealized CNN VLSI chip or simulator.

2.3 The DSP-based MTT Algorithms

The combined estimation and data association problem of MTT has traditionally been one of the most difficult problems to solve. To describe these algorithms, we need to define some terms and symbols. A *track* is a state trajectory estimated from the observations (measurements) that have been associated with the same target. *Gating* is a pruning technique to filter out highly unlikely candidate associations. A *track gate* is a region in measurement space in which the true measurement of interest will lie accounting for all uncertainties with a given high probability [25]. All measurements within the gating region are considered candidates for the data association module. Once the existence of a track has been verified, its attributes such as velocity, future predicted positions and target classification characteristics can be established. The *tracking function* consists of the estimation of the current state of the target based on the proper selection of uncertain measurements and the calculation of the accuracy and credibility of the state estimate. The following factors degrade this estimate:

- model uncertainties due to target maneuvers and random perturbations and
- measurement uncertainties due to sensor noise, occlusions, clutter and false alarms (Figure 2.5 shows images with clutter and occlusions)

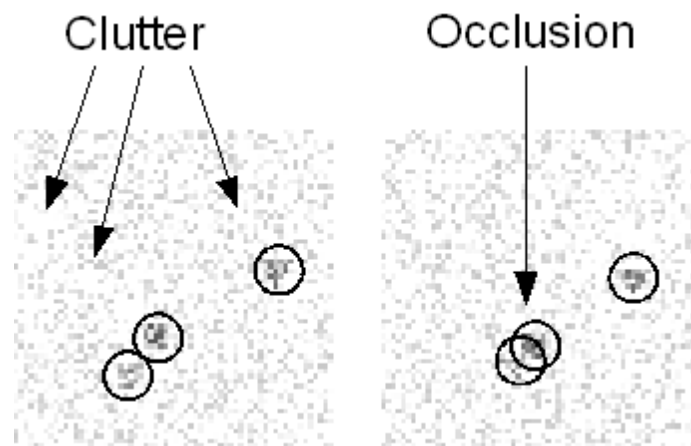


Figure 2.5 Clutter and occlusion in a simulated test video. Clutter can be anything that may be confused with a legitimate target, while occlusions occur when one target moves in front of the other, hiding it entirely (or partly) from view. The targets are circled for easier identification.

2.3.1 Data Association

Data association is the linking of measurements to the measurement origin such that each measurement is associated with at most one origin. For a set of measurements and tracks each measurement/track pair must be compared to decide if measurement i is related to track j . For m

measurements and n tracks, this means $m*n$ comparisons, and for each comparison multiple hypotheses may be made. As n and m increase the problem becomes computationally intensive. Additionally, if the sensors are in an environment with significant noise and many targets, then the association becomes very ambiguous.

There are two different approaches to solving the data association problem: (i) deterministic (assignment) – the best of several candidate associations is chosen based on a scoring function (accepting the possibility that this might not be correct) (ii) probabilistic (Bayesian) association – use classical hypothesis testing (Bayes' rule), accepting the association hypothesis according to a probability of error, but treating the hypothesis as if it were certain. The most commonly used deterministic assignment algorithms are the following:

Nearest Neighbor (NN) – the measurement closest to a given track is assigned in a serial fashion. It is computationally simple but is very sensitive to clutter.

Global Nearest Neighbor (GNN) – the assignment seeks a minimal solution to the summed total distance between tracks and measurements. This is solved as a constrained optimization problem where the cost of associating the measurements to tracks is minimized subject to some feasibility constraints. This optimization can be solved using a number of algorithms, such as the JVC (Jonker-Volgenant-Castanon) [28] algorithm, the auction algorithm [29] and signature methods [30]. These are all polynomial time algorithms.

The most commonly used probabilistic algorithms are the following:

Multihypothesis Tracking (MHT) [24],[25] – the MHT is a multi-scan approach that holds off the final decision as to which single observations are to be assigned to which single track. This is widely considered the best algorithm but is also the most computationally intensive ruling out real-time implementation on our architecture.

Probabilistic Data Association Filters (PDAF) – the PDAF technique forms multi-hypotheses too after each scan, but these are combined before the next scan of data is processed. Many versions of this filter exist, the PDA for single tracks, Joint PDA (JPDA) for multiple tracks, Integrated PDAF etc. [25],[31].

Based on data in the literature [25], we decided to work with deterministic assignment algorithms because they are high performance with calculable worst-case performance since they have a computational complexity of $O(n^3)$ (where n is the number of tracks and measurements) which was essential given our real-time constraints. We also restricted ourselves to the so-called 2-D assignment problems where the assignment depends only on the current and previous measurements (frames). The data assignment algorithms perform so-called *unique* assignment,

where each measurement is assigned to one-and-only-one track as opposed to *non-unique* assignment, when a measurement may belong to multiple tracks. We implemented two types of assignment algorithms a NN approach and the JVC algorithm. Since non-unique assignment would be very useful in certain situations such as occlusions, we modified the NN algorithm and added a non-unique assignment mode to it.

2.3.2 2-D Assignment Algorithms

Of the two algorithms we implemented, the NN algorithm is the faster one and for situations without clutter, it works adequately. It can be run in unique assignment mode, where each track is assigned one and only one measurement (the one closest to it) and in non-unique assignment mode, when all measurements within a track's gate are assigned to the track which makes it possible handle cases of occlusion.

The JVC algorithm is implemented as described in [28]. It seeks to find a unique one-to-one track to measurement pairing as the solution \hat{x}_{ij} to the following optimization problem:

$$\min \left(\sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij} \right) \quad (2.2)$$

$$\sum_{i=1}^n x_{ij} = 1, \sum_{j=1}^n x_{ij} = 1 \quad (2.3)$$

$$0 \leq x_{ij} \leq 1 \quad \forall i, j \quad (2.4)$$

Where n is the number of tracks and measurements (it is easy to generalize the algorithm if there are more measurements than tracks), $i, j=1 \dots n$, c_{ij} is the probable cost of associating measurement i with track j calculated based on the distance between the track and the measurement and x_{ij} is a binary assignment variable such that

$$x_{ij} = \begin{cases} 1 & \text{if } j \text{ is assigned to } i \\ 0 & \text{otherwise} \end{cases} \quad (2.5)$$

The JVC algorithm consists of two steps, an auction-algorithm-like step [29] then a modified version of the Munkres algorithm [32] for sparse matrices.

Our experiments indicate that the JVC algorithm is indeed superior to the nearest neighbor strategy while only affecting the execution time marginally.

2.3.2.1 The Utilized Distance Measures

The assignment cost c_{ij} in equation (2.2) may be calculated in many different ways. The most straightforward way is to simply let the cost matrix be the Euclidian distance matrix:

$$C_e = \sqrt{(M-L)^T (M-L)} \quad (2.6)$$

where M is the matrix of measurement vectors, L is the matrix of target positions for the current frame and C_e is the resulting cost matrix.

Another measure frequently used in tracking applications (a computer vision in general) is the Mahalanobis or statistical distance, which takes into account the correlations of the measurement vector data set, and is scale-invariant, i.e. not dependent on the scale of measurements. This property is very helpful, when the measurement vectors contain other coordinates besides distances, such as the features described in the next paragraph. The definition for the Mahalanobis distance is:

$$C_m = \sqrt{(M-L)^T \text{cov}(M)^{-1} (M-L)}. \quad (2.7)$$

If the covariance matrix is the identity matrix then the Mahalanobis distance is the same as Euclidean distance.

An interesting possibility is the inclusion of feature measures beside position coordinates into the measurement vector of each target. Features such as average grayscale intensity, major and minor axis can help to increase the assignment cost (thereby directly increasing the tracking accuracy) more easily between targets that are located near each other but are very dissimilar in one of these features.

2.3.3 Track Maintenance

We have devised a state machine for each track for easier management of a track's state during its lifetime. Each track starts out in the 'Free' state. If there are unassigned measurements after an assignment run, the remaining measurements are assigned to the available 'Free' tracks and they are moved to the 'Initializing' state. If in all of the next i frames the 'Initializing' tracks are assigned measurements, they become 'Confirmed'; otherwise, they are deleted and reset to 'Free'. If a 'Confirmed' track is not assigned any measurement in a frame, the track becomes 'Unconfirmed'. If during the next ' v ' frames it still does not get a measurement, it becomes 'Free', i.e. the track is deleted. Figure 2.6 shows the state machine.

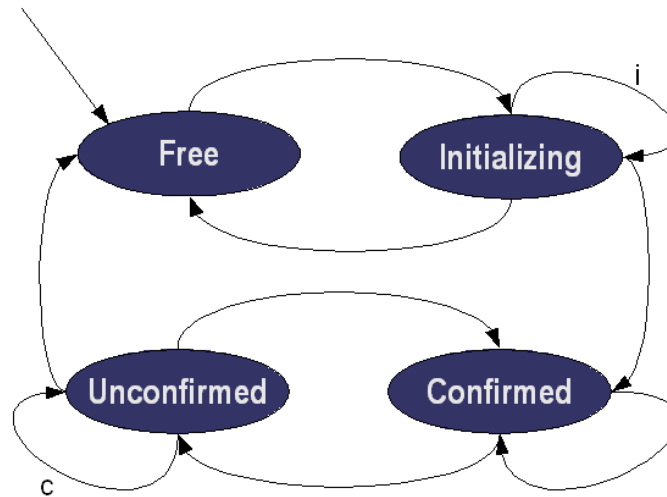


Figure 2.6 The track maintenance state machine of the MTT algorithm. The state machine starts from the Free state; ‘ i ’ and ‘ c ’ are two parameters for the number of initialization measurements and the number of unconfirmed frames respectively.

2.3.4 State Estimation

State estimation filters are a research area all by themselves with wide ranging applications. Since the goal of this research was not the development of new state estimation approaches, we decided to use method that would fit the following criteria:

- The estimation method must be relatively light in terms of computational burden, since we are planning to use it in a real-time system
- The filter must be easily tunable; i.e. the number of parameters used to tune the filter must be low. This is advantageous, because the tuning of these parameters is usually not trivial.
- The filter must complex enough for the motion estimation of maneuvering targets

It is obvious, that these are conflicting criteria, since estimating complex maneuvers will require more parameters, but we realized, that the faster our tracking systems, the less complex the target maneuvers seem to be, since our sampling frequency is higher.

Considering everything, we decided to implement two types of filters: a fixed-gain state estimation filter and a multiple model filter with fixed multiple models.

2.3.4.1 Fixed-gain State Estimation Filters (α - β and α - β - γ Filters)

These filters are also called time-invariant Kalman filters, or α - β - γ filters [24]. The main difference between a “regular” Kalman filter and time-invariant Kalman filters is that in a time-invariant filter, plant variations through time are accommodated by modeling them as noise.

This is often the most realistic assumption if the variations are unknown to system designer or user in advance. Additionally, the filter equations are simpler, requiring less computational resources, but the tradeoff is that the estimation will not be accurate when the underlying assumptions (that the system is a kinematic system, see the next paragraph) are not met.

We will briefly describe the algorithm behind these filters. Linear dynamical systems with time-invariant coefficients in their state transition and measurement equations lead to simpler optimal estimation techniques than are needed for the time-varying case. The state estimation covariance and filter gain matrices achieve steady-state values that can often be computed in advance. Two common time-invariant systems are constant-velocity and constant-acceleration systems, so called *kinematic systems*.

Let us assume a constant velocity model: starting with some initial value, the object's velocity evolves through time by process noise of random accelerations, constant during each sampling interval and independent. With no process noise, the velocity is constant; process noise can be used to model unknown maneuverings of a non-constant velocity target. The cumulative result of the accelerations can in fact change the object's velocity arbitrarily much, so we model a maneuvering object as one with high process noise. We assume position measurements are only available, subject to measurement noise of constant covariance. Clearly, the more that is known a priori about the motion the better the predictions will be.

Assume the object state (its position and velocity) evolves independently in each of the (X ; Y ; Z) dimensions. For instance, in the Y dimension, it evolves according to

$$\mathbf{y}(k+1) = \mathbf{F}_y \mathbf{y}(k) + \mathbf{v}(k) \quad (2.8)$$

where

$$\mathbf{F}_y = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \quad (2.9)$$

for sampling interval Δt , error vector $\mathbf{v}(k)$, and $\mathbf{y} = [Y, \dot{Y}]^T$. The equations for the other two spatial dimensions are similar, and in fact have identical \mathbf{F} matrices. Thus for the complete object state $\mathbf{x} = [X, \dot{X}, Y, \dot{Y}]^T$, \mathbf{F} is a (4x4) block-diagonal matrix whose blocks are identical to \mathbf{F}_y . The error vector $\mathbf{v}(k)$ can be described with a simple covariance structure: $\mathbf{E}(\mathbf{v}(k)\mathbf{v}^T(j)) = \mathbf{Q}\delta_{kj}$.

The α - β filter for state prediction has the form

$$\hat{\mathbf{x}}(k+1|k+1) = \hat{\mathbf{x}}(k+1|k) + \begin{bmatrix} \alpha \\ \beta / \Delta t \end{bmatrix} [\mathbf{z}(k+1) - \hat{\mathbf{z}}(k+1|k)] \quad (2.10)$$

Where $\hat{\mathbf{x}}(k+1|k+1)$ is an updated estimate of \mathbf{x} given $\mathbf{z}(k+1)$, the measurement at time $k+1$. Here we assume that $\mathbf{z}(k+1)$ consists of the two state components $(X; Y)$ (but not (\dot{X}, \dot{Y})). The state estimate is a weighted sum of a state $\hat{\mathbf{x}}(k+1|k)$ *predicted* from the last estimate to be $\mathbf{F}\hat{\mathbf{x}}(k|k)$ and the *innovation*, or difference between a predicted measurement and the actual measurement. The predicted measurement $\hat{\mathbf{z}}(k+1|k)$ is produced by applying (here a trivial) measurement function to the predicted state.

The α - β filter is a special case of the Kalman filter. For our assumptions, the optimal values of α and β can be derived (see [25] for details) and depend only on the ratio of the process noise standard deviation and the measurement noise standard deviation (which can be approximated based on a priori information). This ratio is called the object's *maneuvering index* λ , and with the piecewise constant process noise we assume,

$$\alpha = -\frac{\lambda^2 + 8\lambda - (\lambda + 4)\sqrt{\lambda^2 + 8\lambda}}{8} \quad (2.11)$$

and

$$\beta = \frac{\lambda^2 + 4\lambda - \lambda\sqrt{\lambda^2 + 8\lambda}}{4}. \quad (2.12)$$

The state estimation covariances can be found in closed form as well, and are simple functions of α , β , and the measurement noise standard deviation.

The α - β - γ filter is like the α - β filter only based on a uniform acceleration assumption. Thus, it makes a quadratic prediction instead of a linear one. Broadly, it tends to be more sensitive to noise but better able to predict smoothly varying velocities. Its equation is the following:

$$\hat{\mathbf{x}}(k+1|k+1) = \hat{\mathbf{x}}(k+1|k) + \begin{bmatrix} \alpha \\ \beta / \Delta t \\ \gamma / \Delta t^2 \end{bmatrix} [\mathbf{z}(k+1) - \hat{\mathbf{z}}(k+1|k)] \quad (2.13)$$

With the maneuvering index λ defined as before, the optimal α and β for the case that the target experiences random small changes in acceleration (random jerks) are the same as before and the optimal $\gamma = \beta^2 / \alpha$.

2.3.4.2 Multiple Model Filter with Fixed Multiple Models

We assume that there is *one* correct model for the process, and that the model is fixed or does not change over time, however we do not know what that model is. Over time, as the filter

reaches a steady state, we want to converge on a choice for the single most likely model. For this approach let us assume that the *correct* model M is one of r possible known fixed models,

$$M \in \{\boldsymbol{\mu}_j\}_{j=1}^r. \quad (2.14)$$

We can use the following conditional probability density function as an indicator of the *likelihood* of a measurement \mathbf{z} at step k :

$$f(\mathbf{z}|\boldsymbol{\mu}) = \frac{1}{(2\pi|\mathbf{C}_\mu|)^{n_\mu/2}} e^{-\frac{1}{2}(\mathbf{z}-\mathbf{H}_\mu\mathbf{x}_\mu)^T \mathbf{C}_\mu^{-1}(\mathbf{z}-\mathbf{H}_\mu\mathbf{x}_\mu)} \quad (2.15)$$

where

$$\mathbf{C}_\mu = \mathbf{H}_\mu \mathbf{P}_\mu^- \mathbf{H}_\mu^T + \mathbf{R}_\mu \quad (2.16)$$

We have omitted the k subscript for clarity. Note again that the state vector \mathbf{x}_μ and error covariance matrix \mathbf{P}_μ^- are the *a priori* (predicted) versions at step k , already computed at each filter prediction step using fixed gain state estimation filter time update equations. In other words, the density is conditioned on the model and all of its associated *a priori* (predicted) parameters.

Given a new measurement \mathbf{z} at time step k , and associated *a priori* state and covariance estimates from the fixed gain state estimation filter time update equations, we can use equation (2.15) to compute the recursive probability $p_j(k)$ that candidate model $\boldsymbol{\mu}_j$ is the correct model at that time:

$$p_j(k) = \frac{f(\mathbf{z}|\boldsymbol{\mu}_j)p_j(k-1)}{\sum_{b=1}^r f(\mathbf{z}|\boldsymbol{\mu}_b)p_b(k-1)}. \quad (2.17)$$

One would initialize $p_j(0)$ with some *a priori* estimate of the probability that $\boldsymbol{\mu}_j$ is the correct model. For example, one could consider all models equally likely to begin with, and set

$$p_j(0) = \frac{1}{r}, j = 1, 2, \dots, r. \quad (2.18)$$

Note that $f(\mathbf{z}|\boldsymbol{\mu}_j)$ and $p_j(0)$ are scalars, and at every time step k ,

$$\sum_{j=1}^r p_j(k) = 1. \quad (2.19)$$

The final combined or *model-conditioned* estimate of the state \mathbf{x}_k and error covariance \mathbf{P}_k are computed as a weighted combination of each candidate filter's *a posteriori* state and error covariance estimates. The weight for each candidate model is the model probability given by equation (2.15). The final model-conditioned state estimate is computed as

$$\widehat{\mathbf{x}}_k = \sum_{j=1}^r p_j(k) \hat{\mathbf{x}}_{k,\mu_j}, \quad (2.20)$$

and the final model-conditioned error covariance as

$$P_k = \sum_{j=1}^r p_j(k) \left[P_{k,\mu_j} + \boldsymbol{\varepsilon}_{\mu_j} \boldsymbol{\varepsilon}_{\mu_j}^T \right], \quad (2.21)$$

where $\boldsymbol{\varepsilon}_{\mu_j} = \widehat{\mathbf{x}}_k - \hat{\mathbf{x}}_{k,\mu_j}$.

The Algorithm

To begin with, one would instantiate r independent fixed-gain state estimation filters, one for each of the candidate models. Each of these filters would then be run independently, in parallel, with the addition of the necessary individual density and final probability computations.

At each time update, one would compute the normal a priori filter elements, and then:

1. Using the conditional density function given in equation (2.15), compute the likelihood of the current (actual) measurement \mathbf{z} for each candidate model μ_j
2. Using the previous probability $p_j(k-1)$ for each candidate model μ_j , use the recursive equation (2.17) to compute the probability that each individual model is correct
3. For each candidate model μ_j , compute the *a posteriori* (corrected) state estimate $\hat{\mathbf{x}}_{k,\mu_j}$ and error covariance P_{k,μ_j} using the fixed-gain filter measurement update equations.
4. Given each candidate filter's *a posteriori* (corrected) state estimate $\hat{\mathbf{x}}_{k,\mu_j}$, compute the final *model-conditioned* state estimate $\widehat{\mathbf{x}}_k$ using equation (2.20); and
5. If desired, given each candidate filter's *a posteriori* (corrected) error covariance estimate P_{k,μ_j} , compute the final model-conditioned error covariance P_k using equation (2.21).

As described in [25], the final mode-conditioned state estimate will converge to agree with one of the models, if one of the models is the correct one. In any case, it will converge to some constant mode represented by a fixed weighting of the individual multiple models.

If the actual mode is not constant, i.e. if the process can be switching or varying between different models, one can use various ad hoc methods to prevent convergence on a single mode.

For example:

- One can impose an artificial lower bound on the model probabilities,
- impose a finite memory (sliding window) on the likelihood function, or
- impose an exponential decay on the likelihood function.

A problem with using ad hoc means of varying the blending of fixed multiple models is that the error in the incorrect models (at any moment) can grow unbounded, i.e. the incorrect filters

can get lost. Thus, the filters might have to be re-initialized.

2.4 Automatic Parameter Tuning

A common problem for systems operating in unconstrained environments on visual input (such as a perimeter surveillance system) is that the input can vary substantially, but the systems are expected to work well within a range input conditions, without any intervention from the user. This type of flexibility requires a way to measure the quality of the system's output (i.e. the tracking quality) and to tune the front-end algorithms to improve that quality. Measuring the quality of multitarget tracking algorithms is difficult even if the ground truth is known [36], but if there is no ground truth, then at first glance, it seems impossible. However, to quote Tom Gilb "Anything you need to quantify can be measured in some way that is superior to not measuring it at all." [35], so there must be some good approximation of the tracking quality.

In most tracking applications, the number of tracked targets is very slowly changing and the rate of change in the number of targets is constant (or nearly constant). This means that if the number of tracks changes drastically (from 6 to 12, for example) in a very short period (in a few frames), it is highly probable that the input of the tracking system (the output of the multichannel image processing front-end) has deteriorated. This can happen for a number of reasons, but the two most common are that the lighting conditions have changed, altering the appearance of the target and that the target motion has changed.

2.4.1 Response to Lighting Changes

Changes in lighting conditions are handled by varying the integration time of the sensor. This is a very low-level adjustment independent of all other parameter-tuning actions and could be considered part of the functionality offered by the sensor even though it is implemented in software.

The first step is to measure the average gray value of the image for the current frame. If the difference between the desired average gray level G (usually 128) and the measured average (g) is larger than 10%, the integration time t is adjusted: If $|G - g| > 0.1 \cdot G$, then $t_{k+1} = (G - g) \cdot \lambda + t_k$

2.4.2 Response to Target Motion Changes

Differences in target motion from the user's a priori assumptions can create subtle errors in the image processing steps that limit the accuracy of the tracking results. We considered solutions to the tuning of three very important parameters in the multichannel front-end: the orientation of the front-end filters, the number of openings/closings during detection post

processing and the number of dilations during the prediction step.

Tuning the orientation parameter (ϕ) of the front-end filters online can help to tackle a frequent phenomenon in tracking applications: even though the approximate speed (or speed range) of the anticipated targets is known a priori, the movement direction is not. This means that although our change enhancing front-end filters possess the ability to be more sensitive to motion in a certain direction (using the orientation parameter (ϕ) of the filter described in 2.2.1), based on knowledge available a priori, this capability cannot be leveraged. As the system is working online, however, the output of the tracking systems does contain the required direction data in the form of motion vectors for the individual targets. The algorithm is as follows:

1. Calculate the average motion vector for all targets and determine the direction of the motion (ϑ)
2. Set the orientation (ϕ_{k+1}) for the next frame according to the following formula:

$$\phi_{k+1} = \alpha \cdot \phi_k + (1 - \alpha) \cdot \vartheta \quad \text{where } 0 \leq \alpha \leq 1$$

The parameter α serves to control the response speed of the system to changes in target movement direction: if α is close to 1, change is very slow, if α is close to 0, then change is very fast.

The post processing of the detection images is also an important step in getting the image ready for feature extraction. This step serves as a filter to smooth out target boundaries or enhance target signatures. The filtering consists of the N_D number of morphological opening or closing steps, depending on a priori considerations regarding the tracking scenario. The number of these steps is a parameter that must be consistent with the location of the targets: if two targets get too close to each other for example, too many closing operations could cause their signatures to merge and cause errors in the feature extraction phase. It is possible to modify the number of morphological operations based on the distances of the targets relative to each other. If two targets are so close, that the N_D current morphological closings would merge them, then N_D is decreased. If, however, the average distance between the targets is sufficiently large, then N_D may be increased, if it is less than the optimal amount specified by a human observer (and known a priori). This algorithm requires the calculation of the predicted distances of each pair of targets. Since the predicted locations of the targets are available from the state estimation module, only $O(n^2)$ extra calculations are necessary, where n is the number of active targets in the previous frame.

Optimizing the image-based prediction of future target positions – which can be considered a rough gating method to effectively filter out clutter from the input images – is possible through

the parameter that controls the number dilations (N_D) during the prediction step. Changing this parameter proportionally to the average speed of the targets allows the system consider a wider search area for the possible target locations, but still use the image-base gate to reduce clutter.

The algorithm to calculate N_D for the $k+1^{th}$ frame (N_D^{k+1}) is as follows:

1. Calculate the maximum speed (v_{max}) of the targets in a given frame
2. Let $N_D^{k+1} = \text{round}(N_D^k \cdot \alpha + (1 - \alpha) \cdot v_{max})$ where $0 \leq \alpha \leq 1$ and $N_D \in \mathbb{Z}$

Like before, α serves to control the response speed of the system to changes in target speed to

smooth out jitter. The formula can be so simple, because $[v_{max}] = \frac{\text{pixels}}{\text{frame}}$.

2.4.3 Response to Channel Output Corruption

The result of the algorithms described in the previous sections is that the detection output image remains qualitatively constant if there are no abrupt (unanticipated) changes or errors in the system. However, in an unconstrained environment, it is possible that one or more channel outputs are corrupted beyond repair. It is possible to detect this, because the degradation of the outputs of one of the parallel channels causes the degradation of the detection image. The challenge is then to determine which channel was corrupted and remove that channel from further processing through the channel interaction matrix to prevent the propagation of the error through the system. This is similar to the strategy a human would use to tune the tracking system in a given scenario. An analogy for the algorithm's main idea would be that of a choir: a single false voice can ruin the sound of the whole choir, so the sources for these voices must be eliminated.

The tuning algorithm makes the following strong assumptions (which are typically valid in tracking applications):

1. The parameters of the front-end channels are tuned to detect targets of interest (for example the temporal channel is tuned to a given target speed range)
2. The number of targets or the rate of change in the number of targets is constant or stays within a very narrow interval
3. The structure of the interaction matrix (the selection of logic functions used for row and column operations) is tuned for the given application, therefore it will not be changed.

The insight of the algorithm is that in order to find the channel that contributed the erroneous tracks, it is enough to compare the channel output images with the detection image, without running the feature extraction routines on the images. This means that the running time

of the algorithm is not dependent on the number of erroneous tracks; in effect, the runtime is constant.

Given the above assumptions, the adaptation algorithm is as follows:

1. Let N_k be the number of live tracks at frame k ,
2. If $|N_k - N_{k-1}| > \delta$, then continue, else stop.
3. Let S , ST and T be the binary channel output images for the spatial, spatio-temporal and temporal images respectively, and D the detection image used for feature extraction.
4. Let A , or Age of target t be the number of frames a given target was present during tracking so far. Find and tag all targets, where: $A_t < \tau$.

5. Create a seed image by setting the centroid pixels of the tagged targets, let this be $Seed$.
6. Generate an image containing only the erroneous targets (T_{Seed}), using the RECONSTR template: $T_{Seed} = \text{Re } c(D, Seed)$

7. Calculate the channel error images:

$$\begin{aligned} E_S &= S \otimes T_{Seed} \\ E_{ST} &= ST \otimes T_{Seed} \\ E_T &= T \otimes T_{Seed} \end{aligned}$$

8. Count the number of white pixels on the channel error images, let these be C_S , C_{ST} , C_T .

$$C = \frac{C_S + C_{ST} + C_T}{3}$$

9. If $|C_S - C| < \epsilon$ and $|C_{ST} - C| < \epsilon$ and $|C_T - C| < \epsilon$, then the errors are coming from all channels, so the filters must be tuned manually. Otherwise, find $\max(C_S, C_{ST}, C_T)$, and choose the channel with them maximum error count.
10. If more than one channel is active (the number of 1-s in the channel interaction matrix is greater than 2), then zero out the row and column of the channel in the channel interaction matrix; otherwise stop.

2.5 Experiments and Results

During algorithmic development, we targeted a real-time application for the ACE-BOX [19] platform. The ACE-BOX is a PCI extension stack that contains a Texas Instruments TMS320C6202B-233 DSP and either an Ace4k or an Ace16k CNN-UM chip in addition to 16MBs of onboard memory. The Ace4k chip is a 64x64, single-layer, nearest-neighbor CNN-UM implementation with 4 LAMs (Local Analog Memory for grayscale images) and 4 LLMs (Local Logical Memory for binary images) [24]. We also experimented with the newer Ace16k chips that have 128x128 cells, an optical input and 2 LLMs and 8 LAMs. The Ace4k chip was

manufactured on a 0.5 micron process, while the Ace16k on 0.35. We hosted the Ace16k chip on the Bi-i platform developed by Analogic Ltd (see the Section 1.4 for details).

2.5.1 Algorithm Accuracy Measurements

To validate our choice of using multiple interacting channels during the image-processing phase of the system vs. using a single channel, we ran measurements on five image sequences containing rapidly moving and maneuvering targets in images with differing amounts of noise and clutter. All videos were manually tracked by a human viewer to obtain reference measurements for the target positions in each frame. These positions were compared to the measurements given by the multi-channel front-end and each of the constituent channels as well. The mean square position error was calculated for each of the target locations, and averaged for each image sequence. The relative error compared to the best performing channel was also calculated, since this is a good indicator of the overall performance of a channel under varying conditions. The results of these experiments are shown in Figure 2.7.

The use of the multi-channel architecture allows the system to be able to process markedly different inputs within the same framework and achieve acceptably low error levels. If an “oracle” (a different system, or a human) can provide quantitative input on the performance of each channel, then the system can adapt (through changing values in the interaction matrix) to give more weight to the best performing channel. If no such information can be acquired, the system will still perform relatively well (see the relative errors in Figure 2.7), often very close to the best performing channel.

	Multi		Threshold		Temporal		Spatial		Spatio-temporal	
	MSE	Relative	MSE	Relative	MSE	Relative	MSE	Relative	MSE	Relative
Sequence1	1.36	0.00%	2.13	56.62%	7.01	415.44%	1.73	27.21%	1.83	34.56%
Sequence2	2.07	81.58%	24.84	2078.95%	1.14	0.00%	34.26	2905.26%	10.05	781.58%
Sequence4	0.82	0.00%	0.94	14.63%	1.1	34.15%	0.92	12.20%	0.99	20.73%
Sequence5	1.34	67.50%	0.8	0.00%	6.04	655.00%	0.95	18.75%	0.89	11.25%
Sequence6	1.43	1.42%	1.42	0.71%	10.28	629.08%	4.58	224.82%	1.41	0.00%
<i>Avg. relative error:</i>		30.10%		430.18%		346.73%		637.65%		169.62%

Figure 2.7 Accuracy comparison of different front-end channels and the multi-channel arrangement. The best performing channel is highlighted in bold type for each sequence. The mean square position error (MSE) was calculated for each of the target locations, and averaged for each image sequence and the relative error was compared to the best performing channel. Observe that the output of the multi-channel architecture is – on average – the best performer in these sequences.

We also tested the developed algorithms on several artificially generated sequences in addition to video clips recorded in natural settings (such as a flock of birds flying). We hand tracked some of these videos to be used as ground truth references for assessing the quality of

the tracking algorithms as measured at the output of the complete MTT system. Figure 2.8 shows a few sample frames from the “birds” clip along with the detection images generated by the multi-channel front-end. This sequence contains 68 frames of seagulls moving rapidly in front of a cluttered background.



Figure 2.8 Sample consecutive frames from a test video and the corresponding detection maps of the system. The input video shows birds flying in front of a cluttered background (birds are circled in red on the input frames). Since the birds, leaves and branches of the trees are all moving, detecting the targets (birds) is very difficult and must make full use of the capabilities of the multi-channel front-end (such as the ability to filter based on object size and object speed).

The accuracy of image processing also depends heavily on the noise characteristics of the input image sequence. To measure this, we developed a program to generate artificial image sequences, which allowed us to specify carefully the kinematic properties of the moving targets. We could also mix additive noise to the generated images to study the noise sensitivity of the system. To describe noise levels, we defined the signal to noise ratio (SNR) and peak signal to noise ratio (PSNR) according to the definitions commonly used in image compression applications [33]:

$$SNR = 10 \log_{10} \left\{ \frac{\sum_{i=1}^N \sum_{j=1}^M f(i, j)^2}{\sum_{i=1}^N \sum_{j=1}^M [f(i, j) - f'(i, j)]^2} \right\} \quad (2.22)$$

$$MSE = \frac{1}{N \cdot M} \sum_{i=1}^N \sum_{j=1}^M [f(i, j) - f'(i, j)]^2 \quad (2.23)$$

$$PSNR = 10 \log_{10} \left\{ \frac{255^2}{MSE} \right\} \quad (2.24)$$

where M and N are the dimensions of the image in pixels and $f(i,j)$ and $f'(i,j)$ are the pixel values at position (i,j) in the original image and the noisy image, respectively.

Figure 2.9 shows the results of our analyses. We did not calculate the SNR and $PSNR$ values for the natural image sequences since there was no reference image available to compare to our inputs. The measurement errors were obtained by calculating the distance between a hand tracked / generated target reference position and the output of the multi-channel front-end. The modeling errors were calculated by injecting the reference target positions into the system and measuring the tracking error, while the tracking errors were the difference between the target reference positions and the output of the whole MIT system.

As can be seen from the data in Figure 2.9, the tracking error is always lower than the measurement and modeling error combined, which suggests that these errors cancel each other out somewhat. The performance of the multi-channel front-end is very good in cases where the images are corrupted with high levels of noise, which is due to the noise suppression capabilities of DoG type filters. Lastly, the magnitude of the overall tracking errors is within two pixels for these sequences.

Content Type	Motion Type	SNR	PSNR	Tracking Error	Measurement Error	Modeling Error
Natural	maneuvering and linear, constant speed	N/A	N/A	2.07	1.69	0.49
Natural	stochastic, overlapping, varying speed	N/A	N/A	1.36	0.92	0.82
Generated	maneuvering and linear, constant speed	14.01	17.56	0.82	0.52	0.64
Generated	maneuvering and linear, constant speed	7.63	11.18	1.43	1.17	0.64
Generated	maneuvering and linear, overlapping, constant speed	18.35	20.50	1.34	0.71	0.72

Figure 2.9 Tracking accuracy and noise levels for sample videos. All errors are in pixels. Sub-pixel error values are the result of the sub-pixel accuracy of our state estimation and centroid calculation routines. Higher SNR and PSNR values show lower noise levels. All videos in the table are different image sequences, the 3rd and 4th however contain targets with the same motion properties moving on the same path, but with different image noise levels, which is why the measurement errors are different and the modeling errors are the same for the two sequences.

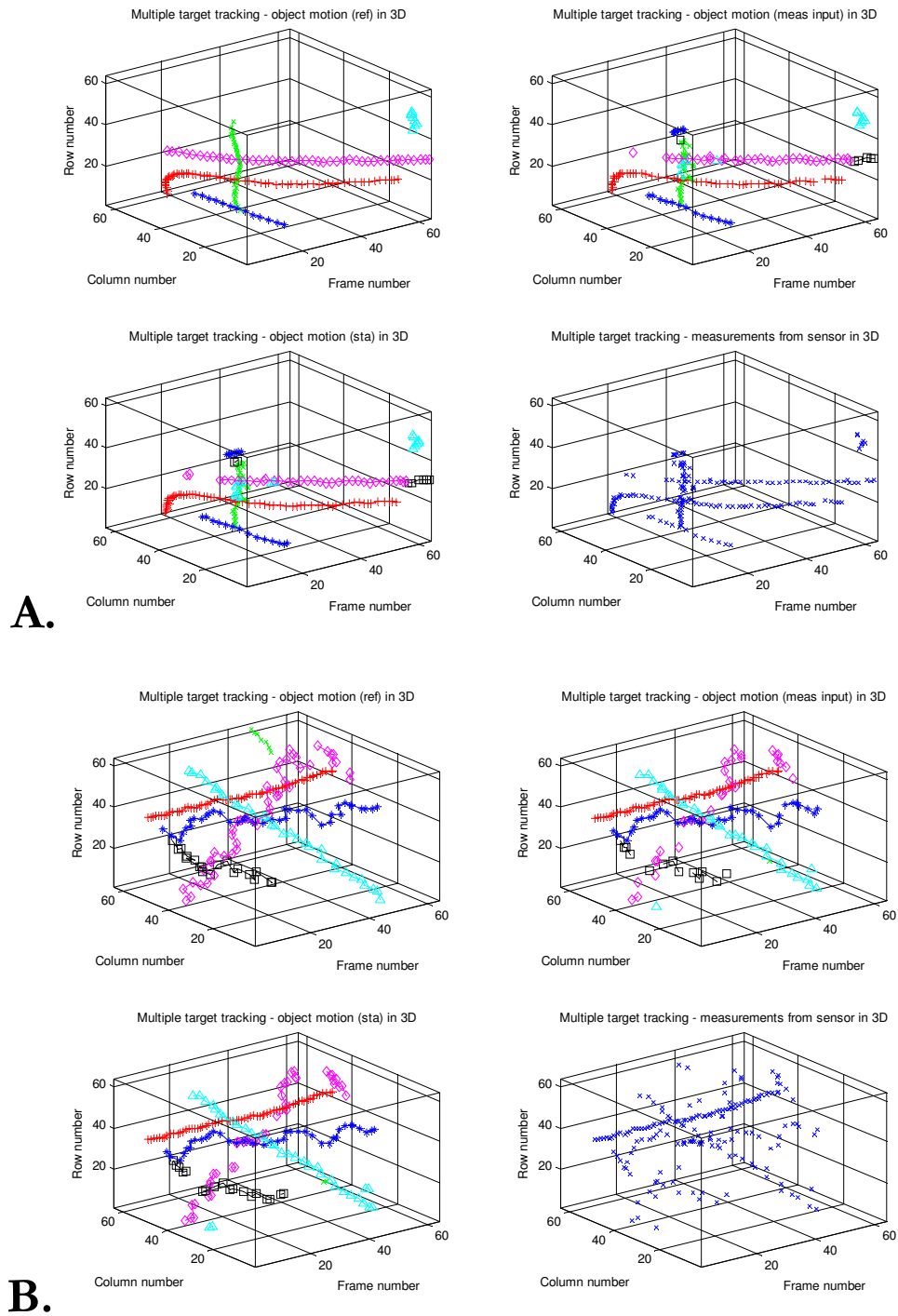


Figure 2.10 Target tracking results for two sample videos. The plots show the tracking results in 3D: two spatial dimensions that correspond to the coordinates in the videos, and a third temporal dimension, which corresponds to the frame number in the sequence. Target tracks are continuous lines in these 3D plots, with different gray levels signifying different tracks. The targets were hand tracked by human observers to generate reference target positions (upper left plots (‘ref’) in A and B). The track states (‘sta’, lower left plots in A and B) are the target location outputs of the MTT system. The ‘meas’ plots (upper right corner on A and B) show the outputs of the data-assignment subsystem. Observe that the kinematic state estimation algorithms smooth out some of the jitter in the direct measurements. Finally, the input

measurements (the coordinates of the centroids of the black blobs in the detection maps) generated by the multi-channel framework are shown in the lower right plots (in both A and B).

Figure 2.10 shows the results of running the system on two video flows (A and B) that contain targets which are maneuvering and sometimes move in front of each other, effectively stress testing the tracking algorithms. Sequence A. was artificially generated while sequence B. is a short natural video clip showing rapidly maneuvering targets with a priori unknown motion and trajectory. We hand tracked each frame of these video flows to facilitate a rigorous comparison of the system's performance against a human observer. It must be noted, that sometimes even we humans had trouble identifying targets in a frame without flipping back-and-forth between frames, which illustrates the need for temporal change detection in the image processing front-end.

We measured the performance of the system at three stages: the output of the multi-channel framework, the output of the data-assignment subsystem and the output of the whole MTT system. This enabled us to visualize and study the effect of different input sequences on various subsystems. We observed, for example, how the kinematic state estimators smooth out the target trajectories when fed the somewhat jittery data from the multi-channel front-end (this was expected and desired).

The measured MTT system outputs show that the system tracked the targets fairly well, although occlusion and missing sensor measurements have caused significant errors as the system merged tracks together and split others (this is common error in tracking systems). To address this issue, we are currently working on incorporating a more advanced state estimation algorithm to model target motion better and include a priori knowledge of target behavior.

2.5.2 Algorithm Performance Measurements

Before implementing the algorithms in C++, we first prototyped them in MATLAB using a flexible simulation framework based on the MatCNN simulator [12]. After the algorithms "stabilized", we ported them to work on the ACE-BOX hardware using the Aladdin Professional programming environment [19].

To enable better comparison of the Ace4k-based algorithm implementation with the pure DSP version, we coded the same algorithms in both cases. Since the data assignment and state estimation algorithms run on the DSP in both cases, only the operations in the multi-channel framework had to be coded for both platforms. We optimized the algorithms to run as fast as possible on each platform, using methods optimized for the platform's characteristics. For example, we used the optimized image processing routines provided by Texas Instruments

Image Processing Library to construct our multi-channel algorithm on the DSP. Figure 2.11 shows the running times of various steps of the algorithms for different parameter settings. The runs differed only in the number of opening/closing iterations applied to the images in the multi-channel front-end, since this is the most costly step of processing. These iterations smooth out the input binary maps to provide better inputs for further processing.

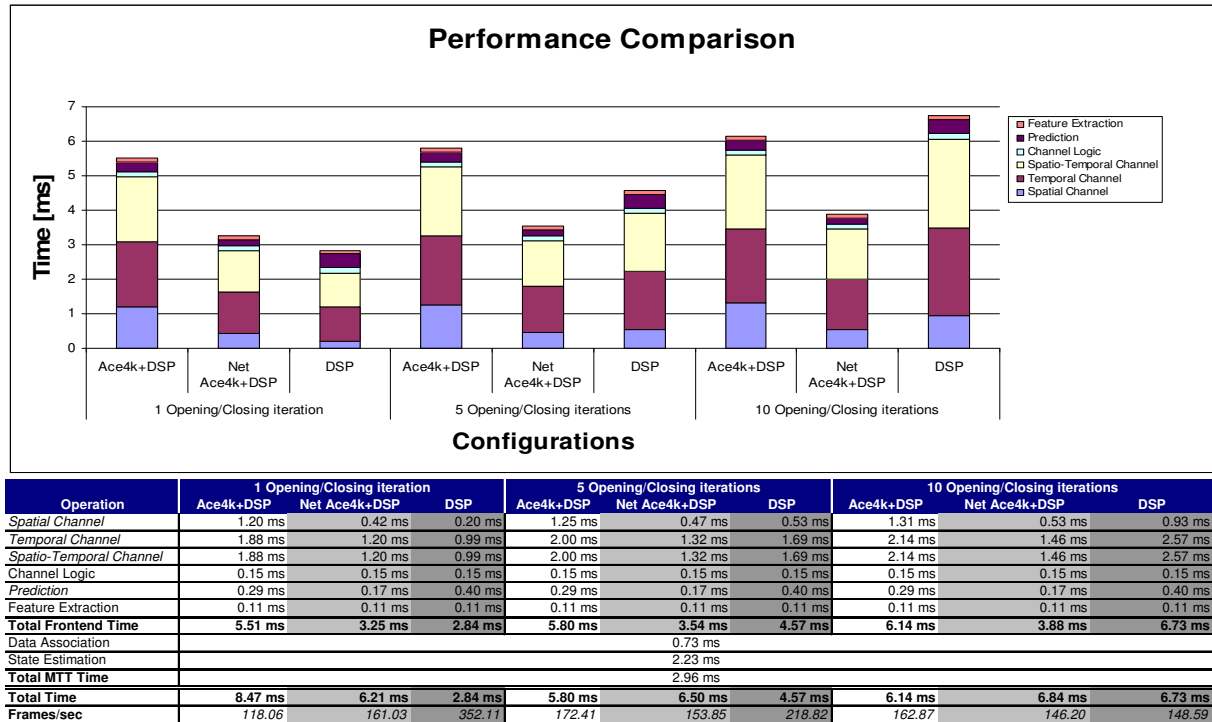


Figure 2.11 Running times for the various subtasks of the MTT system in different configurations. The 2 configurations were: the image processing steps of the multi-channel front-end running on the Ace4k and everything else running on the DSP (Ace4k+DSP and Net Ace4k+DSP column), and all algorithms running on the DSP (DSP column). The Net Ace4k+DSP column contains only the net computing time (without data transfers, which is very significant for the Ace4k). Observe that because of the data transfers, the Ace4k-DSP tandem is slower than DSP-only algorithms if the iteration count is small, but as the iteration count increases, the data transfer speed is balanced by the linear slowing down of the DSP. Using the Ace4k chip, net computing times are always close to, or significantly better than those using the DSP.

2.6 Discussion

During the interpretation of the performance data in Figure 2.11, it is important to note that the Ace4k chip was manufactured on a 0.5 micron process while the TMS320C6202B-233 DSP on a 0.15 micron process, which is a significant advantage for the DSP. Nonetheless, the results

for the multi-channel front-end performance tests highlight several important facts. First, the numbers indicate that the Ace4k has limited potential in practical image processing scenarios, because it is hampered by the slow data transfer speed of its bus and the limited number of onboard memories (4 LAMs and 4 LLMs). Since it does not have an optical sensor, data must be transferred from a DSP for processing, and frequently, partial results of the algorithms must be transferred back to the DSP for storage, because of the limited on-chip memory capacity of the chip. This is the reason why the DSP is faster than the Ace4k-DSP duo if the number of opening/closing iterations is small. As the iteration count increases, the transfer cost is balanced by the linear slowing down of the DSP, which is why the Ace4k becomes the clear winner at higher iterations.

We have some preliminary data of our work with the newer Ace16k processor. This chip is larger (128x128 vs. 64x64), but has a faster data bus, so data transfer times for native size grayscale images are about half that of the Ace4k. Unfortunately, the chip does not have a dedicated binary image readout mode, which slows down the readout of binary images to speeds about 10 times slower than the Ace4k.

The Ace16k has one other nice feature: a built-in resistive grid. The resistive grid can calculate diffused images in as low as 30ns, which enables the very rapid generation of DoG filtered images (which is just a difference of two diffused images). Our experiments indicate that using the resistive grid we can perform the front-end channel calculations about 4 times faster than the DSP for 128x128 sized images (including transfers).

Several lessons can be learned from the tests that have to be addressed to design a competitive topographic visual microprocessor. The clear advantage that topographic image processor have over conventional digital image processors is that all other things being equal, the processing speed remains essentially constant as the size of the array increases, while on DSPs, processing time increases linearly with the image area (which grows quadratically). Further advantages can be gained by using diffusion and trigger-wave based image-processing operators, which are very fast constant time operations on CNN-UM chips, but can only be approximated with iterative approaches on DSPs. However, to realize the full potential of these architectures, they must be designed with practical application scenarios in mind.

They have to feature focal plane input, so the initial images do not have to be transferred through the data bus. The transfer speed of the digital communication bus has to be increased by one, preferably two orders of magnitude, if we factor in the need for higher resolution images. Even though a topographic processor is capable of performing many operations at constant speeds (with respect to image size, as long as the image size is equal to, or smaller than the

number of processing units), some operations are much better suited to traditional DSP implementation (2D FFTs for example). Without a high-speed bus, the transfer of images itself becomes a significant bottleneck that negates the advantages of using a topographic processor in the first place.

2.7 Case Study: The Multitarget Framework in a Real-time Control Environment

The real time performance characteristics of the MTT algorithm framework lend themselves naturally to applications where the control of some physical device is required base on visual input. These applications demand control signals at a predictable rate from the algorithm processing the video input. With the model application, we wanted to verify and demonstrate the following capabilities:

- high frame rate operation
- real time control
- visually verifiable tracking accuracy

These abilities are highly desirable for any system relying on visual input for control purposes.

Our model application imitates a scenario, where a laser pointer has to follow and “tag” up to six rapidly moving targets by shining a laser on them. The targets are moving rapidly in a rectangular area within a plane and our MTT algorithm, which runs on a Bi-i camera, tracks them. The laser is connected directly to the Bi-i camera, and is controlled from the MTT software. A PC communicates with the Bi-i to provide visual confirmation of the tracking and to show an interface for setting algorithm parameters. Figure 1.1 shows the system and connections.

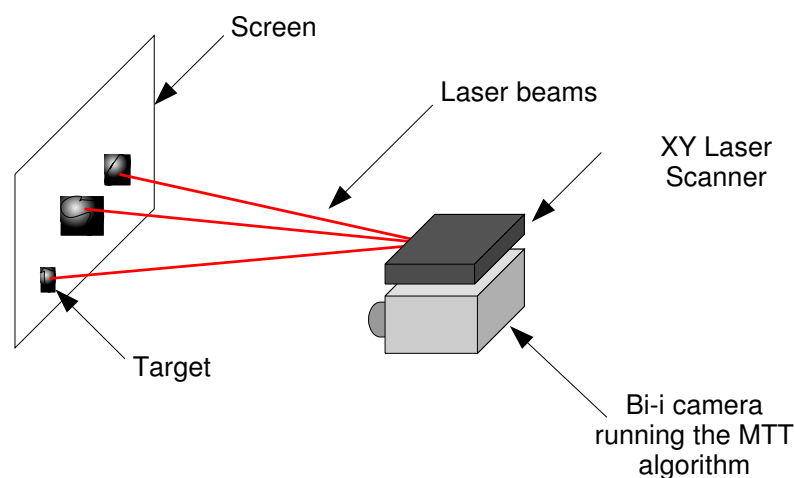


Figure 2.12 Bi-i Controlled Laser Scanner System

The laser positioning system consists of laser scanners in the X and Y planes and some additional driver circuitry. A laser scanner is a galvanometer with an attached mirror. When current is applied, the galvanometer's shaft rotates through part of a circle, when the current is removed, the shaft returns to the rest position. This gives the scanners the ability to scan a rectangle with the stationary laser source. The additional circuits contain digital-analog converters to provide the currents needed to drive the scanners and the logic devices used to interface to the Bi-i camera. The Bi-i camera communicates with the laser scanners through memory-mapped registers. Figure 2.13 shows the schematic diagram of Bi-i-based laser scanner setup, while Figure 2.13 shows the actual hardware itself.

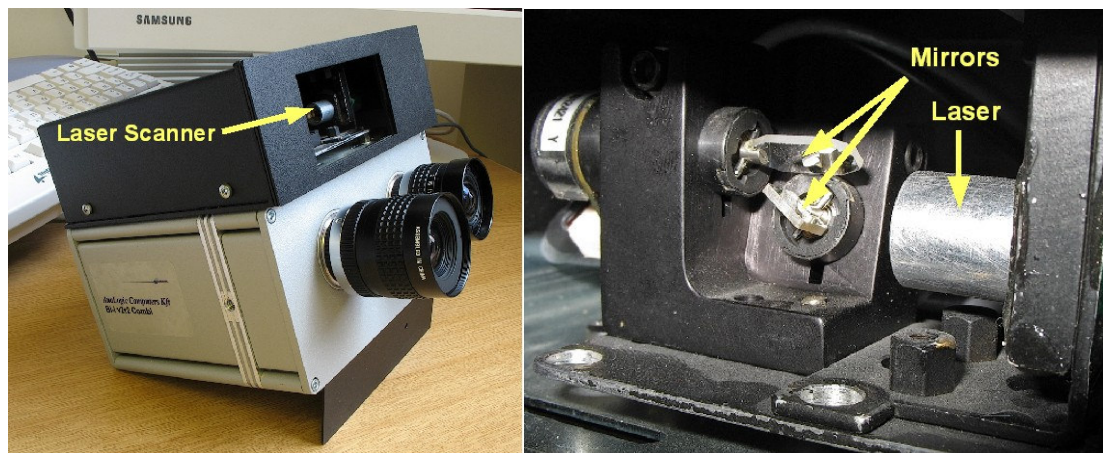


Figure 2.13 The Bi-i camera with the laser scanner attached. On the left image, the Bi-i camera is shown in silver, with the laser scanner assembly attached on top (black). The front window serves as a shutter for the laser light. The right picture shows a close-up of the mirror assembly inside the scanner.

We developed a small calibration routine to map the virtual 2D space of the tracking algorithm to the 2D plane of the environment in order to display the tracking results accurately. The routine displays – using the laser – an adjustable rectangular target area, which encloses the area where the laser will shoot. This has to be aligned with the projected image, but the aspect ratio of the rectangle can be arbitrary. The pixel coordinates received from the MTT routine are translated into this rectangular space, and then the desired mirror angles are calculated from the rectangular coordinates.

For demonstration and testing purposes, we used a moving target generator utility to generate short videos with rapidly moving targets. The utility enables us to specify flexibly the look, the path and the motion characteristics (speed and acceleration) of the individual targets, the complexity of the background and the noise level of the video.

We used an LCD projector to project these videos onto a reflective screen and set up the

laser tracking system to track the simulated targets on the screen. The projector was attached to a computer that was not running the tracking algorithms; its only purpose was to generate the target images.

The laser scanner system “tags” the targets by moving the laser light to a target, and staying at the given position for a few milliseconds. Since the laser has a noticeable warm up time, we switch on the laser light before the first target is highlighted and turn it off after last one was tagged. This does not create visible lines over the path of the laser, because the laser moves very fast. Figure 2.14 shows the laser scanner running on a typical video.

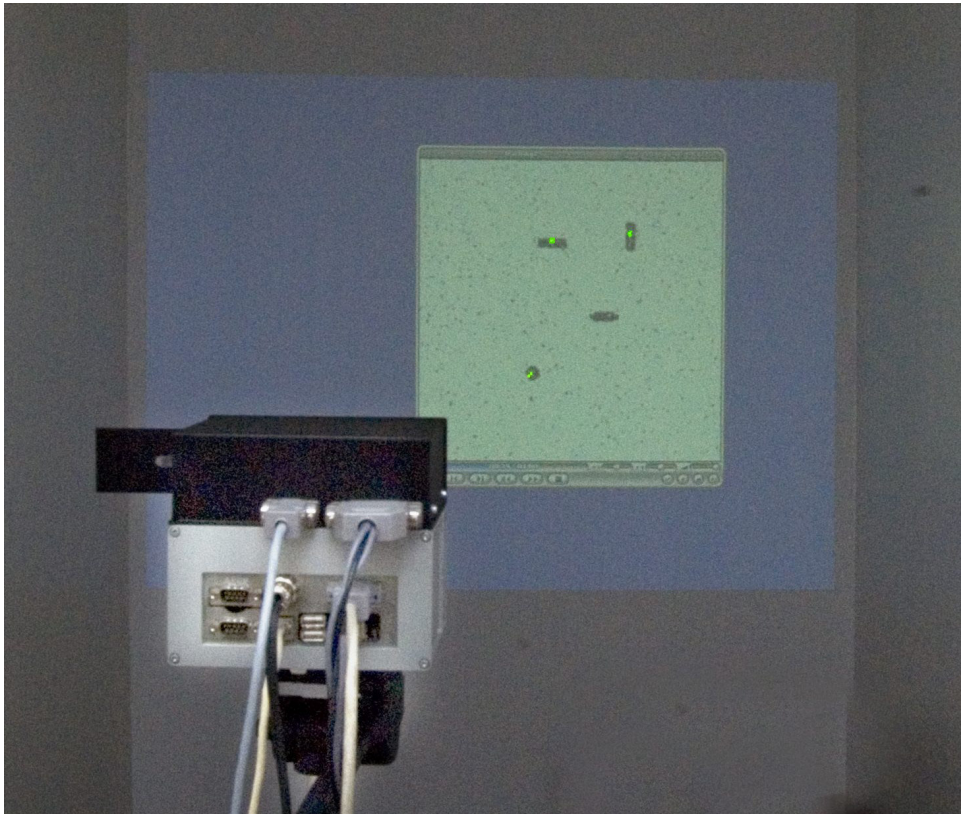


Figure 2.14 The laser scanner system in action

3 Analogic Preprocessing and Segmentation Algorithms for Offline Handwriting Recognition

In many industries, there is a substantial demand for the automatic processing of different types of handwritten materials. Obvious applications include the automatic indexing and processing of archived documents, forms, postal envelopes, notes etc. Even though human handwriting processing has gone through considerable improvement during the past decades, relatively well performing systems are only available for vertical applications. In these scenarios, the utilized vocabulary is very narrow and well defined, such as the recognition of postal addresses on envelopes and the recognition of medical prescriptions; or the writer has to learn a new writing style (graffiti alphabet on PDA-s) so the machine can interpret it.

The area of handwriting recognition consists of two completely different problems: online and offline handwriting recognition. Offline recognition is the reading of handwritten text sometime after the writer has created it. This means that the input of the recognition engine is a binary or grayscale image containing the handwriting. Online handwriting recognition is the interpretation of the human handwriting “real-time” as it is created. Input is usually with a special pen on an electronic notepad, which provides temporal information and trajectory data. The significantly larger amount of input data available makes online recognition an easier task than offline recognition. Our paper is concerned strictly with offline handwriting recognition because it does not require special input devices. The offline handwriting recognition task contains character recognition as a sub-problem that has been studied using CNN algorithms [37] and Gabor filters for hand-printed letters [38].

Producing reasonably well performing offline handwriting recognition systems has been an elusive goal for researchers because of two things: handwriting is a form of self-expression, and as such, the same words can be written in many different styles. Many handwritten words are also ambiguous and can only be recognized in context. This is especially true for the recognition of unconstrained texts, where the vocabulary may be arbitrarily large. The complete research area of offline handwriting recognition is very large, as it comprises the preprocessing of the input, recognition and post-processing of the results, and each of these tasks is an actively researched subject in its own right. This chapter is concerned with solving the preprocessing and segmentation tasks of an offline handwriting system without the use of a grammatical model or input from the character recognizer.

3.1 The Basic Structure of an Offline Handwriting Recognition System

Even though the many recognition systems try to solve the problem in different ways there is a general module structure used by all systems that stems from the properties of the problem itself [42], which is shown in Figure 3.1.

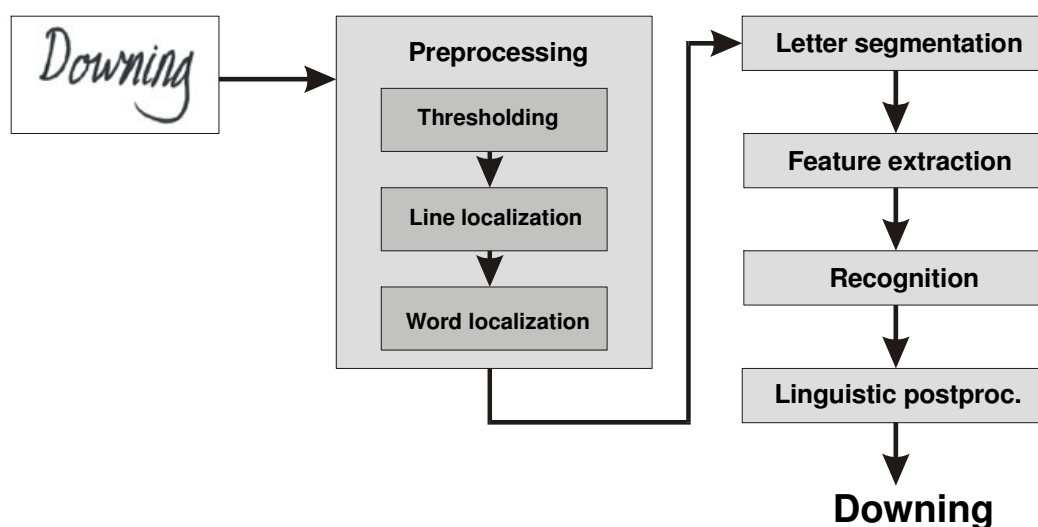


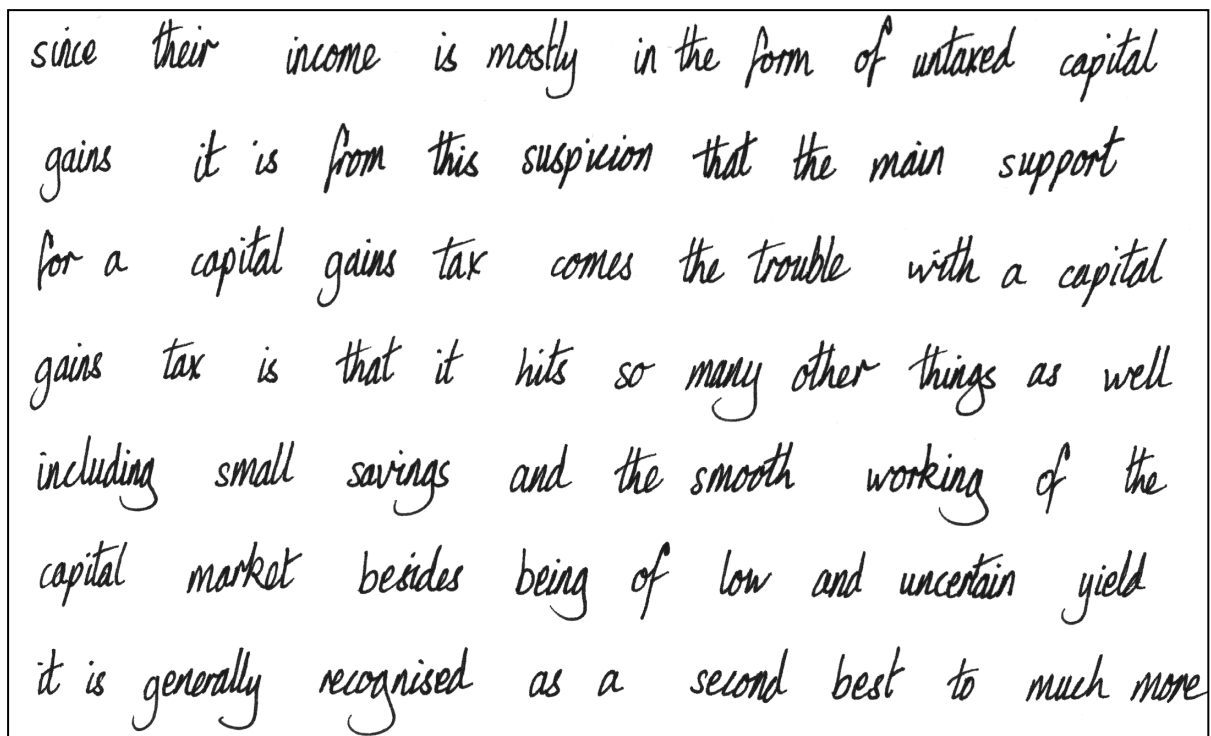
Figure 3.1 Block diagram of an offline handwriting recognition system

For a thorough overview of different handwriting recognition systems and architectures the reader is referred to [40],[41] and [74]. The first step is preprocessing of the input picture. The aim is to maximize the signal to noise ratio of the input and to suppress non-relevant data. This phase contains the thresholding and filtering of the input because the subsequent operations work on binary images. The localization of the lines and words on the image is also performed in this step. The line and word location information permits the subsequent analysis of the words in context, which may boost the recognition rate substantially.

The letter segmentation and feature extraction steps may be performed serially or in parallel. Some systems try to segment the located words into letters then build feature vectors from them that will be processed by the recognition engine; others begin to build the feature vectors from the words before segmenting them into letters [41].

The next step is the recognition of letters or words based on the feature vectors obtained in the previous steps. Many methods exist to accomplish this task, such as neural networks, hidden Markov models (HMMs) and dynamic programming techniques [43],[48],[41]. It is assumed that a suitable recognizer and feature extractor exists to process the segmented letter images. An example system is described in detail in [67], [70].

The output of the recognition engine is a list of [word, confidence] pairs that contain the recognized words and the confidence level of the recognition. The linguistic postprocessor takes this list as its input and based on grammatical and context information chooses the most appropriate word from the list. The postprocessor also corrects syntactic mistakes of the recognition.



since their income is mostly in the form of untaxed capital gains it is from this suspicion that the main support for a capital gains tax comes the trouble with a capital gains tax is that it hits so many other things as well including small savings and the smooth working of the capital market besides being of low and uncertain yield it is generally recognised as a second best to much more

Figure 3.2 An excerpt from the handwritten text database

3.2 Description of the Test Handwriting Database

The described algorithms were tested on a database of 25 handwritten pages collected by Andrew Senior [53],[54]. The database contains 7000 words from the LOB corpus (Lancaster-Oslo / Bergen) that were written by a single writer. An excerpt is shown in Figure 3.2. There is a segmentation file for every page that specifies the words and their location in the image in order to ease verification of the developed algorithms. We have also created a word-count file that specifies the number of lines and the number of words on a page for each page. Since the segmentation algorithms do not distinguish punctuation marks from real words, and the segmentation files do not contain punctuation information either, we have erased the punctuation marks from the page images. This database was chosen because it was the only database found that is freely available and was documented thoroughly.

3.3 The Preprocessing Tasks And Their Solutions

In almost every procedure of the preprocessing algorithms tens or even hundreds of image processing operations are performed on the input images. This is one of the most computationally intensive parts of offline handwriting recognition and motivated the use of the fastest possible parallel hardware architectures (i.e. CNNs) during the design of the algorithms.

These algorithms were designed to take advantage of dual digital and analog processing capabilities of the target ACE-BOX platform [19]. The experiments were performed with the MatCNN simulator in Matlab. We chose this approach because we could relatively quickly test the capabilities of the algorithms. We tried to devise algorithms that could be run entirely on the CNN chip to further explore the possibilities of analogic programming, careful to utilize only linear and nearest-neighbor templates that can be efficiently run on existing hardware implementations. The resolution requirements of the algorithms vary, but all single word algorithms require less than 128x128 pixels, not exceeding the resolution of the ACE16K chip.

In all flowcharts, the steps designed for CNNs are shown in *italic* on gray background, and the ones in normal type and white background use traditional digital methods. The utilized CNN templates may be found in the Appendix.

3.3.1 Locating The Lines

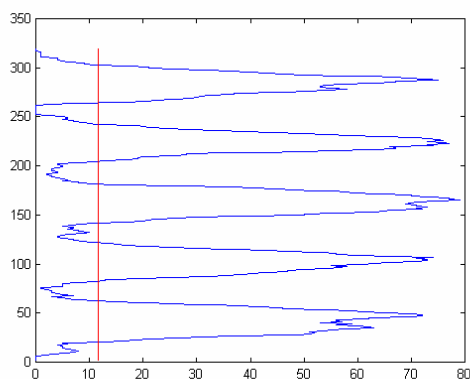
The algorithm found in the literature [42] attempts to find the lines based on local features, but it seemed to us that a line can more readily be thought of as a global structure. Our algorithm is similar to the ones used in OCR systems [44] where lines are localized by computing the horizontal histograms for the entire image at a couple of relevant skew angles then the angle

and position where the histograms have local minima are chosen as the location between lines. Calculating the horizontal histograms requires non-linear templates on CNNs, but can be substituted in this case with the horizontal projection operator.

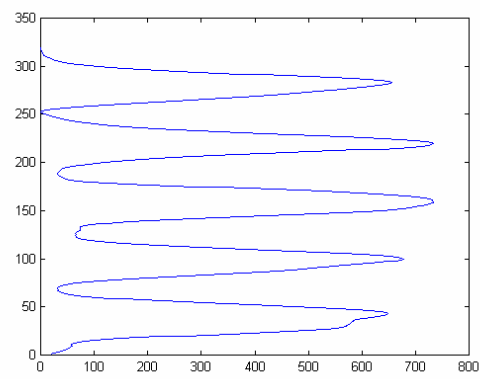
We refined the line-finding algorithm in a number of ways. In our early experiments, the histograms were quite noisy, so we began using a method to blur the words without affecting their location. Based on [68] we compute the pseudo convex hull of each word using the **HOLLOW** template. This template computes the convex hull of the objects in the image if it is run sufficiently long (i.e. till there are no changes on the image). If it is stopped “some time” earlier, then the hull will only be pseudo convex. The running time of the template (37 τ) was found by experimentation and appears to be consistent across the images in our handwriting database. The optimal value is slightly dependent on the overall style and size of the handwriting, but the histogram calculation is very robust against small errors in the convex hull calculation, so the overall algorithm is not affected if the writing is different.

The horizontal histogram computed on the pseudo convex hulls is smoothed further via sliding-window averaging with a window size (p_1) of 10 (it is effectively low pass filtered). The window size was found experimentally, but as a rule of thumb, it can be said that the smaller the average character size of the handwriting, the smaller the needed window. The next step of the algorithm is to find the local maxima of the histogram since these correspond to the location of the lines. Sometimes, more than one closely spaced local maxima correspond to the same line, usually because the skew of a line is substantial. To correct this we introduced a second parameter (p_2) that specifies a threshold within which we associate all maxima with one line. This parameter is set to 80% of the largest local maximum. Finally, we drop those maxima that are smaller than a given percentage of the average local maxima ($p_3=25\%$). The execution of the algorithm is illustrated in Figure 3.3. By adjusting the parameters p_1 , p_2 and p_3 the sensitivity of the algorithm can be varied widely.

The raw histogram



The histogram after smoothing



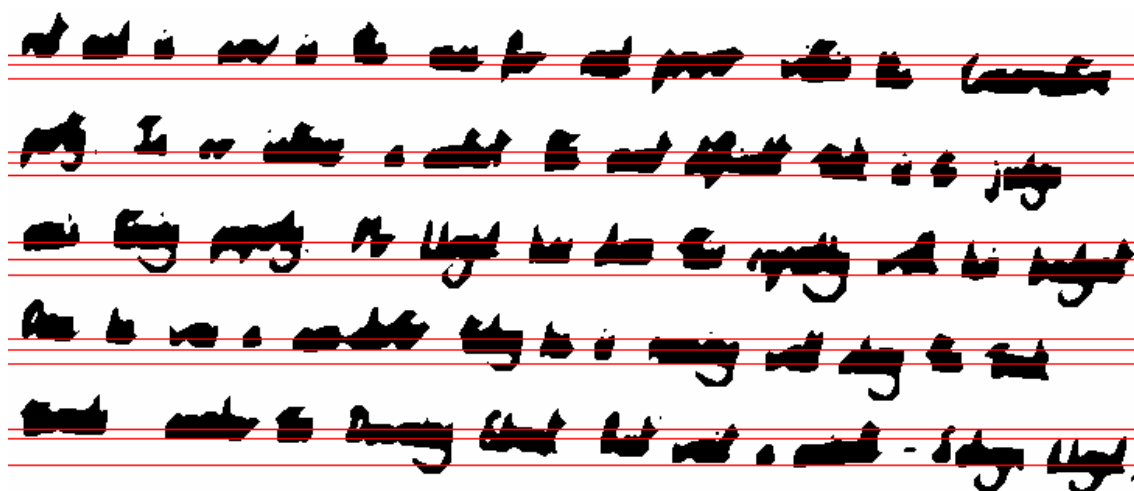


Figure 3.3 The results of the line localization algorithm

3.3.2 Correcting the Line Skew

After we have localized the lines in a given image, it is possible to correct their skew to somewhat normalize the word images. This is needed because the skewed line images introduce extra noise from the point of view of further processing and the word recognizer. It is somewhat like histogram equalization for image processing algorithms.

The skew correction algorithm first finds the lowest points of the pseudo convex hulls using the **LOCAL SOUTHERN ELEMENT (LSE)** detector template from the CNN Template Library [16]. These points follow the baseline of the lines closely, but contain some noise due to letters such as g, j, y, p, and q. To eliminate this noise, the **PRUNE** and **FIGREC** templates are applied in succession. The **PRUNE** template keeps those black pixels that have a black neighbor (in 4-connected sense) while the **FIGREC** template recalls only those parts of the original **LSE** filtered image that remained after the application of **PRUNE**. The last step of the algorithm uses linear regression with outlier rejection to fit a line to remaining black pixels, to calculate the angle of that line and to rotate the original image with that angle. The results of each step of the algorithm are shown in Figure 3.4.

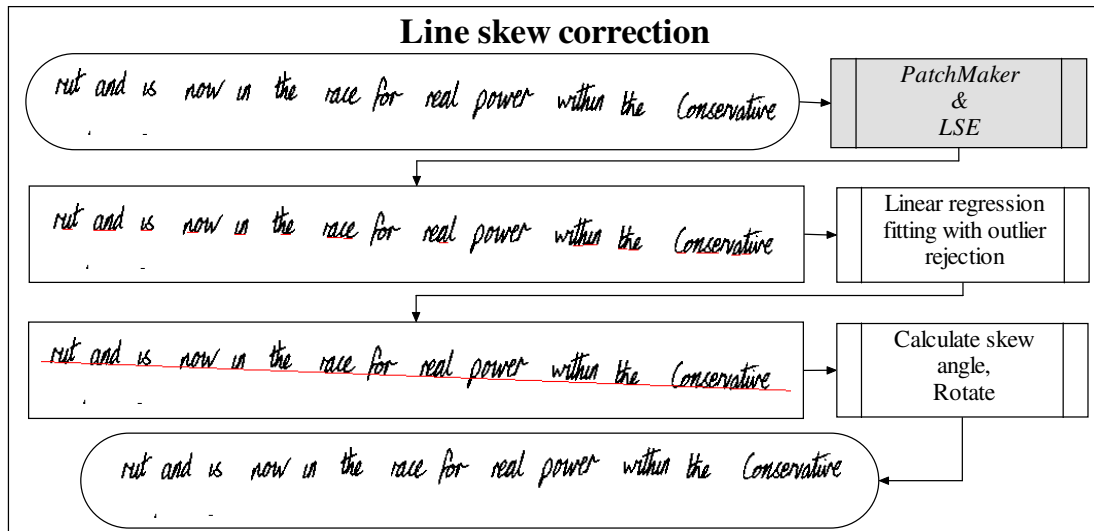


Figure 3.4 The line skew correction algorithm

3.3.3 Segmentation of Lines into Words

To aid further grammatical processing, context analysis and later reconstruction of each run of the algorithm, the lines are stored separately and in sequence as they appear on a page. Relevant information is also stored with each line, such as its original image, the de-skewed image etc. The data about the constituent words is also stored along with each line.

After the lines have been processed, the next step is to locate the words on each line. This could be easily achieved with the calculation of vertical histograms and identifying the local minima in those histograms, but we looked for a fast analogic approach that provides almost the same results in this particular application. The conceived algorithm is as follows: first, we compute the pseudo convex hull for the line, and then apply the **VCCD** template. This template detects the vertically connected components in an image by shifting them downwards until they disappear. The result of the template for these types of images is (since there is usually only vertically connected component, i.e. a word) that horizontal lines appear in the last row of the image corresponding to the largest horizontal extent of the word images. This makes it possible to extract the word images from the line very easily. The steps of the algorithm are illustrated in Figure 3.5. One may ask, what happens, if the pseudo convex hulls of the words overlap? This is not nearly as big a problem as it may first seem (in fact there are no instances of such a configuration in our database), but these sites can be identified because there will be two parallel horizontal lines where the overlap occurs.

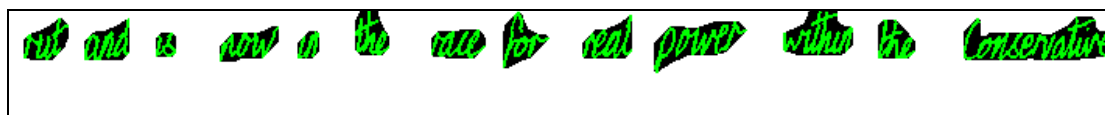


Figure 3.5 Locating the words within a line. The bottom red lines mark the location of words

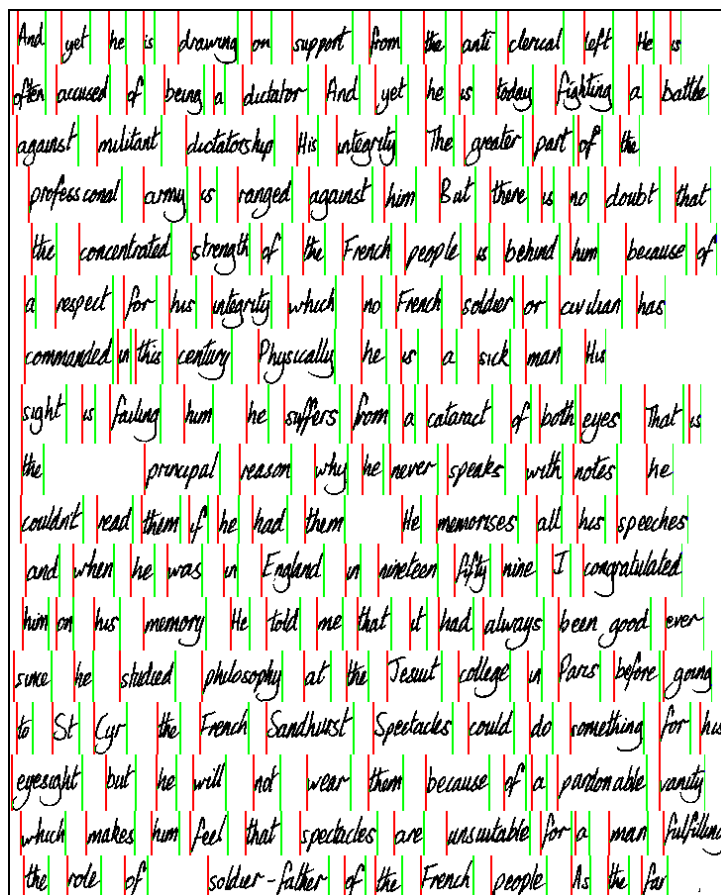


Figure 3.6 Word and line location results for a sample page. Red lines mark the start of words and green lines mark the end

3.4 Segmentation Of Words Into Letters

After the words have been located within the lines, all successive operations proceed at the word level since larger context information is only used at the linguistic post processing stage of recognition. In segmentation-based systems, where the basic units of recognition are letters, it is crucial to segment the words into letters as accurately as possible. Unfortunately, this is almost impossible to do correctly as illustrated by the Sayre paradox [56]: “To recognize a letter, one must know where it starts and where it ends, to isolate a letter, one must recognize it first”. This problem can be circumvented by over-segmentation. Over-segmentation means that we first try to find the smallest possible meaningful segments (letters, or parts of letters), which are called

primitive segments, and then later we try to assemble these primitive segments into letters based on input from the character recognizer. Our algorithm also uses this approach, but tries its best to segment the words into letters as accurately as possible in order to minimize the computational cost associated with the assembly and re-recognition of compound segments. Under segmentation must also be avoided because later these errors cannot be corrected easily and may degrade the performance of the recognizer severely. It is also a great problem that many times the word image in itself does not contain enough information to correctly segment it into letters as illustrated in Figure 3.7.



Figure 3.7 An example for a word that cannot be correctly segmented based on the word image alone (the word is “Conservative”)

3.4.1 Localization of the Upper and Lower Baselines, Skew Correction

Vertical location of a character within a word is important information for its recognition, even human readers rely on it to discern letters [37]. This location is best described relative to the upper and lower baselines of the word. The parts of letters that descend under the lower baseline are called descenders while parts that ascend over the upper baseline are called ascenders. Ascenders and descenders are distinguishing features of many letters such as p, q, b, l, j, k etc. Encoding the location and number of descenders in the feature vectors passed to the character recognizer could aid in the accurate recognition of the letters. For this to be possible however, the baselines of the words must be calculated. The algorithm is somewhat similar to the one used to calculate the line skew, but there are important differences.

The pseudo convex hulls are created as before, but a mask needs to be generated to exclude parts of the word where only erroneous elements would be located (the upper half, when computing the lower baseline, and the lower half for the upper baseline). This is not needed when calculating the baseline for the whole line because there are enough local southern elements on a line (since it is much longer), that the regression fitting will still be accurate. We generated this mask by letting the **HOLLOW** template run longer (for 58 τ), running the **FINDCENTER** CNN algorithm [16] on the result, and then generating a shadow from the center

point using the **SHADOWUP** or **SHADOWDOWN** templates. The result is a mask that can be used to specify where the following templates should be applied.

Once this is done, the local northern elements (LNE-s) have to be calculated in addition to the southern elements because the upper baselines are also needed. The LNE-s are calculated with the LNE template. Linear regression line fitting with outlier rejection is used to estimate the baselines. Outliers are points that are farther from the word center than a given threshold; this threshold can be estimated from the writing, it should be about half of the average word height. After the baselines are found, the image is rotated so the lower baseline is horizontal. The whole process is illustrated in Figure 3.8.

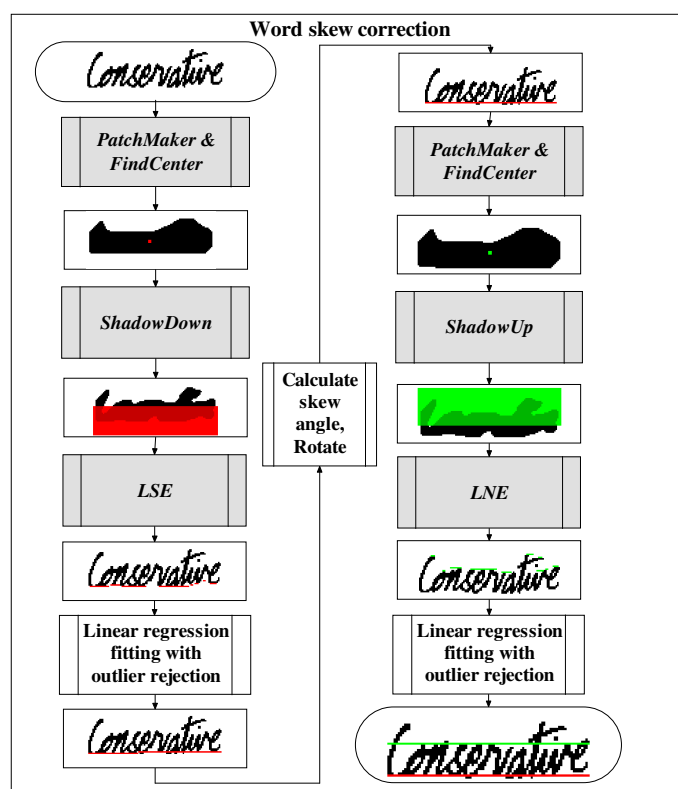


Figure 3.8 The word baseline locator algorithm

3.4.2 Images Aiding in Segmentation

The letter segmentation algorithm developed by us is based on the same ideas as the segmentation algorithm described for touching numerals in [55]. The basic idea is that we can gain meaningful information about the structure of a word by skeletonizing it and its background, and locating the junction- and endpoints of the skeleton. By using these special points, we are (mostly) able to construct the segmentation boundaries.

3.4.3 Skeletons of the Background

While pondering the problem of letter segmentation, we noticed an interesting property of the skeletonized word image backgrounds and their correct segmentation boundaries: each correct boundary line should start above the foreground skeleton (i.e. the word) and end under it, and it may only cross the foreground skeleton once. This is a necessary condition. If we skeletonize the whole background at the same time, identifying which endpoints are above and below the word skeleton becomes an extra and not so trivial task. However if we take a different approach, this step can be skipped.

Our main idea was that if the foreground skeleton image is 4-connected, then the skeleton could be used as a barrier between the upper and lower parts of the background skeleton. This can be exploited by flood filling the background from the upper and lower baselines at the same time. This ensures, that if there are breaks in the word then the flood will not “overflow”. These floods can then be separately skeletonized, and the characteristic points of the skeleton identified. Flood filling can be very efficiently accomplished with trigger waves on CNNs. A thorough description of trigger waves with CNNs, their properties and some applications can be found in [58]. The template used for filling the background is the **BPROP** template and the filling process is shown in Figure 3.9.



Figure 3.9 The background filling process using the **BPROP** template

The skeletonization algorithm described in [16] is suitable for general purposes but in this particular application, the structure of the resulting skeleton is crucial, so we experimented with different ordering of the skeletonization templates. This has a huge impact on the final skeleton, as shown in Figure 3.10, which shows a selection of the possible different skeleton template orderings. The most important criteria for the skeletonized background were:

- the vertical skeleton branches should split as little as possible, since this makes the endpoint connection step (see next section) much easier
- the endpoints of the vertical skeleton branches should be as close to each other as possible (for the same reason as in the previous point)

After running tests with different orderings of the skeleton templates, we analysed the results and selected the ordering f) for later use. This method also generated skeletons that contain relatively long straight segments, which is advantageous.



Figure 3.10 Effect of different orderings of the skeletonization on the skeleton. The number sequences after the template base name represent the order of the individual templates used in the iterative skeletonization algorithm (these templates may be found in the Appendix)

3.4.3.1 Locating the Endpoints of the Background Skeleton

The endpoints of the background skeleton must be found since they form the basis of the letter segmentation algorithm. By definition, a point on an 8-connected skeleton is an endpoint if and only if it has exactly one neighbor. This definition closely follows our intuitive notion of an endpoint except for two special cases. These two exceptions occur when there are branches on the skeleton that are exactly 1 pixel long (this is shown in Figure 3.11). Even though the interpretation is ambiguous (they could be considered only slight curves), we will mark them as endpoints since we do not want to lose an endpoint of the skeleton under any circumstances.

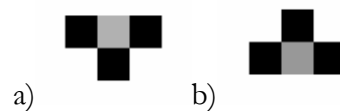


Figure 3.11 Special endpoint morphologies relevant for letter segmentation. The grey pixels may be black or white. Only those morphologies are important for letter segmentation, where the 1 pixel branch could be a vertical endpoint (these are shown in the figure)

To find the endpoints we first erase those points that have only 1 neighbor with the **GETEP** template, then subtract the result from the original image. Afterwards, we find the special cases with the **TOEP** and **BOTEP** templates and merge these points with the previous ones. We use the pseudo convex hulls as a mask to constrain the search area near the writing, because skeletons sometimes have stray endpoints.

3.4.4 Skeletons of the Foreground

The foreground skeleton must be 4-connected for the above algorithm to work and skeletonizing with the **SKLHV** templates (applying them in ascending order) can ensure this. However, if the original word image was itself not 4-connected, then the resulting skeleton will not be 4-connected either; therefore the original image has to be preprocessed to ensure 4-connectedness. This can be done using the **CONN4SE**, **CONN4SW**, **CONN4NE** and **CONN4NW** templates that fill a given pixel according to local rules.

3.5 Segmenting Words into Letters

After studying the skeletonization images of handwritten words, we have found only two requirements that have to be filled by a correct segmentation boundary:

- no endpoint can be part of two boundaries at the same time
- a segmentation boundary must start at a top endpoint and end at a bottom endpoint

Unfortunately, these are not enough to filter and match the already identified skeleton endpoints so some other heuristics are needed, which are illustrated in Figure 3.12.

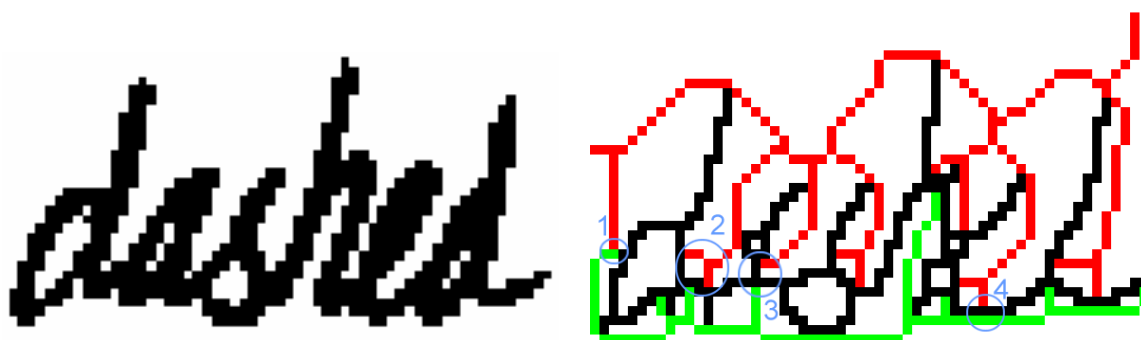


Figure 3.12 Segmentation heuristics using skeleton endpoints. Sample locations where different skeleton endpoint relationships require connection heuristics: locations 1, 3 and 4 require distance measurement while location 4 requires further post-processing

At location 1, the top and a bottom endpoints are located right next to each other so it is certainly a segment boundary, since this can only happen where the top and bottom floods have met because of a word break (here it signifies the start of the first letter). Location 2 shows an instance where the skeleton needs post processing to unite the short branches that add only clutter to the problem. We present a novel approach to solve this in the next section. Location 3 shows the average case where the top and bottom endpoints are located close to (within a specified distance d), but not right next to each other. There is a boundary point where the shortest path connecting these points intersects the word skeleton. Finally location 4 shows a special case that is quite common, when there is only a top endpoint with a horizontal (or near horizontal) line segment on the lower background skeleton. This is usually a segmentation point, but has to be found separately from the other cases.

3.5.1 Post processing the Endpoints of Skeletons – Parallel Distance

Approximation

The objective of the algorithm is to replace the endpoints of two short branches of a skeleton with one endpoint located approximately halfway between the original points and on the same side of the word skeleton.

This problem can be efficiently solved using trigger waves with isotropic templates. Trigger waves generated by an isotropic template (**CPATCH**) propagate with the same speed in all directions, so they can be used to measure distance on an image.

In order to measure the distance between two points, we must be able to tell whether the

two wave fronts met while the waves propagated for a specified time (say t). If they have met, then the two points are closer than two times the distance the individual wave fronts covered. We can detect whether the wave fronts merged by running the inverse template (**CWPATCH**) for the same amount of time (t). This will have the effect that in all places where the patches stayed convex (which is the same as saying where the circles did not merge) only the original starting pixels will remain, but where the wave fronts did merge, multi-pixel patches will be visible. This is illustrated in Figure 3.13. We can give an upper estimate of the distance based on the running time of the templates.

This algorithm has a very nice property from a performance point of view: the execution time is only dependent on the distance one wants to measure and not on the number of points in the image. The absolute or relative position of the points also does not affect the execution time. This is in stark contrast to algorithms possible on a serial digital processor, where the point relationships must be evaluated one pair at a time, resulting in an algorithm whose runtime is $O(n^2)$, where n is the number of points in the image. Notice that the digital algorithm is not dependent on the distance of the points. In a situation where the question is what the distances between points (or certain points) are, the digital algorithm might be more appropriate, since the distances are explicitly calculated. In the current application, however, only points within an a priori given distance must be detected, and here the wave-based approach is faster.

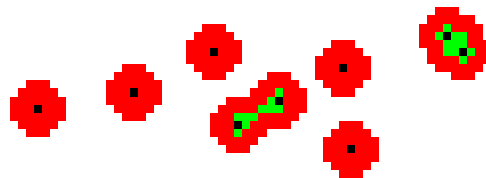


Figure 3.13 Distance approximation with trigger waves. The black points are the starting points; the red circles show the maximal extent of the wave fronts (after **CPATCH**) and the green patches the result after applying the **CWPATCH** template

The “skeleton short branch merger” algorithm proceeds as follows (its flow chart is shown on Figure 3.14):

1. Initialize a white image with the branch endpoints as the only black pixels.
2. Run the template **CPATCH** for $\alpha\tau$. This generates black circles with a radius r around the initial points. Where the starting points were close to each other, the circles will merge.
3. Run the inverse template **CWPATCH** for $\alpha\tau$. This shrinks the circles back to a point everywhere where the circles did not merge (they stayed convex), and to more than one

pixel where they did.

4. Remove single pixels from the image with the **FIGEXT** template.
5. Find the center pixels of the remaining patches with the **FINDCENTER** CNN algorithm.

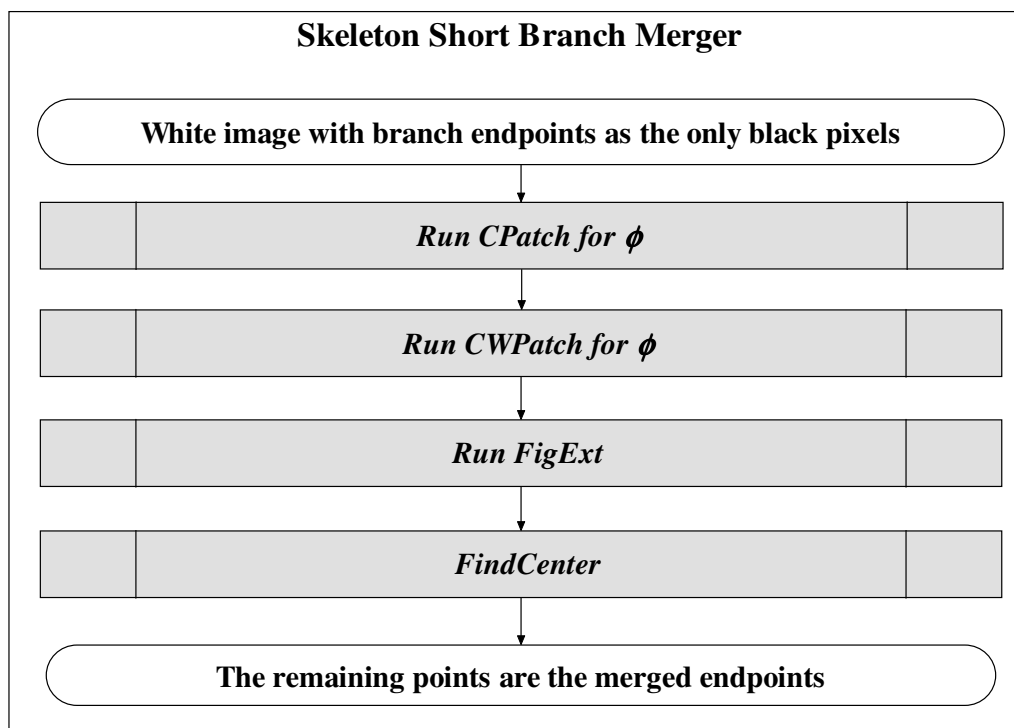


Figure 3.14 Flowchart of the skeleton short branch merger algorithm

3.5.2 The Letter Segmentation Algorithm

1. Find the skeleton endpoints
2. Merge the endpoints on short branches, separately for the top half of the background skeleton and for the bottom half (post-processing for situations similar to location 2 in Figure 3.12)
3. Find the word breaks, i.e. those locations, where the top endpoints and bottom endpoints are right next to each other (location 1 in Figure 3.12). Store these as segmentation boundaries and remove them from further processing.
4. Find those skeleton endpoint pairs where one is on the top skeleton, the other on the bottom skeleton and they are closer to each other than some predefined distance d (location 3). This can be accomplished by the algorithm described in 3.5.1, but run **FINDCENTER** on the intersection of the foreground skeleton and the patches remaining after **CWPATCH**. Add the center points to the segmentation boundaries and remove them from further processing.

5. From the remaining top endpoints, generate vertical lines that are at most n pixels long using the **DPROP** template. If these reach the bottom skeleton and the lines intersect the foreground skeleton, then add the lowest intersection points to the segment boundaries.

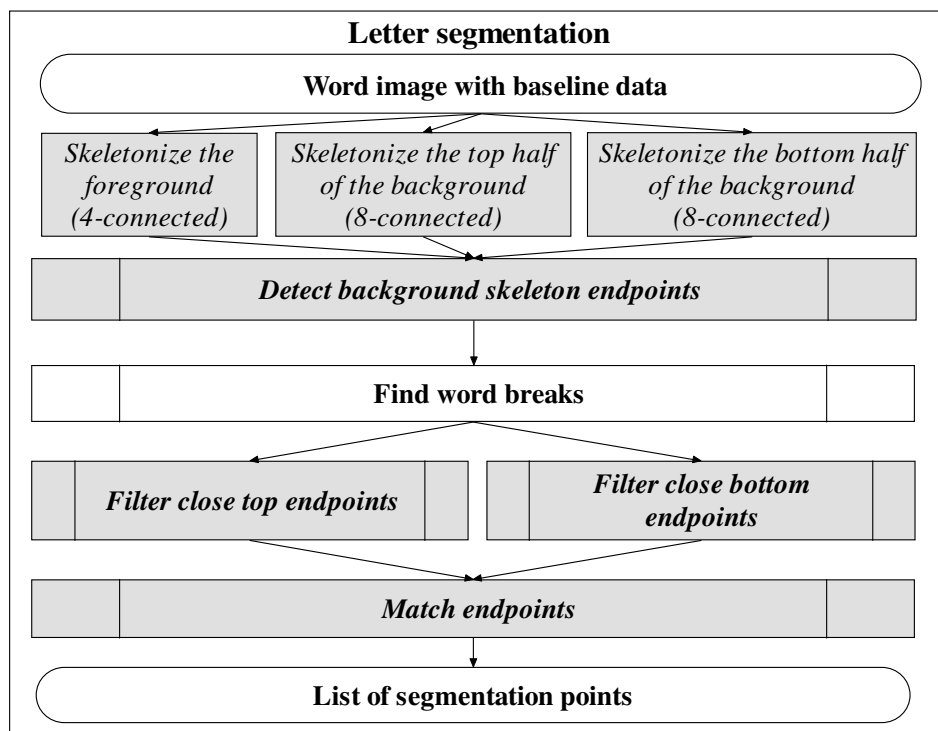


Figure 3.15 Flowchart of the letter segmentation algorithm

3.6 Discussion of the Results

We have run these line, word and letter segmentation algorithms on 10 pages of the database. The line segmentation algorithm found every line on every page correctly. Statistics for the word segmentation algorithm are shown in Figure 3.16.

Total number of lines processed:	169
Correctly segmented lines:	146 (86.39 %)
Incorrectly segmented lines:	23 (13.61 %)
Lines w. more words than expected:	12 (7.1 %)
Lines w. fewer words than expected:	11 (6.51 %)

Figure 3.16 Word segmentation results

These results are quite good, since 87% of the lines were segmented correctly into words,

and out of the erroneous 23 there were only 11 lines with less words than expected. This is important, because merging words afterwards is easier than splitting up words. This means, that 93.49% of the lines were either correctly segmented, or can be easily corrected.

Assessing the results of the letter segmentation algorithm is not as straight forward as that of the other two. The problem is that there are many equivalent segmentation boundaries, which cannot be identified easily by a computer program; one has to inspect the word images manually, one by one, which requires an enormous amount of time and is error prone. Using a statistical approach, we can evaluate the algorithm differently. Since the goal was over-segmentation, we can assess the algorithm by comparing the number of segmentation boundaries with the number of letters specified in the segmentation file, which will give us a rough estimate of the effectiveness of the algorithm. These statistics are shown in Figure 3.17.

Total number of words found:	1201
Total correctly (over)segmented words:	890 (74.10 %)
Total incorrectly (under)segmented words:	311 (25.90 %)

Figure 3.17 Letter segmentation results for 7 pages

We also inspected the segmentation of random words visually to further grade the algorithm. Results for a few words can be seen in Figure 3.18. Note, that there are a couple of problem sites (encircled in blue), some of which are difficult to resolve even for a human reader. It is also evident why automatic evaluation of the segment boundaries is very hard.

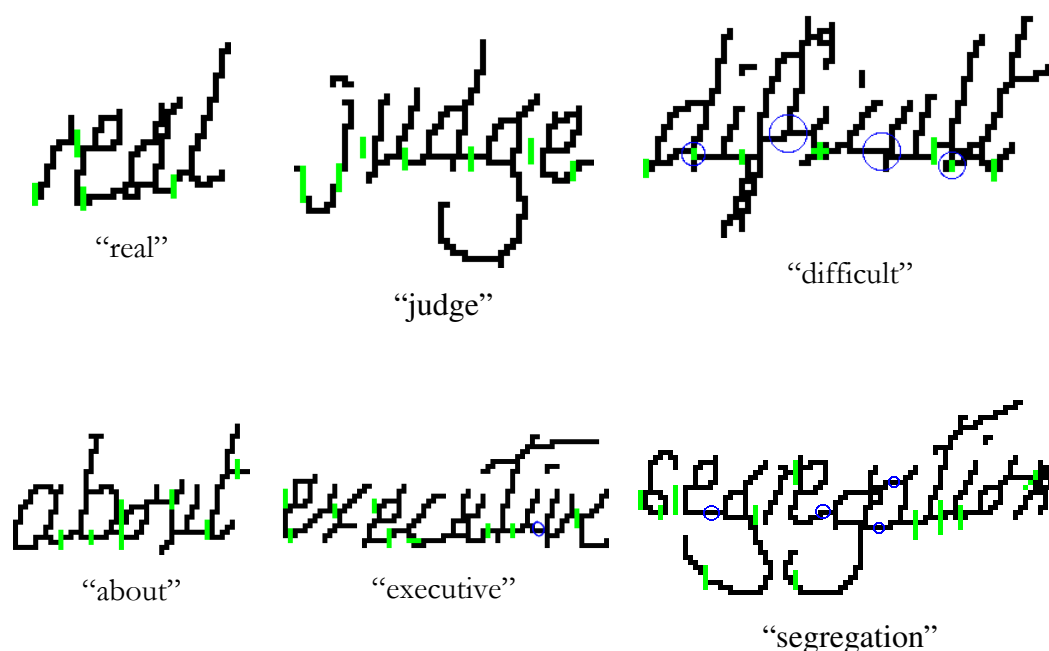


Figure 3.18 Letter segmentation results for a few sample words. The segment boundaries have been manually enlarged to be clearly visible. The circles indicate problem spots

3.7 Conclusions

We have demonstrated that analogic algorithms can be utilized effectively in the preprocessing and segmentation problems of off-line handwriting recognition. By avoiding iterative methods and using propagating wave-based approaches where possible, the inherent parallel processing capabilities of CNN arrays can be greatly exploited.

It is also evident from the results of the letter segmentation algorithms, that a linear “feed-forward” approach (segmentation \rightarrow recognition \rightarrow preprocessing) may not be the best and most robust way to tackle the problem, if additional linguistic information is available. Without linguistic resources it is very hard to handle ambiguities in the handwriting styles, but it is true the better the letter level segmentation of a word, the easier it is to recognize it even without linguistic input.

Studies conducted on human subjects indicate that readers do not read a word linearly from left to right, letter by letter, rather they try to identify the most distinctive letters first, such as the ones that are at the beginning and the end of the word [37], and those with some prominent features. If this limited information is sufficient to recognize the word with the help of context information, they move on, otherwise they attempt to recognize more letters from the word. Based on this knowledge and the results of the research described in this thesis, my colleague, Kristóf Karacs started working on an offline handwriting recognition system that would utilize an experimental framework, which would enable us to imitate this recognition process. It is evident that the use of a lexicon based linguistic system - which is able to generate all grammatically correct forms of the words of a language – from the earliest stages of processing can provide additional information to support the segmentation and recognition phase. This information could enable the dynamic reduction of lexicon size, selecting only candidate words that fit a given feature set. Since segmentation is so tightly coupled to recognition, an approach that utilizes recognition information during segmentation is expected to work better than a feed-forward mechanism [75]. The structure of the described handwriting recognition system is shown on Figure 3.19.

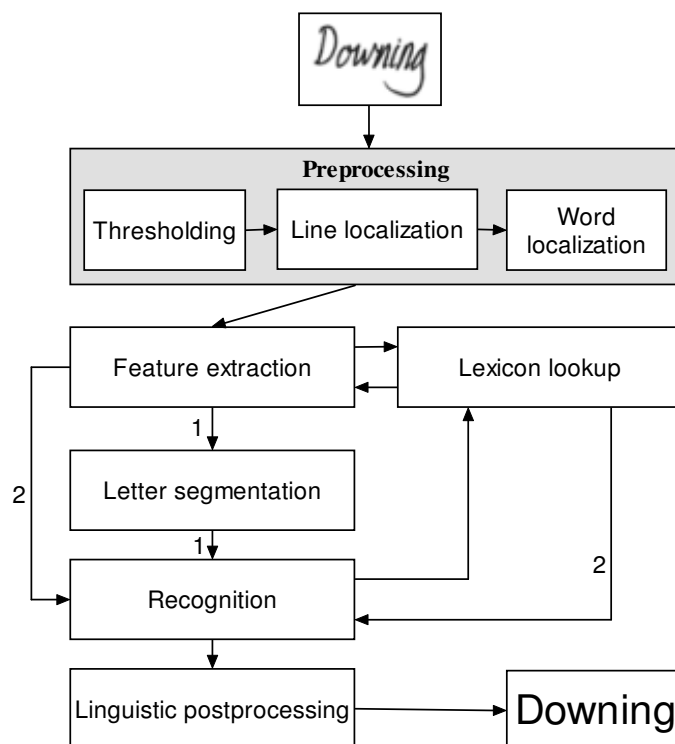


Figure 3.19 Block diagram of the proposed handwriting recognition system. The numbered lines represent alternative paths to recognition. Path 1 shows the case when the lexicon cannot help in reducing the number of candidate words and the word must be further segmented and recognized. Path 2 illustrates the case when the lexicon lookup narrows down the number of candidate words so effectively that recognition is “trivial”.

Human subjects rely heavily on so-called perceptual features such as ascenders, descenders, t-crossings, loops and word length to identify words and letters when reading. The use of these features as the basis of recognition is also worth considering, since the detection of these features lends itself well to CNN algorithms.

Finally, it is worth noting that the same principle – using linguistics from the early stages of recognition in a constructive way – would probably be equally promising in speech recognition, where the problem is essentially the same, but instead of letters, the words and phonemes must be correctly segmented and recognized.

4 Summary

In the dissertation, I have described two applications of the CNN-UM based wave computing combined with conventional digital algorithms. These applications tackle real world problems for which no adequate solutions have been presented in the literature.

I introduced a multitarget tracking system, which is capable of tracking targets moving at high speed in a plane, and generating control signals to drive actuation. The CNN-based image processing front-end of the MTT system is based on principles gleaned from the modeling of mammalian retinal processing. The input image is processed in parallel channels that enhance different aspects of the image, and these parallel channels are later combined into a single – and very sparse – input representation, which is the basis of all further processing. The targets are characterized by numeric features extracted (measured) from the input, and these features may be used to filter out unwanted ones. Several different (and very efficient) measurement-to-track data association routines were implemented along with fixed gain state estimation filters. The output of the tracking algorithm was used to tune the internal parameters of the image processing front-end to adapt to changes in the environment and hardware instability. I also presented a demo application of the algorithm where the tracking results are used to drive a laser scanner device to tag certain tracked targets.

In the 3rd chapter, I presented algorithms to handle the tasks of preprocessing handwriting for recognition. These algorithms segment a handwritten page into lines, the lines into words, and the words into letters solely based on the image of the handwriting. During the course of this research, I devised an algorithm that is able to detect points in an image that are closer than a given distance. The advantage of this algorithm compared to previous approaches is that it is parallel so the execution time is independent of the number of points and their location in the

image. Based on the results of the segmentation algorithms and discussions with my colleagues we decided that much better segmentation results – and hence recognition accuracy – can only be obtained if available linguistic knowledge is exploited in a constructive way during the segmentation process, much like the way humans approach the same problem.

4.1 Methods of Investigation

During my research, I relied on the tools of many disciplines. In the design of the tracking algorithms I applied algorithms used in radar tracking and the results of studies that describe their accuracy and efficiency. I analyzed the efficiency of algorithms with methods from algorithm theory to be able to compare the proposed algorithms with those previously published in the literature. For the CNN-UM algorithms, I utilized the template classes and the accrued experience with them previously published in the literature. It was an important consideration to choose templates that could be executed reliably on the CNN-UM chips available in our laboratory ensuring the immediate practical use of the algorithms. In image processing algorithms, I relied on the results of binary mathematical morphology and their CNN-UM implementations.

In general, an important aspect of my algorithms is that for maximum speed and efficiency, they utilize CNN and classical digital solutions executed on their respective platforms. I tested the algorithm on PCs with Intel x86 architecture processors using the Matlab software suite augmented with the MatCNN simulator and executed them on the ACE-BOX and Bi-i systems. Both systems contain mixed-mode (analog-digital) CNN-UM chips; the former contains the Ace4k with 64x64 resolution, the latter the Ace16k with 128x128. I also actively participated in the design and development of the development environments of these systems.

4.2 New Scientific Results

1. Thesis: Adaptive, multitarget tracking algorithm and system

An important subtask in video flow processing is the tracking of targets moving at arbitrary speeds with high precision and reliability. The challenges in these applications are the filtering of the objects, the modeling of their motion, and – especially – the tuning of the algorithm parameters during execution because of change environmental conditions. I created an algorithm to solve these problems, which utilizes CNN-UM processors to filter efficiently out the objects in the images and allows the easy adjustment of its parameters in order to generate consistent output. I combined this algorithm with one of the best so-called

data association algorithms described in the literature and created a complete system that is able to track multiple objects in real-time. Publications: [1],[10],[11],[15]

1.1 I developed an algorithm, which is able to efficiently track multiple objects in a video flow and extract and classify their kinematic properties

The algorithm – relying on ideas from the mammalian retina – extracts the important image features relevant to the task in several parallel channels, and combines them through a special method developed by me. The results are then further filtered, and using optimal data association methods the kinematic properties extracted, once the objects have been located on the filtered image. The method also enables the user to filter the tracked objects based on morphologic or kinematic properties.

1.2 I demonstrated that using the above algorithm, object saliency is better on the filtered image in an average sense than on the individual channels.

I combine the output of the individual filter channels using a custom method (which can be tuned thru several parameters). I showed that the data association algorithms provide better results in an average sense (when no *a priori* assumptions can be used) if executed on the combined filtered image than if run on the individual channels.

1.3 I demonstrated that by feeding back the results of the tracking to the multichannel front-end, the accuracy of the tracking could be enhanced.

I used statistical and qualitative analysis on the tracking results to compute measures to judge the accuracy of the tracking. I developed an algorithm to adjust the parameters of the multichannel front-end based on these measures to increase the tracking accuracy.

2. Analogic segmentation algorithms for offline handwriting recognition

Segmentation problems are among the most difficult in offline handwriting recognition: segmenting pages, lines and words before the commencement of the actual recognition. The more accurate the segmentation, the easier and more accurate the recognition will be. I developed analogic algorithms that are able to efficiently locate and segment an image of a handwritten page into lines, the lines into words and the words into letters. The algorithms exploit the wave computing capabilities of the CNN-UM architecture. Publications: [3], [14]

2.1 I developed methods to segment handwritten images into lines, and lines into words.

I created an efficient algorithm to segment handwritten pages into lines, even if the lines are somewhat skewed or non-straight. I also showed a method to reliably segment lines into words for further processing.

2.2 I created a new algorithm to segment handwritten words into letters and showed a new wave computing-based solution to find pairs of points in parallel, which are closer to each other than a given distance.

I developed a word segmentation algorithm, which does not rely on semantic information, thus it can be used for unfamiliar languages and texts. I utilized a wave computing based method to detect points, which are within a given distance from each other. An important advantage of this algorithm compared to conventional methods is that the execution time is independent of the number of points and their location.

4.3 Application Areas of the Results

All of the algorithms described in the dissertation present solutions to real-world problems. I showed that execution speed and accuracy of the multitarget tracking algorithm (1st thesis) enables its use in control applications. To demonstrate this, with the help of my colleagues, I built a laser targeting-tracking system, which is able to track and target with a laser multiple objects moving at high speed in real time.

The multitarget tracking algorithm is also used in a software system whose task is the surveillance and monitoring of indoor and outdoor industrial areas. There is great demand today for complex surveillance systems, which take over the boring and error-prone tasks from human personnel, but are able to trigger alarms, when needed. The use of the algorithm in this setting has many advantages:

- It enables the triggering of alarms based on complex motion patterns (motion trajectory, direction and speed, the number of moving objects, etc.)
- Kinematic properties may be used during object identification and classification, which – in many cases – simplifies the task.
- The object tracking system supplies object location and speed prediction information, which may be used to optimize processing at the later stages of the surveillance algorithm

I also designed the algorithms for the preprocessing tasks of offline handwriting recognition

(2nd thesis) with ease of use in mind. This means, that each of the templates is executable on one of the commercially available VLSI CNN-UM chips, and the (possibly) low resolution of the processors is not a barrier to application (128x128 in the case of Ace16k). I collaborated closely with my colleagues who are working on the recognition of handwritten characters and words so that it would be possible to interface the systems easily.

5 Bibliography

5.1 Publications Related to CNNs and CNN Technology

- [1] L. O. Chua and L. Yang, “Cellular Neural Networks: Theory and Applications”, *IEEE Trans. on Circ. & Syst.*, Vol. 35, pp. 1257-1290, 1988.
- [2] T. Roska and L. O. Chua, “The CNN Universal Machine”, *IEEE Trans. on Circuits and Systems*, Vol. 40, pp. 163-173, 1993.
- [3] L. O. Chua, and T. Roska, “The CNN Paradigm”, *IEEE Trans. on Circuits and Systems.*, Vol. 40, pp.147-156, 1993.
- [4] L. O. Chua and T. Roska, “Cellular Neural Networks and Visual Computing” *Cambridge University Press*, Cambridge, UK 2002.
- [5] L. O. Chua “CNN: A Paradigm for Complexity”, *World Scientific Pub. Co.*, 1998.
- [6] L. O. Chua, “CNN: a Vision of Complexity ”, *Int. J. of Bifurcation and Chaos*, Vol. 7, No. 10, pp. 2219-2425, 1997.
- [7] T. Roska „Computational and Computer Complexity of Analogic Cellular Wave Computers”, *Proc. IEEE Intl. Workshop on Cellular Neural Networks and their Applications*, 2002. pp. 323-335.
- [8] G. Liñán, S. Espejo, R. Domínguez-Castro, A. Rodríguez-Vázquez, “Ace4k: An analog I/O 64×64 visual microprocessor chip with 7-bit analog accuracy”, *International Journal of Circuit Theory and Applications*, Vol. 30, No. 2-3, pp.: 89-116, 2002
- [9] A. Rodríguez-Vázquez, G. Liñán, L. Carranza, E. Roca, R. Carmona, F. Jiménez, R. Domínguez-Castro, and S. Espejo, “ACE16k: The Third Generation of Mixed-Signal SIMD-CNN ACE Chips Toward VSoCs”, *IEEE Transactions On Circuits and Systems*, Vol. 51, No. 5, pp. 851-863, 2004.

- [10] S. Espejo, R. Carmona, R. Domínguez-Castro and A. Rodríguez-Vázquez “A VLSI Oriented Continuous-Time CNN Model”, *International Journal of Circuit Theory and Applications*, Vol. 24, No. 3, pp. 341-356, 1996.
- [11] Cs. Rekeczky, T. Roska, and A. Ushida, "CNN-based Difference-controlled Adaptive Nonlinear Image Filters", *International Journal of Circuit Theory and Applications*, Vol. 26, pp. 375-423, July-August 1998.
- [12] Á. Zarándy: “The Art of CNN Template Design”, *International Journal of Circuit Theory and Applications*, Vol. 27, No. 1, pp. 5-23, 1999
- [13] T. Kozek, T. Roska and L. O. Chua: “Genetic Algorithm for CNN Template Learning”, *IEEE Transactions on Circuits and Systems–I: Fundamental Theory and Applications*, Vol. 40, pp. 392-402, June 1993.
- [14] L. Nemes, L. O. Chua, and T. Roska: “Implementation of Arbitrary Boolean Functions on the CNN Universal Machine”, *International Journal of Circuit Theory and Applications*, Vol. 26, No. 6, pp. 593-610, 1998.
- [15] H. Harrer and J. A. Nossek, "Discrete-time Cellular Neural Networks", *International Journal of Circuit Theory and Applications*, Vol. 20, pp. 453-468, 1992.
- [16] T. Roska and L. Kék, “CNN Software Library (Templates and Algorithms), Version 7.2”, *Analogical and Neural Computing Laboratory, Computer and Automation Research Institute, Hungarian Academy of Sciences (MTA SZTAKI)*, DNS-CADET-15, Budapest, 1998.
- [17] MatCNN is available from: <http://lab.analogic.sztaki.hu/Candy/matcnn.html>
- [18] UMF Diagrams of CNN Algorithms and Specifications:
http://cnn-technology.itk.ppke.hu/UMF_Library.pdf
- [19] Analogic Computers Ltd. <http://www.analogic-computers.com>

5.2 Publications Related to Multitarget Tracking

- [20] V.S.S Hwang, “Tracking feature points in time-varying images using an opportunistic selection approach”, *Pattern Recognition*, Vol. 22, No. 3, pp. 247-256, 1989.
- [21] K. Sethi and R. Jain, “Finding trajectories of feature points in a monocular image sequence”, *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 9., No. 1, pp.56-73, 1987.
- [22] B. K. P. Horn and B.G. Schunck, “*Determining optical flow*”, *Artificial Intelligence*, Vol. 17, pp. 185-203, 1981.
- [23] H.H. Nagel, “Displacement vectors derived from second order intensity variations in image sequences”, *Computer Vision Graphics and Image Processing*, Vol. 21., No. 1, pp. 85-117, 1983.

- [24] S. Blackman and R. Popoli, “Design and Analysis of Modern Tracking Systems”, Artech House, 1999.
- [25] Y. Bar-Shalom W.D. Blair ed., “Multitarget-Multisensor Tracking: Applications and Advances, Vol. III”, Artech House 2000.
- [26] B. Roska and F.S Werblin, “Vertical Interactions across Ten Parallel Stacked Representations in Mammalian Retina”, *Nature* 410 (2001) pp 583-587.
- [27] D. Marr, “Vision”, Freeman Publishers, 1982.
- [28] R. Jonker R. and A. Volgenant, “A Shortest Augmenting Path Algorithm for Dense and Sparse Linear Assignment Problems”, *J. Computing*, Vol. 38, pp. 325-430, 1987.
- [29] D.P. Bertsekas, “The Auction Algorithm: a Distributed Relaxation Method for the Assignment Problem”, *Annals of Operation Research: special issue on parallel optimization*, Vol. 14, pp. 105-123, 1988.
- [30] M. Balinski, “Signature Methods for the Assignment Problem”, *Operations Research*, Vol. 33, No. 3, pp. 527-536, May 1985.
- [31] Y. Bar-Shalom and E. Tse, “Tracking in a cluttered environment with probabilistic data association”, *Automatica*, Vol. 11, pp. 451-460, 1975.
- [32] J. Munkres, “Algorithms for the Assignment and Transportation Problems”, *J. Soc. Indust. Applied Math*, Vol. 5, No. 1, pp. 32-38, 1957.
- [33] Jain, A. K., “Fundamentals of Digital Image Processing”, Prentice Hall, Englewood Cliffs, NJ, 1989.
- [34] Cs. Rekeczky, I. Szatmári, D. Bálya, G. Tímár, and Á. Zarándy, “Cellular Multi-Adaptive Analogic Architecture: A Computational Framework For UAV Applications”, *IEEE Transactions on Circuits and Systems I*, Vol. 51, No. 5, pp. 864-884, 2004
- [35] T. DeMarco and T. Lister, “Peopleware – Productive Projects and Teams”, 2nd ed., Dorset House Publishing, New York, NY, 1999.
- [36] O. E. Drummond “Methodology for Performance Evaluation of Multitarget Multisensor Tracking”, *Signal and Data Processing of Small Targets*, 1999, Proc. SPIE, Vol. 3809, pp. 355-369

5.3 Publications Related to Offline Handwriting Recognition

- [37] Ertugrul Saatci and Vedat Tavsanoğlu: “Multiscale Handwritten Character Recognition Using CNN Image Filters”, *Proceedings of the 13th International Joint Conference on Neural Networks*, IJCNN 2002, Honolulu, 2002.

- [38] Tamás Szirányi and József Csicsvári: “High-Speed Character Recognition Using a Dual Cellular Neural Network Architecture (CNND)”, *IEEE Transactions on Circuits and Systems*, Vol. 40, pp.223-231, 1993.
- [39] L. Schomaker and E. Segers: „Finding features used in the human reading of cursive handwriting”, *International Journal On Document Analysis And Recognition*, Vol. 2, pp.13-18, 1999.
- [40] G. Lorette: „Handwriting recognition or reading? What is the situation at the dawn of the 3rd millenium?”, *International Journal On Document Analysis And Recognition*, Vol. 2, pp.2-12, 1999.
- [41] T. Steinherz, E. Rivlin and N. Intrator: Offline cursive script recognition – a survey, *International Journal On Document Analysis And Recognition*, Vol. 2, pp.90-110, 1999.
- [42] G. Kim, V. Govindaraju and S. Shrihari: An architecture for handwritten text recognition systems, *International Journal On Document Analysis And Recognition*, Vol. 2, pp. 37-44, 1999.
- [43] A. El-Yacoubi, M. Gilloux, R. Sabourin and C.Y Suen: „An HMM-based approach for offline unconstrained handwritten word modeling and recognition”, *IEEE Transactions On Pattern Matching And Machine Intelligence*, Vol. 21, No. 8, pp.752-760, 1999.
- [44] S. N. Shrihari et al.: „Analysis of Textual Images Using The Hough Transform”, *Machine Vision and Applications*, Vol. 2, pp.141-153, 1989.
- [45] Y. Nakajima, S. Mori, S. Takegami and S. Sato: „Global methods for stroke segmentation”, *International Journal On Document Analysis And Recognition*, Vol. 2, pp. 19-23, 1999.
- [46] Il-Seok Oh and C.Y. Suen: „Distance features for neural network-based recognition of handwritten characters”, *International Journal On Document Analysis And Recognition*, Vol. 1, pp.73-88, 1998.
- [47] P. Gader, M. Mohamed and J.H. Chiang: „Comparison of Crisp and Fuzzy Character Neural Networks in Handwritten Word Recognition”, *IEEE Transactions On Fuzzy Systems*, Vol. 3, pp. 357-363, 1995.
- [48] M. Mohamed and P. Gader: „Generalized Hidden Markov Models – Part II: Application to Handwritten Word Recognition”, *IEEE Transactions On Fuzzy Systems*, Vol. 8, pp. 82-94, 2000.
- [49] Chiang: „A hybrid neural network model in handwritten word recognition”, *Neural Networks*, Vol. 11, pp.337-346, 1998.
- [50] Horváth G., Dunay R., Pataki B., Strausz Gy., Szabó T., Várkonyiné Kóczy A.: Neurális hálózatok és műszaki alkalmazásaik, Műegyetemi Kiadó, 1998.

- [51] MATLAB (4. és 5. Verzió). Numerikus módszerek, grafika, statisztika, eszköztárak, Typotex Kft. Elektronikus Kiadó, 1999.
- [52] Postal Service tests handwriting recognition system, (1999)
<http://www.govexec.com/dailyfed/0299/020199k1.htm>
- [53] A. Senior and A.J. Robinson: „An Off-line Cursive Handwriting Recognition System”, *IEEE Transactions On Pattern Matching And Machine Intelligence*, Vol. 20, pp. 309-321, 1998.
- [54] A. Senior LOB Database: ftp://svr-ftp.eng.cam.ac.uk/pub/reports/Senior_tr105.ps.Z
- [55] Y. Chen and J. Wang: „Segmentation of Single- or Multiple-Touching Handwritten Numeral String Using Background and Foreground analysis”, *IEEE Transactions On Pattern Matching And Machine Intelligence*, Vol. 22, pp.1304-1317, 2000.
- [56] Sayre: „Machine Recognition of Handwritten Words: A Project Report”, *Pattern Recognition*, Vol. 5, No. 3, pp. 213-228, 1973.
- [57] R.C Gonzales and R.E Woods: Digital Image Processing, Boston, Addison-Wesley, 1992.
- [58] Cs. Rekeczky, L. Chua : „Computing with Front Propagation: Active Contour and Skeleton Models in Continuous-Time CNN”, *Journal of VLSI Signal Processing*, Vol. 23, pp. 373-402, 1999.
- [59] ParaScript Inc. AddressScript Literature:
<http://www.parascript.com/objects/addressscript.pdf>
- [60] IBM Document Analysis and Recognition:
<http://www.almaden.ibm.com/cs/DARE/homepage.html> and
<http://www.almaden.ibm.com/cs/dare.html>
- [61] B. Chandler, Cs. Rekeczky, Y. Nishio, and A. Ushida: “CNN Template Optimization by Adaptive Simulated Annealing”, *Proceedings of the International Symposium on Nonlinear Theory and its Applications (NOLTA'96)*, pp. 445-448, Kochi, Japan, October 1996.
- [62] T. Roska and L. O. Chua, “The CNN Universal Machine: An Analogic Array Computer”, *IEEE Transactions on Circuits and Systems–II: Analog and Digital Signal Processing*, Vol. 40, pp. 163-173, March 1993.
- [63] P. Kinget and M. Steyaert. “Analog VLSI Integration of Massive Parallel Processing Systems”, Ed.Kluwer Academic Publishers, 1996.
- [64] P.P Civalierie and M. Gilli, “On Stability Of CNNs”, *Journal of VLSI Signal Processing*, Vol. 23, pp. 429-437, 1999.
- [65] K. R. Crouse and L. O. Chua, "Methods for Image Processing in Cellular Neural Networks: A Tutorial", *IEEE Trans. on Circuits and Systems*, Vol. 42, pp. 583-601, October 1995.

- [66] P. Thiran, K. R. Crouse, L. O. Chua, and M. Hasler, "Pattern Formation Properties of Autonomous Cellular Neural Networks", *IEEE Trans. on Circuits and Systems*, Vol. 42, pp. 757-774, 1995.
- [67] Tímár Gergely, Karacs Kristóf, "Offline-kézírásfelismerés hibrid neurális architektúrával", XXV. Országos Tudományos Diákköri Konferencia, Eger, 2001.
- [68] Morita M., Bortolozzi F., Facon J. and Sabourin R., „Morphological approach of handwritten word skew correction”, X SIBGRAPI'98, International Symposium on Computer Graphics, Image Processing and Vision, Rio de Janeiro, Brazil, Oct. 1998.
- [69] Graham R. L., Yao F. F.: „Finding the Convex Hull of a Simple Polygon”, *J. Algorithms*, Vol. 4, pp. 324-331, 1983.
- [70] Karacs Kristóf, „Kézírás-felismerés”, Budapesti Műszaki és Gazdaságtudományi Egyetem, 2001.
- [71] S. Edelman, T. Flash, and S. Ullman., „Reading cursive handwriting by alignment of letter prototypes”, *International Journal of Computer Vision*, Vol. 5, pp. 303-331, 1990.
- [72] M. Cheriet and C. Y. Suen, „Extraction of key letters for cursive script recognition”, *Pattern Recognition Letters*, Vol. 14, pp.1009-1017, 1993.
- [73] M. Cote, E. Lecolinet, M. Cheriet, and C. Y. Suen, „Automatic reading of cursive scripts using human knowledge”, Proceedings Of The Int. Conf. on Document Analysis and Recognition, pp. 107-111, 1997.
- [74] R. Plamondon and S. Srihari: “On-Line and Off-Line Handwriting Recognition: A Comprehensive Survey”, *IEEE Transactions On Pattern Matching And Machine Intelligence*, Vol. 22, pp. 63-84, 2000.
- [75] S. Madhvanath and V. Govindaraju: “The Role of Holistic Paradigms in Handwritten Word Recognition”, *IEEE Transactions On Pattern Matching And Machine Intelligence*, Vol. 23, pp.149-164, 2001.

5.4 The Author's Publications

5.4.1 Journal Papers

- [1] **G. Timar**, Cs. Rekeczky: “A real-time multitarget tracking system with robust multichannel CNN-UM algorithms”, *IEEE Transactions on Circuits and Systems I*, Vol. 52(7), pp. 1358 – 1371, July 2005
- [2] Cs. Rekeczky, I. Szatmari, D. Balya, **G. Timar**, A. Zarandy: “Cellular multiadaptive analogic architecture: a computational framework for UAV applications”, *IEEE Transactions on Circuits and Systems I*, Vol. 51(5), pp. 864 – 884, May 2004

- [3] **G. Tímár**, K. Karacs, Cs. Rekeczky, “Analogic Preprocessing and Segmentation Algorithms For Off-line Handwriting Recognition”, *Journal of Circuits, Systems and Computers*, Vol. 12(6), pp. 783-804, Dec. 2003

5.4.2 International Conference Papers

- [4] Cs. Rekeczky, **G. Tímár**: „Multiple Laser Dot Detection and Localization within an Attention Driven Sensor Fusion Framework”, *IEEE International Workshop On CNN Theory and Applications CNNA 2005*, Vol 1., May 2005
- [5] D. Balya, **G. Tímár**, I. Szatmári, Cs. Rekeczky: “Efficient off-line feature selection strategies for on-line classifier systems”, *IEEE International Joint Conference on Neural Networks IJCNN 2004*, Vol. 1, July 2004
- [6] Cs. Rekeczky, **G. Tímár**, D. Balya, I. Szatmári, A. Zarandy: “Topographic and non-topographic neural network based computational platform for UAV applications” *IEEE International Joint Conference on Neural Networks IJCNN 2004*, Vol. 3, July 2004, pp:1763 - 1768
- [7] D. Balya, **G. Tímár**, Gy. Cserey, T. Roska: “A New Computational Model for CNN-UMs and its Computational Complexity”, *IEEE International Workshop On CNN Theory and Applications CNNA 2004*, Vol 1., July 2004
- [8] **G. Tímár**, D. Balya: “Regular small-world cellular neural networks: key properties and experiments” *International Symposium on Circuits and Systems, ISCAS 2004*. Vol. 3, May 2004 pp. III - 69-72
- [9] D. Bálya, **G. Tímár**, I. Szatmári, and Cs. Rekeczky: "Classification of Spatio-Temporal Features: the Nearest Neighbor Family", *IEEE European Conference on Circuit Theory and Design ECCTD 2003*, Krakow, Sept., 2003
- [10] **G. Tímár**, Cs. Rekeczky, L. Orzó and Sz. Tóké: " Sensing-Computing-Actuation in a Multi-Target Tracking Framework", *IEEE European Conference on Circuit Theory and Design ECCTD 2003*, Krakow Sept., 2003
- [11] **G. Tímár**, D. Bálya, I. Szatmári, and Cs. Rekeczky: „Feature Guided Visual Attention with Topographic Array Processing and Neural Network-Based Classification”, *Proc. International Joint Conference on Neural Networks*, Portland, USA, July 20-24 2003.
- [12] Szatmári, D. Bálya, **G. Tímár**, Cs. Rekeczky, and T. Roska: "Multi-Channel Spatio-Temporal Topographic Processing for Visual Search and Navigation", *SPIE Microtechnologies for the New Millennium 2003*, Gran Canaria May, pp. 297-306

- [13] Cs. Rekeczky, D. Bálya, **G. Tímár**, and I. Szatmári: "Bio-Inspired Flight Control and Visual Search with CNN Technology", *IEEE International Symposium on Circuits and Systems ISCAS 2003*, Bangkok May, pp.III-774-777
- [14] **G. Tímár**, K. Karacs, Cs. Rekeczky, "Analogic Preprocessing and Segmentation Algorithms For Off-line Handwriting Recognition", *Proc. 7th IEEE International Workshop on Cellular Neural Networks and their Applications*, Frankfurt am Main, Germany, July 2002., pp. 407-414
- [15] Cs. Rekeczky, **G. Tímár**, and Gy. Cserey, „Multi-Target Tracking With Stored Program Adaptive CNN Universal Machines”, *Proc. 7th IEEE International Workshop on Cellular Neural Networks and their Applications*, Frankfurt am Main, Germany, July 2002., pp. 299-306

6 Appendix: CNN Templates, Operators and Subroutines

This appendix lists the templates, basic operators and subroutines used in the dissertation. Where the Ace4k implementations are identified as ‘not stable’, it means that the operation of that particular function could not be implemented in a way that could be run reliably and repeatedly across different Ace4k chips. The operations themselves are possible and stable in principle, but not on the Ace4k.

6.1 Basic Operators

Threshold – thresholds a grayscale input image at a given grayscale level. The output is a binary image defined as follows:

$$\text{Thr}(\Phi_{ij}, \vartheta) = \begin{cases} 1 & \text{if } \Phi_{ij} \geq \vartheta \\ 0 & \text{otherwise} \end{cases}$$

CNN implementation: by using the THRESH template.

Ace4k, Ace16k implementation: available (not stable).

Erode – calculates erosion of a binary input image with a specified structuring element B . The set theoretical definition of the erosion based on Minkowski subtraction is as follows (- denotes translation):

$$\text{Erode}(\Phi_{Bin}, B) = \Phi_{Bin} \otimes B = \cap \{ \Phi_{Bin} - b : b \in B \}$$

CNN implementation: by using the EROSION template (feed-forward single-step erosion using B-templates) or PROPE (feedback continuous erosion by a trigger-wave using A templates).

Ace4k, Ace16k implementation: iterated single step morphology - available (stable), continuous trigger-wave computing – available (not stable).

Dilate - calculates dilation of a binary input image with a specified structuring element B . The set theoretical definition of the dilation based on Minkowski addition is as follows (+ denotes translation):

$$Dilate(\Phi_{Bin}, B) = \Phi_{Bin} \oplus B = \cup \{ \Phi_{Bin} + b : b \in B \}$$

CNN implementation: by using the DILATION template (feed-forward single-step dilation using B-templates) or PROPD (feedback continuous dilation by a trigger-wave using A templates).

Ace4k, Ace16k implementation: iterated single step morphology - available (stable), continuous trigger-wave computing – available (not stable).

Reconstruct* - calculates conditional (specified by a binary mask M) dilation of a binary input image with a specified structuring element B . The set theoretical definition of the reconstruction based on Minkowski addition is as follows (+ denotes translation):

$$Rec(\Phi_{Bin}, B, M_{Bin}) = \Phi_{Bin} (+)_{M_{Bin}} B = \cup \{ (B + \phi) \cap M_{Bin} : \phi \in \Phi_{Bin} \}$$

CNN implementation: by using the RECONSTR template (single-step conditional dilation) or PROPR (conditional continuous dilation by a trigger-wave).

Ace4k, Ace16k implementation: iterated conditional single step morphology - available (stable), continuous conditional trigger-wave computing – available (not stable).

Sobel – enhances the edges on a grayscale input by performing a convolution with a nearest neighbor directional Sobel-type operators (this assumes an 8-connected image):

$$\Phi_{Out} = S(\Phi_{Gray}, \sigma)$$

$$\sigma = 1:$$

$$B_1^{SH} = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} B_2^{SH} = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} B_2^{SV} = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} B_1^{SV} = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}$$

$$\Phi_{SH} = (\Phi_{Gray} * B_1^{SH}) + (\Phi_{Gray} * B_2^{SH})$$

$$\Phi_{SV} = (\Phi_{Gray} * B_1^{SH}) + (\Phi_{Gray} * B_2^{SH})$$

$$\Phi_{Out} = \Phi_{SH} + \Phi_{SV}$$

CNN implementation: by using the different variants of the SOBEL template.

Ace4k, Ace16k implementation: available (stable).

Laplace – enhances the edges on a grayscale input by performing a convolution with the nearest neighbor discrete Laplace operator (the image can be either 4-connected or 8-connected):

* This operator is also called “Recall” in the CNN literature

$$\Phi_{Out} = L(\Phi_{Gray}, \sigma)$$

$$B^{(4)} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}, B^{(8)} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

$$\Phi_0 = \Phi_{Gray}$$

$$\text{for } i = 1 \text{ to } \sigma$$

$$\Phi_0 = \Phi_0 * B$$

$$\text{end}$$

$$\Phi_{Out} = \Phi_0$$

CNN implementation: by using the different variants of the LAPLACE template.

Ace4k, Ace16k implementation: available (stable).

Gauss – calculates a low-pass filtered version of a grayscale image by performing a convolution with the nearest neighbor discrete Gaussian operator (the image can be either 4-connected or 8-connected):

$$\Phi_{Out} = G(\Phi_{Gray}, \sigma)$$

$$B^{(4)} = \frac{1}{4} \times \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}, B^{(8)} = \frac{1}{12} \begin{pmatrix} 1 & 2 & 1 \\ 2 & 0 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

$$\Phi_0 = \Phi_{Gray}$$

$$\text{for } i = 1 \text{ to } \sigma$$

$$\Phi_0 = \Phi_0 * B$$

$$\text{end}$$

$$\Phi_{Out} = \Phi_0$$

CNN implementation: by using the GAUSS template.

Ace4k, Ace16k implementation: available (stable).

Diffuse – calculates a linear low-pass filtered version of a grayscale input image. The formulation of the operation is as follows (* denotes convolution):

$$Diffus(\Phi, \sigma) = \Phi * G(\sigma)$$

CNN implementation: the above equation describes a linear convolution by a Gaussian kernel. Under fairly mild conditions at some time t this corresponds to the solution of a diffusion type partial differential equation. After spatial discretization this can be mapped to a CNN structure programmed by template DIFFUS. In this form the transient length is explicitly related to G ($t \approx \sqrt{\sigma_1}$).

Ace4k implementation: iterated convolution - available (stable), continuous diffusion – available (not stable).

Ace16k implementation: continuous diffusion using the chip's resistive grid capability

CDiffuse – calculates a linear low-pass filtered version of a grayscale input image. The formulation of the operation is as follows (* denotes convolution):

$$CDiffus(\Phi_1, \Phi_2, \alpha, \sigma_1, \sigma_2) = \alpha \Phi_1 * G_1(\sigma_1) + (1 - \alpha) \Phi_2 * G_2(\sigma_2)$$

CNN implementation: the above equation describes a homotopy in between two different linear convolutions by a Gaussian kernel. Under fairly mild conditions at some time t this corresponds to the solution of a constrained diffusion type partial differential equation. After spatial discretization this can be mapped to a CNN structure programmed by template CDIFFUS. In this form the B term directly approximates G_2 , while the transient length is explicitly related to G_1 ($t \approx \sqrt{\sigma_1}$).

Ace4k implementation: iterated convolution - available (stable), continuous diffusion - available (not stable).

Ace16k implementation: continuous diffusion using the chip's resistive grid capability and on chip image arithmetic

6.2 Subroutines

Open – calculates N - step opening on a binary input image:

```

 $F_{Open}^{(n)} = Open(F_{Bin}, B, p_n)$ 
   $F_0 = F_{Bin}$ 
  for  $i = 1$  to  $p_n$ 
     $F_0 = Erode(F_0, B)$ 
  end
  for  $i = 1$  to  $p_n$ 
     $F_0 = Dilate(F_0, B)$ 
  end
 $F_{Open}^{(n)} = F_0$ 

```

Close – calculates N - step closing on a binary input image:

```

 $\Phi_{Close}^{(n)} = Close(\Phi_{Bin}, B, p_n)$ 
   $\Phi_0 = \Phi_{Bin}$ 
  for  $i = 1$  to  $p_n$ 
     $\Phi_0 = Dilate(\Phi_0, B)$ 
  end
  for  $i = 1$  to  $p_n$ 
     $\Phi_0 = Erode(\Phi_0, B)$ 
  end
 $\Phi_{Close}^{(n)} = \Phi_0$ 

```

6.3 Linear, Isotropic CNN Templates

$$A = \begin{bmatrix} a_2 & a_1 & a_2 \\ a_1 & a_0 & a_1 \\ a_2 & a_1 & a_2 \end{bmatrix}, \quad B = \begin{bmatrix} b_2 & b_1 & b_2 \\ b_1 & b_0 & b_1 \\ b_2 & b_1 & b_2 \end{bmatrix}, \quad \tilde{z}$$

Template	Feedback (A)			Control (B)			Threshold	BCond
	a_0	a_1	a_2	b_0	b_1	b_2	z	B_c
HOLE	3	1	0	4	0	0	-1	-1
FIGREC	4	0.5	0.5	4	0	0	3	-1
FIGEXT	1	0	0	8	1	1	-1	-1
PRUNE	3	0.5	0	0	0	0	-1.5	0
BPROP	3	0.25	0.25	0	0	0	3.75	-1
WPROP	3	0.25	0.25	0	0	0	-3.75	1
CPATCH	3	0.25	0.2	0	0	0	3.65	-1
CWPATCH	3	0.25	0.2	0	0	0	-3.65	-1
HOLLOW	3	0.25	0.25	0	0	0	2.25	-1
GETEP	3	0.25	0.25	0	0	0	-1.25	-1
THRESH	2	0	0	0	0	0	-0.5	X
EROSION4	0	0	0	1	1	0	-4	1
EROSION8	0	0	0	1	1	1	8	1
DILATION4	0	0	0	1	1	0	4	-1
DILATION8	0	0	0	1	1	1	8	-1
RECONSTR4+	0	0	0	1	1	0	4	-1
RECONSTR8+	0	0	0	1	1	1	8	-1
GAUSS4	0	0	0	0	1/4	0	0	ZF
GAUSS8	0	0	0	0	2/12	1/12	0	ZF
LAPLACE4	0	0	0	-1	1/4	0	0	ZF
LAPLACE8	0	0	0	-1	1/8	1/8	0	ZF
SOBEL*	0	0	0	0	*	*	0	ZF
PROPE	3	0.25	0.25	0	0	0	-3.75	1
PROPD	3	0.25	0.25	0	0	0	3.75	-1
PROPR	2	0.25	0.25	2	0	0	0.75	-1
DIFFUS	0	0.15	0.1	0	0	0	0	ZF
CDIFFUS	0	2/24	1/24	0	2/12	1/24	0	ZF

Figure 6.1 Table of the linear isotropic CNN templates used by the algorithms referred to in the paper

Remarks:

$t_p \geq 5\tau$: the transient length of a non-coupled CNN operation

+: this solution is a conditional iterative dilation

*: see the possible non-isotropic B operators in basic operator description

6.4 Linear, Non-Isotropic Templates

DPROP:

$$A = \begin{bmatrix} 0 & 1.75 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B = 0, \quad z = 3.75$$

LSE:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & -1 & -1 \end{bmatrix}, \quad z = -3$$

LNE: Same as **LSE**, but the B matrix must be rotated around the center element by 180°

VCCD:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & -1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad z = 0$$

CONN4SE:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & -1 & -1 \end{bmatrix}, \quad z = -3$$

CONN4SW, CONN4NW, CONN4NE: Same as **CONN4SW**, but the B matrix must be rotated around the center element by 90° , 180° and 270°

TOPEP:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 0 \\ -1 & 1 & -1 \\ -1 & -1 & -1 \end{bmatrix}, \quad z = -5.5$$

BOTEP:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 1 & -1 \\ 0 & 0 & 0 \end{bmatrix}, \quad z = -5.5$$

SKELE1:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 7 & -1 \\ 0 & -1 & 0 \end{bmatrix}, \quad z = -3$$

SKELE2:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 7 & 0 \\ -0.5 & -1 & -0.5 \end{bmatrix}, \quad z = -3.4$$

SKELE3:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 1 & 1 \\ -1 & 7 & 1 \\ 0 & -1 & 0 \end{bmatrix}, \quad z = -3$$

SKELE4:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} -0.5 & 0 & 1 \\ -1 & 7 & 1 \\ -0.5 & 0 & 1 \end{bmatrix}, \quad z = -3.4$$

SKELE5:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 7 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \quad z = -3$$

SKELE6:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} -0.5 & -1 & -0.5 \\ 0 & 7 & 0 \\ 1 & 1 & 1 \end{bmatrix}, \quad z = -3.4$$

SKELE7:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 7 & -1 \\ 1 & 1 & 0 \end{bmatrix}, \quad z = -3$$

SKELE8:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0 & -0.5 \\ 1 & 7 & -1 \\ 1 & 0 & -0.5 \end{bmatrix}, \quad z = -3.4$$

SKLHV1:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} -1 & -1 & 0 \\ -1 & 7 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \quad z = -2$$

SKLHV2 ... SKLHV8: Same as SklHV1, but the B matrix must be rotated around the center element one by one