

# Mozgókamerás járműdetekció sokprocesszoros architektúrák és multifoveális adatfeldolgozás használatával



Szerző: PÁLFFY András BSc IV. évfolyam

Konzulens: HORVÁTH András adjunktus

Információs Technológiai és Bionikai Kar

Pázmány Péter Katolikus Egyetem

*OTDK dolgozat*

2014 Május

# Contents

|                                                                      |           |
|----------------------------------------------------------------------|-----------|
| <b>List of Figures</b>                                               | <b>v</b>  |
| <b>1 Introduction</b>                                                | <b>1</b>  |
| 1.1 Image processing . . . . .                                       | 1         |
| 1.2 ADAS systems . . . . .                                           | 2         |
| 1.3 Embedded systems . . . . .                                       | 2         |
| 1.4 Task description . . . . .                                       | 3         |
| <b>2 Existing solutions</b>                                          | <b>4</b>  |
| 2.1 Non-image processing alternatives . . . . .                      | 4         |
| 2.2 Existing solutions based on image processing . . . . .           | 4         |
| 2.2.1 Foreground-background segmentation . . . . .                   | 4         |
| 2.2.2 Algorithms for moving platform . . . . .                       | 5         |
| <b>3 Development circumstances</b>                                   | <b>8</b>  |
| 3.1 Eutecus Inc. . . . .                                             | 8         |
| 3.2 VirVa Lab . . . . .                                              | 8         |
| 3.3 Introducing the Hardware . . . . .                               | 9         |
| 3.4 The foveal image processing method . . . . .                     | 10        |
| 3.4.1 Comparison of full-frame and foveal image processing . . . . . | 10        |
| 3.4.2 Preprocessing . . . . .                                        | 12        |
| 3.4.3 Fovea placer . . . . .                                         | 12        |
| 3.4.4 Foveal algorithm . . . . .                                     | 13        |
| 3.5 Implemented set of functions . . . . .                           | 13        |
| 3.6 Process of algorithm development . . . . .                       | 15        |
| 3.6.1 Matlab simulation . . . . .                                    | 15        |
| 3.6.2 C++ implementation . . . . .                                   | 15        |
| 3.6.3 Hardware implementation . . . . .                              | 15        |
| <b>4 Designing and implementing the algorithm</b>                    | <b>16</b> |
| 4.1 Challenges in the task . . . . .                                 | 16        |
| 4.2 Constraints and presumptions . . . . .                           | 17        |

|          |                                        |           |
|----------|----------------------------------------|-----------|
| 4.3      | Description of the algorithm . . . . . | 17        |
| <b>5</b> | <b>Results</b>                         | <b>23</b> |
| 5.1      | Test videos . . . . .                  | 23        |
| 5.2      | Distance approximation . . . . .       | 24        |
| 5.3      | License-plate detection . . . . .      | 24        |
| 5.4      | Image dataset . . . . .                | 25        |
| <b>6</b> | <b>Conclusion</b>                      | <b>31</b> |
|          | <b>References</b>                      | <b>34</b> |

# List of Figures

|     |                                                                       |    |
|-----|-----------------------------------------------------------------------|----|
| 2.1 | Background-foreground segmentation example . . . . .                  | 5  |
| 2.2 | Shadow based detection example . . . . .                              | 7  |
| 2.3 | Haar-like features, example 1 . . . . .                               | 7  |
| 2.4 | Haar-like features, example 2 . . . . .                               | 7  |
| 3.1 | Introducing Bi-i smart Cubes . . . . .                                | 9  |
| 3.2 | Functional block-diagram of the system . . . . .                      | 10 |
| 3.3 | VGA vs FullHD resolution . . . . .                                    | 11 |
| 3.4 | Resolution scale up . . . . .                                         | 12 |
| 3.5 | Illustration of the software framework . . . . .                      | 14 |
| 4.1 | Chosen input images to illustrate the algorithm . . . . .             | 18 |
| 4.2 | Examples of Cr channels . . . . .                                     | 18 |
| 4.3 | Results of the colorfilters . . . . .                                 | 19 |
| 4.4 | Cropping the horizontal lane of the lamps . . . . .                   | 20 |
| 4.5 | Vertical projection of the red lamps . . . . .                        | 20 |
| 4.6 | Edge-map for license plate detection . . . . .                        | 21 |
| 4.7 | Alternative detection method based on wheels . . . . .                | 22 |
| 5.2 | Example frame from the bird's-eye view . . . . .                      | 24 |
| 5.1 | Example images of detected vehicles . . . . .                         | 28 |
| 5.3 | Example frame from the bird's-eye view with reference image . . . . . | 29 |
| 5.4 | License plate detection examples . . . . .                            | 29 |
| 5.5 | Samples from the false-negative dataset . . . . .                     | 30 |
| 5.6 | Samples from the false-positive dataset . . . . .                     | 30 |

## Abstract

Konzulensek:

- Dr. Horváth András - *Pázmány Péter Katolikus Egyetem, Információs Technológiai és Bionikai kar*
- Dr. Rekeczky Csaba - *Eutecus Inc., Berkeley, CA*

Az általunk érzékelt ingerek jelentős százaléka (több, mint 90) vizuális információ. Logikus tehát, hogy a mesterséges intelligencia kutatásában nagy jelentőséget tulajdonítunk a gépi látásnak. Az is fontos szempont, hogy a képfeldolgozás jelentősen olcsóbb más szenzoros megoldásokkal szemben figyelembe véve a kinyerhető információ mennyiségét. Napjainkban a kamerák képesek nagy tér és időbeli felbontással adatokat rögzíteni, azonban a központi kérdés még mindig az adat feldolgozása. A nyers adat semmivel sem ér többet, mint egy véletlenszerű bitfolyam: a lényeg az ebből kinyert információ.

Számos kritikus esetben szükség van az információ valós idejű s azonnali feldolgozására.

Néhány gyors folyamatban, mint amilyen például a közlekedés, fontos, hogy a másodperc törtsze alatt tudjunk adott eseményeket detektálni. Egy ilyen detekció, még, ha nem is 100%-ig pontos, többet ér, ha időben működik az esetek nagy százalékában, mintha két másodperccel később ismer fel minden eseményt pontosan.

Alacsony fogyasztású, sokprocesszoros eszközökkel lehetőség nyílik óriási adatok gyors feldolgozására, olyan módon, hogy közben a fogyasztás alacsony marad, így ez akár beágyazott rendszerként működhet egy autóban.

Azonban ezen architektúrákra teljesen más szemléletre és algoritmusokra van szükség, melyek képesek csupán a lokális információ alapján döntéseket hozni, objektumokat detektálni, anélkül hogy a kép minden pixelét feldolgozzák.

Dolgozatomban egy ilyen algoritmus tervezésének lépéseit, s a módszer alkalmazhatóságát vizsgálom egy olyan összetett probléma esetén, mint a mozgókamerás járműdetekció. Az elkészült algoritmust egy több mint 400 pozitív és több mint 1000 negatív mintát tartalmazó, általam készített kép adatbázison teszteltem, amin több mint 89%-os találati arányt ért el.

## Abstract

Most of the impulses (more than 90 %) received by humans are visual. It is rational that computer vision is very important in the science of artificial intelligence. Nowadays, cameras can record videos with high temporal and spatial resolution. Processing this data is a much more complicated task. Still, the recorded data is worthless without proper processing tools to gain the information from it.

In many cases, a solution is even harder to give since it is crucial to process the input and make decisions real-time. For example, in special areas like the transportation it is highly important to detect events extremely fast. Such a detection should be rather make mistakes (and have an accuracy below 100%) than perform without a single error but a two second delay.

With low-power, multi-processor devices it is possible to process huge amount of data in very short time. These units can perform as embedded systems. However, developing for these architectures requests a completely different algorithm designs.

In this paper a foveal algorithm was defined for the task of the real-time vehicle detection on the input video stream of an on-board camera. After defining the possible positions of the lamps and the license plates, the algorithm is able to decide and detect different sized, coloured and shaped cars from a clear rear view.

Our software was tested on numerous test videos containing scenes with two or more visible cars, and we got impressive results both in detecting and tracking. A hit accuracy of 89% was reached on an image dataset containing more than 400 positive and 1000 negative input examples.

# Chapter 1

## Introduction

### 1.1 Image processing

Image processing is a special case of signal processing, where the input is an image (a photo, a video frame, etc.) and the output is either also an image, or some extracted information for later usage. In less specific words, (high-level) image processing (also called: computer vision) is the science of gaining and using information from an image, video frame, or any visual input.

Computer vision technology is gaining foothold in more and more areas in our life. Some examples without a claim of completeness: traffic monitoring, security surveillance, medical assistance systems, unmanned aerial vehicles, quality control, animal tracking, and so on. Computers use sensory input to make decisions and answer questions like "How many cars passed today" or "Should the alarm be triggered by this intrusion", etc.

It is fair to ask why should computers do these tasks when they are really easy to accomplish by a person. Although it is true in most cases, it would be either extremely boring or exhaustive to watch a monitor all day, and log the events. Furthermore, some functions are so complicated or fast, that even if a human is capable of doing it, the result would be too late to be processed. Computers make mistakes too (a lot), but they perform much better in certain tasks, like fast, real-time approximations.

As we discussed the advantages and disadvantages of computer vision, it is clear that it can't fully replace human control in the near future. However, there are many areas where rather a human-machine cooperation would be more useful than an autonomous surveillance system. In these fields typically the monotonous routines can be easily substituted by a computer vision system with the close or not so close supervision of the human operator. Such a task could be a space observation project, automatic scanning for surveys, or ADAS systems.

## 1.2 ADAS systems

The word ADAS is an acronym that stands for **A**dvanced **D**river **A**ssistance **S**ystems. Such a system is able to sense the environment and the parameters of a car, truck or other vehicle it is installed in and after processing the data it is able to give advices, notifications, or warnings to the driver, or -in case of an emergency,- intervene either by breaking, steering, accelerating, or the combination of these. In the unfortunate case when the accident is unavoidable, some systems can execute pre-crash steps like tensioning the seatbelts, readjusting the seats, closing the windows, and decreasing the speed to minimise personal injuries. The most important parts of these very complex tasks are the detection and decision making. In this thesis we will only discuss those software and hardware solutions which are dedicated to perform these subtasks.

It should be stated that although the challenges and methods are similar, driver assistance systems do not aim to reach fully autonomous driving. Instead, their main purpose is to help and protect the driver and the passengers while making the travel as comfortable as possible.

There are many various type of ADAS, but basically, we can classify them into two groups along the aims given above. The ones in the first are planned to comfort the driver. For example, autonomous cruise control, traffic sign detection, or automatic parking could be mentioned. The other group is more important in many aspects since its members were designed to protect human life, and if possible, the vehicles, too. The most common ones are lane departure warning systems, driver drowsiness detections, blind spot detections, and collision avoidance systems.

Systems like these are developed by many leading companies. Currently they are rather extra functions for high-end cars. It is believed that in the close future, they will spread very fast, and reach cheaper models too.

For reasonably priced, real-time and reliable detection solutions, embedded systems can provide a good platform.

## 1.3 Embedded systems

An embedded system is a microprocessor-based system that is built and designed to perform one or a very few particular task([1]). That means the hardware and the software implemented on it can be optimized very effectively for performance. This is why embedded systems are frequently used in problems with real-time computing constraints . These systems are usually designed to be able to fit in a bigger system, like household appliances or -in our case- cars. Thus, size and power-consumption are crucial. Developing for such a hardware could be really hard, since the processing power of the architecture is limited. To avoid system overload, traditional algorithms and methods cannot always be applied. For example, in image processing, dilation and erosion are well-known and widely used processes. However, they are almost impossible to be implemented on an embedded system caused by their high complexity. During the development, limitations of the system must be respected. A full HD image contains

approximately 2 millions pixels. To apply a small,  $3 * 3$  kernel on it (to perform edge detection for example), more than 19 million operation are required. That is too much to calculate in real-time on an embedded system. Thus, simple and low-complexity functions are welcomed.

### 1.4 Task description

In this paper, we will present a foveal algorithm with the main purpose of detecting and tracking cars on the input image when the camera is mounted on a moving vehicle.

This algorithm will provide the basic information source for a collision prevention system. Thus, the algorithm should be able to run on designated hardware in real-time with reliable accuracy. In these applications real-time processing plays a crucial role, even more important than flawless detection. Such an algorithm should rather find a lower percentage of the cars in real time than identifying all of them, but with a few seconds of delay. Needless to say that such late information would be useless in this task, since a possible collision could have already happened.

We decided to design the algorithm to detect cars only. This is a rational decision because recognizing trucks, buses, and other ground-vehicles would mean a completely different task. Only cars heading away from the camera will be considered since the primary application cases will be parallel travelling scenarios, like highways and main roads. Our future plans include detection of incoming traffic and larger set of vehicles.

The algorithm is designed to be implemented on an embedded system (1.3) since the beginning. Every benefits and disadvantages of these systems had to be considered and kept in mind.

It is important to note that just like no other ADAS solutions, neither this algorithm can exceed the abilities of an average driver. However, it is not supposed to do so. A relaxed, well-prepared driver should be able to detect all cars that this algorithm can. Despite of this fact, accidents happen because drivers become tired and make mistakes. This is the main reason why collision prevention systems like this are necessary and should be developed.

## Chapter 2

# Existing solutions

### 2.1 Non-image processing alternatives

First, it should be mentioned that systems with the same or similar purpose (collision avoidance, automatic following distance control, etc.) based on other sensors than cameras (like radars or laser sensors) already exist. With simple and fast calculations, relative speed can be approximated. These systems are able to measure distances with much higher accuracy than any image processing method. On the other hand, the only information they supply is the distance of the detected obstacle. Recognizing the object ahead offers the advantage that presumptions can be made about its future behaviour. For example, in case of an approaching vehicle, a human driver will assume that it will continue its movement with similar speed and direction. However, if the observed obstacle is a pedestrian, these presumptions will be completely different. These classifications cannot be done with distance measurement sensors only, and this is why on-board cameras are necessary.

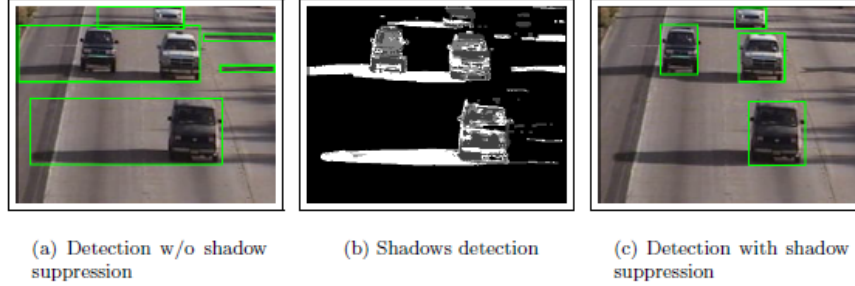
We suppose that an ideal solution would combine distance measuring and image processing methods into one system to merge as much information as possible, and make the decisions based on the fused data. Systems described here are already available at some leading companies. However, these are closed systems with limited capabilities. Implementing such a solution using the algorithm introduced in this thesis is currently in progress.

### 2.2 Existing solutions based on image processing

#### 2.2.1 Foreground-background segmentation

Foreground-background segmentation methods are basically image based movement detections. If a given project declares a task that requires a camera in a fixed position, it is a reasonable decision to choose a detection method based on foreground-background segmentation. Such an algorithm is able to learn the "background" of an image with differentiating a few frames. Should a pixel show none or very little change in its

## 2.2 Existing solutions based on image processing



**Figure 2.1:** Background-foreground segmentation. The first image is the input with the foreground objects bounded. After shadows were detected (the second image) the bounding boxes are tightened, as shown on the third image. Source: [2]

neighbourhood it will be considered as background. This learning could be performed off-line (before the detection progresses), or on-line. In the latter case, the separation is adaptive, as the learning can be redone at given times.

These solutions could give an excellent base for tasks like traffic monitoring, classification, or passing car counting, as moving points are separated, and connected ones could be managed as an object. Thus, all possible vehicles are very well separated. The fact that the camera does not move has even more advantages. For example, perspectives are fixed and can be calibrated off-line, so approximating sizes and distances can be rather accurate. With fixed lengths, reliable velocity measuring can be implemented. Also, the regions where the desired objects may appear can be selected and marked very precisely. Based on these additional information (size, position, and speed) the decision itself should be quite relatively easy. On the other hand, we must mention that both the learning method and the change detection require full-frame processing and difference calculation as every pixel needs to be checked in every frame. This could be quite slow and resource-intensive. We should also mention that not every moving path is an object, as in certain lighting circumstances a vehicle and its shadow move almost identically and have similar size parameters. Effective methods to remove shadows from input images have been described [2]. Certainly, process time of these filters sum up with the detection's time. Notice that although shadow is considered as a noise in these algorithm, it can be really useful in different approaches, where the shadows below the cars are the main features the algorithm looks for. (2.2.2).

Although foreground-background segmentation has many benefits, clearly for the current task it is not possible to use it because our solution is desired to work on moving a platform.

### 2.2.2 Algorithms for moving platform

As discussed in subsection 2.2.1, without a background separation, it is significantly more difficult to segment an object. However, solutions for moving platform image acquisition system exist, usually with similar approaches. All solutions cited in this

## 2.2 Existing solutions based on image processing

---

paper are full-frame processing systems, thus it is rather complicated to compare them with our solution.

A very popular idea is to segment the road in front of the car detecting shadow-like features ([3]). After determining the shadow point, regions above them are selected as possible vehicles. An example is shown on figure 2.2. In the cited article, further features are inspected, like vertical edges, and symmetry. Although vehicles tend to be symmetrical from behind, this property is easily lost with a slightly smaller view angle. Thus, we could not base our algorithm on symmetry. Segmentation is very resource-intensive, and the presumption that the car is above its shadow is not always correct. Difficult scenarios, -when more than one light source is available, or two shadows are hard to separate- are not discussed in the paper. Thus, this solution could be exceeded.

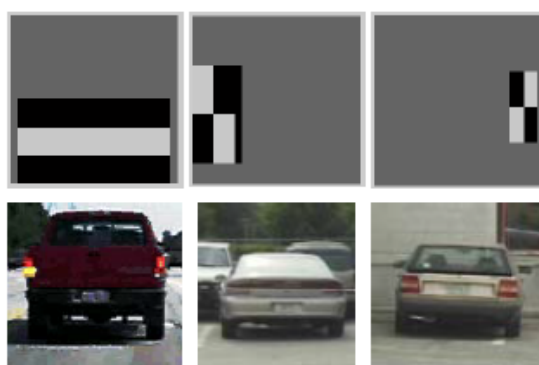
Another approach that is used quite often is to use Haar-like features([4],[5]). Inspired by the well known computer vision problem of detecting faces, these algorithms slide windows over the picture and try to match the predefined patterns. Some example images of these patterns are shown on figure 2.3 and 2.4.

## 2.2 Existing solutions based on image processing

---



**Figure 2.2:** Road segmentation by shadow-like features. After shadows are found on the input images (left) the areas above them are processed (right). Source: [3]



**Figure 2.3:** In this figure one can see different example images (second row) and the matching pattern for them in the first row. Source: [3]



**Figure 2.4:** Examples for the patterns implemented by Haar-like features. The first pattern validates the existence of a shadow below the car while the other two match the wheels. Source: [2]

## Chapter 3

# Development circumstances

*The system, figures and technological details introduced in this chapter are proprietary to EUTECUS, INC.*

### 3.1 Eutecus Inc.

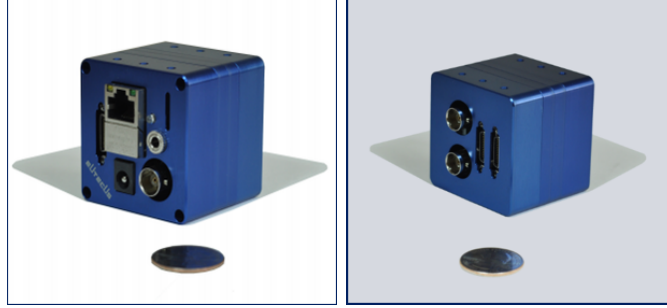
The Eutecus, Inc. is located in Berkeley, CA founded in 2003. A Hungarian subsidiary in Budapest, named the Eutecus EU/HU Ltd. exists since 2008. They create, promote and commercialize leading edge vision and video analytics systems using Cellular Visual Technology<sup>TM</sup>. Their multifunctional solution for Video Analytics is the so called MVE<sup>TM</sup>, Multi-core Video Analytics Engine. That is a set of embedded video analytics tools implemented on FPGAs, ASSPs and ASICs[6]. All the algorithms are concurrently designed, and massively parallel, optimized for multi/many-core processor architectures used in embedded applications and products. Their primary business model, is to provide intellectual property working closely with FPGA providers. Eutecus also could provide customization, integration, technical support and training services.

### 3.2 VirVa Lab

The acronym VirVa Lab stands for Virtual Video Analytics Laboratory. It is a research laboratory that connects Eutecus US-EU/HU (The Company) and Pázmány Péter Catholic University from multiple locations literally around the globe.

The aim of this cooperation is to provide an opportunity for motivated students to take part in algorithmic research and learn about the hardware-firmware-software tools used by the Company. Also, innovations produced in the lab are up-to-date, and planned to be implemented and streamlined into marketed products in 6 months.

Although several different projects take place in the VirVa lab, regular meetings are hold to discuss requests, and feedbacks to help cooperation. All algorithms are being developed in the foveal system introduced in 3.4. Current ADAS orientated projects are:

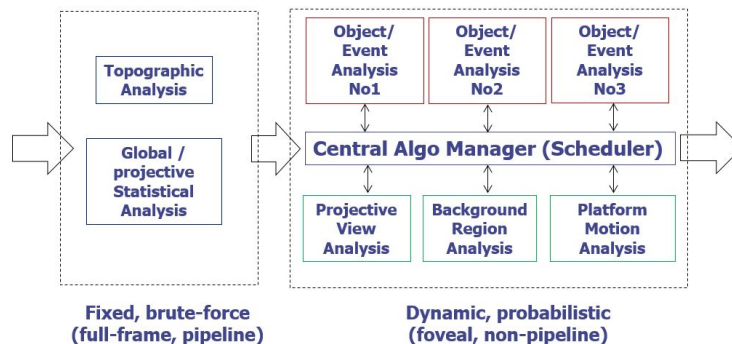


**Figure 3.1:** Sample images of the Bi-i Smart Cube Family. Notice the modularity of the design and the variety of inputs. (Proprietary to EUTECUS, INC.)

- **information and warning traffic sign detection:** These two algorithms aim to detect standard traffic signs located along the road.
- **lane departure alerting system:** This solution uses local measurements only to detect left, right and middle road lines. Fused results of these will be used in perspective approximation, which is very important in every image processing tasks, especially in the case of moving platform.
- **driver drowsiness detection:** Our only project that is based on an interior camera is able to detect and track heads in the cabin.
- **collision prevention based on car detection:** Finally, the algorithm discussed in this paper is also developed in the VirVa Lab.

### 3.3 Introducing the Hardware

Eutecus Inc. has developed a hardware family named Bi-i Smart Cubes(See figure 3.1 for example images). Members of this are Video Analytics hardware modules based on Altera’s Cyclone IV GX 50-75 FPGA Platform. Units have a quite compact size ( $\sim 5cm \times \sim 5cm \times \sim 5cm$ ) and are designed to accept 3G-SDI, HDMI and CamLink as an input. It is important to mention that the input of the cubes can be multichannel, thus one hardware can process the images of more than one camera. Also, they are able to run multiple detection at once, as you can see on the functional block-diagram on figure 3.2. Their modular housing-board design make it possible to upgrade or change one or more blocks. As an output, Ethernet, CamLink, 3G-SDI, mSD card connections are given. Ethernet port can be used also for setting up parameters.



**Figure 3.2:** On this figure, the system's block diagram is introduced. Notice the separated groups of full-frame and foveal processes. (Proprietary to EUTECUS, INC.)

## 3.4 The foveal image processing method

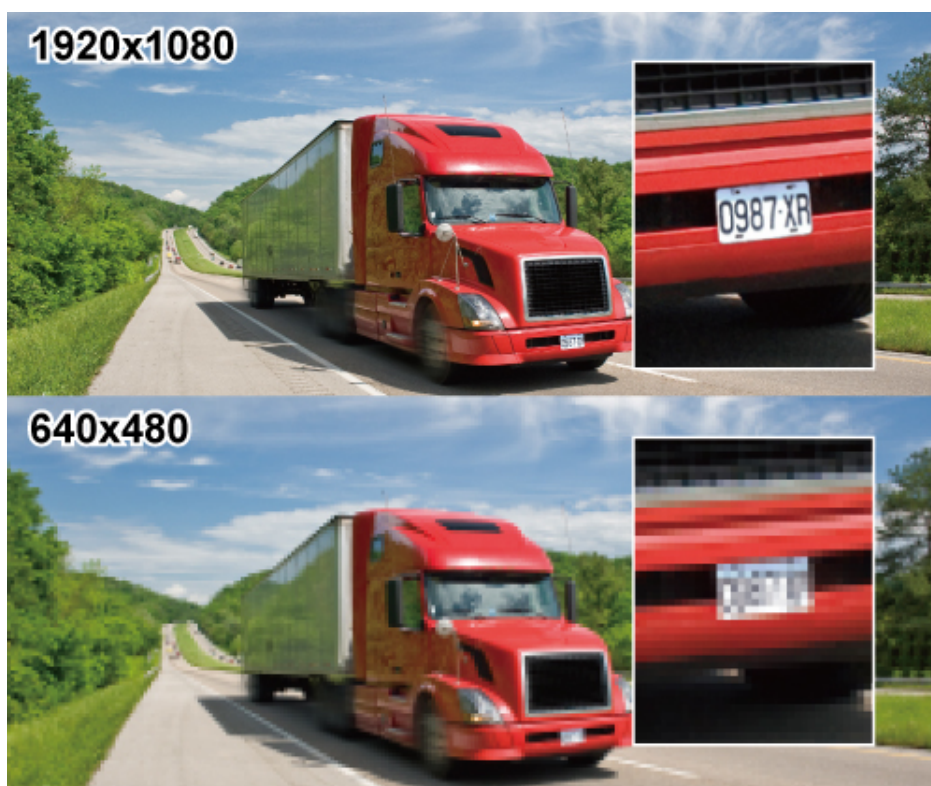
### 3.4.1 Comparison of full-frame and foveal image processing

Almost all existing image processing methods are based on full-frame processing, which means that with some predefined indexing and iterating method, all pixels of the input image are examined and processed. In case of observing one or few static images, this is not really a problem as running time is not that important. However, in real-time video processing, there is not always time for that. In many cases, the processing is performed using kernels, (convolution matrices) thus every pixel is processed along its neighbourhood. Of course, this can make the processing time even longer. (For example, a simple edge detection uses a matrix with a minimum size of 3x3)

Thanks to the current camera technologies, high definition (commonly referred as HD) records are standard now even among average consumers. One of the most common resolutions is the full HD, which means 1080 rows and 1920 columns of pixels. Working on such a video stream with any full-frame processing method means that every input frame contains more than 2 million pixels to process. It is not always a solution to work with smaller resolution video streams, as in many cases details are very important for the recognition. An example for such a resolution decrease is shown on figure 3.3. As you can see, the license-plate is no longer recognisable.

These videos are usually produced with at least 30 fps (frames per second). However, we do not have the technology to process at a similar rate using reasonable resources. Generally speaking, 30 fps processing on a powerful desktop computer is considered fast. On embedded systems this rate is usually around 15 fps. Example of similar systems can be found in resources [3] [5] with 15 and 11 fps processing speed, respectively. A frequent solution for this problem is to simply under-sample the stream. This means, for example, throwing away every second frame resulting in a 15 fps input and half the time to process it. On the other hand, half of the input is then lost.

The foveal system is inspired by the human vision. Our attention -caused by biological reasons- is focused in a relatively small area called the fovea. This is where



**Figure 3.3:** Same image is shown in HD(upper) and in VGA(lower) resolution. As you can see, license-plate number is not readable with such an image quality drop. (Image source:[7])

our sight is the sharpest and figures or shapes are recognised most easily. Movements or sudden changes are detected outside of the area too (more precisely, actually) but the resolution is much smaller. The idea behind the environment is the same. Certain areas of the actual frame (called foveae) are selected, and the detection algorithms are run on each cropped input. Of course, this method will provide much less input data for the algorithm. However, as its only task is to decide whether there is a car on the image, it will be shown that it is enough.

With such a different approach, much less pixels are processed. For example, with 30 foveae, each with a default size of  $150 \times 150$  pixel the number of processed pixels are:

$$30 \times 150 \times 150 = 675000$$

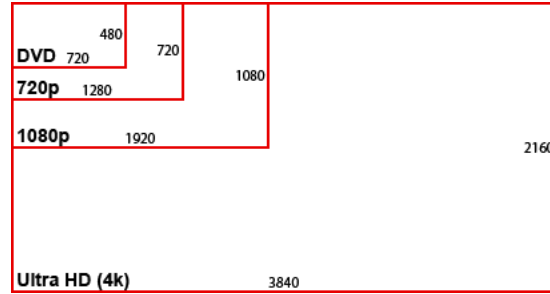
while processing the whole full HD frame would consider

$$1080 \times 1920 = 2073600$$

pixel. Thus, only  $\frac{675000}{2073600} = \sim 32.5\%$  of the image is processed with the same efficiency(see

### 3.4 The foveal image processing method

---



**Figure 3.4:** Demonstration of the ratios of the most popular resolutions. As you can see, 4k images contains four times more pixels than full HD. (image source: [8])

results in chapter 5). (The values used in this calculation are the typical parameters of our algorithm.)

Such a system is not just more resource-efficient (as shown above) than a full-frame processing one, but it is far more easy to up-scale. 4k resolution is expected to be the next big step in the video technology. As shown on figure 3.4 4k images contains four times more pixels than full HD. After a camera update, in case of a full-frame processing, a four-time faster system would be required (or the resolution has to be downscaled). Alternatively, you could divide the image for four part and process them on four existing system combined. Dividing the picture and managing the boundaries would extend the process time even more.

With a foveal based system, the change will not result such dramatic requests. If the 4k frame is an image of the same area with more details, the foveae should find the objects from further. As will be shown a car, for example, is recognizable on an image with a size of 100x100 pixel. On 4k, this size is reached from a much larger distance. If the 4k frame covers a four time bigger area, four times more fovea needed. Thanks to the highly parallel set-up, this extra resources would be very easy to add.

#### 3.4.2 Preprocessing

Although the environment used for this algorithm is not a part of a full-frame processing method, some hardware-accelerated, optionally executed full-frame data collection for preprocessing is implemented. For example, a fast edge-map generator is available. Also, the environment is able to generate a probability map based on the spatial-temporal changes. The reason for this is to provide a priori information to the Fovea placer algorithms (3.4.3). Please remember that these preprocessing methods are not detections, just some helping tools for the placers.

#### 3.4.3 Fovea placer

**Fovea Placer Algorithms (FPAs)** are methods which are designed to place the foveae on the input images.

The simplest FPA is the random placer that places the foveae on random coordinates generated in range of the input frame sizes. A slightly more advanced one is a random placer with a predefined **Region Of Interest**(ROI). A ROI is a bitmap determining the regions where foveae should or should not be placed. For example, for the car detection introduced in this paper, a horizontal cut was applied, as there is very little probability to find cars on the sky.

More sophisticated placers with significantly higher efficiency can be developed.

As default, all foveae are placed and replaced by the placer. However, every fovea has the ability to disable this method, and request a new position defined by itself.

#### 3.4.4 Foveal algorithm

The functionality of a Fovea is mainly defined by the **Foveal Algorithm** (commonly referred as an FA). All projects currently developed in VirVa lab are foveal algorithms.

All measurements and decisions made by these are local. Their only inputs are the cropped image from the placer, and their previously saved properties (these fields are free to use between certain limits, however, rational load is advised).

Such an algorithm needs to be simple and relatively fast, as it will be used in a real-time system. To reach this, mainly the predefined functions in section 3.5 should be used.

One FA's outputs are:

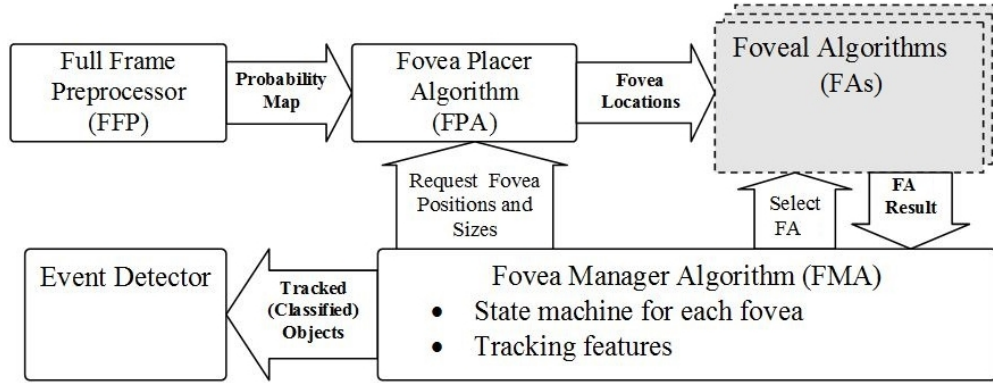
- whether the object it is searching for were found on the input
- flag to enable/disable automatic replacement
- optional: request for new position
- optional: request for new foveal size

A fovea is able to disable automatic replacement performed by the placer, and use a new position calculated during the process of the FA. Also, new fovea size can be determined. This is very useful when an already detected object needs to be tracked. In such a case, FA tries to approximate the movement of the object and the change in its size, and provides the information to the placer.

### 3.5 Implemented set of functions

Since the algorithm is meant to be implemented on an embedded system, it is crucial to design the algorithm as simple and fast as possible. Also, it would be pointless to use a method in the simulation that is impossible or very hard to compile for the hardware.

To prevent that, all functions that are available on the hardware already (with an optional hardware acceleration) have a bit-accurate copy in the Matlab development environment. These are:



**Figure 3.5:** An illustration of the software framework. Rules and connections between the parts of the system are represented. (Proprietary to EUTECUS, INC.)

- **statistics:** A powerful set of functions with capability to determine the mean, variance, minimum and maximum values of an image.
- **color-filter:** This function requires a three channel image input, the id (1:Y,2:Cb, 3:Cr) of the channel to work on, and a list of weighted parameters to filter for. Output is a grey-scaled image representing how far are pixel values from the input parameters.
- **histogram:** Yet another statistical function with the specific task to produce the histogram of a 3-channel input image. The output is a 1D vector.
- **integral image:** Implements the well known image-processing tool. Output is a 2D matrix, which value at any point is the sum of all the pixels above and to the left from it.
- **edge detection:** Sobel based edge-detection with the parameter of horizontal or vertical output is requested.
- **projection:** A transformation from 2 to 1 dimension. When executed, both vertical and horizontal projection are calculated.
- **peak-finder:** A very handy and fast tool to find peaks on vectors. (For example: histogram, projection vectors)

These functions are also called Foveal algorithmic blocks (FABs) as they are easy to use or implement and give the FAs a modular build-up.

## **3.6 Process of algorithm development**

### **3.6.1 Matlab simulation**

MATLAB is a high-level interpreted programming language and environment designed for numerical computation. It is very convenient to develop in, as thousands of functions are implemented and included in the software. Also, excellent visualization tools are ready to use which are necessary to easily debug and develop image processing methods.

These are the reasons why Matlab was chosen for the first development state. All methods (discussed in section 3.5) defined in MVE<sup>TM</sup> have a bit-accurate copy available in Matlab.

Also, a very useful development environment with a GUI were developed. It is able to read video, execute algorithms on it, and save the output. One can jump to specific frame, step the video stream, or even run the algorithm backwards.

Caused by the reasons above developing and testing image processing algorithms in Matlab is extremely convenient. On the other hand, Matlab is Java based, therefore most non-mathematical calculations are slightly slow compared to a C++ environment. Executions on videos can run on a decent 7-10 fps. Also, the available functions and algorithms are limited if we since the resources of the embedded system must be considered from the first state of the development.

### **3.6.2 C++ implementation**

After an algorithm was developed in the Matlab environment, and it passed the basic validation test on the selected video set, its implementation to the hardware is scheduled. However, first, only a virtual Bi-i is used with the exact same functions and blocks available on the real one. The whole hardware is simulated in C++, which is a quite strict object-based programming language. At this level of development, many smaller bugs can be discovered and fixed. (for example: casting problems, unsigned or signed variables, etc.) The algorithm in this state can already be executed with ~30 fps, which is considered as real-time. It should be mentioned that this is still a simulation.

### **3.6.3 Hardware implementation**

In the final development state, the C++ implementation is ported to the real hardware. Thanks to the second state, the algorithm has already been a very similar simulated environment, thus most of the bugs should be revealed. With this development step, the process is not yet done. To streamline the solution into the market, it has to pass complicated validation tests in its final form. Also, serious stress tests are applied often executed for as long as a week on the exact same hardware.

## Chapter 4

# Designing and implementing the algorithm

### 4.1 Challenges in the task

As mentioned before, the algorithm was planned to be used on a moving platform image acquisition system. Thus, many traditional video processing methods, like foreground-background detection (discussed in 2.2.1), could not be used in this case, as there is no fix background or ego motion position. (Please note that these constraints do not mean that the presented solution would not work on a standing platform.) Also, the moving platform means dynamic perspective change. This makes approximating distances (and selecting how large objects to look for) a much more difficult task.

Being a foveal algorithm, the design needs an approach completely different from full-frame solutions. Due to by the software hierarchy introduced in 3.4, the method's only input is a small, cropped part of the image. This means it has much less information to work with. Also, the algorithms usually search for objects all over the picture, and then they make a decision on the selected regions. Our software has no or very little impact on choosing the possible areas. Its only defined task is to decide if an input image has a car on it, or not.

All challenges mentioned above are true for any moving platform problem implemented in this or a similar foveal system. The fact that we are look for vehicles makes it even more complicated. If an algorithm is designed to find traffic signs on a picture, it is a very reasonable decision to use the standardized shapes, sizes and colours as conditions. For example, speed limit tables (in most European countries) are round, with a red border, white interior, and the speed is written with black in the middle. In my task, a general description like that is not possible to give, caused by the fact that cars, (even with trucks, buses excluded) comes in significantly different sizes, shapes, and colours. That makes it very hard to define conditions for the detection.

After analysing numerous videos and images, we recognized that instead of defining proper thresholds, we should looking for some general patterns which are easy to implement, and would match with a high percentage of the cars. All parameters should

be defined relatively since the diversity of the objects.

## 4.2 Constraints and presumptions

Caused by the reasons above, we had to define some constraints to simplify the detection while keeping the number of false-positive hits (5.4) as low as possible and keeping the algorithmic definition of a car general enough to detect them in practice. These constraints are generally true in most of the use cases (e.g. the view angle of the car is right). These properties help us to define the exact problem we want to deal with.

- **Rear view with a decent angle:** As discussed in section 1.4, namely the vast number of different car variants, this system was designed to prevent rear-end collisions in environment like highways.
- **The image of the car has a decent resolution and sharpness:** The farthest car identified during the test were bounded by a 75x75 sized box. As you can see, distance, and size in pixels have a tight connection. In general, no cars below the projected size of 80x80 pixel are intended to be recognized.
- **Has a rear pair of red lamps:** The lamps have to contain significantly higher intensity values on the Cr channel (the red channel of the YCbCr colour space) compared to the surrounding areas. All thresholds are set by an adaptive histogram based method.
- **Visible and standard license plate:** For the tests, parameters were calibrated to match the standard European plates. Cars without a plate are easily missed.

## 4.3 Description of the algorithm

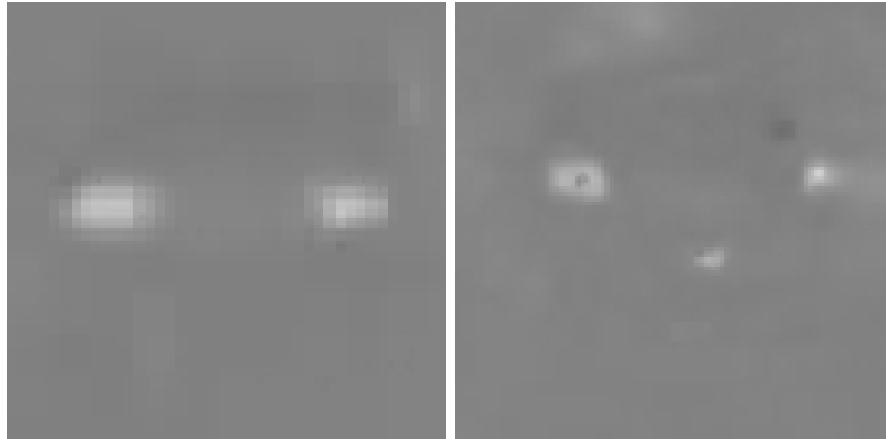
Two cars (shown on figure 4.1) were chosen to illustrate the main steps of the algorithm. These images were cropped by the fovea placer from the test video, and provided to the foveae as an input. As described in subsection 3.4.4, no other input is given (aside the previously saved and actual properties of the foveae, like positions, sizes, counters, etc.)

The environment uses YCbCr colour space. In this 3D space, Y is the luminance, CB and CR are the blue-difference and red-difference chroma components. On figure 4.2, Cr channels of the inputs are shown. As you can see, only the red lamps have significant contrasts with their neighbourhoods, and are clearly visible. This is caused by the fact that Cr and Cb values are usually change in a quite narrow range. Notice that on the right image, there are three brighter patches because a reflection plate is equipped in the middle at the bottom is also visible.

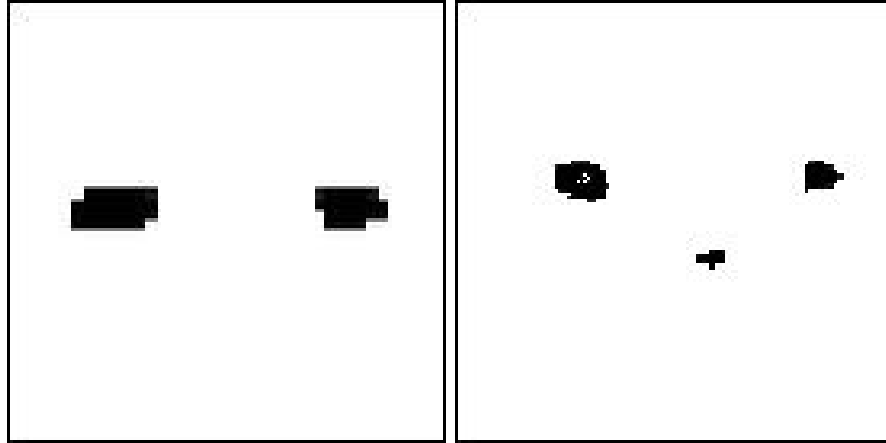
A simple histogram (see section 3.5) provides the most common value of the Cr channel. A colorfilter with this parameter and adaptive range is applied. After that, only the pixels of the lamps remained black, as their values were not close enough to



**Figure 4.1:** Two cars were selected to illustrate the main steps of the algorithm. These images were cropped by the fovea placer from the test video, and provided to the foveae as an input. Following images were made during the processing of these inputs and are arranged respectively (results from the left image will be shown on the left in every figure, and around.)



**Figure 4.2:** The environment uses YCbCr colour space. On these figures, Cr channels of the inputs are shown. As you can see, only the red lamps have significant contrasts with their neighbourhoods, as Cr and Cb values are usually change in a quite narrow range. Notice that on the the right there are three patches because a reflection plate is equipped in the middle at the bottom.



**Figure 4.3:** A simple histogram provides the most common value of the Cr channel. A colorfilter with this parameter and adaptive range is applied. After that, only the pixels of the lamps remained black, as their values were not close enough to the filter's parameter. As you can see, these images are binaries, while the previous ones were grey-scaled.

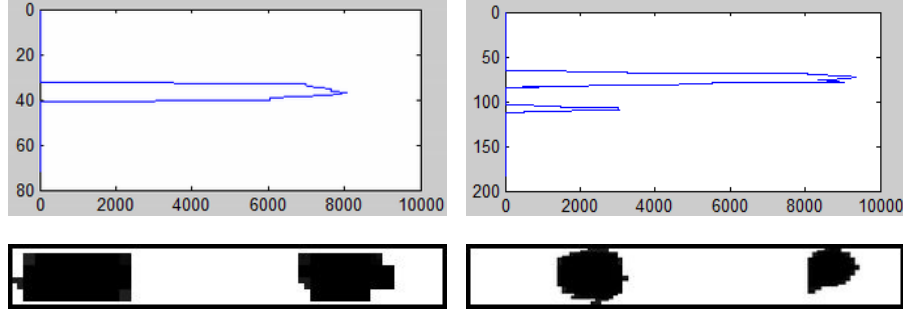
the filter's parameter. With this method, we created a very accurate segmentation for the lamps. The output of the filter has to be inverted. As you can see, these images are binary, while the previous ones were grey-scaled.

As shown on figure 4.2, a vehicle could have more than two red patches on it. These objects (for example: stickers, brake lamps, etc.) could mislead the lamp localization. To prevent that, a crop is performed to keep only the rear lamps. Horizontal projections of the filtered images are shown in the upper row of figure 4.4. The highest peaks of these are selected to cut the horizontal lane of the supposed lamps. Results of the cropping are shown in the second row. You can see on the right bottom image that the third patch from figure 4.2 is now dismissed. This step assumes that the camera is mounted with zero or relatively small rotation along the horizontal axis. To fix such a transformation is easy but very resource-intensive.

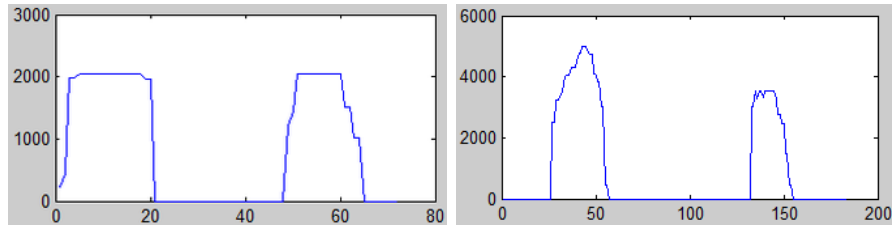
After the cropping, a vertical projection is executed (demonstrated on figure 4.5). Notice the two typical, standalone peaks in both vectors. Further conditions were set up for minimum heights and distance between them. As you can see, the right input has around twice as much red pixel in the cropped area. This is an excellent example to represent the difficulty of setting up these conditions. Clearly, a common threshold would fail. Thus, parameters of these conditions are always adaptive.

To detect the license-plates, horizontal edge-maps are used. Notice the strong outlines of the plates on figure 4.6. Strongest horizontal edges are chosen during the process. Each gap between neighboring chosen edges is analysed. The algorithm searches for a bright, rectangle-shaped object (license-plate candidate) in these gaps. See section 5.3 for results.

To decrease the numbers of false positive hits, an artificial delay were implemented,



**Figure 4.4:** As shown on figure 4.2, a vehicle could have more than two red patches on it. These objects (stickers, brake lamps, etc.) could mislead the lamp localization. To prevent that, horizontal crop is performed. The highest peaks of horizontal projections are selected to cut the horizontal lane of the supposed lamps. You can see on the right bottom image that the third patch from figure 4.2 now is dismissed.



**Figure 4.5:** After the cropping, a vertical projection is executed. Notice the two typical peaks on both figures. Further conditions were set up for minimum heights for distance between them. The parameters of these conditions are always adaptive. For example, the right input has around twice as much red pixel in the cropped area. Thus, a common threshold would fail.

namely we have to detect the car on at least three consecutive frames. This feature caused a significant improvement for the hit accuracy, while three frame means only  $\frac{1}{10}$  second delay (assuming the video stream's rate is 30 fps).

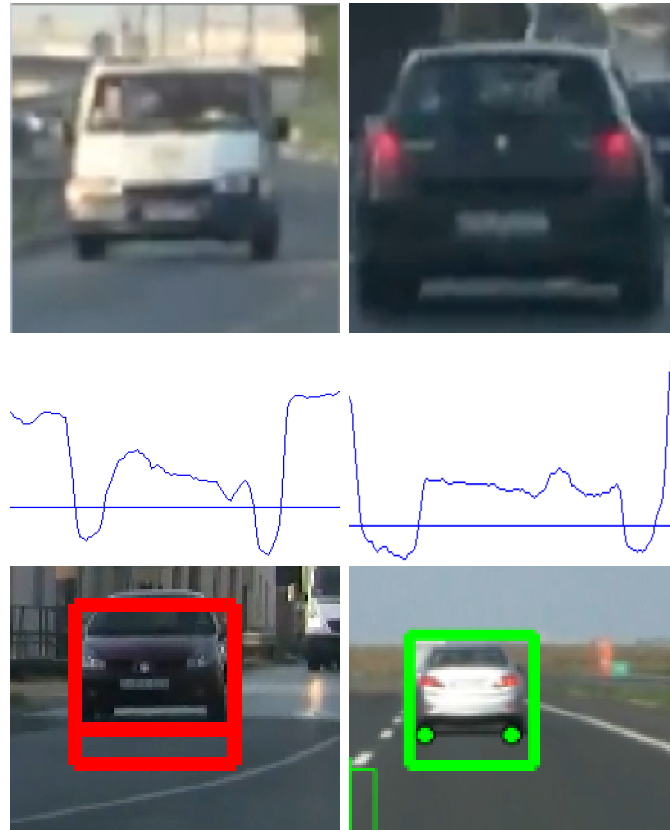
An effective tracking mechanism was necessary to be implemented since all the found cars should be monitored until they are within the visual range of the camera. Thus, when one of the foveae detects a car, it will disable the automatic replacement, and will request a new position, calculated by the coordinates of the lamps. Horizontally, the fovea tries to rearrange itself to the mean of the x coordinates of the lamps. Vertically, the lamps lower bound is intended to be in the center, because many cars have very long shaped lamps. If the tracking system would try to keep the average height of these lamps in the center, in some unfortunate cases the fovea could "slip off" the car upwards.

In the unlikely case when all conditions are given, but for some reason, our algorithm does not recognize the input, an alternative solution were provided. In this standalone



**Figure 4.6:** To detect the license-plates, horizontal edge-maps are used. Notice the strong outlines of the plates. Strongest horizontal edges are chosen during the process. Each gap between neighboring is analysed. The algorithm searches for a bright, rectangle-shaped object. See section 5.3 for results.

FA, cars were recognized by the typical shape of wheels. On figure 4.7, the steps of this alternative detection system is illustrated. A basic, variance-based function tries to locate the bottom of the wheels. Then a cropped region above this index is selected. After a vertical projection of the Y channel, I got the result vectors shown in the second row. You can see that the typical outlines of the wheels appear. With some simple conditions, false positive errors can be filtered using only these vectors. If this FA is used, it is a necessary constraint that the car's wheels being processed have to be visible. Should the vehicle get too close, the tracking will fail. Please note that this alternative detection method were not included during all presented tests in chapter 5.



**Figure 4.7:** This figure illustrates an alternative detection method that was also implemented. Its decision system is based on the shape of the wheels. In the upper row, input images are shown. The second row demonstrates how the typical outline of the wheels is appearing in the vertically projected vectors. Third row represents that algorithm can calculate the vertical and horizontal positions of the wheels.

## Chapter 5

# Results

### 5.1 Test videos

I have developed a foveal algorithm that is able to detect and track multiple cars simultaneously with a good hit ratio under circumstances declared in 1.4. Some constraints (detailed in 4.2) were defined to simplify the detections while keeping the number of false-positive hits (5.4) as low as possible. The algorithm was tested on numerous video examples recorded with an on-board dash camera, and we got impressive results. The software succeeded well on videos recorded on the highway with good visual circumstances. However, we got slightly lower (but still usable) hit accuracy on rainy, foggy and night scenes. Also it should be mentioned that cars with red painting or not regular lamps are detected poorly.

Please find example result videos for all these cases at [this link](https://users.itk.ppke.hu/palan1/.ReportVideos)<sup>1</sup>. On every video and image presented in this report, red color represents a hit (a car was detected), yellow means a not validated hit (possible position of car), and foveae marked blue detected no cars. Some of the videos contain only the hits (marked with red) to have a clear overview. Each detection was executed with 20-30 foveae placed by the random fovea placer supported with ROI (see 3.4.3 for details). Thus, the results represented here were reached by processing only around ~32.5% of the image. More sophisticated placers have reached similar results with even fewer, 7-10 foveae [9].

On figure 5.1 on page 28 you can observe some vehicles detected by the algorithm in the videos. Each row represents one car captured on different frames. As you can see, the algorithm detected and tracked the vehicles as they were passing by and the object seemed to grow bigger. Also, the view angle changed significantly during the tracking. Please notice the truck which was also detected, since it fulfils the simple conditions we set up in 4.1.

---

<sup>1</sup><https://users.itk.ppke.hu/palan1/.ReportVideos>

## 5.2 Distance approximation

To demonstrate the accuracy of the algorithm and the usefulness of the information produced, I generated an animated bird's-eye view of the section of the road in front of us. Please note that all information is gained from the videos by the implemented software, no other sensors (like radars) were used. An example of this overview can be seen on figure 5.2. Distances, positions (and even the colours) of the cars are estimated from the output of one fovea with some trigonometrical and optical calculation. The output contains the position of the fovea, the distance of the lamps, and the average color of the car (the area between the lamps). The only fixed values are the coefficients of the linear equations which describe the edge lines. These are used to calculate the position of the vehicles along the x (horizontal) axis. In the near future these data will be provided by an other foveal algorithm which aims to detect lanes and is currently under development.

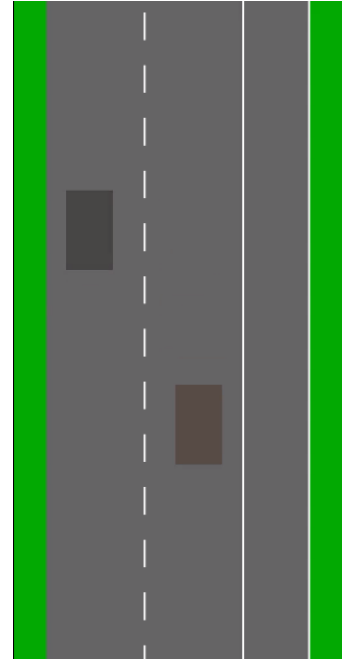
A snapshot of a scene with multiple detections is shown on figure 5.3. The cars are detected and their approximated positions are transformed to the 2D overview in the top right corner. The full video is available on-line on the link given above.

## 5.3 License-plate detection

Although it was not an intended purpose, during the development, a basic license-plate localization system was implemented to help decreasing the false positive hits. This condition resulted in the ability that the algorithm can detect and mark license-plates on cars, if the car is not too bright. In the latter case, contrast between the plate and the other parts of the car is too low. On figure 5.4, two sample frames are shown. On the left image, both licence-plates are localized.

Such a function could be used by real-time, automatic license-plate recognition systems used in private areas where vehicles can enter only with permissions. Another application field could be police patrols. Since the algorithm is designed for moving platform, it should be possible to implement a system that detects, recognises and analyses the license-plate that belongs to the car ahead. Connected to a database, the system would be able to alert the officers if a vehicle is stolen or has no valid documents.

Distance approximation (5.2) is currently based on the distances of the lamps. However, this could provide misleading information since a bigger car would be considered closer than in real life. The other way around,



**Figure 5.2:** A frame cut from the automatically generated bird's-eye view videos. The car equipped with the camera is in the left lane at the bottom, outside the image.

smaller cars might be detected further. To eliminate this, we plan to use the size of the license plate also for distance approximation, as they have standard sizes and shapes. Thus, a fix and general distance reference is available on every car. In both cases, we can still detect if the vehicle is approaching which is crucial.

## 5.4 Image dataset

In every decision system the mistakes can be organised into two main groups. First, a mistake is a type I error (or with more common appellation: false positive error) if the system classifies an input as a hit despite it is not. In our example this typically means that something that is not a car is still recognized as one. On the other hand, a type II error (also called: false negative error) is a mistake that occurs when an input is not recognized although it should be. In this current task, a false negative error is when a car is not detected although every constraint is satisfied.

The actual task determines which kind of error should be minimised. In certain projects it could be more important to find every possible desired object than the fact that some mistakes will occur. Such a project can be a manually supervised classification, where appearing false positives can be eliminated. Also, a system with an accident avoidance purpose should be over-secure, and rather recognise non dangerous situations as a hazard than to miss a real one. However, it is needless to say that a solution like that with many false alerts could be slightly annoying for the users.

Unfortunately, it is really complicated to decrease both kind of mistakes at the same time. Should you set up more strict rules in the decision making, the rate of false positives will decrease, but simultaneously, chances of false negatives will increase. On the other hand, if you loose a condition from the detection method, false negative errors may be reduced. However, caused by the exact same thing (less strict conditions) false positives could appear.

To measure the hit-rate of the algorithm, a validation test with an image-database has been set. Considering the fact, that our algorithm works with only a cropped image of a supposed car, it would be really hard to use an existing, well-known dataset, as usually these contains more than one car per image. To test the rate of both false-positive and false-negative errors, images about cars and "not cars" are included.

More than 400 pictures manually taken of cars in urban area (Corvin negyed, Budapest, 2013.11.06) with wide variety of types, brands, sizes, colours, etc. All pictures contain exactly one car viewed from the back. Please remember that as discussed in 3.4.4, the foveal algorithm's task is to decide whether the input image is a car. It is not planned to find the possible objects on a high quality image, as it is the Fovea placer's task (3.4.3).

Also more than one thousand images were collected that do not contain cars. The latter group includes manually taken pictures in urban area and automatically cut ones from the test videos. Our aim was to produce a set of images which are likely to be captured with an on-board camera. Thus, this set of pictures mainly contains cropped images of roads, sky, clouds, trees, buildings, etc. In other words, many objects and

examples that could be found near a highway but are not cars included.

The whole database, -including both image sets- can be downloaded at [this link](#)<sup>1</sup>. The uploaded image sets are identical to the ones used for false-negative and false-positive sets. On figure 5.5 and 5.6 some examples are shown from our datasets.

These images were not used for learning, only for testing, as there is no learning method in the system. They neither had any other special role during the development (previous tests for example).

Please note that although these tests were run on static images, the software was planned and implemented to be used with video input. Every fovea has three main states: free; possible, but not validated hit; and validated hit. If a fovea is in the status of possible, but not validated hit, it means that an object was found with similar properties of a car, but it is not able to make a decision based only on the actual and previous frames. In these cases the fovea is placed on the same or on a nearby position to finish the decision. It is understandable that testing this algorithm with images is not a perfect solution, although there is no other more precise method. Alternatively, we could cut out example frames from the videos with a video annotation tool, but it would have resulted a smaller dataset with less diversity.

To simulate a video stream, each image have been given as an input for the test fovea three times. After executions, we checked the status of the fovea and logged the results.

In table 5.1 results of the false-negative tests are shown. On 431 test images 46 mistakes occurred. These inputs were not recognised causing false-negative errors. 284 input images (~65 %) were recognised as cars. For the rest of the pictures the algorithm got in the possible hit but could not validate the detection based on one frame. This means a hit accuracy as high as 89% if we assume that not validated hits would be validated in the next few frames. Please note that these pictures were taken in urban area from not always ideal view angles. In a highway like test environment (which are planned to be the primary application area) an even higher percentage should be received.

In table 5.2 results of the false-positive tests are shown. It can be seen that the algorithm made a mistake on 11 pictures (~1.07%) and couldn't make a decision in 19 cases (~1.8%). In a worst case scenario (assuming that all not validated images would be recognised as a car in the next frames) the ratio of false positives is still below ~3%.

The tests introduced above are completely reproducible and both image sets are available on-line. Comparison with articles about the similar topic has been done. Some results of this are collected in table 5.3. In the comparison, [3], [5], and [4] were involved. Our algorithm performed very well in aspect of accuracy as only [4] reached higher detection rate. However, this is a learning based, off-line system that was tested on a much smaller database.

From these three competitors, only [3] was developed without a learning method. All three of them use full-frame processing methods, thus it is rather complicated to compare them.

---

<sup>1</sup>[https://users.itk.ppke.hu/palan1/image\\_database](https://users.itk.ppke.hu/palan1/image_database)

|                               |               |
|-------------------------------|---------------|
| number of test images:        | 431           |
| number of validated hits:     | 284           |
| number of not validated hits: | 101           |
| number of false negatives:    | 46            |
| accuracy:                     | <b>~89.3%</b> |

**Table 5.1:** Results of the false-negative tests. The algorithm succeeded to recognize as many as 89% of the cars on the input images.

|                               |               |
|-------------------------------|---------------|
| number of test images:        | 1021          |
| number of validated hits:     | 11            |
| number of not decided images: | 19            |
| number of false positives:    | 30            |
| number of rejected inputs:    | 991           |
| rate of false positives:      | <b>~2.93%</b> |

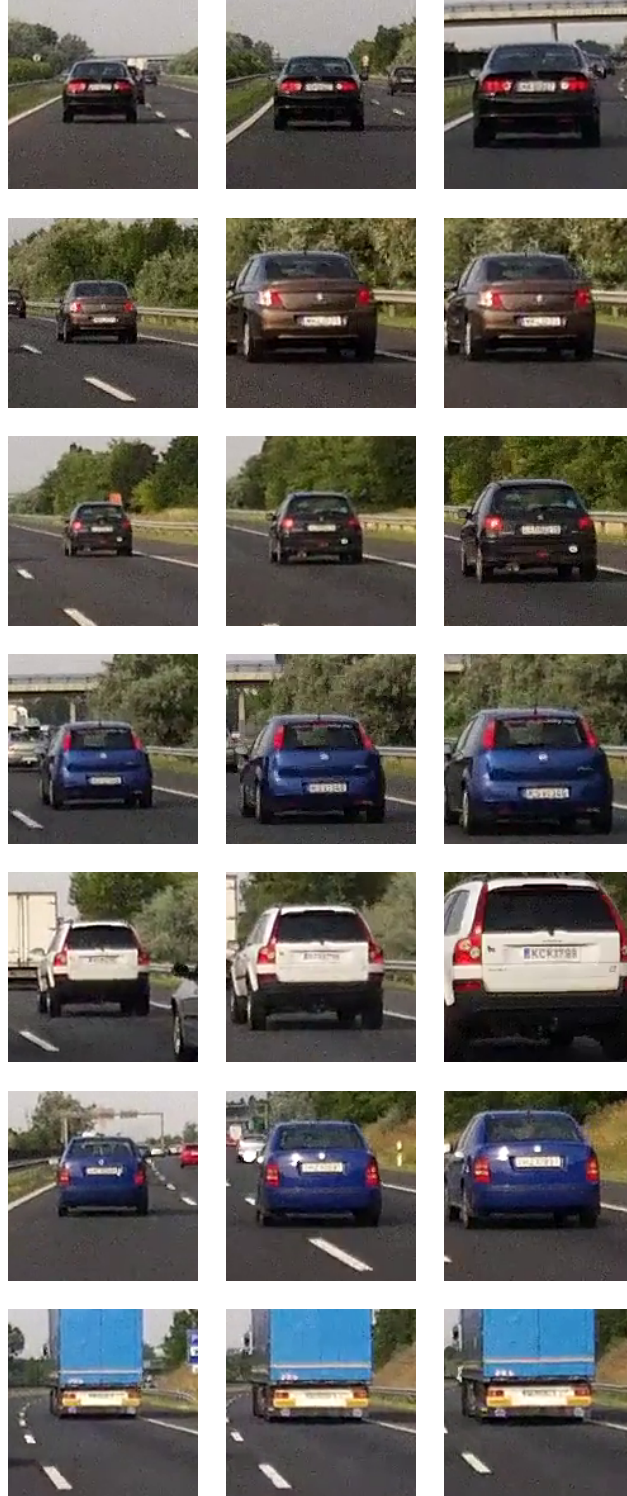
**Table 5.2:** Results of the false-positive tests. As you can see, the algorithm made a mistake on 11 pictures (**~1.07%**) and couldn't decide in 19 cases (**~1.8%**).

|                        |               |                     |       |       |
|------------------------|---------------|---------------------|-------|-------|
| Algorithm:             | Our algorithm | [3]                 | [5]   | [4]   |
| number of test images: | 431           | not known           | 516   | 149   |
| rate of accuracy:      | <b>~89.3%</b> | 83.7 <sup>1</sup> % | 88.6% | 96.0% |

**Table 5.3:** Comparison to other similar solutions. Third algorithm reached an outstanding hit rate. However, this system is not embedded, is based on learning, and has a processing fps as low as 15. The fourth algorithm's percentage is excellent too. On the other hand, it is also a heavily off-line system based on learning.

No information was included in either of the articles about the false positive tests, thus I could not compare the results. Only [4] mentions it that on 80 images, 24 detections were false.

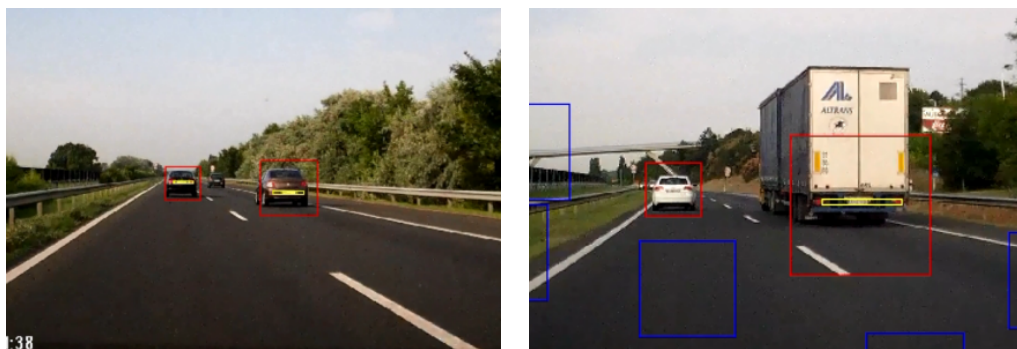
Considering that our solution is real-time, designed for embedded systems and has no learning ability, no similar system is known.



**Figure 5.1:** Sample images of detected vehicles. These pictures were automatically saved from the videos during testing the algorithm.



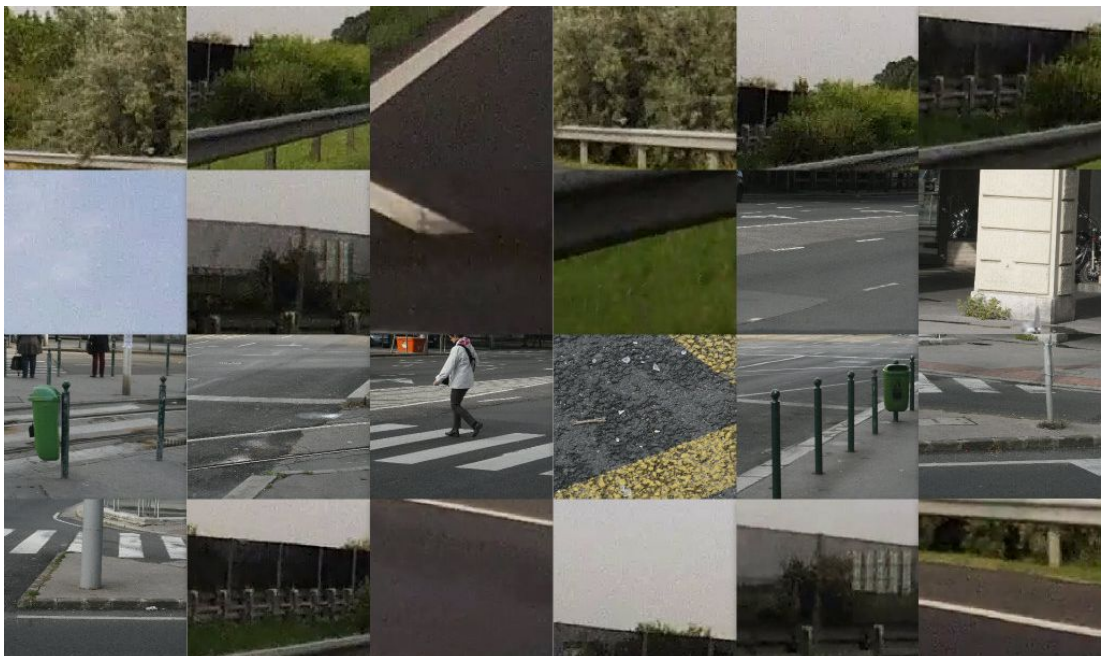
**Figure 5.3:** Sample image to demonstrate the efficiency of the detection based overview. Both car are recognised, tracked and positioned in the projected 2D space. Also, license plates(5.3) are detected and marked with yellow.



**Figure 5.4:** Demonstrating the license plate detection feature. On the left image, both plates are detected. On the right, you can see an example when the detection fails caused by the brightness of the car. However, the truck's plate is localized correctly.



**Figure 5.5:** Some example from our false-negative dataset. Notice that many different coloured, shaped and sized vehicles were included.



**Figure 5.6:** Some example from our false-positive dataset.

## Chapter 6

# Conclusion

In this study, we discussed the main differences between full frame- and foveal image processing in the light of resource efficiency, speed, and parallel implementation opportunities.

A foveal algorithm was defined for the task of the real-time vehicle detection on the input video stream of an on-board camera. Our approach is a simplified, adaptive pattern matching for rear lamps and license plates. After defining the possible positions of the lamps and the license plates, the algorithm is able to decide and detect different sized, coloured and shaped cars from a clear rear view.

In case of a detection, the included tracking system is able to determine the movement and size alteration of the tracked object, and replace the fovea based on this knowledge.

The novelty of this approach is that all decisions are made locally, this way the execution of the foveal algorithms is scalable and fits ideally multi-parallel architecture.

Our software was tested on numerous test videos containing scenes with two or more visible cars, and we got impressive results both in detecting and tracking. To define an objective rating, a database was set with more than one thousand images from different sources. A hit accuracy of  $\sim 89\%$  was reached.

These results could be improved with a more sophisticated license plate detection or a combined method of the wheel and lamp detecting algorithms. Also, all the parameters should be optimised on a large input database set (like the one I defined in section 5.4).

As mentioned in section 2.1, a stand-alone image processing vehicle detection (and neither of the alternatives, like radars) is not enough to perform the task (collision warning and vehicle detection) sufficiently. Thus a fusion of sensors is required. For example, the region of interest could be chosen by the obstacles detected by the radar or a laser sensor. The more precise distance approximation would improve our algorithm too (for instance, in choosing the size of the fovea). One or more additional camera with different view angles could help the validation, and the depth to points can be determined, too. However, for this, one has to find the corresponding points on the images, which is a rather complicated task. A combined system of the presented

---

algorithm and a radar sensor is planned. Also, as the Bi-i is able to handle more than one video stream, a multi-camera system is considered.

Although this is a demo solution working in a Matlab development environment, the implementation of the algorithm on the simulated hardware has recently started. A bit-accurate model of the discussed algorithm is implemented in C++ language and the preparations of the hardware implementations have begun.

## Acknowledgements

Ezen dolgozat kizárólag saját erőmből soha nem valósulhatott volna meg, ezért itt szeretném megköszönni azoknak, akik hozzájárultak a létrejöttéhez.

Köszönöm Dr. Horváth Andrásnak, hogy fáradtságot nem ismervé segített és irányt mutatott a fejlesztés és dokumentálás nehézségei során. Köszönöm Dr. Rekeczky Csabának, aki lehetőséget adott, hogy valódi, ipar közeli feladatot válasszak a dolgozat témájául, illetve tudásával, és tapasztalatával sok kérdésben segített ki.

Köszönöm egyetememnek és karomnak, a Pázmány Péter Katolikus Egyetem Információs Technológiai és Bionikai Karának, hogy a kutatáshoz helyet, időt, és keretet nyújtott. Hálával tartozom az Eutecus cég alkalmazottjainak, akik fejlesztő környezetük rendelkezésre bocsájtásával és technikai támogatásukkal lehetővé tették a fejlesztést.

Köszönöm családomnak és barátaimnak biztatásukért, türelmüket és szeretetüket. Köszönöm Hajdu Ákosnak, és Hunyady Mártonnak, akik rendkívül precíz és szakmai lektorálással segítettek.

Külön köszönettel adózom Kovács Lórántnak, Huszár Domonkosnak, és Melles Fanninak, akik támogatása és tanácsai különösen sokat jelentettek.

Pálffy András

# References

- [1] S. Heath, *Embedded Systems Design*. Newnes, Oct. 2002.
- [2] A. Prati, I. Mikic, C. Grana, and M. M. Trivedi, „Shadow detection algorithms for traffic flow analysis: a comparative study,” in *Intelligent Transportation Systems, 2001. Proceedings. 2001 IEEE*, 2001, p. 340–345. [Online]. Available: [http://ieeexplore.ieee.org/xpls/abs\\_all.jsp?arnumber=948680](http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=948680)
- [3] S. Han, Y. Han, and H. Hahn, „Vehicle detection method using haar-like feature on real time system,” *World Academy of Science, Engineering and Techonology*, vol. 59, 2009. [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.192.8220&rep=rep1&type=pdf>
- [4] D. C. Lee, *Boosted Classifier for Car Detection*, 2007. [Online]. Available: [http://www.cs.cmu.edu/afs/cs/Web/People/dcleee/car\\_boosted.pdf](http://www.cs.cmu.edu/afs/cs/Web/People/dcleee/car_boosted.pdf)
- [5] J. Choi, „Realtime on-road vehicle detection with optical flows and haar-like feature detectors,” *Urbana*, vol. 51, p. 61801, 2007. [Online]. Available: [http://reason.cs.uiuc.edu/jaesik/projects/Vehicle\\_Detection/jaesik\\_cvpr.pdf](http://reason.cs.uiuc.edu/jaesik/projects/Vehicle_Detection/jaesik_cvpr.pdf)
- [6] C. Rekeczky, J. Mallett, and A. Zarandy, „Security video analytics on xilinx spartan-3A DSP,” *Xcell J*, vol. 66, p. 28–32, 2008. [Online]. Available: <http://soc.echelonembedded.com/files/Spatan3ADSPIMGPROC.pdf>
- [7] „OD-2050HD.” [Online]. Available: <http://www.airlive.com/product/OD-2050HD>
- [8] „What is the resolution? (ultra HD (4k), 1080p, 720p, DVD).” [Online]. Available: <http://www.rtings.com/info/what-is-the-resolution-ultra-hd-4k-1080p-720p-dvd>
- [9] K. Lóránt, „Designing multifoveal image processing algorithms inspired by the human visual system’s attention mechanism and implementing on multiprocessor architectures,” PPKE ITK, Tech. Rep. November, 2013.