Institute of Parallel and Distributed Systems
University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3383

Real-time detection and tracing of vehicles via camera systems

Andreas Harnisch

Course of Study:	Computer Science
Examiner:	Prof. Dr. rer. nat. habil. Paul Levi
Supervisor:	Dipl.-Inf. Bernd Eckstein
Commenced:	August 20, 2012
Completed:	February 19, 2013
CR-Classification:	I.2.10, I.4.8, I.4.9

Contents

List of Figures

List of Tables

List of Listings

List of Algorithms

Abstract

Traffic surveillance and analysis is an important matter to increase the safety on the roads. By monitoring the traffic appropriate measures be taken when dangerous situations appear or to keep the traffic flow steady. Therefore it is necessary to use a system that can detect and track vehicles so that details of the traffic situation can be evaluated. A distinction of the vehicle types is also useful to get more detailed information about the traffic. In this diploma thesis different approaches for vehicle detection and tracing via camera systems are implemented and evaluated. The implementation is a plug-in for the program FloDiEdi done in C++. Three different approaches are implemented and evaluated. The first and second approach use the Shi-Tomasi feature detector and an modified Lucas-Kanade feature tracker. Perspective transformation is used in the second and third approach too. In the third approach a combination of position estimation and template matching is used to track vehicles. For speed calculation the first approach uses cross ratio. The second and third, due to perspective transformation, use a pixel to meter conversion to get the distance in meters. The extracted information are visually presented such as the ID, the type and the speed of the vehicle is displayed and additionally stored to an xml file. Furthermore in the xml file the lane and the size of the vehicle is stored. The camera system is a stationary system which captures the images of the street. The first approach achieves a high detection rate but the tracking does not work well, so just around 10% of the vehicles were successfully tracked. In the second approach the detection rate is lower than in the first but tracking works more reliable with an tracking rate of around 30%. The third approach has the same detection rate as the second approach but the amount of successful tracked vehicles is over 70%. Concerning the computational time, the third approach is seven times more faster than the first and second, which are similarly slow. After evaluating the three methods the fastest of the three approaches and most accurate approach is the third one. There is still room for more improvements such as achieving a speed up by using the GPU for matrix operations or parallel programming.

List of abbreviations

BSD	Berkeley Software Distribution
CUDA	Compute Unified Device Architecture
IDE	Integrated Development Environment
FFT	Fast Fourier Transformation
FPS	Frames Per Second
FloDiEdi	Flow Diagram Editor
GIMP	GNU Image Manipulation Program
GPL	General Public License
GPU	Graphics Processing Unit
LGPL	Lesser General Public License
ROI	Region Of Interest
SVD	Singular Value Decomposition

1 Introduction

1.1 Motivation

The number of vehicles sold worldwide is permanently increasing. As a result the streets are getting more and more congested. It can be observed every holidays in Germany that there are huge traffic jams, especially in the summer holidays[1]. Long traffic jams are a result of more cars on the street. An idea to confront this problem is simple but not always feasible - build more or wider streets. Building streets costs a lot of money, takes a lot of time to build and of course the ground needs to be available or bought which costs also money. In Singapore a "Certificates of Entitlement" is necessary to be permitted of driving on the streets[2]. These certificates are sold on auctions where the government regulates the amount of certificates. This is another way of dealing with the number of cars on the street but reduces the freedom to drive a vehicle. A solution for the problem of congested streets would be to adapt the speed limit to the current traffic volume or the admittance of driving on parts of the streets where it is normally not allowed, like the emergency lane. In a real world example it is possible to act before a traffic jam occurs. When observing the traffic and a general slow down is noticed the emergency lane could be opened to let the traffic flow more fluently. For observation the amount of cars on the street needs to be taken into consideration. Also the current speed, compared to the average speed, is important.
In this diploma thesis different approaches for vehicle detection and tracing are introduced, implemented and compared with each other. The necessity for vehicle detection and tracing is that the traffic can be evaluated, which allows an intelligent traffic management.

Another use is better safety on the streets by monitoring the traffic and sending information to the vehicles. Therefore assistant systems, which are build in the vehicles, receive information about the road regulations. This can then be used for automated fines for traffic offender. This idea can be seen in "Live In-vehicle Smart Assistant" (LISA) as part of the "Advanced Safety and Driver Support for Essential Road Transport" (ASSET-Road) project of the European Commission [LIS]. In Germany currently also a big discussion about the prevention of wrong-way driver is on-going. In Bavaria there are big warning signs, with a violent yellow color, erected on the slip roads to prevent people from taking the wrong road[3]. Other ideas are that manufacturers of navigation systems post a warning on the screen of the device if the

[1] http://www.adac.de/infotestrat/adac-im-einsatz/motorwelt/Autobahnstaubilanz.aspx
[2] http://www.singapur.diplo.de/Vertretung/singapur/de/01/Leben__und__Arbeiten/Willkommen__in__Singapur.html
[3] http://www.dvr.de/aktuelles/sonst/1965.htm

driver takes the wrong slip road as for example in the upcoming Mercedes Benz S-Class and E-Class[4].

The implementation is intended as a plug-in for the program FloDiEdi. As FloDiEdi is implemented in C++ to make use of the functionality OpenCV and the Qt framework offers, this implementation is done in C++. The OpenCV library is used here mainly for basic operations provided in this library. Implementations of the functions used in the thesis are gernerally implemented from scratch.

No personal data was gathered during filming of the video for the evaluation.

1.2 Preliminary work

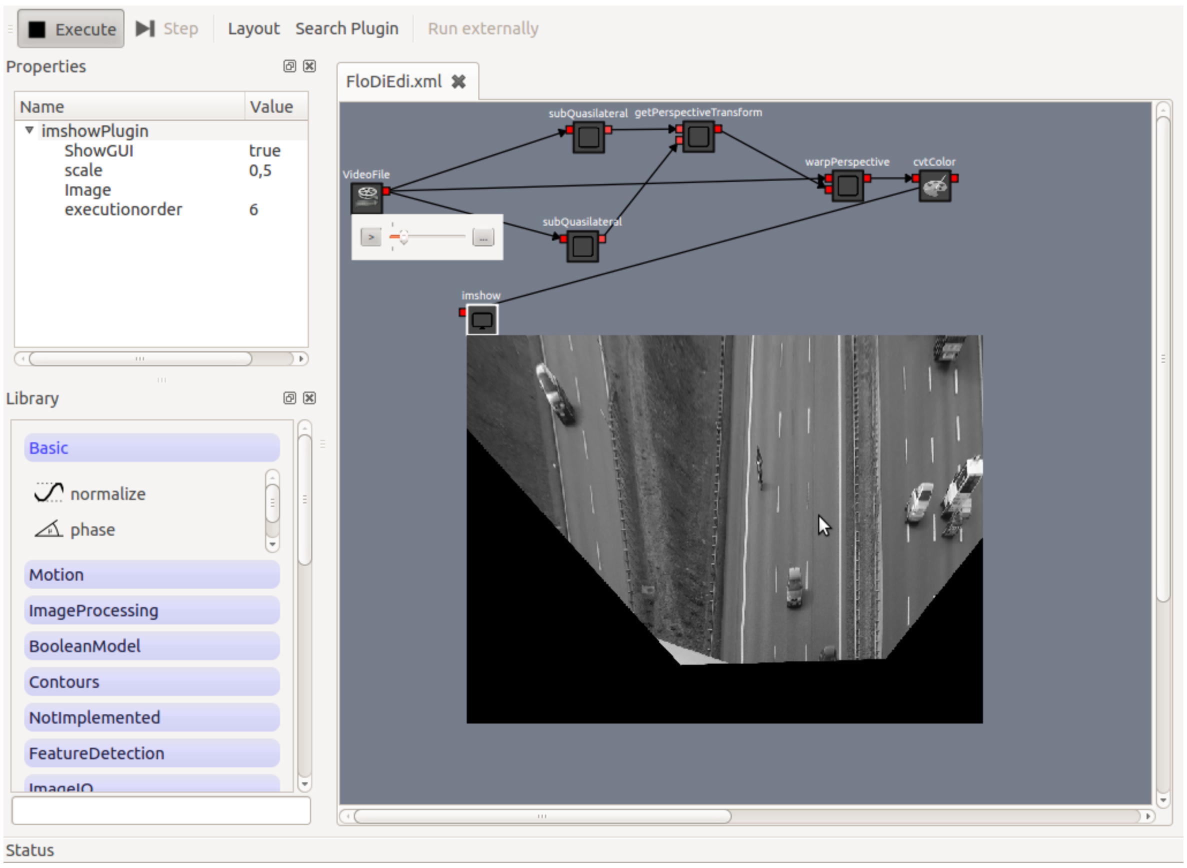
1.2.1 Flow Diagram Editor (FloDiEdi)

The program FloDiEdi[5] is developed by research fellows working at the institute for parallel and distributed systems (IPVS) of the University of Stuttgart. This program is a flow diagram editor whose main focus is on image processing. The graphical user interface allows to easily connect different quadrangle blocks whose purpose is to compute or analyze images or videos. These blocks can be grouped into functions such as:

- Load data from different sources (e.g. capture images via webcam or load variables, images or movies from a file)
- Data storage (Store data as images or matrix values as numbers)
- Algorithms for image processing (Filtering, histograms, drawing, ...)
- Display data representations or images
- Tools (Cycle timer, counter, ...)

Each blocks represents an implemented plug-in which contain functions for image processing. This way FloDiEdi allows image processing in a convenient manner. FloDiEdi is similar to the image processing program Khoros developed by Khoral Research, Inc.[6]. In contrast to the commercial program Khoros, FloDiEdi is open source and free of use for everyone as it uses a GPLv3 license[7]. FloDiEdi uses the OpenCV computer vision library and the Qt framework which offer a rich library of functionality. Currently FloDiEdi supports Linux but support for Windows is planned. See figure 1.1 on page 12 for an screen shot of the FloDiEdi environment with the possible settings and functionality.

[4]http://www.daimler.com/dccom/0-5-7153-49-1567517-1-0-0-0-0-0-16694-0-0-0-0-0-0-0-0.html
[5]http://sourceforge.net/projects/flodiedi/
[6]http://www.khoral.com/
[7]http://www.gnu.org/licenses/gpl-3.0



Name	Value
imshowPlugin	
ShowGUI	true
scale	0,5
Image	
executionorder	6

Figure 1.1: FloDiEdi environment with different blocks. The result of this interconnection is a perspective transformed image and conversion to a gray scale image.

1.2.2 OpenCV

OpenCV[8] is an abreviation for Open Source Computer Vision Library which is developed by Intel® and Willow Garage[9]. It is released under the BSD license[10] which allows free commercial and academic use. OpenCV has interfaces for Java, Python, C and C++. There is official support for a number of operation systems such as Windows, Linux, Mac OS, iOS and Android but also ports exist for Blackberry 10[11] or maemo[12]. The focus is on efficient and real-time applications. It contains a lot of functionality from multiplication of matrices and scalars to tracking of objects. OpenCV covers functionality[13] for:

[8] http://opencv.org/
[9] http://www.willowgarage.com/
[10] http://opensource.org/licenses/bsd-license.php
[11] https://github.com/blackberry/OpenCV
[12] https://garage.maemo.org/projects/opencv
[13] http://docs.opencv.org/modules/core/doc/intro.html

- Core functionality, including data structures.
- Image processing which covers filtering, transformations, histograms, etc.
- Video analysing module for motion analysis, image subtraction and object tracking.
- 3D module including 3D reconstruction.
- Feature detection, description and matching in 2D.
- Object detection for various and predefined (eye, hand, face, ...) objects.
- Interface for image and video capturing as well as support for different codecs.
- GPU module for taking advantage of GPU accelerated algorithms.

The OpenCV version used for developing was 2.4.2 which was released on July 4th, 2012.

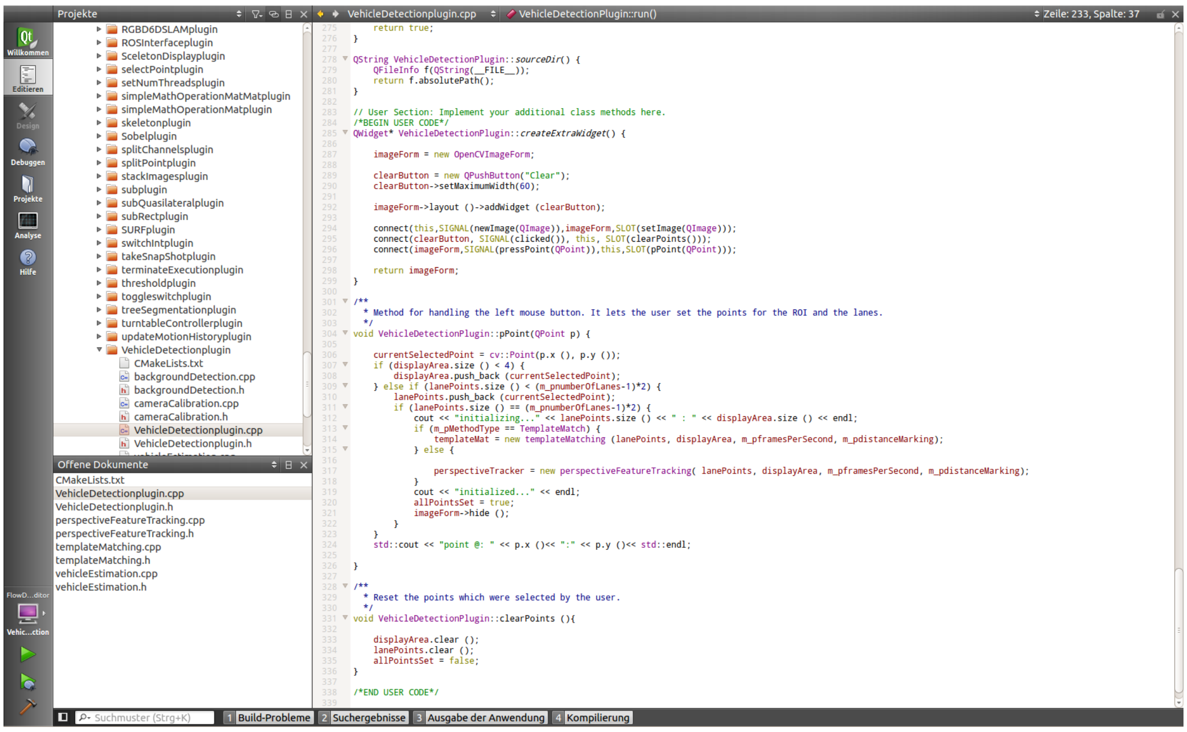
1.2.3 Qt

On the homepage of Qt it is described as a cross-platform application and UI framework for developers using C++ or QML, a CSS & JavaScript like language[14]. A short overview of some modules that are offered by Qt:

- Core functionality like basic data types (strings, integer, ...), file writing and reading and more.
- GUI components such as buttons, mouse events or draw functions.
- Network module for programming of TCP, UDP or SSL connections.
- OpenGL module for graphic programming
- Database module to initialize und utilize a SQL database
- Multimedia module for video and audio playback or transformation of audio streams.
- and more modules[15].

Qt offers also an integrated development environment (IDE) that is called Qt Creator. It offers rich functionality for development and debugging, see Figure 1.2. Shortcuts for many functions are available to increase the efficiency and offer an easy to handle environment. Qt uses the GPLv3 and LGPLv21 licenses[16]. The Qt version used is 4.8.0.

[14] http://qt.digia.com/
[15] http://doc.qt.digia.com/stable/modules.html
[16] http://qt.digia.com/Product/Licensing/



```
    return true;
}

QString VehicleDetectionPlugin::sourceDir() {
    QFileInfo f(QString(__FILE__));
    return f.absolutePath();
}

// User Section: Implement your additional class methods here.
/*BEGIN USER CODE*/
QWidget* VehicleDetectionPlugin::createExtraWidget() {

    imageForm = new OpenCVImageForm;

    clearButton = new QPushButton("Clear");
    clearButton->setMaximumWidth(60);

    imageForm->layout ()->addWidget (clearButton);

    connect(this,SIGNAL(newImage(QImage)),imageForm,SLOT(setImage(QImage)));
    connect(clearButton, SIGNAL(clicked()), this, SLOT(clearPoints()));
    connect(imageForm,SIGNAL(pressPoint(QPoint)),this,SLOT(pPoint(QPoint)));

    return imageForm;
}

/**
  * Method for handling the left mouse button. It lets the user set the points for the ROI and the lanes.
  */
void VehicleDetectionPlugin::pPoint(QPoint p) {

    currentSelectedPoint = cv::Point(p.x (), p.y ());
    if (displayArea.size () < 4) {
        displayArea.push_back (currentSelectedPoint);
    } else if (lanePoints.size () < (m_pnumberOfLanes-1)*2) {
        lanePoints.push_back (currentSelectedPoint);
        if (lanePoints.size () == (m_pnumberOfLanes-1)*2) {
            cout << "initializing..." << lanePoints.size () << " : " << displayArea.size () << endl;
            if (m_pMethodType == TemplateMatch) {
                templateMat = new templateMatching (lanePoints, displayArea, m_pframesPerSecond, m_pdistanceMarking);
            } else {

                perspectiveTracker = new perspectiveFeatureTracking( lanePoints, displayArea, m_pframesPerSecond, m_pdistanceMarking);
            }
            cout << "initialized..." << endl;
            allPointsSet = true;
            imageForm->hide ();
        }
    }
    std::cout << "point @: " << p.x ()<< ":" << p.y ()<< std::endl;

}

/**
  * Reset the points which were selected by the user.
  */
void VehicleDetectionPlugin::clearPoints (){

    displayArea.clear ();
    lanePoints.clear ();
    allPointsSet = false;
}

/*END USER CODE*/
```

Figure 1.2: Qt Creator IDE which offers functionality for development, debugging and analysis of C++ or QML code. On the left are the options to switch between projects, build the project, etc . On the right side is the window of the editor for programming. The top left window is a project or file browser. The lower left window shows the open files.

Structure of the thesis

This thesis is structured to give first a general introduction to image processing before discussing and evaluating the approaches for vehicle detection and tracing. The fundamentals are stated in chapter 2 on page 15. This chapter covers the general internal build of cameras, the procedure of image aquisation and basic techniques used in image processing. In chapter 3 on page 31 are the methods and algorithms stated which are used in the thesis to solve the task of vehicle detection and tracing. Chapter 3 is divided into the different levels of extracting knowledge from images. To each level the corresponding method are assigned. The chapter 4 lists and explains the three approaches that are used to detect and trace vehicles. The preparational work, which is done beforehand, is also part of this chapter. In chapter 5 on page 66 the the different approaches are evaluated. Evaluated are the successful detected vehicles, the tracking of the vehicles and the type estimation. In chapter 6 is an an overview about the results of the approaches given. Furthermore the personal opinion is stated and an outlook for related work is given. In the appendix are the instructions for using the plug-in.

2 Fundamentals

In this chapter an introduction into the image retrieving process and fundamentals of image processing, in regard of the thesis, are presented.

2.1 Camera hardware

2.1.1 Digital image capturing

The camera model used for analog and digital images is a pinhole camera, see Figure 2.1. The pinhole camera model serves as a general approach to describe how cameras work. The area that is going to be captured is focused through the lense at the camera. Capturing digital

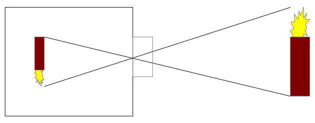


Figure 2.1: Pinhole camera as a model for cameras which captures images with a lens.

images is quite similar to capture analog images. When using analog techniques the camera opens the shutter of the lens and lets light pass through to expose the photographic film. The image is then captured on the photographic film.
Whereas in digital image capturing, the lens shutter is also opened but the light does not incidence on a photographic film but a sensor. There are two general cases how the image is captured. In the first, the light is exposed on a single sensor which has a RGB (**R**ed, **G**reen and **B**lue) matrix with a specific pattern of photodiods installed. Figure 2.2 shows an arrangement of the photodiods in an Bayer filter [Bay76]. This pattern is built similar to the human retina. An big disadvantage of single sensor camers, especially for image processing tasks, is that the colors of the image are interpolated during the processing of the pixels. This can lead to problems when detecting small objects. The interpolation process can make a white line with a green background into a yellowish color at the border of the line. If the image is probed

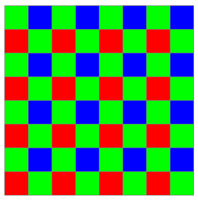


Figure 2.2: Arrangement of photodiods in the way a Bayer filter is built.

for yellow objects, yellow objects will be detected at the line. But those detected objects are not there in reality, which makes it necessary to deal with this problem. Therefore it is very important that the source image is of good quality with low distortions.
In the second case there is a sensor with photodiods for each primary color. Therefore the light is splitted by a prism into the three primary colors. Now each sensor captures only its corresponding color. The purpose of this is to achieve a higher light exposure. By this method the colors of the image become more saturated and more precise. Also when the illumination is bad the captured images have still a good quality with relatively low noise. A principal configuration of the capturing is in Figure 2.3. The camera, a Panasonic HDC-HS700 which was used for capturing the images of the street, has three sensors and a resolution of 1920×1080 [Pan10].

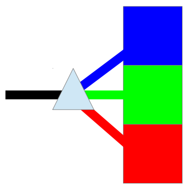


Figure 2.3: Principal functionality of a camera with three sensors. The light is splitted into eacher primary color for each of sensors.

The voltage emitted by the photodiods represents the value of a pixel. This is done by sampling and digitizing the values of the signal. Sampling is a techniques for measuring in steps of Δt. Digitizing is the process of forming analog values to discrete digital values or digital numbers. This is not limited to a fixed sampling time, as it might be beneficial if it varies over time on special tasks. A conversion of an analog-ditigal conversion of a signal is in Figure 2.4.

2.1.2 Camera internals

To work with cameras it is necessary to have information about the camera. Especially if it is not only used for taking images without further processing. A camera can be described by

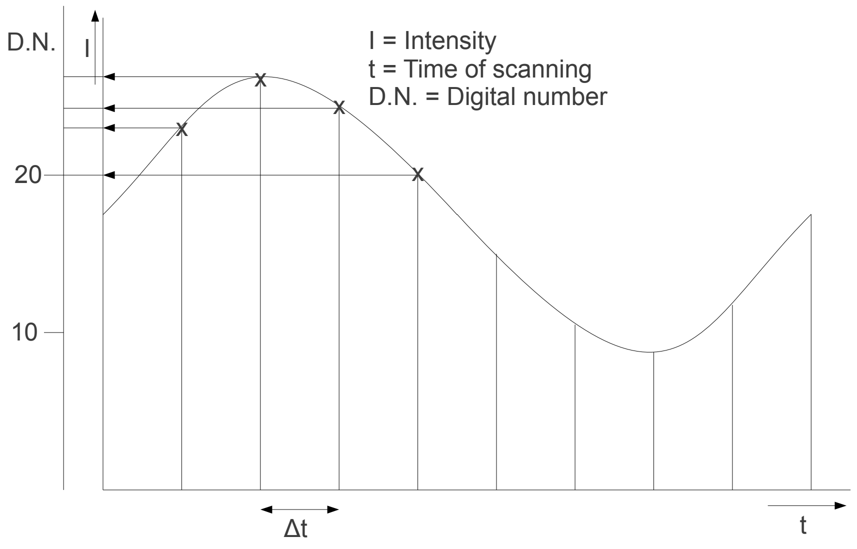


Figure 2.4: An analogue to digital conversion. Sampling at every time Δt and discretisation of the intensity values.

intrinsic and extrinsic parameters. For an optimal calibration and accurate measurements it is mandatory that those parameters are known.
The camera matrix, which describes the intrinsic parameters, is shown in Equation 2.1.

$$K = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \tag{2.1}$$

It is necessary to calculate the parameters as manufacturer generally do not provides these matrices. The parameters f_x and f_y can be calculated with the equation in Equation 2.2 and Equation 2.3.

$$f_x = F \cdot s_x \tag{2.2}$$

$$f_y = F \cdot s_y \tag{2.3}$$

In the equations is F the focal length in mm and s_x and s_y are scaling parameters. The calculation of these scaling factors can be seen in Equation 2.4 and Equation 2.5.

$$s_x = \frac{(Image_{width})}{(Sensor_{width})} \tag{2.4}$$

$$s_y = \frac{(Image_{height})}{(Sensor_{height})} \tag{2.5}$$

The unit of s_x and s_y is $\frac{px}{mm}$. The other two parameters, to create a camera matrix, are values of the principal point which are in u_0 and v_0. The principal point in a perfect camera would be at the center of the image. In reality this is mostly not possible due to imprecise assembling. To fill in these values for an initial guess, the center can be used but the exact point has to be calculated.
Now the intrinsic camera matrix M can be created. As concerning distortions there are two types which have the main impact. These are the radial and the tangetial distortion [BK08]. The effects of radial distortion can be seen in Figure 2.5. Radial distortion is corrected by Equation 2.6.

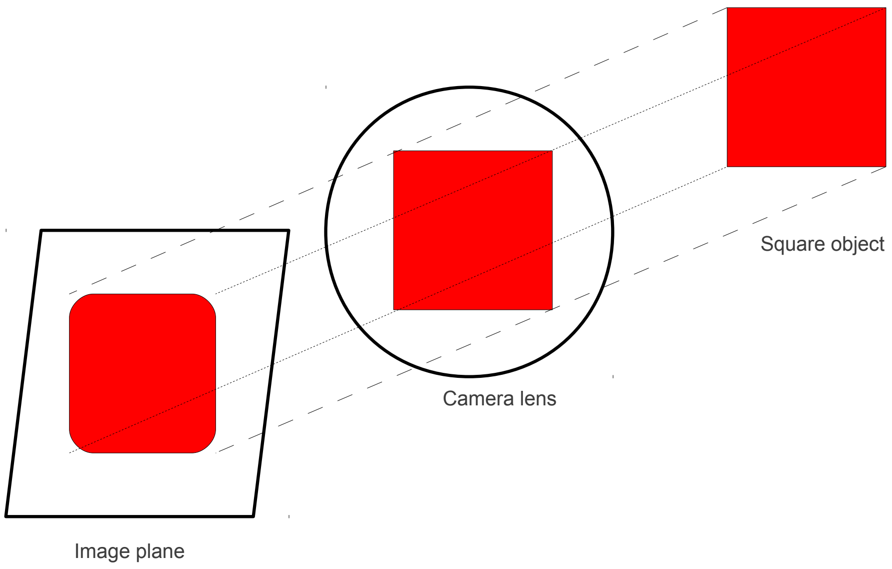


Figure 2.5: By radial distortion a square object can loose its sharp edges and have a rather round shape. Due to the shape of the lens the rays get bent more as the rays that are further away from the center.

$$\begin{aligned} x_{corrected} &= x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\ y_{corrected} &= y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \end{aligned} \tag{2.6}$$

Tangential distortion is removed by Equation 2.7.

$$\begin{aligned} x_{corrected} &= x + (2p_1 y + p_2(r^2 + 2x^2)) \\ y_{corrected} &= y + (p_1(r^2 + 2y^2) + 2p_2 x) \end{aligned} \tag{2.7}$$

OpenCV uses a vector representation with five coefficients to describe radial and tangetial distortion. The parameters are k_1, k_2, k_3, p_1 and p_2 where the k_i is used for the radial distortion and p_i is for the tangetial distortion.

As stated above the camera also has extrinsic paramers. These parameters are a rotation matrix R and a translation matrix T. The extrinsic parameters are necessary if the origin of the camera in 3D space needs to be calculated. The rotation matrix states the rotation of the camera and the translation matrix the translation of the camera. To calculate the origin C see equation 2.8. For rotation matrices it is true that $R^{-1} = R^T$, see [MW06].

$$C = -R^{-1} \cdot T = -R^T \cdot T \tag{2.8}$$

To calculate the origin of a camera another possible method is to solve the perspective-n-point problem using RANSAC as suggested by Bolles and Fischler in [BF81].

2.2 Image processing fundamentals

2.2.1 Frequency domain and spatial domain

The definition of spatial domain by Gonzalez and Woods is “The term spatial domain refers to the aggregate of pixels composing an image. Spatial domain methods are procedures that operate directly on the pixels.” [GW92]. The advantage of using spatial domain is that the results of the operations can visually be evaluated and the viewer can decide whether the resulting image is good or not. But it has to be kept in mind that for further processing it might be advantageous to optimise it for machine recognition and not for the human viewers.
The frequency domain is the domain for signals with respect to the frequency. The definition for spatial frequency by Jensen is “Spatial frequency is the number of changes in brightness value per unit distance for any part of an image.” [JLR87]. The unit can be 1/pixel, 1/mm, etc . About the frequencies and their characteristics it can be said that high frequencies are often details or noise in images because there are many changes between the brightness values. Whereas low frequencies are areas with the same brightness. This is because in areas with the same brightness there is no or low change to the brightness.
An example how a signal looks in the frequency domain and in the spatial domain is in Figure 2.6. To transform the one-dimensional image from the spatial domain to the frequency domain fourier transformation is used. See Equation 2.9 for the formula of one-dimensional images. One-dimensional images can be transformed from the frequency domain to the spacial domain by using the inverse fourier transformation in Equation 2.10. For the two-dimensional case the formula for fourier transformation is in Equation 2.11 and the inverse fourier transformation in Equation 2.12. This transformation does not lose information by changing from one domain into another.

$$F(u) = \int_{-\infty}^{\infty} f(x)\ e^{-2\pi i x u}\, \mathrm{d}x \tag{2.9}$$

$$f(x) = \int_{-\infty}^{\infty} F(u)\ e^{2\pi i u x}\, \mathrm{d}u \tag{2.10}$$

$$F(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\ e^{-2\pi i(ux+vy)}\, \mathrm{d}x\, \mathrm{d}y \tag{2.11}$$

$$f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(u,v)\ e^{2\pi i(ux+vy)}\, \mathrm{d}u\, \mathrm{d}v \tag{2.12}$$

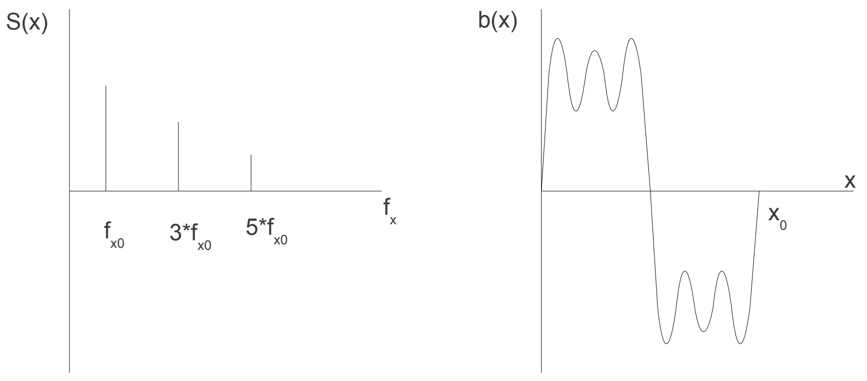


Figure 2.6: An signal shown in the frequency domain on the left with the three frequencies and their amplitude. The same signal in the spatial domain on the right where the three frequencies are overlain by each other resulting in this image.

The first contribution in this field was done by Jean-Baptiste-Joseph Fourier. He stated that “each periodic function can be expressed as the sum of sines and/or cosines of different frequencies, each multiplied by a different coefficient.” [GW92] which is now known as the fourier series. A well known and often used algorithm for transformation from spatial domain to frequency domain is the **F**ast **F**ourier **T**ransform (FFT) algorithm by Cooley and Tukey [CT65]. Linear filtering is similar to a concept called convolution in the frequency domain. In the spatial domain it is therefore often referred to as –convolving a mask with an image– hence named a convolution mask.

2.2.2 Region of interest

The region of interest (ROI) is used to focus on a specific part of an image. An image can have different ROI’s to reduce the amount of pixels which are processed. The ROI in this diploma thesis is the street in the center of the image. Other parts like the streets in parallel do not to be processed and can therefore be ignored, thus gaining faster processing of the image. See Figure 2.7 for the use of a selected region of interest. The difference between the region of interest and the whole image is in this example over twice the size of the marked region. So only a small part of all pixels needs to be processed.

2.2.3 Features

Generally spoken a feature is a point which is of interest for processing. Features can be edges, corners or blobs. Therefore for finding features corresponding methods such as edge detectors like the canny filter [Can86], corner detectors like Tomasi-Shi feature detector [ST94] or a blob detector such as the Laplacian of Gaussian [Bur82] can be used. Features can also be used for

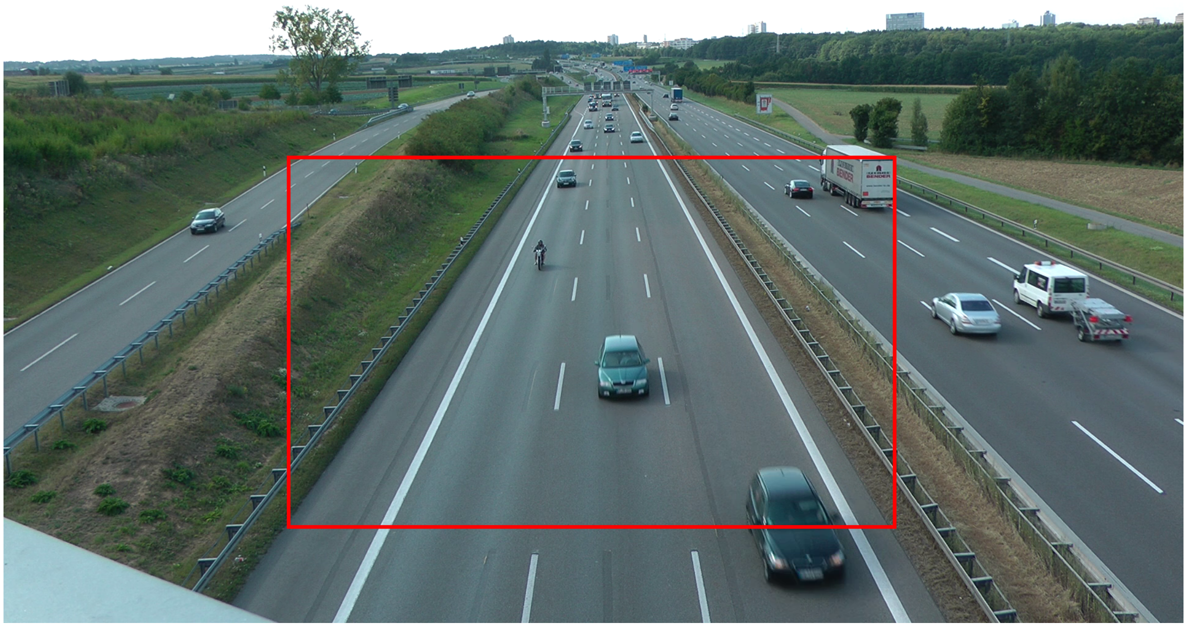


Figure 2.7: The region of interest is selected in an image of a street. The part of the image which is out of the focus does not need to be processed so unnecessary computational time can be reduced.

tracking as these features, if in the next or other image still present, can be found again. This works reliable if the features are good features. Whether a feature is good has to be defined first. The neighbourhood of the pixel can help to make such a decision. Depending on the user-defined neighbourhood it can contain several pixels connected to the center pixel. In Figure 2.8 two common neighbourhood types, the four- and eight-neighbourhood, are shown. Imagine a black pixel whose eight-neighbourhood is completely white. Assume that this constellation is fixed, this point can be found again in other related images.

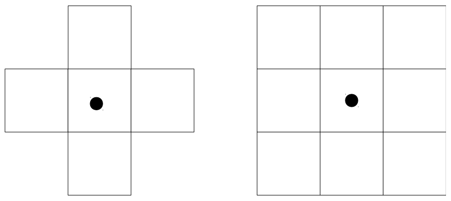


Figure 2.8: On the left side the four-connected neighbourhood and on the right side the eight-connected neighbourhood is shown. The black dot marks the center pixel.

2.2.4 Structural element

A structural element is used in mathematical morphology as a shape or form for processing an image. The size of a structural element is normally symmetrical $N \times N$ where N is an odd number. Chosing an odd number makes it possible to know exactly where the center or anchor point in the structural element is. Knowing where the center is, is needed for the allignment of the structural element to the original image. The structural element does not need any numerical values for processing. The size or shape is the object of interest as this is used for the transformation [GW92]. When the size is $N \times M$ the resulting image can be reshaped that the objects do not resemble the origin object anymore. See Figure 2.9 for dilation performed on a binary image with an $N \times N$ structural element. In Figure 2.10 dilation is applied with a $N \times M$ structural element to a binary image. This is not necessarily a problem if an advantage can be drawn out of the resulting reshaping of the object.

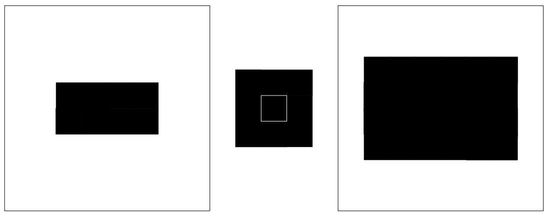


Figure 2.9: On the left is the original binary image. In the middle is the $N \times N$ structural element. The right shows the resulting image when performing dilation with the $N \times N$ structural element.

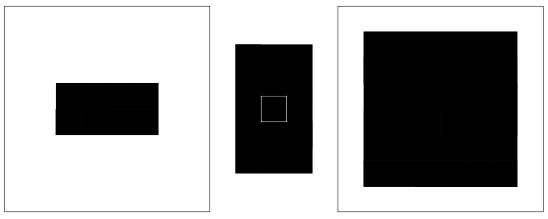


Figure 2.10: The left shows the orignal binary image. In the middle is the $N \times M$ structural element shown. On the right is the image after performing dilation with the $N \times M$ structural element.

2.2.5 Mathematical morphology

Mathematical Morphology is a tool for extracting image components that are useful for representation and description [Ser88]. The technique was originally developed by Matheron and Serra at the Ecole des Mines in Paris in the year 1998[1]. Mathematical morphology is a theoretical model for digital images. It is mostly used on binary images. The logical operations "AND" and "OR" are the basic moduls. From these the basic operations dilation and erosion are created. The definition of dilation is $A \oplus B = \{c|c = a + b$ for some $a \in A$ and $b \in B\}$. Dilation is sometimes also called Minkowski addition. The operation dilation is associative as well as commutative. This means that $(A \oplus B) \oplus C = A \oplus (B \oplus C)$. When using dilation the resulting image has all of the original "1" pixels in the image, the borders are expaned and in case there are small holes these are filled. It can be described as an logical "OR" operation in conjunction with the structural element. Erosion is defined as $A \ominus B = \{x|x + b \in \ for\ every\ b\}$. It is sometimes called Minkowski subtraction. In regard of dilation, erosion is whether associative nor commutative. The result of erosion can be described as shrinking of the object, as the boundaries of the object are etched or eroded. Erosion is in terms of logical operations a "AND" operator as only those shapes that are the size of the structural element remain. Morphological operations cover [Lev11]:

- Thinning $A \otimes B = A - (A \circledast B)$
- Thickening $A \odot B = A \cup (A \circledast B)$
- Translation $(A)_z = \{w|w = a + z$, for $a \in A\}$
- Reflection $(\hat{B} = \{w|w = -b$, for $b \in B\}$
- Complement $A^C = \{w|w \notin A\}$
- Difference $A - B = \{w|w \in a, w \notin B\} = A \cap B^c$
- Dilation $(A \oplus B) = \{z|(\hat{B})_z \cap A \neq \varnothing\}$
- Erosion $(A \ominus B) = \{z|((B)_z \subseteq A\}$
- Opening $\Rightarrow A \circ X = (A \ominus X) \oplus X$
- Closing $A \bullet B = (A \oplus B) \ominus B$
- Hit-or-miss transform $A \circledast B = (A \ominus B_1) - (A \oplus \hat{B}_2)$
- Boundary extraction $\beta(A) = A - (A \ominus B)$
- ...

Annotation to the definition of the operations:

- A is the original image
- A^c is the complement image
- A∪B is the union of the images A and B
- A∩B is the intersection of the images A and B

[1] http://cmm.ensmp.fr/~serra/apub.htm

- $A - B = A \cap B^c$ is the difference between the image A and B
- #A is the cardinality of A

The result of the morphologial operation erosion is shown in Figure 2.11. For an example of dilation refer to Figure 2.12. Opening is performed in Figure 2.13. Closing is performed in Figure 2.14. The transformed shape depends generally on the structural element, the original shape and the performed operation.

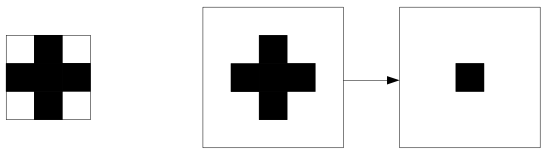


Figure 2.11: Erosion is applied on an image. On the left side is the structural element, in the middle is the original image and the eroded image is on the right.

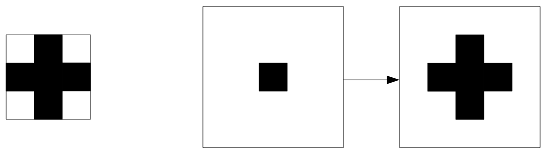


Figure 2.12: Dilation is applied on an image. On the left side is the structural element, in the middle is the original image and the dilated image is on the right.

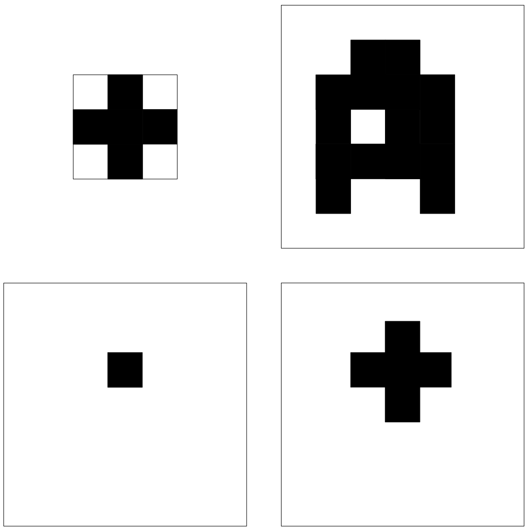


Figure 2.13: Opening performed on an image. The structural element is depicted at the upper left and the original image is at the upper left. Erosion of the original image is shown in the lower left. The lower right image shows the dilation of the eroded image which is the result of the operation.

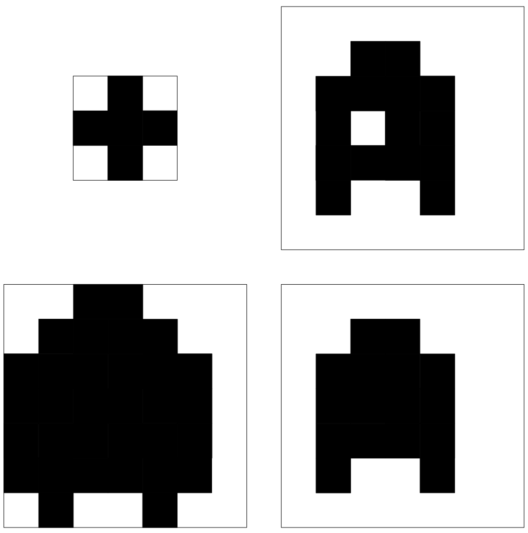


Figure 2.14: Closing performed on an image. The structural element is depicted at the top left. The original image is at top right. On the lower left is the dilated version of the image The lower right shows the final result.

2.2.6 Information and knowledge

There is a distinction between information and knowledge. Even though in colloquial language it is sometimes not distinguished and the two words are used with the same meaning. In the oxford dictionary information means "facts provided or learned about something or someone"[2]. The definition "Data only becomes 'information' when it is transmitted" is taken from [Ker11]. Therefore information, in terms of image processing, can be the intensity of a pixel value, the color of a pixel, the speed of a vehicle, the amount of vehicles on the street,.... These are just a few examples what information can be.
Whereas knowledge is explained as "facts, information, and skills acquired through experience or education; the theoretical or practical understanding of a subject"[3]. Knowledge is the combination of information with each other and reasoning. It is not just a collection of information which are somehow loosely related. "Information becomes 'knowledge' when it is interpreted" is defined in [Ker11], which is taken as the definition for this thesis. Examples for knowledge can be that the motorway is currently heavy loaded with cars or motor trucks. Most offences of the road traffic regulations such as speeding or distance offences are commited mostly on the left lanes, in regard to right hand traffic. There is a traffic jam for x meters and the resulted delay by taking this road is y minutes. The generalisation of this definition means that we can gain knowledge from information.

2.2.7 Thresholding

When thresholding is applied the values of a pixel in an image are either set to the minimal value or maximum value. The decision depends on whether the current value of the pixel is above or below the threshold value and is set correspondingly. Algorithm 2.1 shows an pseudo implementation of an binary threshold algorithm.

Algorithm 2.1 Threshold algorithm

```
procedure THRESHOLDING($thresholdValue$, $minValue$, $maxValue$)
    for $pixel(x, y) \in Image$ do
        if $pixel(x, y) \leq thresholdValue$ then
            $pixel(x, y) \leftarrow minValue$
        else
            $pixel(x, y) \leftarrow maxValue$
        end if
    end for
end procedure
```

[2] http://oxforddictionaries.com/definition/english/information?q=information
[3] http://oxforddictionaries.com/definition/english/knowledge?q=knowledge

2.2.8 Cross ratio

Cross ratio is located in the field of linear algebra. The property of cross ratio is the correlation between points in two different spaces [Tay63]. For using cross ratio it is mandatory to have four collinear points in both spaces. These spaces are the image space and the real world space for example. The points have to have in the same distance in each space. This means that these points have to be chosen in way that the distance between each point can be calculated. Therefore all distances have to be known either in the real world or in the image space. Assume cross ratio is calculated and all the distances in the image space are known. For the real world the three points are known. The distances between the points in the image space are calculated first. Afterwards the Equation 2.13 is solved for λ. $|XY|$ in the equation stands for the distance between the point X and Y.

$$\lambda = [RSTU] = \frac{|RT|}{|RU|}\frac{|SU|}{|ST|} = \frac{|rt|}{|ru|}\frac{|su|}{|st|} \tag{2.13}$$

As the three real world points have the same ratio of distances, which are used for calculating the cross ratio in the image space, are calculated. The equation is then transposed so that the final missing distance between two points can be calculated. Now the distance to the point is known. In figure 2.15 it becomes clearer how these points have to be located and their relationship is defined. The points r, s, t, u are in the image space, the points R, S, T,

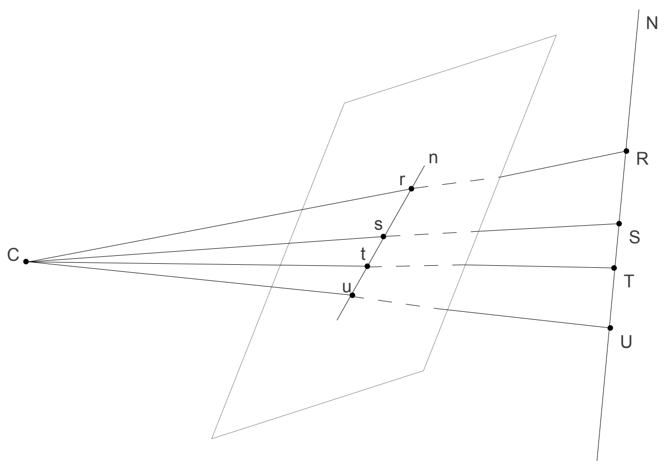


Figure 2.15: The correlation between points in two different spaces (r, s, t, u in the image space and R, S, T, U in the real world) from the origin of the camera C. This correlation is used for cross ratio which is invariant to perspective transformation.

U is in the real world space and C is the camera. One advantage of cross ratio is that the computational effort is small. Cross ratio is invariant to perspective transformations. But the downside is that the seven of eight points and their distances have to be known previously. So without knowledge about these facts cross ratio will not work. Cross ratio can be used for calculating distances in different spaces which is done in this diploma thesis.

2.2.9 Perspective transformation

When taking images of two parallel lines, the lines meet at the horizon. In the case of a view on a street it is a one-point perspective as there is one vanishing point. Marsh describes it as “Perspective projections map parallel lines in world coordinate space to parallel lines in the viewplane if and only if the lines are parallel to the viewplane.”[Mar04]. To get the lines in parallel, like it is in reality, a perspective transformation has to be performed on the image. For the perspective transformation matrix four points of the original image are needed. These points represent the perspective of the image. Also the points of the rectangular shape are needed which represent the shape of the image it is transformed to. The perspective matrix P for 2-dimensional images is a 3x3 matrix. It is populated with the coefficient for the transformation. The formula for transforming x- and y-coordinate into the respective u- and v-coordinate is in Equation 2.14, taken from [HZ04].

$$\begin{aligned} u_i &= \frac{c_0 \cdot x_i + c_1 \cdot y_i + c_2}{c_6 \cdot x_i + c_7 \cdot y_i + c_8} \\ v_i &= \frac{c_3 \cdot x_i + c_4 \cdot y_i + c_5}{c_6 \cdot x_i + c_7 \cdot y_i + c_8} \end{aligned} \tag{2.14}$$

So a point z can be transformed by using Equation 2.15.

$$w = P \cdot z \tag{2.15}$$

In the equations are x and y the original points and u and v are the transformed points. To aquire the perspective matrix the equation 2.16 is used.

$$\begin{pmatrix} x_0 & y_0 & 1 & 0 & 0 & 0 & -x_0 \cdot u_0 & -y_0 \cdot u_0 \\ x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1 \cdot u_1 & -y_1 \cdot u_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2 \cdot u_2 & -y_2 \cdot u_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3 \cdot u_3 & -y_3 \cdot u_3 \\ 0 & 0 & 0 & x_0 & y_0 & 1 & -x_0 \cdot v_0 & -y_0 \cdot v_0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1 \cdot v_1 & -y_1 \cdot v_1 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2 \cdot v_2 & -y_2 \cdot v_2 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3 \cdot v_3 & -y_3 \cdot v_3 \end{pmatrix} \cdot \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \end{pmatrix} = \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ v_0 \\ v_1 \\ v_2 \\ v_3 \end{pmatrix} \tag{2.16}$$

This equation is taken from [Svo06]. Now the perspective matrix P is built in Equation 2.17 by use of direct linear transformation and set c_9 to one.

$$P = \begin{pmatrix} c_0 & c_1 & c_3 \\ c_4 & c_5 & c_6 \\ c_7 & c_8 & 1 \end{pmatrix} \tag{2.17}$$

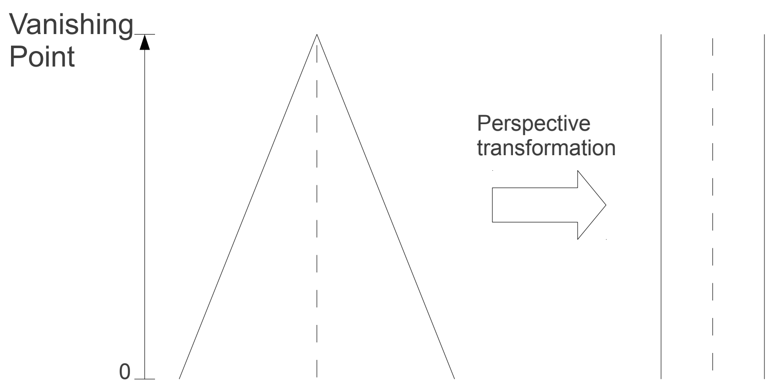


Figure 2.16: A one-point perspective view on a road with on the left where the lines meet the vanishing point. On the right is the perspective transformed result where the lines are parallel.

Setting the coefficient to one is possible as there is a total number of freedoms in a 2D perspective transformation which is eight [HZ04]. The result of a perspective transform can be seen in Figure 2.16. To solve the equation the singular value decomposition (SVD) method can be used. The SVD solving method is explained in [GVL96].

2.2.10 Image stabilisation

When taking images it is necessary to hold the camera in a fixed position to get a good result. Light shaking of the camera can be surpressed using stabilisation systems. When there is not only light shaking but heavy shaking or even camera movements, it is necessary to have a reference point or trackable feature. The reference point is needed to know in which direction the camera has moved. Assuming a camera based tracking system. If the object of interest is about to move out of the visible area the camera has to try to follow the object as long as possible. To know how much the camera has moved the reference point can be used to get the translation between images. If possible choose the reference point as a stationary point or region in the image that is never concealed. Using algorithms it is possible to think of various approaches to solve the calculation of the translation. For example a template matching algorithm can be used which searches in the image for the template. The difference between the found positions of the template in the current and the previous image is the offset of camera movement. The approach used here is phase correlation which is described in [JZWL09] and the pseudo implementation is in Algorithm 2.2. Phase correlation uses a FFT and image registration to estimate the offset between two similar images. Another possible method is the use of histograms as explained in [SM09].

Algorithm 2.2 Image stabilisation algorithm

```
procedure STABILIZEIMAGE($inputImage$)
    Set $ROI_{reference}$ and $ROI_{current}$ on $inputImage$
    $Matrix_{result}$ ←Phase correlation ($ROI_{reference}$, $ROI_{current}$)
    Get position of max value in $Matrix_{result}$
    Add offset to ROI
end procedure
```

3 Level model to extract knowledge from images

In this chapter methods for extracting knowledge from images for vehicle detection and tracing are explained. The model of how knowledge is extracted from an image is shown in Figure 3.1 which is taken from [Lev11].

3.1 Low-level preparation

In this subsection methods for low-level preparations are stated and explained. The methods here are noise filtering, image subtraction and mathematical morphologial.

3.1.1 Noise filtering with Gaussian low pass filter

Noise occurs in the process of capturing an image. When looking at noise in the frequency domain, it is visible as high frequency which might not be directly obvious to the observer. In the spatial domain, noise is visible to the observer directly. An example of a noisy image in the spatial domain of the famous Lena[1] can be seen in Figure 3.2. Noise is for people not a big problem as it can still be recognized what is on the picture up to a relatively high level of noise. For computers on the other hand it can be very problematic. For slightly low noise in images a gaussian low pass filter can be used to smoothen the image and get rid of the noise. By smoothening the image losses its sharpness, depending on the strength of the smoothening. The gaussian filtering works with a kernel that passes every pixel and for each pixel it sets a new value which is calculated by the kernel. The handling of the borders can be choosen freely but one has to think of how this will affect the pixels in the border region. The gaussian filter works like a moving window. It is moved over the whole image and each existing value of each pixel is new calculated. The moving window is also called a convolution mask. For a calculation of a pixel value refer to equation Equation 3.1.

$$Pixel_{x,y} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} Pixel_{x-i+k,y-j+k} \cdot Mask_{i,j} \tag{3.1}$$

An application of a gaussian low pass filter can be seen in Figure 3.3. In the spatial domain, noise filtering is a neighbourhood operation. Jähne describes it as “A neighbourhood operator

[1]Copyright ©Playboy Enterprises, Inc. Image taken from http://sipi.usc.edu/database/download.php?vol=misc&img=4.2.04

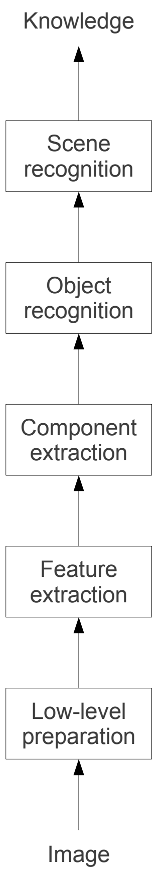


Figure 3.1: The level-model of getting from the image to knowledge. The infomation on each ascending level gets more precise.

N, connected by suitable operations, links the value in the neighbourhood to a point and write the result back at the point. This operation is done for alle points of the signal."[Jäh05].

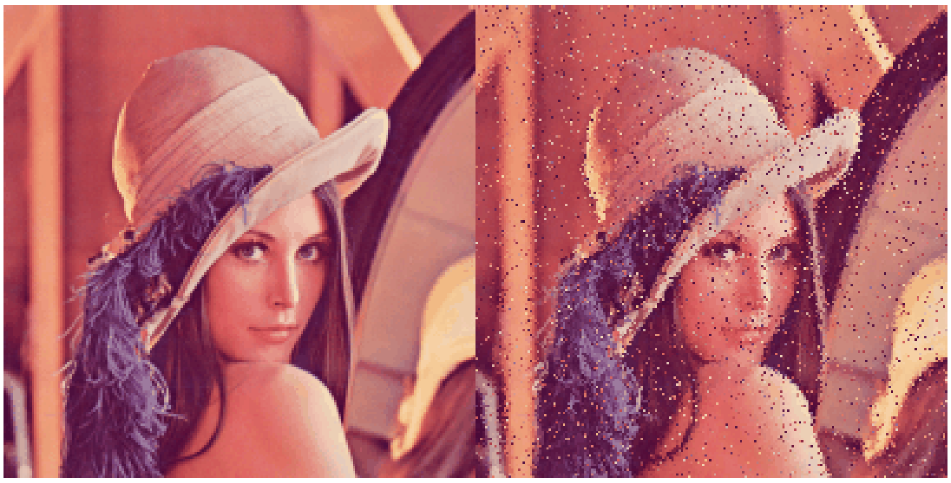
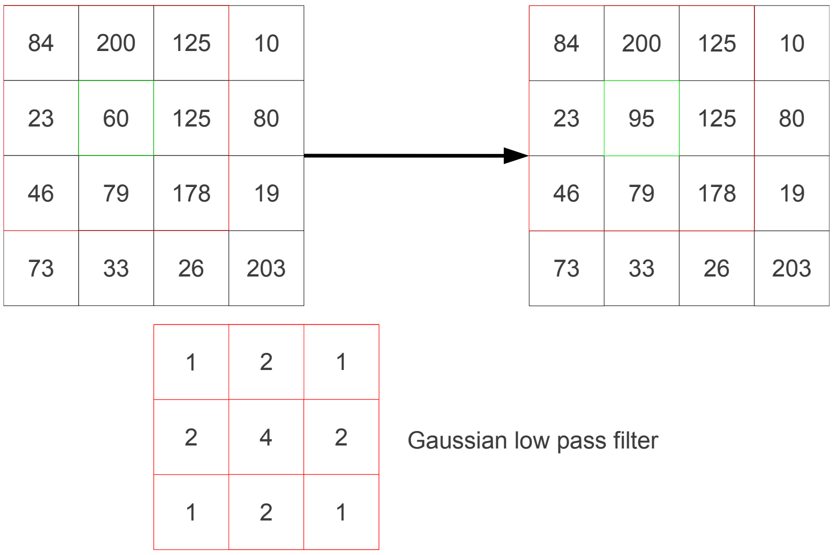


Figure 3.2: The original image of Lena on the left. On the right side is the same image but distorted with noise.



84	200	125	10
23	60	125	80
46	79	178	19
73	33	26	203

84	200	125	10
23	95	125	80
46	79	178	19
73	33	26	203

1	2	1
2	4	2
1	2	1

Figure 3.3: A 3x3 Gaussian low pass filter is applied to an pixel. The image before filtering is on the left and the image after filtering is on the right. The used Gaussian low pass filter is at the bottom.

3.1.2 Image subtraction

Image subtraction is an operation where a image is subtracted by another. Thinking about a static scene with moving objects in the foreground. The background stays the same over the whole time when changes in illumination are neglected. The principle of the extraction of the foreground objects can be done by subtracting the foreground from the background, see Equation 3.2.

$$Image_{Foreground} - Image_{Background} = Image_{Segmented} \tag{3.2}$$

The resulting image is not clearly revealing the difference of the images. Even slight changes in illumination can result in unusable results if it is not treated properly. This means further processing has to be done. This operation can be combined by binarization with thresholding followed by morphologial operations. By threshold the pixels with low values can be filtered out and shadows casts can be reduced. The keypoint is to choose the threshold level thoughtful. After the binarization the image can be processed by eriosion, dilation or the combination of them in opening or closing, see subsection 2.2.5 for details. With morphologial closing holes in the image get filled where in contrast opening lessens the remaining pixels. An illustration of image subtraction result is in Figure 3.4.

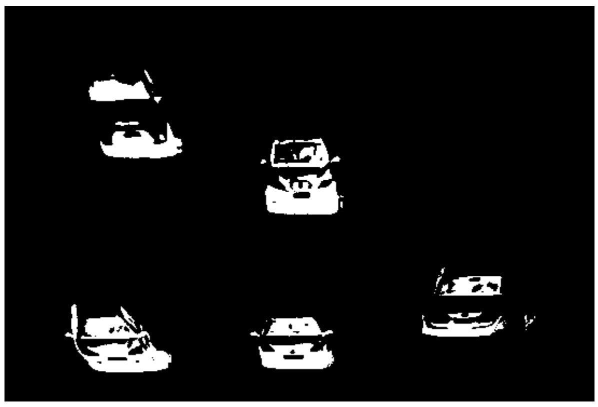


Figure 3.4: The result of the image subtraction of the current image from the already aquired background image. The rough contour of the vehicles can already be seen.

3.2 Feature extraction

The term features is explained in subsection 2.2.3. How features can be detected is explained in this section. Feature extraction can be used to make objects trackable. A use for features is also the object recognition.

3.2.1 Shi-Tomasi feature detector

For feature detection the algorithm proposed from Shi-Tomasi [ST94] can be used. The functionality of this algorithm will now be explained in detail. As explained in the fundamentals there are different features which can be detected. This algorithm is a corner feature detector. The algorithm calculates the gradients in x- and y-direction of the image by solving Equation 3.3.

$$\begin{aligned} I_x &= I \cdot G_y \cdot G_x \frac{\partial}{\partial x} \\ I_y &= I \cdot G_x \cdot G_y \frac{\partial}{\partial y} \end{aligned} \tag{3.3}$$

Afterwards the matrix L is created and calculated in Equation 3.4.

$$L = \begin{pmatrix} I_x^2 & I_x \cdot I_y \\ I_x \cdot I_y & I_y^2 \end{pmatrix} \tag{3.4}$$

In the next step the eigenvalues λ_1 and λ_2 are calculated and the maximum and the minimum is taken. Shi-Tomasi uses the Equation 3.5 to decide whether a point is a good feature or not.

$$Pixel_{x,\ y} > minValue + (maxValue - minValue) \cdot QualityVariable \tag{3.5}$$

The value of the "quality variable" is in the range from zero to one. If the value is near one the features are better and are more likely to be found again in images. This means generally less found features but more stable ones. The pseudocode implementation can be seen Algorithm 3.1.

Algorithm 3.1 Feature detection algorithm

```
procedure FeatureDetection($Input_{Image}$, $Vector_{features}$)
    $Matrix_{Eigenvalue}$ ← Eigenvalues from $input_{Image}$
    $min$ ← Minimum value of $Matrix_{Eigenvalue}$
    $max$ ← Maximum value of $Matrix_{Eigenvalue}$
    for $row = 0$ to $Input_{rows}$ do
        for $col = 0$ to $Input_{columns}$ do
            if $MatrixEigen$(row, col) > min + (max − min) · $QualityLevel$ then
                $Vector_{features}$ ← Point(row, col)
            end if
        end for
    end for
end procedure
```

3.2.2 Lucas-Kanade feature tracker

As the vehicles need to be tracked to make an assumption about the speed or wrong-way driving a method for tracking is needed. The tracking was done with an extended Lucas-Kanade tracker [TK91] in an implementation by Jean-Yves Bouguet [Bou00]. Algorithm 3.2 shows a pseudo implementation of the feature tracker implementation by Bouguet.

This algorithm by Lucas-Kanade uses optical flow and works successfully if the requirements are fulfilled. These are:

- The brightness of the pixels in the neighbourhood are constant
- The displacement is small
- Aperture is not a problem

The advantage of the pyramidical Lucas-Kanade implementation, in contrast to the original algorithm, can deal with bigger displacements. As the word pyramidical implies it uses a resized version of the original image in a pyramidic like shape. This is visualized in Figure 3.5.

Algorithm 3.2 Pyramidical implementation of the Lucas-Kanade tracker

```
procedure FEATURETRACKER($inputImage$, $previousImage$, $u_L$)
    Build the pyramid for $inputImage$ and $previousImage$ with: $(Size\ of\ the\ image)/2^L$
    Pyramidical guess: $g^{L_m} = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$
    for $L = L_m$ to 0 do
        Locate point $u_L$ on image: $I_L$: $u^L = \begin{bmatrix} p_x & p_y \end{bmatrix}^T = u_L/2^L$
        Derive image $I_L$ with respect to x: $I_x(x,y)$
        Derive image $I_L$ with respect to y: $I_y(x,y)$
        Calculate spatial gradient G:
        $G = \sum_{x=p_x-w_x}^{p_x+w_x} \sum_{x=p_y-w_y}^{p_y+w_y} \begin{bmatrix} I_x(x,y)^2 & I_x(x,y)I_y(x,y) \\ I_x(x,y)I_y(x,y) & I_y(x,y)^2 \end{bmatrix}$
        Initialize of iterative vector: $\overline{v}^0 = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$
        for ($k = 1$ to $K$ do
            Image difference: $\delta I_k$
            Mismatch vector: $\overline{b_k} = \sum_{x=p_x-w_x}^{p_x+w_x} \sum_{x=p_y-w_y}^{p_y+w_y} \begin{bmatrix} \delta I_k(x,y)I_x(x,y) \\ \delta I_k(x,y)I_y(x,y) \end{bmatrix}$
            Optical flow: $\overline{\eta}^k = G^{-1} \cdot \overline{b_k}$
            Guess for next iteration: $\overline{v}^k = \overline{v}^{k-1} + \overline{\eta}^k$
        end for
        Final optical flow on level L: $d^L \overline{v}^k$
        Guess for the next level L-1: $g^{L-1} = \begin{bmatrix} g_x^{L-1} & g_y^{L-1} \end{bmatrix}^T = 2(g^L + d^L)$
    end for
    Final optical flow vector: $d = g^0 + d^0$
    Location of point on J: $v = u + d$
end procedure
```

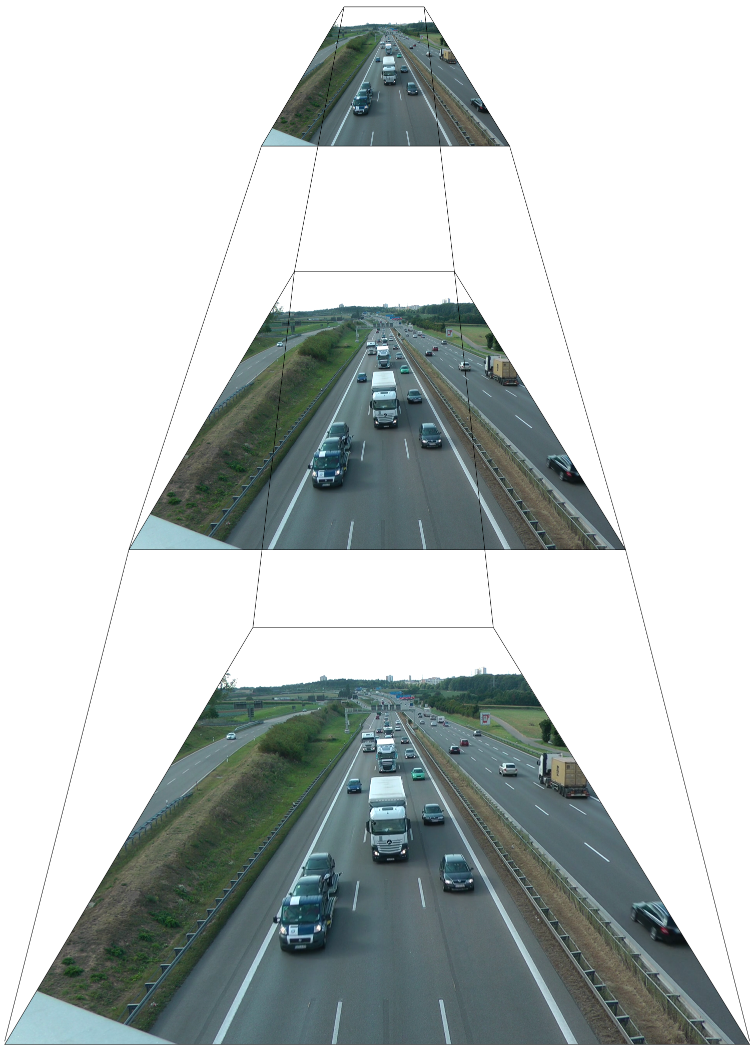


Figure 3.5: Pyramidical view of the original image at the bottom and the corresponding resized versions with a scaling factor of two between each level. Here are three levels displayed.

3.3 Object extraction

In this subsection methods for object extraction will be explained in detail for later use in the vehicle detection and tracing method.

3.3.1 Connected-component labeling

The connected-component labeling algorithm is used to group pixels which are connected in a neighbourhood. For an example of a four-connected neighbourhood refer to Figure 2.8. The algorithm works on binary images. On a sidenote, there must be the possibility to set the pixel values for example in a range from 0-255. If a higher range is choosen then more connected components can be found and distinguished. How many components will be found in the image has to be pre-estimated, when in doubt a higher range is preferable. Before starting, the values of the binary image have to be set to either zero or one. If it is not already done a threshold algorithm can be used. The pixels which are white are then set to the value one instead of 255. This is useful as the algorithm takes this as given and takes advantage of this knowledge. When the procedure is called, a counter is initialized with the value of two. When iterating the pixels in the matrix it will be skipped if the value is not one. The one indicates that it is an unprocessed pixel in the image. If the value of a pixel is zero then the pixel is not of interest and is skipped. Pixels with other values than zero or one are already processed and can also be skipped. If an pixel with the value one is found this pixel and all the pixels in the neighbourhood with the value one are set to the value of the counter. This is done recursively done for all the pixel in the neighbourhood. After assigning the value the counter is increased by one and the algorithm continues to iterate over the image. In Figure 3.6 the steps in the labeling algorithm can be seen. The connected-component labeling algorithm can be seen at Algorithm 3.3.

Algorithm 3.3 Connected-component labeling

```
procedure CONNECTEDCOMPONENTLABELING($inputImage$, $regionVector$)
    $regionCounter \leftarrow 2$
    for $row = 0$ to $inputImage_{rows}$ do
        for $column = 0$ to $inputImage_{columns}$ do
            if $inputImage_{row,column} = 1$ then
                $Value\ of\ pixels\ in\ connected\ region \leftarrow regionCounter$
                $regionVector \leftarrow$ connected points in region
                $regionCounter++$
            end if
        end for
    end for
end procedure
```

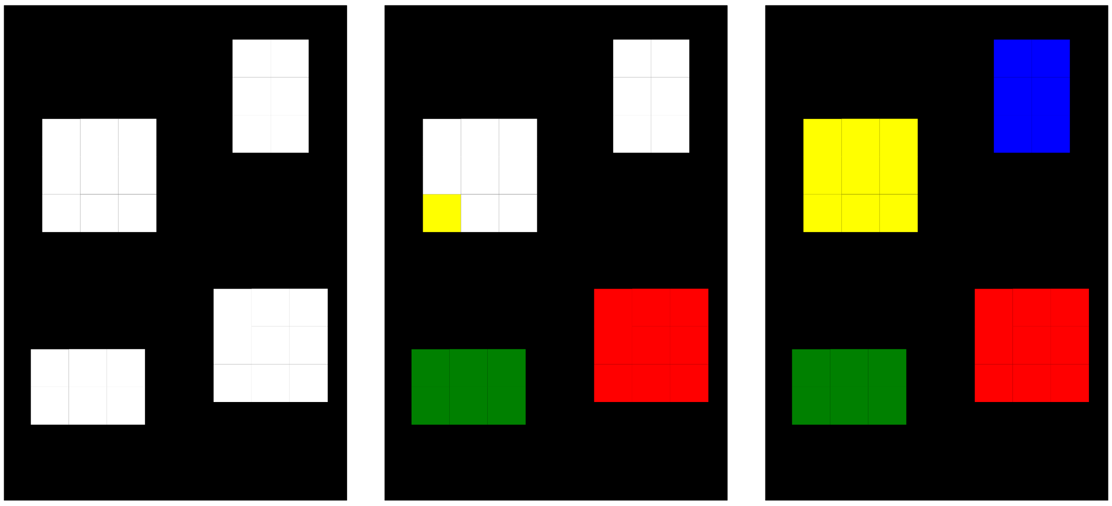


Figure 3.6: An example of the connected-component labeling algorithm. On the left is the unprocessed binary image. The middle shows the algorithm during processing and on the right is the result. The colors are only used for the purpose of visibility.

3.3.2 Template matching

Template matching is used for finding a template T in an image. The template needs to be smaller than the image and it is moved across the image to check every pixel. By moving over the image for each pixel, except the border pixels or where the template does not fit completly, a matching is calculated. To match templates there are different matching methods, such as the squared difference in Equation 3.6. In the case of squared difference the perfect match would be zero whereas a bad matching is large. Another method to calculate the matching of a template is the correlation matching method in Equation 3.7. When using this method the perfect matching results in a large value and bad or mismatches would be either small or zero [BK08].

$$Result_{SquaredDifference}(x, y) = \sum_{x',y'} (T(x', y') - I(x + x', y + y'))^2 \tag{3.6}$$

$$Result_{CorrelationMatching}(x, y) = \sum_{x',y'} (T(x', y') \cdot I(x + x', y + y'))^2 \tag{3.7}$$

3.4 Objects recognition

When analysing images the detection of objects is a difficult task. Because the problem is, how these objects are defined. Even when defining something simple as a chair it is not an easy task. People see a chair and normally know instantly that it is a chair. For a computer it

must be defined in detail or there must be a training for recognizing chairs. How to descripe a chair? It has four legs, a seat and a back?! Starting with the legs, how are they connected or aligned to each other? How long are the legs? The seat is connected to the legs and the back? There are many things to think of when modelling a chair. For a simple modelling assume the chair has for legs and these are straight downward in the same distance to each other. The seat is on top of the legs and the back is mounted to the seat. This description would be satisfying for a lot of chairs but for three legged chairs this would not be sufficient.
If the task for object detection is limited to detecting one object and the characteristics of the object are known it facilitates the task. In the case of vehicle detection it is assumed that on the motorway are only vehicles. The main task here is to distinguish between the different types of vehicles like motor bikes, cars and motor trucks.

3.4.1 Type estimation

For estimation of the type the vehicles are grouped into three different types, cars, motor trucks and motor bikes. To distinguish each type of vehicle it is mandatory to know the characteristics of each type. Motorcycles distinct from the other two in size, especially in width. Other distinctive features motorcycles have are one headlight, two tires, no licence plate on the front, the windscreen is small or no windscreen at all. The most distinctive and easiest to recognize feature of a motorcycle is the width. For distinction of cars and motor truck it gets more difficult. Both share a big windscreen and have a similar width as they, except for compact cars, can be very broad. The number of headlights is not a good distinctive feature as some cars can have fog lights. This leaves the length as a good distinctive feature. Another possibility is to use the ratio of the width and length to estimate the type. For a bike the ratio is around 2:1. A car has an approximate ratio of 1:1. For a truck the approximate ratio is 3:1. As the distinction between motor bike and truck is very hard by just taking the ratio it is necessary to take the real size into account as the width of a normal motor bike is around 1 - 1.5 meters and a truck is over 3 meter in width.

3.5 Scene recognition

On the scenes recognition level a scene can be created from the extraced information so far. In a scene the object types and their position are known. Also the relation between those objects is now known.

3.5.1 Speed calculation

Calculating the speed of an object can be done by using the time period and travelled distance. To aquire these infomations the frames per second of the video is used to know how much time has passed in X frames. This is calculated by $\Delta t = (1/fps) \cdot X$. For the estimation of the travelled distance cross ratio was used. Advantages of this method are fast computation and being invariant to projective transformations. As already explained in subsection 2.2.8.

With four known points located on the image the cross ratio λ can be calculated. By taking λ and the three corresponding known points in the real world the distance of a fourth unknown point can be calculated.
As a requirement the points have to be on a collinear line. At first three points in the real world are choosen whose distance is known. These points are on the markings on the street, see Figure 3.7. This is done as the distances between spacing and marking are known to the

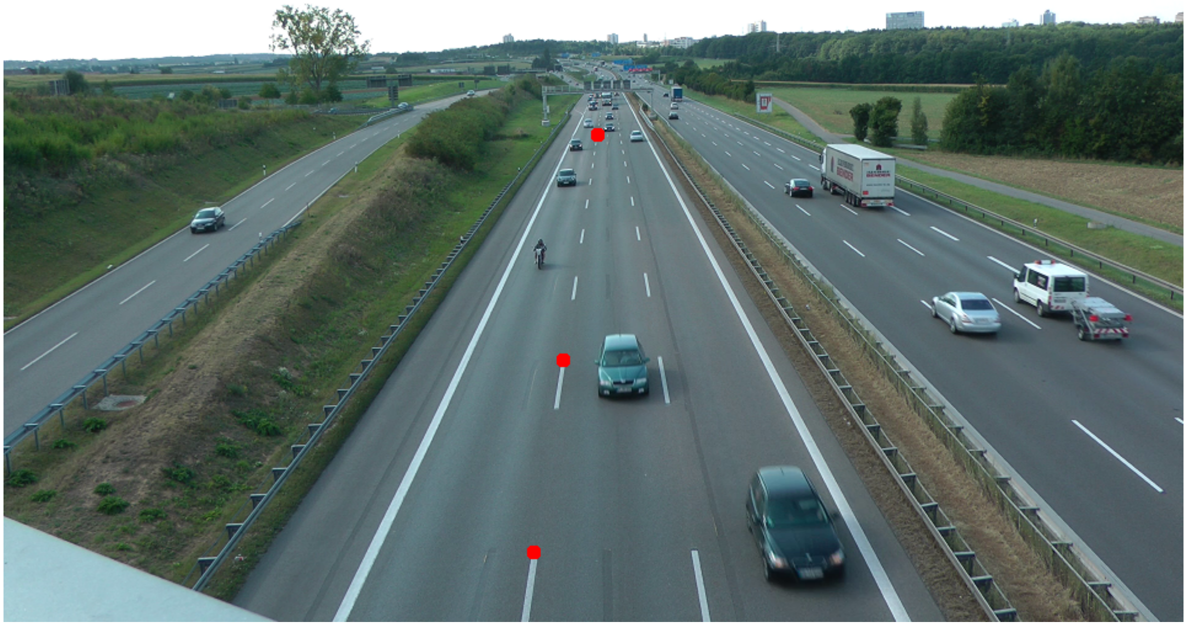


Figure 3.7: Three colinear points marked red on the image. The fourth point would be the position of a vehicle, which is moved in y-direction to be on the collinear line. These four points can then be used for calculation of the distance by using cross ratio.

fact that these have regulated distances. The length of a marking in Germany is six meters and the spacing between two markings is twelve meters on a motorway [RMS80]. Then the three corresponding points in the image were chosen. This leaves two more points, one in the real world and one in the image. The fourth point on the image is the detected position of the vehicle. Now the distances between the points in the image are calculated. By having these four distances the formula could be solved and λ is known. As λ has the same value for the distances in the real world and on the image this leaves now the calculation of the distances of the points in the real world. The absolute distance of the vehicle is calculated by transforming the cross ratio equation and calculating it. There is one problem as the vehicle is seldom exactly on the line. Therefore the x-coordinate of the position of the vehicle was changed to be exactly on the line. This moves the vehicle in x-direction and leads to inaccurateness. For a decent accuracy this is sufficient. The Pythagorean theorem is used for the error estimation, see Figure 3.8. Taken that the angle of a car changing the lane is at maximum 25°. Taking

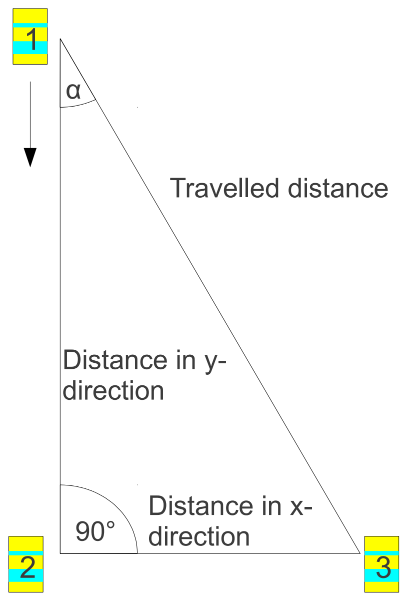


Figure 3.8: Pythagorean theorem usage for error estimation. The yellow vehicle marked with "1" is the previous position, the number "2" shows the position of the car when using only the x-direction and the number "3" is the actual position of the vehicle. The difference between the real travelled distance and the distance travelled in y-direction is between vehicle "2" and "3".

the speed is at $120\frac{km}{h}$, which is $33.\overline{3}\frac{m}{s}$, the accuracy is at over 90%. In Equation 3.8 is an calculation for the accuracy estimation.

$$\cos(25°) = \frac{b}{33.\overline{3}} \Rightarrow b = \cos(25°) \cdot 33.\overline{3} = 30.2m \frac{30.2m}{33.\overline{3}m} = 0.906\% \tag{3.8}$$

An observed normal lane change, the angle α is roughly 10° which leads to an higher accuracy at 98.5%, see Equation 3.9. As cross ratio is invariant to perspective transformations it works in perspective images.

$$\cos(10°) = \frac{b}{33.\overline{3}} \Rightarrow b = \cos(10°) \cdot 33.\overline{3} = 32.8m \frac{32.8m}{33.\overline{3}m} = 0.985\% \tag{3.9}$$

To average the speed the current calculated and the previous speed is added to smoothen the errors. As the speed of an vehicle is not changing rapidly if there is no obstacle or such in the way. For more accuracy one could take the distance travelled in y-direction into account.

3.5.2 Speed calculation in perspective transformed images

The calculation of the speed in a perspective transformed image is compared to the previous introduced method in subsection 3.5.1 a trivial task. But only if having already gathered information about the real world and the camera. This infomation is the known distance between two points and the number of frames per second in which the camera is recording. The points do not have to be at the same position but the distance in a unit (meters is the unit of choice in this thesis) has to be known. The frames per second are necessary to calculate the time difference between the two images. It is not mandatory to use every frame if the application is not relying on it. Taking every n-th frame is possible, but then the time between two frames has to be multiplied by the number of frames passed.
The size of a pixel in the meters is calculated in Equation 3.10.

$$distance_{Pixel} = \sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2} \tag{3.10}$$
$$\text{Pixel in meter} = \frac{distance_{Pixel}}{distance_{Meter}}$$

These two steps have to be done only once if the time between each calculation is equidistant. Also the resizing of the image can now be done as the pixel in meter distance would differ from the actual value. Now the distance of the previous position and the current position needs to be calculated. The calculation of speed is now done as all variables are known. The formula can be seen in Equation 3.11.

$$Speed = distance_{Pixel} \cdot \text{Pixel in meter} \cdot \text{fps} \cdot 3.6 \tag{3.11}$$

In the formula the coefficient 3.6 originates of the conversion from $\frac{m}{s}$ into $\frac{km}{h}$. The result of the equation is the speed in $\frac{km}{h}$.

4 Methods for vehicle detection and tracing

4.1 General preparational work

In this section the work is explained which has to done before the actual processing is started. This covers camera calibration and background extraction. These two steps can be done beforehand.

4.1.1 Camera calibration

As first step the image is corrected with the camera parameters.To correct the image captured by a camera, which is represented as a matrix, it has to be undistorted with the distortion matrices. For a detailed description see 2.1.1. As the vendor of the camera is mostly not shipping the camera with matrices to undistort the images, these matrices were acquired by using camera calibration. Therefore the plug-in cameraCalibration for FloDiEdi was implemented which calculates them with the help of a chessboard that is moved across space. To make it usable for other tasks the calculated matrices are stored in a file. As OpenCV is providing a big part of the functionality of FloDiEdi, the format of choice for saving is the yml format which is used by OpenCV. A number of images, which shows the chessboard in different positions, have to be captured. This means that the chessboard has to be moved forth, back, up and down. When the chessboard is positioned the capture button has to be pressed to take a snapshot. All of the detected corners on the chessboard are displayed to help the user in decide whether the image is good or not. In case not all corners are detected and the user clicked the capture button the number of successfully captured images will not increase.

4.1.2 Background extraction

Background extraction is the process to aquire an image without the objects in the foreground. A background is, in terms of this thesis, an empty street without vehicles. Three possibilities to achieve this background image are explained here. The obvious one with no computation at all is to wait until the street is empty and use this image as the background image. But this has a disadvantage when trying to take the image on heavy loaded streets. It is very hard to acquire such a picture off an street with steady traffic as at least one vehicle will be often present.
The second option is to take several images on the street and use the parts of the image where

no vehicle is. When every part of the image is captured the single parts need to be put together to the background image.
The third option is by using a background extraction algorithm. A principle of extraction is to use the images and use an addition. So a simple background extraction algorithm is by using weighted addition of the previous background image and the current captured image, see Equation 4.1.

$$Image_{background} = \alpha \cdot Image_{current} + (1 - \alpha) \cdot Imagebackground \tag{4.1}$$

To acquire the image of the background a Gaussian Mixture-based Background/Foreground Segmentation Algorithm from Zivkovic [Ziv04] is used. The algorithm is fast and also uses shadow removal techniques to achieve good results. This implementation is one of the algorithms provided by OpenCV in the class BackgroundSubtractorMOG2. A real-time implementation is also possible which achieves good results. This was done by Butler et al. in [BBS05]. Furthermore the scene is static which means that after acquiring the background it will not change. This is true if the changes in illumination are neglected when clouds cover the sky, dawn or twilight. In comparison with the result of the Mixture of Gaussians method, the sum method has problems in terms of accuracy. After a thousand processed images there are still light remainings of visible vehicles. The background is stored, like the camera parameters during the calibration, in the yml format to support file handling with OpenCV. A number of background extraction algorithms are presented, explained and evaluated in the thesis of Brutzer [Bru10].

4.2 Approach I: Background subtraction, feature detection and feature tracking

The idea for the first approach was to get the vehicles, extract features from the vehicles on the image and track those features. A similar approach is used in [GS08]. The proposed method only detects and tracks vehicles without processing further information. The data structure for storing the information is shown in Listing 4.1. All collected information of a vehicle in this approach is saved in this vector.

4.2.1 Preparation and preprocessing of the image

Before starting with the processing of the images the mounting and positioning of the camera has to be considered. Cameras mounted on a pole are deep-seated and will not shake much in case of wind or vehicles passing by. The camera for capturing was mounted on a tripod and was standing on a bridge to get a good overview of the traffic scene. When there was weak wind it was stable but as a motor trucks are passing the bridge started to swing and shake. Therefore the image had to be stabilised at first. To prevent a shaken image the image stabilisation algorithm from Algorithm 2.2 is implemented. As reference a part of the image with no moving elements is choosen. This way no vehicles can falsify the result. When no stabilisation action is done the results of the whole processing can be falsified. It might happen

Listing 4.1 Data structure for storing all information aquired in the first approach for an vehicle.

```
<listing name="Data structure">
     struct vehicleDescription {
     //Stores the ID-number assigned to the vehicle.
     int IDNumber;
     The rectangle which covers the area of the vehicle.
     cv::Rect carHull;
     The current position of the vehicle.
     cv::Point2i center;
     // Current speed of the vehicle.
     int speed;
     // Defines the type of vehicle: 1 for motor bikes 2 for motor truck and 3 for cars.
     int type;
     // The vector containing the found feature points of the vehicle which is used for
         tracking.
     std::vector <cv::Point> vehicleFeatures;
   };
</listing>
```

that by shaking the vehicles appear not to move between two frames. This leads to false speed calculations or problems in tracing the vehicles.
As stated in subsection 2.1.1 the resolution of the camera is 1920×1080. The aspect ratio of the camera is therefore 16:9 and on the left and right side is an area in the image which contains partially irrelevant regions that are not relevant for processing. In these regions can be the road in the opposite direction, trees and so on. This means that these regions of the image can be ignored in terms of vehicle recognition as there is nothing relevant for this task. Concerning the high resolution of the camera and the amount of data that needs to be processed a region of interest (ROI) is used. By using the ROI the computing time is optimised and the focus is kept on the important part of the image.
For finding the ROI in the image a frame of the videostream was grabbed. The frame was loaded with the program GIMP[1] (GNU Image Manipulation Program) to manually find the x- and y-coordinates of the ROI. The stabilisation algorithm returns an offset for the x- and y-coordinate that is added to the previously selected points of the ROI. By this method the ROI is moved horizontal and vertical but the size of the ROI remains the same. The whole operations are done only on the ROI.
As further computing time optimisation the white lines, which limit the width of the street, are choosen as a further reduction of the ROI. Therefore a mask matrix is created to have the ability to check whether a point is in or outside the focused area of the image. As mask matrix a single-channel matrix with 8-bit depth is used to preserve memory and the size of the previously defined ROI. In the mask matrix the non-relevant pixels are set to zero to indicate that the pixel is outside the focus and have the value one if it is inside the focus. By doing so no unnecessary features are found, have to be tracked and filtered out afterwards as they are not in the scope. The size of used memory will be unchanged by this. To find the white

[1]http://www.gimp.org/

lines hough transformation [GW92] is used. At first the image is converted to a binary image. Afterwards the image processed by the OpenCV function cv::HoughLinesP. This method uses an approach which is described in [MGK00]. The output of the function is a vector with the start and end point of the lines. By having the points the length of the vector is calculated and evaluated. Taking the two longest lines was investigated but this was not successful. The result was that the lines were both on the left or right side. Therefore the position of the end and start points were taken into consideration. By knowing that one of the two points, which define the line, has to be in to lower part of the image. For the left line in the lower left part and in the lower right part for the right line. The points however were not exactly at the corner points of the image. For this an additional constraint was added. The point has to be near the corner point. Therefore a rectangle with width and height of 100 pixels is set as possible range where these points are. Now the lanes get properly detected. See Figure 4.1 for a visual representation.

This was however not sufficient to create the mask matrix as the line was not from the top to

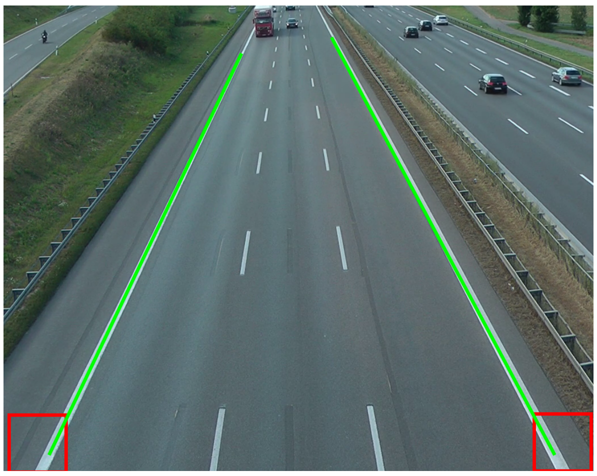


Figure 4.1: At the bottom are the red rectangles marking the search area for the white lines. The two successful detected lines are marked green.

the bottom of the image. As the start and end point of the line is known, the linear equation $y = m \cdot x + b$ is used to calculate the points at the top and bottom. The slope m is calculated with Equation 4.2.

$$m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} \tag{4.2}$$

As OpenCV uses a coordinate system where the point (0, 0) is at the top left the offset of height of the image had to be added to the offset b. The formula used is described in Equation 4.3.

$$b = (height_{image} - y_1 - x_1 \cdot m) \tag{4.3}$$

Finally the mask matrix is initialised with “1”, iterated and for all points which were not inside the the traffic lane borders is set to zero. The implementation can be seen in Algorithm 4.1.

The resulting ROI is shown in Figure 4.2. Another possible approach is introduced in [GO00] using a histogram-based segmentation and decision trees. As last step the previously aquired

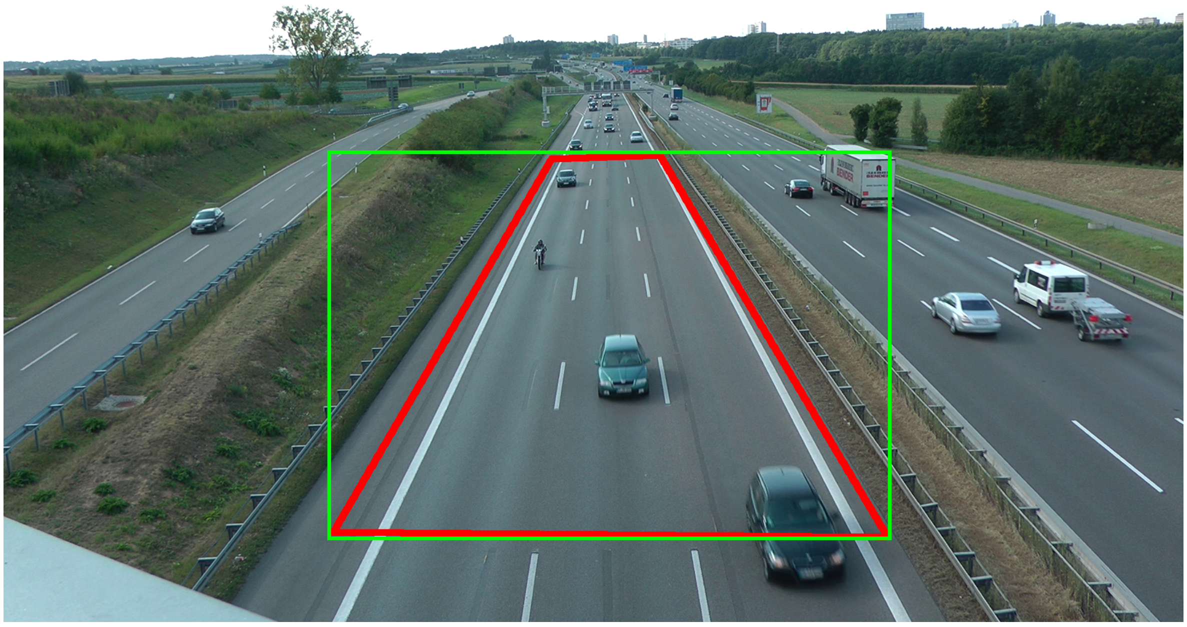


Figure 4.2: In green is the rectangular ROI marked and in red is fitted ROI. The ROI changed from a recangular shape to trapezoid shape, covering now only the street.

background is loaded from a file. The background image is also converted to a grey scale image for later usage in the foreground extraction.

4.2.2 Foreground extraction

The aquiration of the foreground is done by subtracting the current image from the previously obtained background, see Equation 4.4.

$$M_{Foreground} = M_{CurrentImage} - M_{Background} \tag{4.4}$$

Therefore the input image is converted to a grey scale image. The result of the subtraction do not provide accuracy on a high level. Therefore the image is taken under further processing. As next step the image is binarised by thresholding which is applied to the image. Depending on the level of the threshold value the resulting shapes of the cars do not contain a lot of shadow. But for vehicles with a similar color like the street the detection does not achieve good detection results. The foreground image is processed by thresholding to convert it to a binary image. After this the morphological operation closing is applied. This way smaller holes in the mask matrix can be filled which compensates small errors which are caused by the subtraction. The difference in the image before and after closing can be seen in Figure 4.3.

Algorithm 4.1 Detection of the street limiting lines

```
procedure LIMITROI($roiImage$, $maskMatrix$)
    $Lines_{Hough} \leftarrow$ houghTransform($roiImage$)
    for $line_i \in$ vector $Lines_{Hough}$ do
        if ($line_i \in$ left corner) & ($|line_i| > |line_{left}|$ then
            $line_{left} \leftarrow line_i$
        else if ($line_i \in$ right corner) & ($|line_i| > |line_{right}|$) then
            $line_{right} \leftarrow line_i$
        end if
    end for
    $maskMatrix \leftarrow 1$
    for all $pixel_{x,y} \in maskMatrix$ do
        if $pixel_{x,y}$ not between traffic lane borders then
            $pixel_{x,y} \leftarrow 0$
        end if
    end for
end procedure
```

4.2.3 Vehicle extraction

The mask matrix from the previous step contains the vehicle shapes. Extraction of the single vehicles from this mask matrix is conducted. For extracting the vehicles from the mask matrix a connected-component labeling algorithm was implemented. The connected-component labeling pseudo algorithm can be seen in Algorithm 3.3. During the labeling of the image each found vehicle is added to the vector which contains all found cars in the current image. The vector uses the struct defined in Listing 4.1. Therefore a center point is calculated, the rectangular hull which contains the shape of the vehicle, the type of detected vehicle and an temporary ID. The detection of the type is done by using the width and length of the rectangular hull. The size, width and length, for each type is preset. The pseudo implementation is in Algorithm 4.2. For the filling of the shapes the OpenCV function cv::Floodfill is used.

4.2.4 Feature detection and assignment

So far all vehicles are extracted and stored in the vector. In this step the features in the image are detected and assigned. For this detection the Shi-Tomasi feature detector is used. This detector was chosen as vehicles have a rectangular shape and can contain corners. The ROI matrix is processed with the Shi-Tomasi feature detector which is explained in subsection 3.2.1. Before the detection the variable for the quality of features is set. When processing the image and finding a feature it is first checked wheter the point is between the traffic lane borders. This is done by checking the mask matrix if the point is on a “1” in the mask matrix. The next step is to assign the feature to the corresponding vehicle. Therefore the vector with the vehicles is iterated and checked wheter the point is inside the hull. If it is, then the feature point is added to the vector **vehicleFeatures** of the vehicle otherwise the next vehicle is

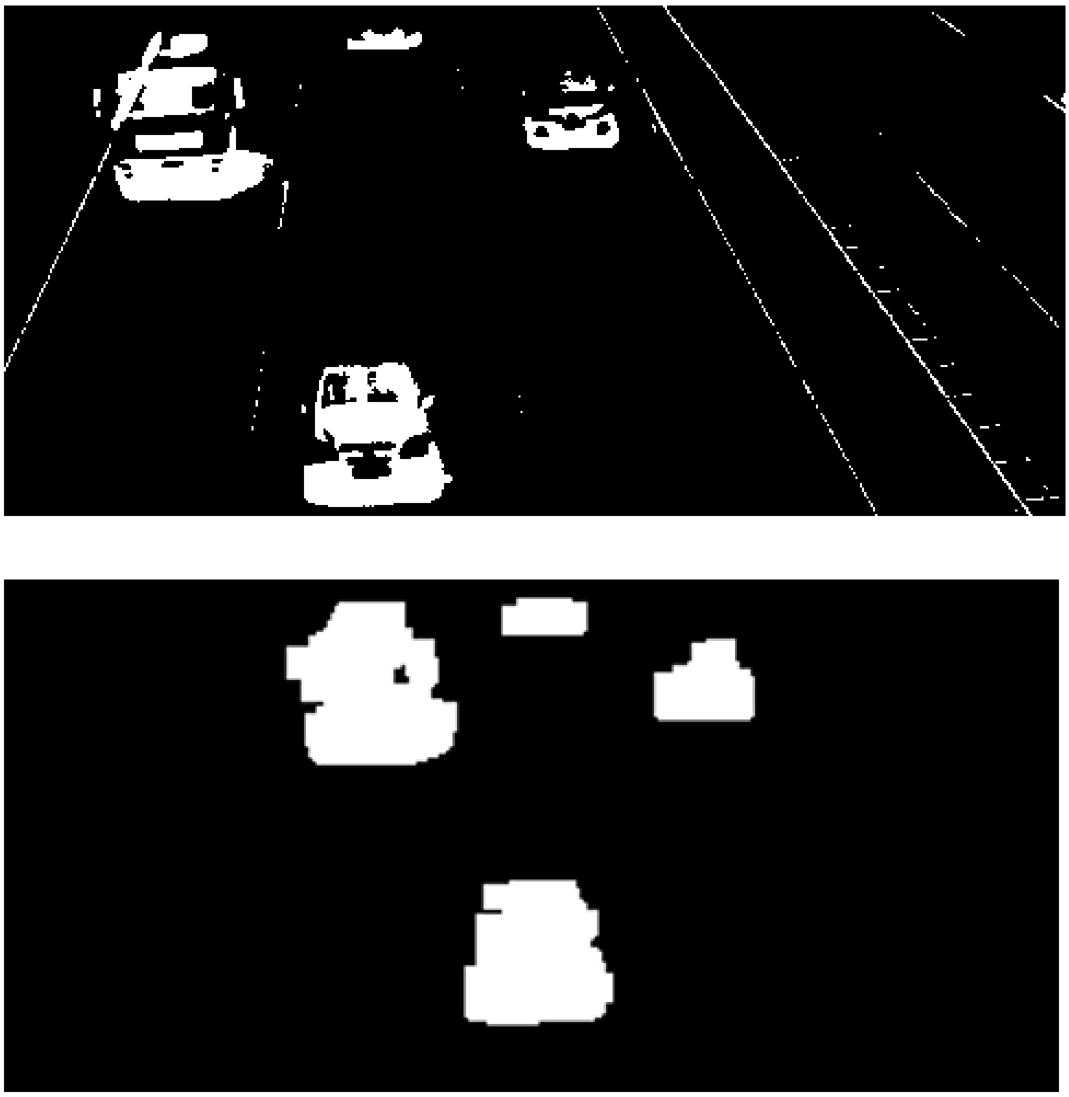


Figure 4.3: The upper image shows the extracted foreground. On the bottom is erosion applied to the image to fill small holes and remove single pixels which do not belong to the foreground.

checked. If it is not inside any vehicle hull the feature point is discarded. The vehicle have features that are traceable. In Figure 4.4 the found features are marked by red points.

Algorithm 4.2 Extraction of vehicles from the mask matrix

```
procedure EXTRACTVEHICLESHAPE($Input_{Image}$, $maskMatrix$)
    $counter \leftarrow 2$
    $IDCounter \leftarrow 0$
    for $Pixel(x, y) \in maskMatrix$ do
        if $Pixel(x, y) \neq 0$ then
            Set value of all pixels connected with the same value to counter
            Get rectangle
            Calculate center position
            Set ID
            Push vehicle information to vector
            IDCounter++
        end if
    end for
end procedure
```

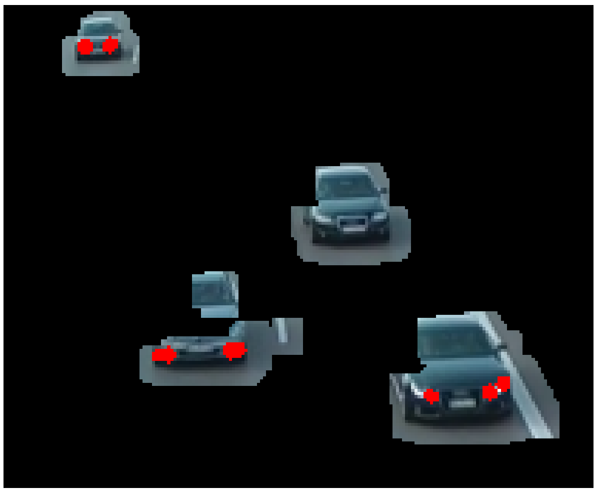


Figure 4.4: Features detected in the foreground image with the Shi-Thomasi detector. The image is overlayn by the mask matrix to see the detected shapes of the vehicles.

4.2.5 Feature tracking and speed calculation

As every vehicle has its features these features can be used for tracking the vehicle. For tracking the previous image and the previous vehicle vectors have to be saved for each iteration. The tracking algorithm used is the pyramidical implementation of the Lucas-Kanade tracker, explained in subsection 3.2.2 on page 35. The tracking is done by iterating the vector with the previous vehicles. For each feature of the vehicle the tracker function is called with the feature point, the input image and the previous image as input. The return value of the function is the new estimated position of the feature point. This point is searched in the vector of the

vehicles, extracted from the current image. If the feature point is found inside the hull of a vehicle then the vehicle is considered successfully tracked.
When a feature is successfully tracked the current and the previous position is known. This enables the calculation of the speed. Therefore the two positions are passed to the speed calculation function.
The function uses cross ratio to compute the distances from the image coordinates to real world distances in meter. Therefore four colinear points have to be predefined in the image and their distance in the real world have to be known. A possibility is the use of the distance the marker posts have as there is a regulation about the distance between two marker posts. But it might be that these are due to an previous accident are damaged or removed. To be on the safe side the markings on the road were used. As stated in subsection 3.5.1 the distance between two lines is regulated by law in [RMS80]. The distance in pixel is calculated by using the pythagorean theorem. But with the x-coordinate set to be on the collect line of the three predefined points. Then the cross ratio value λ was calculated. Now the real world distances are used and the cross ratio formula is transformed using Equation 4.5.

$$distance_d = \lambda \cdot (distance_a \cdot distance_b)/distance_c \tag{4.5}$$

The distance was calculated for both, the previous and current position. Finally the positions were subtracted and converted to km/h with consideration of the frame rate as passed time, see Equation 4.6.

$$Speed = 3.6 \cdot (distance_{previous} - distance_{current})/\Delta time \tag{4.6}$$

The value of Δt is 1/(frames per second) and the coefficient 3.6 is the used by converting from m/s to km/h. This is shown in Equation 4.7.

$$x\frac{m}{s} = x \cdot \frac{(60 \cdot 60)sec}{1000m} = x \cdot 3.6\frac{km}{h} \tag{4.7}$$

If the vehicle is found for the first time the calculated speed is then stored in the current vehicle. Otherwise the previous speed and the current calculated speed is added and divided by two to smoothen the result in case of inacurate speed calculation. As the speed calculation needs two frames this works after the completion of the initialisation. The result of tracking can be seen in Figure 4.5.

4.2.6 Processing of the collected data

In this last step the vehicles are drawn and the data is saved to a xml file. For the drawing of the detected vehicles the list of current vehicles is iterated and for each vehicle the ID and the current speed is displayed. The hull, which was detected in subsection 4.2.3, is drawn around the vehicle. To show the type of the detected vehicle the color blue, red and green is used as border for cars, motor bikes and motor trucks.
For storing the vehicles the xml format of LISA [LIS] is used. It is possible to store all kinds of information in the proposed form. A sample of the extracted information stored in the xml file can be seen in Listing 4.2. The written data is stored in the home directory of the logged in user in the folder “XMLData”. In Figure 4.6 the completly processed image is shown.

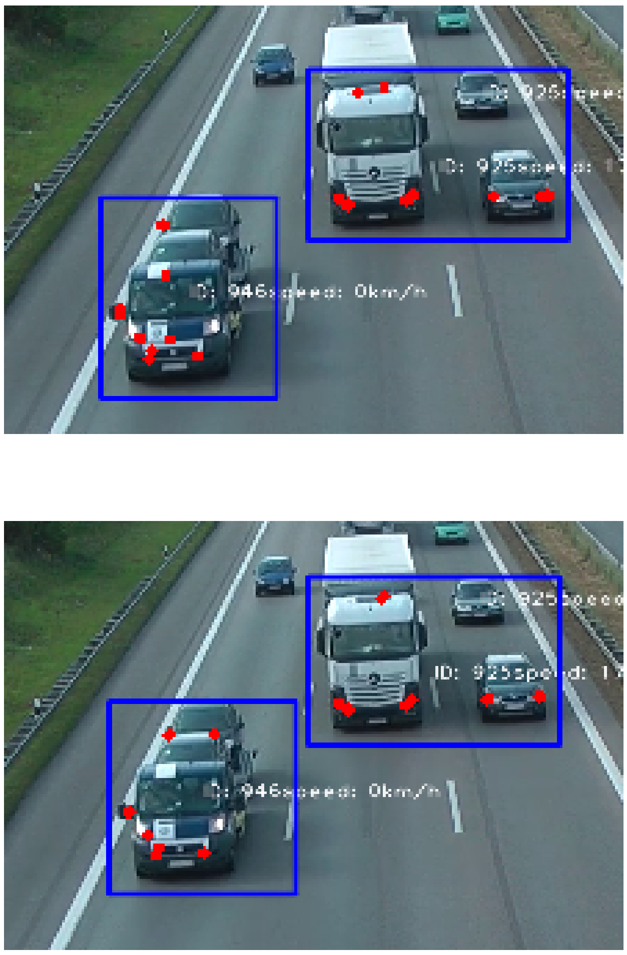


Figure 4.5: The blue vehicle gets detected correct and grouping of three vehicles is done due to bad detection. The grouping of the vehicles is caused by being close to each other and due to shadow cast. Both images show the same region but have at a different time.

Listing 4.2 Output of approach I in xml format for usage with LISA

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<VehicleList xmlns="http://www.asset.eu/lisa">
  <Vehicle ID="197" TimeStamp="19" VehicleType="Bike" Width="318" Length="74" PosLong="723"
      PosLat="1015" LaneID="0" Speed="67" /></Vehicle>
  <Vehicle ID="52" TimeStamp="19" VehicleType="Passenger Car" Width="266" Length="472"
      PosLong="1357" PosLat="528" LaneID="2" Speed="62" /></Vehicle>
</VehicleList>
```

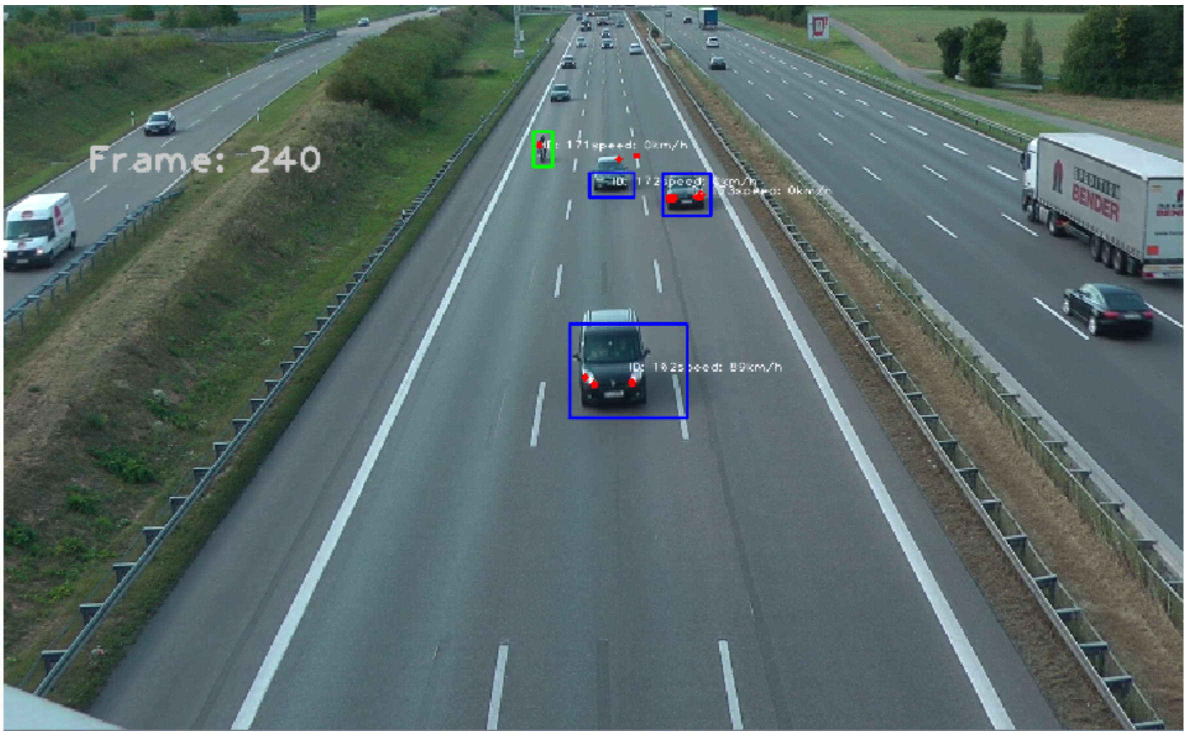


Figure 4.6: The image shows the processed and marked image using the first approach. The vehicles are marked with the rectangles and the gathered information is printed. On the upper left is additionaly the frame number shown.

4.3 Approach II: Perspective transformation and feature tracking

In this approach of vehicle detection and tracking the image is perspective transformed. The tracking of vehicles is done by feature tracking. The ROI is selected manually with the mouse. This makes this approach more adaptable to different views of the street. The data structure is in Listing 4.3. Before starting with this approach the user has to set the number of lanes which. As standard this is set to three. Also the distance between the road markings has to be set as this is later used for the speed calculation.

Listing 4.3 Data structure for storing all information aquired in the second approach for an vehicle.

```
<listing name="Data structure">
  struct templateVehicle {
    //Stores the ID-number assigned to the vehicle.
    int IDNumber;
    // The rectangle which covers the area of the vehicle.
    cv::Rect carHull;
    // The current position of the vehicle.
    cv::Point2i center;
    // Current speed of the vehicle
    int speed;
    // Defines the type of vehicle: 1 for motor bikes 2 for motor truck and 3 for cars.
    int type;
    // The vector containing the found feature points of the vehicle which is used for
        tracking.
    std::vector <cv::Point> vehicleFeatures;
    // The lane the vehicle is driving on
    int lane;
  };
</listing>
```

4.3.1 Preparational work

After starting the program there are a various things to set up. At first the perspective matrix P is calculated. For the details about the calculation refer to subsection 2.2.9. The user input is used to get the four points of the perspective transformation and computes automatically the other four points to have a rectangular image. For this the points, which are chosen by the user, are enclosed by a minimum bounding rectangle. Before continuing with the next step the background, which was already extracted, is loaded from a file. The background is warped with the calculated perspective matrix and converted to a grey scale image to use it for the foreground extraction. For later use the resulting image is refered to as perspectiveBackground. in the rest of this approach.

4.3.2 Foreground extraction

The foreground extraction in the second approach is similar to the method in the first approach. The difference is that the image is perspective corrected. At first the current input image is warped using the perspective matrix P. Then the warped image is converted to a grey scale image, called greyInput. Then the greyInput is subtracted by the perspectiveBackground, see Equation 4.8.

$$Foreground = greyInput - perspectiveBackground \tag{4.8}$$

The resulting image has no clear shapes of the vehicles and even results in vehicles being split in two or more parts of connected pixels. Therefore the image has to be further processed. Thresholding is applied to the image which binarises it. It can also reduce the amount of shadows if the threshold value is set to a reasonable value. Then the morphological operation closing is done to reduce this problem. As this operation fills small holes, see subsection 2.2.5, it can also connect the split up vehicles together again. This leads to better results concerning the vehicle shape. The application of morphological closing to the image is shown in Figure 4.7. This binary image is uses as mask to find pixels in the foreground and for the vehicle extraction.

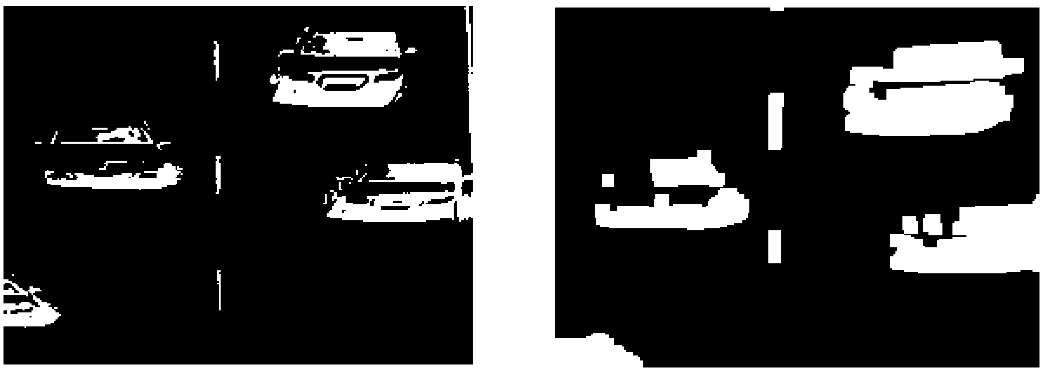


Figure 4.7: The image on the left is before the morphological operation closing is performed. On the right image is the image after closing. Some single pixels are removed and the vehicles have a more rectangular shape.

4.3.3 Vehicle extraction

For extracting the vehicles a connected-component labeling algorithm is used as in the first approach. The enhanced binary image contains the shapes of the vehicles and can be processed. The method for extraction is to iterate over each pixel in the image and label each connected pixels. By doing so the vehicles which are in the foreground will be extracted. The extracted vehicles obtain an **ID**, the **center** position of the vehicle, the **lane** the vehicle is on, the **type** and the **size** of the vehicle is estimated. The **ID** is an increasing counter, starting with zero. After each assignment the counter is increased by one. The **lane** detection is done by checking if the **center** is to the left of the first lane, if so then the vehicle is on the left. If it is not, the check is be done for the second, third, . . . lane. The **carHull** of a vehicle is a return value of the used function cv::Floodfill. The size is derived from the size of the rectangle and the **center** is the center of the rectangle. The type is estimated by the ratio of the length and width of the rectangle. Through observation the following ratios were found out: A motor bike has roughly the length to width ratio of 2:1, a car has around 1:1 and a motor truck has around 6:4. Every vehicle is stored in a vector with the type **templateVehicle**. Although the closing operation was performed in the previous step, there is still a small amount of vehicles which have not been properly connected. Therefore the height of the vehicle is increased by a few pixels. Now there is a check to find the rectangles which intersect with others. When two rectangles intersect the length of these two is combined and the width is set to the higher value of the two. A last step checks wheter the size of a vehicle is above 1.5 times the size of a lane. The value of 1.5 is choosen as due to shadows the vehicle appear larger. This is done as in dense traffic vehicles, which are close to each other, can be distinguished.

4.3.4 Feature detection

To detect the features the Shi-Tomasi detector is used. This detector extracts edges as features. See subsection 3.2.1 for a detailed explanation. As in the first approach the features are only detected in the specified ROI, which is selected by the user at the beginning. Every time a feature is detected a check wheter the feature is in the rectangle of a vehicle is conducted. Therefore the vector which contains the vehicles is iterated as long as no vehicle rectangle is found that contains the point of the feature. If the vehicle is found the feature point is added to this vehicle in the **vehicleFeatures** vector.

4.3.5 Feature tracking and speed calculation

Up to now the vehicles have been extracted, the features in the image are detected and assigned to the corresponding vehicle. The tracking of the features is now conducted. This procedure is simliar as in the first approach. For the next step in the application flow the previously found features are tracked and the speed calculation is done. To track the features the Lucas-Kanade tracker from section 3.2.2 is used. When using the tracker the current image, the previous image, the current list of vehicles and the previous list of vehicles are mandatory. As in the initialisation of the program there are no previous images or previous vehicle lists, the tracking can start after the second iteration. To track the features the vector containing the previous vehicles is iterated. Then the current and previous image is processed by the feature tracker, which returns the position of the feature in the current image.
As the position of the features are points, it is checked wheter the point is in any car and the tracking is successful. Therefore are the current vehicles iterated and for each vehicle is

Listing 4.4 Output of approach II in xml format for usage with LISA

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<VehicleList xmlns="http://www.asset.eu/lisa"></Vehicle>
  <Vehicle ID="0" TimeStamp="0" VehicleType="Bike" Width="40.6342" Length="33.3008"
      PosLong="755" PosLat="553" LaneID="0" Speed="0" /></Vehicle>
  <Vehicle ID="1" TimeStamp="0" VehicleType="Passenger Car" Width="13.2241" Length="1.56285"
      PosLong="1086" PosLat="443" LaneID="1" Speed="0" /></Vehicle>
  </VehicleList>
```

checked if the point is enclosed in the rectangle **carHull** of the vehicle. If it is enclosed the vehicle is tracked, otherwise the next vehicle is checked.
When having tracked a vehicle between the two consecutive images the speed of this vehicle is calculated. At first the distance between the two center points, the variable **center**, of the vehicles is calculated with pythagorean theorem. At the beginning in the subsection 4.3.1 the user has to set the ROI and mark the distances between two road markings. The distance between these is fixed and can therefore be used to calculate the size of a pixel in the real world. With the size of a pixel the actual distance in meter is calculated. As the frame rate is used as constraint for the passed time between the two images. The used Δt is 1/fps. To transform from m/s to km/h the value has to be multiplied with the value 3.6. Before the actual speed calculation the distance in pixels is calculated by Equation 4.9.

$$Distance_{Pixel} = \sqrt{(x_{current} - x_{previous})^2 \cdot (y_{current} - y_{previous})^2} \tag{4.9}$$

The Equation 4.10 is used to calculate the speed of the vehicle.

$$Speed_{km/h} = (Distance_{Pixel} \cdot pixelSize \cdot 3.6)/fps \tag{4.10}$$

In Figure 4.8 is the successful tracking of three vehicles over ten consecutive frames shown.

4.3.6 Processing of the collected data

In this last part of the program the collected data about the vehicles is mapped on the image and stored to a file. For the mapping the vector of vehicles is iterated and around each vehicle is a rectangle drawn with the size of **carHull** around the center of the vehicle position stored in **center**. The color of the border line depends on the type of vehicle. It is blue if it is a car, green for a motor bike and a motor truck is framed with a red color. The displayed information as text on the image is the **ID**, the **type** , the **lane** and the **speed** of the vehicle. The saved file uses also the xml format from LISA [LIS]. In comparison to the first approach the saved information is enhanced by the **lane** the vehicle drives on. The files are stored in the home directory of the user in the folder XMLData. If it does not exist it will be created. An example can be seen in 4.4. In Figure 4.9 is the image shown which is the output of the plug-in.

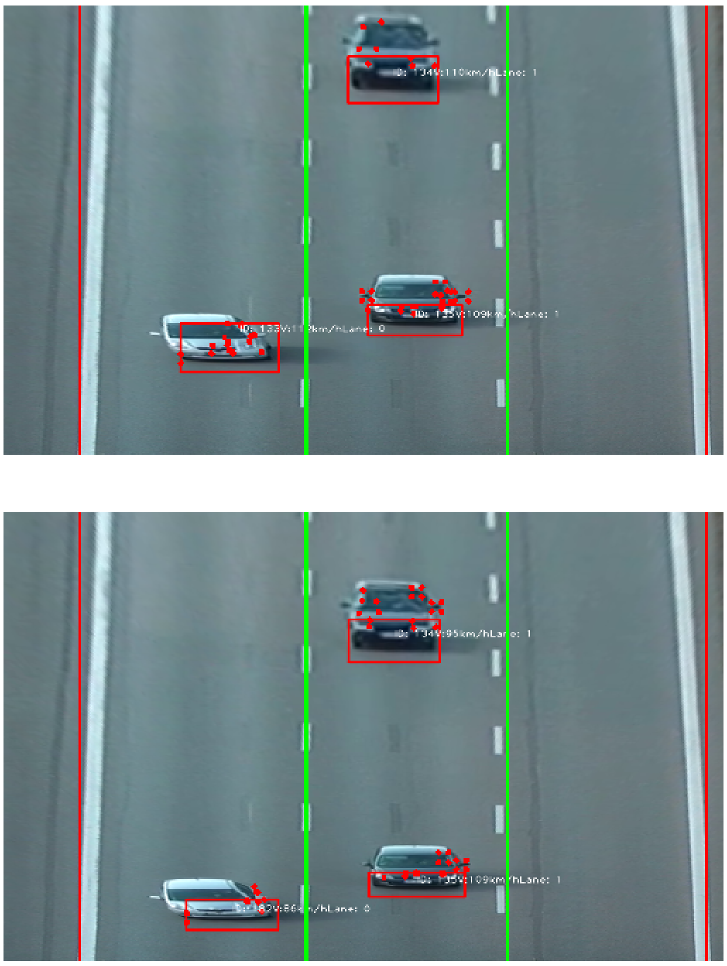


Figure 4.8: Three detected vehicles in the upper image. The lower image shows the same detected vehicles after 10 frames successfully tracked. In both images is the same part of the image selected.

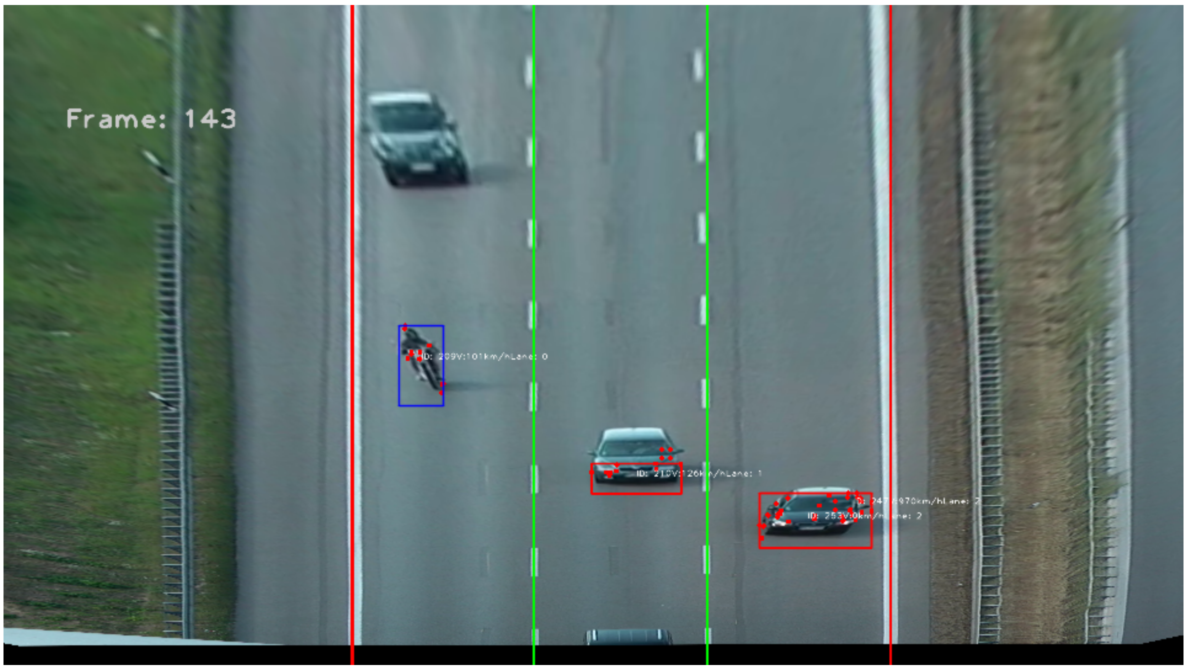


Figure 4.9: The resulting image when processing the video. All vehicles are marked and their information is printed. As additional information is the frame rate shown.

4.4 Approach III: Perspective transformation, position estimation and template matching

This approach allows the user to select the ROI and the road markings in the image. The input images are perspective transformed to get a bird's eye view of the street. Different to the other two approaches is that no feature is detected and tracked but the new position of the vehicle is estimated. Additionally template matching is used to confirm the correctness of the estimation. The focus in this approach is also the achievement of real-time processing.

4.4.1 Preparational work

At first the user has to define the region of interest and the road markings manually. To do this the Qt framework is used the get the coordinates which are clicked on the image. The road markings have to be clicked to have a reference of the pixel size in meter. Therefore the gap between two road marking is selected as this distance is regulated, see [RMS80].
After all points have been set the perspective matrix is calculated. To get the perspective matrix a total of eight points is needed. Four of these points, which define the perspective projection are already selected by the user. The other four points are set automatically to the enclosing rectangle of the selected region of interest.
As last step in the preparation the background image is transformed by the perspective matrix and converted to a grey scale image. This image is stored in the variable **greyBackground**. The data structure for this approach is in Listing 4.5.

Listing 4.5 Data structure for storing all information aquired in the third approach for an vehicle.

```
<listing name="Data structure">
  struct templateVehicle {
    //Stores the ID-number assigned to the vehicle.
    int IDNumber;
    // The rectangle which covers the area of the vehicle.
    cv::Rect carHull;
    // The current position of the vehicle.
    cv::Point2i center;
    // The current speed of the vehicle.
    int speed;
    // Defines the type of vehicle: 1 for motor bikes 2 for motor truck and 3 for cars.
    int type;
    // The lane on which the vehicle is on.
    int lane;
    // This matrix contains the vehicle extracted from the image.
    cv::Mat vehicleImage;
  };
</listing>
```

4.4.2 Foreground extraction

In this approach the foreground is extracted by using the same method as in the second approach. At first the input image is converted to grey scale and stored in the variable **greyInput**. The foreground is extracted using the method of subtracting the **greyInput** from the grey background image **greyBackground** in Equation 4.11.

$$Foreground = greyInput - perspectiveBackground \tag{4.11}$$

This leads not to perfect results. Therefore the image needs to be further processed. The image is binarised by thresholding which, if a good value is found, can also reduce the amount of shadows. The binary image now has either the values “0” or “1” even though it is possible to store values in the range of $0 \ldots 255$. This is done to use the connected-component labeling algorithm in the next step of vehicle extraction. Finally the image is processed by the morphological operation closing. The advantage of this is that the binary image may contain holes which can be filled and a connected shape of a vehicle is the result. This binary image is used for the vehicle extraction step.

4.4.3 Vehicle extraction

Extracting the vehicles is done by a connected-component labeling algorithm. This is also done in the previous two approaches. This algorithm iterates over the image and if a point is found whose value is 1, the value of this and all pixels interconnected with the same value ate set to the value of the counter. The counter is initialised with the value of two, as zero and one is already taken. This way each vehicle gets a unique label. For setting the values the OpenCV function cv::Floodfill was used. The return values of this function is the center of the shape and a bounding box. The center point is used as **center** for the vehicle and the bounding box is stored as the **carHull**. The type of vehicle is estimated by using the real world sizes of the detected vehicle. Therefore the width and length are multiplied by the pixel size in meters. At first a check wheter the width of the vehicle is larger than two meters is performed, if it smaller than two meters the vehicle is a motor bike. Otherwise it is tested if it is a car or a motor truck. This is done by using the length in meters. If the vehicle is longer than six meters it is a truck and otherwise a car. Six meters are chosen as cars with trailers should be still detected as cars. Also vans or sport utility vehicles are larger than normal cars. Even though the small holes were filled in the previous step there might still be vehicles that are not connected. Therefore the length of the vehicle is increased by a small amount and it is checked wheter there are intersecting bounding boxes. If they intersect they are combined in length and the width is set to the higher value.

4.4.4 Position estimation

In the first two approaches the vehicles were tracked by using a feature tracker. For this approach the new position of the vehicle is being estimated. To estimate the position the current speed of the vehicle is taken to calculate the distance driven in the the time Δt. For the calculation the predefined input for the distance is taken into account as it defines the size of a pixel. Each vehicle in the vector is iterated. For the new position of the vehicle the **speed** variable from each vehicle is used to calculate the travelled distance in pixels. Distance is calculated by the Equation 4.12.

$$Distance = \frac{Vehiclespeed}{distance\ of\ marking \cdot fps \cdot 3.6} \tag{4.12}$$

The travelled distance is added to the x-coordinate to estimate the position.
If there is no previous calculated speed, the assumption is made that the vehicle drives at the speed of the average speed of the lane which is continuously updated.

4.4.5 Template matching

To ensure that the vehicle is tracked correct, template matching is used. The **vehicleImage** is used as template for the matching. From the position estimation the new position of the vehicle is known. Therefore the vector with the current vehicles is iterated and each vehicle is checked if the estimated position is in the **carHull** rectangle. If this is true the **vehicleImage** of the currently iterated vehicle and the vehicle from the previous vector whose match is searched, is resized to the same size. The matching with the squared difference in Equation 4.13

$$Result_{SquaredDifference}(x, y) = \sum_{x',y'} \left(T(x', y') - I(x + x', y + y')\right)^2 \tag{4.13}$$

is calculated and if the matching is good, meaning the resulting value is near zero, the vehicle is successfully found.

4.4.6 Speed calculation

In this approach the speed calculation is done separately instead of during the feature tracking as in the first and second approach. As the vehicles are matched with the template matching, the two center points of the vehicles are used to calculate the distance in pixels. To get the distance the pythagorean theorem is used in Equation 4.14.

$$Distance_{Pixel} = \sqrt{(x_{current} - x_{previous})^2 \cdot (y_{current} - y_{previous})^2} \tag{4.14}$$

The user provides the defined length in meter for the selected length in pixels. Therefore the pixel size in meters is calculated. As next step the distance in meters is calculated by $Distance_{Pixel} \cdot pixelSize$. With the real distance and the Δt from the frame rate the speed in m/s and then in km/h can be calculated. This is done using the Equation 4.15

$$Speed_{km/h} = Distance_{Pixel} \cdot pixelSize \cdot 3.6/fps \tag{4.15}$$

The calculated speed is stored in **speed** of the current vehicle. To smoothen the possible error the speed of the current and the previous vehicle is added and divided by two. In Figure 4.10 is the successful tracking shown.

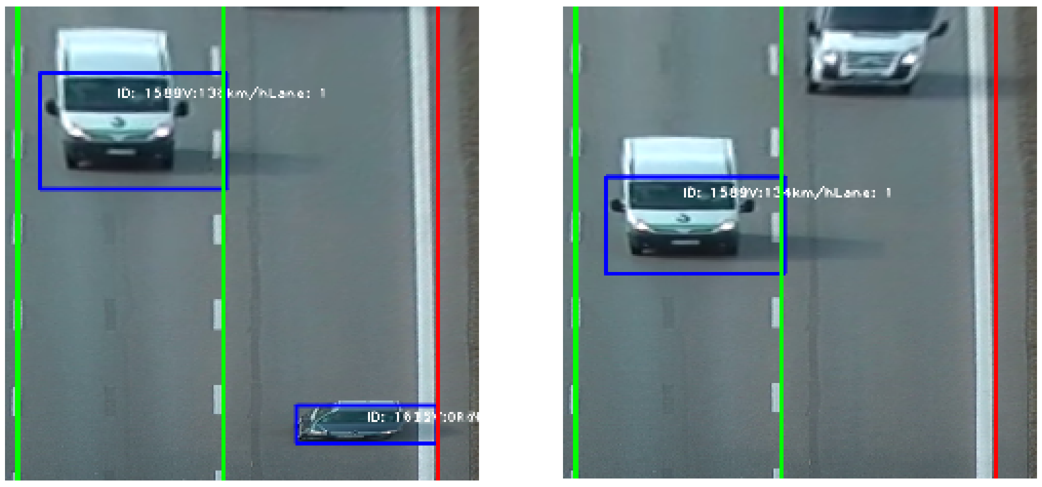


Figure 4.10: The tracking of two vehicles in the left image. On the right did the blue vehicle leave the camera view. Both images represent the same area in the image.

4.4.7 Processing of the collected data

In this final part of the approach, the collected data in the approach is now dispayed on the image and stored to a file. The information displayed cover the **ID**, **speed** and the **lane** as text and the vehicle type as a colored rectangle. The color of the rectangle is blue for cars, green for motor bikes and red for motor trucks.
All of the information as the **ID**, **size**, the **lane** the vehicle is currently on, the **speed** and the **type** of the vehicle is stored in an xml file. But there is a little change in the data of the size, the size is multiplied by the pixel size to get the real size of the vehicle. The files are stored in the home directory in the subfolder XMLData. If it does not exist it will be created automatically. The xml file has the format which is compatible with LISA [LIS]. This way the plug-in could be used to deliver the data directly to the server where the data is further processed. The data format used is in Listing 4.6. In Figure 4.11 is the image shown with the detected vehicles.

Listing 4.6 Output of approach III in xml format for usage with LISA

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<VehicleList xmlns="http://www.asset.eu/lisa">
<VehicleList>
  </Vehicle><Vehicle ID="2" TimeStamp="8" VehicleType="Truck" Width="18.3556"
      Length="7.10187" PosLong="1035" PosLat="645" LaneID="1" Speed="116" />
  </Vehicle><Vehicle ID="3" TimeStamp="8" VehicleType="Passenger Car" Width="32.6686"
      Length="24.6926" PosLong="637" PosLat="662" LaneID="0" Speed="125" />
  </Vehicle><Vehicle ID="43" TimeStamp="8" VehicleType="Passenger Car" Width="32.6686"
      Length="24.6926" PosLong="637" PosLat="662" LaneID="0" Speed="119" /></Vehicle>
</VehicleList>
```

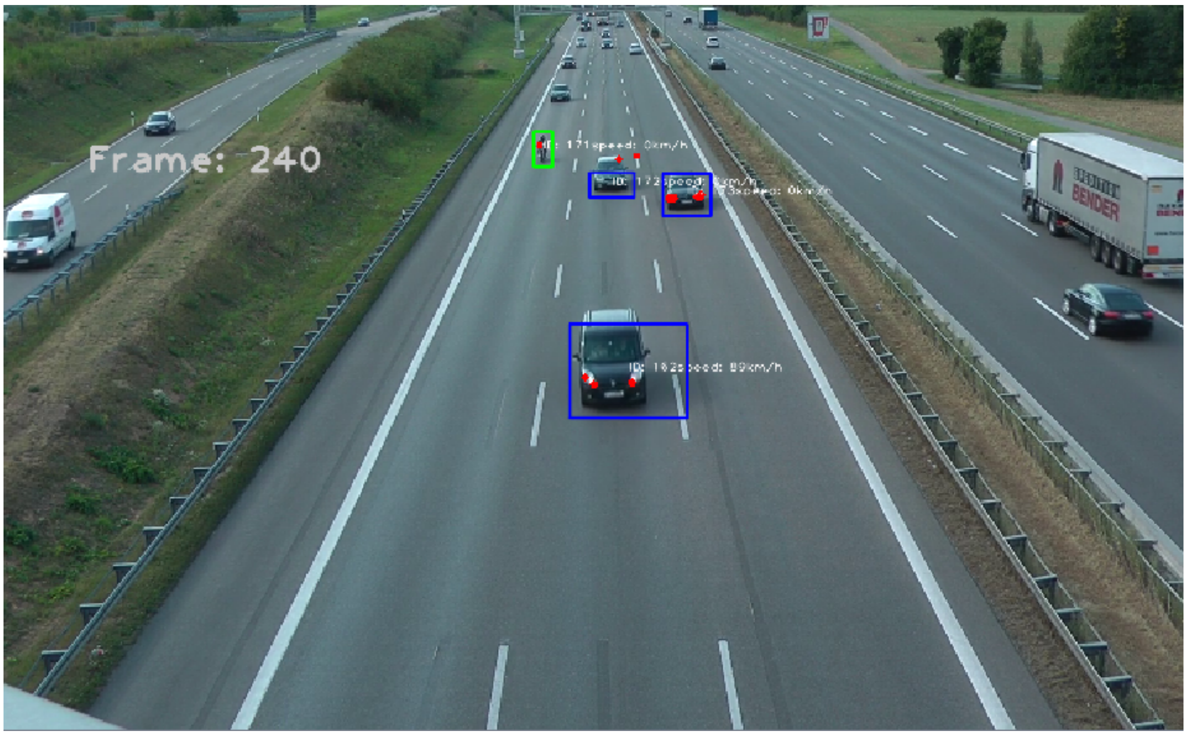


Figure 4.11: The image shows the processed and marked image using approach three. Every detected vehicle is marked and the previously extracted information is printed.

5 Methods for vehicle detection and tracing

5.1 Evaluation

The evaluation takes a closer look at the characteristics of the approaches. Covered here are the computational time as the goal was to achive real-time computation. The detection rate and the tracking of the vehicles are a measurement of reliability of an approach. The tracking of the vehicles and the type estimation are also evaluated here. For the evaluation a sequence of 500 video frames used. A total of 30 different vehicles appear in the sequence. Of these 30 vehicles is one of them a motor bike, three motor trucks and the remaining 26 vehicles are cars. For the tracking a small region of the image was used which can be seen in Figure 5.1. For the evaluation an IBM ThinkPad X60 with an Intel Core 2 CPU T5500 which is clocked at 1.66GHz and 3GB RAM. The operating system is Ubuntu 12.04[1]. The video was recorded at 50 frames per second but it was converted so it is running at 25 frames per second. The format of the video is H.264[2] which already uses a fair amount of resources for playback. For the conversion the program Handbrake[3] was used. The average computation time for processing a video frame in FloDiEdi is 75 milliseconds. The accuracy of the speed calculation is not evaluated as there is no reference vehicle whose speed is known.

5.1.1 Computational time

As real-time is an important part of this thesis this is evaluated first. For the rating are ten consecutive frames processed by each of the approaches. The time is clocked using the two plug-ins cycleTime and doubleDisplay, shown in Figure 5.2. The plugin cycleTime has four different output connectors, each in the double format, one for the cycles per second, the average number of cylces per second, the time for one step and the average time for one step in milliseconds. One step is equal to one processed frame of video. To display the results the doubleDisplay plug-in is used which has one input in double format and displays it. The first approach is quite slow and needs for one step 3,402 milliseconds, so the resulting speed is not near real-time. On average are 3,011 milliseconds passed in the second approach for one step, which is only slightly faster. The third approach processes one frame on average in 470 milliseconds. The fastes approach is the third. This is mainly because the feature detection and

[1]http://www.ubuntu.com/
[2]http://www.mpegla.com/main/programs/AVC/Pages/Intro.aspx
[3]http://handbrake.fr/

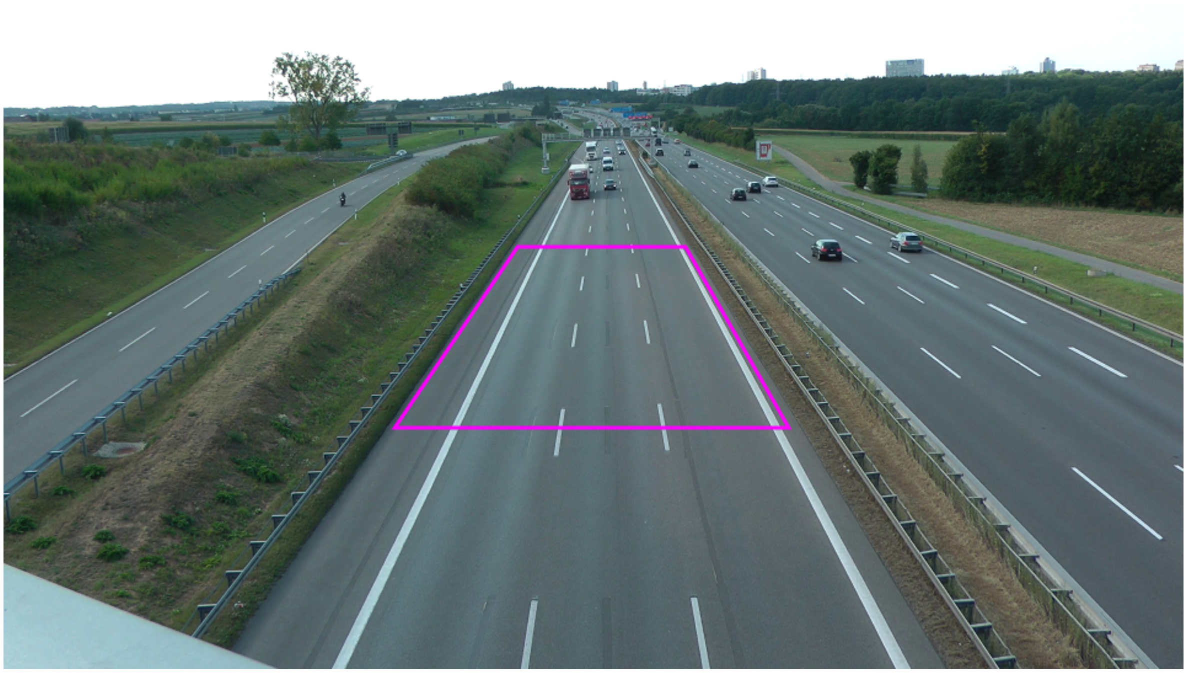


Figure 5.1: The pink marked area is the window used to rate wheter the tracking was successful or not. This does not count for the seperate evaluation of the complete tracking of a vehicle.

tracking are very time consuming tasks. The position estimation and the template matching in contrary is very light weight. The results are presented in the Table 5.1.

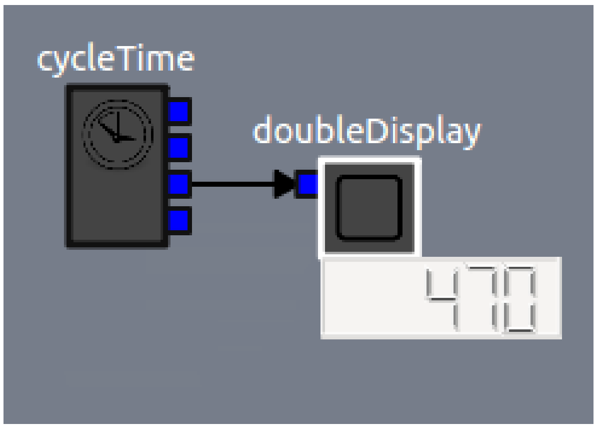


Figure 5.2: The two provided plug-ins of FloDiEdi to clock the average time of a step for the evaluation of the computational time. The displayed number is the clocked time for the third approach.

Approach	**Average computational time**
Approach I	3,402 milliseconds
Approach II	3,011 milliseconds
Approach III	470 milliseconds

Table 5.1: This table shows the average computational time for one frame. Averaging is done over ten consecutive frames. The fastes and near real-time approach is the third approach.

5.1.2 Vehicle detection rate

The reliability of the approaches consits mainly of the successful detection and the tracking of the vehicles. Only correct detected vehicles are counted here. Table 5.2 shows the result of the successful detected vehicles. There is also a table showing the vehicles being grouped together as these are too close to each other or the shadow is being falsely detected as part of the vehicle. This can be seen in Table 5.3.

Approach	**Detected vehicles**
Approach I	28 vehicles
Approach II	27 vehicles
Approach III	24 vehicles

Table 5.2: The number of successfully detected vehicles. A total of 30 vehicles are in the evaluated time frame.

Approach	**Grouped vehicles**
Approach I	3 vehicles
Approach II	3 vehicles
Approach III	1 vehicles

Table 5.3: During the evaluation of the video a small number of vehicles are falsely detected as these vehicles are too close together.

5.1.3 Tracking

Successful tracking is important to know how many vehicles are on the street and to do speed calculations. If the tracking is not working for every new frame there will be completly new vehicles thus making it hard to make any assumption about the traffic scene. For the evaluation of successful tracking are two distinctions made. The first is that the the tracking has to work in the predefined area which is shown in Figure 5.1. Second the tracking works in the whole image from the appearance of the vehicle at the top of the image to the vanishing at the bottom. In Table 5.4 the results of the tracking in the pink window are shown. Table 5.5 shows the result of the vehicle tracking in the whole image.

Approach	**Successful tracked vehicles**
Approach I	2 vehicles
Approach II	10 vehicles
Approach III	24 vehicles

Table 5.4: The amount of vehicles which were successfully tracked in the pink window which is shown above in Figure 5.1.

Approach	**Successful complete tracked vehicles**
Approach I	0 vehicles
Approach II	7 vehicles
Approach III	22 vehicles

Table 5.5: The amount of successfully tracked vehicles in the whole image.

5.1.4 Vehicle type detection

In this evaluation the correct type of vehicle is probed. There are three types where the approaches have to distinguish. These three types are cars, motor trucks and motor bikes. Distinction of these types is important to get the information about the load of the different types. Motor trucks with a trailer are much heavier than a normal car which means the streets gets more damaged by these heavy vehicles. The first approach uses the size in pixels of the vehicle to estimate the type. For the second approach the type estimation is done by using the ratio of height to width. The third approach uses the real size of the bounding box of the vehicle. Therefore the size of the box in pixels was multiplied with the pixel size in meters. In Table 5.6 is the result of the evaluation of the successful type detection.

5.2 Appraisal of results

Starting with the first approach which ranked in last position. The overall performance and reliability of the first approach was not satisfying. Concerning the performance it is very slow

Approach	**Correct detected vehicle type**
Approach I	19 vehicles
Approach II	16 vehicles
Approach III	20 vehicles

Table 5.6: Most correct vehicle types were detected by the third apporach. This is due to the consideration of real world sizes of the vehicles.

with 3402 milliseconds processing time for one frame. The detection rate was the highest but it also detects the most grouped vehicles. Tracking was not working reliable as only two vehicles were tracked in the marked area and no tracking in the whole image.
The second best in the evaluation is the second approach. Performance was slightly better than the first with 3011 milliseconds for one frame. The detection of two or more vehicles grouped to one vehicle was the same as the first approach but it was a bit less accurate with the detection of single vehicles. Tracking worked in comparison with the first approach, with five times more tracked vehicles, much better.
The best approach is the third approach. The reason is that it has the fastest computation time of the three approaches for one frame with 470 milliseconds. The detection rate was the worst but also only one detection of grouped vehicles. For the tracking it was the best, even the tracking over the complete image was working very well.

6 Conclusion of work and outlook

As image processing is used in a wide field of applications such as for surveillance, in vehicles as supportive system or even in home entertainment systems there will be much more to come. Nowadays it is possible to use high resolution cameras as these are located in a reasonable price segment. It depends on the actual field of application. With a look at the Microsoft Kinect® sensor the with a maximum camera resolution of 640×480[1]. For body movement detection this is an sufficient resolution. But more precise details cannot be recognised such as fingers or finger movement.
Concerning the overall bad performance in achieving real-time computation, as stated in the previous chapter, the CPU is not cutting-edge technology. So running an approach at 25 or even 50 frames per second was tried to achieve. In my opinion real-time in terms of traffic surveillance systems is achieved if the frame rate is lower than 25 frames per second which should suffice. If storing the captured images to a hard drive with an resolution of 1920×1080 and 50 frames per second the demand of memory is big for long term surveillance. For an 11 minutes long recording are over 2GB of space are needed and for a whole day of recording around 262GB. To increase the overall performance and storage size it is possible to resize the image as the used high resolution images may not be mandatory. Neglecting the amount of storage needed there are some ideas to improve the results and increase reliability.
To increase the vehicle extraction rate it is possible to make an update the background every n-th frames. This way changes in illumination may not have a big impact on the system. But this does not necessarily solve the problem when the weather situation changes to heavy rain, thick fog or heavy snow. If this is the case other sensors have to be included in the system such as laser or radar sensors. In the paper by Foresti and Snidaro a sensor network is used to handle changes in the wheater and day and night changes [FS02]. The collected data of the different sensor (infrared, radar, optical, ...) has therefore be fused.
Instead of template matching it is possible to use Speeded Up Robust Features (SURF) and a different templates of vehicles and vehicle types to detect and to typify them. SURF is able to detect feature points in image [BETVG08]. If a matching template is found then the type is known and if there are only different kind of vehicles on the street, could even be used without tracking, but this may not appeal to real world examples as there is only a limited number of different vehicles. Or if not many templates are available, it is possible to use an evolutionary algorithm that has the prototype of a motorbike, car and motor truck and let the program achieve a better distinction between the types.
Apart from the direct methods of extracting, detecting and tracking of vehicles there are other things that have to be considered when using this for traffic surveillance. The problem

[1] http://msdn.microsoft.com/en-us/library/microsoft.kinect.depthimageformat.aspx

with several vehicles getting grouped together for example. It could be better to use image subtraction in a different color model than the RGB such as the HSV (**H**ue, **S**aturation, and **V**alue). Also the handling of vehicles being occluded by other vehicles, animals or objects other than vehicles on the road and also the handling of wrong way drivers, as adressed in the introduction. To detect wrong-way driver one can think of different ideas. The first idea is that the windscreen and the headlights can not be detected. A vehicle with a special coachwork could be detected false and a false warning is sent out. This is not very reliable. The other and better idea because is to check the position of the vehicle during the speed calculation. During the calculation the previous and current position of one vehicle is processed and a warning can be posted if the vehicle is driving with a negative speed compared to the other vehicles in the image. This is very effective because the property of having a negative speed, compared to the other vehicles, is a unique property that wrong-way driver have.
To give a general speed up of the approaches it is possible to make use of multithreading and/or GPU-accelerated programming. For taking advantage of a fast GPU there is a technique which was developed by nVidia called CUDA (**C**ompute **U**nified **D**evice **A**rchitecture). On the website of nVidia is claimed that there is a performance boost in FFT of ten times with the NVIDIA cuFFT library and the NVIDIA cuSPARSE library can speed up sparse matrix operations[2].
If work is based on this thesis I would recommend using the third approach as it is light weight in terms of needed performance and achieves high reliability. But instead of the position estimation used in the approach it is better to use a Kalman filter for higher accuracy in terms position estimation[Kal60].

[2]https://developer.nvidia.com/gpu-accelerated-libraries

A Apendix

A.1 Instructions for using the vehicleDetection plug-in

General information about the plug-in

The plug-in provides three different methods to detect and track vehicles. These methods are FeatureTrack, FeaturePersp and TemplateMatch. FeatureTrack is a method which uses feature detection with the Shi-Tomasi feature detector and for tracking of the features the Lucas-Kanade feature tracker, implemented in a pyramidical approach. FeaturePersp uses perspective transformation to create a top-down view of the street and for detection and tracking similar to the method in FeatureTrack. The TemplateMatch method uses also perspective transformation but the vehicle position is estimated and affirmed by template matching of the found vehicle. The information extracted by each method is drawn on the output image. If selected by the user the data is written to the folder “XMLData”, which will be created if it does not already exists, in the home directory. The plug-in has one inport and one outport also an image and a button labeled “clear”. As input type for the inport and outport is the OpenCV type Mat used. The view shows the current image from the inport. The user can set the points on the image which define the perspective transformation (marked red) and the lanes (marked green). This is illustrated in Figure A.1. By clicking on the clear button all selected point in the view are deleted.
For the input the plug-in VideoFile can be used. There has to be a video file selected and the outport connected to the inport of the VehicleDetection plug-in. The outport can be connected to the imshow plug-in to display the images. To make the plug-in flexible and usable for different kinds of views on streets there are a number of settings to make this possible. To customize the plug-in the possible settings are:

- The type of method used.
- Frame rate of the input video.
- The number of lanes.
- The distance between two specified points.
- Write the data to a file.
- Extract the background to a file.
- Set number of frames used for background extraction

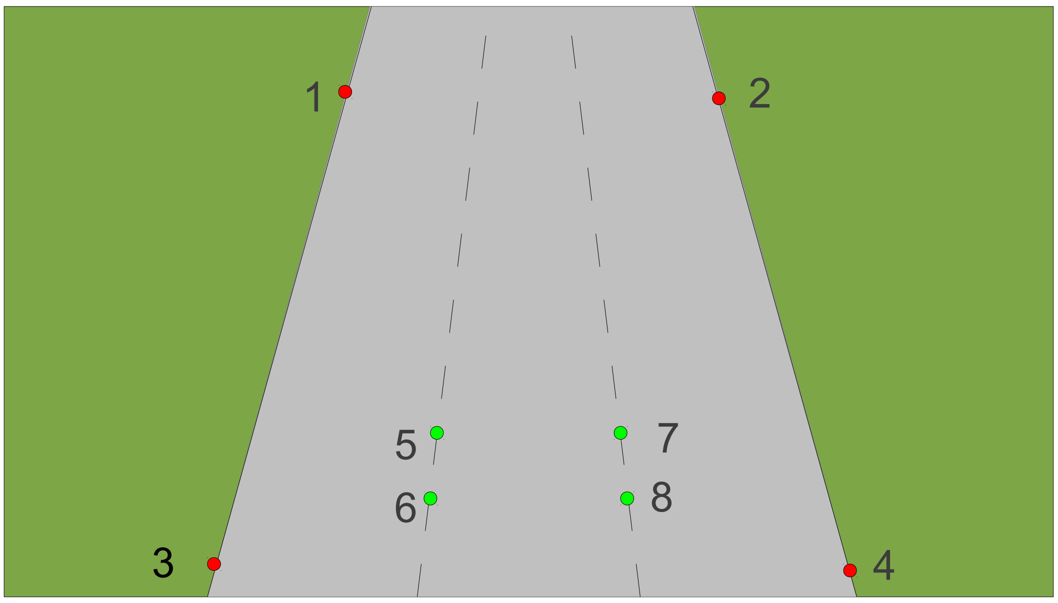
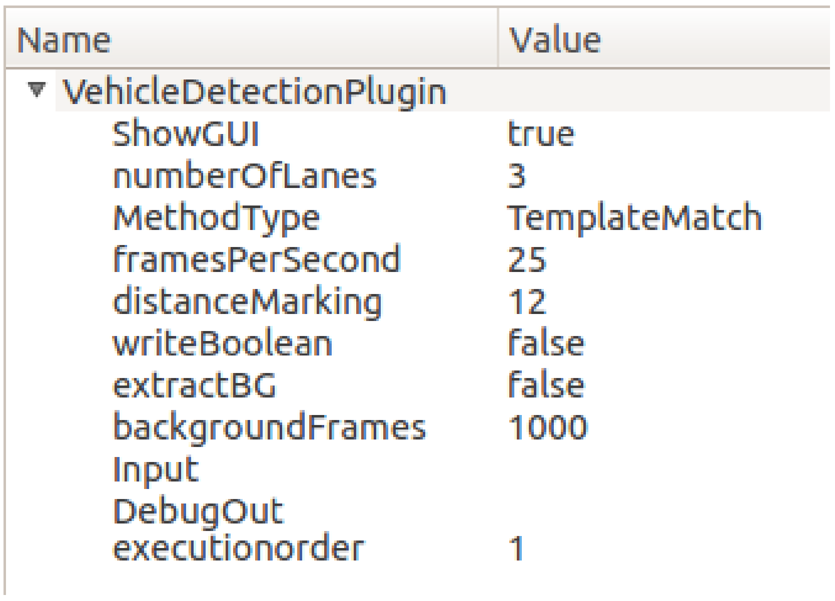


Figure A.1: This is an example of a street with 3 lanes. At first the ROI points in red have to be selected and afterwards the lane points in green. The lane points need to be set to the known distance for correct calculation of speed.

The user has to provide a 3-channel uchar matrix called “backgroundMatrix” where the background is stored. The filename is fixed to “background.yml” and the file must be placed in the home directory or the directory the user starts FloDiEdi. For undistorting the image a file called camera-parameters.yml must be provided which contains the intrinsic camera matrix which is named “intrinsicMatrix” and the distortion matrix called “distortionCoefficientMatrix”. When selecting FeatureTrack in the method selection field the plug-in starts automatically. If TemplateMatch or FeaturePersp is selected, the user has to select four points, which define the perspective transformation, first. The order of setting the points is top left, top right, lower left and lower right. After the selection the user has to click on point which have a known distance, such as the distance between two distance markings on a lane. The points have to be set from left to right, so starting from the most left lane. Wheter the top or bottom point is chosen first do not matter. The available settings are shown in Figure A.2.

Detailed explanation of the settings

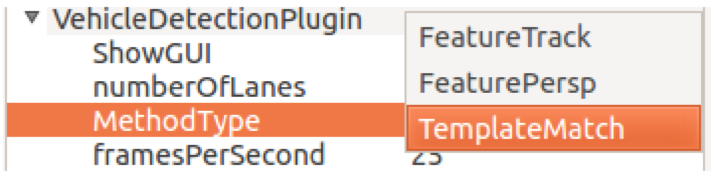
There is a number of settings that can be made to use the plug-in. All settings are shown Figure A.2. The numberOfLanes option allows the user set the number of lanes in the image. The standard value is set to three.



Name	Value
▾ VehicleDetectionPlugin	
ShowGUI	true
numberOfLanes	3
MethodType	TemplateMatch
framesPerSecond	25
distanceMarking	12
writeBoolean	false
extractBG	false
backgroundFrames	1000
Input	
DebugOut	
executionorder	1

Figure A.2: The available settings for the VehicleDetection plug-in are shown here. The number of lanes in the image, the type of detection and tracking, frames per second of the video and more can be changed.

The option method types lets the user select between the three methods, see Figure A.3.



▾ VehicleDetectionPlugin
ShowGUI
numberOfLanes
MethodType
framesPerSecond

Figure A.3: The user can select which method is used. The three available methods are FeatureTrack, FeaturePersp and TemplateMatch which represent the three approaches from the thesis.

By setting the value in frameRate, the user sets the frame rate of the video which is running. This value is used for speed calculations of the vehicles as Δt. The predefined value is 25.
The distanceMarking is used to set the distance between the first two points while selecting the lanes. This input is used by the second and third approach to do the speed caculations. It is important that the distance is choosen accurate as the distance is used for the speed calculation. It is preset to the value “12”, meaning the distance between the two points in the image is twelve meters. In Germany the distance between two road markings is mostly twelve meters on a motor way [RMS80].
WriteBoolean can be set to true to enable writing the collected data to a file. The file is located in the folder “XMLData” in the home directory. The filename contains a timestamp of the time of starting.

extractBG is used, if set to true, to determine wheter to extract the background from a sequence of images. The amount of images used can be set by the user in setting backgroundFrames.

Bibliography

[Bay76] B. E. Bayer. Color imaging array, 1976. URL http://www.patentlens.net/patentlens/patent/US_3971065/en/. (Cited on page 15)

[BBS05] D. E. Butler, V. M. Bove, Jr., S. Sridharan. Real-time adaptive foreground/background segmentation. *EURASIP J. Appl. Signal Process.*, 2005:2292–2304, 2005. doi:10.1155/ASP.2005.2292. URL http://dx.doi.org/10.1155/ASP.2005.2292. (Cited on page 45)

[BETVG08] H. Bay, A. Ess, T. Tuytelaars, L. Van Gool. Speeded-Up Robust Features (SURF). *Comput. Vis. Image Underst.*, 110(3):346–359, 2008. doi:10.1016/j.cviu.2007.09.014. URL http://dx.doi.org/10.1016/j.cviu.2007.09.014. (Cited on page 71)

[BF81] R. C. Bolles, M. A. Fischler. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981. doi:10.1145/358669.358692. URL http://doi.acm.org/10.1145/358669.358692. (Cited on page 19)

[BK08] G. Bradski, A. Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library.* O'Reilly Media, 2008. (Cited on pages 18 and 39)

[Bou00] J.-Y. Bouguet. Pyramidal Implementation of the Lucas Kanade Feature Tracker Description of the algorithm, 2000. URL http://robots.stanford.edu/cs223b04/algo_tracking.pdf. (Cited on page 36)

[Bru10] S. Brutzer. Background Subtraction in der Videoüberwachung, 2010. URL http://elib.uni-stuttgart.de/opus/volltexte/2010/5523. (Cited on page 45)

[Bur82] P. J. Burt. Fast algorithms for estimating local image properties. Technical Report IPL-TR-022, Rensselaer Polytechnic Institute (Troy, NY US), 1982. URL http://dx.doi.org/10.1016/S0734-189X(83)80049-8. (Cited on page 20)

[Can86] J. Canny. A Computational Approach to Edge Detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-8(6):679 –698, 1986. doi:10.1109/TPAMI.1986.4767851. URL http://dx.doi.org/10.1109/TPAMI.1986.4767851. (Cited on page 20)

[CT65] J. W. Cooley, J. W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of Computation*, 19(90):297–301, 1965. (Cited on page 20)

[FS02] G. Foresti, L. Snidaro. A distributed sensor network for video surveillance of outdoor environments. In *Image Processing. 2002. Proceedings. 2002 International Conference on*, volume 1, pp. I–525 – I–528 vol.1. 2002. doi:10.1109/ICIP.2002.1038076. (Cited on page 71)

[GO00] J. Gonzalez, U. Ozguner. Lane detection using histogram-based segmentation and decision trees. In *Intelligent Transportation Systems, 2000. Proceedings. 2000 IEEE*, pp. 346 –351. 2000. doi:10.1109/ITSC.2000.881084. (Cited on page 48)

[GS08] R. García, D. Shu. Vision based Vehicle Tracking using a high angle camera. 2008. URL http://www.ces.clemson.edu/~stb/ece847/projects/VISION_BASED_VEHICLE_TRACKING.pdf. (Cited on page 45)

[GVL96] G. Golub, C. Van Loan. *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, 1996. (Cited on page 29)

[GW92] R. C. Gonzalez, R. E. Woods. *Digital Image Processing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 1992. (Cited on pages 19, 20, 22 and 47)

[HZ04] R. Hartley, A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2004. URL http://dx.doi.org/10.1017/CBO9780511811685. (Cited on pages 28 and 29)

[Jäh05] B. Jähne. *Digitale Bildverarbeitung*. Springer Berlin Heidelberg, 2005. doi:10.1007/3-540-27384-0_20. URL http://dx.doi.org/10.1007/3-540-27384-0_20. (Cited on page 32)

[JLR87] A. S. Jensen, L. Lindvold, E. Rasmussen. Transformation of image positions, rotations, and sizes into shift parameters. *Applied Optics*, 26(9):1775–1781, 1987. (Cited on page 19)

[JZWL09] R. Jia, H. Zhang, L. Wang, J. Li. Digital Image Stabilization Based on Phase Correlation. In *Artificial Intelligence and Computational Intelligence, 2009. AICI '09. International Conference on*, volume 3, pp. 485 –489. 2009. doi:10.1109/AICI.2009.489. (Cited on page 29)

[Kal60] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960. URL http://www.cs.unc.edu/~welch/kalman/media/pdf/Kalman1960.pdf. (Cited on page 72)

[Ker11] S. Kernbach. Evolutionäre und kollektive Robotik Lecture, 2011. (Cited on page 26)

[Lev11] P. Levi. Image Understanding Lecture, 2011. (Cited on pages 23 and 31)

[LIS] European Commission - Advanced Safety and Driver Support in Essential Road Transport. URL http://www.project-asset.com/data/ASSET_DEL3_2_Smart_information_provision_and_regulation_support_concepts_V2_0.pdf. (Cited on pages 10, 52, 58 and 64)

[Mar04] D. Marsh. *Applied Geometry for Computer Graphics and CAD*. Springer Undergraduate Mathematics Series. Springer, 2004. (Cited on page 28)

[MGK00] J. Matas, C. Galambos, J. Kittler. Robust Detection of Lines Using the Progressive Probabilistic Hough Transform. 78(1):119–137, 2000. (Cited on page 47)

[MW06] G. Merziger, T. Wirth. *Repetitorium der höheren Mathematik*. Binomi Verlag, 5 edition, 2006. (Cited on page 19)

[Pan10] Panasonic. *Panasonic Bedienungsanleitung High Definition Camcorder Modell-Nr. HDC-SD700 HDC-SD707 HDC-TM700 HDC-HS700*. Panasonic, 2010. URL http://dlc.panasonic-europe-service.com/EUDocs/GetDoc.aspx?did=189273&fmt=PDF&lang=de&src=3. (Cited on page 16)

[RMS80] Forschungsgesellschaft für das Straßenwesen - Richtlinien für die Markierung von Straßen RMS Teil 1 Abmessungen und geometrische Anordnung von Markierungszeichen RMS-1, 1980. (Cited on pages 41, 52, 61 and 75)

[Ser88] J. Serra. *Image analysis and mathematical morphology: Theoretical advances*. Image Analysis and Mathematical Morphology. Academic Press, 1988. URL http://dl.acm.org/citation.cfm?id=1098652. (Cited on page 22)

[SM09] M. Shakoor, M. Moattari. Digital Image Stabilization Using Histogram-Based Sorting. In *Knowledge Acquisition and Modeling, 2009. KAM '09. Second International Symposium on*, volume 3, pp. 266 –268. 2009. doi:10.1109/KAM.2009.125. (Cited on page 29)

[ST94] J. Shi, C. Tomasi. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on*, pp. 593 –600. 1994. doi:10.1109/CVPR.1994.323794. (Cited on pages 20 and 35)

[Svo06] T. Svoboda. Homography from point pairs, 2006. URL http://cmp.felk.cvut.cz/cmp/courses/XE33PVR/WS20072008/Lectures/Geometry/homography.pdf. (Cited on page 28)

[Tay63] C. Taylor. *Geometrical conics including anharmonic ratio and projection: with numerous examples*. Macmillan, 1863. URL http://archive.org/details/cu31924031263779. (Cited on page 27)

[TK91] C. Tomasi, T. Kanade. Detection and Tracking of Point Features. Technical report, Carnegie Mellon University, CMU, 1991. URL www.ces.clemson.edu/~stb/klt/tomasi-kanade-techreport-1991.pdf. (Cited on page 36)

[Ziv04] Z. Zivkovic. Improved adaptive Gaussian mixture model for background subtraction. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 2, pp. 28 – 31 Vol.2. 2004. doi:10.1109/ICPR.2004.1333992. (Cited on page 45)

All links were last followed on February 17, 2013.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben.
Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet.
Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens.
Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht.
Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

(Andreas Harnisch)