VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

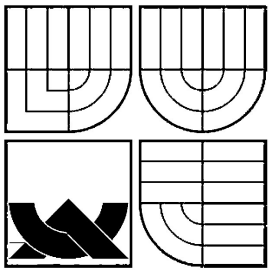
HARDWARE-BASED OBJECT DETECTION METHOD

DISERTAČNÍ PRÁCE
DOCTORAL THESIS

AUTOR PRÁCE
AUTHOR

ING. LUDĚK BRYAN

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

HARDWAROVĚ ORIENTOVANÁ METODA DETEKCE OBJEKTŮ
HARDWARE-BASED OBJECT DETECTION METHOD

DISERTAČNÍ PRÁCE
DOCTORAL THESIS

AUTOR PRÁCE
AUTHOR
ING. LUDĚK BRYAN

VEDOUCÍ PRÁCE
SUPERVISOR
DOC. ING. VLADIMÍR DRÁBEK, CSC.

BRNO 2007

Licenční smlouva
poskytovaná k výkonu práva užít školní dílo

uzavřená mezi smluvními stranami:

1. Pan/paní

Jméno a příjmení: **Luděk Bryan**

Bytem: **Vrchlického 6, 664 51, Šlapanice**

Narozen/a (datum a místo): **1.1.1979 Svitavy**

(dále jen „autor“)

a

2. Vysoké učení technické v Brně

Fakulta **informačních technologií**

se sídlem **Božetěchova 2, 612 66, Brno**

jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....

(dále jen „nabyvatel“)

Čl. 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
   - ☒ disertační práce
   - ☐ diplomová práce
   - ☐ bakalářská práce
   - ☐ jiná práce, jejíž druh je specifikován jako .....

   (dále jen VŠKP nebo dílo)

Název VŠKP: **Hardware-Based Object Detection Method**

Vedoucí/ školitel VŠKP: **Doc. Ing. Vladimír Drábek, CsC**

Ústav: **Ústav počítačových systémů**

Datum obhajoby VŠKP: _____

VŠKP odevzdal autor nabyvateli v*:

- ☒ tištěné formě – počet exemplářů ……..
- ☒ elektronické formě – počet exemplářů ……..

* hodící se zaškrtněte

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizovaní výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti
   - ☒ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: …….

…….. ……
Nabyvatel Autor

Abstrakt

V této práci je navržena nová hardwarová metoda detekce objektů v obraze. Základem této metody je paralelní extrakce příznaků pomocí jednoduchého porovnávání vzorů. Díky paralelismu je metoda vhodná pro hardwarovou implementaci. Součást práce je také experimentální architektura, založená na implementaci v FPGA. Navržená metoda může být rozšířena o adaptivní učící se systém, který automaticky nastavuje parametry v závislosti na okolním prostředí. V práci je navržen tento adaptivní systém pomocí dynamické rekonfigurace FPGA. Na ukázku toho, že navržená metoda může být použitá pro řešení problémů v reálném prostředí, byla navržena a vyhodnocena případová studie zabývající se detekcí státní poznávací značky v obraze.

Abstract

This thesis presents a new hardware designed object detection method. The core of the proposed method is a highly parallel feature extraction performed by simple template matching elements. Due to the high parallelism the proposed method is suitable for hardware implementation. Part of the thesis is also an experimental architecture based on FPGA implementation. The proposed method can be enhanced by an adaptive learning system that automatically sets the method parameters according to environment conditions. An adaptive learning system implementation using FPGA dynamic reconfiguration is suggested in the thesis. To show that the proposed method can be used for real life situations, a case study dealing with license plate detection has been evaluated.

Klíčová slova

Detekce objektů, hardware, FPGA, počítačové vidění, porovnávání vzorů, rekonfigurace, státní poznávací značka

Keywords

Object detection, hardware, FPGA, computer vision, template matching, reconfiguration, license plate

Luděk Bryan: Hardware-Based Object Detection Method, disertační práce, Brno, FIT VUT v Brně, 2007

Declaration

This thesis is a result of my own work. Section 8.1 was done in collaboration with Dr. Otto Fučík. This work was not submitted for a degree at any other university. Some parts of this thesis have been already published as articles [1] [2] [3] [4] [5] [6] [8] [11] [12] [13] [15] [16] [17].

Luděk Bryan

Acknowledgements

I would like to thank to Doc. Vladimír Drábek and Dr. Otto Fučík for their support and encouragement.

Contents

Chapter 1

Introduction

Forsyth and Ponce [24] defined the goal of computer vision "... to model and automate the process of visual recognition, a term we interpret broadly as perceiving distinctions between objects with important differences between them." Computer vision has been an important part of world-wide research since the 1970s, when computers started to be capable of processing large amounts of data. The computer vision field may serve for many purposes, including surveillance, robot control (Figure 1.1), autonomous vehicle driving, image set organization, scene analysis, or face detection.

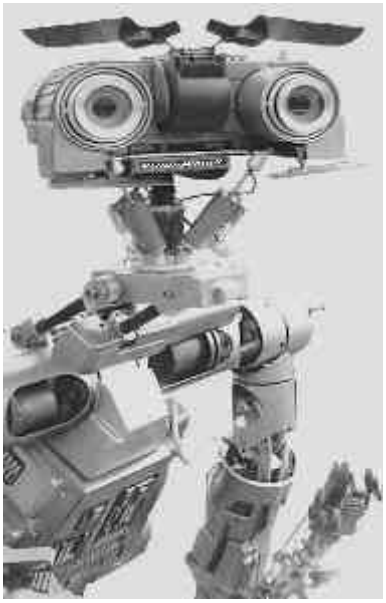


Figure 1.1: Example of an autonomos robot system

Another relatively new area of computer science is programmable hardware (PLDs, Section 2.1), which deals with hardware circuits capable of changing their internal structure, even during circuit operation (Section 2.4). PLDs are largely used in *embedded systems* (Chapter 2).

The goal of this thesis is to link these three fields - computer vision, embedded systems and programmable hardware, by suggesting a new method for object detection and designed for programmable hardware implementation.

1.1 Why Is a New Method Being Proposed for PLD Implementation in Embedded Systems?

Software solutions for computer vision are today very well explored. However, a different situation is in the field of VLSI design. This area is now quickly growing as new devices are on the market, both powerful and reasonably priced. Mainly PLD[1] devices are now affordable even for small businesses.

An important feature of PLD chips is the possibility to be reprogrammed by the user. This feature, called reconfiguration, can be used not only for debugging and standard operation, but also for some more advanced operations. Mainly *dynamic reconfiguration* allows the user to quickly reprogram a design (or even a part of one) on a chip. Thus, completely new computer-related methods are being developed such as evolutionary computation, and evolvable hardware.

Embedded systems are becoming part of many things of everyday use, including cars, and home appliances. Making a design for an embedded system is comparable to designing a standard system. The biggest difference is a design space limited by price, size, and power consumption. It makes sense though to create the method suitable for the embedded systems, which significantly extends possible range of target applications.

To summarize, as new technologies have developed, a gap has risen in the field of computer vision techniques targeting PLDs, while this area is becoming more important as small businesses can use powerful PLDs for their embedded systems.

1.2 Why Do We Need Hardware Methods?

As stated by Leibson [109], the majority of programers today are used to sequential thinking. This was mostly caused by the dominating position of purely sequential computers for a long period of time. As predicted by Liebson, this habit has to change as the multi-core processors and PLD chips are getting a bigger share of the market.

With the mentioned considerations, one question naturally comes to mind: What are the advantages of hardware design (either general VLSI design, or specifically PLD design) over standard processor programming?

- The design can be highly parallel.
- Comparing to parallel processor or processor arrays, implementation of any arbitrary parallel algorithmis is straightforward, not limited by the processor instruction set.
- Real-time designs are suitable.
- Custom behaviour of inputs / outputs is natural.
- Lower power consumption can be reached. Govindu et al. [111] compared CPU and PLD implementation of the same algorithm and the PLD implementation was nearly 8 times more energy efficient.
- For PLDs, chip reconfiguration is possible.

Of course, there are also disadvantages of hardware design:

[1] Topic of PLD chips will be discussed in Section 2.1

- Implementation phase is difficult and time consuming.
- Sequential behaviour is expensive and difficult to implement.

As a result, it's not possible to say universally whether the processor programming or hardware design is better. Some tasks are suitable for hardware implementation, while others are more suitable for the processor. Usually, the most efficient way is by using either the processor, for naturally sequential or easy tasks, and parallel processors or a hardware custom design, together with a processor, for difficult or naturally parallel tasks.

1.3 What is a Method Designed for Hardware?

There are many methods that can be efficiently *implemented* in hardware. This thesis will go a little deeper and will also distinguish methods *designed* for hardware. What should be the features of such a method?

- Intended for hardware implementation
- Naturally parallel
- Not efficient software implementation
- Quick or immediate reaction required (software implementation may suffer from interrupt delays)
- Use of specific hardware features, like reconfiguration for PLD chips

A method designed for hardware is more of an abstract idea, the listed items are more a clue than a definition. Some of the methods that fall into the category are listed in Section 3.4.8.

1.4 Chapter Description

In Chapters 2 and 3, all necessary theoretical basis are laid. Also, image algebra used in latter chapters is introduced. Chapter 4 defines the goals of the thesis. Based on those goals, detection method is suggested in Chapter 5. In Chapter 6, experimental architecture of the proposed method is suggested. To show that the proposed method can be used in real-life projects, the method was utilized in two case studies in Chapters 8 and 9. The whole thesis is summarized and results are discussed in Chapters 10 and 11.

1.5 Chapter Conclusions

In the introduction chapter, three fields important for this thesis are discussed - computer vision, embedded systems and programmable hardware. The topic of this thesis is laid - suggesting a new method for object detection designed for programmable hardware implementation. Following, questions concerning hardware are asked and answered, explaining certain concepts and decisions made in this thesis.

Chapter 2

Technological Background

As discussed in Chapter 1, an experimental architecture for the proposed method will be presented with the case studies following. For this reason, different ways of designing different architectures will be discussed in this section, which will delineate embedded system field and PLDs from other architectures.

2.1 Digital Devices

This section is an introduction into digital system implementation, because the proposed detection method is not intended to be implemented on a standard processor system. Digital devices discussed in this section will be introduced in Section 2.2 from the other perspective, i.e. which devices can be used for which type of computing systems. There are three ways how to implement a digital system.

1. *Processor* is a standard device for implementing a digital system. There are many types of processors, ranging from general purpose processors through cheap low-power single-chip processors to specialized digital signal processors (DSPs). Processor advantages are mainly their universality, easy application development and well established software support. Their disadvantage is serial processing of instructions, which comes from their nature. Although this disadvantage has been reduced significantly by special processor architectures (processor arrays, VLIW, DPSs ...), processors still can not compete with the special-purpose devices (discussed later) in the field of parallel tasks.

2. *ASIC* [87] stands for Application Specific Integrated Circuit, and is basically a logic circuit manufactured for a specific task. First, an expensive matrix of the circuit has to be made, which is used to manufacture actual chips. The advantages are very high performance and low price for a large series. The disadvantages are the high price of the matrix (thus high price for a small series), and difficult development and testing.

3. *Programmable logic devices* (PLDs) are circuits that allow the user to program their internal structure. They can benefit from a design specific for the application, while the development costs are lower than for ASICs and can also be used for a small series. Another advantage over ASICs is the possibility of the reconfiguration of some devices (Section 2.4). The disadvantages are much lower density and speed compared to the ASICs.

After the short introduction to different digital components, now we will focus on PLDs. There are several types of programmable logic devices. The first type, called simply PLDs, were followed by CPLDs (Complex Programmable Logic Devices). Although the CPLDs speed and density are low, they are still in use for their low power consumption and non-volatility. Today, FPGAs (Field Programmable Gate Arrays) are the most common programmable devices used.

2.2 Classification of Computer Systems

There are many ways in which computer systems can be classified (e.g. Flynn's classification [115]). We will use a more vague classification based on what devices are usually used in the system, adopted from [104]:

- *Supercomputers, or mainframes* consist of a large set of processors operating in parallel.
- *Desktop computers* use a processor or a set of a few processors as a primary operating unit. Performance can be boosted by additional board with PLD or ASIC chip(s).
- *Portable computers* are not primarily dedicated for intensive computations, and contain one (possibly multicore) processor.
- *PDAs, handhelds and cellphones* are relatively new devices where the tradeoff between performance and power consumption is a big issue. Custom-built processors are usually solution to this tradeoff.
- *Embedded systems* are a part of some consumer electronic device. Power issues may be important in the case of battery operation. Building blocks are usually a processor, a DSP processor for signal processing operations, or a PLD chip for I/O operation and performance boosting. For a large series, custom-built ASIC chips can be used.
- *Nanocomputers and biocomputers* do not exist yet, though there is a lot of research in these fields. In the future, these systems may be an answer to many problems addressed by todays systems.

For this thesis, important are the embedded systems, which will draw our attention for the rest of this section.

As can be seen, embedded systems are interlinked with many research ares. Digital devices are becoming part of many things of everyday use, for example cars and home appliances. These devices are also getting much smarter - cars are trying to avoid collision with another car, or a refrigerator finds out what food is missing and orders it on-line. Some of these "smart" tasks require human-like behaviour, such as object visual recognition or deciding how to react to arising situation.

These tasks must usually be processed in real-time, i.e. we must be sure to receive the solution in a constant maximum time limit. There are two reasons for that, we will look at them on an example of vision-based vehicle detection for collision avoidance [32].

1. *Quick reaction is required.* If a system is trying to avoid collision with another car, the brakes must be hit a very short time after car detection.

2. *Data flow must be continuous.* Images from video signal are processed in order to detect a car, and a new image comes from the sensor every n milliseconds. If the computation took too long, image queue could be filled and come of the images would have to be discarded.

Today, designers are not limited to processor or ASIC implementation, as they can use large FPGAs for their designs at affordable cost. It allows demanding algorithms to be processed, utilizing the parallelism of hardware.

2.3 FPGAs

We assessed PLDs to be the target architecture for the proposed detection method. Choosing between the types of PLDs, the most suitable will be an FPGA for its large logic resource, possibility of fast reconfiguration speed, and reasonable price. In this section, we will cover more closely some topics related to FPGA design.

FPGAs are programmable logic devices based on SRAM technology that allow high speed and density, but are volatile. The first FPGA was introduced in 1985 by Xilinx [94]. Today leading manufacturers of FPGAs are Xilinx, Altera, Lattice, and Actel. In the following text, only Xilinx devices will be considered.

The Xilinx FPGA structure [88] (Figure 2.1) consists of *input/output buffers* (IOBs), *configurable logic blocks* (CLBs) carrying logic function and registers, and interconnecting routes between CLBs. A CLB is a basic building block for the design function, contains two or four (depending on the device) slices, which consist of two 4-input look-up tables and two flip-flops.

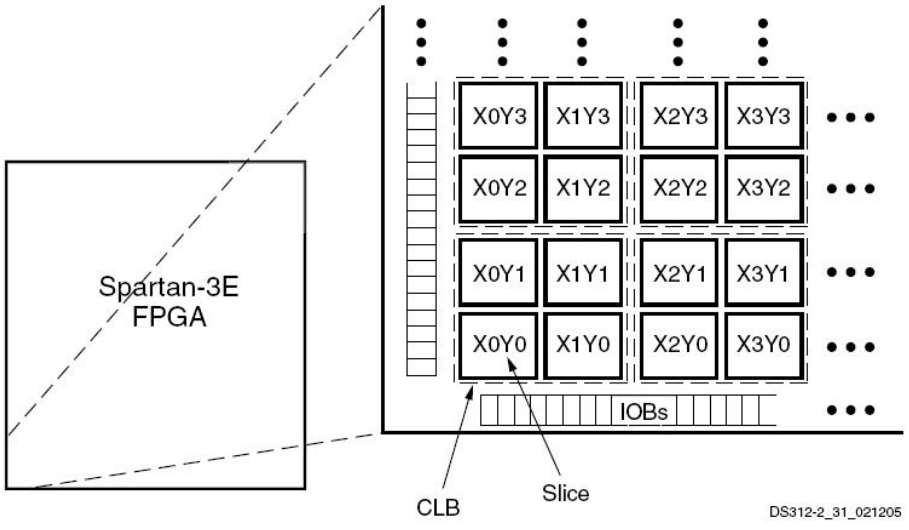


Figure 2.1: Spartan-3E internal structure

Standard FPGAs use a number of small, fast on-chip memory blocks, called block RAM (SelectRAM^{TM}for Xilinx products), or BRAM for short. The BRAM standard size for low-end FPGAs is 512 kB. Each of these memory blocks can be accessed from two independent ports, which make them a very powerful in a variety of designs.

2.3.1 HDL, Simulation and Synthesis

There are several ways how to design the structure of an FPGA.

1. *CAD system* is a program where the designer draws a scheme from basic elements like logic gates or an adder.
2. *A Hardware Description Language* (HDL) is similar to a programming language, but is meant for designing hardware. The two most widely used HDLs are Verilog and VHDL.
3. *Higher level languages* like Handel-C [80] or System-C [81] are becoming more important in the field. An interesting language is SA-C [82] which was created for the development of image processing algorithms.

As a result of the average design size having grown significantly over the last two decades, HDLs have become a standard design tool. In this thesis, we will use exclusively VHDL.

Two basic operations that we can perform with an HDL are simulation and synthesis. A simulator is a computer program whose input is an HDL code, and output is the behaviour of this code. A synthesizer is a computer program whose input is an HDL code, and output is a design file that can be uploaded to an FPGA.

There are two important things that concern us and that we will present for every synthesis experiment.

1. *Occupation of an FPGA* will be shown absolutely as number of slices and relatively as a percentage occupation of a reference chip.
2. *Longest logic path delay* is very important, as its reciprocal value determines the maximum clock frequency.

The longest logic path is very important if we consider the synchronous design style, which is considered a standard today. FPGAs are already designed to comply with this design style as they contain a sufficient number of registers.

2.4 FPGA Reconfiguration

A major advantage of FPGAs is reprogrammability. Unlike with ASICs, a user can download a new design (or even a part of the design) to an FPGA, even if it is already wired into the system. With reprogrammability, following capabilities are gained:

1. It is possible to download a new version of the design to the chip, either during the operation (once every few weeks or months) or during the design development (once every few minutes).
2. If we need to perform certain space-demanding operations, but not all of them at once, we can replace the operation unit which is not being used by one that we currently need. This allows us fitting designs that would not fit into FPGA otherwise, or the other way around we can save money by buying a smaller FPGA.
3. Reconfiguration can be a part of an FPGA operation. For example, certain circuit structures can be modified according to the environmental changes. Similar task is reconfiguration in order to develop circuits using evolvable hardware methods.

2.4.1 Static Reconfiguration

The easiest way to perform a reconfiguration is to synthesize a whole new FPGA design, stop operation of the current one, reprogram the FPGA and issue the reset signal to set all the register values to their initial state. This procedure is called *static reconfiguration*. Static reconfiguration is suitable for case number 1, and eventually for some cases of 2 or 3.

2.4.2 Partial Static Reconfiguration

Sometimes, we may need to reprogram only a part of our design. A reason is usually shorter reprogramming time, if we need to make changes more often. The reprogramming procedure is similar to a static reconfiguration; the difference is that only a part of the FPGA is reprogrammed.

There are two ways how to obtain the bitstream for the partial reconfiguration.

- Use the standard synthesis tool with partial reconfiguration support. Then, for example, the bitstream can be obtained by synthesizing two FPGA designs (the initial one and modified one), and run a synthesizer tool over these designs that creates a partial design consisting only of the changes in these two designs.
- If we can not use the synthesizer for some reason to change the design parts (for example evolvable hardware methods in case 3), we can modify the existing bitstream quickly by special software, for example JBits from Xilinx Inc. This solution is much more difficult compared to previous one.

Partial static reconfiguration is suitable for cases 2 or 3, if we do not need the rest of the design to be running during reconfiguration.

2.4.3 Partial Dynamic Reconfiguration

Partial dynamic reconfiguration is similar to partial static reconfiguration, but it can be performed while the non-reconfigured part of the FPGA is running. Dynamic reconfiguration has to be supported by the FPGA chip. Today, most of the FPGAs on the market have this feature, sometimes with certain limitations (for example Spartan III family can reconfigure only the whole column of CLBs at a time). Besides obvious advantages like saving of time and not interrupted processing, dynamic reconfiguration allows us to reconfigure part of an FPGA by itself.

2.4.4 Reconfiguration Applications

With the growing size of PLDs, reconfiguration has become an important research topic over the last few years. In this section, we will go through a few examples of advanced usage of an FPGA reconfiguration.

A popular application suitable for reconfiguration is a crossbar implementation, where connections need to be changed over time. For example, Young et al. [92] developed a 928x928 port crossbar using dynamic reconfiguration on a Xilinx Virtex II 6000. Srivastava et al. [84] used reconfiguration for adaptive image filtering. Scalera et al. [91] used FPGA reconfiguration to switch the FPGA configuration from microphone sensor mode to image recognition mode after identifying an object by a sound.

In the automotive industry, Hübner et al. [93] proposed multi-function ECU (Electronic Control Unit) using a reconfigurable FPGA, instead of the classical use of multiple single-function ECUs.

Among many others, these few examples show that FPGA reconfiguration is a strong tool allowing us to use completely new, unexplored techniques.

2.5 Chapter Conclusions

In the technological background chapter, we discussed different ways how to design a computer system. Embedded systems and PLDs are important for this thesis. Particular attention was paid to FPGAs, which will be the target technology for the experimental architecture in Chapter 6. Reconfiguration capabilities of FPGAs were also discussed, as reconfiguration will be used in Chapter 7.

Chapter 3

Theoretical Background

In this chapter, we will discuss all important branches of science related to this thesis. The core of the thesis, which is situated mainly in Chapter 5, relies on two fields of study - Pattern Recognition (section 3.3), with an emphasis on Spatial Filters (section 3.2) and Object Detection (section 3.4). We will focus only on techniques related to images, i.e. we will not consider sound processing or other areas of signal processing. As this work is based on processing in programmable hardware, some issues will be discussed with regards to this.

3.1 Image Algebra

While studying relevant literature sources, it was difficult to find a suitable set of mathematical definitions for working with spatial filters, or with other techniques used in this thesis. The requirements were that a set should be easy to use and not imply an implementation style. For example, Davies [18] uses an algorithmic style of description which already implies an implementation style, it is therefore unsuitable. Jahne et al. [20] use a mathematical description style, but many things are left undefined, and thus definitions are rather more informative than comprehensive.

Ritter [22] developed a comprehensive theoretical background for image algebra based on sets, with applications for image operations [23]. Their notation is designed to be universal and can be used for virtually any description of image operations. However, this strength is also the main weakness of these definitions, as a description of even simple operations gets tedious and rather difficult to understand.

Based on these facts, an original theoretical background is defined in this section based on the *image* and *neighborhood* of a pixel, both defined as matrix and the *structuring element* defined as a set of ordered pairs. This notation is easy to use and sufficient to describe all results and computations throughout the thesis. This work has been published in [11].

Basic Considerations

For mathematical operations, we will use some of the symbols from the table located in the abbreviation section.

In the following definitions, terms that are being defined are underlined. To define ranges, the notation of a double dot will be used, e.g. if x can be any integer between 0 and n, the notation will be $x \in \{0..n\}$.

Image

First, let's define the basic entities. An *Image* (Definition 3.1.1) is a matrix of size w, h consisting of *pixels* (picture elements). The ordering of pixels in a matrix is based on standards used for image processing, pixel $0, 0$ is in the top left corner. Each pixel is defined by its color. For our purposes, we will consider only gray-scale images, where the only color used is gray, thus, pixel can be described by one integer value between 0 (black) and $d-1$ (white), where d is the number of gray levels[1], called *gray depth*. Together with image, we define a *pixel* as one element of the image matrix.

Definition 3.1.1 *Image G of size $w \times h$ and gray depth d is a matrix*

$$G = \begin{pmatrix} g_{00} & g_{10} & \cdots & g_{w0} \\ g_{01} & g_{11} & \cdots & g_{w1} \\ \vdots & \vdots & \ddots & \vdots \\ g_{0h} & g_{1h} & \cdots & g_{wh} \end{pmatrix}$$

where $g_{ij} \in \{0..d-1\} \forall i \in \{0..w-1\}, j \in \{0..h-1\}$

Value g_{ij} (pixel)of image G can be also written as $G_{i,j}$.

For easier notation, we also define the set of all images with certain width, height, and gray depth as $\mathcal{G}^d_{w,h}$, images with certain width and height $\mathcal{G}_{w,h}$ and the set of all images $\mathcal{G}$ are in Definition 3.1.2.

Definition 3.1.2 *Let us denote the set of all images of size $w \times h$ and gray depth d as*

$$\mathcal{G}^d_{w,h}$$

The set of all images of size $w \times h$ is

$$\mathcal{G}_{w,h} = \bigcup_{d \in \mathbb{N}} \mathcal{G}^d_{w,h}$$

and the set of all images is

$$\mathcal{G} = \bigcup_{\substack{w \in \mathbb{N} \\ h \in \mathbb{N}}} \mathcal{G}_{w,h}$$

Sometimes, we need to refer to a certain pixel. This is done through *coordinates* (x, y) (Definition 3.1.3), that are pointing to a pixel $G_{x,y}$.

Definition 3.1.3 *The coordinates of a pixel in an image $G \in \mathcal{G}_{w,h}$ is an ordered pair (x, y) s.t. $x \in \{0..w-1\}$ $y \in \{0..h-1\}$*

[1] During the text, we will also employ black/white images, i.e. pixel values will be only 0 and 1. This case is treated as normal gray-scale image with $d = 2$

Structuring Element

As we already said, filter is applied to the neighborhood of a pixel. For each pixel that we apply filter to, neighborhood has the same shape. Thus, we need to define the shape of the neighborhood we are applying filter to. This shape is usually called a *structuring element*. We will define structuring element as a set of ordered pairs of integer numbers (Definition 3.1.4). These ordered pairs will then be used as relative coordinates to the filtered pixel.

Definition 3.1.4 *Structuring element S is a set of ordered pairs of integer numbers, i.e.*

$$S \subseteq \mathbb{Z} \times \mathbb{Z}$$

Again, we will also define the set of all structuring elements (Definition 3.1.5).

Definition 3.1.5 *Let $\mathcal{S}$ be the set of all structuring elements.*

Pixel Neighborhood

Now, we can define the *neighorhood* of a pixel covered by a structuring element (Definition 3.1.6). Neighborhood is a matrix of the same size as an image, but contains only the actual pixels that are used as source values for a filter function. The rest of the matrix is filled with $\perp$.

Definition 3.1.6 *The neighborhood $\mathcal{N}_{x,y}^{G,S}$ of a pixel at coordinates (x,y) in image $G \in \mathcal{G}_{w,h}$ with structuring element $S \in \mathcal{S}$ is a matrix*

$$\mathcal{N}_{x,y}^{G,S} = \begin{pmatrix} n_{00} & n_{10} & \cdots & n_{w0} \\ n_{01} & n_{11} & \cdots & n_{w1} \\ \vdots & \vdots & \ddots & \vdots \\ n_{0h} & n_{1h} & \cdots & n_{wh} \end{pmatrix}$$

where $n_{x'y'} = G_{x',y'}\ \forall (x',y')\ s.t.\ \exists (i,j) \in S \quad \begin{matrix} x' = i + x, \\ y' = j + y, \end{matrix}$

$n_{x'y'} = \perp$ *otherwise*

Value n_{ij} of neighborhood N can be also written as $N_{i,j}$.

Definition 3.1.7 *The set of all neighborhoods with structuring element S is*

$$\mathcal{N}^S = \bigcup_{\substack{G \in \mathcal{G} \\ x \in \{0..w-1\} \\ y \in \{0..h-1\}}} \mathcal{N}_{x,y}^{G,S}$$

where w, h are width and height of an image G.
The set of all neighborhoods is

$$\mathcal{N} = \bigcup_{S \in \mathcal{S}} \mathcal{N}^S$$

Spatial Filter

Finally, a spatial filter (can be reffered to only as a filter) is defined in Definition 3.1.8. A filter transforms the neighborhood of any image at any coordinates, but with defined shape (structuring element), to a natural number. Thus, while defining a filter function, the structuring element is already known, but the filter can be applied to any pixel of any image. An actual filter function F is not specified here; its variants are described in more detail in Section 3.2.

Definition 3.1.8 *Filter of structuring element S is a function $F : \mathcal{N}^S \rightarrow \mathbb{N}$*

Definition 3.1.9 *The set of all filters of structuring element S is denoted*

$$\mathcal{F}_S$$

The filter is supposed to be applied to all pixels in an image. This is shown in Definition 3.1.10, where G and G' are the input image and the filtered image, respectively, and S is a structuring element of a filter.

Definition 3.1.10 *Let $G \in \mathcal{G}_{w,h}$ be an image, $F \in \mathcal{F}_S$ a filter. Filtered image $G' \in \mathcal{G}_{w,h}$ filtered by F of source image G is an image G' so that*

$$G'_{x,y} = F(N) \ s.t. \ N \in \mathcal{N}^{G,S}_{x,y} \ \forall x \in \{0..w-1\}, \forall y \in \{0..h-1\}$$

In Figure 3.1, you can see an example of an application of a square filter of size 3×3 to an image. The pixel marked X is the one we modified. The neighborhood of pixel is outlined by the thick line. The same filter is applied to all pixels in the image.

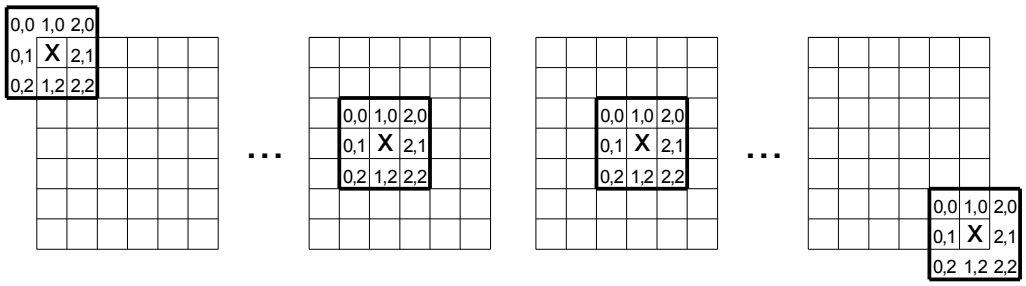


Figure 3.1: Example of filter application

Image Border

In Figure 3.1, you can see that not all pixels can be processed. Filtering pixels that are close to the border of an image may require accessing pixels outside an image. For example, let's consider a rectangular structuring element of size w_f and h_f. Then, the effected pixels are those with the coordinates x, y in Equation 3.1, where x_f and y_f are horizontal and vertical position of the filtered pixel in an image, w_f and h_f are the width and height of the filter, and w_i and h_i are the width and height of the image.

$$
\begin{array}{lllll}
G_{x,y}: & x \leq \frac{w_f-1}{2} & \wedge & 0 \leq y < h & \vee \\
 & 0 \leq x < w & \wedge & y \leq \frac{h_f-1}{2} & \vee \\
 & x > (w - \frac{w_f-1}{2}) & \wedge & 0 \leq y < h & \vee \\
 & 0 \leq x < w & \wedge & y_f > (h - \frac{h_f-1}{2}) &
\end{array}
\tag{3.1}
$$

In order to filter all pixels correctly, an image must be virtually enlarged so that missing pixels behind the border are replaced by made-up values (Definition 3.1.11).

Definition 3.1.11 *Enhanced image* GE *is a matrix*

$$
GE = \begin{pmatrix}
\ddots & \vdots & \vdots & \vdots & \\
\cdots & ge_{-1-1} & ge_{0-1} & ge_{1-1} & \cdots \\
\cdots & ge_{-10} & ge_{00} & ge_{10} & \cdots \\
\cdots & ge_{-11} & ge_{01} & ge_{11} & \cdots \\
 & \vdots & \vdots & \vdots & \ddots
\end{pmatrix}
$$

Value ge_{ij} *of extended image* GE *can be also written as* $GE_{i,j}$.

There are a few different solutions on how to extend an image. Let's assume the original image $G \in \mathcal{G}_{w,h}$ and resulting extended image GE

1. *Periodic extension*,

   $$GE_{x,y} = G_{x \bmod w, y \bmod h}$$

   which is theoretically correct method, is based on the assumption of the Fourier transform, that an image is periodically repeated in the whole space. In reality, the results are questionable, and hardware realization is not straightforward.

2. *Constant extension*

   $$GE_{x,y} = G_{x,y} : \forall x \in \{0..w-1\}, y \in \{0..h-1\}$$
   $$GE_{x,y} = \text{const otherwise}$$

   assumes constant color behind the image border. The advantage is easy hardware realization; the disadvantage is the occurrence of sharp edges between the image and the border.

3. *Extrapolating extension*

   $$
   \begin{array}{lll}
   GE_{x,y} = G_{x,y}: & \forall x \in \{0..w-1\}, & y \in \{0..h-1\} \\
   GE_{x,y} = G_{w-1,y}: & \forall x \in \{w..\infty\}, & y \in \{0..h-1\} \\
   GE_{x,y} = G_{0,y}: & \forall x \in \{-\infty..-1\}, & y \in \{0..h-1\} \\
   GE_{x,y} = G_{x,h-1}: & \forall x \in \{0..w-1\}, & y \in \{h..\infty\} \\
   GE_{x,y} = G_{x,0}: & \forall x \in \{0..w-1\}, & y \in \{-\infty..0\} \\
   GE_{x,y} = G_{0,0}: & \forall x \in \{-\infty..0\}, & y \in \{-\infty..0\} \\
   GE_{x,y} = G_{w-1,0}: & \forall x \in \{w..\infty\}, & y \in \{-\infty..0\} \\
   GE_{x,y} = G_{0,h-1}: & \forall x \in \{-\infty..0\}, & y \in \{h..\infty\} \\
   GE_{x,y} = G_{w-1,h-1}: & \forall x \in \{w..\infty\}, & y \in \{h..\infty\}
   \end{array}
   $$

   tries to adapt the outside pixels according to the pixel(s) on the border. In its easiest form, outside pixels are set to the closest border pixel. This solution can be efficient, and is affordable in hardware.

4. *Shrinking output image* considers $s_f/2$ (where s_f is size of a filter mask) border pixels not being used for filtering. This method can lead to very easy implementation, but shrinks the output image at every direction, which is in many cases unacceptable.

We defined universal filter with fully arbitrary structuring element. However, there are certain prevalent ways how to design a filter, either because it's easier or most efficient. Most common and natural choices for structuring element shape are specified in the following list:

- Filter mask is of rectangular shape, usually square.
- Width and height of the structuring should be odd-sized to keep the symmetry of the pixel with the result.

3.2 Spatial Filters

Spatial filters (Definition 3.1.8) are widely used for image processing applications. Their basic idea is to change pixel value according to values of this pixel and certain number of its neighbouring pixels. This procedure is applied to all pixels in an image. In this section, specific types of popular spatial filters will be discussed.

3.2.1 Linear Spatial Filters

Every cell in a structuring element of a linear spatial filter is a number, called weight. An example of weights in square mask of size 3×3 is in Figure 3.2. The result of an application of a linear spatial filter to one pixel is a two-dimensional convolution of a mask's weights and corresponding pixels in an image. This is described in Definition 3.2.1.

$W_{0,0}$	$W_{1,0}$	$W_{2,0}$
$W_{0,1}$	$W_{1,1}$	$W_{2,1}$
$W_{0,2}$	$W_{1,2}$	$W_{2,2}$

Figure 3.2: Weights in the mask for a linear filter

Definition 3.2.1 *Linear filter F is a filter over $\mathcal{S}$ so that*

$$\exists\{f_{i,j} \in \mathbb{R} | \forall (i,j) \in S\} \ s.t. \ \forall \ N = \mathcal{N}_{x,y}^{G,S}, \ F(N) = \sum_{(i,j)\in S} n_{x+i,y+j} \times f_{i,j}$$

Typical representatives of linear filters are shown in Figure 3.3 with corresponding sample images in Figure 3.4. A flat average filter works as low-pass filter; the neighborhood of every pixel is simply averaged and the result is a smoothed image. A high-pass filter works the other way around - because of negative values around the center pixel, high frequencies from an image are displayed.

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

-1	-1	-1
-1	8	-1
-1	-1	-1

Figure 3.3: Flat average and high-pass filters

Figure 3.4: Original image, flat average and high-pass filter

The problem of linear filters is their uniformity, they work the same way for the whole image. For example, in a flat average linear filter, this leads into blurring edges. An enhancement of linear filtering is *adaptive filtering* [83], where a filter adapts its coefficients according to the neighobrhood.

3.2.2 Morphological Filters

As can be seen from the nature of linear filters, they use all neighbouring pixel for computation. Thus, even if the pixel is significantly distorted, it still gets the same importance as any other pixel. For this and other reasons, we are forced to search outside the area of linear filters. Unlike linear spatial filters, their non-linear colleagues are not limited by applying convolution only, which allows them to perform arbitrary functions.

The best known group of non-linear spatial filters are those based on *mathematical morphology* [20]. Morphology is a theory about analyzing the shape and form of objects. The shape of a filter mask, which is called a structuring element, plays an important role for morphology filters.

Use of morphology in image processing applications is very wide, starting at common noise removal through segmentation methods [97] [98] to medical applications [95] [96].

Rank-Value Filters

Commonly used morphological filters are [99] dilation, erosion and median. These filters can be implemented as *rank-value filters*. Rank-value filters are based on sorting pixel values covered by a structuring element, and then picking one of these values. For erosion, the pixel with the minimum value is chosen, for dilation maximum, and for median the middle value is selected. Mathematical definitions of those filters are defined below.

Definition 3.2.2 *Dilation filter is a filter over* $\mathcal{S}$ *so that* $\forall N \in \mathcal{N}_{x,y}^{G,S}$

$$F(N) = max\{n_{ij} \in N\}$$

Definition 3.2.3 *Erosion filter is a filter over* $\mathcal{S}$ *so that* $\forall N \in \mathcal{N}_{x,y}^{G,S}$

$$F(N) = min\{n_{ij} \in N\}$$

Definition 3.2.4 *Let* $M \subseteq \mathbb{N}$ *is a set,* $x \in M$*.*

$$M_{<x} = \{y \in M | y < x\}$$
$$M_{>x} = \{y \in M | y > x\}$$

Median of set M *is then* $m \in M$ *so that*

$$|M_{<m}| - |M_{>m}| \leq 1$$

Median filter is a filter over $\mathcal{S}$ *so that* $\forall N \in \mathcal{N}_{x,y}^{G,S}$

$$F(N) = median\{n_{ij} \in N\}$$

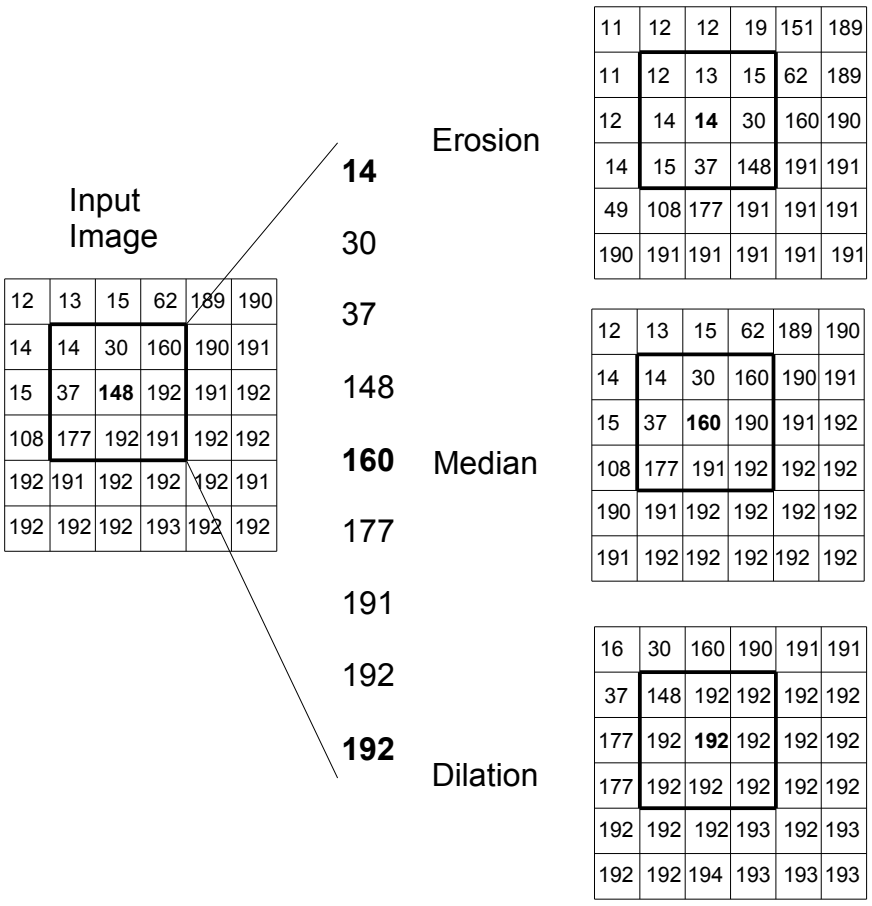


Figure 3.5: Erosion, median and dilation

In Figure 3.5, examples of applying erosion, median and dilation filters are depicted. The pixel that is being processed is printed bold, and the structuring element (rectangle 3×3) is contoured. The values inside the structuring element are then sorted, and an appropriate value is picked and placed as the result. For erosion the minimum value is selected, the medium value for median, and the maximum for dilation. This procedure (sorting the values covered by a structuring element and picking one) is done for every pixel in an image (Definition 3.1.10).

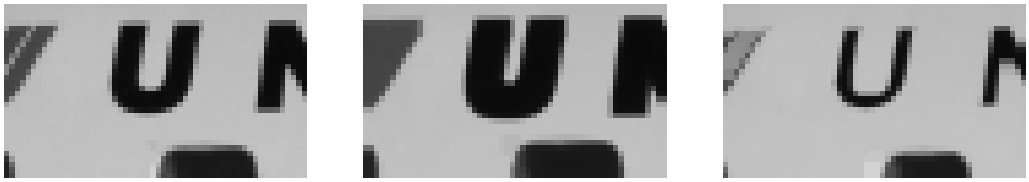


Figure 3.6: Original image, erosion and dilation

The examples in Figure 3.5 are fragments of real images. You can see the original image and application of erosion and dilation in Figure 3.6. It's clear from these images how dilation and erosion work - dilation expands the bright pixels (higher values) and erosion expands the dark pixels (lower values) into their neighborhoods. It may be confusing that erosion in this example makes the object (a letter) look thicker. The point is that the object here is dark, which virtually inverts the function of the rank filters.

Erosion and dilation filters are usually used for processing binary images, but can be extended to gray-level images, as we saw in the previous examples. Binary image processing can be very efficient in hardware, as necessary operations are reduced to logical AND / OR.

The notation of erosion operation is in Equation 3.2 and the dilation operation is in Equation 3.3, where G and G' are the original and the filtered image respectively; S is a structure element.

$$erosion: \ G' = G \ominus S \tag{3.2}$$

$$dilation: \ G' = G \oplus S \tag{3.3}$$

Filters consisting of application of dilation and erosion consecutively [31] are also particularly useful. Examples of those filters are *opening* in Equation 3.4 and *closing* in Equation 3.5. Opening and closing can be utilized, for example, for discarding excessively bright (or dark) pixel values while keeping the original shape of the objects.

$$opening: \ G' = G \circ S = (G \ominus S) \oplus S \tag{3.4}$$

$$closing: \ G' = G \bullet S = (G \oplus S) \ominus S \tag{3.5}$$

If we need to get rid of both extensively dark or bright pixels, we can use a combination of opening and closing. A combination of opening and closing is called *open-close* (Equation 3.6) and combination of closing and opening is *close-open* (Equation 3.7).

$$open - close: \ G' = (G \circ S) \bullet S \tag{3.6}$$

$$close - open: \; G' = (G \bullet S) \circ S \tag{3.7}$$

Use of a *median* filter is mainly related with noise-reduction. It is similar to a linear flat-average filter in the meaning that it reduces the occurrence of peak values in an image, but it preserves sharp edges and does not blur an image. The comparison of the median filter with the linear flat-average filter and the open-close filter is showed in Figure 3.7. There are also more advanced methods for image denoising based on median filtering, for example the one suggested by Eng et al. [77].

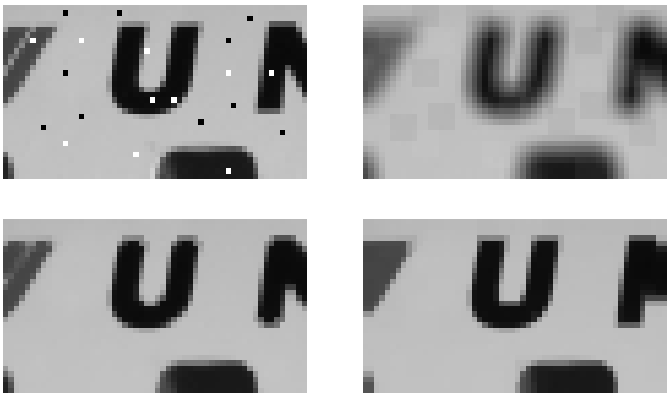


Figure 3.7: Original noisy image, flat average, median, open-close

Hit-or-Miss Filter

Hit-or-miss (or sometimes called hit-and-miss) is a morphological operation, usually used for binary images. In this thesis, we will assume binary implementation only. The structuring element of the hit-or-miss operation contains two possible values - representing the foreground and the background. An example of such structuring element is depicted in Figure 3.8.

	1	
1	0	1
	1	

Figure 3.8: Example of hit-or-miss structuring element

If the foreground and the background pixels in an image and in a structuring element match, the result of the filter application is 1 - see Definition 3.2.5.

Definition 3.2.5 *Hit-or-miss filter is a filter over $\mathcal{S}$ so that*

$$\exists\{f_{i,j} \in \{0,1\} | \forall (i,j) \in S\} \; s.t. \; \forall \; N = \mathcal{N}_{x,y}^{G,S}, \; F(N) = 1 \Leftrightarrow n_{x+i,y+j} = f_{i,j} : (i,j) \in S$$

A hit-or-miss filter can be used for pattern detection. For example, the filter with the structuring element shown in Figure 3.8 will detect one isolated background pixel between the four foreground pixels.

Usually, we need to detect more patterns together. The solution is simple - make more structuring elements, filter the input image with all of them and apply a logical OR operation to the resulting images. As an example, there are four structuring elements in Figure 3.9, each representing one diagonal edge. The application of all these filters is shown in Figure 3.10.

0	0
0	1
1	1

0	0
1	0
1	1

1	1
1	0
0	0

0	0
1	0
1	1

Figure 3.9: Structuring elements representing diagonal edges

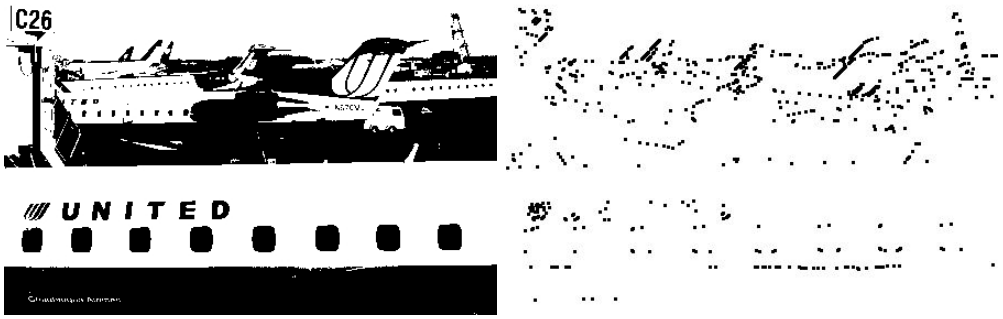


Figure 3.10: Original image and application of hit-or-miss filters

3.3 Pattern Recognition

Pattern recognition is a computer vision technique for extracting high-level information from an image. This can be useful for object detection, object tracking, defect detection or robotic systems.

Pattern recognition is usually decomposed into three step pipeline - *preprocessing*, *feature extraction* and *classification*. Each of these is described in the following chapters. However, in reality, many modifications of this pipeline exist, or some of the steps may be dissolved.

3.3.1 Preprocessing

Preprocessing enhances the image for easier processing during the next steps. This involves mainly removing noise from an image, but also much more sophisticated methods can be utilized. A list of feasible methods follows, with a brief description for each of them. For a complete reference list, see some of the image processing texts (i.e. [25]).

- *Point operators* are applied individually to every pixel in an image, without information about its neighborhood. Processing cost is usually low in comparison with other methods, but utilization is limited. Point operators include *thresholding*, *contrast stretching*, *linear*, or *non-linear operators*.
- *Spatial filters* are more thoroughly discussed in Section 3.2. Unlike point operators, transformation of a pixel is done according to its neighborhood, not only according to the pixel itself.
- *Histogram techniques* modify an image according to the changed shape of its histogram (briefly discussed in Section 3.4.5), or use histogram information for setting values for further processing.
- *Transform operations*, including the well-known Fourier or wavelet transform, do not just modify the image as previous methods do, but convert it to completely different form. Although transforms can be very powerful, implementation in hardware is very space consuming and difficult to implement (discussed in Section 3.4.3).

3.3.2 Feature Extraction

The second step is *feature extraction* and is the essential part of pattern recognition. The goal is to recognize features in an image that indicate the presence of regions of interest. In other words, we want to transform an image from a spatial domain to a feature domain, where data represent more abstract quantities. Usually, the features are placed in the *feature vector* $(v_1, v_2, \cdots, v_n)$.

If we use feature extraction for object detection or recognition, it should treat objects independently without regard to their placement or other properties. Usually, three characteristics are desirable [31]:

- *Resistance against changes in illumination* is usually required. We should propose algorithms sturdy enough to handle shifts in intensity. Otherwise, our system is limited to exact lighting conditions.
- *Rotational invariance* is necessary when the objects in an image or sensor are not constantly oriented.
- *Scale invariance* means that the algorithm should work, if possible, equally well for closer or more distant objects.

For feature extraction, methods have been previously suggested by many authors, dependent on the target application [20] [34], some of them in hardware [33] [31]. The spatial filters that we will use in Section 3.2 are one of the more popular methods. An edge detector is an example of commonly used method, because edges are usually an important clue for pattern recognition.

3.3.3 Classification

The last step, known as *Classification*, decides whether there are any regions of interest, and where, upon the found features.

Classification can be either *supervised* or *unsupervised*. For supervised classification, a training data set is provided and learning has to be performed prior to classification. In unsupervised classification, the classifier decides about types of regions in an image solely

from the information included in this image. In this thesis, we will deal only with the supervised classification.

The fundamental idea of classification [25] is to gather certain *features* from an image[2] during the feature extraction phase, and decide what class the image belongs to.

There are several classification methods. One of the simplest methods is the *minimum distance classifier*. Each class is defined by one point in the n-dimensional feature space. A feature vector belongs to the closest class.

Bayesian decision theory is considered universal statistical method for classification. This method has been thoroughly described in many books, for example by Duda et al. [19]. *Neural networks* are also often used as a classification technique [26] [100].

AdaBoost

Choosing the right set of features extracted from an image during the feature extraction is very important for successful classification. This issue is particularly important for the proposed method, as it is designed for hardware thus probably highly parallel and producing many features. That's why possible usage of the AdaBoost method in this thesis is discussed. In this section, a short introduction to the AdaBoost method is presented.

Boosting [113] refers to a general and effective method of producing a very accurate prediction rule by combining rough and moderately inaccurate rules. *AdaBoost* is a commonly used boosting method that was introduced in 1995 [112].

Input to the AdaBoost algorithm is a training set $(x_1, y_1) \cdots (x_m, y_m)$ where each x_i is in a domain space X and each y_i is in a label space Y. At the beginning, each training example (x_i, y_i) is assigned a *weight* $D_0(i)$, determining how important the training example is. At the beginning, all weights $D_0(i)$ are the same. A *learning algorithm* is then applied repeatedly in rounds $t = 1..T$. The purpose of the learning algorithm is to get the *weak hypothesis* $h_t : X \to \{-1, +1\}$. The error ϵ_t of the weak hypothesis is measured with respect to the distribution D_t:

$$\epsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$$

Coefficient α_t is derived from ϵ_t which shows the quality of weak hypothesis h_t:

$$\alpha_t = \frac{1}{2} ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

Weights of incorrectly classified examples are increased in order to give harder examples more attention in the next round:

$$D_{t+1}(i) = \frac{D_t(i)\ exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where Z_t is a normalization factor.

Strong classifier is a function of weak classifiers $h_t(x)$

$$H(x) = sign \left(\sum_{t=1}^{T} \alpha_t h_t(x) \right)$$

[2]Classified can be also an image slice, or even a pixel. In the text, only term *image* will be used

There are many possibilities for choosing weak classifiers. For example, Viola and Jones [53] suggest the Haar-like image features shown in Figure 3.11. The features are the difference between the sum of the pixels within rectangular regions (gray and white rectangles in Figure 3.11). Viola and Jones also suggest an *integral image* implementation for fast computation of these features.

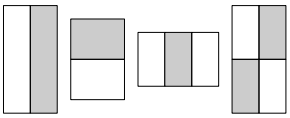


Figure 3.11: Example of features suggested by Viola and Jones

For a more thorough description of the AdaBoost learning algorithm, please refer to Freund's introduction [113] or a well written illustrative tutorial by Rätsch [110].

3.4 Object Detection

Object detection is a technique used in the computer vision branch of computer science. The goal of the object detection is to identify a specific object, or an object belonging to a specific class of objects, in an image.

Object detection is a very important part of computer vision. Object detection can either work as a stand alone method (object tracking, face detection, ...), or as a part of a larger system. Examples of larger systems are face detection with compression of the background following, or sign detection with OCR of the text following.

There are many object detection methods, with lots of different features and demands. It's impossible to make a completely universal solution for a reasonable cost, we must trade off between features like functionality, speed and hardware demands. In Figure 3.12, there is a brief overview of possible object detection methods. After preprocessing (de-noising, thresholding, domain transforms), feature extraction techniques (FE) take place. The focus is laid on FE in this thesis, therefore we will go through the FE methods mentioned in Figure 3.12 with a brief description of each of them. In Section 5.1, one technique discussed will be the basis of the method proposed in this thesis.

Following FE, classification takes place, introduced in Section 3.3.3. Classification is only a marginal part of this thesis; its application for the proposed method will be discussed in Section 5.3.3.

3.4.1 Edge Detection

There are two main methods of edge detection [18], *template matching* and the *differential gradient* method. There are also other methods for edge detection not discussed here, for example a median filter suggested by Zou et al. [62].

The template matching approach is basically linear filtering (Sectin 3.2.1)using a set of masks to detect a gradient. The field of masks used for edge detection is very well explored, and there are several sets widely used. An example of two of the masks of such a set is in Figure 3.13.

The differential gradient method takes the g_x and g_y gradients in the x axis and y

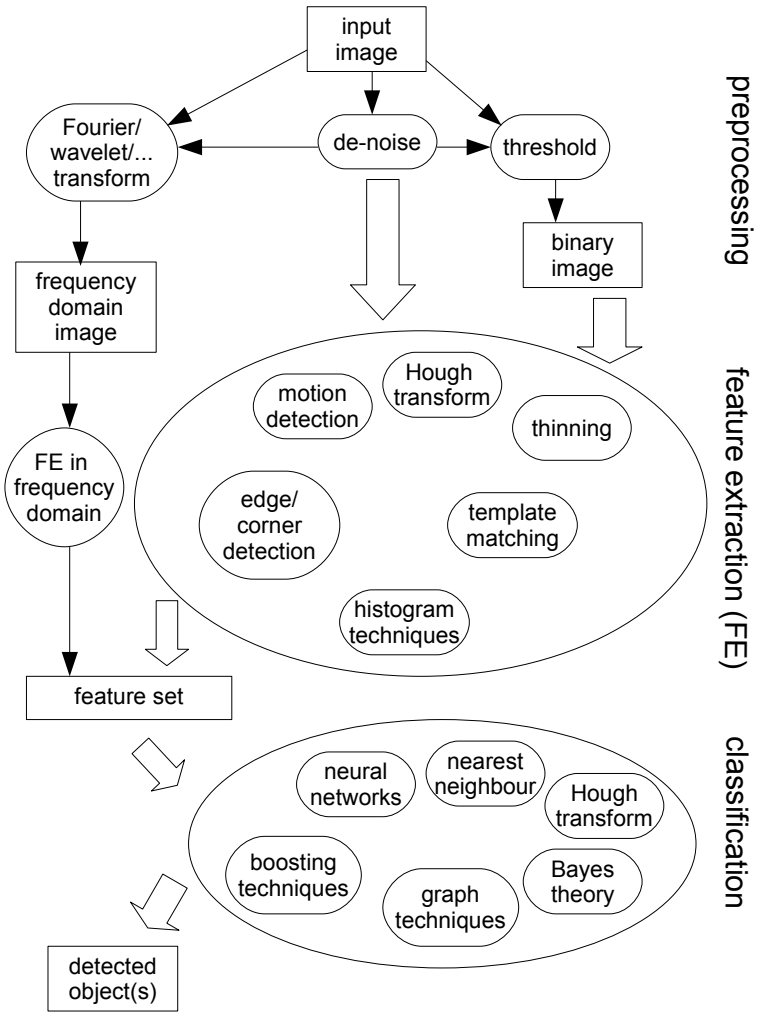


Figure 3.12: Pattern recognition for object detection

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}$$

Figure 3.13: Robinson "3-level" masks for 0° and 45°

axis, respectively, and computes the gradient g as

$$g = \sqrt{g_x^2 + g_y^2}$$

Both edge detection methods are relatively simple, but may give very good results under many circumstances. Edge detection is a technique suitable for hardware implementation, for example Torres-Huitzil et al. [85] suggested an edge detection method suitable for an FPGA implementation. An example of an edge detection application is in Figure 3.14. Usually, edge detection is only the first step in a method [86].

3.4.2 Hough Transform

The Hough transform, introduced in 1962 by Hough [57], detects particular shapes. It is commonly used for regular curves such as lines, circles and ellipses. However, a *generalized* Hough transform [58] can detect any type of shape. Here, we will talk only about the Hough transform used for line detection.

The main point behind the Hough technique for line detection is that each input measurement (e.g. coordinate point) indicates its contribution to a globally consistent solution (e.g. the physical line which gave rise to that image point) [25].

There is infinite number of lines that we can run through every point in an image. Each of these lines can be described by the length of a normal taken from the origin to the line r and its orientation ϕ. Basis for computing the Hough transform is the *Hough space graph*, which is a graph of r against ϕ. A point in an image is a sinus wave in the Hough space graph. By plotting every pixel of the image into the Hough space graph, we get a set of sinus waves. Wherever the sinus waves intersect in the Hough space graph, there is a line between the corresponding points in the image. Thus, finding the lines in the image is transformed into searching for a cumulation of pixels.

An example of Hough transform application to an aerial image is in Figure 3.14 [59].

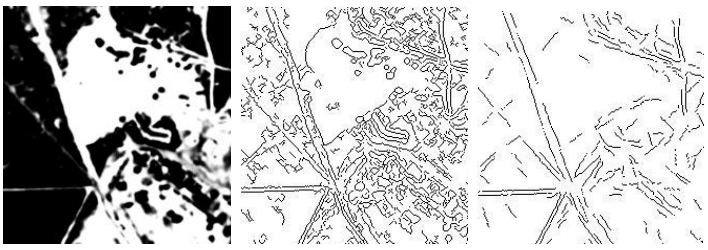


Figure 3.14: Input image, edge detection and Hough transform

The Hough transform is a powerful tool for line (or other shape) detection [63]. However, the computational requirements are very high and it is not suitable for hardware implementation.

3.4.3 Frequency Domain Transforms

In this section, we will briefly discuss transforms from a spatial domain into a frequency domain. The most popular ones are Fourier, cosine, and wavelet[3] transforms. Basic idea of these transforms is that we may obtain important features from an image in the frequency domain, which would be normally unaccessible in the spatial domain.

Considering that we can use transforms as a preprocessing technique, we can perform feature extraction in the frequency domain. For feature extraction, it's likely we will use different techniques than the ones developed for feature extraction in the spatial domain. Transforms are also powerful tools used for compression; for example, JPEG compression is based on a cosine transform and JPEG 2000 is based on a wavelet transform.

Unfortunately, all mentioned transforms are computationally expensive techniques. Implementation in hardware is possible, but very costly. This statement was confirmed by the implementation of a 2-D wavelet transform in FPGA [17], published in [13] and [15].

[3]The wavelet transform actually transforms an image into a space-frequency domain, not into a frequency domain

There are hardware-friendly methods like Haar filters [69] based on adding or subtracting, however, this particular method gives only limited results.

3.4.4 Thinning

Thinning [18] is a technique that skeletonizes an object. This can be very useful for object detection because it extracts the shape information out of an object. However, the method is multi-pass from its nature, and together with other features it is not suitable for hardware implementation.

This method is not meant to be a "stand-alone" method, but usually prepares the scene to be processed by another mentioned methods.

3.4.5 Histogram Techniques

A gray-level histogram gathers information about the brightness distribution of an image or an image slice, it is a dependency graph with the brightness on the x-axis and number of pixels with the corresponding brightness on the y-axis. An example of an histogram is in Figure 3.15. A histogram is particularly useful for finding an appropriate threshold for segmentation [64] [67]. Histogram creation is suitable for hardware implementation.

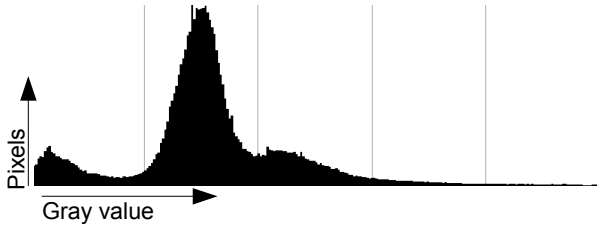


Figure 3.15: Example of a histogram

3.4.6 Motion Detection

Having two consecutive frames from a video stream, it's possible to extract moving objects from them [42] [90]. If the camera is not moving, this is a relatively easy operation, even in hardware [70]. However, any camera movement or changes in the weather make this task much more difficult [107].

3.4.7 Template Matching

The basic idea of template matching is to have a set of predefined image slices called templates and try to find similar image slices within an image. A classical template matching approach is to create a linear filter for each template, with the template as a mask.

Templates are usually of a smaller size, but not necessarily. For example, Zhao and Davis [39] suggested a template matching method for detecting people in an image.

Template matching is not limited to the spatial domain. Template matching can be performed on images passed through the wavelet transform [101], or frequency domain transforms. There are also many enhancements to standard template matching, for example using a distance transform [106] in order to overcome partially missing data.

However, linear filters require many multiplication operations, which are not suitable for hardware implementation. The method suggested by Villasenor et al. [102] overcomes this problem by implementing templates as binary images, which reduces multiplication operation into addition. Another solution is to use some cheaper non-linear function that shows how much the template and the image slice are alike.

3.4.8 Hardware Architectures

There are universal architectures that support the parallelization of image processing operations. Cavadini et al. [68] suggest VLSI architecture based on an SIMD processor array. Ratha and Jan [89] use an FPGA array to run image processing tasks. These solutions may be efficient, but they are more of a way how to implement processors than a hardware solution.

The area of hardware implementation that deals with less sophisticated image operations is relatively well explored. VLSI median implementation has been proposed by Breveglieri and Piuri [74] and FPGA median implementation by Maheshwari et al. [75]. Histogram equalization in FPGA has been suggested by Li et al. [56]. Etiennne-Cummings et al. [55] developed a single VLSI chip for histogramming, segmentation, and simple pattern matching.

Much work was done in the field of hardware acceleration of standard algorithms. For example, Zemcik et al. [114] proposed an AdaBoost based face detection system with FPGA acceleration. Scalera et al. [91] used an FPGA for detection of moving objects in an image sequence.

As discussed in Chapter 1, there are computer vision methods specifically designed for hardware implementation. Evolvable hardware [27] methods definitely fall into this category. Salami, Sakanashi, Iwata et al. [116] [117] [118] developed their compression methods using evolvable hardware. Sekanina [119] developed an evolvable image filter and Martinek et al. [120] implemented it in a real FPGA system. Iwata et al. [121] and Torresen et al. [122] implemented classification methods for pattern recognition systems in evolvable hardware.

Even though we have shown some of the promising hardware implementations of image processing methods, the area of hardware designed methods is not well explored yet.

3.5 Chapter Conclusions

In this chapter, important theoretical fields connected with this thesis were introduced and discussed. Basics of pattern recognition were introduced in Section 3.3. Using these basics, object detection techniques were presented in Section 3.4. Each of the methods was discussed with relation to suitability for hardware implementation. In Section 3.4.8, hardware architectures and methods were presented.

For proper definitions of the image operations throughout this thesis, some mathematical apparatus was needed. In Section 3.1, the theoretical background was defined, based on *image* and *neighborhood*, both of which are defined as a matrix, and *structuring element* defined as a set of ordered pairs. This notation is easy to use and sufficient to describe all results and computations throughout this thesis.

Using the mentioned mathematical definitions, basic types of filters are presented in Section 3.2. For this thesis, the most important are the morphological filters presented in Section 3.2.2, particularly the hit-or-miss filter described in Section 3.2.2.

Chapter 4

Goals

As was specified in the Introduction section, the goal of this thesis is to develop an object detection method suitable for programmable hardware implementation. Now, other features the method should have will be discussed.

In Section 2.2, today's rapidly developing area of embedded systems was described. Embedded systems can contain not only standard processors, but also programmable devices, such as those discussed in Section 2.1. The proposed method should be suitable for embedded system implementation, as object detection is often used in embedded systems.

The proposed method should actively use the reconfigurability of PLD chips, which delineates the method from general VLSI methods. Reconfiguration could also be used for adaption to environment changes.

Now the vital question comes: What should be the *new* feature of the proposed method, so that the proposed method is not only a parallelized version of a software method? The answer lies in the nature of hardware implementation, which allows massive parallel processing. However, for massive parallel processing the basic computation unit should be as simple as possible in order to place a large number of them into the target device. So the final goal is to implement a method that will consist of very simple basic elements, and the strength of the method will be in the implementation of a large number of those elements. Then, the basic hypothesis of this thesis is:

The proposed method, though based on very simple elements, can compete with commercially used methods due to the massive parallelization of those simple elements.

Summarized, the goal is to implement an object detection method with these features:

1. *Method will be designed for hardware* as stated in Section 1.3.
2. *Simple basic building blocks* will be used in order to use massive parallelization.
3. *Real-time processing* is required so that a solution for every image is found in a constant maximum time limit. If possible, computation should be done "on-the-fly", i.e. the data will be processed while being received at the input, and output will be available after certain constant time delay. The "on-the-fly" feature is not necessary for real-time hardware implementation, but it brings some advantages like minimum delay or no need of buffer implementation.
4. There are *limited resources* in an embedded system, we should not use a high-power PC processor or large memory blocks.
5. *Adaptability to changes* in the environment using PLD reconfiguration should be possible.

6. *Comparable results* with commercially used methods are required.

An object detection method will be proposed in Section 5.1, according to the features listed above.

Chapter 5

Template Based Detection Method

In Chapter 4, the expected functionality of the proposed method is suggested. Coming from these expectations, a new method is outlined in Section 5.1, perfected in Section 5.2 and the final method is proposal in Section 5.3. Matters related to the hardware realization of this method are discussed in Chapter 6.

The proposed method together with experimental architecture was described in [1] and [2].

5.1 Method Proposition

To decide what technique will be used for the proposed method, we will go through Section 3.4. The method should be suitable for FPGA implementation and should profit from the advantages of FPGAs. This is not the case for any frequency domain transforms, Hough transform, thinning, or motion detection techniques. Edge detection is suitable for an FPGA implementation, but is a rather simple method and can be efficiently implemented in a DSP processor. Histogram techniques are suitable for an FPGA implementation and profit from hardware implementation. However, histogramming usually does not give sufficient results for the method as a whole, but can be used as a part of the method for segmentation (Section 5.2.2). The method suggested by Viola and Jones, mentioned in Section 3.3.3, uses only a limited number of features, therefore it can not be considered a method designed for hardware. Template matching, is suitable for hardware implementation in non-linear filter form. Template matching can also benefit from hardware implementation by utilizing massive parallelization in the case of larger number of filters.

Probably easiest way, in which to implement the template matching technique, is shown in the example in Figure 5.1. Our goal is to find the united express logo and the windows in the picture. This is done through a unit called *template searching*, which is trying to find a place in the picture that is the same (or similar) to a template. Output from this unit are the coordinates of the located object.

In this form, the method may suffer from some important problems.

- The object will not look the same every time. I.e. we do not want to search for one object, but for certain class of objects. For example, we want to find an airplane (not only united express) or we may be searching for a sign (not identical with the one on the airplane).

- The object may consist of some repeating patterns. In the example, these are windows and certain letters. Having only one template for each of these repeating

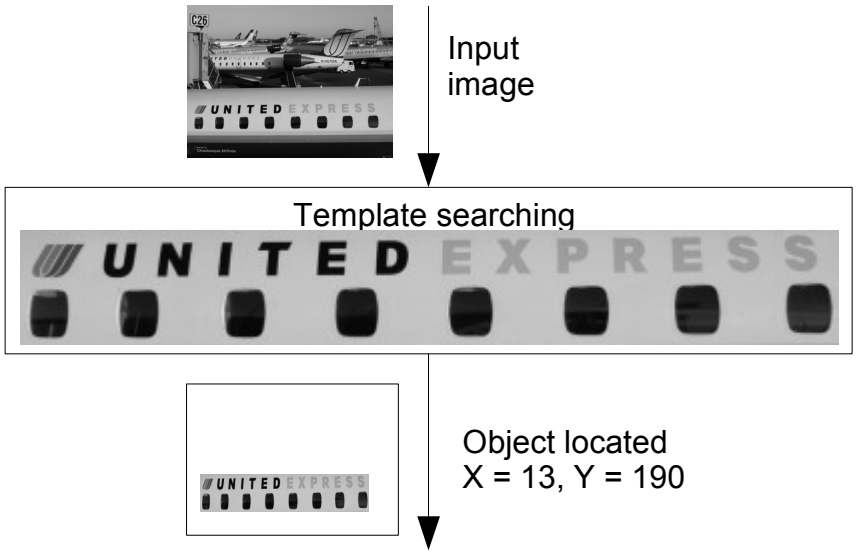


Figure 5.1: Example of template matching method

objects may significantly save implementation resources.

- There may be parts of an object not holding any information, e.g. white space between letters or windows. However, appropriate detection hardware must be still implemented.

These issues may be solved by more sophisticated detection methods. However, the requirement from Chapter 4 is to suggest very simple detecting elements.

The basic idea on how to resolve this is to *decompose the searched object into smaller objects* (templates), *and search for each of these templates separately*, used by Qiang and Bo [108], for example. Presence of an object is then determined by dependencies of these templates. This way we can allow higher miss rate of the simple elements, while the object as a whole should still be detected. By this decomposition, we also will switch from searching for *instances of an object* to searching for *objects belonging to a certain class*. This feature is caused by the possibility of locating objects that do not exactly match a template.

This is depicted in the example in Figure 5.2. The searched object is again an airplane with the United Express logo. Templates in our case are letters, the logo and the shape of a window[1]. You may notice that we searched only once for repeating letters in the word "express", since the same letters are considered to be the same pattern.

All templates are located in the image and results are put into a new image, called a *template occurrence image*. Representation of those templates in a template occurrence image may vary; we use a simple black rectangle for every template found.

The final step is searching for an object. In this case, it is the simple detection of a burst of black rectangles.

Now, we are ready to generally define the method in Definition 5.1.1.

[1]An object does not have to be decomposed to semantic elements, like letters. Decomposition can be done over smaller elements, for example parts of the letters.

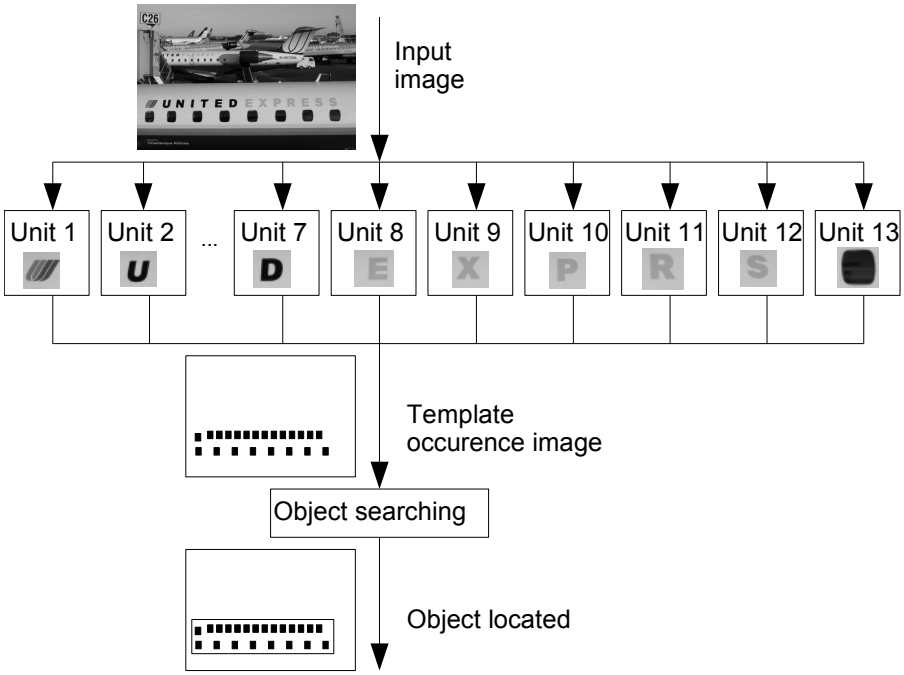


Figure 5.2: Proposition of the method

Definition 5.1.1 *Method*

1. *Define parameters of templates:*
   - *Number of templates*
   - *Size of each template*
   - *A bitmap image for each template*
2. *Search for each of these templates in an input image in hardware. If a template is present, mark this in the template occurrence image (temporary image of same size as input image).*
3. *According to the template occurrence image, decide whether the searched object is present and where.*

Unlike in the example, templates in a real system are expected to be smaller, automatically generated and their number will be in hundreds.

How does Definition 5.1.1 comply with goals of Chapter 4?

1. Feature extraction can be efficiently implemented in hardware; each template can be implemented in parallel. Questions of hardware suitability from Section 1.3 will be a topic throughout the rest of this thesis, mainly in Sections 5.3, 7 and 8.6.
2. Achieving a simple template matching element will be the goal of the following sections 5.2 and 5.3.
3. A major part of the algorithm, filtering, will be computed *"on-the-fly"*. The classification part of the algorithm (actual object detection based on feature occurrences) can be also implemented in hardware.

4. The processor part should not be too computationally difficult, thus an embedded processor will be suitable.

5. The method proposed in Chapter 7 allows adaptation to the environment or slow changing object.

6. The results in Section 8.5 show that the method's efficiency is comparable to commercial methods.

5.2 Development Phases

As was said in Section 5.1, we aim to implement a large number of search units, each searching for one fragment of the object.

For the method, the standard sequence of steps of pattern recognition introduced in Section 3.3 should be used - preprocessing, feature extraction, and classification. The main focus should be laid on quality feature extraction, as it makes classification easy and less source-consuming.

- *Preprocessing* will be a preliminary step to feature extraction. It should make feature extraction easier and more effective. The input to a preprocessing unit is an input image $G \in \mathcal{G}^d_{w,h}$, and the output is an image of the same size $G^p \in \mathcal{G}^{d_p}_{w,h}$.

- *Feature extraction* will be the core of the method. A filter bank implemented in hardware will search for features indicating an occurrence of an object. The input to the feature extraction unit is the output image from the preprocessing unit G^p. The output is an image of the same size containing information about matching templates $G^f \in \mathcal{G}^{d_f}_{w,h}$. This output image is called a template occurrence image.

- *Classification* should be easy, as it will profit from high-quality feature extraction. In the simplest case, searching for the largest bunch of extracted features should be enough. The input is template occurrence image G^f, and the output is a set of coordinates of the image G pointing to the located objects $c \in 2^{(x,y)}$ (if no object is found, empty set is returned).

There were a few development phases before the final solution was reached. They are presented in this section. This should help to understand why certain decision were made. The final phase is in a separate Section 5.3. For all phases, a section named *class of objects* is included, which discusses what kind of objects the method is capable of dealing with.

5.2.1 Version 1

In the first version, there is no preprocessing phase. The feature extraction method is suggested here and basic ideas about classification are presented.

Feature Extraction

The first consideration is what technique we will use for feature extraction. It should work as in the example in Figure 5.2 - input will be an image, output will be an image with parts matching the templates displayed. For this requirement, the closest technique is a set of *spatial filters*, each searching for one template.

The next question concerns on what type of filters should be used. Requirements for the filter can be summarized in the following points, as they have been described in Section 3.3.2 and in Chapter 4.

- Implementation in hardware must be relatively cheap.
- A filter must show good resistance against changes in illumination.
- If possible, a filter should be rotationally and scale invariant. This requirement can be realized by implementing a number of filters for each template, each differently angled or zoomed.

There are many ways in which filters with these features can be implemented. Linear filters with convolution may be suitable, but are very space demanding in hardware because of necessary multiplication, so non-linear filters are preferable.

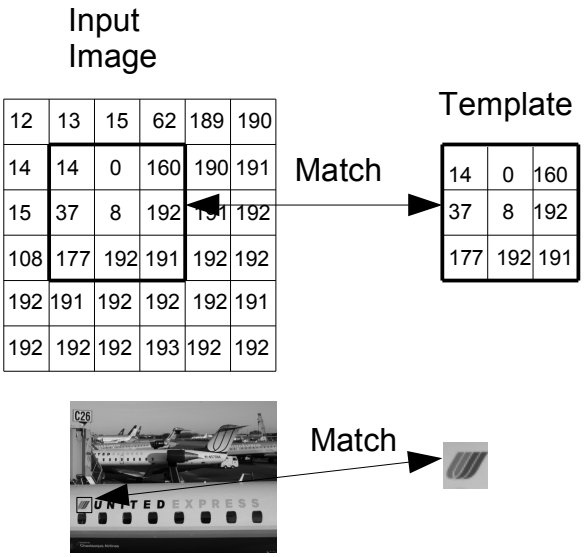


Figure 5.3: Example of version no. 1

Basically, we need to compare similarities between two images (template and a slice of an image). This leads us towards using a comparison operation in the filter. Two operations Suitable for comparing are the *mean of absolute differences* and the *mean square error*, both used, for example, by Chang and Hwang [38]. However, these methods are still quite expensive, mainly the mean square error. The mean of absolute differences is relatively suitable for hardware, but we could not do the important step in Section 5.2.2, making the implementation significantly cheaper.

As the most natural cheap method, the one proposed by Gause et al. [30] may come to mind, which will be our version 1 method. The basic concept of this filter is the simple comparison of a template and an image slice. For our purposes, we define the template for version 1 as the structuring element. For simplicity, now we define only one template and one filter. Every pixel of the template is compared to a value of a relevant image pixel, and if all pixels match, the result of the filter is 1. This filter is very similar to the hit-or-miss filter from Section 3.2; the only difference is that the gray depth is considered to be arbitrary. An example of this is in Figure 5.3.

Definition 5.2.1 *Version 1 template of structuring element S and image G is a function*

$$T : S \to \{0, 1..d-1\}$$

where d is gray depth of image G.

Definition 5.2.2 *Version 1 filter is a filter over $\mathcal{S}$*

$$F(N) = \begin{cases} 1 & n_{x+i,y+j} = T_{i,j} \ \forall (i,j) \in S \\ 0 & otherwise \end{cases}$$

where $N = \mathcal{N}_{x,y}^{G,S}$

Classification

In this thesis, there will be no straight answer for how to implement the classification part, because it largely depends on the application. For our purposes, we will use two extreme versions of classification:

1. *Minimal classification* where object is detected when the number of matched templates exceeds a certain number in an area of the objects' size
2. *Maximal classification* where object is detected when there are slices matching templates at the exact position as on the searched object

In reality, we will probably use something between those two. For instance, specific slices have to be placed in certain areas.

Class of objects

This version finds only image slices exactly matching a template. Thus, for maximal classification, only the exact object that we are searching for is detected, with an allowance for differences between the slices. In Figure 5.4, you see an example of such a case.

Figure 5.4: Example of version 1, maximal classification

For minimal classification, slices must also match templates exactly, but they can be differently organized than in the original object.

Things to Improve

Unfortunately, in reality, objects in the image will never be exactly the same as those in the template. Reasons for this are changing illumination, noise and other issues, like different distance or position of the object from the camera. Overall, there are several big disadvantages coming with the first version:

- Not resistant against changes in illumination.
- Not noise-resistant.
- Still quite expensive for parallel implementation in hardware, as for every template, $w \times h$ (width, height) n-bit comparators are required

5.2.2 Version 2

In this version, we will make a fairly big step. Instead of finding image slices *exactly matching* templates, we will search for *shapes* typical for a certain class of objects. It changes the class of the searched objects, and also helps to achieve one of the goals - maximizing hit rate even at the expense of more false alarms.

Feature Extraction

The form of the template has changed significantly. It's not a small gray-scale image any more, it is a matrix where each element is a binary value saying whether a corresponding pixel should be bright or dark to match.

Naturally the question comes - what value of the pixel means bright and what value means dark? Generally, we have to decide for a value called *divider* (Definition 5.2.4), which is a limit between dark and bright pixels.

Definition 5.2.3 *Version 2 template of structuring element S is a function*

$$T : S \rightarrow \{0, 1\}$$

Definition 5.2.4 *Version 2 filter is a filter over $\mathcal{S}$ so that*

$$F(N) = \begin{cases} 1 \ ((n_{x+i,y+j} <= D \ \forall T_{i,j} = 0) \ \wedge \\ \quad (n_{x+i,y+j} > D \ \forall T_{i,j} = 1)) \ \forall (i,j) \in S \\ 0 \ otherwise \end{cases}$$

where $N = \mathcal{N}_{x,y}^{G,S}$ and $D \in \{0, 1, ..d\}$ divider between dark and bright pixels

Now, we will discuss how to find the dividing value. This problem is identical to the *thresholding* problem. There are more ways, some more or less effective and costly than others.

Global Thresholding

The first group of dividing functions contains functions with the same value for the whole image.

1. *Constant thresholding value* is easiest way to implement, unfortunately obviously very ineffective.

2. *Information from the whole picture* is used to obtain the thresholding value. Methods widely used for this purpose are using histograms. Unfortunately, a threshold computed from a whole image histogram may not give desirable values - if the object we search for is in the dark or bright part of the image.

Local Thresholding

Unfortunately, none of the global thresholding functions are suitable. One solution is to compute the dividing point for each pixel separately, where the divider value will be a function of the pixel neighborhood.

1. *Average between maximum and minimum* value in the neighborhood is relatively easy and cheap to implement and may be effective (evaluation will be done in Section 5.3.1).

2. *Average of all of the pixels* in the neighborhood is comparable to the previous method, considering computational complexity.

3. *Median* value of the neighborhood pixels, compared to the previous two methods, is much more difficult and expensive to implement, mainly in hardware [73] [74] [75]. One solution how to make implementation cheaper might be, for example, to implement in hardware only an estimation of the median like the one suggested by Suorania et al. [76].

4. *Value based on the neighborhood histogram* is probably the most expensive, and conventional methods are likely to fail due to the small neighborhood.

The function which will be actually used for the final method version will be discussed in Section 5.3.1.

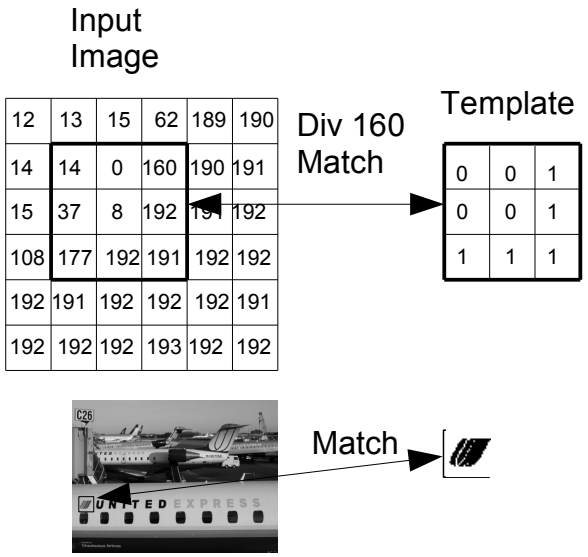


Figure 5.5: Example of version no. 2

Class of objects

A vital question comes with this version. Will changing the feature extraction method badly effect the class of objects we are trying to find?

Theorem 5.2.1 *For parameters properly adjusted, the set of objects found by version 1 is a subset of set of objects found by version 2.*

Proof 5.2.1 *We will let the version 1 template be T_1. We will let the version 2 template be T_2 of structuring element S from T_1 of structuring element S using following rules:*

1. *M is arbitrary function, V_m is a value derived from values in T_1: $V_m = M\{T_1(i,j) : (i,j) \in S\}$.*

2. *Value in T_2 will be zero if corresponding value in T_1 is less or equal to V_m, or one for corresponding value more than V_m: $T_2(i,j) = 0 \Leftrightarrow T_1(i,j) \leq V_m, T_2(i,j) = 1 \Leftrightarrow T_1(i,j) > V_m$*

3. *Set divider $D = V_m$*

In the following steps, we will review facts about detection. F^1 is a version 1 feature extraction filter, F^2 is a version 2 feature extraction filter.

1. *The only objects detected by T_1 are those with pixel values equal to T_1 values: $F^1(N) = 1 \Leftrightarrow n_{x+i,y+j} = T_1(i,j) : (i,j) \in S$*

2. *To detect an object by T_2*

   - *All pixel values with corresponding dark template values must be less or equal to V_m*
   - *All pixel values with corresponding bright template values must be more than V_m*

   $F^2(N) = 1 \Leftrightarrow ((n_{x+i,y+j} \leq D \ \forall T_2(i,j) = 0) \vee (n_{x+i,y+j} > D \ \forall T_2(i,j) = 1)) : (i,j) \in S$

Any object detected by T_1 should be detected by T_2. A sufficient condition is that every value detected by T_1 is also detected by T_2.

$$(T_1(i,j) \leq D \ \forall T_2(i,j) = 0) \wedge (T_1(i,j) > D \ \forall T_2(i,j) = 1$$

As a result from how template T_2 was created, this equation will be always true.

The version 2 algorithm is capable of finding all objects in a class found by version 1. The problem may occur the other way around - if version 2 is going to produce too many false alarms. The answer is not trivial and may significantly affect the method's quality. However, the step made in version 2 leads to very cheap hardware implementation, which is one of the primary goals.

5.3 Final Proposal

In the final version, we will try to make the method compliant with the simple processing element requirement.

The core of the method, feature extraction, will be crucially considering the utilization of an FPGA chip, because we plan to implement many filters in parallel. It's likely that the number of these filters will be limited by the size of the chip. Thus, our goal is to make filter implementation as cheap as possible.

To achieve this goal, we will analyse steps used in pattern recognition, namely preprocessing and feature extraction. The basis of this concept can be seen in Figure 5.6 - there is only one preprocessing unit, but there are n feature extraction units (filters). Thus, the size of one feature extraction filter will impact the overall chip occupation significantly.

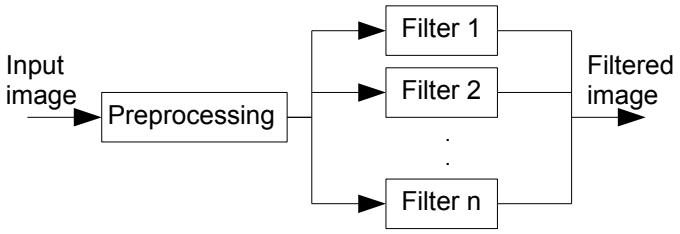


Figure 5.6: Hardware preprocessing and feature extraction data flow

This indicates that we should not be afraid of the space-consuming implementation of the preprocessing unit for the sake of getting as simple filters as possible.

Looking at the version 2 algorithm, values from the image are compared to the divider value in every filter. However, divider is the same for all filters, because they are based solely on values from the image, not from a template. This indicates the possibility of moving part of this comparison to the preprocessing part, and keeping only necessary small part in the feature extraction.

We want to do as much work in preprocessing as possible, that means leaving only the necessary part for feature extraction. This necessary part is comparing 1-bit values in templates with 1-bit values derived from an image, saying whether the pixel is dark or bright. This implementation will be very cheap in hardware - if templates are known at time of synthesis, it would require only invertors for dark pixels and one AND gate through all values (chip occupancy will be discussed more in Section 6.10).

5.3.1 Preprocessing

Generally, the preprocessing unit has to convert an image to a binary image. How this conversion is done is largely application specific. In the simplest case, we could adopt the method of computing the divider as in Section 5.2.2. In this thesis, preprocessing will be made suitable for the case studies - the license plate detection and the road sign detection.

During experiments, preprocessing gave much better results if the edge detection was performed first, as can be seen in Table 5.1. The meaning of the values in the table will be described later in this section. Therefore, the whole preprocessing operation is implemented as a set of two filters - the edge detection filter and thresholding filter.

For the edge detection filter (1^{st} stage), the output of the filter (pixel in the output image) will be the response of the edge filter. Many methods could be used for edge detection. The most common approach would be the linear filters from Section 3.4.1. But, as mentioned, these filters are not suitable for hardware implementation. As an alternative, a 3×3 non-linear filter was developed. The function of this filter is the difference between the minimum and maximum of the pixels in a neighborhood (Definition 5.3.1).

The second stage is actual segmentation. A non-linear 3×3 filter computes a value from the neighborhood, and outputs 1 if the pixel value is greater than this value, or 0 otherwise. There are more options for what the function for the dividing value could be, this function is identical to the divider functions discussed in Section 5.2.2. Experiments for all the discussed functions were made. Functions based on the histogram did not give satisfactory results due to the small neighborhood, and global thresholding functions often discarded important local features. Hence a decision among the local thresholding techniques had

to be made - average between minimum and maximum, median, or neighborhood average. All three techniques were applied to the experiment in a subsidiary case study in Chapter 9. For each of these experiments, the *image class quality* was computed (defined in Section 5.4.5) which refers to the object detection quality. Image class quality is a real number in the range $(0, \infty)$ and higher number refers to better quality.

Technique	Quality with edge det.	Quality w/out edge det.
Min-max average	5.3	1.3
Average	2.8	1.2
Median	2.3	1.2

Table 5.1: Image Class Quality for Preprocessing Techniques

Results are in Table 5.1. Clearly, the average between the minimum and maximum gives the best results for the test case. Even though this result does not necessarily imply that the min/max technique is generally the best, we will use it for our experiment since the preprocessing part is only marginal in the context of this thesis.

Definition 5.3.1 *The preprocessing filter phase 1* $P_1 \in \mathcal{F}_S$ *and preprocessing filter phase 2* $P_2 \in \mathcal{F}_S$ *are defined as follows.*

Let $N = \mathcal{N}_{x,y}^{G,S}\ \forall G \in \mathcal{G}_{w,h}\ \forall w, h \in \mathbb{N}\ \forall x \in \{0..w-1\} \forall y \in \{0..h-1\}$.

Let us define average of minimum and maximum as

$$MM(N) = \frac{max\{n_{ij} \in N\} + min\{n_{ij} \in N\}}{2}$$

Then, we can define

$$P_1(N) = MM(N)$$

$$P_2(N) = \begin{cases} 1\ G_{x,y} > MM(N) \\ 0\ otherwise \end{cases}$$

This method proved to be very effective in preserving shapes and removing noise, which are basic considerations for the quality of feature extraction. An Example of a preprocessed image is in Fig. 5.7.

5.3.2 Feature Extraction

With the feature extraction filter, we are going back to version 1. The difference is, now we will work with binary image and templates, which is much better for hardware implementation. In this version, we also start working with a filter bank instead of one filter. The filter bank was expected to be used in previous version too, but was omitted for simplicity.

To define a template bank, we have to first define a template as a function assigning each structuring element either 0 or 1 (Definition 5.3.2). This value determines whether the corresponding pixel should be dark or bright. The definition of a template bank follows in Definition 5.3.4 as a function assigning each possible template either a number or $\perp$. The feature extraction filter in Definition 5.3.6 then puts identification numbers to template occurrence image to positions where corresponding templates fit.

Definition 5.3.2 *Template for structuring element S is function*

$$T : S \rightarrow \{0, 1\}$$

Definition 5.3.3 *Set of all templates for structuring elements S is denoted*

$$\mathcal{T}_S$$

Definition 5.3.4 *Template bank B for structuring element S is function*

$$B : \mathcal{T}_S \rightarrow N_\perp$$

Definition 5.3.5 *Set of all template banks for structuring elements S is denoted*

$$\mathcal{B}_S$$

Definition 5.3.6 *Let $N = \mathcal{N}_{x,y}^{G,S}$ be a neighborhood for some $G \in \mathcal{G}_{w,h}^2$ $w, h \in \mathbb{N}$ $x \in \{0..w-1\} y \in \{0..h-1\}$ and $T \in \mathcal{T}_S$ a template. Operator $\approx$ (match) is defined as follows:*

$$N \approx T \Leftrightarrow \forall i, j \in S : N_{x+i,y+j} = T(i, j)$$

Feature extraction filter $F \in \mathcal{F}_S$ for template bank $B \in \mathcal{B}_S$ is every filter that satisfies the following condition:

$$F(N) \neq \perp \Rightarrow \exists T \in \mathcal{T}_S \text{ s.t. } B(T) = F(N) \wedge T \approx N$$

An illustration of preprocessing and feature extraction is shown in Figure 5.7. On the right and left side real-life image and an artificially made close-up image are shown, respectively. For preprocessing, the min-max average method without an edge detection filter has been used. In the feature extraction phase, we are searching for templates $T_1..T_n$ included in the template bank. Finally, the pattern occurrence image is created with the located slices matching templates. In the right image, three occurrences of template "E" (template T_3) have been found.

5.3.3 Classification

In Section 5.2.1, we outlined two possible classification implementations. As stated before, the proper classification method depends on the application for which it is being used.

First, basic terms are defined. Be aware that term *neighborhood* defined earlier will be used here for a different purpose than spatial filters.

Definition 5.3.7 *Size $|T|$ of the template $T \in \mathcal{T}_S$ is*

$$|T| = |S|$$

Size $|N|$ of the neighborhood $N \in \mathcal{N}$ is

$$|N| = |\{n_{ij} | n_{ij} \in N, n_{ij} \neq \perp\}|$$

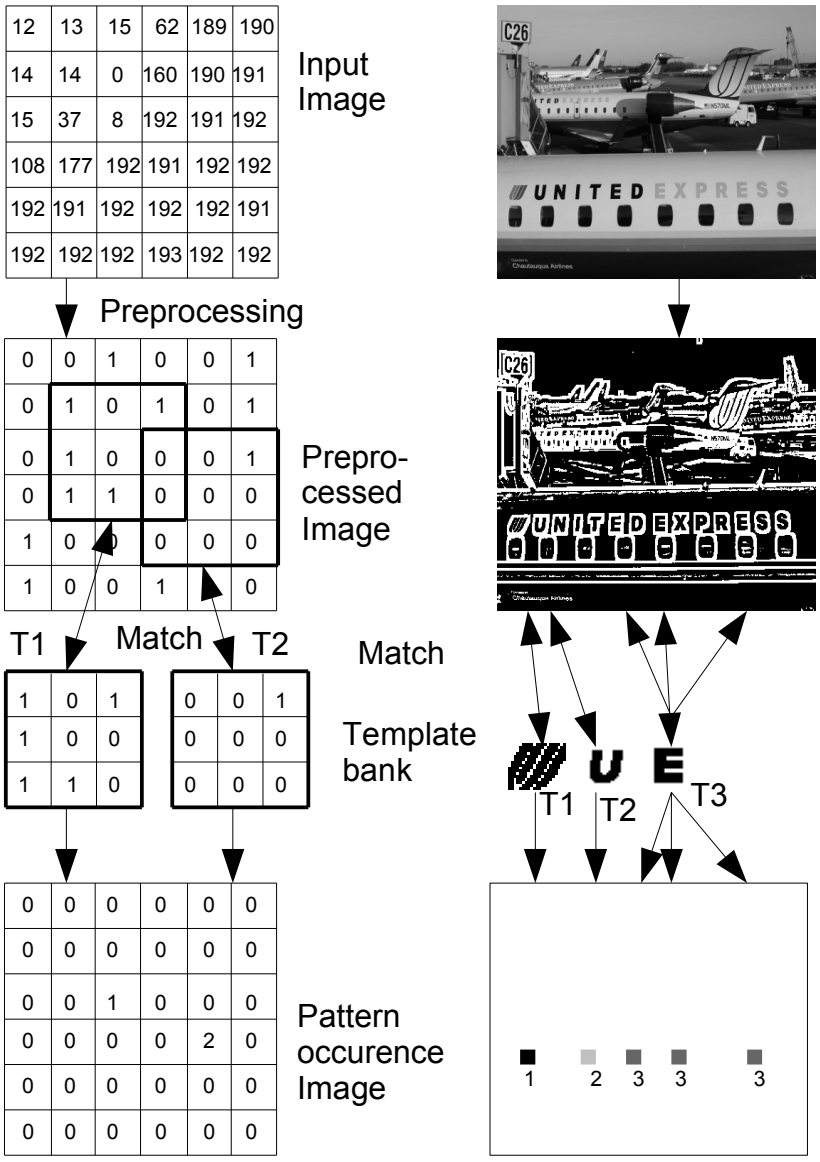


Figure 5.7: Example of preprocessing and feature extraction

Definition 5.3.8 *Let G' be a filtered image by the feature extraction fitler $F \in \mathcal{F}_S$ for the template bank $B \in \mathcal{B}_S$ of source image G. Let $N' \in \mathcal{N}_{x,y}^{G',S}$ be a neighborhood. The rating R of neighborhood $N \in \mathcal{N}_{x,y}^{G,S}$ for bank B is a number*

$$R_{B,N} = \frac{\sum\limits_{(i,j)\in S} a_{i,j}}{|S|}, \text{ where } a_{i,j} = \begin{cases} 1 & N'_{x+i,y+j} > 0 \\ 0 & otherwise \end{cases}$$

The rating is a very important feature of a neighborhood. It shows how high the proposed method rates a particular neighborhood. Here it is defined as a number of template detections counted over one pixel (minimal classification method). Generally, any other classification method could be used. However, for future computations, only rating as defined in Definition 5.3.8 will be used.

The *best match* (Definition 5.3.9) is a neighborhood of shape S_{BM} that has the highest rating of all neighborhoods with the same shape inside the neighborhood N.

Definition 5.3.9 *Best match* $BM \in \mathcal{N}^{N,S_{BM}}$ *in neighborhood* $N \in \mathcal{N}^{G,S}_{x,y}$ *for bank* B *is a neighborhood such that*

$$R_{B,BM} \geq R_{B,C} \; \forall C \in \mathcal{N}^{N,S_{BM}}$$

An output of the whole proposed method (Definition 5.3.10) is the best match in the whole image, or $\perp$ if the algorithm assumes that there is no object in an image.

Definition 5.3.10 *Let* N *be the neighborhood of image* $G \in \mathcal{G}_{w,h}$ *such that*

$$N_{x,y} \neq \perp \;\; \forall x \in \{0..w-1\} \; \forall y \in \{0..h-1\}$$

Limit $L \in \mathbb{N}$ *is a number defined by the user.*

Let BM *be the best match in* N *for bank* B*. Then classification output of image* G *with bank* B *and limit* L *is*

$$U^L_{G,B} = \begin{cases} BM & R_{BM} > L \\ \perp & otherwise \end{cases}$$

To check whether the best match detection is correct or not, we have to first define an *object*[2].

Definition 5.3.11 *The object* O_G *of image* G *is either neighborhood that the user defined as an object or* $\perp$ *if there is no object in* G*.*

The surrounding of neighborhood $O_G \in \mathcal{N}^G$ *is the neighborhood* $N \in \mathcal{N}^G$ *where*

$$O_{x,y} = \perp \Leftrightarrow N_{x,y} \neq \perp$$

Now, we can determine whether we found the right object or not.

Definition 5.3.12 *Let* O_G *be the object and* $U^L_{G,B}$ *be the classification output. With image* G*, limit* L*, and bank* B*, classification output hits object* O *when*

$$U = O$$

The definition of *Hit* can also differ slightly in real-life implementation. Mainly, absolute accuracy may not be required, which would allow the classification output to be shifted against the object.

Rating by AdaBoost

AdaBoost (Section 3.3.3) could be particularly useful for classification, giving each template a different importance. Classification with AdaBoost is defined as follows:

[2]Here, we defined only one object in an image. If needed the definition can easily be changed to describe more objects in an image.

Definition 5.3.13 *Let G' be the filtered image by the feature extraction fitler $F \in \mathcal{F}_S$ for the template bank $B \in \mathcal{B}_S$ of the source image G. Let $N' \in \mathcal{N}_{x,y}^{G',S}$ be a neighborhood. The rating by AdaBoost R of neighborhood $N \in \mathcal{N}_{x,y}^{G,S}$ for bank B is a number*

$$R_{B,N} = \frac{\sum_{(i,j)\in S} \alpha_p \times a_{i,j}}{|S|}, \text{ where } p = N'_{x+i,y+j}, \quad a_{i,j} = \begin{cases} 1 & N'_{x+i,y+j} > 0 \\ 0 & otherwise \end{cases}$$

where α_p is a coefficient assigned to every template.

Coefficients α_p have to be obtained by the AdaBoost learning algorithm on the training set of images with marked objects. An example of such a learning algorithm follows:

1. Using conventional methods, obtain a large number of templates based on the objects in the training images (for example, pick template-size slices from objects and threshold them).
2. Select the best template with respect to the weights given to the training images (for example, sum of the weights of the training images where the object was hit).
3. Set α for the selected template (for example, based on the number of object hits).
4. Increase the weights of the training images that were missed by the template.
5. Repeat from step 2 on until the number of templates is sufficient.

5.3.4 Class of objects

The set of objects detected in a class is the same as in version 2 (Section 5.2.2), the advantage of this version is the possibility of easier hardware realization.

5.4 Setting of Parameters

In this section, important features of the proposed method will be analyzed, which allows us to properly set parameters. The outputs of this section will be used in the case study for setting the parameters of the real-life experiment in Sections 8.2 and 8.3. First, we define basic concepts.

5.4.1 Noise

For the real-life environment, we have to consider that two images of one object will never be the same. Some factors are sensor quality and environment changes. For our purposes, we deal with these factors altogether by defining a *noisy pixel* and an *ideal image*.

Definition 5.4.1 *A noisy pixel (x, y) is a pixel whose value is different than the value expected by the user. The ideal value $i_{x,y}$ of a noisy pixel (x, y) is the value that the user expects to observe.*

Definition 5.4.2 *The ideal image $I \in \mathcal{G}_{w,h}$ of image $G \in \mathcal{G}_{w,h}$ is an image that*

$$I_{x,y} = i_{x,y} \ \forall x \in \{0..w-1\} \forall y \in \{0..h-1\}$$

where $i_{x,y}$ is the ideal value of pixel $G_{x,y}$

The *noise coefficient* (Definition 5.4.3) is a real number showing how noisy a neighborhood is. Limitation to a maximum noise of 0.5 comes from the probability nature of C, i.e. 0 means no noise in the image and 0.5 means completely random noise. Values above 0.5 would form inverted original image, which is not desirable for our meaning of C.

Definition 5.4.3 *The noise coefficient C_N of neighborhood $N \in \mathcal{N}^{G,S}$ is*

$$C_N = \begin{cases} C_R \; C_R \leq 0.5 \\ 0.5 \; otherwise \end{cases}$$

$$where \; C_R = \frac{\sum abs(N_{x,y} - I_{x,y})}{|S|} \; \forall N_{x,y} \neq \perp$$

where $I \in \mathcal{G}$ is the ideal image of image G

5.4.2 Image Quality

In this section, we will deal with *image quality*, which will help us in the following sections to properly set the method parameters. Definition 5.4.4 defines the quality as a fraction of the ratings of two arbitrary neighborhoods.

Definition 5.4.4 *The quality Q of neighborhood $N \in \mathcal{N}^{G,S}$ against neighborhood $N_{ref} \in \mathcal{N}^{G,S}$ for bank $B \in \mathcal{B}_S$ of image $G \in \mathcal{G}$ is defined as*

$$Q = \frac{R_N}{R_{N_{ref}}}$$

Image quality in Definition 5.4.5 determines how well an image is suitable for the proposed method.

Definition 5.4.5 *Image quality Q_G is the quality of object O of image G for bank B against the best match in the surrounding of the object O, where bank B is a bank that gives the highest Q_G.*

Figure 5.8 illustrates the meaning of Definition 5.4.5. In this figure is a typical example of the rating distribution in an image. The x-axis is the rating and the y-axis is the number of detections with corresponding ratings. The distance between an object rating and the best non-object rating determines the quality of the image.

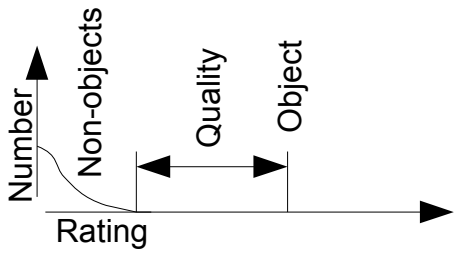


Figure 5.8: Typical distribution of ratings in an image

In case the object rating on the x-axis falls into the non-object zone, the quality will be less than 1, which means the object can not be detected by the proposed method.

5.4.3 Sensitivity of Rating to Noise

In this section, we will try to explore more about the changes of a rating according to the noise in an image.

By adding noise to an ideal image, two things will happen simultaneously:

1. New templates can be detected in noisy places. Because the bank has been designed for the object, we assume the detection of new templates is caused purely by noise, i.e. newly detected templates are not influenced by previously detected templates.

2. Detected templates will not be detected any more, if they are affected by noise.

For the first case, the probability that the noise will create a slice in one specific place that one specific template will detect is $C^{|T|}$. Probability that one of the $|B|$ templates will be detected is $P = |B| \times C^{|T|}$. For a neighborhood of size $|N|$, the average number of template detections will be $P \times |N|$. The Average number of template detections counted over one pixel is then P, which is also most probable rating of the neighborhood N.

For the second case, the probability that an area of size $|T|$ is *not* hit by a noisy pixel is $(1 - C)^{|T|}$. For k detected templates in neighborhood N, in average $k \times (1 - C)^{|T|}$ templates will *not* be affected by noise. Counting the not affected templates over one pixel we get new rating, which is $k/|N| \times (1 - C)^{|T|} = R_I \times (1 - C)^{|T|}$.

We can not determine an exact rating based on the noise coefficient. We can determine only the probability that a particular rating will occur. That's why we deal with only the rating that occurs with the highest probability.

From the previous statements, the following conclusion comes: The most probable rating R of neighborhood $N \in \mathcal{N}_{x,y}^{G,S}$, with noise coefficient C, and bank B designed for neighborhood N, is a number

$$R = |B| \times C^{|T|} + R_I \times (1 - C)^{|T|}$$

where I is the ideal image of G, $N_I \in \mathcal{N}_{x,y}^{I,S}$ is the neighborhood, R_I is the rating of N_I and $T \in B$ is a template.

5.4.4 Size of Templates

In this section, we will try to conclude with an equation for the optimal template size for an image class. Following computations will be done on one image that is a typical representative of the class.

For the computations, an important assumption is made, that there is a neighborhood with a noise coefficient[3] $C = 0.5$, of at least the size of an object in an image. This may not be true for all images in a class, but it's a sufficient condition if there are such neighborhoods in a significant part of an image class. An example of such a neighborhood would be an irregular pattern such as sand, snow, a forest in the distance, or objects illuminated by the sun from the specific angle. Obsertvations of the image class used in Chapter 8 show that around 90% of images satisfy the assumption, mainly because of the road illumination or the stripes.

Let there be an image $G \in \mathcal{G}$ with object $O \in \mathcal{N}^{G,S}$ with a noise coefficient of C_O and bank $B \in \mathcal{B}$. T is a template of bank B. I is the ideal image of G. Let there be a neighborhood $N \in \mathcal{N}^{G,S}$ in the surrounding of an object O with the noise coefficient $C_N = 1/2$. Then

[3] In this context, noise can be any part of an image where pixels are considered to be random

$$R_O = |B| \times C_O^{|T|} + R_I \times (1 - C_O)^{|T|}$$

where $N_I \in \mathcal{N}_{x,y}^{I,S}$ is the neighborhood, R_I is the rating of N_I

Assuming C_O being close to 0, $|B| \times C_O^{|T|}$ will be very low and compared to the rest of the equation insignificant, thus we get

$$R_O = R_I \times (1 - C_O)^{|T|}$$

The rating for neighborhood N is

$$R_N = |B| \times C_N^{|T|} + R_I \times (1 - C_N)^{|T|}$$

$R_I \times (1 - C_O)^{|T|}$ becomes insignificant compared to the rest of the equation, because N is in the surrounding and R_I is supposed to be very low here. From above, $C_N = 1/2$ resulting in

$$R_N = \frac{|B|}{2^{|T|}}$$

The best match of the surrounding can be either the rating of the noisy place, or the best match of the ideal image surrounding. Then the quality of the image is

$$Q = \begin{cases} \frac{R_I(1-C_O)^{|T|}}{\frac{|B|}{2^{|T|}}} \; for \; \frac{|B|}{2^{|T|}} > R_S \\ \frac{R_I(1-C_O)^{|T|}}{R_S} \; otherwise \end{cases}$$

where R_S is the rating of the best match of the surrounding of object of I.

Now, our goal is to find a maximum Q dependent on $|T|$. A quality function dependent on $|T|$ consists of two functions on the two intervals $< 1, t >$ and (t, ∞), where t is a value of $|T|$ where these two functions switch. The switching point is:

$$\begin{aligned} \frac{|B|}{2^t} &= R_S \\ 2^t &= \frac{|B|}{R_S} \\ t &= \log_2 \frac{|B|}{R_S} \end{aligned}$$

For the first interval, we will find a maximum Q using the first derivative of Q:

$$\begin{aligned} Q &= \frac{R_I(1 - C_O)^{|T|}}{\frac{|B|}{2^{|T|}}} \\ Q &= \frac{R_I}{|B|} \times 2^{|T|} \times (1 - C_O)^{|T|} \\ Q' &= \frac{R_I}{|B|} \times 2^{|T|} \times \ln 2 \times (1 - C_O)^{|T|} + \\ &\quad + \frac{R_I}{|B|} \times 2^{|T|} \times \ln(1 - C_O) \times (1 - C_O)^{|T|} \\ Q' &= \frac{R_I}{|B|} \times 2^{|T|} \times (1 - C_O)^{|T|} \times (\ln 2 + \ln(1 - C_O)) \end{aligned}$$

The maximum of the function is at $Q' = 0$

$$\begin{aligned} 0 &= \frac{R_I}{|B|} \times 2^{|T|} \times (1 - C_O)^{|T|} \times (\ln 2 + \ln(1 - C_O)) \\ 0 &= 2^{|T|} \times (1 - C_O)^{|T|} \end{aligned}$$

Q' does not cross 0 for $T \in < 1, \log_2 \frac{|B|}{R_S} >$, i.e. Q is either rising or falling on the whole interval. To find out whether Q is rising or falling, we will find Q' for $|T| = 1$ and for $R_I > 0$, $|B| > 0$, $C_O \in < 0, 1/2 >$

$$\begin{aligned} Q' &= \frac{R_I}{|B|} \times 2^{|T|} \times (1 - C_O)^{|T|} \times (\ln 2 + \ln(1 - C_O)) \\ Q' &= \frac{R_I}{|B|} \times 2 \times (1 - C_O) \times (\ln 2 + \ln(1 - C_O)) \\ \frac{R_I}{|B|} &> 0 \ \wedge \ (1 - C_O) > 0 \ \wedge \ (\ln 2 + \ln(1 - C_O)) > 0 \ \Rightarrow \\ Q' &> 0 \end{aligned}$$

The function $Q(|T|)$ is rising on the interval $< 1, \log_2 \frac{|B|}{R_S} >$.
For the second interval, the first derivative of Q is

$$\begin{aligned} Q &= \frac{R_I (1 - C_O)^{|T|}}{R_S} \\ Q &= \frac{R_I}{R_S} \times (1 - C_O)^{|T|} \\ Q' &= \frac{R_I}{R_S} \times \ln(1 - C_O) \times (1 - C_O)^{|T|} \end{aligned}$$

The limit of the function is at $Q' = 0$

$$\begin{aligned} 0 &= \frac{R_I}{R_S} \times \ln(1 - C_O) \times (1 - C_O)^{|T|} \\ 0 &= (1 - C_O)^{|T|} \end{aligned}$$

Q' does not cross 0 for $|T| \in < \log_2 \frac{|B|}{R_S}, \infty >$, i.e. Q is either rising or falling on the whole interval. To find out whether Q is rising or falling, we will find Q' for $|T| > 1$ and for $R_I > 0$, $R_S > 0$, $|B| > 0$, $C_O \in < 0, 1/2 >$

$$\begin{aligned} Q' &= \frac{R_I}{R_S} \times \ln(1 - C_O) \times (1 - C_O)^{|T|} \\ \frac{R_I}{R_S} &> 0 \ \wedge \ \ln(1 - C_O) < 0 \ \wedge \ (1 - C_O)^{|T|} > 0 \ \Rightarrow \\ Q' &< 0 \end{aligned}$$

The function $Q(T)$ is falling on the interval $< 1, \log_2 \frac{|B|}{R_S} >$.

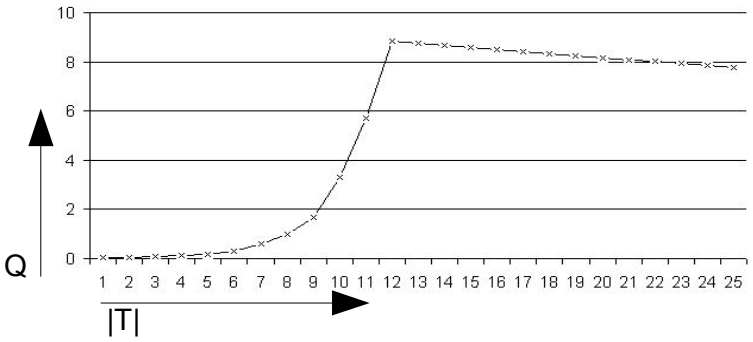


Figure 5.9: Example of Q dependent on $|T|$

The function consists of two intervals. The function is rising on $|T| \in < 1, \log_2 \frac{|B|}{R_S} >$ and it is falling on $|T| \in < \log_2 \frac{|B|}{R_S}, \infty >$. For illustration, an example of the function is in Figure 5.9.

Resulting, the size of the template $|T|$ for the highest image quality Q_G is

$$|T| = log_2 \frac{|B|}{R_S} \tag{5.1}$$

where $|B|$ is the number of templates in the bank and R_S is the rating of the best match of the surrounding of an object of the ideal image of image G.

5.4.5 Image Class

The proposed method is applicable to various object classes. Also, the successful detection of an object is largely dependent on the object's surrounding. That's why we define the *image class* as a reference term for the following computations.

Definition 5.4.6 *Image class A is a set of images.*

Definition 5.4.7 *Image class quality* Q_C^B *for class A and bank B is the quality of the object with the lowest rating from A against best match in the surrounding of an object from A*

Figure 5.10 will help illustrate the meaning of image class quality. In this figure, there is a typical distribution of ratings of detected objects in a class[4]. The x-axis is a rating, the y-axis is the number of detections with corresponding ratings, involving the whole image (not only detection of objects). On the right, there is a set of detected objects with relatively high rating. On the left, there is a set of non-objects with supposingly low rating. Image class quality is determined by the worst of the objects and the best of the non-objects.

In most cases in real life, there will be some objects missed by the algorithm. In these cases, the object and the non-object sets blend together and $Q_A^B < 1$, thus the significance of this value is low. For these cases, *hit rate* is defined as a fraction of the images with successfully detected objects.

[4]Class quality can be also used for multiple objects in a single image

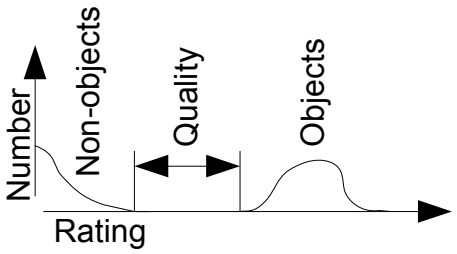


Figure 5.10: Typical distribution of ratings in a class

Definition 5.4.8 *The hit rate* $H_A^{B,L}$ *for class A, bank B, limit L, classification output* $U_{G,B}^L$ *and object* O_G *is*

$$H_A^{B,L} = \frac{|G| \ s.t. \ U_{G,B}^L \ hits \ Q_G}{|G|} \ \forall G \in A$$

Theoretically, image class A consists of all images of all scenes that the user considers valid. However, for real-life experiments and learning, we can use only a subset of A. For accurate image quality estimation, we have to choose the subset representing as many different scenes as possible.

5.4.6 Class Compactness

In this section, we will examine issues related with the number of templates in a bank. Template bank, and consequently the number of templates, would be ideally defined as a bank resulting in the highest class quality.

In real life, we can create templates from the subset of a class as described in Section 5.5, and manually determine a sufficient number of templates. This estimate of a sufficient number of templates will be based on the object size, average rating, and compactness of the class.

An important feature related to the number of templates is the *compactness* of image class, which is a function of the number of templates dependent on the size of the image class. In other words, compactness specifies how many more templates we have to add to successfuly detect new image in the class. There can be three basic types of compactness:

1. *Constant* compactness is defined by the function $|B| \sim a$. This type of compactness is ideal for the proposed method. It will be the case of detecting one object with different backgrounds, or objects consisting of the same slices, only scattered throughout the object.

2. *Logarithmic* compactness is defined by $|B| \sim a_1 + a_2 \times logn$, where n is the number of images in the class. This type of compactness is still acceptable, indicating the class consists of images with objects made from sub-objects (letters, numbers etc).

3. *Linear* compactness is defined by $|B| \sim a \times n$, where n is the number of images in the class. The class with this type of compactness is not suitable for the proposed method, indicating the class objects do not share one universal template bank.

5.5 Creating a Template Bank

Now we should discuss an issue that has been omitted so far - how to create the templates. An algorithm was suggested that creates a suitable template bank for every particular image class. Input to the algorithm is an image (or a set of images) with the coordinates of an object in the image. The principle of the algorithm is in Figure 5.11. First, the bank is filled with templates and sorted by quality. The better half of the templates is kept and the other half is replaced by new templates. The bank is sorted again and the process repeats, either on the same image or on the next image in the set, depending on the particular implementation. The algorithm may stop when the bank quality is sufficient, but the algorithm may also continue in an infinite loop as proposed in Chapter 7.

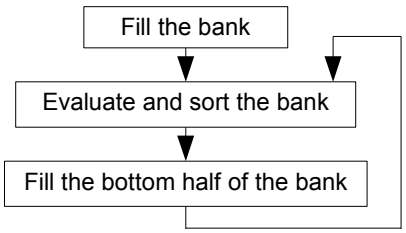


Figure 5.11: Creating a template bank

How are the templates obtained and evaluated? The algorithm takes an image and the coordinates of the object in that image. The algorithm then creates many templates based on small slices from the object in that image. The evaluation of a template is based on how many occurrences were found inside the object and how many outside. There may also be included a template occurrences limit, i.e. if the number of occurrences exceeds this limit, the template rating is zero. This prevents finding templates that are not related to the object structure, like empty template or completely filled template.

5.6 Chapter Conclusions

In this chapter, a template based detection method was proposed, fulfilling the requirements from Chapter 4. This method was roughly outlined in Section 5.1 and compared to the items from Chapter 4. The method was perfected in Section 5.2, and the final version was proposed in Section 5.3. The final method proposal consists of the preprocessing, feature extraction and classification, described using the definitions from Section 3.1.

Preprocessing consists of two filters - an edge detection filter and a local thresholding filter. A binary preprocessed image is passed to the feature extraction stage, which consists of a large set of filters that run in parallel. Each of the feature extraction filters then compares an image slice to a template, which is basically a small binary image representing a part of the object. The set of those templates is called a template bank. The final phase of the object detection is classification. For classification, various standard techniques can be used depending on the application that the method is being used for. Two of them were suggested in Section 5.3.3, using a sum of the features or the Adaboost method.

In Section 5.4, important parameters of the proposed method were analyzed. Results from two of them, the size of templates in Section 5.4.4 and the number of templates in Section 5.4.6 are used later in a case study to set the corresponding parameters. Section 5.5 describes the learning algorithm for creating the templates of this method.

Chapter 6

Experimental Architecture

In this chapter, we will deal with the experimental architecture for the method proposed in Section 5.3.

6.1 Why Is the Experimental Architecture Presented?

In this thesis, the new object detection method suitable for hardware implementation was suggested. And the suitability for hardware puts a question - How to confirm that the method is really "hardware friendly"? By only estimating the number of equivalent gates occupied by the method implementation, we may miss certain important factors effecting possible real-world implementation:

- Maximum propagation delay of the method implementation (could possibly cause the method to not be feasible for real implementation)
- Availability of other circuitry, like clock management circuits or memories, sufficiently large and fast
- Availability and price of a target implementation platform, i.e. we should know if the proposed method can compete with other contemporary methods

To summarize, by analyzing the hardware requirements without considering specific architectural issues, we may miss some important aspects of the proposed method. That's why experimental architecture is presented. The results of the experimental architecture experiments will show whether the method can be implemented in real life with today's technologies, and how it stands in comparison to other contemporary methods.

6.2 Overall Scheme

Figure 6.1 shows the overall scheme. The connecting line descriptors show the format of the data. Input to the system is a serial stream of data with bit length D representing pixels coming from a sensor. The pixels are ordered from left to right, lines from top to bottom. The whole scheme works "on-the-fly", i.e. whenever a new pixel comes to the input, a new pixel is computed and appears at the output (with a certain delay, caused by the serial to matrix unit).

Block *serial to matrix* (Section 6.4) is necessary, because filters work with surrounding of the actual pixel. It converts the serial pixel input to a matrix of $N \times N$ pixels, that

contains the actual pixel and its surroundings. For this purpose, $N-1$ image lines must be stored in memory. For FPGA implementation, the most suitable are internal Block RAMs.

With this matrix of pixels ($N \times N$ pixels with bit depth D), we can do the preprocessing in the *edge detection* and *threshold* units.

After preprocessing, feature extraction can take place using a template bank. It compares all $|B|$ templates with an image slice coming from the threshold unit, and if some of the templates fit, the proper output signal is set. There is a maximum of $|B|$ output signals, one for each template. However, some of the templates can share output signals which reduces the number of signals. Then, the appropriate signal is set when any of the joint templates match.

The output image should be an array with the identification numbers of matched templates, so the set of $|B|$ signals must be converted to a number of the matched template in *arr2num* unit. This number is finally the serial output forming a template occurrence image.

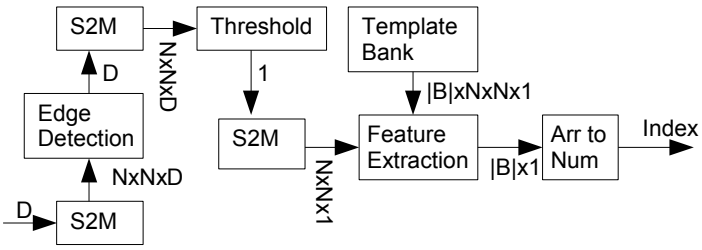


Figure 6.1: Overall hardware scheme

6.3 Simulator and HW platform

For the following chapters, we will need to provide some simulation and synthesis results. For simulation, we will use the ModelSim XE II 5.7g Starter Edition.

Device family	Device	Slices	BRAM bits	Price (March 2007)
Spartan-3	XC3S1500	13312	576k	$74
Spartan-3	XC3S5000	33280	1872k	$242
Spartan-3E	XC3S1600E	14752	648k	$63
Virtex-II	XC2V3000	14336	1728k	$630
Virtex-II	XC2V8000	46592	3024k	$8323
Virtex-II Pro	XC2VP30	13696	2448k	$405
Virtex-II Pro	XC2VP100	44096	7992k	$6235

Table 6.1: Edge Detection Synthesis Results

One of the Xilinx devices will be used for synthesis experiments, because there is a free synthesis software Xilinx ISE Webpack. There is a list of Xilinx FPGAs that could be used for synthesis in Table 6.1. This list includes low-cost families of the Spartan-3 and Spartan-3E, and high-end families of the Virtex-II and Virtex-II Pro. Virtex-II Pro also contains either one or two PowerPC processors.

According to the synthesis results in Section 6.10, we do not need vast BRAM resources or fast propagation through the chip, thus we can choose one of the low-cost models. Resulting, Spartan-3 XC3S1600E (E platform optimized for logic resources) has been chosen as a reference target FPGA, for its sufficient logic resources and affordable price. We could also consider Spartan-3 XC3S5000 if we need more templates implemented, or the Virtex-II Pro XC2VP30 in case we need a processor on the chip (now, we assume a processor placed on the board with an FPGA).

6.4 Serial to Matrix

The proposed method operates on the pixel neighborhood, and expects input pixels coming serially. This situation is suitable for hardware implementation, where only the relevant parts of the image will be stored in the memory. Several possibilities for implementation exist, including a standard systolic array [103] or the cellular architecture introduced by Porter et al. [105]. For our purposes, a systolic array implementation is fully satisfactory. Our implementation of systolic array engine called serial to matrix (S2M) is described in this section.

In [6], the S2M unit was introduced that converts serial data input (in our case pixels from a camera) into a matrix of values, representing a pixel and its neighborhood. As you can see (for example in Figure 3.1), this hardware circuit is necessary for the application of a filter.

6.4.1 Logic Scheme

Logically, the system consists of a FIFO memory that can store $h - 1$ lines of the image, where h is the height of the output matrix. There are h data outputs from the FIFO - the first one is at the very beginning of the FIFO, next outputs follow after the image width w_I. The last output is at the end of the FIFO. All of these outputs run through five registers operated by the input pixel clock - outputs from these registers create the $w \times h$ matrix. This matrix "runs" through the image, pixel by pixel, in the order of the input pixels. This scheme is depicted in Figure 6.2.

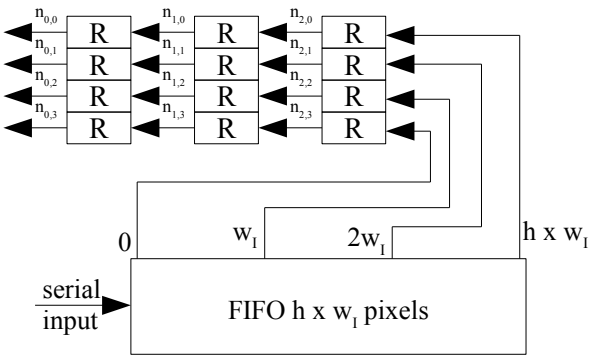


Figure 6.2: Logic scheme of an S2M unit for a 3×4 filter size

6.4.2 Hardware Scheme

As a result from the logic scheme that we will need to access more memory cells at each pixel clock cycle. There are two solutions for the standard memory types:

1. *One memory provides time multiplex in one pixel clock cycle.* Unfortunately, this solution may be unviable for more outputs - for example, for the standard 40 ns pixel clock and 8 outputs, the memory must be accessed every 5 ns.

2. *One memory is dedicated for every output.* For FPGA circuits, this is very handy solution because we can use on-chip block RAMs (Section 2.3), whose size roughly corresponds to the size of an image line. In the following text, we will consider only this concept.

Hardware realization (Figure 6.3) is different from the logic scheme. The FIFO is implemented by a set of block RAMs. One of the ports is used for the input and one for the output, which avoids possible collisions when accessing the Block RAMs.

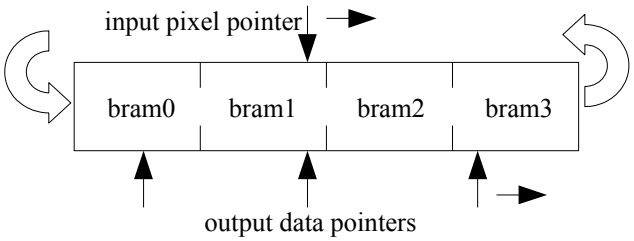


Figure 6.3: S2M logic scheme

Block RAMs, ordered consecutively, form one large round buffer. Pixels are written to this buffer serially. Pointers for output data are spaced uniformly within the buffer. All pointers (input and output) are moving the same way, one step per one pixel clock.

It's clear from the picture that the spacing between output pointers (which is equal to the length of an image line) must be the same or greater than the size of a block RAM. Otherwise there would be more than one reading from the block RAM at a time, which is not possible. What should be done if line is shorter than a block RAM? The easiest, but inefficient solution is to not use the n lowest bits of block RAMs, which will lead into using only every 2^nth word in the block RAMs. This virtually makes the block RAMs n times smaller.

A picture of a hardware scheme is in Figure 6.4. The input address register points to a place where the next pixel will be stored. The address register is incremented every pixel clock and is overflowing to zero, which complies with our goal of storing pixels serially to a round buffer. The lower bits of the address are used as the address to the block RAMs, while the upper bits set one of the chip select signals. Data input to all of the block RAMs is the same (a pixel from a camera) and also the address of the input data is the same.

Data output is realized in a similar way to the input, the difference is that there are more outputs coming from the different parts of the global memory, thus there are more units taking care of the address and the chip select signal. For every input to the output matrix, there is one address register. The main issue is that an address can point to any block RAM, so we have to create a unit (the address unit) that passes an address to an

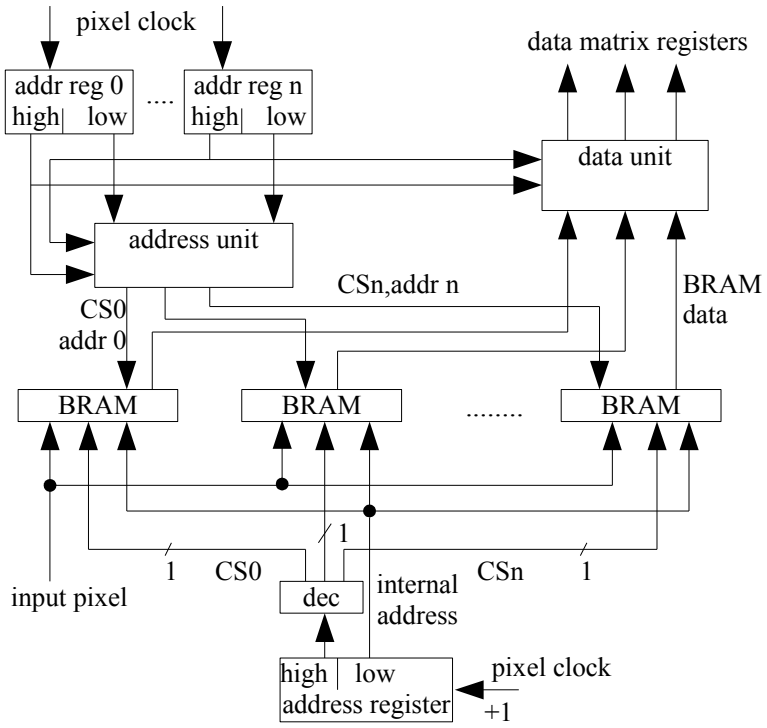


Figure 6.4: S2M hardware scheme

appropriate block RAM, and then sets a similar network to get data from the correct block RAM to the correct matrix input (data unit).

6.4.3 Image Borders

The introduced model does not address the problem with the image borders presented in Section 3.1. With this configuration, the outside pixels on the left and on the right are pixels from the other side of an image, while the outside pixels at the top or the bottom are even more random values from the image. Probably the easiest solution is to add a module to the very beginning of the datapath, that virtually adds a certain number of relevant pixels around an image. A detailed description of this module will not be placed here due to its easy implementation and low resource demands.

6.4.4 Synthesis Results

As the S2M unit can not be synthesized by itself, because of too many output signals, the synthesis results will be included together with the following units.

6.5 Edge Detection

The edge detection unit is a non-linear filter that outputs the difference between the minimum and the maximum in the pixel neighborhood. As subtraction is a simple operation, our main goal is to design a unit that finds the minimum or the maximum of an array of integer values.

A binary tree implementation seems to be the best option for the low propagation delay and resource utilization. The design principle is shown in Algorithm 6.5.1. Only operative parts have been included. The array of input values is in $temp(0, n),\ n \in < 0, 2^{size\_log\_c} >$. *size_log_c* is a binary logarithm of *size* and the *size* is the number of input values. The output is placed at signal MAX.

Algorithm 6.5.1 *Maximum Unit principle VHDL Code*

```
type t_temp is array (0 to size_log_c, 0 to 2**size_log_c-1)
   of std_logic_vector(depth_c-1 downto 0);
variable temp: t_temp;
...
   for j in 1 to size_log_c loop
      for i in 0 to (2**size_log_c)/(2**j)-1 loop
         if temp(j-1, i*2+1) > temp(j-1, i*2) then
            temp(j,i) := temp(j-1, i*2+1);
         else
            temp(j,i) := temp(j-1, i*2);
         end if;
      end loop;
   end loop;
MAX <= temp(size_log_c,0);
```

6.5.1 Synthesis Results

	Slices	BRAMs	Max. delay
Edge det.	468	4	21.0 ns

Table 6.2: Edge Detection Synthesis Results

The synthesis results can be found in Table 6.2. The unit S2M was included in the synthesis. Both chip occupation and propagation delay are very low, showing that the edge detection unit can be used in the system without any problems.

6.6 Thresholding

The thresholding unit is a non-linear filter that computes an average value of the maximum and minimum in the pixel neighborhood and outputs 1 for a pixel value above the average, or 0 otherwise. A comparison to the average is an easy operation, thus we lay the main focus on the min-max average part.

The design principle is almost identical to the Algorithm 6.5.1, and so is not included here. The only difference is that the resulting number is not an output, but is used for the thresholding operation.

6.6.1 Synthesis Results

The synthesis results can be found in Table 6.3. The unit S2M was included in the synthesis. Both chip occupation and propagation delay are very low, showing that the thresholding unit can be used in the system without any problems.

	Slices	BRAMs	Max. delay
Thresholding	352	4	14.4 ns

Table 6.3: Thresholding Synthesis Results

6.7 Feature Extraction Unit

The feature extraction unit is the most important part of the design. It compares the output of the thresholding unit to all of the templates. For implementation, it compares one bit array to n template bit arrays. The output is an n bit vector indicating which template arrays match.

The implementation process consisted of two phases. First, the circuit with no prior knowledge of the internal structure of the target circuit was designed. Because synthesis results did not meet the expectations, and because a more precise definition of the logic elements was needed for the design in Chapter 7, another design was suggested after examination of the internal chip structure. As the results were quite surprising, both mentioned designs are presented.

6.7.1 Design with no Knowledge about Target Platform

The design principle is shown in Algorithm 6.7.1. Only the operative parts have been included. The input bit array is in $temp(0, n),\ n \in < 0, 2^{size\_log\_c} >$. $size\_log\_c$ is a binary logarithm of $size$, and $size$ is number of input values. $size_c$ is the size of a template and $elements_c$ is the number of templates. $templates$ is an array that holds the template values.

First, the array arr_eq is created. It contains the results of the comparison of the input vector and all of the templates. If all bits from the arr_eq belonging to one template are equal to 1, i.e. all template bits match the input vector, the corresponding bit in the output vector OCC is set to 1.

Algorithm 6.7.1 *Feature extraction principle VHDL Code*

```
type templates_t is array(elements_c-1 downto 0) of
   std_logic_vector(size_c-1 downto 0);
signal templates: templates_t;
signal arr_in: std_logic_vector(size_c-1 downto 0);
type ttemp is array(size_log_c downto 0, 2**size_log_c-1 downto 0)
   of std_logic;
variable temp: ttemp;
...
   for el in 0 to elements_c-1 loop
      for i in 0 to size_c-1 loop
         arr_eq(el)(i) <= templates(el)(i) AND arr_in(i);
      end loop;
   end loop;

   for el in 0 to elements_c-1 loop
      for i in 0 to 2**size_log_c-1 loop
         temp(0,i) := arr_eq(el)(i);
      end loop;
```

```
      for j in 1 to size_log_c loop
         for i in 0 to (2**size_log_c)/(2**j)-1 loop
            temp(j,i) := temp(j-1, i*2+1) AND temp(j-1,i*2);
         end loop;
      end loop;
      OCC(el) <= temp(size_log_c,0);
   end loop;
```

	Slices	BRAMs	Max. delay
Feature extraction	13854	0	8.87 ns

Table 6.4: Feature Extraction Synthesis Results

The synthesis results can be found in Table 6.4 for 500 templates of size $5x5$ pixels (typical values used in experiments). Unit arr2num (Section 6.8) had to be included in the synthesis (however, the arr2num unit occupies only a small part of the FPGA chip), because the feature extraction unit has many output signals. The propagation delay is surprisingly low. As may be expected, the chip occupation is very high.

6.7.2 Design with Knowledge about Target Platform

Look up tables (LUTs) are the basic building blocks in the Xilinx FPGAs. Every LUT is configured by a 16bit array. The output of the LUT is the $n-th$ value of this array, where n is a binary number formed by the 4 LUT inputs. The template values will be stored in the LUTs arrays - all bits are reset to zeros, except the one for which the input combination matches the corresponding part of the template.

For a 5×5 template, there are 25 one-bit comparisons needed. The output is 1 when all template bits match the input, or 0 otherwise. This function can be performed using 8 LUTs. The organization of such a scheme is in Figure 6.5. The first six LUTs are comparing 24 input bits to the template values. The 7th LUT keeps the $8000h$ value which forms the 4 input AND. The 8th LUT works in a similar way to the 7th, while treating also the 25th bit of input.

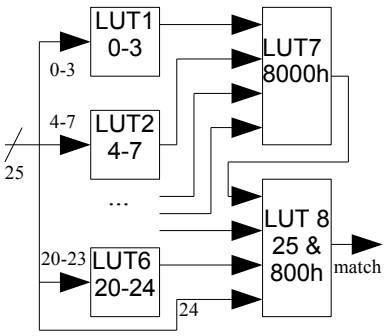


Figure 6.5: Fiting a 5x5 template comparison to 8 FPGA slices

The results in Table 6.5 for 500 templates show a big improvement in the occupation of the FPGA. By placing the design directly into the LUTs, the design size was reduced to 32% of the previous design. However, this 32% is a reduction of the original design

	Slices	BRAMs	Max. delay
Feature extraction	4537	0	18.31 ns

Table 6.5: Feature Extraction with LUTs Synthesis Results

together with the Arr2Num unit. If we also consider the results of Chapter 7, an estimate is that placing the design directly into the LUTs reduces its size to 18%.

6.8 Arr2Num Unit

The arr2num unit converts the n bit signal to the number of the position of the first non-zero signal. The design principle is shown in Algorithm 6.8.1. Only the operative parts have been included. *elements_log_c* is a binary logarithm of the number of templates. The input is the signal OCC from Algorithm 6.7.1, the output is NUM, which is the number of the first non-zero signal in OCC.

Algorithm 6.8.1 *Arr2num principle VHDL Code*

```
NUM <= (others => '0');
for i in 0 to elements_c-1 loop
   if OCC(i) = '1' then
      NUM <= conv_std_logic_vector(i, elements_log_c);
   end if;
end loop;
```

6.9 Classification

The architecture of the classification part will not be discussed thoroughly in this thesis since it is only a marginal topic and is largely application specific.

Minimal classification (only counting the number of active pixels in a region, Definition 5.3.12) could be efficiently implemented in hardware as a filter of the same size as an object. The architecture would be based on the concept introduced in Section 6.4, the important pixels would be on the right side and left side of the sliding window. The active pixels on the right side (entering the sliding window) would be added into a classification register, the active pixels on the left (leaving the sliding window) would be subtracted.

The architecture could be easily modified to perform the function that will be proposed in Section 8.4 by creating an array of n registers, where n is the number of features. Then, the architecture would perform the same function as in minimal classification for each of the registers.

Both of the architectures are limited by available BRAMs on an FPGA chip, as their number grows linearly with the height of the sliding window.

6.10 Overall Synthesis Results

In this section, the results from the previous sections are all put together. The target FPGA is the Xilinx Spartan III XC3S1600E.

	Slices	BRAM	Max. delay
Edge detection	468	2 kB	21.0 ns
Thresholding	352	2 kB	14.4 ns
Feature extraction	4537	0	18.31 ns
XC3S1600E	14752	648 kB	N/A
Total	5357 [36%]	4 [1%]	21.0 ns

Table 6.6: Overall Synthesis Results

The results are shown in Table 6.6. The design occupies 36% of the XC3S1600E FPGA chip. Maximum propagation delay of 21 ns allows "on-the-fly" implementation, as a normal HDTV cameras' pixel clock is usually 40 ns.

These results show that the method can be implemented in a real system, in fact there is no need of the more expensive, cutting-edge technology, FPGAs. More experimental results, including speed up estimation compared to a DSP solution, will be presented in the case study in Sections 8.5 and 8.6.

6.11 Chapter Conclusions

In the experimental architecture section, an architecture for the method from Chapter 5 was proposed. The architecture is based on implementation to an FPGA chip. Each of the units that the experimental architecture consists of was described in a separate section and synthesized to check the FPGA occupation and the maximum propagation delay.

The architectural core is the feature extraction unit (Section 6.7). An improvement was suggested by placing the templates directly into the processing elements of the target FPGA (Section 6.7.2), because the chip occupation is critically dependent on the implementation of this unit.

In Section 6.10, the synthesis results of all the units were put together. Thanks to the implementation of the feature extraction unit directly into the processing elements, it shows that the whole experimental architecture can fit into a standard low-cost FPGA. The propagation delay turned out to be low enough for the "on-the-fly" implementation.

Chapter 7

Adaptive Templates

The method proposal brings up a question: How should the bank be updated if the environment or objects change, and as a result, the bank does not give satisfactory results any more? This question will be answered in this chapter. Two techniques involved with updating the bank are proposed - a method using static reconfiguration and a method using dynamic reconfiguration. A description of the the dynamic reconfiguration approach was published in [3].

7.1 Static Reconfiguration

For the static reconfiguration technique, there will be predefined banks for different environment conditions or different objects at the computing system location. These banks can be, for instance, created on a PC from real images using the algorithm from Section 5.5. For example, there will be only one bank for an environment without changes (like a car detection in a tunnel), but there will be day/night/strong sun banks for the outside environments. The FPGA chip will then be statically reconfigured with the bank that fits best with the actual weather conditions.

7.2 Dynamic Reconfiguration

The dynamic reconfiguration block scheme is in Fig. 7.1. It is a modification of the hardware scheme in Figure 6.1. Together with the normal feature extraction process (using "current" template bank) there is a parallel branch for the feature extraction of new templates being tested ("test bank"). The results from both of these branches are evaluated and compared in the processor. The testing branch creates new templates for the test bank, and removes the worst templates. The normal branch adds good templates to the current bank from the test bank.

In order to replace old templates with new ones using FPGA dynamic reconfiguration, the templates have to be in fixed predefined positions. Probably the easiest way to do this is to place the templates in a regular array. Here a solution using standard Xilinx ISE tools is suggested with the direct placement of templates into LUTs, as described in Section 6.7.2.

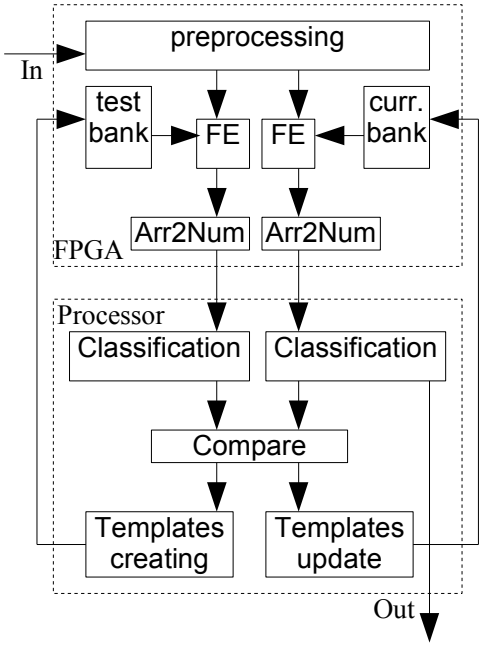


Figure 7.1: Dynamic reconfiguration

7.2.1 Creating the Regular Slice Array of Templates

Each slice of a Spartan-3 FPGA consists of two LUTs, so one template will only occupy 4 slices. This can be achieved by setting constraints in the UCF file. An example of implementation of half of the template follows:

```
INST "LUT4_10" LOC=SLICE_X70Y0;
INST "LUT4_20" LOC=SLICE_X70Y0;
INST "LUT4_30" LOC=SLICE_X70Y1;
INST "LUT4_40" LOC=SLICE_X70Y1;
```

$LUT4\_n$ is the label of the LUT component instantiation in the VHDL code, where n is the number of each of the 8 LUTs of the template. The last number (in this case, 0) is the template number. A simple C program automatically creating such a UCF file was created. Placement of LUTs iinnto the slices is shown in Figure 7.2. Numbers $XnYn$ show the number of a slice, numbers inside LUTs are numbers of the LUTs discussed earlier in this paragraph. In the figure, you see the beginning of the first (template 0) and the second (template 50) column.

Placement of the design in the XC3S1600E FPGA with two banks of filters, each containing 500 templates, is shown in Figure 7.3.

7.2.2 Reconfiguration of Banks

As can be seen in Figure 7.1, we need to be able to reconfigure both current and test banks by the processor. As we have shown before, template values are stored in the LUT configuration. Thus, we do not need to reconfigure anything else but the LUTs configurations in certain columns. The most efficient way is to directly change the configuration data for

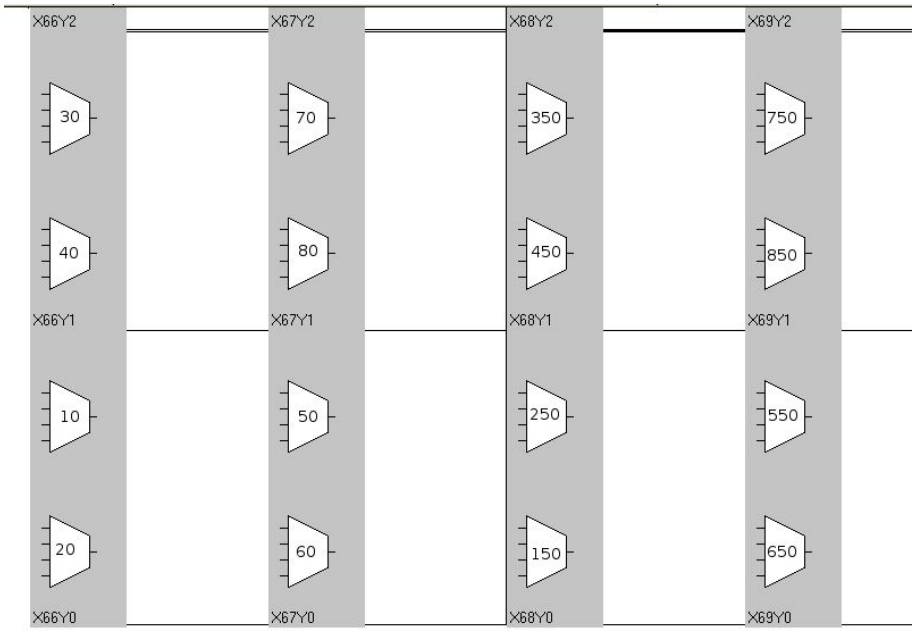


Figure 7.2: Position of LUTs in slices

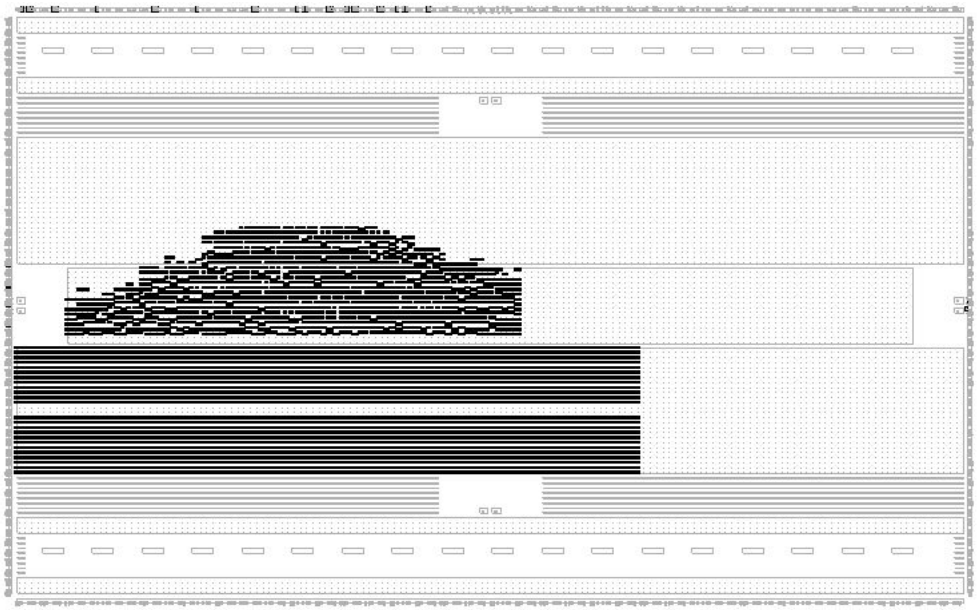


Figure 7.3: Placement of two template banks and related logic

the partial reconfiguration in the processor. To find out what bits in the bitstream are related to each LUT configuration, reverse engineering was used. However, this technique has not been tested on real hardware yet, as all the tests were performed only by software tools.

7.3 Chapter Conclusions

In the adaptive templates section, two techniques of dealing with the set of templates for the method from Chapter 5 were proposed. The first technique (Section 7.1) assumes different sets of pre-made templates are being downloaded to the system using static reconfiguration. The second technique, compared to the first one is considerably advanced. It offers automatic creation of suitable templates during system operation. It also uses dynamic reconfiguration for its operation, as described in Section 7.2.

Chapter 8

Case Study - License Plate Detection

To demonstrate that the proposed method suggested in Chapter 5 can be competitive with methods currently used in real life, the proposed method had to be implemented in an actual real-life situation and compared with some existing methods. This happened to be a difficult task, as the real-world methods and the testing data sets are usually proprietary. Fortunately, there was a chance to test the method on the real-life problem of detecting the license plates, the Unicam system[1]. In this chapter, the proposed method will be evaluated and compared to the method based on a DSP processor architecture currently used in the Unicam system.

Traffic related applications using computer vision are becoming more and more popular. With increasing computer performance, complex image operation tasks can be done. Therefore, there are more and more applications where a video camera is the only system input, which makes the whole system cheaper and more portable.

Red light running and speeding are no more the only traffic related tasks performed by computer vision techniques. For example, Sun et al. [78] suggested a real-time precrash system using the Haar wavelet transform, and Matsushita et al. [79] insert informatory signs into real images.

8.1 Introduction to the Unicam System

The Unicam system [37] is based on video-detection devices that use a microprocessor to analyze video image input from a video camera. There are two basic techniques, tripline and tracking used to detect traffic. Tripline techniques monitor specific zones on the video image to detect the presence of a vehicle, whereas video tracking techniques employ machine vision algorithms to identify and track vehicles as they pass through the field of view. Video technology can offer a wide variety of traffic information. In addition to conventional data such as volume, presence, occupancy, density, speed and classification, other data such as dwell time, incident detection and even origin destination information can be obtained. Video can also be used to provide surveillance information on a roadway as well as for photo-enforcement of traffic violations like red-light running and speeding.

[1]Thanks is given to Camea ltd. for supporting this research by providing me with the hardware boards and sample images. However, none of the outputs of the research part of this thesis have been used for commercial purposes by Camea ltd.

The Unicam system offers advanced machine vision technology to provide automated video image vehicle detection systems. It uses proprietary algorithms and special image processing modules to achieve great accuracy while keeping the overall system price low.

Some possible applications of the Unicam system are as follows:

- *Unicam Redlight* records red-light-violating vehicles. This option continuously monitors the lights at an intersection.
- *Unicam Section Control* records speeding vehicles, measures the average speed of a vehicle measured in the relatively long distance between two of more installed cameras.

8.1.1 Innovation

We showed that the embedded processing solution can be more efficient than the traditional one [4]. It is more compact, more easily serviced, etc., but in fact, the main advantage is that the system is distributable. This means that the cameras can be placed remotely from the PC that integrates the system functionality, and that the connection of the camera units can be done through the network connection which is easier to implement than a proprietary analog or digital video signal connection. In addition, this approach allows for the creation of widely distributed systems running in the agglomerations with long distances between the intelligent cameras and PCs. The basic differences between the traditional computer based approach and the embedded processing approach are shown in Fig. 8.1.

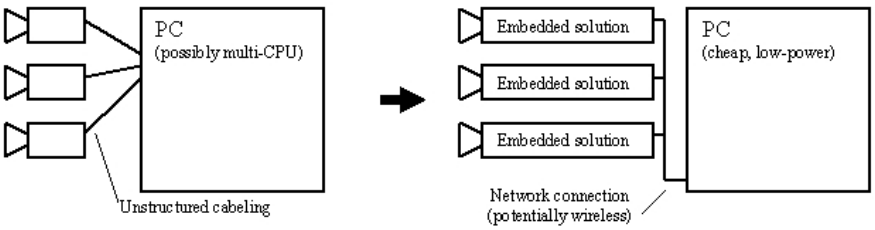


Figure 8.1: Traditional and embedded approach

8.1.2 Inter Chip Communicating Subsystem

To develop the experimental architecture proposed in Chapter 6, it was necessary to prepare a hardware platform to be ready for image processing experiments. This section shows part of this effort, an on-board communication system, introduced in [8].

This system allows communication between every two devices on the board also with the support of the dynamic reconfiguration of the FPGA on-board chip. The system can be controlled and dynamically reconfigured by any of the devices connected.

In Figure 8.2 the device called the *core* is the master of all of the communication. A device that wants to communicate with some other device requests the core for transfer of the data. A detailed description of this system is in [12].

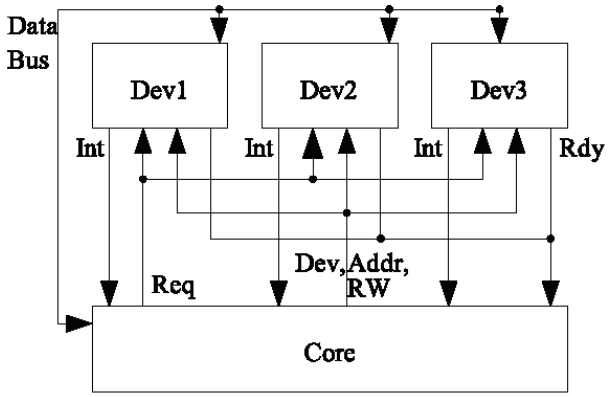


Figure 8.2: Inter chip communication core

8.2 Number of Templates

To determine the optimal number of templates, we will apply matters discussed in Section 5.4.6. The compactness of the class should be *logarithmic*, because the characters appearing in a license plate are a subset of all characters (thus compactness can not be constant), but the character set is finite (thus compactness can not be linear). With this knowledge, a few experiments were performed on the testing set. The best results reached were for the bank size of 300 to 700 templates. The number of templates was set to 500.

8.3 Size and Shape of Templates

Optimal size of the template is crucial for the proper function of the method. For easier implementation, there will be an initial requirement to the shape of the template, of it being square. To determine the right size, we will use the theory from Section 5.4.4, specifically the Equation 5.1:

$$|T| = log_2 \frac{|B|}{R_S}$$

where $|T|$ is the template size, $|B|$ is the number of templates in the bank and R_S is the rating of the best match of the surrounding of an object. The number of templates $|B|$ from Section 8.2 is 500. Now we will compute R_S. The average size of the license plate in the testing set is 3000 pixels. The number of detected templates in the best match is usually ranging around 10. Resulting,

$$R_S = \frac{10}{3000} = \frac{1}{3}$$

The optimal size of the template is then

$$|T| = log_2 \frac{|B|}{R_S} = log_2 \frac{500}{0.0033} = 17.2$$

Now we know how many pixels a template should contain, the question about the template shape remains. As license plates are rectangular and characters are not of any particular shape, the template shape should be rectangular. In particular, the shape of

the characters does not entitle the template to be stretched in either height or width. Resulting, the shape of the templates will be the square.

Considering the shape of the curve in Figure 5.9, we should rather choose larger than smaller templates. The closest larger square shape to 17.2 is a square of size 5×5 pixels, which will also be the shape used in the following experiments.

8.4 The Proposed Method Application

Implementation of the preprocessing and feature extraction is completely the same as proposed in Section 5.3. The classification part differs, which will be described in Section 5.3.3.

Preprocessing

An Example of an original image and a preprocessed image is are Fig. 8.3.

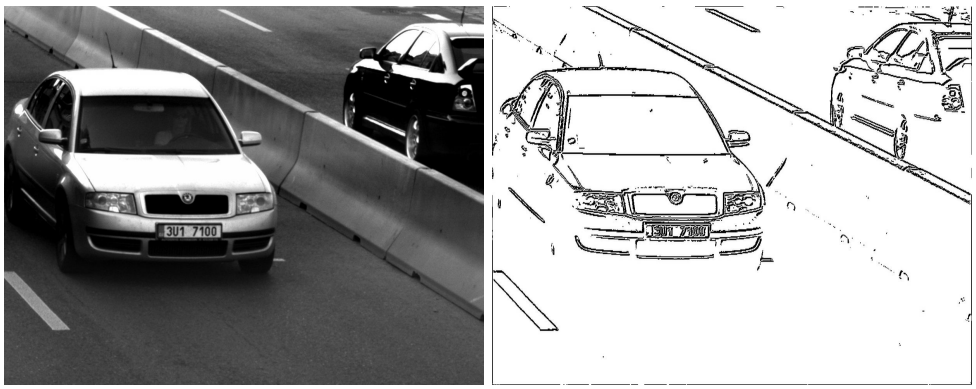


Figure 8.3: Original image and preprocessing

Feature Extraction

A 5×5 template size has been chosen and teh number of templates is 500, as specified in Sections 8.2 and 8.3. A template bank is created based on the algorithm in Section 5.5. An example of the template bank subset is shown in Fig. 8.4. An example of an image after feature extraction is in Fig. 8.5.

Figure 8.4: A template bank subset

Classification

The task of classification is to find the license plate, if present, in an image. A successful classification example is shown in Figure 8.5.

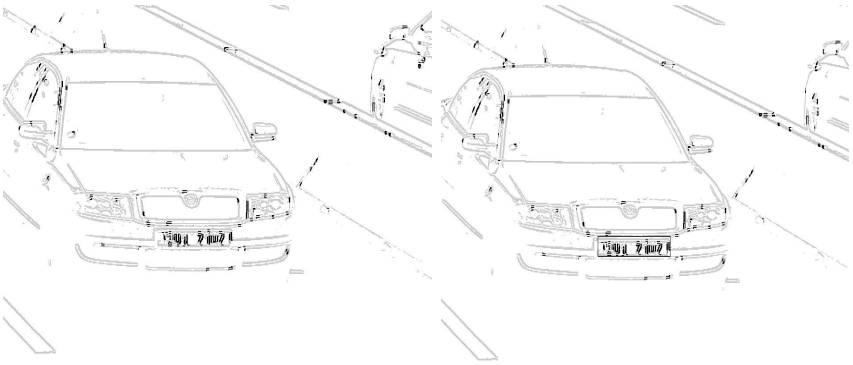


Figure 8.5: Feature extraction and classification

Classification, as presented in Section 5.3, suffers significantly from the regular patterns outside of the license plate. For example, if a template detects the pattern of a vent (Fig. 8.6), there may be more detected templates there, than in the license plate. However, although the number of templates is very high, there are usually only one or two templates involved.

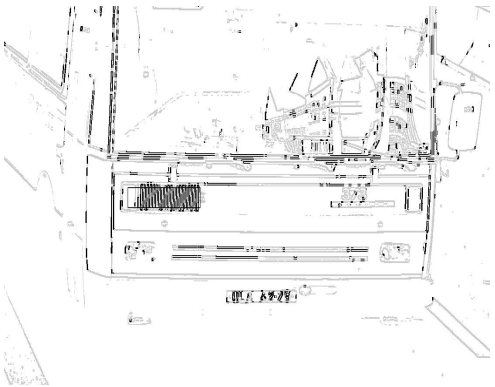


Figure 8.6: Example of the regular pattern problem

The problem of regular patterns and other classification problems lead us to a modification of the rating function defined in Section 5.3. One possibility is to consider not only the number of templates, but also how various they are (i.e. how many different templates are in the area). Then, this modified rating R_m of neighborhood N can be generally any function f of rating R of neighborhood N and a set L of detected templates in the neighborhood N.

$$R_m = f(R, L)$$

The regular pattern problem seen in the last example showes that this case study is very important part of this thesis. If the proposed method testing was performed only

on a limited number of images, the results would not have told us much about the actual method's properties.

Another possible improvement is considering that different parts of the neighborhood should contain different templates or different numbers of templates.

Definition 8.4.1 *The neighborhood part* $P \in \mathcal{N}_{x,y}^{G,S}$ *of neighborhood* $N \in \mathcal{N}_{x,y}^{G,S_N}$ *is a neighborhood such that*

$$S \subseteq S_N$$

The partitioned rating R_p of neighborhood N is

$$R_p = \sum R_{mi}$$

where R_{mi} is a modified rating of the neighborhood part of neighborhood N for $i \in \mathbb{N}$. An example of an application of a partitioned rating for the presented case study is in Figure 8.7. There are four parts of the license plate in the example. The border part should include only templates with straight lines or gradients, parts with numbers and letters can include any template, while the empty part should not include any template (i.e. function R_m will get negative results if any templates were detected).

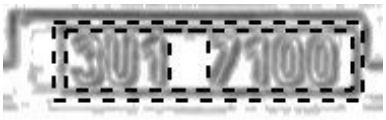


Figure 8.7: Partitioning of the license plate

For experiments, partitioning of the license plate wasn't used, mainly because the training images in different sets differed slightly in size. The modified rating was used in the experiments that considers how various the templates are, which helped significantly:

$$R_m = R \times (1 + |NT|I)$$

where $|NT|$ is the number of templates detected in neighborhood N and $I \in \mathbb{R}^+$ is a coefficient that represents the influence of $|NT|$.

8.5 Experimental Results

The method was tested on a set of images from real traffic. The results are compared to the method that is currently used for detection in Unicam cameras, and has been developed for many years. For evaluation, the metrics suggested by Mariano et al. [72] seem to be suitable. However, there is no data available concerning the accuracy of detection for the currently used method. As a result, only the binary information, whether the object is detected or not, was used.

The basis for the evaluation is the hit rate introduced in Definition 5.4.8. If a vehicle is present, the algorithm has to find the license plate position for successful detection. If there is no vehicle present (road images), the algorithm has to return "no license plate".

The proposed method has not been implemented as a whole in hardware yet. All tests were performed by a C program that works the same way as the design works in hardware.

The method needs to be trained first for every class of images by creating an appropriate template bank. For every image set, a subset containing 10 to 30 images was selected. Using a C program implementing the method from Section 5.5, 500 templates were obtained that were used as a bank for the whole image set. From the same subset, the limit value from Definition 5.3.10 was also obtained for each of the image sets. The limit value is used to cover images without any license plate.

Images were divided into the following groups:

- *Normal images*: standard quality images ranging from a little darker to a little brighter, also including images with the sun making shadows on the road
- *Light*: the license plate is too bright, possibly caused by too wide of an aperture
- *Dark*: the license plate is too dark, either because of bad lighting conditions or too narrow of an aperture
- *Shadows*: light coming from the side creating shadows on the license plate
- *Road*: road images without a car, under normal weather conditions
- *Snow road*: road images without a car, with bright light causing a reflection on the road, or with snow on the road, including images taken while snowing
- *Tilted*: images tilted between $-7°$ and $7°$, included to find out how precisely the camera has to be installed

Figure 8.8: An example of a light, a dark, a shadowy and a snow image

The results are presented in Table 8.1. *Current* and *proposed* are the two compared methods, *image set* is the number of images in each testing set, and the *occ* column is

Table 8.1: Experimental results

	Current	Proposed	Set	Occ
Normal	94.1%	97.1%	377	33%
Light	96.7%	100%	29	3%
Dark	62.5%	93.6%	63	3%
Shadows	90.2%	78.4%	162	1%
Road	100%	100%	233	51%
Snow road	99.1%	99.7%	321	9%
Tilted images	40.4%	91.8 %	98	N/A
Total	96.6%	98.6%	1283	100%

the rough estimate of percentages showing how often each group takes place in real-life. The results are shown in percents of successful detections. The last line, *total*, shows an estimate of the overall hit rate using the *occ* column.

The proposed method hit rate shows 2% better results than the current one. The credibility of this result may be affected by the limited test set, estimation of the *occ* values and the need of training set for each testing image group. However, the results still show that the proposed method quality is at least comparable to the current one.

8.6 Speed Up and Price

In this section, the proposed method is compared to two other implementations. First, the with implementation of the current method used in Section 8.5, implemented on the embedded processor. This comparison is the most important, because the current method serve as a reference. Second, we compare the proposed method implemented in C, on a PC computer, and in hardware to show the speed up achieved by moving the algorithm to hardware.

8.6.1 Comparison to the Current Method

The proposed method computes outputs "on-the-fly". Considering an image slice of size of 860×105 and a maximum propagation delay 21.0 ns, total computation time is

$$T_{hw} = 860 \times 105 \times 21.0ns = 1.9ms \tag{8.1}$$

The current method runtime on the embedded processor TMS 320C6416 is on average $T_{dsp} = 30$ ms, depending on the image complexity. The algorithm has been optimized for parallel processing in the processor. The speed up of the proposed method is

$$S_{dsp} = T_{dsp}/T_{hw} = 30/1.9 = 15.8 \tag{8.2}$$

Even though the hardware and software solutions work differently (dedicated hardware can not be used for other purposes while not in use), the speedup of 15.8 shows that the hardware solution is at least comparable to the DSP one.

The price of the suggested FPGA chip, the Spartan-3E 1600, is around $63. The price of the DSP chip, the TMS 320C6416, is around $150. The fraction of the hardware solution price over the DSP solution price is

$$P_{dsp} = \$63/\$150 = 0.42 \tag{8.3}$$

A simple price comparison is not going to give us a precise answer about which solution is cheaper. With the hardware solution, we still need to use some kind of processor for subsidiary functions. Also the other way around, with the DSP solution, we may need an FPGA chip for the low-level system functions. Still, the chip prices give us pretty good look on the final solution price.

8.6.2 Comparison to Software Implementation

The proposed method running on the Pentium M 1.6 GHz processor takes $T_{pm} = 9350$ ms. The speed up of the proposed method running on an FPGA is

$$S_{pm} = T_{pm}/T_{hw} = 9350/1.9 = 4921 \tag{8.4}$$

This speed up shows how efficient FPGA implementation may be for highly parallel tasks compared to processor implementation. Not only is the speed up itself extensive, but an FPGA runs at 48 MHz while the processor runs at 1.6 GHz.

However, these numbers do not reflect any real-life situation. The speedup is more or less proof that it makes no sense to implement the method in software, and that the method is suitable for hardware implementation. Considering statements from Section 1.3, the speed up helps to realize one of the goals that the method was designed for hardware implementation.

8.7 Categorization against Existing Methods

There are many ways of dealing with license plate detection. The method used for comparison in Section 8.5 is based on edge filters. Kamat et al. [40] use the Hough transform implemented on DSP processors. Hsieh and Yu [45] use algorithms based on the wavelet transform. Two consecutive images are compared for changes by Cui and Huang [42]. Li et al. [49] use a histogram and neural networks. Dalaff et al. [54] suggest a method based on following the image background. Porikili and Kocak [51] use the covariance descriptor and a neural network. Matas and Zimmermann [50] search for extremal regions, which are defined by contiguous sets of pixels. Zhang et al. [52] use Haar-like features proposed by Viola [53] together with the AdaBoost learning method. Other methods can be found in [41] [43] [44] [47] [48].

Not many license plate detection methods have been implemented in hardware. Bellas et al. [65] propose a method using opening and closing filters implemented in an FPGA. Cuchiara et al. [46] uses an FPGA implementation for comparing two consecutive images.

8.8 Chapter Conclusions

In the case study section, a system for the license plate detection based on the architecture proposed in Chapter 6 was presented. In Section 8.1, the Unicam system was briefly presented, which is the environment where the case study takes place. In Sections 8.2 and 8.3, the number of templates and the size of templates, respectively, were determined using the outcomes from Section 5.4.

The proposed method is applied to the license plate detection environment in Section 8.4. The proposed method was evaluated and compared to the method currently used in the Unicam system in Section 8.5. The evaluation was made on a set of almost 1300 real-life images from different environments. The results show that the detection quality of the proposed method is at least comparable to the current one being used.

In Section 8.6.1, the current method and the proposed method's hardware implementations are compared. The speed up of the proposed method against the current one is 15.8. Solution of the proposed method using FPGA seems to be approximately twice cheaper than the current DSP based solution.

In Section 8.6.2, the proposed method was compared to the software implementation of the same method. A speed up of about 5000 shows that the proposed method was designed for hardware, as discussed in Section 1.3.

Chapter 9

Subsidiary Case Study - Road Signs

To confirm the universality of the proposed method and to find out more about the method's properties, one more case study concerned with the detection of road signs is introduced. As this case study is only subsidiary, a rather artificial example of a road sign poster has been chosen.

This case study consists of two experiments. The first experiment should confirm the method utility and is similar to the case study from Chapter 8. The second experiment should show the proposed method's features and limitations, as the method samples and counter-samples were given very similar to each other.

9.1 Experiment One - Standard Function

The goal of this study is to create a template bank capable of detecting road signs of a specific type. The testing image is in Figure 9.2. Road signs meant for detection are in the first seven rows. Items that should not be detected are the informatory signs in the bottom of the image and the text accompanying the road signs.

The testing image was modified for training. The training image is in Figure 9.2. The training image consists of a part considered to contain objects marked by a dashed line. The rest of the training image is the background - algorithm is trying to find templates that match the object and *do not* match the background. The training set has been chosen to reflect main characteristics of objects, i.e. training set contains both circular and triangular signs.

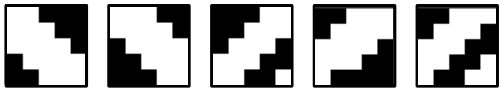


Figure 9.1: A template bank subset

To make things easier, the size (5×5) and number (500) of templates were kept from the previous case study. Implementation of the algorithm from Section 5.5 has been used to create a template bank. The first few templates from the bank are shown in Figure 9.1. The image after the application of the bank is in Figure 9.3.

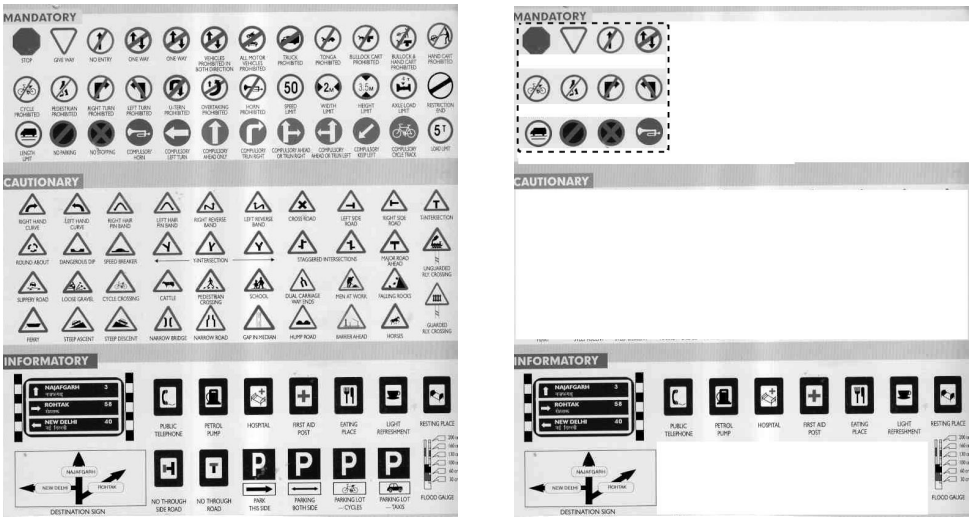


Figure 9.2: Testing image and training image of road signs for experiment one

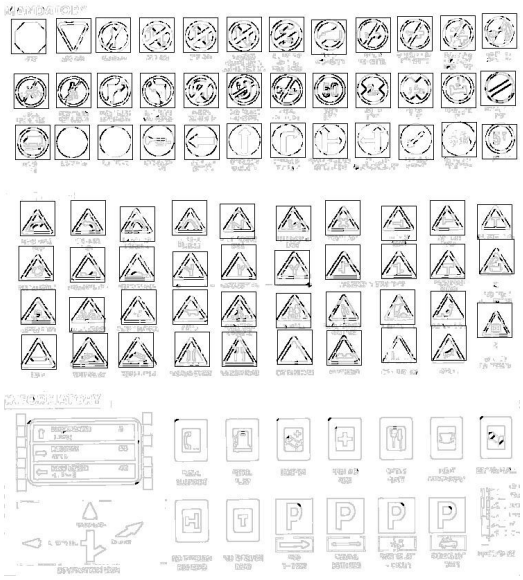


Figure 9.3: Road signs results for experiment one

Looking at the results in Figure 9.3, the questions about image class quality from Section 5.4.5 will arise. A distribution graph of object ratings is in Figure 9.4. On the x-axis there is a rating (based on a number of templates and a number of varying templates), and on the y-axis there is the number of objects with a particular rating. As can be seen, objects and non-objects are well divided, the distribution is very similar to the sample in Figure 5.10.

The quality of the set based on definition 5.4.7 is a fragment of the worst rating of an object and the best rating of a non-object, which in this case it is

$$616/116 = 5.3$$

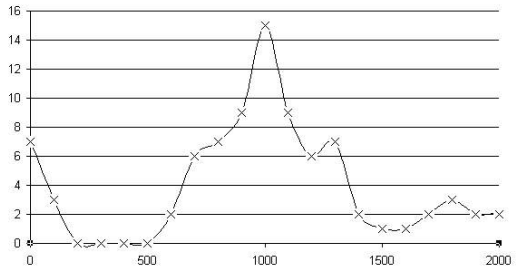


Figure 9.4: Rating distribution of experiment one

The values of this equation are also noticeable in Figure 9.4 as boundaries of the rating with zero occurrences.

Image class as we defined it (round and triangular road signs as objects, informatory road signs and text as background) has a good quality. However, if we wanted the triangular road signs to the be the background (not detected), image class quality would probably get significantly worse, as round and triangular signs share a lot of similar features. These ideas lead us into more general thoughts about the proposed method, its usage and limitations, studied in Section 9.2.

9.2 Experiment Two - Method Limitations

In this experiment, the proposed method's limitations are observed. Unlike in the previous experiment, where the samples and counter-samples were distinct, the samples and counter-samples are chosen to be very similar. The first image in Figure 9.6 is a training image for the experiment. The training set contains the 12 mandatory road signs on the left, the counter-samples are 12 different mandatory signs and some of the informatory signs. Cautionary signs are left alone.

The distribution of the rating is shown in Figure 9.5. Unlike in experiment one, there is no clearly recognizable boundary between objects and non-objects. It's not possible to properly set the limit value for classification (value from experiment one has been kept) because of this. The resulting detection is on the right image in Figure 9.6.

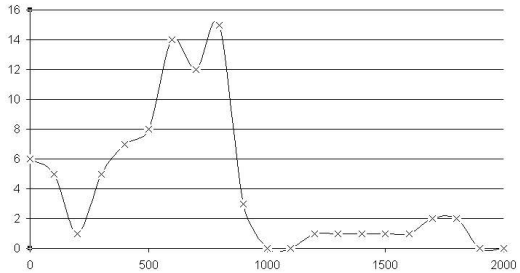


Figure 9.5: Rating distribution of experiment one

The image quality was determined only from the samples and counter samples, because it's not clear in this example what group the other signs should belong to. However, the image quality was

$$520/664 = 0.78$$

which means some of the learning samples had a worse rating than some of the counter samples. This result strengthens the assumption that the classes suitable for the proposed method must considerably vary on the object level comparable to the template size. Discussion about this topic will be covered in Section 10.1.

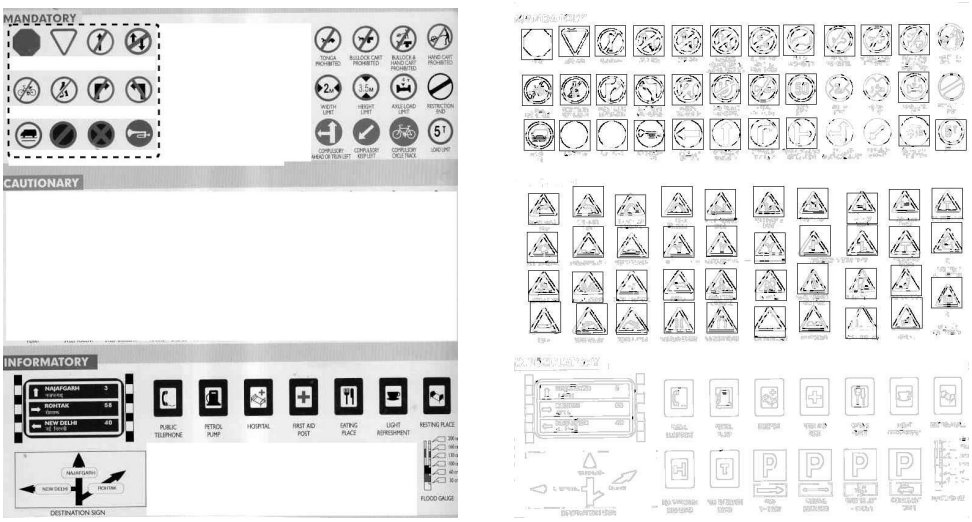


Figure 9.6: Training image and results of road signs experiment two

9.3 Chapter Conclusions

In the subsidiary case study chapter, another confirmation of the proposed method efficiency was achieved. The subject of the case study was a set of different types of road signs. In the first experiment in Section 9.1, the goal was to detect similar road signs as the ones in the training set. The outputs of the experiment were processed by techniques suggested in Section 5.4.5, resulting in the image class quality being 5.3, which means the quality of detection is good.

In the second experiment in Section 9.2, the goal was to push the limits of the method and to find an object set where the method fails. This was done by setting the training samples and counter samples to objects treated very similarly by the proposed method. The image class quality of this experiment was 0.78, which shows that the method failed in this experiment, as was expected.

Chapter 10

Discussion

This chapter is a general discussion about the proposed method.

10.1 Suitable Image Classes

Here, we will broadly discuss the suitability of image classes for the proposed method. These assumptions are based on an educated guess from my knowledge of the proposed method, together with experiments made on the method, for example the ones in Chapter 9.

Suitable Objects

Suitable objects are generally man-made objects with certain typical shapes:

- Cars, airplanes or other means of transport
- Consumer products made of plastic, iron, wood, ceramics or other non-deformable materials
- Printed boards
- Food products with fixed shapes like biscuits, chocolates or eggs
- Larger patterns with typical shapes
- Detection of defects with typical shapes like holes made by a tool with a specific shape
- Different types of signs, like road signs or logos
- Aerial images of buildings or other objects with typical shapes
- Printed letters, including license plates of vehicles

Non-Suitable Objects

Objects not suitable for the method are generally natural objects with irregular changing patterns

- Small patterns like sand or dirt, in an average size of few pixels, are not suitable for the proposed method.

- Food products with shapes with a large degree of freedom like pizza, bread, cakes or meat products
- Patterns with irregular shapes like wood or rocks
- Detection of cracks or other randomly shaped defects
- Natural objects like trees or water
- Aerial images of natural objects like mountains, desserts or forests
- Animals or humans, including faces or gestures

Problematic Objects

Objects rotationally and scale invariant can be generally detected by the method, but new templates have to be created for different angles or zooms. This leads into larger hardware demands and a higher price, which may be unacceptable in some cases.

Problematic Surroundings

An advantage of the proposed method is its resistance against a noisy surroundings, if we comply with the requirements on the size of templates in Section 5.4.4. Problematic surroundings are those with similar shapes as the templates. For example, for an object consisting of letters, a problematic surrounding would be one with letters of a similar size as object letters. A solution may be to involve a more sophisticated classification stage.

10.2 Categorization against Existing Methods

The necessity of creating a new template bank for every image class may be understood as a disadvantage compared to standard methods. However, this is a basic feature of our method and may be highly desired, for two reasons. First, a higher hit-rate may be achieved by proper training. Second, we may easily change the target object for detection by simple creating a new bank, which makes the method much more universal. Switching from searching for one object class to searching for completely different objects may be an easy step made by the operator, not requiring programmer interference.

Following is the discussion of the proposed method against methods with similar features.

10.2.1 Edge Detection

Described in Section 3.4.1, this method is easy and sometimes may give very good results.

Advantages of the Proposed Method

- More resistant against noise
- Adjustable by setting the template size
- Variable - detection of different objects by changing the template bank

Advantages of the Edge Detection Method

- Easy to implement
- Spends fewer resources

10.2.2 Hough Transform

Described in Section 3.4.2, this method is difficult to implement, but very powerful.

Advantages/Features of the Proposed Method

- Easier to implement in hardware
- Oriented on the shape of the whole object

Advantages/Features of the Hough Transform

- More complex, suitable for dealing with missing information
- Oriented on lines in the object

10.2.3 Motion Detection

Described in Section 3.4.6, this method is easy to implement in hardware for fixed environments, but very hard to implement otherwise.

Advantages/Features of the Proposed Method

- Detects also not-moving objects
- Does not suffer heavily from environment changes

Advantages/Features of the Motion Detection Method

- Perfectly detects moving objects, even if other methods fail

10.3 Application Specific Image Compression

Object detection is usually only the first step to other computer vision tasks, as can be seen in Section 8.1. One of the tasks where object detection can be helpful is application specific image compression. The basic idea of this technique [60] is to compress different image parts at a different compression level according to the importance in an image. Generally, an image can be divided into n areas by the proposed object detection method, each compressed by a different compression level [61].

For the Unicam system, application specific compression can be used for on-line monitoring of traffic, in case there is limited bandwidth in the data path. An example[1] is shown in Figure 10.1. There are three areas, located by the proposed method, bordered with the dashed lines. The most important area is the license plate, compressed with a low ratio in order to keep as many details as possible. The second area is the vehicle,

[1] Example was made artificially only for illustration

Figure 10.1: Application specific image compression for the Unicam system

compressed by a medium ratio to keep the shape and vehicle features, but not necessarily all details. The rest of the image, i.e. the road and other vehicles, is compressed with the highest ratio to save as much bandwidth as possible.

Another possibility how to reduce the amount of data is to utilize the template identifiers. The idea is simple. If a template matches an image slice, it means the image slice looks like the template. Provided there is a copy of the same template bank in the receiver side, parts of the image can be reconstructed from those template identifiers. However, there are limitations that may prevent this method from being applied. First, a grayscale image can not be fully reconstructed from binary templates. This is not a problem for binary images, or if we need to observe only the shape of an object. Second, a new method would have to be developed in order to merge the standard compression method and the proposed template identifiers method.

10.4 Chapter Conclusions

The discussion chapter consists of more general thoughts. In Section 10.1, the topic of the suitable and unsuitable image classes takes place. In Section 10.2, the proposed method is compared its alternatives, with pros and cons on both sides. In Section 10.3, an example of the larger system for application specific image compression using the proposed method is suggested.

Chapter 11

Conclusions

A New method for object detection has been proposed. An important feature of the proposed method is that it has been *designed for hardware implementation*. The proposed method is based on very simple elements, filters detecting only one template each. The filters are implemented in a parallel matrix in an FPGA. Although based on simple elements, the proposed method's results are comparable to commercial method results.

11.1 The Proposed Method Features

The proposed method follows the standard three step pattern recognition scheme - preprocessing, feature extraction, and classification.

Preprocessing consists of two filters - an edge detection filter and a local thresholding filter. A binary preprocessed image is passed to the feature extraction stage, which consists of a large set of filters that run in parallel. Each of these filters compares an image slice to a template. A template is basically a small binary image representing a small part of an object. The set of these templates is called a template bank. The final phase of the object detection process is classification. For classification, various standard techniques can be used depending on the application that the method is being used for. Two of them were suggested in Section 5.3.3, using a simple sum of the features or using the Adaboost method (Section 3.3.3).

11.2 Experimental Results

To show that the proposed method can be used in a real-life system, a case study dealing with the license plate detection was utilized. The proposed method was evaluated and compared to the method based on DSP processor architecture currently used in the Unicam system in Section 8.5. The evaluation was made on a set of almost 1300 real-life images from different environments. The results show that the detection quality the proposed method is at least comparable to the current one.

In Section 8.6.1, the proposed method and the DSP Unicam method hardware implementations were compared. The speed up of the proposed method against the current one is 15.8, and the solution of the proposed method using an FPGA seems to be approximately twice cheaper than the current DSP based solution.

In Section 8.6.2, the proposed method was compared to the software implementation of the same method. A speed up of about 5000 shows that the proposed method was designed for hardware, as discussed in Section 1.3.

11.3 Goal Fulfillment

The requirements specified in Chapter 4 have all been met, namely:

1. *The method is designed for hardware*, as the core of the method, feature extraction, is feasible for FPGA implementation. The proposed method can also profit from reconfiguration as shown in Chapter 7.
2. *Simple building blocks* are used. Template matching units are very simple, and can be placed in only a few FPGA slices, as shown in Section 6.7.2.
3. *Real-time processing* is used. Feature extraction is computed *"on-the-fly"*.
4. *Limited resources* are considered for method implementation. A standard FPGA is sufficient for the hardware part. The processor part, classification, is not too computationally difficult, thus an embedded processor implementation is suitable.
5. *Adaptability to changes* is a part of the method. The technique proposed in Chapter 7 allows adaptation to the environment or slow object changes.
6. The *results* in Section 8.5 show that the method efficiency is comparable to commercial methods.

11.4 Original Contribution

- In Section 3.1, I suggested a set of mathematical definitions for working with basic computer vision operations.
- In Section 5.3, I proposed a new method for object detection designed for hardware implementation.
- In Section 5.4, I analyzed the proposed method and suggested optimal size of the templates. I defined image quality for the method and image class quality.
- In Chapter 6, I proposed a hardware scheme for the proposed method, and confirmed its workability by simulation and synthesis of all of the major parts.
- In Chapter 7, I extended the proposed method to adaptation to environmental changes using partial dynamic FPGA reconfiguration.
- In Section 8.4, I applied the proposed method to the real-world problem of a license plate detection. To set the method parameters properly, I used the theory from Section 5.4 in Sections 8.2 and 8.3.
- In Section 8.5, I compared the detection abilities of the proposed method to a real-world license plate detection method. The testing set was composed of nearly 1300 real-world images.
- In Section 8.6, I compared the speed up and price of the proposed method to a real-world license plate detection method.
- In Section 9.1, I confirmed the proposed method's ability to detect different types of objects by applying the method to another case study.

- In Section 9.2, I explored the proposed method's limits by setting the learning set and counter set to objects that are very similar.

11.5 Chapter Summary

In this section, fundamentals from all chapters are summarized. Basically, parts of each of the chapter conclusion sections are put together here.

In Chapter 1, three fields important for this thesis are discussed - computer vision, embedded systems and programmable hardware. The goal of this thesis is laid - suggesting a new method for object detection designed for programmable hardware implementation. Following, questions concerning hardware are asked and answered, explaining certain concepts and decisions made in this thesis.

In Chapter 2, we discussed different ways how to design a computer system. Embedded systems and PLDs are important for this thesis. Particular attention was paid to FPGAs, which was the target technology for the experimental architecture in Chapter 6. Reconfiguration capabilities of FPGAs were also discussed, as reconfiguration was used in Chapter 7.

In Chapter 3, important theoretical fields connected with this thesis were introduced and discussed. Basics of pattern recognition were introduced in Section 3.3. Using these basics, object detection techniques were presented in Section 3.4. Each of the methods was discussed with relation to suitability for hardware implementation. In Section 3.4.8, hardware architectures and methods were presented.

For proper definitions of the image operations throughout this thesis, some mathematical apparatus was needed. In Section 3.1, the theoretical background was defined, based on *image* and *neighborhood*, both of which are defined as a matrix, and *structuring element* defined as a set of ordered pairs. This notation is easy to use and sufficient to describe all results and computations throughout this thesis.

Using the mentioned mathematical definitions, basic types of filters are presented in Section 3.2. For this thesis, the most important are the morphological filters presented in Section 3.2.2, particularly the hit-or-miss filter described in Section 3.2.2.

In Chapter 4, goals to be completed in this thesis are defined. The primary goal is to show that the proposed method, though based on very simple elements, can compete with commercially used methods due to the massive parallelization of those simple elements.

In Chapter 5, the template based detection method was proposed, fulfilling the requirements from Chapter 4. The method was roughly outlined in Section 5.1 and compared to the items from Chapter 4. The method was perfected in Section 5.2, and the final version was proposed in Section 5.3. The final method proposal consists of the preprocessing, feature extraction and classification definitions, all of them described using the definitions from Section 3.1.

In Section 5.4, important parameters of the proposed method were analyzed. Results from the two of them, the size of templates in Section 5.4.4 and the number of templates in Section 5.4.6 are used later in the case study to set the corresponding parameters. Section 5.5 describes the learning algorithm of creating the templates for the method.

In Chapter 6, an architecture for the method from Chapter 5 was proposed. The architecture is based on implementation to an FPGA chip. Each of the units that the experimental architecture consists of was described in separate section and synthesized to check the FPGA occupation and the maximum propagation delay.

The architectural core is the feature extraction unit (Section 6.7). An improvement was suggested by placing the templates directly into the processing elements of the target FPGA (Section 6.7.2), because the chip occupation is critically dependent on the implementation of this unit.

In Section 6.10, the synthesis results of all the units were put together. Thanks to the implementation of the feature extraction unit directly into the processing elements, it shows that the whole experimental architecture can fit into a standard low-cost FPGA. The propagation delay turned out to be low enough for the "on-the-fly" implementation.

In Chapter 7, two techniques of dealing with the set of templates for the method from Chapter 5 were proposed. The first technique (Section 7.1) assumes different sets of pre-made templates are being downloaded to the system using static reconfiguration. The second technique, compared to the first one is considerably advanced. It offers automatic creation of suitable templates during system operation. It also uses dynamic reconfiguration for its operation, as described in Section 7.2.

In Chapter 8, a system for the license plate detection based on the architecture proposed in Chapter 6 was presented. In Section 8.1, the Unicam system was briefly presented, which is the environment where the case study takes place. In Sections 8.2 and 8.3, the number of templates and the size of templates, respectively, were determined using the outcomes from Section 5.4.

In Chapter 9, another confirmation of the proposed method efficiency was achieved. The subject of the case study was a set of different types of road signs. In the first experiment in Section 9.1, the goal was to detect similar road signs as the ones in the training set. The outputs of the experiment were processed by techniques suggested in Section 5.4.5, resulting in the image class quality being 5.3, which means the quality of detection is good.

Chapter 10 consisted of more general thoughts. In Section 10.1, topic of the suitable and unsuitable image classes takes place. In Section 10.2, the proposed method is compared to the methods that could be used instead of the proposed method, with pros and cons on the both sides. In Section 10.3, an example of the larger system for application specific image compression using the proposed method as its part is suggested.

11.6 Future Research

Even though the method has been tested on real data and all parts have been synthesized, there is a long way to real implementation. Consequently, the first future goal is to make the system work in real hardware system, processing real data on site. A statistical evaluation of the results of this on-site system will help to find possible weak spots in the method, resulting in making the method more efficient.

Having the real hardware system would also allow us to analyze some other method's features. For example, in Section 1.2, we stated that a PLD implementation can be significantly more energy efficient than a processor implementation. If this statement was confirmed, it would be an important advantage of the proposed method.

References

Author's publications (Bryan L., Crha L.)

[1] Bryan L., Fučík O., Drábek V.: HW-Based Object Detection Method for Traffic Monitoring, 6th Electronic Circuits and Systems Conference, Bratislava, SK, 2007, accepted for publication

[2] Crha L.: System for the license plate detection and image compression using hardware, In: Proc. of the 7th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems, Bratislava, SK, SAV, 2004, p. 274-276, ISBN 80-969117-9-1

[3] Bryan L., Fučík O.: FPGA Implementation of a Reconfigurable License Plate Detection Method, Proceedings of the 2007 Engineering of Reconfigurable Systems and Algorithms, Las Vegas, NV, US, 2007

[4] Fučík O., Zemčík P., Tupec P., Crha L., Herout A.: The Networked Photo-Enforcement and Traffic Monitoring System Unicam, Proceedings of Engineering of Computer-Based Systems, Los Alamitos, US, IEEE CS, 2004, p. 423-428, ISBN 0-7695-2125-8

[5] Crha L., Fučík O., Šustek J.: Environment for Hw/Sw Codesign of Embedded Systems, Proceedings of the 8th IEEE Design and Diagnostic of Electronic Circuits and Systems Workshop, Sopron, HU, UWH, 2005, ISBN 9639364487

[6] Crha L., Fučík O., Drábek V.: Image filter implementation in FPGA used for the license plate detection, Proceedings of 38th International Conference Modeling and Simulation of Systems, 2004, Ostrava, CZ, MARQ, 2004, ISBN 80-85988-98-4

[7] Marek T., Novotný M., Crha L.: Design and Implementation of the Memory Scheduler for the FPGA - Based Router, Proc. of the Field Programmable Logic and Application 2004, Leuven, BE, Springer, 2004, p. 1133-1139, ISBN 3-540-22989-2

[8] Crha L.: Nové metody komprese, Sborník príspěvků ze semináře Počítačové Architektury & Diagnostika, Brno, CZ, FIT VUT, 2003, ISBN 80-214-2471-0

[9] Crha L.: Systém pro aplikačně specifickou kompresi obrazu, Zborník príspevkov Československého seminára pre študentov doktorandského štúdia Počítačové architektúry & Diagnostika, Bratislava, SK, SAV, 2004, p. 94-100, ISBN 80-969202-0-0

[10] Zemčík P., Herout A., Crha L., Fučík O., Tupec P.: Particle rendering engine in DSP and FPGA, Proceedings of Engineering of Computer-Based Systems, Los Alamitos, US, IEEE CS, 2004, p. 423-428, ISBN 0-7695-2125-8

[11] Bryan L.: A Set of Definitions for Working with Spatial Filters, Proceedings of the 13th Student Conference and Competition on Electrical Engineering, Information and Communication Technologies, 2007 Volume 4, Brno, CZ, VUT Brno, 2007, p. 430-434, ISBN 80-214-3410-3

[12] Crha L., Fučík O., Zemčík P., Drábek V., Tupec P.: Inter chip communicating system with dynamically reconfigurable hardware support, Proceedings of the 6th IEEE Design and Diagnostic of Electronic Circuits and Systems Workshop, Poznan, Poland, 2003, p. 311-312, ISBN 83-7143-557-6

[13] Venard O., Blanchard Y., Lionti R., Crha L: Single chip FPGA realization of a 2D multicomponent wavelet transform, 3rd IEEE International Symposium on Image and Signal Processing and Analysis, Rome, IT, 2003, ISBN 953-184-062-8

[14] Crha L.: Jak se píše procesor, ABC Linuxu, Vol. 2005, Praha, CZ, ISSN 1214-1267, `http://www.abclinuxu.cz/clanky/programovani/jak-se-pise-procesor`

[15] Crha L: 2D Multicomponent Wavelet Transform, Preprints of IFAC Workshop on Programmable Devices and Systems Conference, FEI VŠB, Ostrava, CZ, 2003, p. 384-390, ISBN 0-08-044130-0

[16] Crha L: CPLD, FPGA and DSP communication, Proceedings of the 9th Student Conference and Competition on Electrical Engineering, Information and Communication Technologies, Brno, CZ, 2003, p. 619-623, ISBN 80-214-2379-X

[17] Crha L.: Wavelet Transform Implementation in FPGA, diploma thesis, ESIEE Paris, FR, FIT VUT Brno, 2002

Other publications

[18] Davies, E.R.: Machine Vision: Theory, Algorithms, Practicalities, 3^{rd} edition, San Francisco, CA, USA, Elsevier inc., 2005, ISBN 0-12-206093-8

[19] Duda R., Hart P., Stork D.: Pattern Classification, Second edition, John Wiley & Sons Inc., New York NY, 2000, ISBN 0-471-05669-3

[20] Jahne B., Hausecker H., Geisler P.: Handbook of Computer Vision and Applications, Academic Press, San Diego 1999, ISBN 0-12-379770-5

[21] Russ J.: The Image Processing Handbook, Third Edition, CRC Press, Boca Raton, FL, 1999, ISBN 0-8493-2532-3

[22] Ritter G.: Image Algebra, Center for Computer Vision and Visualization, Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL, 1993, 1999

[23] Ritter G., Wilson J.: Handbook of Computer Vision Algorithms in Image Algebra, CRC Press, Boca Raton, FL, 1996, ISBN 0-8493-2636-2

[24] Forsyth D., Ponce J.: Computer Vision: A Modern Approach, Prentice Hall, 2003, ISBN 978-0130851987

[25] Fisher B., Perkins S., Walker A., Wolfart E.: HIPR - HyperMedia Image Processing Reference, Department of Artificial Intelligence, University of Edinburgh, UK, 1994, `http://www.cee.hw.ac.uk/hipr/html/hipr_top.html`

[26] Ripley B.: Pattern Recognition and Neural Networks, Cambridge University Press, Cambridge, UK, 1996, ISBN 0 521 46086 7

[27] Bentley P.: Evolutionary Design by Computers, 1st edition, Morgan Kaufmann, 1999, ISBN 978-1558606050

[28] Amit Y.: 2D Object Detection and Recognition, Models, Algorithms, and Networks, The MIT Press, Cambridge, MA, 2002, ISBN 0-262-01194-8

[29] Peat J.: Scientific Writing Easy When You Know How, BMJ Books, London, UK, 2002, ISBN 0 7279 1625 4

[30] Gause J., Cheung P., Luk W.: Reconfigurable Shape-Adaptive Template Matching Architectures, Proceedings of the 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2002, p. 98-107, ISBN 9780769518015

[31] Porter R.: Evolution on FPGAs for Feature Extraction, PhD Thesis, Queensland University of Technology, Brisbane, Australia, 2001

[32] Sun Z., Debis G., Miller R.: On-Road Vehicle Detection: A Review, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 28, no. 5, May 2006, p. 694-711, ISSN 0162-8828

[33] Bariamis D., Iakovidis D, Maroulis D., Karkanis S.: An FPGA-based Architecture for Real Time Image Feature Extraction, Proceedings of the 17th International Conference on Pattern Recognition, 2004, p. 801-804, ISBN 0-7695-2128-2

[34] Rigoll G., Kosmala A.: New Improved Feature Extraction Methods for Realtime High Performance Image Sequence Recognition, Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, 1997, p. 2901-2904, ISBN 978-0818679193

[35] Parker, J.R.: Gray Level Thresholding in Badly Illuminated Images, Transactions on Pattern Analysis and Machine Intelligence, vol. 13, no. 8, 1991, p. 813-820, ISBN 0162-8828/91/0800

[36] Knuth D.: The Art of Computer Programming, Volume 3: Sorting and Searching, Third Edition, Addison-Wesley, 1997, ISBN 0-201-89685-0

[37] `http://www.camea.cz`

[38] Chang Ch., Hwang J.: On the Face Detection with Adaptive Template Matching and Cascaded Object Detection for Ubiquitous Computing Environment, Lecture Notes in Computer Science, Computational Science and Its Applications, Springer-Verlag, Berlin, 2005, Heidelberg, p. 1204-1212, ISSN 0302-9743

[39] Zhao L., Davis L.: Closely Coupled Object Detection and Segmentation, Proceedings of the 10th IEEE International Conference on Computer Vision, 2005, p. 454-461, ISBN 1550-5499/05

[40] Kamat V., Ganesan S.: An Efficient Implementation of the Hough Transform for Detecting Vehicle License Plate Using DSPs, Proceedings of the Real-Time Technology and Applications Symposium, IEEE Computer Society, 1995, p. 58-59, ISBN 0-8186-8016-4

[41] Lim D., Choi S., Jun J.: Automated Detection of All Kinds of Violations at a Street Intersection Using Real Time Individual Vehicle Tracking, Proceedings of the 5th IEEE Southwest Symposium on Image Analysis and Interpretation, IEEE Computer Society, Washington, DC, USA, 2002, p. 126-130, ISBN 0-7695-1537-1

[42] Cui Y., Huang Q.: Automatic License Extraction from Moving Vehicles, Proceedings of the 1997 International Conference on Image Processing, IEEE Computer Society, Washington, DC, USA, 1997, p. 126-130, ISBN 0-8186-8183-7

[43] Lee H., Chen S., Wang S.: Extraction and Recognition of License Plates of Motorcycles and Vehicles on Highways, Proceedings of the 17th International Conference on Pattern Recognition, IEEE Computer Society, Washington, DC, USA, 2004, p. 356-359, ISBN 0-7695-2128-2

[44] Jia W., Zhang H., He X.: Mean Shift for Accurate Number Plate Detection, Proceedings of the 3rd International Conference on Information Technology and Applications, IEEE Computer Society, Washington, DC, USA, 2005, p. 732-737, ISBN 0-7695-2316-1

[45] Hsieh J., Yu J., Hung K.: Multiple License Plate Detection for Complex Background, Proceedings of the 19th International Conference on Advanced Information Networking and Applications, IEEE Computer Society, Washington, DC, USA, 2005, p. 389-392, ISBN 0-7695-2249-1

[46] Cucchiara R., Piccardi M., Prati A., Scarabottolo N.: Real-time Detection of Moving Vehicles, Proceedings of International Conference on Image Analysis and Processings, IEEE Computer Society, Washington, DC, USA, 1999, p. 618-624, ISBN 0-7695-0040-4

[47] Yang F., Ma Z.: Vehicle License Plate location Based on Histogramming and Matematical Morphology, Proceedings of the 4th IEEE Workshop on Automatic Identification Advanced Technologies, IEEE Computer Society, Washington, DC, USA, 2005, p. 89-94, ISBN 0-7695-2475-3

[48] Ko M., Kim Y.: License Plate Surveillance System Using Weighted Template Matching, Proceedings of the 32nd Applied Imagery Pattern Recognition Workshop, IEEE Computer Society, Washington, DC, USA, 2003, p. 269-275, ISBN 0-7695-2029-4/03

[49] Li G., Zeng R., Lin L.: Research on Vehicle License Plate Location Based on Neural Networks, Proceedings of the 1st International Conference on Innovative Computing, Information and Control, IEEE Computer Society, Washington, DC, USA, 2006, p. 174-177, ISBN 0-7695-2616-0/06

[50] Matas J., Zimmermann K.: Unconstrained Licence Plate and Text Localization and Recognition, Proceedings of the IEEE Intelligent Transportation Systems, 2005, p. 225- 230, ISBN 0-7803-9215-9

[51] Porikli F., Kocak T.: Robust License Plate Detection Using Covariance Descriptor in a Neural Network Framework, Proceedings of the IEEE International Conference on Video and Signal Based Surveillance, 2006, p. 107-111, ISBN 0-7695-2688-8/06

[52] Zhang H., Jia W., He X., Wu Q.: Learning-Based License Plate Detection Using Global and Local Features, Proceedings of the The 18th International Conference on Pattern Recognition, 2006, p. 1102-1105, ISBN 0-7695-2521-0/06

[53] Viola P., Jones M.: Rapid Object Detection using a Boosted Cascade of Simple Features, Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2001, p. I511-I518, ISSN 1063-6919

[54] Dalaff C., Reulke R., Kroen A., Ruhe M., Schischmanow A., Schlotzhauer G., Tuchscherer W., Kahl T.: A Traffic Object Detection System for Road Traffic Measurement and Management, Proceedings of Image and Vision Computing, New Zealand, 2003, p. 78-83, ISBN 0-473-10523-3

[55] Etiennne-Cummings R., Pouliquen P., Lewis M.: Single Chip for Imaging, Color Segmentation, Histogramming and Pattern Matching, Electronic Letters, Vol. 38, No. 4, 2002, p. 172 174, ISBN 0-7803-7335-9

[56] Li X., Ni G., Cui Y., Pu T., Zhong Y.: Real-time image histogram equalization using FPGA, Proc. SPIE Vol. 3561, Electronic Imaging and Multimedia Systems II, p. 293-299, 1998, 1998SPIE.3561..293L

[57] Hough P.: Method and Means for Recognizing Complex Patterns, U.S. Patent 3069654, 1962

[58] Duda O., Hart P.: Use of the Hough Transformtion to Detect Lines and Curves in Pictures, Comm. ACM, Vol 15, ACM Press, New York, NY, USA, 1972, p. 11-15, ISSN 0001-0782

[59] Rowe N.: Image Processing, U.S. Naval Postgraduate School, `http://www.cs.nps.navy.mil/people/faculty/rowe/imageprocover.htm`

[60] Ding J., Furgeson J., Sha E.: Application Specific Image Compression for Virtual Conferencing, Proceedings of the The International Conference on Information Technology: Coding and Computing, 2000, p. 48-54, ISBN 0-7695-0540-6

[61] Moni S.: Application-Specific Image Compression for Multimedia Applications, Journal of Electronic Imaging, July 1998, Volume 7, Issue 3, p. 464-473

[62] Zou Y., Dunsmuir W.: Edge Detection Using Generalized Root Signals of 2-D Median Filtering, Proceedings of the 1997 International Conference on Image Processing, 1997, p. 417-420, ISBN 0-8186-8183-7/97

[63] Jung C., Schramm R.: Rectangle Detection based on a Windowed Hough Transform, Proceedings of the 17th Brazilian Symposium on Computer Graphics and Image Processing, IEEE Computer Society 2004, p. 113-120, ISBN 1530-1834/04

[64] Rahman C., Badawy W., Radmanesh A.: A Real Time Vehicles License Plate Recognition System, Proceedings of the IEEE Conference on Advanced Video and Signal Based Surveillance, 2003, p. 163-167, ISBN 0-7695-1971

[65] Bellas N., Chai S., Dwyer M., Linzmeier D.: FPGA implementation of a license plate recognition SoC using automatically generated streaming accelerators, Proceedings of the 20th International Parallel and Distributed Processing Symposium, 2006, ISBN 1-4244-0054-6

[66] Ng H.: Automatic Thresholding for Defect Detection, Proceedings of the 3rd International Conference on Image and Graphics, Elsevier Science Inc. New York, NY, USA, 2004, p.1644-1649, ISBN 0-7695-2244-0/04

[67] Solihin Y., Leedham C.: Integral Ratio: A New Class of Global Thresholding Techniques for Handwriting Images, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 21, No. 8, August 1999, IEEE Computer Society, p. 761-768, ISBN 0162-8828/99

[68] Cavadini M., Wosnitza M., Thaler M., Troster G.: A VLSI Architecture for Real Time Object Detection on High Resolution Images, Proceedings of the 8th European Signal Processing Conference, 1996, Trieste, Italy

[69] Gupta A., Mukerjee A.: Computational Models for Object Detection and Recognition, Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, 2004

[70] Aguilar-Ponce R., Tessier J., Emmela C., Baker A., Das J.: Real-Time VLSI Architecture for Detection of Moving Object Using Wronskian Determinant, 48th Midwest Symposium on Circuits and Systems, Vol. 1, 2005, p. 875- 878, ISBN 0-7803-9197-7

[71] Ikeda M., Kondo T., Nitta K., Suguri K., Yoshitome T., Minami T., Iwasaki H., Ochiai K., Naganuma J., Endo M., Tashiro Y., Watanabe H., Kobayashi N., Okubo T., Ogura T., Kasai R.: SuperEnc: Mpeg-2 Video Encoder Chip, IEEE Micro, Jul-Aug 1999, p. 56-65, ISSN: 0272-1732

[72] Mariano V., Min J., Park J., Kasturi R., Mihalcik D., Li H., Doermann D., Drayer T.: Performance Evaluation of Object Detection Algorithms, Proceedings of the 16th International Conference on Pattern Recognition, IEEE Computer Society, 2002, ISBN 1051-4651/02

[73] Vega-Rodrigues M., Sanchez-Perez J., Gomez-Pulido J.: An Fpga-Based Implementation for Median Filter Meeting the Real-Time Requirements of Automated Visual Inspection Systems, Proceedings of the 10th Mediterranean Conference on Control and Automation, 2002, Lisbon, Portugal, ISBN 962-442-228-1

[74] Breveglieri L., Piuri V.: Digital Median Filters, Journal of VLSI Signal Processing 31, p. 191206, 2002, ISSN 0922-5773

[75] Maheshwari R., Rao S., Poonacha P.: FPGA Implementation of Median Filter, 10th International Conference on VLSI Design, IEEE Computer Society, 1997, p. 523-524, ISBN O-8186-7755-4/96

[76] Suorania R., Estola K.: New Class of Order Statistic Filters for Running Median Estimation, IEEE International Conference on Acoustics, Speech, and Signal Processing, Volume 3, 1993, p. 27-30, ISBN 0-7803-0946-4/9

[77] Eng H., Ma K.: Noise Adaptive Soft-Switching Median Filter for Image Denoising, IEEE International Conference on Acoustics, Speech, and Signal Processing, Volume 6, IEEE Computer Society, 2000, p. 2175-2178, ISBN 0-7803-6293-4

[78] Sun Z., Miller R., Bebis G., DiMeo D.: A Real-time Precrash Vehicle Detection System, Proceedings of the 6th IEEE Workshop on Applications of Computer Vision, IEEE Compuer Society, 2002, p. 171-177, ISBN 0-7695-1858-3/02

[79] Matsushita N., Hihara D., Ushiro T., Yoshimura S., Rekimoto J., Yamomoto Y.: ID CAM: A Smart Camera for Scene Capturing and ID Recognition, Proceedings of the 2nd IEEE and ACM International Symposium on Mixed and Augmented Reality, IEEE Computer Society, 2003, p. 227-237, ISBN 0-7695-2006-5/03

[80] `http://www.celoxica.com/technology/c\_design/handel-c.asp`

[81] `http://www.celoxica.com/technology/c\_design/systemc.asp`

[82] Draper B., Najjar W., Bohm W., Hammes J., Rinker B., Ross C., Chawathe M., Bins J.: Compiling and Optimizing Image Processing Algorithms for FPGAs, Proceedings of the 5th IEEE International Workshop on Computer Architectures for Machine Perception, IEEE Computer Society, 2000, p. 222-232, ISBN 0-7695-0740-9/00

[83] Palenichka R., Zinterhof R., Ivasenko I.: Adaptive Image Filtering and Segmentation Using Robust Estimation of Intensity, Proceedings of the Advances in Pattern Recognition, Joint IAPR International Workshops, Springer-Verlag, London, UK, 2000, p. 888-898, ISSN 0302-9743

[84] Srivastava N., Trahan J., Vaidyanathan R, Rai S.: Adaptive Image Filtering Using Run-Time Reconfiguration, Proceedings of the International Parallel and Distributed Processing Symposium, IEEE Computer Society, 2003, p. 180-187, ISBN 0-7695-1926-1/03

[85] Torres-Huitzil C., Arias-Estrada M.: An FPGA Architecture for High Speed Edge and Corner Detection, Proceedings of the 5th IEEE International Workshop on Computer Architectures for Machine Perception, IEEE Computer Society, 2000, p. 112-117, ISBN 0-7695-0740-9/00

[86] Shin M., Goldgof D., Bowyer K.: Comparison of Edge Detectors Using an Object Recognition Task, Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition 1999, p. 1360-1366

[87] ASIC Design Tutorial, `http://www.tutorial-reports.com/hardware/asic/`

[88] Spartan-3E FPGA Family Data Sheet, `http://direct.xilinx.com/bvdocs/publications/ds312.pdf`

[89] Ratha N., Jain A.: FPGA-based Computing in Computer Vision, Proceedings of the 1997 Computer Architectures for Machine Perception, 1997, ISBN 0-8186-7987-5/97

[90] Rigoll G., Kosmala A.: New Improved Feature Extraction Methods for Real-Time High Performance Image Sequence Recognition, Proceedings of the 1997 IEEE International Conference on Acoustics, Speech, and Signal Processing, IEEE Computer Society, 1997, p. 2901-2905, ISBN 0-8186-7919-0/97

[91] Scalera J., Jones III C., Soni M., Bucciero M., Athanas P., Abbott A., Mishra A.: Reconfigurable Object Detection in FLIR Image Sequences, Proceedings of the 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2002, p. 284-286, ISBN 1082-3409/02

[92] Young S., Alfke P., Fewer C., McMillan S., Blodget B., Levi D.: A High I/O Reconfigurable Crossbar Switch, Proceedings of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, IEEE Computer Society, 2003, ISBN 0-7695-1979-2

[93] Hübner M., Becker J.: Seamless Design Flow for Run-Time Reconfigurable Automotive Systems, DATE Workshop on Future Trends in Automotive Electronics and Tool Integration, 2006, p. 47-51

[94] Donlin A.: Applications, Design Tools and Low Power Issues in FPGA Reconfiguration, in Designing Embedded Processors, Springer Netherlands, 2007, p. 513-541, ISBN 978-1-4020-5868-4

[95] Jan S., Clapworthy G., Rooze M.: Morphology-Based Data Elimination from Medical Image Data, IEEE Computer Graphics and Application Journal, March/April 2000, p. 46-52, ISBN 0272-1716/00

[96] Yamamoto S., Matsumoto M., Tateno Y., Iinuma T., Matsumoto T.: Quoit Filter - a New Filter Based on Mathematical Morphology to Extract the Isolated Shadow, and Its Application to Automatic Detection of Lung Cancer in X-ray CT, Proceedings of the 1996 International Conference on Pattern Recognition, IEEE Computer Society, 1996, p. 3-8, ISBN 1051-4651/96

[97] Gu L., Kaneko T., Tanaka N., Haralick R.: Robust Extraction of Characters from Color Scene Image Using Mathematical morphology, Proceedings of Fourteenth International Conference on Pattern Recognition, 1998, p. 1002-1004, ISBN 0-8186-8512-3-2

[98] Barata T., Pina P., Granado I.: Segmenting at Higher Scales to Classify at Lower Scales. A Mathematical Morphology Based Methodology Applied to Forest Cover Remote Sensing Images, Proceedings of the International Conference on Pattern Recognition 2000, ISBN 1051-4651/00

[99] Haralick R., Sternberg S., Zhuang X.: Image Analysis Using Mathematical Morphology, IEEE Transactions on Pattern Analysis and Machine Intelligence, 1987, p. 532-550, ISSN 0162-8828

[100] Bianchini M., Maggini M., Sarti L, Scarselli F.: Recursive neural networks for object detection, Special issue on neural networks and kernel methods for structured domains, 2005, p. 1040-1050, ISSN 0893-6080

[101] Nakano T., Morie T., Iwata A.: A Face/Object Recognition System Using FPGA Implementation of Coarse Region Segmentation, SICE Annual Conference in Fukui, Fukui University, Japan, 2003, p. 1552-1557

[102] Villasenor J., Schoner B., Chia K., Zapata C., Kim H., Jones C., Lansing S., Mangione-Smith B.: Configurable Computing Solutions for Automatic Target Recognition, Proceedings of the 36th ACM/IEEE Conference on Design Automation, 1999, p. 697-702, ISBN 1-58133-109-7

[103] Yu H., Leeser M.: Automatic Sliding Window Operation Optimization for FPGA-Based Computing Boards, 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2006, p. 76-88, ISBN 0-7695-2661-6/06

[104] InetDaemon.com: Classification of Computer Systems, `http://www.inetdaemon.com/tutorials/computers/types.shtml`

[105] Porter R., Frigo J., Gokhale M., Wolinski C., Charot W., Wagner C.: A Programmable, Maximal Throughput Architecture for Neighborhood Image Processing, 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2006, p. 279-280, ISBN 0-7695-2661-6/06

[106] Hezel S., Kugel A., Manner R., Gavrila D: FPGA-based Template Matching using Distance Transforms, Proceedings of the 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2002, p. 89-98, ISBN 1082-3409/02

[107] Fan J., Elmagarmid A.: Statistical Approaches to Tracking-Based Moving Object Extraction, Proceedings of the 1999 International Conference on Information Intelligence and Systems, IEEE Computer Society, Washington DC, USA, p. 375-381, ISBN O-7695-0446-9/99

[108] Qiang L., Bo Z.: Template Matching Based on Image Gray Value, Visual Communications and Image Processing, Proceedings of the SPIE, Volume 5960, 2005, p. 614-622, 2005SPIE.5960..614L

[109] Leibson S.: Challenges in Consumer Electronics for 21st Century, Keynote lecture, Worldcomp 2007, Las Vegas, NV, USA, June 2007, `http://www.world-academy-of-science.org/worldcomp07/ws/keynotes/keynote_leibson`

[110] Rätsch G.: Adaboost, a tutorial for Machine Learning Summer Schools 2003, Canberra, Australia, February 2-14, `http://informatik.unibas.ch/lehre/ws05/cs232/_Folien/08_AdaBoost.pdf`

[111] Govindu G., Zhuo L., Choi S., Gundala P., Prasanna V.: Area and power performance analysis of a floating-point based application on FPGAs, Proceedings of the 7th Annual Workshop on High Performance Embedded Computing, 2003

[112] Freund Y., Schapire R: A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences, August 1997, p. 119-139, ISSN 0022-0000

[113] Freund Y., Schapire R.: A Short Introduction to Boosting, Journal of Japanese Society for Artificial Intelligence, September 1999, p.771-780, ISSN 0912-8085

[114] Zemčík P., Herout A., Beran V., Granat J.: Hardware Accelerated Image Analysis in FPGA, Proceedings of SCCG, Casta-Papiernicka, SK, 2006

[115] Flynn M.: Some Computer Organizations and their Effectiveness, IEEE Transactions on Computers 24, September 1972, IEEE Computer Society, p. 948-960, ISSN 0018-9340

[116] Sakanashi H., Iwata M., Higuchi T.: A Lossless Compression Method for Halftone Images using Evolvable Hardware, Proceedings of the 4th International Conference on Evolvable Systems: From Biology to Hardware, 2001, p. 314-326, ISBN 3-540-42671-X

[117] Salami M., Murakawa M., Higuchi T.: Data Compression Based on Evolvable Hardware, Proceedings of the 1st International Conference on Evolvable Systems: From Biology to Hardware, 1996, p. 169-179, ISBN 3-540-63173-9

[118] Tanaka M., Sakanashi H., Salami M., Iwata M., Kurita T., Higuchi T.: Data Compression for Digital Color Electrophotographic Printer with Evolvable Hardware, Proceedings of the 2nd International Conference Evolvable Systems: From Biology to Hardware, Springer, 1998, p. 106-114, ISBN 3-540-64954-9

[119] Sekanina L.: Evolvable Components, From Theory to Hardware Implementations, Springer-Verlag, Berlin, Germany, 2004, 194 p., ISBN 3-540-40377-9

[120] Martinek T., Sekanina L.: An Evolvable Image Filter: Experimental Evaluation of a Complete Hardware Implementation in FPGA, Lecture Notes in Computer Science, no. 3637, 2005, Germany, p. 76-85, ISSN 0302-9743

[121] Iwata M., Kajitani I., Yamada H., Iba H., Higuchi T.: A Pattern Recognition System Using Evolvable Hardware, Proceedings of the 4th International Conference on Parallel Problem Solving from Nature, Springer-Verlag, London, UK, 1996, p. 761-770, ISBN 3-540-61723-X

[122] Torresen J., Bakke J., Sekanina L.: Recognizing Speed Limit Sign Numbers by Evolvable Hardware, Lecture Notes in Computer Science, No. 3242, 2004, p. 682-691, ISSN 0302-9743

List of Mathematical Symbols and Abbreviations

Symbol	Meaning
$abs(x)$	Absolute value of number x
$\|s\|$	Number of elements in set s
$s.t.$	So that
$\mathcal{G}_{w,h}^{d}$	Set of all images of size $w \times h$ and gray depth d
$\mathcal{G}_{w,h}$	Set of all images of size $w \times h$
$\mathcal{G}$	Set of all images
$\mathcal{N}^{S}$	Set of all neighborhoods with structuring element S
$\mathcal{N}$	Set of all neighborhoods
$\mathcal{F}_{S}$	Set of all filters of structuring element S
$\mathcal{T}_{S}$	Set of all templates for structuring elements S
$\mathcal{B}_{S}$	Set of all template banks for structuring elements S

Table 11.1: Mathematical symbols and definitions

Abbreviation	Meaning	Explanation
PLD	Programmable Logic Device	2.1
FPGA	Field Programmable Gate Array	2.3
CPLD	Complex Programmable Logic Devices	2.1
ASIC	Application Specific Integrated Circuit	2.1
DSP	Digital Signal Processing	
DSP	Digital Signal Processor	
VLIW	Very Large Instruction Word	
VLSI	Very Large Scale Integration	
VHDL	VHSIC Hardware Description Language	2.3.1
CPU	Central Processing Unit	
CLB	Configurable Logic Block	2.3
IOB	Input Output Buffer	2.3
RAM	Random Access Memory	
BRAM	Block RAM	2.3
SRAM	Static RAM	
I/O	Input / Output	
PDA	Personal Digital Assistent	
CAD	Computer Aided Design	2.3.1
ECU	Electronic Control Unit	2.4.4
FE	Feature Extraction	3.3.2
JPEG	Joint Photographic Experts Group	
2-D	2 Dimensional	
SIMD	Single Instruction Multiple Data	
Pixel	Picture Element	3.1
PC	Personal Computer	
FIFO	First In First Out	
LSB	Least Significant Bit	
MSB	Most Significant Bit	
S2M	Serial to Matrix	6.4
LUT	Look Up Table	6.7.2
HDTV	High Definition Television (1280 x 720)	
Occ	Occurrence, occur	
Temp	Temporary	
Arr	Array	
Num	Number	

Table 11.2: Abbreviations

Appendix A

Evaluation Program

To perform evaluations in Chapters 8 and 9, a C program has been programmed that simulates the proposed method's function. Following are the command line options and examples of use.

parameter	values	initial	meaning
-p	string	../_t1/	input file path
-f	string	001_glob	input file name
-FilterOnly	string	none	Perform only image filtering[1]
-Evolv	0 / 1	1	Bank creation (1) or object detection (0)
-CompX	int	5	Template size X
-CompY	int	5	Template size Y
-Iter	int	20	Number of iterations for the bank creation
-Elem	int	300	Number of elements in the bank
-Interactive	0 / 1	1	Shows the progress
-SPZFindOnly	0 / 1	0	Perform only classification operation
-BankUpdate	0 / 1	1	Consider also templates in existing bank
-LocalBank	0 / 1	0	Use a local bank
-MergeBank	0 / 1	0	Merge two banks together
-EmptyBank	0 / 1	0	Create an empty bank
-SegmOnly	0 / 1	0	Perform the preprocessing only
-WithoutSegm	0 / 1	0	Omit preprocessing
-WithoutEdge	0 / 1	0	Omit edge detection in preprocessing
-BW	0 / 1	0	Output image in BW suitable for printing
-Separator	int	-Elem / 2	How many templates are active
-MaxSPZCompOut	int	200	Max. number of templates outside an object
-MinAccuracy	int	5	Threshold of the object detection
-MinSPZDistance	int	0	Minimal distance between located objects
-NumSPZs	int	139	Maximum number of objects
-CompInfluence	real	1.8	Influence of number of different templates

Table A.1: C program parameters

Implicit bank name is _bank.bmp. In case a local bank is used (-LocalBank), bank

[1]Filter can be one of the following: none, FlatAverage, SampledGaussian, HighPass, Erosion, Dilation, Median, OpenClose, HitMiss, BinMedian, BinAvg, BinMinMax

name is _bank[FileName].bmp. Coordinates of the license plate in an image (if we want to create a template bank) have to be stored in a file SPZPos.txt in format filename x y.

To create a template bank _bank.bmp from image 0000.bmp:

```
a.exe -f 0000 -BankUpdate 0
```

To detect an object in image 0000.bmp using a bank in _bank.bmp:

```
a.exe -f 0000 -Evolv 0
```

The source codes of the C program can be found at
`http://www.fit.vutbr.cz/~crha/phd/case_study.cpp` and
`http://www.fit.vutbr.cz/~crha/phd/case_study.h`
The example of the image of road signs used in Chapter 9 can be found at
`http://www.fit.vutbr.cz/~crha/phd/signs.bmp`
and an example of a vehicle from Chapter 8 at
`http://www.fit.vutbr.cz/~crha/phd/0000.bmp`