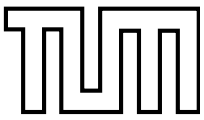


INSTITUTE FOR REAL-TIME COMPUTER SYSTEMS
TECHNISCHE UNIVERSITÄT MÜNCHEN
PROFESSOR G. FÄRBER

Robot Control and Lane Detection with Mobile Phones

Bernhard Zeisl

Bachelor's Thesis

Robot Control and Lane Detection with Mobile Phones

Bachelor's Thesis

Executed at the Institute for Real-Time Computer Systems
Technische Universität München
Prof. Dr.-Ing. Georg Färber

Advisor: Assoc. Prof. Dr. rer. nat. habil. Thomas Bräunl

Author: Bernhard Zeisl
Arcisstraße 31
80799 München

Submitted in February 2007

Acknowledgements

I would like to thank my advisor, Associate Professor Thomas Bräunl for providing me with such an interesting and challenging project and for all the assistance he has given to me. Under his supervision I enjoyed the freedom to give own ideas and approaches a chance, however he always supported me with helping advices.

A special thanks to Adrian Boeing, who always had an open ear, when I came up with some problems or questions concerning my thesis. It was a pleasure to work with him and get his point of view.

I want to thank the institute and its staff for the opportunity of writing this thesis as well as for all the assistance I got.

Thanks to my parents for all the things they have done for me. They always supported me and especially in the end helped me to manage both, thesis and preparation for my study abroad year in Australia.

Laura, for all her kindness, help and tolerance over the semester and especially during the writing of the thesis.

Finally, I would like to thank all my other friends for their assistance, patience and friendship during the year.

Abstract

One cannot imagine daily life today without mobile devices such as mobile phones or PDAs. They tend to become your mobile computer offering all features one might need on the way. As a result devices are less expensive and include a huge amount of high end technological components. Thus they also become attractive for scientific research.

This work addresses the exploration of mobile phone usage for robot control and lane marking detection. Basis for the work was an already available controller for mobile robots, named EyeBot. I developed an application, which allows to operate the controller from the mobile phone via a Bluetooth link. The results are promising and facilitate the usage of the mobile as remote control for other tasks. However, I detected some restrictions in terms of communication as the mobile phone was not able to process a continuous video stream fast enough.

On top of the communication between the robot controller and the mobile phone, there are several applications possible, which might take over robot control. In this context I developed a sophisticated algorithm for lane detection, which uses the input data from the mobile phone's camera. To address the special aspects of the lane marking appearance - they exhibit a constant orientation over different parts of the recorded image - I used the theory of steerable filters to get an optimized edge detection. Further on I developed a line extraction and clustering algorithm, which separates the binary edge detected image in several tiles. Lines in these tiles can then be described in terms of orientation of the eigenvector by performing a principal component analysis.

The application was implemented platform independent to run on any system supporting C++. The outcome was an application on the mobile phone, which lends itself to be further used for lane tracking and direct or remote robot control.

Contents

List of Figures

List of Tables

List of Symbols

API	Application Programming Interface
CPU	Central Processing Unit
FPU	Floating Point Unit
GUI	Graphical User Interface
IDE	Integrated Development Environment
IO	Input-Output
LAN	Local Area Network
PC	Personal Computer
PCI	Peripheral Component Interconnect
RAM	Random Access Memory
RCS	Institue for Real-Time Computer Systems
ROM	Read Only Memory
SDK	Software Development Kit
SDL	Simple Directmedia Layer
STL	Standard Template Library
UI	User Interface
USB	Universal Serial Bus
UWA	University of Western Australia

Chapter 1

Introduction

1.1 Motivation

This work was conducted in cooperation with the *Robotics and Automation Laboratory at the University of Western Australia, Perth* supervised by Prof. Dr. rer. nat. habil. Thomas Bräunl [34]. Research in this lab concentrates on all types of autonomous mobile robots, including intelligent driving and walking robots, autonomous underwater vehicles, and unmanned aerial vehicles. The work also includes design of embedded controllers and embedded operating systems, as well as simulation systems. In this context my challenge was to analyze the use of inexpensive, high-end mobile phones as embedded controllers.

Figure 1.1: Example robots at the lab using the EyeBot controller

So far there exists a specialized embedded controller, named *EyeBot* for industrial and robotics applications. It is a controller for mobile robots with wheels, walking robots or flying robots (Figure 1.1).
Embedded controllers, such as the EyeBot controller are a necessity for building small mobile robots. Unfortunately, low volume production makes these controllers quite expensive and it is difficult to keep up with increasing hardware speed.

On the other hand the following points make the use of a mobile phones interesting as alternative processing and control unit:

- A mobile phone offers high performance at a low price.
- Product live cycles are very short. Therefore new phones are always at the technological edge.
- Periphery like camera and Bluetooth are included by default.
- Development tools and compilers are already existent. In combination with a large developer community this enables a quick development of new applications.

Therefore, the idea to use commodity consumer hardware, such as mobile phones or PDAs is obvious.
Note that when using a mobile phone, still a small external board for sensor input and actuator output (for example motor drivers) is required in order to drive a robot.

As a result it is reasonable to further explore two main features of a mobile phone. First, there must be a communication possible between the mobile phone and the remaining (low level) embedded controller. For this purpose the use of Bluetooth as communication technology enjoys the advantages of a high speed link in combination with a wireless interface.

Second, the processing capabilities of a mobile phone should be analyzed in more detail. It would be convenient to be able to port already existing algorithms to the mobile device. The choice to develop and implement a lane marking detection algorithm was due to the following facts. One the one hand, the camera of the mobile phone is included in the testing and its usability can be analyzed, too. On the other hand, the idea was to build a lane detection and tracking application for a mobile phone that everybody is able to easily install and use in his own car. Devices can be attached to the vehicle wind shield, as it is possible with portable navigation systems. The communication with the car system may then be possible via Bluetooth, e.g. for communication turn signals or drving speed.

1.2 Objectives

In order to test the communication capabilities an application has to be developed, which allows a remote control of the EyeBot controller. The EyeBot controller already offers a Bluetooth interface, which is prepared for a remote control. As a result the application

on the mobile phone has to receive data via an established Bluetooth link, interpret the message and execute the sent commands. Moreover, it must also offer an opportunity to control the EyeBot via the mobile phones buttons.

If this communication is possible the lane detection application can sit on top and send appropriate driving commands via the Bluetooth link to the controller. However the task was to develop and implement a lane detection algorithm with low computational effort. Constraints as limited computing power, abdication of complex operations (e.g. Hough transform) and scalability in terms of resolution, computing power and precision have to be taken into consideration.

The programming language C++ was chosen instead of Java, because in image processing tasks C++ was expected to have a more efficient execution. The closer hardware programming of C++ offers an advantage over the interpreted Java language.

Moreover, the lane detection algorithm must be independent from the mobile phone platform to be able to run on *any* system supporting C++. A further goal at the Robotics and Automation Laboratory is to implement the algorithm on the next generation model of the EyeBot.

1.3 Hardware Devices

The used hardware devices are the EyeBot controller and the mobile phone. They will be shortly described in the following section in terms of their capabilities.

1.3.1 EyeBot Controller

The EyeBot consists of a powerful 32-Bit microcontroller board with a graphics display and a digital grayscale or color camera. The camera is directly connected to the robot board, hence no frame grabber is used. This allows to write powerful robot control programs without a big and heavy computer system and without having to sacrifice vision.

It comprises following important features (from [28]):

- 25MHz 32bit Controller (Motorola 68332)
- 2MB RAM, 512KB ROM
- Large graphics display (LCD) with 128×64 pixel
- Integrated digital color camera with 64bit and QCIF resolution of 176x144 pixels.
- Bluetooth communication module with emulation of serial communication
- Running self developed RoBIOS as operation system.

Figure 1.2: Nokia 6260 mobile phone

A next generation EyeBot controller has been developed and released. It is based on an ARM9 processor and includes Bluetooth, LAN, USB and two stereo cameras. An embedded Linux distribution is used to operate the controller. This new controller will also support remote control from the mobile phone. As mentioned before the lane detection algorithm will be developed in a way that it will also work with this model. For this task the mobile phone serves as a good performance reference, because it also based on an ARM processor.

1.3.2 Mobile Phone

The used mobile phone was a Nokia 6260 [8]. It was announced on the 14th of June in 2004 and comprises the following features:

- ARM9 32-bit CPU with 123 MHz
- Shared memory for storage is 8MB
- Operating system Symbian OS v7.0s (stack size with 4 kb, unlimited heap size). Programs are executed in place in ROM or flash RAM, rather than being loaded into RAM
- Developer platform Nokia S60 2nd edition, feature pack 1
- 176x208 pixel display with 16 bit color depth
- Integrated digital camera with VGA resolution of 640 × 480 pixels
- Bluetooth support
- Symbian API for Bluetooth and camera

Chapter 2

Related Work

In the following, I will present related work concerning the use of embedded controllers and vision systems. The second part concentrates in more detail on approaches for lane detection.

2.1 Embedded Controller and Vision System

At the Robotics and Automation Lab a family of small autonomous mobile robots has been designed in the last years. All these robots share the EyeBot controller and a common operating system. These robots are powerful enough to perform simple image processing tasks on board and in real-time. Although the controller runs at moderate speed, it is fast enough to compute basic image operations on a low resolution image in real time. For example the whole process of image acquisition, Sobel edge detection and display of the result for a 80x60 grayscale image can be performed at a rate of about 10 frames per second. This enables the robots to perform several complex tasks such as navigation, map generation, or intelligent group behavior [2, 27].

Further, the robots are able to communicate with each other via a network, named the EyeNet. In this continuously changing environment with mobile agents coming and going a self-configured network is the only option [37]. Hence the EyeNet works without a central control using a wireless token ring network. Schmitz extended the EyeNet in his diploma thesis [29] to also operate with Bluetooth and wireless LAN. This offers the advantages of higher data rates and existing protocols included in the network. He further developed remote control and monitoring applications for both cases to run on a PC. My work attaches to this remote control application for the EyeBot, but runs on a standard mobile phone.

In terms of an embedded vision system Daimler-Chrysler developed a *lane departure warner* and a *lane-change assistant* [3]. The formers alerts the driver to an unintentional change on the motorway, whereas the latter warns against risky lane changes. As Figure

2.1 shows, most accidents are caused as a result from inattention of the driver. Long stretches of road covered at constant speed in monotonous surroundings can easily give rise to inattentiveness. Therefore, the need for driver assistance systems is obvious.

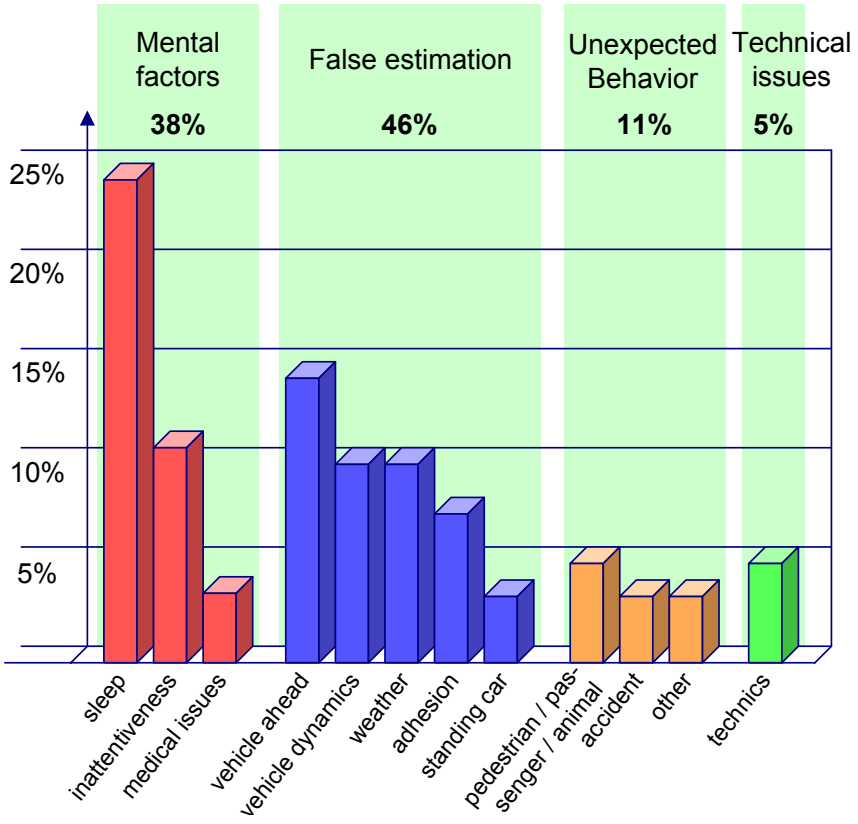


Figure 2.1: Causes for deadly traffic accidents (source [31])

The *lane departure warner* was designed especially for trucks. A miniature camera is mounted behind the windscreen. Thereby the vehicle position within the traffic can be determined by using lane markings as a guide, which are evaluated on real-time basis of the provided images. A warning is issued if the driver fails to take evasive action within a certain time. A special feature is that a stereo sound alerts the hazard in the direction of the (crossed) lane marking. Thus a instinctive response of the driver shortens reaction time. One year after market launch the system was installed in 600 Mercedes-Benz Actros trucks [3].

A further development covers a steer-by-wire system based on a lane departure recognition. Not only a acoustic warning is issued, but also the *active lane assistant* will automatically intervene to initiate the necessary corrective steering movements.

2.2 Lane Detection

In this section I will take a look at the current state of the art in lane detection. The aim is to give a brief overview of the research which has taken place in the past 15 to 20 years. For these purposes it is useful to group the methods into categories in terms of their contribution. My investigations in lane detecting methods have shown that there exist similarities in the way algorithms work. Considered works were [1, 4, 5, 13, 14, 16, 18, 17, 19, 32, 35, 36, 38]. In fact all lane detection and lane position tracking systems follow a similar flow. This common structure is illustrated in Figure 2.2.

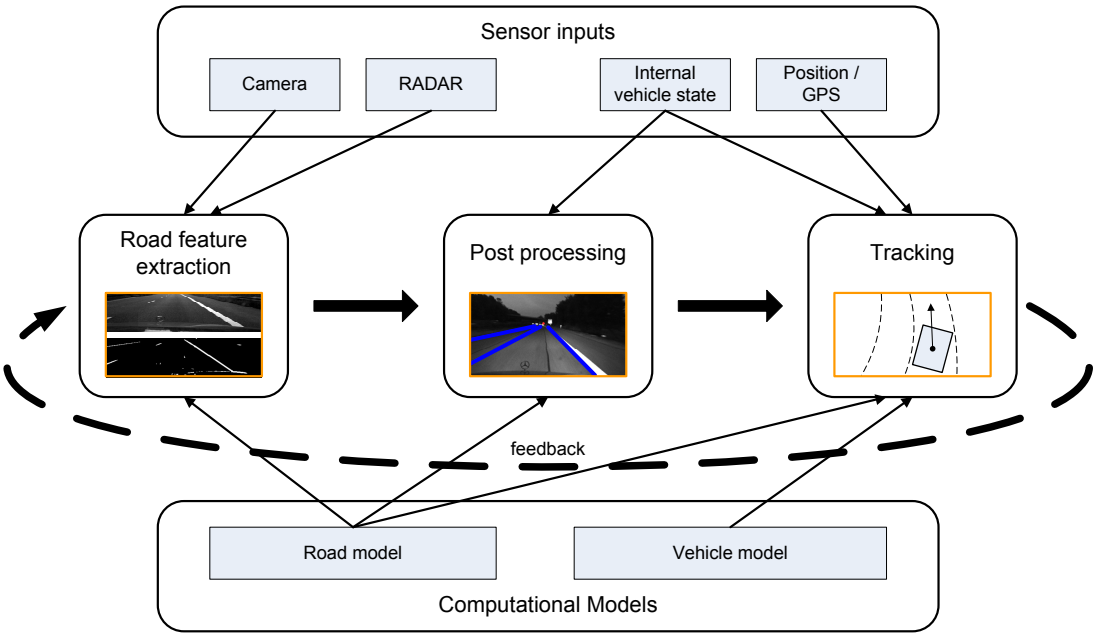


Figure 2.2: A common system structure for lane position detection systems, as it is also suggested in [17]

2.2.1 Road Modeling

First, a model for the road is proposed. This can be as simple as straight lines or more complex. An underlying road model in the whole approach of lane position detection will heavily increase system performance. This is because the model supports in eliminating false detected lines and presents the decision basis for further outlier removal.

A variety of different road modeling techniques have been established. An outline is given in the following.

Due to the perspective distortion of the image all lane markings intersect in one point, the vanishing point. With the same argumentation road markings form parallel lines in the inverse perspective warped image. Both properties can be used to define the direction of possible lane markings in the recorded video frame.

However, this approach is only true if the road is almost straight. If the road is curved like in urban areas a method of curve parameterization is needed. It is possible to model roads with piecewise constant curvatures. In situations where this approximation is not sufficient, a model with deformable contours such as splines can lead to better results[35, 36]. On the other hand a more detailed description of the road causes more expenses in computational effort. Hence a trade-off between accuracy and complexity has to be made.

In a highly populated environment like towns, streets possess a complex shape and characteristic. In such situations it is possible to use maps of the area in combination with up to date GPS data to be able to have an idea what the road course looks like.

In all cases it is essential to choose the model according to the type of used system and environment, which the system is intended for. For example a stable control system might only require a look-ahead of about 10 meters. As the upper part of the recorded image can be cut and only the close-up range is considered in the following calculation steps, a linear road model is sufficient. However a lane departure warning system requires to predict the vehicle trajectory a few seconds ahead. For driving on a motor-way with an appropriate speed, this implies a look-ahead of 30 to 40 meters, or even more. In this situation a linear road model is not enough. Hence a parabolic or spline based road model will perform a lot better.

2.2.2 Road Marking Extraction

Next, a sensing system is used to gather information about the vehicle's environment. Other approaches of lane detection and tracking have used additional sensor information like GPS or radar for an improved positioning to enhance lane position estimates. However, in this paper I focus on information from a camera, because this is the sensor available on the mobile phone.

Road marking extraction is a key component in lane position detection, because it generates the input for all following steps. With a bad performing image feature extraction, the subsequent steps can be perfect, but the general outcome might not be satisfactory. In general road and lane markings show a huge variety of shapes, thus making it difficult to generate a single feature extraction technique.

Edge based techniques can work well with solid and segmented lines. However this method will fail with the detection of irrelevant lines. If an image contains many extraneous lines, the result will be unsatisfactory.
Edge detection might work unsatisfactory in a greater distance from the car, as objects are smaller. Therefore, a separation of the image in close-up range and far field is helpful. With emphasis on the near field, edges belonging to lane marking boundaries can be extracted well. With an iterative approach it is possible to just investigate an area in the range of previously detected lines (region of interest) [1].
A continuative method is also to take the direction of lane marking in the filtering procedure into account. Steerable filters offer such a tool to tune the edge filter in

direction of the expected lane orientation. The point is that a set of basis filters is adequate to steer the filter in any arbitrary orientation. Moreover, the filters are separable, which results in less computational effort. The application of steerable filters [16, 17, 18] has also shown that they deal well with different lightning situations, as well as with shadows.

Frequency based techniques on the other hand deal effectively with additional edges, but have problems with complex shadowing. The LANA system used in [14] moreover is restricted to diagonal edges, limiting the efficiency during lane change maneuvers.

Adaptive road templates build upon a matching of current road scenes with predefined textures. Before matching, the recorded image is inverse perspectively warped. This removes perspective distortion from the image and therefore reduces the number of required templates. These methods generally assume a constant road surface texture and therefore fail if this is not the case. However, the technique is useable for the far field of a road scene, where lane markings are difficult to identify by other methods [17].

Statistical criteria such as energy, homogeneity and contrast are also used to distinguish between road and non-road area [13]. This approach of lane boundary detection especially addresses the peculiarities of country road fields, where other methods might fail, because of the loose road boundaries.

Similarly to road modeling, an appropriate choice of a feature extraction method also depends on the environment and the type of system itself. A restriction to possibly appearing types of roads and system environments in combination with the optimization of the system to these situations will generate better results than trying to detect all possible variations of road markings without any limitations.

2.2.3 Postprocessing and Outlier Removal

In postprocessing the previously extracted features are used in combination with the defined road model to create and improve an estimate of the lane position. Moreover, postprocessing is one of the most important steps, as it ties together the previous feature extraction stage with the forthcoming tracking stage.

One of the most common used techniques is the Hough transform, which enables to identify lines in an image [6, 19]. The underlying principle of the Hough transform is that there exists an infinite number of potentially differently oriented lines, which pass trough any image point. Points that form a straight line exhibit the same theoretical line. Hence potential lines passing through most image points also represent a line in reality. The Hough transform produces quite good results, but demands high computational effort. Therefore, it is not suitable for my application, which requires low computational costs.

Another method is to validate extracted features [1, 32, 38]. This can be done by enhanc-

ing features based on orientation or likelihood. As an example features with an orientation in direction to the vanishing point are likely to represent lane boundaries. In the inverse perspectively warped image, possible line markings will show a slope in just a very small range, allowing to discard other detected features.

When using a second camera the additional depth information in stereo vision is applicable for culling features on elevation. Moreover, features are investigated in the left and right stereo image and a matching between them generates a more comprehensive statement about possible lane markings.

Approaches with neural networks and statistical methods like dynamic programming or Hidden Markov Models have also been developed [12]. As these techniques need an advanced computational approach, they are not considered any further.

The outcome of the postprocessing step is always an estimate of the current lane position based on the extracted features. To improve the performance, information from previous detected lines are considered as well. Thereby, an estimate for future lines is included, because lane position and orientation do not change rapidly between video frames. Moreover, this allows to limit the investigated region in the image to an area around already detected lines. In combination with the concluding tracking stage and a closed loop feedback, an improved estimation heavily supports the outlier removal in postprocessing.

2.2.4 Position Tracking

At last a vehicle model can be used to refine the gained estimates over time. The two most common tracking techniques used in lane detection systems are Kalman filtering and particle filtering.
The Kalman filter is a recursive estimator[6]. This means that only the estimated lines from the previous time step and the current extracted features are needed to compute the estimate for the current lines. It produces an ideal estimation under the assumption of Gaussian noise.
Particle filters offer the advantage that, with sufficient input data, they approach the Bayesian optimal estimate and, hence, can be made more accurate than the Kalman filter.

Feature extraction and vehicle position tracking are often combined into a closed loop feedback system. Hence, the tracked lane position defines an a priori estimate of the location and orientation of subsequently extracted features.

2.2.5 Resulting Assumptions

A significant improvement to the accuracy of lane detection can be achieved by applying some assumptions based on the road structure, lane shape and possible environment. The

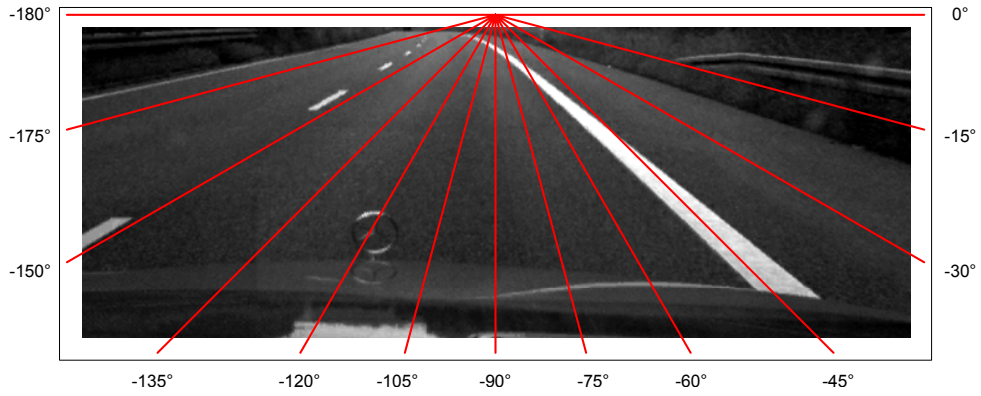


Figure 2.3: Predefined general lane marking orientation

following definitions have been made and are used in the further steps of my algorithm to detect possible lines and to remove outliers.

1. The automotive video sequences for which the algorithm will be tested, have been recorded at motorways or similar streets that do not feature strong curves. Hence, lane markings are represented as straight lines in the observed area. If curves occur they will be neglected.

2. Lane markings are displayed as white lines or dashed white lines, as they can be found on German roads.

3. The interesting part for lane position estimation lies only in the lower part of the recorded video frame. Therefore each image is cropped so that it contains just the recorded road.

4. All lines in the perspective distorted image are geared to predefined orientations as displayed in Figure 2.3. Therefore, they will intersect in the vanishing point, which is located for initialization in the middle of the upper image border, but will be updated subsequently in progress.

5. Roads do not show a steep incline, but are almost plain. If roads would have a rapid change in altitude difference, this would have impacts on the slope of lane markings in the image, as well as on the position of the vanishing point. Moreover, an adjusted part of the image would have to be cropped.

In comparison to the structure given in Figure 2.2 some parts are omitted for my approach. As sensor input there is only the video from the camera. Additional sensor inputs were not available for my development environment. In my approach there is no position tracking included, because the goal was to implement and test a lane detection algorithm and not to control a vehicle. Finally, the feedback of estimated lane markings was not implemented as there does not exist a reliable prediction. However, previously detected lane markings are taken into consideration in the postprocessing stage to support the removal of outliers. The final structure for my approach is illustrated in Figure 2.4.

In chapters 5 and 6 the remaining two stages for a lane marking estimation are discussed.

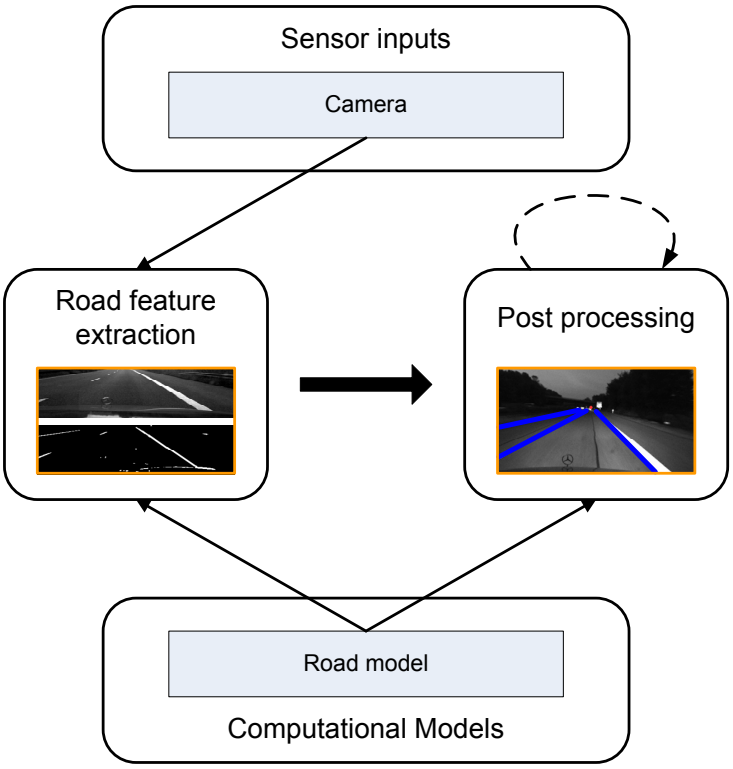


Figure 2.4: System structure for the covered lane detection approach

Chapter 3

Software Development

The operating system Symbian is intended to run on mobile phones. This profoundly affects the design of its software system.

Symbian is based on techniques similar to the PC industry. It uses a device driver architecture and offers an abstract API for each device. However, there are some important points, where Symbian OS is different from PC operating systems:

- Resources at a mobile phone are constrained. As an example the CPU is many times slower and there is less memory available.
- A mobile phone features no hard disk. This means that there is no disk-backed virtual memory. Therefore, the common assumption with PC operation systems that there is an infinite amount of space in which to place program and data files, does not hold true.
- Power management is critical, because the mobile phone is expected to function several hours between recharges and there are other uses of power such as phone calls.

As a result the operating system has to be compact and tackle errors such as out-of-memory and others, as Symbian OS systems virtually never reboot.

3.1 Symbian and the Nokia S60 Platform

Symbian itself has a flexible architecture, which allows different mobile phone platforms to run on top of the core operating system. The two most widely used are the S60 platform from Nokia and UIQ, which was developed by Symbian itself and is licensed to several phone manufactures, such as Sony Ericsson ans Motorola. The general system structure is illustrated in Figure 3.1. It can be seen that the S60 platform builds on top of Symbian OS. Hence all functionality of Symbian OS is useable when developing for a Nokia S60 platform plus a number of additional functions. These are provided through Avkon user

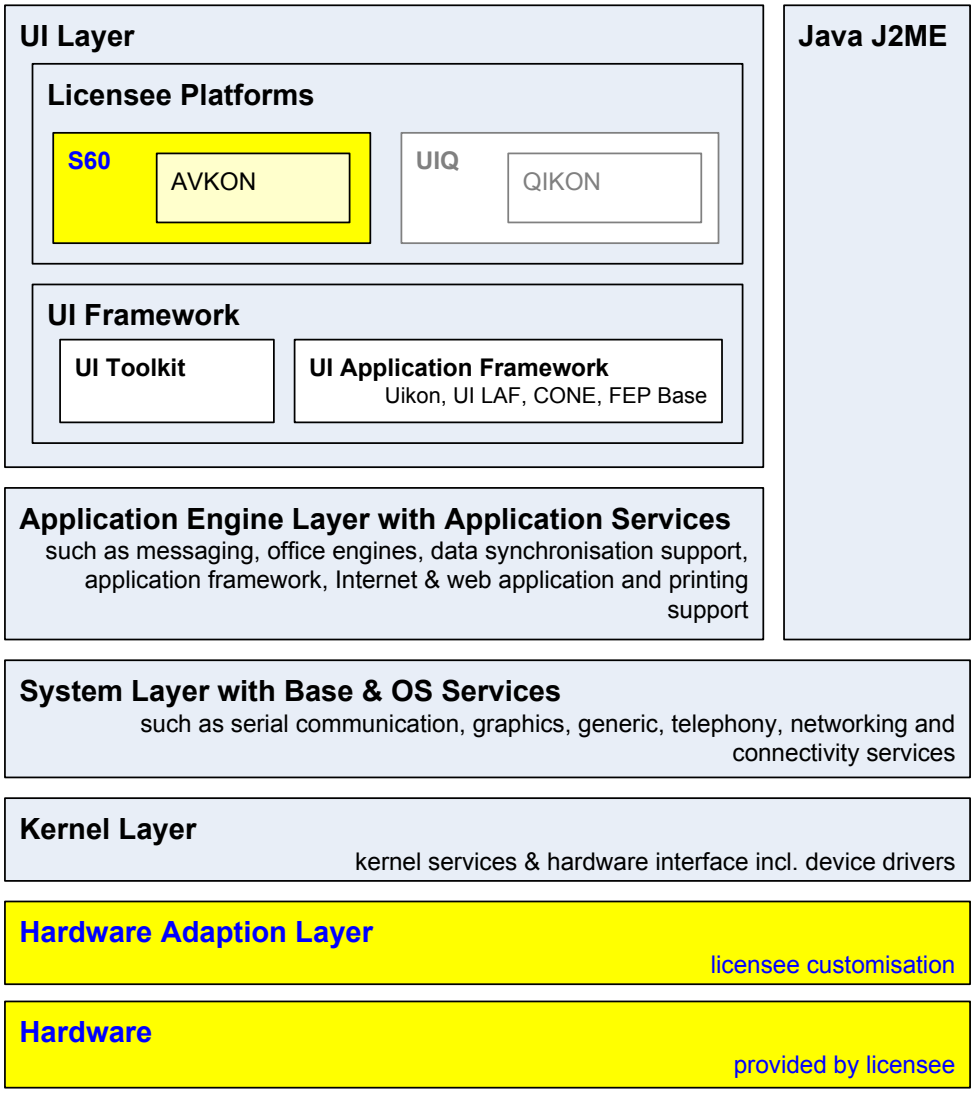


Figure 3.1: Symbian OS and the S60 platform (taken from [7] and [25])

interface layer added to the underlying Uikon application framework. Beside the mobile phone itself, a hardware adaption layer is also provided by Nokia, as it implements the interface between the specific hardware and the Symbian OS.

Once an application is running, "events" are channeled to it via the control environment CONE, another part of the Symbian OS framework. This component notifies an application of key presses, as well as more advanced events like machine shutdown or focus changes.

The Avkon layer on top of Uikon provides functionality specific to the S60 platform. A number of key base classes are provided for creating a S60 application. These include `CAknApplication`, `CAknDocument` and `CAknAppUi`. All S60 applications are derived from these three base classes.

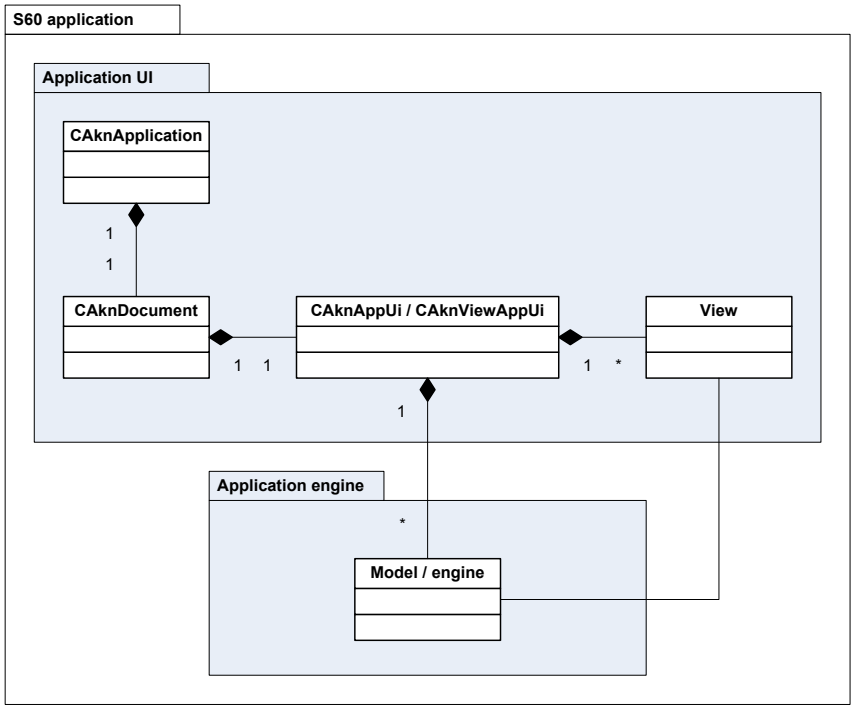


Figure 3.2: S60 application relationship (from [22])

Moreover, an S60 application is normally split into two parts, the engine and the user interface (UI), as a reason of aiding maintainability, flexibility and portability. The application engine (or also named the application model) deals with the algorithms and data structures needed to represent the applications data. The application UI deals with the on-screen presentation of the application data and the overall behavior of the application [11, 22]. In Figure 3.2 this relationship is illustrated, which hold true for every S60 application.

A more detailed description of Symbian and the Series 60 Platform can be found on Forum Nokia, which is a very good point to get started with Symbian development. In the following only important limitations of Symbian are mentioned, which affected the design of my applications. To be able to further extend my written applications, basics of configuring and compiling a Symbian project are discussed.

3.2 Characteristics and Limitations of Symbian

3.2.1 Exception Handling and Stack Cleanup

Instead of C++ exceptions, which were not part of the C++ standard when Symbian OS was designed, the operating system uses a lightweight exception-handling mechanism, called a *leave*. Leaves may occur as a result of an error condition or abnormal event,

such as insufficient memory to complete a request. The leave propagates the error to a point in the calling code where it can be handled, called a TRAP harness, which is mostly provided by the UI framework. Therefore, the function transfers control directly to the statement following the TRAP macro under which it was invoked. This is carried out by setting the stack pointer to the context of the original TRAP harness and, hence, jumping to the desired program location.
However, because of the jump any local resources, such as memory allocated on the heap, will be orphaned. This potentially leads to memory or resource handle leaks, because the destructor of instanciated objects will not be called, and any resources they claim cannot be recovered.

This key aspect of Symbian OS exceptions has far-reaching implications. It forces a clear distinction between objects, which can be safely orphaned, and those which cannot. Such a differentiation can be seen in the naming convention. All types beginning with T can be freely allocated on the stack, because their data is contained internally. Pointers to objects are not used as members.
Objects beginning with a C cannot safely be orphaned. They must be accessible somewhere so they can be cleaned up, when an error occurs.

The cleanup stack is the Symbian OS mechanism for handling this problem. It is used to keep track of resources to which the only pointer is a local variable. In the event of a leave, the cleanup stack will destroy each of the resources placed upon it.

There are some important places in code which should never leave. These are C++ constructors and destructors. Therefore, Symbian OS classes typically use a two-phase construction to avoid leaves occurring in construction code.

When creating an object of class `CDemoClass` in C++ this would look like

```
CDemoClass* objPointer = new CDemoClass();
```

In Symbian this is not applicable, if during the construction an error can occur. The pointer `objPointer` would be lost and, hence, the instantiated object as well. As a result a two phase construction is used, which behaves as follows. The object is created by

```
CDemoClass* objPointer = CDemoClass::NewL();
```

The `L` at the end of `NewL()` shows that this function may leave. `NewL()` is implemented with the usage of the cleanup stack:

```
CDemoClass* CDemoClass::NewL()
{
    CDemoClass* self = new CDemoClass();
    CleanupStack::PushL(self);
    self->ContructL();
    CleanupStack::Pop(self);
}
```

In the first phase the object is created with the normal C++ operand `new`. However, the constuctor must not contain any code that can leave. Then the pointer is saved on the cleanup stack. Afterwards code with possible memory leaks, which implicate a leave, is execute in the method `ConstructL()`. If everything went right during construction, the pointer is deleted from the cleanup stack and the object has been created successfully.

3.2.2 Multi Tasking and Active Objects

In Symbian OS, every process can have one or more threads. The main thread of a process is created when the process is started.
Moreover Symbian OS implements preemptive multitasking, so that it can run multiple applications and servers simultaneously. It is optimized for thread switching. This means that every thread is allowed to execute for a limited period of time until the system scheduler passes execution to another thread.

Active objects are used to implement nonpreemptive multitasking within the context of a single thread. Under cooperative multitasking the running task decides when to give up the CPU. Active objects and the active scheduler provide a framework to program cooperative tasks. In both cases the scheduler is responsible for selecting the thread with the highest priority being ready to run next.

Note that Symbian OS is optimized for single-threaded applications, and the use of active objects is preferred instead of multi-threading. A general rule is to always use single-threading in Symbian OS programs, unless there is an absolute need for mulit-threading.

Some advantages of single-threading compared to multi-threading are:

- Using multiple threads increases memory consumption as each thread has its own stack allocated.
- A context switch between threads is slower than switching between active objects in the active scheduler loop.
- Some resources can only be used from the main thread. For example, dynamic arrays are not thread-safe.
- Because there is no need for mutual exclusion from concurrent access, the use of active objects increases performance and makes the application less error-prone.

Active objects can be thought of as subtasks for the thread with the fact that an active object that is executing is not interrupted (like thread execution is). The active object itself makes the decision when it has completed its task and returns control to the active scheduler. Then the active scheduler is able to select another active object under execution.

The user interface application, which is a basis for every Symbian GUI application, has a main thread with an active scheduler automatically installed. However, when writing servers or creating own threads, the active scheduler must be installed before the active

objects can be used. To make a class an active object it is sufficient to inherit from the class `CActive` and implement the abstract function `CActive::RunL()`. The active object can then make an asynchronous request for a service and process the request. This is done in `CActive::RunL()`, which is called by the active scheduler when the request has completed. The member `TRequestStatus iStatus` of `CActive` consequently informs about the status of the performed request.

A more detailed description of multitasking in Symbian OS and the concept of active objects can be found at [21, 24].

3.3 Development Environment

3.3.1 S60 SDK and Command Line Tools

To test and run developed applications a software development kit provided by Nokia is necessary. The SDK delivers all the tools, which are required to build C++ for Symbian OS projects. Furthermore the package contains a S60 device emulator, documentation and sample applications.

Emulator

With the emulator it is possible to test applications without a real device. This is convenient because the application does not have to be installed on the mobile device every time. Moreover the emulator allows debugging the program, which is possible on the phone only with additional commercial equipment.

A tough problem I faced, was running an application on the emulator with Bluetooth support. To the date of my development Nokia only supported PCI Bluetooth adapters, which must be mapped to a COM-port of the PC. The emulator can then be configured to use this COM-port as IO interface. The widespread USB Bluetooth dongles were not officially supported. However, after some research I found a method to get Bluetooth running on the emulator with a USB dongle.
Nokia provides an additional tool named *Nokia Connectivity Framework* "to visualize, construct, maintain, and modify emulation or testing environments that utilize Nokia SDK emulators" [20]. After same configuration the framework installs a special driver for the Bluetooth device and maps it to the emulator. To get the whole setup running, at first the configuration of Bluetooth dongle and emulator in the connectivity framework must be started. If the emulator and only the emulator is stopped, the application can be executed and debugged with a working Bluetooth connection out of the IDE. A disadvantage of this method is that the Bluetooth dongle is exclusively used by the framework and not useable for other programs anymore. To install the developed application on the mobile device via a Bluetooth connection, the framework must be shut down and the original device driver installed beforehand.

Command Line Tools for Building a Project

All important settings for a Symbian C++ project are stated in the *component definition file,* always called `blf.inf`. In this file all sub-projects with their project specification files are listed. For small projects this may be only one file, as in my case.

The single *project specification file* identifiable at the ending `*.mmp`, contains all the information for building the project. For example it includes information about target type, sources, include files and libraries the project should be linked against. The project specification is generic, so that it allows building the project for different target platforms.

To build the project from the command line makefiles for the different target platforms have to be generated at first. This is done by the command

```
bldmake bldfiles
```

After this step a file named `abld.bat` is generated. This batch file offers the different variants for building the project. For example

```
abld build winscw udeb
```

will generate the project for the emulator, while

```
abld build armi urel
```

will generate build specific files. `winscw` and `armi` specify the different target platforms, while `udeb` and `urel` state, if the build tools shall generate a debug or release version of the project.

To be able to install the project on the mobile phone a device package has to be created. The rules for storing the application on the mobile phone must be stated in a `*.pkg` file. Finally calling

```
makesis *.pkg
```

generates a `*.sis` file, which is a device readable file format and allows the installation of the application..

Current ARM processors, as used in the Nokia 6260, have two different instruction sets. ARM4 with 32 bit length and THUMB with 16 bit length. ARM4 has a slightly richer instruction set and a compiled program will be faster compared to THUMB. On the other hand THUMB mode is more compact, thus has a smaller size. However this is not true for all mobile phones. Devices with a 16 bit memory bus will use the THUMB mode. A program compiled for THUMB will then be smaller and run faster.
ARMI is the solution for this uncertainty what to use. This is the fastest choice to use since a ARMI program can be linked with THUMB and ARM4 libraries, while the others cannot.

Figure 3.3 illustrates the interaction of the described tools.

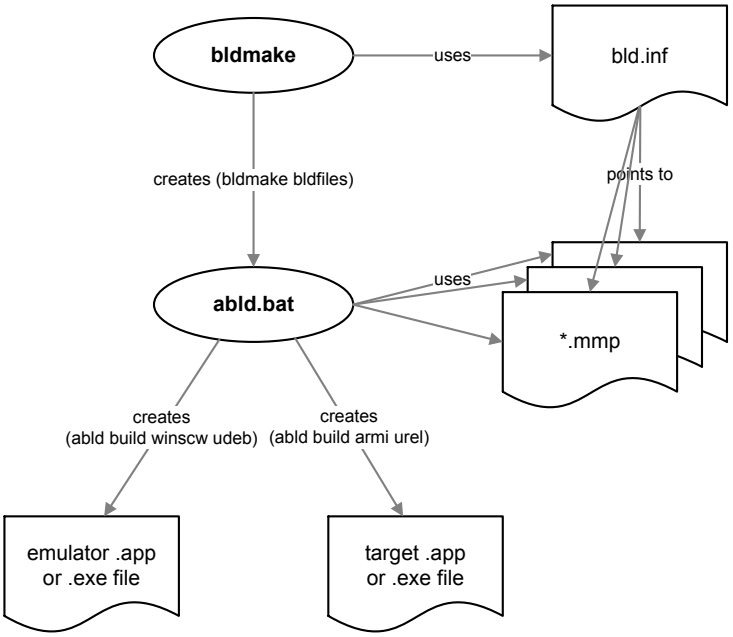


Figure 3.3: Interaction of build tools

3.3.2 Carbide.C++ IDE

Carbide is a new generation of mobile development tools from Nokia. The tools focus on tree primary development areas.

Crabide.j is the development environment for the Java Platform, Micro Edition.

Carbide.vs is a plug-in for Microsoft Visual Studio that allows to develop C++ code for Symbian OS platforms.

Carbide.C++ is the recommended choice for C++ development according to Nokia. This tool as based on the open source project Eclipse and is provided int three different editions, which are *Professional*, *Developer Edition* and *Express.*

I chose Carbide.C++ in version 1.0 for development, as it is the only edition, which is available for free. A big disadvantage of all IDEs was, that there has been no GUI designer included. This resulted in a lot of effort in writing the GUI for my applications, as I had to code it myself. However, at the end of my Bachelor's Thesis I noticed that Nokia released an UI designer for Carbide.C++, which is available in editions *Developer Edition* and *Professional*. I would suggest to use this tool for furthher development, as building the GUI "by hand" is a lot of work and slows development heavily.

Carbide.C++ offers a convenient project wizard to get started with the development of applications. Thereby a program stub is provided to meet the restrictions of a Symbian application. Moreover it is possible to import projects via the previous described component definition file (`blf.inf`) and the project specification file (`*.mmp`). Project specific settings like includes and libraries can then easily be handled with the IDE.

Note that the appropriate SDK must be installed to be able to build applications for the emulator or the target device

3.4 Application Programming Interfaces

The following mentioned APIs have been used for this project. Therefore, their functional range is described in more detail.

3.4.1 Bluetooth

Like many other communications technologies, Bluetooth is composed of a hierarchy of components, referred to as a stack. This stack is shown in figure 3.4.

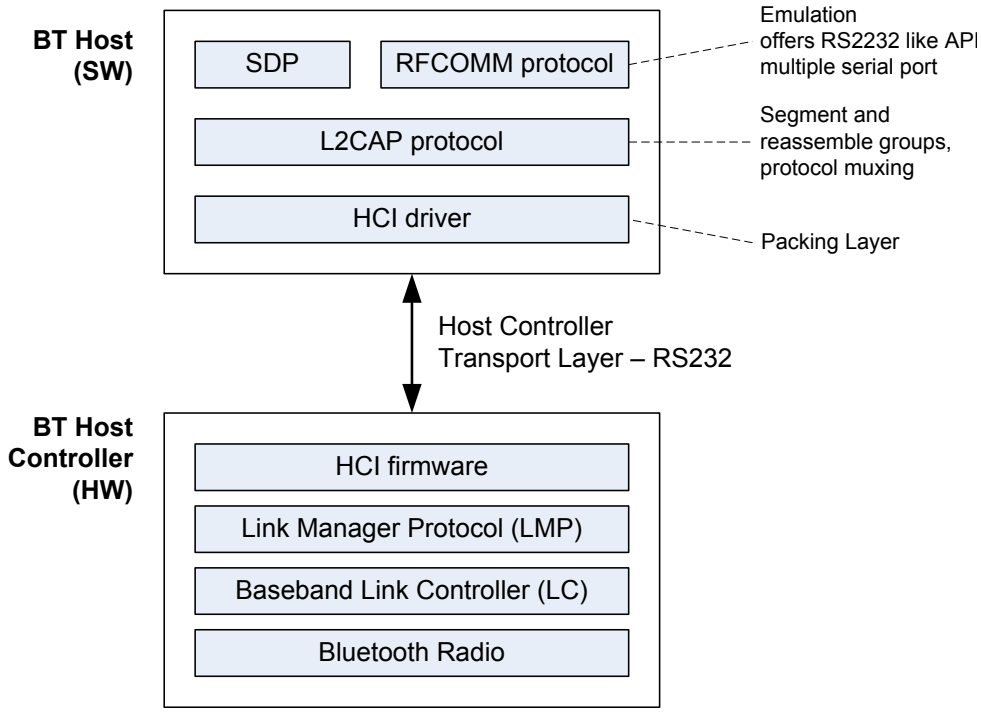


Figure 3.4: The Bluetooth stack (source: Symbian 7.0 developer documentation [23])

The Bluetooth Host Controller components provide the lower-level of the stack, which are typically implemented in hardware, and to which applications do not have direct access.

The Bluetooth Host components allow applications to send or receive data over a Bluetooth link, or to configure the link. RFCOMM allows an application to treat a Bluetooth link in a similar way as if it were communicating over a serial port. I used this method for communicating with the Eyebot controller. The Logical Link Control And Adaptation Protocol (L2CAP) allows a more detailed control of the link. It controls how multiple

users are multiplexed, handles packet segmentation and transmits quality of service information. The Service Discovery Protocol (SDP) is used for locating and describing services available through a Bluetooth device. The Host Controller Interface (HCI) driver combines the higher level components to communicate with the hardware.

Symbian provides a number of APIs for these different layers.

- There are Bluetooth sockets, which encapsulate access to L2CAP and RFCOMM through a socket interface.
- For a service discovery two APIs are provided. First a Bluetooth Service Discovery Database with which an application is able to advertise its services and second a Bluetooth Service Discovery Agent, which allows to discover the services, which are available on a remote device.
- These two APIs are enclosed in a relatively simple Bluetooth UI. It provides a dialog by which a remote device can be chosen and a link can be established.
- A Bluetooth Security Manager enables services to set appropriate security requirements that incoming connections must fulfill. In my developed application there are no security directives, hence, I have not used this API.

More information about the Bluetooth API can be found in the documentation enclosed in the SDK [7]. For a illustration of the usage of the different APIs in the program code the online documentation of my developed application is a good reference.

3.4.2 Mobile Phone Camera

The camera API is an open, generic API for controlling simple on-board digital camera devices. The API offers three main functions, which can be used by an application. These are methods to control camera settings as flash mode, contrast and brightness of the image, as well as a digital zoom level. Functions for capturing still images, which involves transferring the image from the camera to the application, where the image format and size are adjustable. Finally it includes the possibility to capture videos, which requires the use of a frame buffer.

The API provides a view finder, with whose help it is possible to transfer view finder data to a specific portion of the screen using direct screen access. Hence, the frames from the camera are directly transferred to the display memory. This is a lot faster than capturing the image, saving and then displaying it on the screen. I used a slightly modified functionality of the view finder in my lane detection application, where the view finder data is transferred as a bitmap.

As before more information is provided in my online documentation of the application or in the SDK documentation [7].

Chapter 4

Bluetooth Communication Software

The aim of this section is to describe the requirements, the design and the implementation of the developed application enabling a remote control of the controller via a Bluetooth connection.

The Eyebot controller has a LCD, which informs the user about the current controller state. Figure 4.1 illustrates an image of this display.

Figure 4.1: Eyebot controller LCD

Below the LCD there are four white buttons, which are used for user input. The overlaid functions for the buttons can be seen from a menu in the last row of the displayed data.

The goal for the application on the mobile phone is to reproduce this appearance on its own screen. Moreover, control has to be possible with the phone buttons. As a result, a bi-directional communication from and to the controller has to be established. Furthermore, the application has to provide an opportunity to choose which remote device to connect to.

4.1 Design

In the design phase I have considered three important parts, which are connection dependent on the one side and affect the application design on the other side. They are discussed in more detail in the follwing.

4.1.1 Transmission Protocol and Commands

The operating system on the controller already defines a transmission protocol to use. Moreover, there are several commands available for accessing the display on the controller. These commands are also used for a remote control and transmitted via the Bluetooth link.

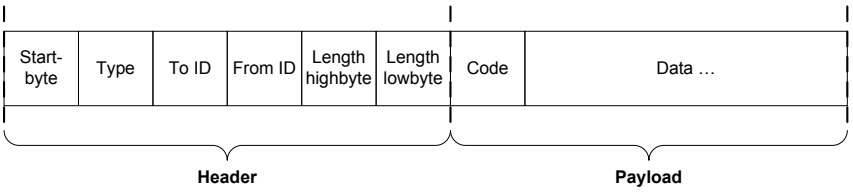


Figure 4.2: Transmission Protocol

Figure 4.2 illustrates the used protocol. The individual parts and their meaning are explained in the following.

Start byte The start byte is always of value 0xC6 and signalizes the beginning of an new datagram.

Type The Type byte specifies the type of the current datagram. Possible values are 1, 2 and 3, which stand for OS, User and Synchronization. In my application only type OS is used.

To ID This determines the ID of the receiver of the message. The mobile phone or a PC always has an ID value of 0. The device ID at the Eyebot controller can be set manually. The ID is important if more than one controller is sending data, because thus it is possible to differentiate which message belongs to whom.

From ID It indicates who has sent the message. Possible values are the same as for To ID.

Length highbyte & lowbyte The length of the sent payload is specified by these two values. As a result the maximum length for data to transmit is $2^{16} - 2$, because one byte is used to state the command code.

Code is the first byte of the payload data and determines which action to perform on the device. Possible values are listed in Table 4.1. These commands have to be interpreted on the mobile phone.

Code	*Parameter*	*Function*
`Clear`	-	Clears the display.
`Line`	`x1, y1, x2, y2, color`	Draws a line from $(x1, y1)$ to $(x2, y2)$ in color `color`
`Area`	`x1, y1, x2, y2, color`	Draws an are with $(x1, y1)$ as top left corner and $(x2, y2)$ as bottom right corner in color `color`
`SetPixel`	`x, y, color`	Sets the pixel at position (x, y) to color `color`
`InvertPixel`	-	Is not implemented - use SetPixel instead.
`Menu`	`string`	Sets the command menu of the controller. 4 character are used for each of the four menu entries.
`PutChar`	`char`	Writes the character char at the current position to the display.
`PutString`	`string`	Writes the null terminated string `sting` at the current position to the display.
`SetPos`	`row, col`	Sets the current position to (row, col).
`SetString`	`row, col, string`	Same as PutString, but the position is updated to (row, col) before.
`KeyPressed`	`keyCode`	Informs about an occurred keystroke.
`CodeImage`	`code, depth, data`	Transmits an image. Data contains the RGB values.
`VWPosition`	`position`	Transmits the current position of the robot on the playground.

Table 4.1: EyeBot commands

Data The data field in the protocol contains the parameters for the specifc command code.

For compatibility with a newer version of the Eyebot controller I have introduced five more commands, which are displayed in Table 4.2. They allow a flexible adaption of the display size. Because the size of the display can be greater than 256 pixel in one direction, it was required to split some parameters into high and low bytes. Moreover, there is an additional variant for functions containing pixel positions as parameters, which allows values greater the size of one byte.

In my developed application messages are received via a Bluetooth link. If the header is correct, the command is extracted from the payload and appropriate executed.
Before the structure of the developed application is discussed in more detail, I want to give an overview of the user interface, which was planned to implement.

Code	*Parameter*	*Function*
`ResGraphics`	`widthH, widthL, heightH, heightL`	Informs about resolution of the controller display. The size is in pixel. Both, for width and height are two parameters necessary, as the size can be greater than 256 pixel.
`ResText`	`rows, cols`	Informs about the resolution of the controller display for displaying text.
`SetPixelL`	`x1H, x1L, y1H, y1L, color`	New function for setting pixels with 16bit position values.
`LineL`	`x1H, x1L, y1H, y1L x2H, x2L, y2H, y2L, color`	New function for drawing a line with 16bit start and end point values.
`AreaL`	`x1H, x1L, y1H, y1L x2H, x2L, y2H, y2L, color`	New function for drawing an filled area, again with 16 bit position values for the top left and bottom right corners.

Table 4.2: Additionally introduced commands

4.1.2 User Interface

I decided to develop an application with different views. One main view, which is displayed after launching the program and serves as overview for the other views created in further steps. Additionally, there can be several so-called remote views, where each of them displays the output from one remote Eyebot controller. As a result, the user interface was designed to support several simultaneous connections to different controllers. Each of them with its personal view and the possibility to change quickly between the different views.

Figure 4.3 illustrates the possible navigation through my application. After start-up the main view is shown, but no connection is established so far. To initiate a connection to a remote device, the 'Options' menu offers an appropriate entry. After selecting to create a new remote session, a device inquiry is started. The result of the inquiry is displayed as soon as there are devices discovered. If there is no controller in range, a message is displayed, which allows to retry the inquiry or to navigate back to the main view. Note that such erroneous situations are not shown in the illustration to keep track of the correct workflow.

Having chosen the desired remote controller, a connection is established, which is signalled with an acoustic tone on success. The phone screen is updated, if the display on the remote controller changes and, hence, commands via the Bluetooth link are received. Moreover, the controller can be operated by the number buttons 1 to 4 on the mobile phone. This is illustrated via four white boxes in the lower part of the screen. The corresponding assignment of functions to these buttons is shown in the menu row of the simulated controller display.

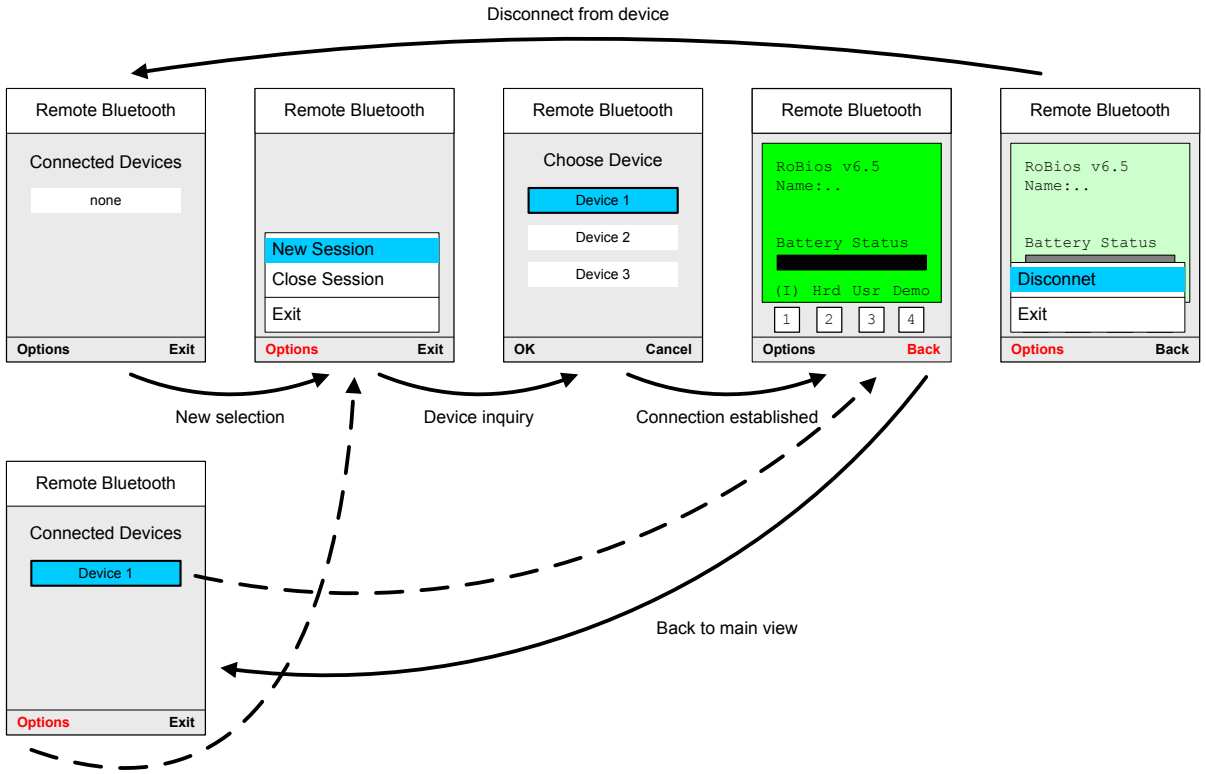


Figure 4.3: Mock-up of the designed user interface with possible view transitions

By selecting to 'Disconnect' from the remote device, the session is closed and the application returns back to the main view. Otherwise it is possible to navigate back to the main view without canceling the connection via the function 'Back'. The main view changes then a little bit and displays in a list the currently open sessions, which is shown in the lower image part of figure 4.3. Thus, it would be possible to open a further connection to another controller. By choosing one of the currently available session out of the list, the according remote view is displayed. Of course the display is also updated if not in focus and therefore is always up to date.

4.1.3 Class Dependencies

To be able to implement the designed user interface structure, I developed a sophisticated class structure, which is displayed in figure 4.4. I stuck to the convention of Symbian OS to cleanly separate user interface and application engine. Furthermore I designed the connection specific part of the application in a generic way, so that it is possible to add additional types of connections beside Bluetooth. These borders in the class design are illustrated in colors green, blue and orange.

Back to the user interface part, the Application, the Document and the Application UI classes are predetermined by Symbian, as mentioned in 3.1. In my application there

exists one MainView and none, one or more RemoteViews, depending on the state of the application. The corresponding Containers to these views manage the belonging data and serve to update the views corresponding to changes caused by the application engine. Therefore, these containers own a dependence to the data representing classes RemoteSessionList and Display of the engine.

The engine itself mainly consists of several RemoteSessions, which are combined together in the RemoteSessionList. This class keeps track of the currently established sessions to remote controllers. A RemoteSession is inherited from CActive, the Symbian provided class to create an active object. This is necessary, as the RemoteSession has to perform several asynchronous commands for establishing a connection. The function `RunL()`, which is called from the active scheduler on changes of the request status implements a state machine. This finite state machine is discussed in more detail in the next section .

The RemoteSession owns beside the already mentioned RemoteView some more classes. These are the Parser, which parses incoming messages an extracts appropriate commands. The class Message is a container for storing a received message and for sending messages according to the protocol introduced in section 4.1.1. In addition, RemoteSession contains a class SoundPlayer, which is used to play-back sounds for specific events.

The lowest part of the class chart shows classes implementing all functions concerning the connection. I paid attention to establish a border between the connection dependent parts and the rest of the engine. Thus there exists an abstract class ConnectionEngine. The blue marked part of the engine, therefore, implements functions defined in this generic class. As a matter of fact this offers the opportunity to create *any* connection dependend class by inheriting from ConnectionEngine and implementing the virtual methods. As an example it would be possible to extend my application by the functionality of an serial communication with minimum effort.

However, my aim was to carry a Bluetooth connection out. Thus BTConnection implements the methods for connecting, disconnecting, sending and receiving. The BTDiscoverer is additional required to perform a device inquiry for Bluetooth devices in range. Both classes make use of BTDeviceParameter, which stores connection depend data for the current established link.

A more detailed description of the mentioned classes is given in appendix B.1. However, implementation specific details can be found in my online documentation located on the enclosed CD.

4.1.4 Finite State Machine for Connection Management

As connection management is a complex process, I decided to use an finite state machine. All possible states together with their transitions are shown in Figure 4.5. The states also show the specific functions that are called.

After the creating of a new session, a device inquiry is performed. If devices have been

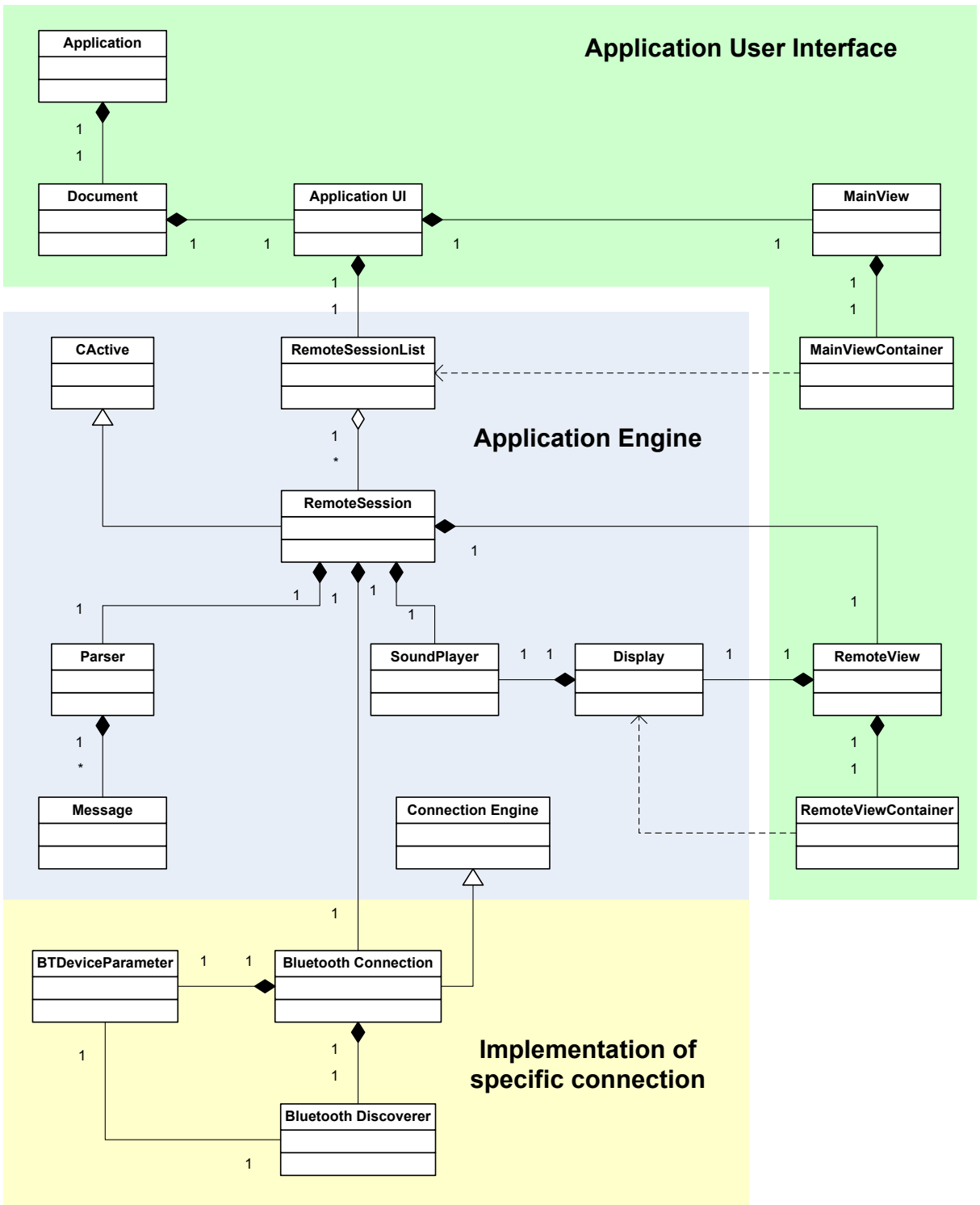


Figure 4.4: Class chart of the application with partition in UI and engine

found, one of them can be chosen and a connection will be established. If an erroneous device has been selected or it does not provide the required service the control navigates back to the device selection. Note that the service check is considered in the state diagram, however, it is not implemented for the Bluetooth connection, as all Eyebot controllers offer the required RFCOMM interface.

If the connection has been established successfully and a corresponding view and parser for the remote session has been instantiated, the application starts to receive and parse incoming data. It remains in that state until the connection is closed or the session is terminated.

Sending back data, like keystrokes for controlling the Eyebot controller is not included in the state machine. This is realized via an event handling mechanism. An appropriate function is directly called from the framework when a keystroke emerges. The state machine cycle is shortly interrupted and the data can be sent over the link.

The concept of active objects offers for the purpose of state control a convenient solution. The active object owns one member variable, which keeps track of the current status of the object. This variable can be used to implement state transitions. For example in state 'Select device' the function SelectDeviceL() is called with this status variable as parameter. Once the asynchronous request completes, an error code is stored in the status variable. The active object automatically recognizes the change of its status member and calls the function RunL(). The only thing to implement in RunL() is a check for the status variable. If there was no error, a transition to the next state 'Check service' can be performed, otherwise, the execution terminates and for this example the remote session is closed.

Asynchronous framework functions automatically offer the functionality to report their status via a status variable. Own functions can easily implement a similar behavior in saving an error code in the status variable explicitly. The call to SetActive() in the example source code is necessary to activate the active object to react on status changes of its status member.

4.2 Output

The images in Figure 4.6 have been taken from the developed application. The screenshots illustrate the different views in combination with possible user interaction.

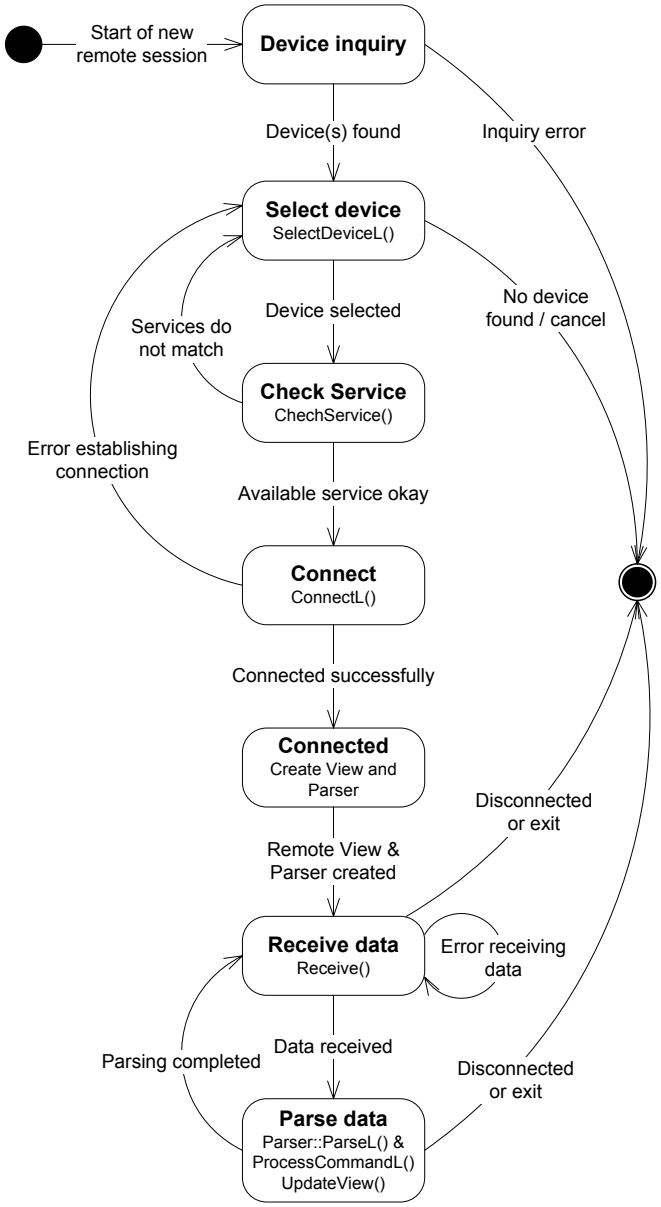


Figure 4.5: Finite state machine, which is implemented in the active object RemoteSession and performs the connection management

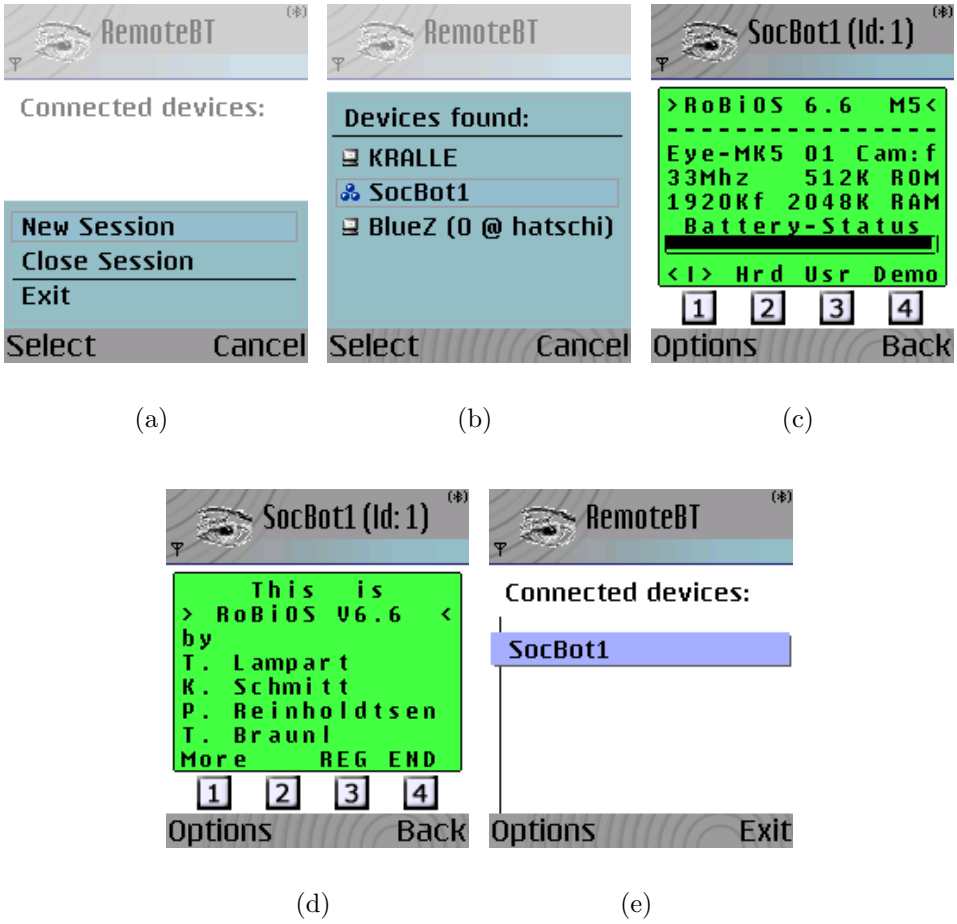


Figure 4.6: Screenshots from the application on the mobile phone.
(a) Main view with options menu to create a new session.
(b) Found Bluetooth devices after inquiry.
(c) and (d) EyeBot display emulation on the mobile phone's screen.
(e) Main view showing that one remote session to 'SocBot1' is currently active.

Chapter 5

Lane Marking Extraction

This and the following chapter concentrate on the developed algorithm for lane marking detection considering the limited capabilities of the mobile phone.

Figure 5.1 shows an ordinary road scene on a highway. The lane boundaries are marked with solid white lines, while the two lanes are separated by a dashed line. Lane markings like these will be extracted and further on detected in described process.

A first step to extract features of a video frame is to have a look at points in the image where brightness changes particularly sharply. This edge points can be associated with boundaries of objects and other kinds of meaningful changes. Therefore, it will be useful to perform a filtering, which exactly identifies these intensity changes.

Figure 5.1: Captured video frame

At first the problem of noise in images in combination with edge detection is addressed. Note that a short mathematical background behind edge detection is presented in Appendix A.1. Afterwards two kinds of edge filters are discussed. First the elementary Sobel

filter and second a more sophisticated approach with steerable filters. Modifications for the use in my implementation are as well treated as a performance comparison between the two different peculiarities of edge filters. In the end the final results of the edge filtering step are presented, which are used for detecting and parameterizing the lines.

5.1 Edge Detection and Noise

A primary problem in edge detection is image noise. Noise in the image for example results from temperature noise of the camera or arises from the lens. The disadvantage of simple edge detectors like the *Prewitt* filter is, that they respond strongly to sharp changes. Moreover an edge detection filter can be seen as a high-pass filter, emphasizing high spatial frequencies and suppressing low ones. As noise is said to be uncorrelated and therefore possesses uniform energy at each frequency, it makes finite difference estimates of image derivatives unusable. Figure 5.2 (a) illustrates an image line containing noise. It can be seen that a maximum evaluation in the first derivative will not lead to sufficient results.

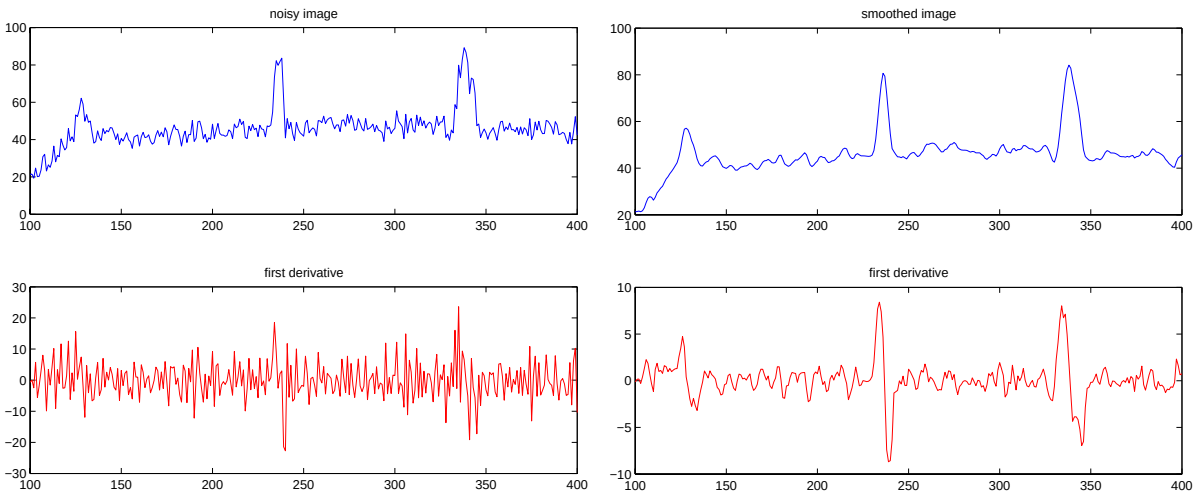


Figure 5.2: (a) A noisy image signal an its derivative; (b) the smoothed image signal and its derivative

By first smoothing the image and then calculating the derivative needed for edge detection suppresses noise and leads to a better result, as shown in (b). An appropriate way to smooth an image is to use a Gaussian filter mask. Since the Fourier transform of a Gaussian is another Gaussian in the frequency domain, applying a Gaussian blur has the effect of low pass filtering the image and hence reducing noise in the image. Moreover a Gaussian operator implicates a number of convenient properties. Some of them will be useful for two dimensional filtering and are explained in the following.

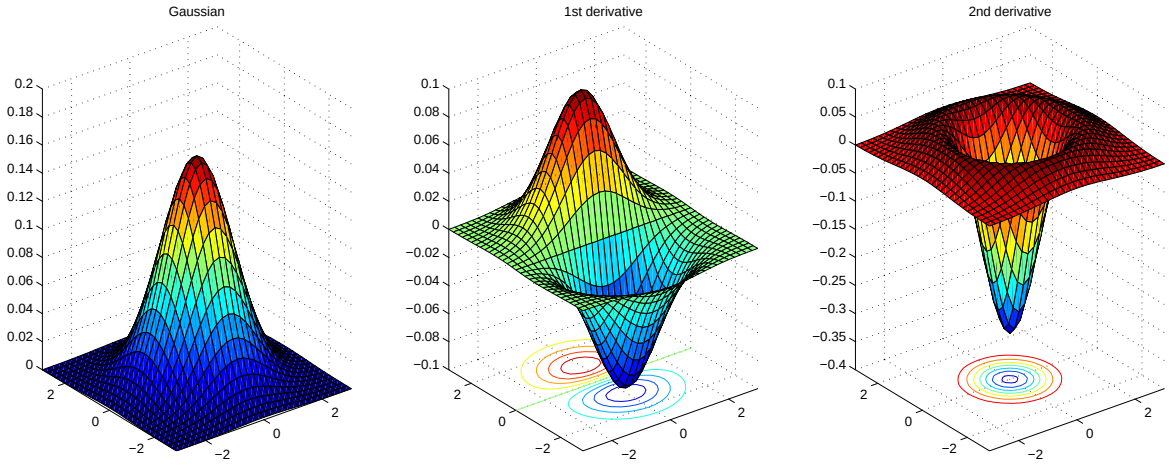


Figure 5.3: A Gaussian and its first and second derivatives. The standard deviation σ was set 1.

Smoothing an image is mathematically expressed by a convolution of the original image I with the Gaussian mask G. Calculating derivatives of the blurred image is a convolution with the filter mask D. The operation can be written as $I_{edge\,detected} = I * G * D$. Both these operations are linear, thus enabling to combine them in a more efficient way to

$$I_{edge\,detected} = I * (G * D)\,.$$

This equation looks quite the same as before, but the main point behind is to calculate at first the derivative of the Gaussian, which can be done beforehand, and then just apply this filter mask on the image. Whit this approach just one filtering instead of two is necessary. Figure 5.3 displays a Gaussian and its first and second derivatives like they would be applied to an image. (Note that the first derivative is directionally sensitive, while the Laplacian of Gaussian filter kernel is not).

A filter mask with $n \times n$ entries would require n^2 multiplications and $n^2 - 1$ additions per pixel, leading to a high computational effort. However the two dimensional Gaussian

$$G(x, y, \sigma) = \frac{1}{2\pi \cdot \sigma^2} \cdot \exp\left(-\frac{x^2 + y^2}{2 \cdot \sigma^2}\right) \tag{5.1}$$

can be rewritten as

$$G(x, y, \sigma) = \frac{1}{\sqrt{2\pi} \cdot \sigma} \cdot \exp\left(-\frac{x^2}{2 \cdot \sigma^2}\right) \times \frac{1}{\sqrt{2\pi} \cdot \sigma} \cdot \exp\left(-\frac{y^2}{2 \cdot \sigma^2}\right), \tag{5.2}$$

which is a separation in terms of the image coordinates. This means that the two-dimensional filter mask can mathematically also be written as dyadic product of two vectors of length n. As a result the filtering is separable in two one-dimensional kernels in x and y direction. Note that the first or second derivative does not change the separability of the filter mask. Therefore, only n multiplications and $n-1$ additions per direction plus an operation for combining both are needed. The computational effort thereby is reduced from n^2 to the order of n.

5.2 First Approach: Sobel Filter

In my approach I tried several filter types to find the one fitting best to my demand in terms of quality but also in low computational effort. The easiest one to try is of course the Sobel filter. Its filter mask for vertical edges is

-1	0	1
-2	0	2
-1	0	1

which is very similar to the elementary Prewitt kernel (stated in Appendix A.1), but emphasizes the direct neighboring pixels. This can be seen as a very rough approximation of an Gaussian filter, but indeed performs some kind of smoothing. The kernel for horizontal edges is as well gained by rotation the mask by 90°. Note that a coefficient of 1/8 is needed to get the right gradient value. However, this does not make a difference for edge detection and therefore integer values are used.

Keeping in mind that the edge strength is given by the gradient magnitude and the edge orienation is in direction of the most rapid change in intensity, the whole edge filtered image is calculated by the norm of the two individual parts for x and y orientation

$$G = \sqrt{G_x + G_y}. \tag{5.3}$$

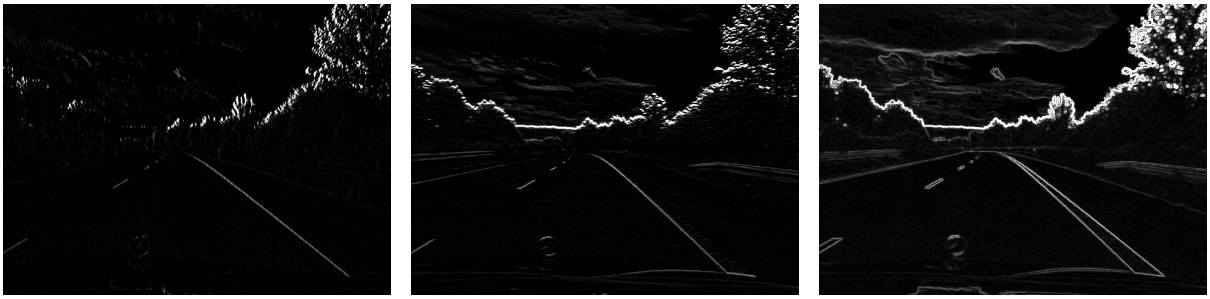
The direction of the steepest increase of intensity is as then defined by

$$\Theta = \arctan\left(\frac{Gy}{Gx}\right).$$

The application of the Sobel filter to the example image, leads to the result shown in Figure 5.4. It can be seen that the lane edges are detected quite well, but there is a very thick edge resulting from the boundary of trees. As the interesting part with lane markings lies only in the lower field of the video frame, the images are appropriately cropped by discarding the upper half of the video frame. Of course this is done before filtering the image.

5.2.1 Additional Requirements

As seen in Figure 5.4 (c), there are two edges detected for every lane marking. However, in the end just one line should be detected per lane marking. To avoid an expensive clustering of left and right edges belonging together to one line, it would be convenient to detect just the inner side of a lane marking. For changes from low to high level sections in the image the result is a maximum positive value, but for changes the other way around from high to low level the outcome is a maximum negative value (see Figure A.1 in Appendix A.1). Therefore, it is possible to detect the left side of a lane marking by taking the maximum positive values as an detected edge. For determining the right side of a lane marking it would be sufficient to consider just the maximum negative values



(a) (b) (c)

Figure 5.4: Example video frame filtered with Sobel filter for (a) horizontal gradient and (b) vertical gradient; (c) displays the combination of both according to equation (5.3)

of the first derivative. The same effect is obtained by multiplying the filter mask with -1. This results in an inverted first derivative, where the maximum values describe the wanted right side of a lane marking now. The same procedure holds true for detecting upper and lower borders of a horizontal lane marking.
As a result the image was split up in different parts and has been filtered with the masks shown in figure 5.5.

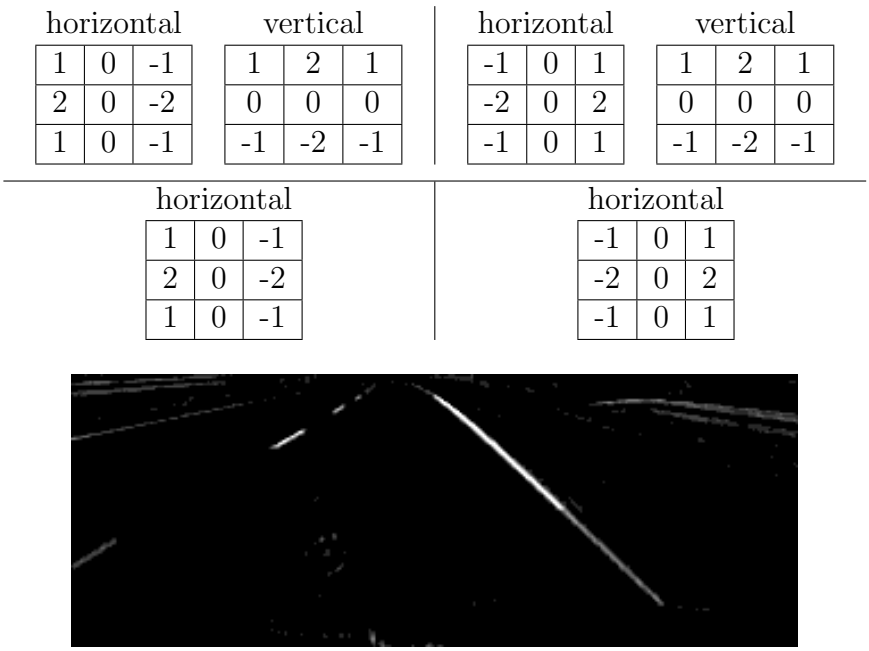


Figure 5.5: Sobel filter masks applied to the four different defined image areas to be able to detect just the inner side of lane markings. The filtering result can be seen in the image.

Especially in my situation when detecting lines, it is not necessary and moreover not wanted to detect all edges in the image, but only the one in direction of possible lane markings. As previously displayed in Figure 2.3 lane markings run dependent on the

image area with a certain degree due to the perspective distortion of the image. Hence it would only be necessary to detect edges in direction of the shown red lines, which radially turn around an expected vanishing point.

Therefore the lower part of the image was just filtered with a vertical sensitive filter mask to omit horizontal edges. This is sufficient, because the filter mask is also sensitive to diagonal edges. In the upper part both horizontal and vertical filter masks were used as line markings tend to show every possible direction. A more detailed partitioning was tried as well, but did not generate better results.

To already consider this specific characteristic of the lane markings in the filtering step, I decided to build and test an edge filter which is direction sensitive. Of course a Sobel filter has a sensitivity to a certain direction as well. However, the goal was to build a filter, which is sensitive to every arbitrary orientation in the image, but without a higher computational effort. The approach to this aim is explained in the following section.

5.3 Second Approach: Steerable Filter

One approach to finding the response of a filter at many orientations is to apply many versions of the same filter, each different from the other by some small rotation angle. A more efficient approach is to apply a few filters corresponding to a few angles and interpolate between the responses. One then needs to know how many filters are required and how to properly interpolate between the responses. With the correct filter set and the correct interpolation rule it is possible to determine the response of a filter of arbitrary orientation without explicitly applying the filter. This is the main concept and intention behind steerable filters, as stated in [10].

At Computer Vision and Robotics Research Laboratory, University of California, San Diego steerable filters have already been successfully used for detecting lane markings. In [16, 17, 18] they argue that steerable filters have a number of desirable properties and therefore are excellent for a lane detection application.

First, each filter can be created to be separable which speeds up processing. By separating the filter into an x and y component, the two-dimensional convolution of the filter with an image splits up into two one-dimensional convolutions. Second, only a finite number of rotation angles of a specific steerable filter are needed to form a basis set for all possible angles, as mentioned before. The number of basis filters depends on the degree of the Gaussian derivative used for smoothing. This will enable me to tune the filter to a continuous range of possible orientations and calculate the response of the filtering for different parts of the image, but with applying the set of basis filters just once to the image. Moreover, steerable filters are robust against poor lighting conditions as well as they appear to be capable for complex shadows, where many other edge detection approaches fail.

Beside the good edge detection results, these properties of steerable filters are also very

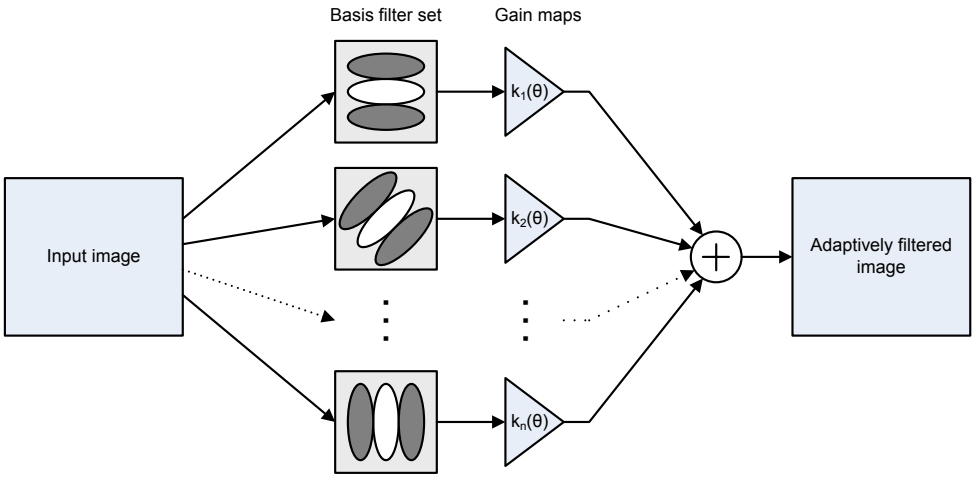


Figure 5.6: Steerable filter architecture

important for my implementation, as processing power is limited on the target device and I have to keep computational effort low.

Figure 5.6 shows a general architecture for using steerable filters. The front end consists of a bank of permanent, dedicated basis filters, which always convolute the image as it comes in. The outputs are multiplied by the appropriate interpolation functions individually for each position and time. The finishing summation builds the adaptively filtered image.

5.3.1 Based on Second Derivative

Publications [16, 17, 18] followed the approach to use a second derivative of a Gaussian as initial input and further on built a steerable set of basis filters. So I decided to start as well with this attempt and compare the results with the one obtained via the Sobel filter.

With the theory given in A.2 it is easy to derive the required basis functions, as well as their interpolation coefficients. However, for the further understanding it is not required to know exactly how the filters are derived. In short, every function that can be expressed as a Fourier series is steerable.

The interpolation functions $k_j(\theta)$ are independent from the Fourier coefficients $a_n(r)$. Therefore, the leading term $\frac{1}{2\pi \cdot \sigma^2}$ in the definition of a Gaussian like in Equation (5.1) can be left out as it just scales the function, but does not change anything in its radial and angular characteristics. Hence, for deriving a steerable basis set based on the frequency response of the second derivative of a Gaussian the function

$$G(x, y) = e^{-\frac{x^2+y^2}{\sigma^2}}$$

is sufficient. Setting the variance σ^2 to 1, which does not affect the further result, the

second derivative, named G_2 becomes

$$G_2(x,y) = \frac{\partial^2}{\partial x^2} G(x,y,\sigma^2=1) = 2\left(2x^2-1\right)\cdot e^{-\left(x^2+y^2\right)}. \tag{5.4}$$

Substituting the Cartesian coordinates x and y with the corresponding polar coordinates r and ϕ, (5.4) can be written as

$$G(r,\phi) = 2\left(2r^2\cos^2\phi - 1\right)\cdot e^{-r^2}.$$

With the substitution $\cos\phi = \frac{e^{\imath\phi}+e^{-\imath\phi}}{2}$ this can be stated as

$$\begin{aligned} G(r,\phi) &= \left(r^2\left(e^{\imath 2\phi} + e^{\imath(\phi-\phi)} + e^{-\imath 2\phi}\right) - 2\right)\cdot e^{-r^2} \\ &= \underbrace{r^2 e^{-r^2}}_{a_{-2}(r)}\cdot e^{-\imath 2\phi} + \underbrace{\left(r^2-2\right)e^{-r^2}}_{a_0(r)}\cdot e^{\imath 0} + \underbrace{r^2 e^{-r^2}}_{a_2(r)}\cdot e^{-\imath 2\phi}. \end{aligned}$$

Because only the coefficients for $n=[-2,0,2]$ are nonzero, a minimum of three basis functions is needed to steer the second Gaussian derivative (see A.2 for further explination). As a result the following equation can be stated

$$\begin{pmatrix} 1 \\ e^{\imath 2\theta} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ e^{\imath 2\theta_1} & e^{\imath 2\theta_2} & e^{\imath 2\theta_3} \end{pmatrix} \cdot \begin{pmatrix} k_1(\theta) \\ k_2(\theta) \\ k_3(\theta) \end{pmatrix}. \tag{5.5}$$

Taking both the real and imaginary part of (5.5) into consideration and requiring that they equal, gives a system of three linear independent equations, which can be easily solved for $k_j(\theta)$

$$\begin{pmatrix} 1 \\ \cos(2\theta) \\ \sin(2\theta) \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ \cos(2\theta_1) & \cos(2\theta_2) & \cos(2\theta_2) \\ \sin(2\theta_1) & \sin(2\theta_2) & \sin(2\theta_3) \end{pmatrix} \cdot \begin{pmatrix} k_1(\theta) \\ k_2(\theta) \\ k_3(\theta) \end{pmatrix}. \tag{5.6}$$

The final solution for the steerable filter at an arbitrary angle θ of the second derivative of a Gaussian is

$$G_2^{\theta} = k_1(\theta)\cdot G_2^{\theta_1} + k_2(\theta)\cdot G_2^{\theta_2} + k_3(\theta)\cdot G_2^{\theta_3}.$$

θ_j has to be chosen in the range between 0 and π. With equally distributed values in the domain this becomes $\theta_1 = 0°$, $\theta_2 = 60°$, $\theta_3 = 120°$. According to (5.6) the interpolation functions become

$$k_j(\theta) = \frac{1}{3}\left[1 + 2\cos\left(2\left(\theta-\theta_j\right)\right)\right].$$

Figure 5.7 shows the obtained three basis filters.

With this approach the obtained basis filters are not separable. However, to reduce computational effort in the leading filtering step it is preferable to design the three basis filters to be separable in x and y direction. For all functions that can be written as a

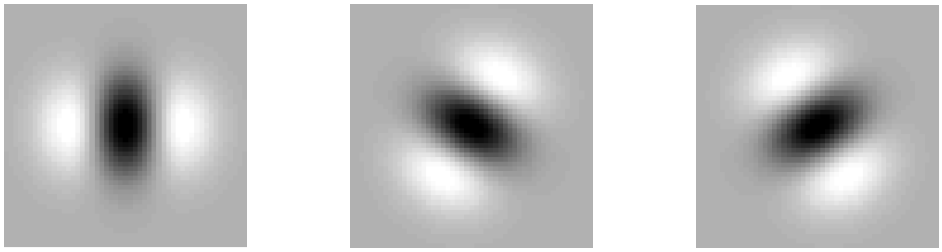


Figure 5.7: Basis filter set for $\theta_1 = 0°$, $\theta_2 = 60°$, $\theta_3 = 120°$

polynomial in x ans y, there exists and x-y separable basis. The initial second derivative of a Gaussian in Equation (5.4) fulfills this condition. With Equation (5.2) it has already been shown that a Gaussian is separable, so the term $2\left(2x^2 - 1\right)$ remains for further investigation. In [10] it is demonstrated how to split this up to a notation like

$$G_2^{\theta} = G(r) \cdot \sum_{j=1}^{3} k_j(\theta) \cdot R_j(x) \cdot S_j(y).$$

For the interpolation functions following

$$k_j(\theta) = (-1)^j \begin{pmatrix} N \\ j \end{pmatrix} \cos^{(N-j)}(\theta) \sin^j(\theta)$$

the results

$$\begin{aligned} k_1(\theta) &= \cos^2(\theta) \\ k_2(\theta) &= -2 \cdot \cos(\theta) \sin(\theta) \\ k_3(\theta) &= \sin^2(\theta) \end{aligned} \tag{5.7}$$

are calculated.

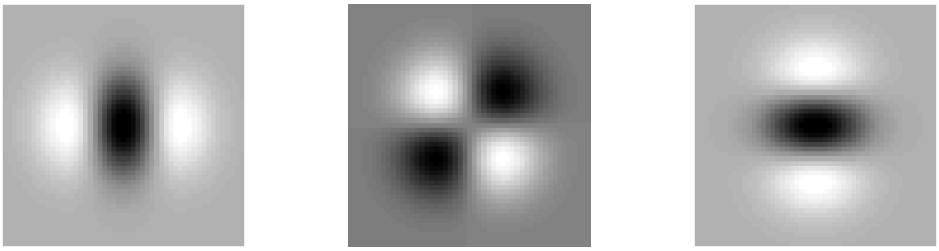
There is also a way given to calculate the terms $R_j(x)$ and $S_j(y)$ of the separable basis functions. However, in [16] is an easier way used to obtain a set of separable steerable basis filters. The three second derivations of a Gaussian are exactly the wanted filters. They are

$$\begin{aligned} G_{xx}(x,y) &= \frac{\partial^2}{\partial x^2} G(x,y) \\ &= \left(4x^2 - 2\right) \cdot e^{-\left(x^2+y^2\right)} \\ G_{xy}(x,y) &= \frac{\partial}{\partial x}\frac{\partial}{\partial y} G(x,y) \\ &= 4xy \cdot e^{-\left(x^2+y^2\right)} \\ G_{yy}(x,y) &= \frac{\partial^2}{\partial y^2} G(x,y) \\ &= \left(4y^2 - 2\right) \cdot e^{-\left(x^2+y^2\right)} \end{aligned} \tag{5.8}$$

With the interpolation functions from (5.7) the response of any rotation of the second derivative $G_2(x, y)$ can be computed using the final result

$$G_2^\theta(x, y) = \cos^2(\theta) \cdot G_{xx}(x, y) - 2 \cdot \cos(\theta) \sin(\theta) \cdot G_{xy}(x, y) + \sin^2(\theta) \cdot G_{yy}(x, y). \quad (5.9)$$

The three basis filter now look like the ones displayed in figure 5.8



(a) Gxx (b) Gxy (c) Gyy

Figure 5.8: Basis set for separable steerable filters based on the second derivatives of a two dimensional Gaussian.

So far only continuous functions in space have been considered. However, for discretely sampled functions, as used when it comes to a computer-aided implementation, the methods are applicable as they are. This is because the order of spatial sampling and steering are interchangeable. Hence, the derived basis function from (5.8) are sampled for x and y values in the domain from -3σ to 3σ to sufficiently cover the filter function. In [10] a range between -2.68 and 2.68 with a sample spacing of 0.67 is suggested, which I applied in my approach. This resulted in an filter with size 9×9. Note that the window size should always be an odd number to ensure that the computed filter result can be assigned to one pixel.

The one-dimensional components in which the filter is separable are easily obtained by taking the middle row as filter in x direction, further denoted as f_1. The filter for the y direction f_2 is then the middle column divided by the the middle value of f_1. This method is applicable for the basis filters G_{xx} and G_{yy}. Filter G_{xy} has a middle column and row respectively, with all zero entries. However, as the filter is symmetric, the one-dimensional kernel f_3 is the same for x and y direction and is calculated via the square root of the diagonal entries.

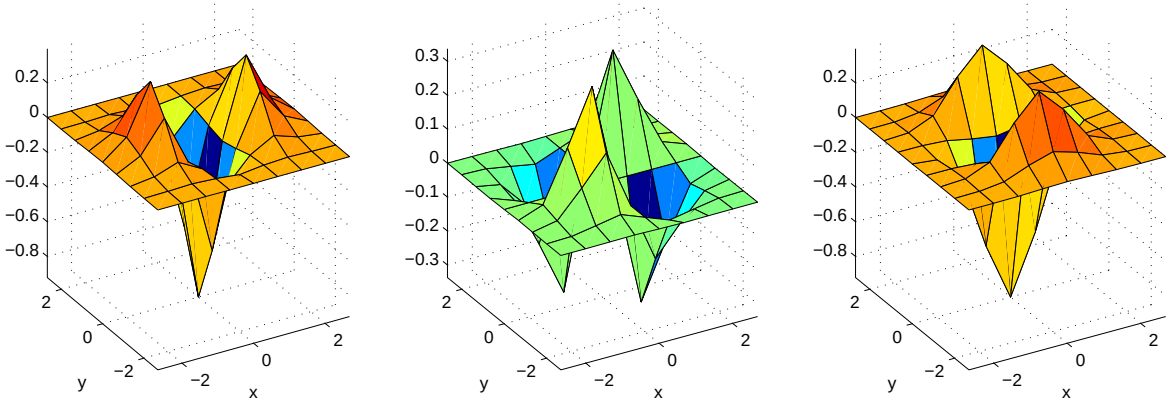
The so determined filter values are shown in Table 5.1 as well as the composition of the basis filter set. The underlying functions are normalized so that the integral over all space of their square equals one. For the functions in (5.8) this ends up in a coefficient of 0.9213 for G_{xx} and G_{yy} and 1.843 for G_{xy}, with which the functions are multiplied.

In Figure 5.9(a) the filters finally obtained separable steerable basis filter set is illustrated. With application of Equation (5.9) the filter in (b) is calculated for $\theta = 30°$. It can be seen that indeed this results in a rotation of 30° of the first basis filter (left most one in the figure).

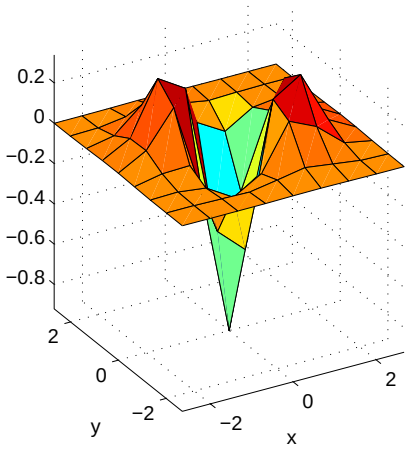
f_1	0.0094	0.1148	0.3964	-0.0601	-0.9213	-0.0601	0.3964	0.1148	0.0094
f_2	0.0008	0.0176	0.1660	0.6383	1.0	0.6383	0.1660	0.0176	0.0008
f_3	-0.0028	-0.0480	-0.3020	-0.5806	0	0.5806	0.3020	0.0480	0.0028

G_2 basis filter	filter in x	filter in y
G_{xx}	f_1	f_2
G_{xy}	f_3	f_3
G_{yy}	f_2	f_1

Table 5.1: 9-tap filters for x-y separable basis set for the second derivative of a Gaussian



(a)



(b)

Figure 5.9: (a) Separable steerable basis filter set for the second Gaussian derivative; (b) result for a rotation of 30°

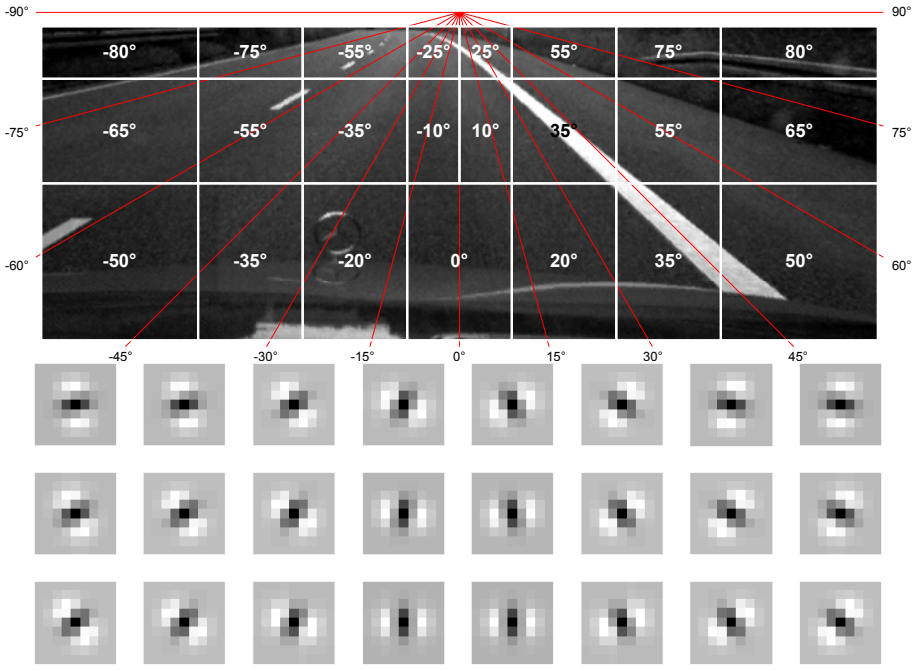


Figure 5.10: Specified direction of steerable filters at dedicated parts of the image according to the expected line orientation

For the application of the filter set to the current video frame, the image is split up into several parts. To each of this parts one dedicated orientation is defined. The angles have been evaluated according to the orientations found, as displayed in Figure 2.3. The result can be seen in Figure 5.10 as well as the resulting orientation of the applied filter. Note that the angles are shifted by -90° in comparison to the predefined orientations, because now the angles define the rotation of the first basis filter G_{xx}, which is oriented to detect vertical lines. Of course this does not mean that all this oriented filters have been applied to the image. Instead only the interpolation functions were adjusted accordingly. The outcome of this filtering procedure with steerable filters is illustrated in figure 5.11.

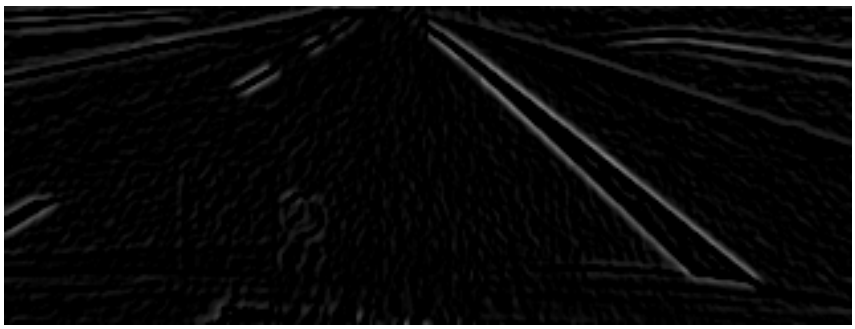


Figure 5.11: Image filtered with steerable filter based on 2^{nd} Gaussian derivative; The filter orientation was according to figure 5.10

5.3.2 Based on First Derivative

The filtering performance looks quite promising. However, as with the Sobel filter, it would be convenient to just detect the inner part of a lane marking. If this is already achieved in the filtering step, effort can be reduced in terms of clustering found lines.

The so far derived steerable filter is based on the second derivative of a Gaussian. As mentioned in Appendix A.1 and illustrated in A.1 this method of edge detection tries to find zero crossings in the second derivation of an image. For a differentiation of inner and outer lane marking edges, it would be necessary to check the function's slope. It would be negative for the left and positive for the right border. In an image like in 5.11 this would require to check for every pixel if it is an edge by evaluating against a threshold. If so, a further analysis of the neighboring pixels in the original image frame will give a statement if the edge is associated with a low-high or high.low transition. This procedure implicates more processing effort than desired.

Therefor, I decided to base the steerable filter on the first derivative, where simply an application of threshold to the filtered image leads to the desired differentiation of inner and out lane boundaries.

The first derivative of a Gaussian in x direction is

$$G_1(x,y) = \frac{\partial}{\partial x}G(x,y) = \frac{\partial}{\partial x}\left(e^{-\frac{x^2+y^2}{\sigma^2}}\right) = -\frac{2x}{\sigma^2}\cdot e^{-\frac{x^2+y^2}{\sigma^2}},$$

which can be stated in polar coordinates as

$$G_1(r,\phi) = -\frac{2}{\sigma^2}\cdot r\cos\phi\cdot e^{-r^2} = -\frac{r}{\sigma^2}e^{-r^2}\cdot\left(e^{\imath\phi}+e^{-\imath\phi}\right).$$

As a consequence only a minimum of two basis filters are required to steer the filter. The formula for the interpolation functions becomes (according to Equation (A.6))

$$\begin{pmatrix}\cos\theta\\ \sin\theta\end{pmatrix} = \begin{pmatrix}\cos\theta_1 & \cos\theta_2\\ \sin\theta_1 & \sin\theta_2\end{pmatrix}\begin{pmatrix}k_1(\theta)\\ k_2(\theta)\end{pmatrix}.$$

To determine a separable steerable basis set it is sufficient to sample the two derivatives in x and y direction, which are

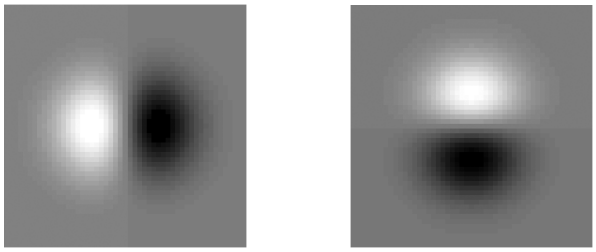
$$\begin{aligned} G_x &= -2x\cdot e^{-\left(x^2+y^2\right)}\\ G_y &= -2y\cdot e^{-\left(x^2+y^2\right)}. \end{aligned}$$

Because G_y is rotated by -90° compared to G_x the interpolation functions yield in

$$\begin{aligned} k_1(\theta) &= \cos\theta\\ k_2(\theta) &= -\sin\theta, \end{aligned}$$

and the complete solution for the separable steerable first derivative of a Gaussian is

$$G_1^{\theta}(x,y) = \cos(\theta)\cdot G_x(x,y) - \sin(\theta)\cdot G_y(x,y).$$



(a) Gx (b) Gy

Figure 5.12: Separable steerable basis set for the first Gaussian derivative

f_1	0.0026	0.0450	0.2833	0.5445	0	-0.5445	-0.2833	-0.0450	-0.0026
f_2	0.0008	0.0176	0.1660	0.6383	1	0.6383	0.1660	0.0176	0.0008

G_1 basis filter	filter in x	filter in y
G_x	f_1	f_2
G_y	f_2	f_1

Table 5.2: 9-tap filters for x-y separable basis set for first derivative G_1

The one-dimensional kernels for the two basis filters are obtained in the same way, as described in the approach for the second derivative. Again they where normalized so that the integral of their square equals one, which resulted in a coefficient of 0.6366 for both G_x and G_y. Table 5.2 lists the finally used filters and their usage for a two-dimensional filtering. In Figure 5.13 the basis filters are displayed, as well as the outcome for a rotation of 60°.

When applied to a video frame, the filter showed a rather good performance. However, to detect the inner boundary of lane markings, the filter orientation as listed in Figure 5.10(a) does not hold true anymore. For the right hand part of the image the outer boundaries are detected. Therefore, the filter orientation was modified by 180° leading to the orientations in Figure 5.14. The result finally obtained for the edge detection step in my approach is displayed in Figure 5.15.

5.4 Threshold Computation

To gain an explicit image of the detected edges, it is appropriate to convert the image into a binary representation, where the value 1 specifies an edge and value 0 is background. For this purpose it is necessary to define a threshold. All values greater than this threshold are set to 1, all smaller this threshold are set to 0.

For developing and testing my algorithm I used Matlab and for the computation of a threshold explicitly the function `graythresh`, which is an adaptive method. It uses Otsu's

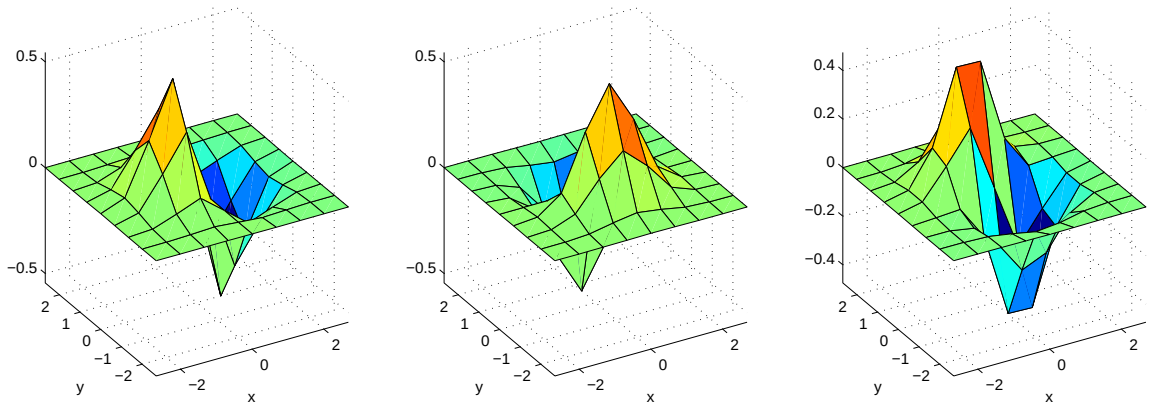


Figure 5.13: Separable steerable basis filter set for the first first Gaussian derivative, as well as the result for a rotation of 60°.

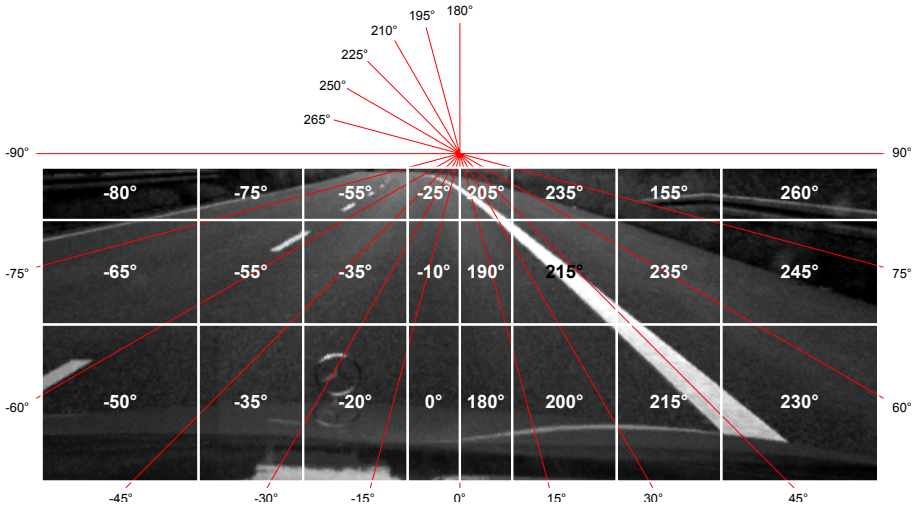


Figure 5.14: Modified filter orientation for the approach based on the first Gaussian derivative

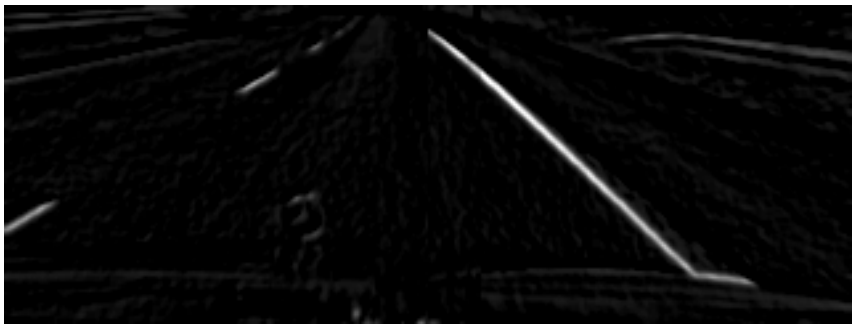


Figure 5.15: Example video frame determined with steerable filter based on the first derivative of a Gaussian

method [26], which chooses the threshold to minimize the intraclass variance of the black and white pixels. This also means that it maximizes the between class variance. Hence, the threshold is always set in a way, that that all important edges are represented as a binary 1 and are perfect separated from the background.

Indeed, this method is not practical for my implementation on the mobile phone. It simply produces too much computational effort. However, I found that the threshold keeps quite constant over the frames of a video. For an original gray level input image with an value range from 0 to 255, the computed threshold was balanced about a value of 50. Therefore, I tried to implement a constant threshold and figured out, that the result was not much worse than with the dynamicly calculated threshold. For this reason in my mobile phone application the threshold is set to a constant value.

5.5 Performance Evaluation

The Sobel filter and the finally derived steerable filter G_1 were applied to a number of different video frames and compared to each other. It turned out that the steerable filter performs better in certain situations. While lane markings are detected similarly well, the Sobel filter additionally detects edges of passing cars or other similar objects. Because the steerable filter is tuned in direction of the lines, it is possible to suppress such interferences. The images in Figure 5.17 illustrate results for several video frames.
The Sobel filter can be seen as a very rough approximation of the first derivative of a Gaussian. Hence, it would also be possible to orient the Sobel filter like the steerable filter with $S = \cos\theta \cdot S_x - \sin\theta \cdot S_y$. However, the result is not sufficiently improved. This comes from the small filter size of the Sobel filter. It simply is not big enough to satisfactorily steer the filter.

Performing both filters on a set of 250 video frames it turned out that the steerable filter is hardly slower than the Sobel filter. The diagram in Figure 5.16 illustrates measurements for a resolution of 320×120 and 160×60 pixel, performed on the development PC.

Because of the better detection results and the small difference in processing time, it is more promising for the following line extraction to use an image filtered with the steerable filter G_1.

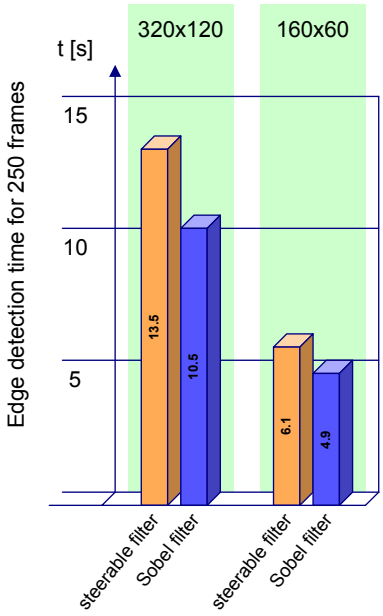


Figure 5.16: Performance of steerable and Sobel filter, operating on a set of 250 frames with a resolution of 320x120 and 160x60 pixels

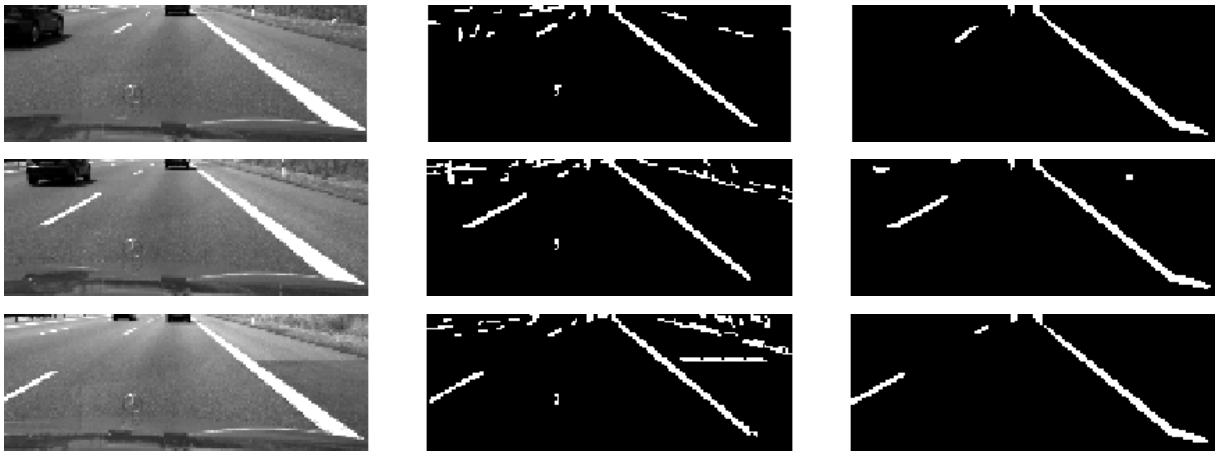


Figure 5.17: Image frames filtered with Sobel filter on the left and steerable filter on the right.

Chapter 6

Postprocessing of Edge Images

In this section the approach is described how the edge filtered image from Chapter 5 is processed further on. The goal of the postprocessing is to find and extract lines in the filtered image. Attention was payed to a robust and fast algorithm, which works properly for different kinds and shapes of roads.

First a mathematical representation of lines is given, which is used further on to describe the detected lines in the image. After that sections are following about the main concept behind finding and parameterizing lines in the image. Some methods are presented to locate outliers and discard incorrectly detected lines. Afterwards an algorithm is described to cluster potential lines. At the end a short recapitulation is given about the discussed line detection and graphical results of the method are illustrated.

6.1 Line Representation

There are a few ways to describe a line in an image plane. One way is to express the coordinate y with respect of x, like $y = k \cdot x + d$, where k denotes the slope and d the point where the line intercepts with the y-axis. The disadvantage of this representation is that vertical lines would need an infinite slope. However, a parametrization with $k = \infty$ is not suitable for further computation.

Another way to parametrize a line is using a implicit form with

$$a \cdot x + b \cdot y + c = 0, \tag{6.1}$$

where the parameters a, b and c are defined only up to scale. Multiplying both sides of the equation by a constant still describes the same line, but with different parameters a, b and c.

Writing this representation in vector notation

$$\begin{pmatrix} a & b \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} + c = \mathbf{n}^T \cdot \begin{pmatrix} x \\ y \end{pmatrix} + c = 0$$

leads to the normal form equation of a line with the normal vector $\mathbf{n}$ perpendicular to the line. $\mathbf{n}$ has the two entries a and b.
Dividing both sides by the norm of $\mathbf{n}$ eliminates the problem with multiple parameter sets for one and the same line as stated above. The equations become

$$\frac{\mathbf{n}^T \cdot \begin{pmatrix} x \\ y \end{pmatrix} + c}{\|\mathbf{n}\|} = 0$$

and further on

$$\mathbf{n}_0^T \cdot \begin{pmatrix} x \\ y \end{pmatrix} = d.$$

This representation is nothing else than the Hesse normal form, where

$$\mathbf{n}_0 = \frac{\begin{pmatrix} a \\ b \end{pmatrix}}{\sqrt{a^2 + b^2}}$$

and

$$d = -\frac{c}{\sqrt{a^2 + b^2}}.$$

Note that d is the distance from the line to the origin. Later on this correlation will be used to compute the distance of a line to an arbitrary point like the vanishing point.

Every point $\mathbf{r}$ on the line may also be described in parameter form with

$$\mathbf{r} = \mathbf{r_0} + \lambda \cdot \mathbf{u}, \tag{6.2}$$

where $\mathbf{r_0}$is an arbitrary point on the line and $\mathbf{u}$ is a vector pointing into the direction of the line.

While the first parametrization with k and d is useless for my further approach, both, parameter set $\begin{bmatrix} a & b & c \end{bmatrix}$ and parameter representation with $\mathbf{r_0}$ and $\mathbf{u}$ will be used.

6.2 Tile Segmentation

After edge detection an image might look like the one in Figure 6.1 and is represented via a binary image with

$$b(x, y) = \begin{cases} 1 & \text{for pixels belonging to a line} \\ 0 & \text{for background pixels} \end{cases}.$$

The idea behind detecting lines in the binary image is to split up the image in several square tiles. With an appropriately chosen size each tile will contain just one part of a line. Experiments have shown that a segmentation in 16×6 tiles leads to the best

Figure 6.1: Sample binary image after edge detection

results. For a used image size of 320×120 pixel this results in an tile size of 20×20 pixel. For an image size of 160×60 pixel as used for the mobile phone and explained in the implementation part this results in a tile size of 10×10 pixel. Note that the following representation is shown with twice the size of the tiles than used in the algorithm, as it is more meaningful for demonstration and explanation.

For each tile a point lying on the line and the direction vector of the line are calculated. Referring to Equation (6.2) the found line is completely parametrized.
For this reason the image area is calculated first via

$$A = \sum_x \sum_y b(x, y).$$

To exclude tiles with insufficient features from further calculations, the ones with an area smaller a constant threshold are skipped. It is pointed out that a value of 5 is well performing.

To get an arbitrary point on a line I decided to calculate the tile's center of mass. If a tile contains just one line, its centroid matches exactly with the centroid of the illustrated line. Therefore, it is possible to calculate one point on a line by computing the image's centroid, which is given by the 1^{st} moment of the object. The equation for the x and y coordinate are

$$x_M = \frac{\sum_x \sum_y x \cdot b(x, y)}{\sum_x \sum_y b(x, y)} = \frac{\sum_x \sum_y x \cdot b(x, y)}{A}$$

and

$$y_M = \frac{\sum_x \sum_y y \cdot b(x, y)}{\sum_x \sum_y b(x, y)} = \frac{\sum_x \sum_y y \cdot b(x, y)}{A}.$$

The result for this step can be seen in Figure 6.2. In the last row there are some tiles with interferences, which do not belong to lines representing a lane. It can also be noticed that there are some tiles, where the center of mass does not lie on any line. This is the case, if two or more discontiguous features have been extracted in one tile. In the following process such outliers have to be detected and discarded.

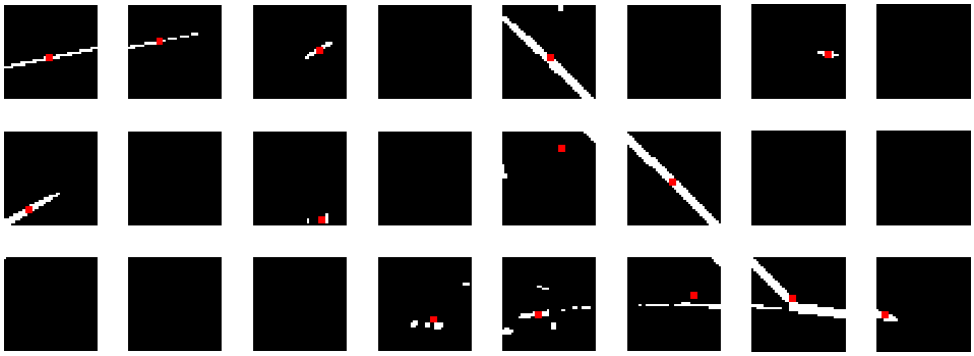


Figure 6.2: Calculated centroids for each tile

Figure 6.3: Dashed line with center of mass

One might think that it is sufficient to evaluate the pixels in the neighborhood of the centroid. If the value of at least some of the pixels is 1 and, therefore, characterizing a detected feature, the center of mass is lying on a line. If not, there are some interferences in the tile and the tile can be rejected. Although this might be correct in the majority of cases, there is one situation where the method fails. Imagine a dashed line like in Figure 6.3. The center of mass is calculated correctly and is lying on the extension of the lane marking, but the neighborhood pixels are all of value 0. As it would be incorrect to skip that tile, a neighborhood evaluation is not promising and other actions are required later-on.

To be able to find the direction vector of the line, I decided to perform a so called *Principal Component Analysis*. This computation of the principal axes of a binary object can be easily done, implementing an eigenvalue decomposition of the covariance matrix for every tile. The ratio of the eigenvalues says something about the shape of the extracted feature and the eigenvector belonging to the greater eigenvalue is showing in direction of the greatest variance.

For the following calculation the coordinate system origin is shifted into the detected center of mass, leading to the substitutes

$$\begin{aligned} x' &= x - x_M \\ y' &= y - y_M. \end{aligned} \tag{6.3}$$

The covariance matrix is built upon the three 2^{nd} image moments, which are σ_{xx}^2, the variance of the binary image for the x-direction, σ_{xy}^2, the correlation between the x and y-

directions, and σ^2_{yy}, the variance of the binary object for the y-direction. These variances can be computed as

$$\begin{aligned}\sigma^2_{xx} &= \sum_{x'}\sum_{y'} x'^2 \cdot b(x', y') \\ \sigma^2_{xy} &= \sum_{x'}\sum_{y'} x' \cdot y' \cdot b(x', y') \\ \sigma^2_{yy} &= \sum_{x'}\sum_{y'} y'^2 \cdot b(x', y')\end{aligned}$$

and are combined to form the covariance matrix

$$\mathbf{C} = \begin{pmatrix} \sigma^2_{xx} & \sigma^2_{xy} \\ \sigma^2_{xy} & \sigma^2_{yy} \end{pmatrix}.$$

In the 2×2 covariance matrix $\mathbf{C}$ there are two eigenvalues λ_1 and λ_2 as well as two eigenvectors $\mathbf{q}_1$ and $\mathbf{q}_2$. The general eigenvalue decomposition is

$$\mathbf{QDQ}^T = \begin{pmatrix} \mathbf{q}_1 & \mathbf{q}_2 \end{pmatrix} \cdot \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{q}_1^T \\ \mathbf{q}_2^T \end{pmatrix} = \mathbf{C}.$$

For extracting these values the following derivation holds true:

$$\begin{aligned}\mathbf{C} \cdot \mathbf{q} &= \lambda \cdot \mathbf{q} \\ (\mathbf{C} - \mathbf{I} \cdot \lambda) \cdot \mathbf{q} &= \tilde{\mathbf{0}}\end{aligned} \tag{6.4}$$

The eigenvalues can be calculated by working out the well known characteristic polynomial

$$\lambda^2 - \left(\sigma^2_{xx} + \sigma^2_{yy}\right) \cdot \lambda + \left(\sigma^2_{xx} \cdot \sigma^2_{yy} - \left(\sigma^2_{xy}\right)^2\right) = 0$$

where the roots are the solution for the eigenvalues

$$\lambda_{1,2} = \frac{\sigma^2_{xx} + \sigma^2_{yy}}{2} \pm \sqrt{\left(\frac{\sigma^2_{xx} + \sigma^2_{yy}}{2}\right)^2 - \left(\sigma^2_{xx} \cdot \sigma^2_{yy} - \left(\sigma^2_{xy}\right)^2\right)}. \tag{6.5}$$

Because the covariance matrix is symmetric ($c_{12} = c_{21}$), the solution for (6.5) are two different positive real eigenvalues.

With the knowledge of the eigenvalues, the two eigenvectors can be computed by inserting λ_1 and λ_2 respectively in $(\mathbf{C} - \mathbf{I} \cdot \lambda)$ and solving the linear equation (6.4) for $\mathbf{q}$. Note that

the matrix $(\mathbf{C} - \mathbf{I} \cdot \lambda)$ does not have full rank, therefore $\mathbf{q}$ can be determined by solving just one row. The eigenvectors are

$$\begin{aligned} \mathbf{q}_1 &= \begin{pmatrix} \lambda_1 - \sigma_{yy}^2 \\ \sigma_{xy}^2 \end{pmatrix} = k_1 \cdot \begin{pmatrix} \sigma_{xy}^2 \\ \lambda_1 - \sigma_{xx}^2 \end{pmatrix} \\ \mathbf{q}_2 &= \begin{pmatrix} \lambda_2 - \sigma_{yy}^2 \\ \sigma_{xy}^2 \end{pmatrix} = k_2 \cdot \begin{pmatrix} \sigma_{xy}^2 \\ \lambda_2 - \sigma_{xx}^2 \end{pmatrix} \end{aligned} \tag{6.6}$$

k_1 and k_2 are just constants. The eigenvectors can be computed by either one expression. If normalized to length 1, both terms are the same and k equals to1.

Suppose that $\lambda_1 > \lambda_2$. These eigenvalues describe the variances of the binary tile along the major and minor principal axes, i.e. $\mathbf{q}_1$ and $\mathbf{q}_2$. The greater eigenvalue λ_1 is identical to the maximum intensity, while λ_2 stands for the minimum intensity in the tile. As a result, a tile like the top left in Figure 6.2 showing just one part of a line has got a high variance in direction of the line and a low variance in direction normal to the line. Therefore, the ratio $\frac{\lambda_1}{\lambda_2}$ will be a high value, enabling to skip all tiles where the ratio is below a certain threshold. I found out, that a threshold between 5 and 10 leads to the best results, skipping areas with no lines in it, but still taking tiles into account where a thick line (lower ratio) has been determined. Figure 6.4 shows an example containing three different tiles with the center of mass marked in red and the direction of the eigenvector belonging to the greater eigenvalue in blue. The second eigenvector would lie perpendicular on the one shown.

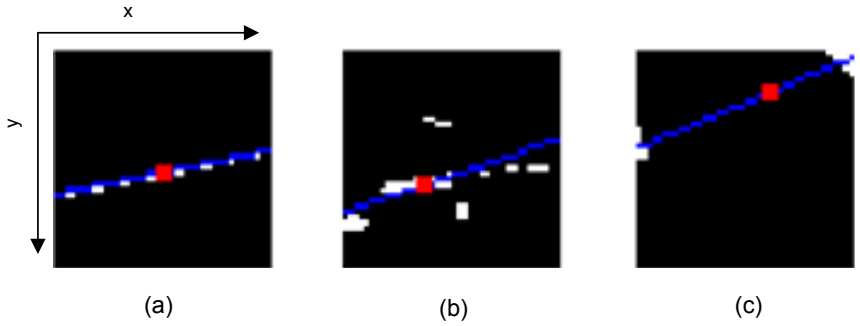


Figure 6.4: Three sample tiles with center of mass and major principal axes

tile	(1)	(2)	(3)
eigenvalues	$\lambda_{1,2} = \begin{cases} 6890 \\ 9.2 \end{cases}$	$\lambda_{1,2} = \begin{cases} 5515 \\ 979 \end{cases}$	$\lambda_{1,2} = \begin{cases} 8604 \\ 57 \end{cases}$
eigenvalue ratio	745.4	5.63	150.9
major eigenvector	$\mathbf{q}_1 = \begin{pmatrix} -0.98 \\ 0.2 \end{pmatrix}$	$\mathbf{q}_1 = \begin{pmatrix} -0.95 \\ 0.32 \end{pmatrix}$	$\mathbf{q}_1 = \begin{pmatrix} -0.92 \\ 0.39 \end{pmatrix}$

Table 6.1: Results for the three images in Figure 6.4

In tile (a) the line has been perfectly detected. In tile (b) there are some interferences caused by imperfect edge detection. However, a look in Table 6.1 shows, that the eigenvalue ratio for this image is less compared to the first one, enabling to discard the line found. Inspecting the results for tile (c) shows an interesting problem of the method. There are two small features detected in the image causing a high variance in direction of the computed major principal axis. However, the detected line does not match with a real line in the image at all. Analyzing the eigenvalue ratio with an value of 150.9 demonstrates that this detected line cannot be rejected by a simple eigenvalue ratio threshold. Thus, there are further actions required for evaluating the computed lines. This is done in the next step of my algorithm, where the distance of the line to the vanishing point is calculated (see section (6.4)).

Figure 6.5 shows the result for sample image in Figure 6.1 for the principal component analysis. The threshold for eigenvalue ratio was set to 10 in this example.

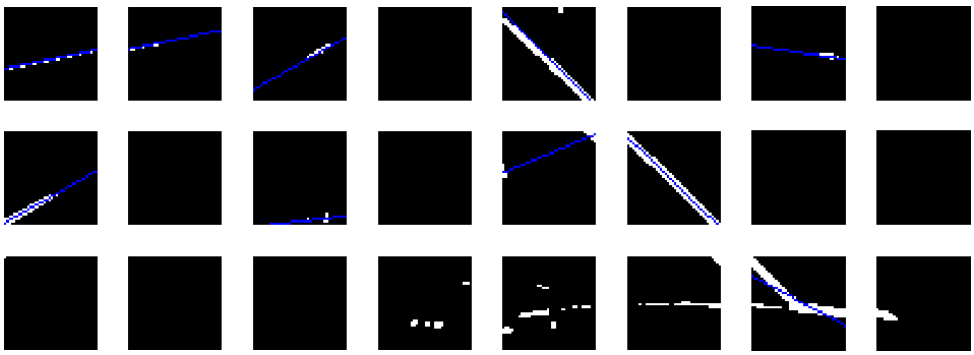


Figure 6.5: Result for principal component analysis with an eigenvalue ratio threshold of 10

The covariance matrix is symmetric and positive semi-definite, which means that all eigenvalues are positive or zero. Therefore, the eigenvalues and eigenvectors are identical to the singular values and singular vectors. Instead of performing an eigenvalue decomposition, it is also possible to implement a singular value decomposition

$$\mathbf{U}\mathbf{S}\mathbf{V}^T = \mathbf{C},$$

with

$$\begin{aligned} \mathbf{U} &= \begin{pmatrix} \mathbf{q}_1 & \mathbf{q}_2 \end{pmatrix} \\ \mathbf{S} &= \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \\ \mathbf{V} &= \mathbf{U}. \end{aligned}$$

$\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices, which must not be the case for $\mathbf{Q}$ in the principal component analysis. $\mathbf{S}$ contains the singular values in decreasing order. The last statement $\mathbf{V} = \mathbf{U}$ holds true, because $\mathbf{C}$ is a symmetric matrix.

For matrices of greater size the singular value decomposition can be computed faster than the eigenvalue decomposition. In my case with a 2×2 covariance matrix, computing

the eigenvalues and eigenvectors with the Equations (6.5) and (6.6) cannot be done more efficiently.

6.3 Finding Global Line Parameters

For every tile where the eigenvalue ratio was not below the defined threshold, the calculated line is described by the center of mass $\begin{pmatrix} x_M \\ y_M \end{pmatrix}$ and one eigenvector $\mathbf{q}$ in direction of the line. For clarity this eigenvector is written as $\mathbf{v}$ further on. Referring to the line parametrization in section 6.1 in parameter form (Equation (6.2)) the line is completely described by

$$l(k) = \begin{pmatrix} x_M \\ y_M \end{pmatrix} + k \cdot \mathbf{v}.$$

However, this parametrization is not suitable for further calculations and is just valid inside the tile. To express the lines in the coordinate system of the whole image a parameterization is necessary.

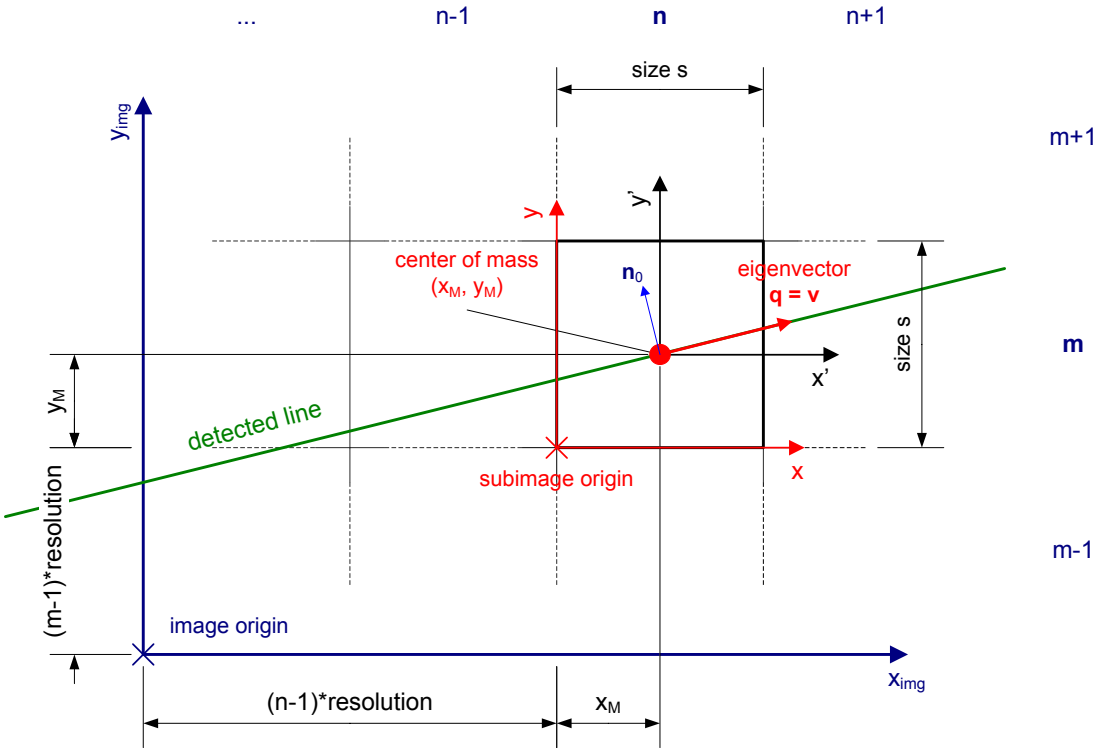


Figure 6.6: Correlation between tile and image coordinates

Figure 6.6 illustrates the correlation between the tile coordinates considered so far and the image coordinates. Note that in an image the coordinate origin is in the top left corner with the y-axis facing downwards. In Figure 6.6 the y-axis faces upwards for a

more familiar representation. All the following derivations are valid just as they are, hence horizontally flipping the image results in the shown coordinate system.

Let x' and y' be a coordinate system with its origin in the center of mass, as previously denoted and illustrated in Figure 6.6. A line in direction of the eigenvector $\mathbf{v} = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$ can than be described as $y' = \frac{v_2}{v_1} \cdot x'$ or in an implicit notation with

$$v_1 \cdot y' - v_2 \cdot x' = 0.$$

Referring to the substitution from Equation (6.3) this becomes

$$v_1 \cdot (y - y_M) - v_2 \cdot (x - x_M) = 0. \tag{6.7}$$

A further investigation of the schema in Figure 6.5 yields to the relation

$$\begin{aligned} x_{img} &= (n-1) \cdot s + x \\ y_{img} &= (m-1) \cdot s + y \end{aligned}$$

or solved for x and y

$$\begin{aligned} x &= x_{img} - (n-1) \cdot s \\ y &= y_{img} - (m-1) \cdot s, \end{aligned} \tag{6.8}$$

where the variable s states the size of one tile in pixels.

Values m and n in Figure 6.6 describe the two-dimensional index of the tile in relation to the whole image. Their range is from 1 to $\frac{image\,width}{size}$ or $\frac{image\,height}{size}$ respectively. Equation (6.8) inserted in (6.7) results in the final line parameterization in dependence of the image coordinates

$$\begin{aligned} v_1 \cdot (y_{img} - (m-1) \cdot s - y_M) - v_2 \cdot (x_{img} - (n-1) \cdot s - x_M) &= 0 \\ -v_2 \cdot x_{img} + v_1 \cdot y_{img} + v_2 \cdot \underbrace{((n-1) \cdot s + x_M)}_{x-offset} - v_1 \cdot \underbrace{((m-1) \cdot s + y_M)}_{y-offset} &= 0. \end{aligned}$$

Compared to Equation (6.1) the three parameters a, b and c have got the following values

$$\begin{aligned} a &= -v_2 \\ b &= v_1 \\ c &= v_2 \cdot ((n-1) \cdot s + x_M) - v_1 \cdot ((m-1) \cdot s + y_M). \end{aligned} \tag{6.9}$$

Thess three parameters are further used to describe a line. If ensured that the eigenvalues are normalized, i.e. their length equals 1, the above parametrization is unique. Moreover, it is exactly the Hesse normal form with the normal vector

$$\mathbf{n}_0 = \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} -v_2 \\ v_1 \end{pmatrix}.$$

Because the second eigenvector belonging to the smaller eigenvalue always lies perpendicular to the detected line, $\mathbf{n}_0$ equals $\mathbf{q}_2$, with $\|\mathbf{q}_2\| = 1$.

6.4 Plausibility Check for Detected Lines

The previous steps of my algorithm provided a robust and fast method for detecting lines in an edge filtered image. However, it cannot be assured that all lines detected belong to lane markings, but represent some other features. In Figure 6.4 tile (c) is an example given, where a line is detected by the algorithm, which does not exist in reality.
As the recorded image of the road shows a perspective distortion, all lines proceed to one point called the vanishing point. To be able to detect wrong lines I decided to measure the distance from every line to the vanishing point, enabling to discard lines where the distance is greater than a certain set threshold.

A distance measurement can be easily done, if the line is parametrized in Hesse normal form. With the parameter set $\begin{bmatrix} a & b & c \end{bmatrix}$ this is

$$\begin{pmatrix} a & b \end{pmatrix} \cdot \begin{pmatrix} x_{img} \\ y_{img} \end{pmatrix} + c = 0. \tag{6.10}$$

Equation (6.10) only holds true if the image point lies on the line. However, if this is not the case, the result equals the distance of the point to the line. Therefore, a distance measurement is done by inserting the vanishing point noted as

$$\mathbf{vp} = \begin{pmatrix} x_{vp} \\ y_{vp} \end{pmatrix}$$

in (6.10), which leads to the final formula

$$d = \begin{pmatrix} a & b \end{pmatrix} \cdot \mathbf{vp} + c,$$

with $\mathbf{d}$ being the distance to the vanishing point.

If the norm of $\begin{pmatrix} a & b \end{pmatrix}$ does not equal to 1 (the length of the eigenvector was not equal to 1) the formula must be normalized as

$$d = \frac{\begin{pmatrix} a & b \end{pmatrix} \cdot \mathbf{vp} + c}{\sqrt{a^2 + b^2}}.$$

Also note that c has to be divided by the therm $\sqrt{a^2 + b^2}$, because c includes a and b in its calculation (see Equation (6.9)).

Figure 6.7 illustrates the different distances, which have been calculated for the sample image. Lines detected in tiles provided with a distance value in red will be skipped in the further computation. A distance threshold between 10 and 30 has generated the best performance.

So far it has not been stated how the coordinate values of the vanishing point are evaluated and set. In most images the vanishing point lies in the middle on the top border, but this might be a rough approximation. Thus, the vanishing point is calculated dynamically for

every frame with the information of the previously found lines. Actually for initialization, coordinates of the middle of the top image border are assigned to the vanishing point. Note that the vanishing point does not have to lie inside the image. As it is just used for computation, the coordinates may also have negative values if situated outside the image area.

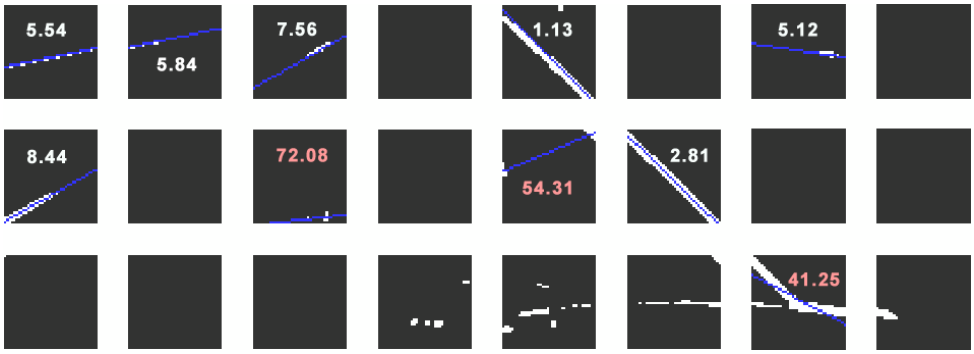


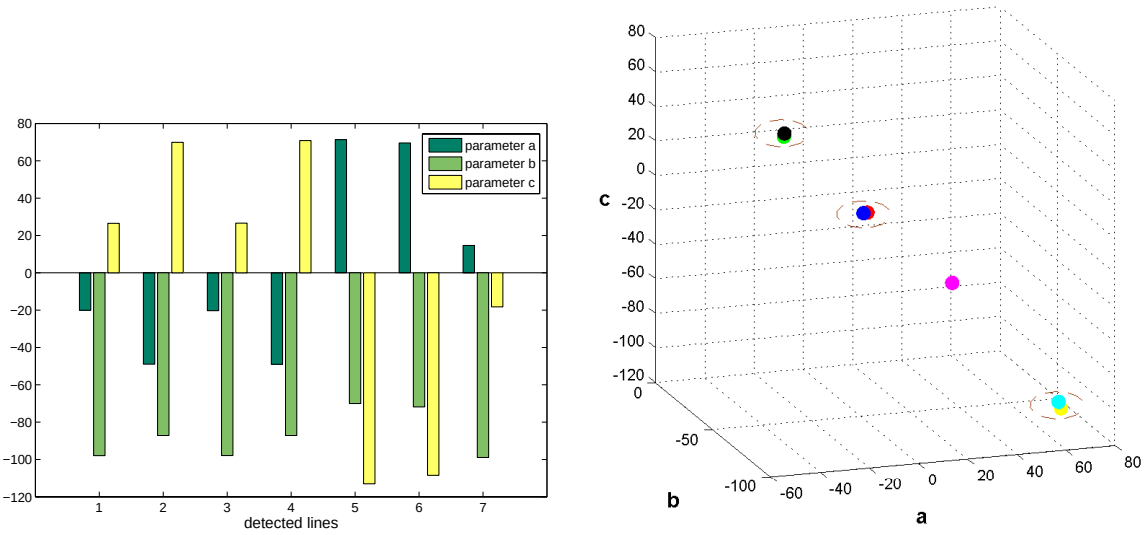
Figure 6.7: Distance from vanishing point for the detected lines

6.5 Clustering Detected Lines

After analyzing each tile, a set of potential lines has been found. Despite the exploration of image area, eigenvalue ratio and distance from the vanishing point to reject wrong detected lines, there still may be some outliers which do not match with a lane marking in reality. Moreover, for one and the same lane marking multiple parameter sets can be existent. This is because features of a lane marking filtered by edge detection are likely to be present in more than just one tile.

Figure 6.8(a) displays the parameter sets [a b c] as introduced in Equation 6.9 for the lines detected in the example image used throughout this paper. It can be seen that line 1 and 3 almost have got the same parameters. The same holds true for lines 2 and 4, as well as for line 5 and 6. The parameter set of line 7 just occurs once.

For the clustering a simple distance measurement in the three-dimensional space will be used. Parameter a and b lie in the range from -1 to 1, as the are the components of the line's normal vector, which lengths equals 1. It turned out that the range of parameter c is about 100 times greater. A distance evaluation with this parameters would not make sense, as parameter c is weighted a lot stronger. To equally weight the three parameters and get the distance between the different parameter sets, their values would have to be scaled considering their statistical distribution. A perfect approach would require to choose the weights so that the variance of the different parameters is the same. However, this would need more computational effort, but does not improve my clustering, I decided to scale the fist two parameters by a predefined constant of value 100. The illustrations in Figure 6.8 already include this scaling.



(a) (b)

Figure 6.8: (a) Line parameters for the detected lines; (b) Displayed as points in terms of their parameters

Having a look at the seven parameter sets display in three-dimensional space, as shown in Figure 6.8(b) makes clear which lines have to be merged. The distance between the clustered lines (assigned with a circle) in relation to the distance between the lines inside such a cluster, demonstrate that a predefined scaling value is sufficient.

To enable an easier understanding of the developed clustering algorithm, Figure 6.9 shows the different steps. The algorithm starts with the first line $\mathbf{l}_1 = \begin{pmatrix} a_1 & b_1 & c_1 \end{pmatrix}^T$ (marked with a red number in the drawing) determining the distance d to all other lines with the general formula

$$d = (\mathbf{l}_1 - \mathbf{l}_n)^T \cdot (\mathbf{l}_1 - \mathbf{l}_n) = (a_1 - a_n)^2 + (b_1 - b_n)^2 + (c_1 - c_n)^2 .$$

If the distance is below a certain threshold, the lines will be clustered. After the first iteration all found line parameters that belong together are reduced to one parameter set by calculating the mean value. In the example lines 1 and 3 are merged. The next iteration of the algorithm observes the remaining lines, again performing a distance measure form the first (remaining) line to the other (remaining) lines. In the example lines 2 and 4 are clustered. This procedure is performed until no more lines are left over. Of course it can happen that no similar line with a low distance is found, as in step 4. If this is the case the line is discarded and not added to the parameter set of clustered lines. Hence, it is necessary to find at least two similar lines in the different tiles.

With this additional restriction wrong lines or outliers are rejected, but it might also happen that correct lines are lost, especially when the lane marking is dashed or in a poor condition. To work against this loss, lanes found in the previous frame are also taken

into account in the clustering algorithm. This is possible as the lane markings do not change abruptly between one frame and the following one, but show a smooth transition. Therefore, just one line has to be found in the image to follow up an already detected line, but at least two lines must be located to take a new line into consideration.

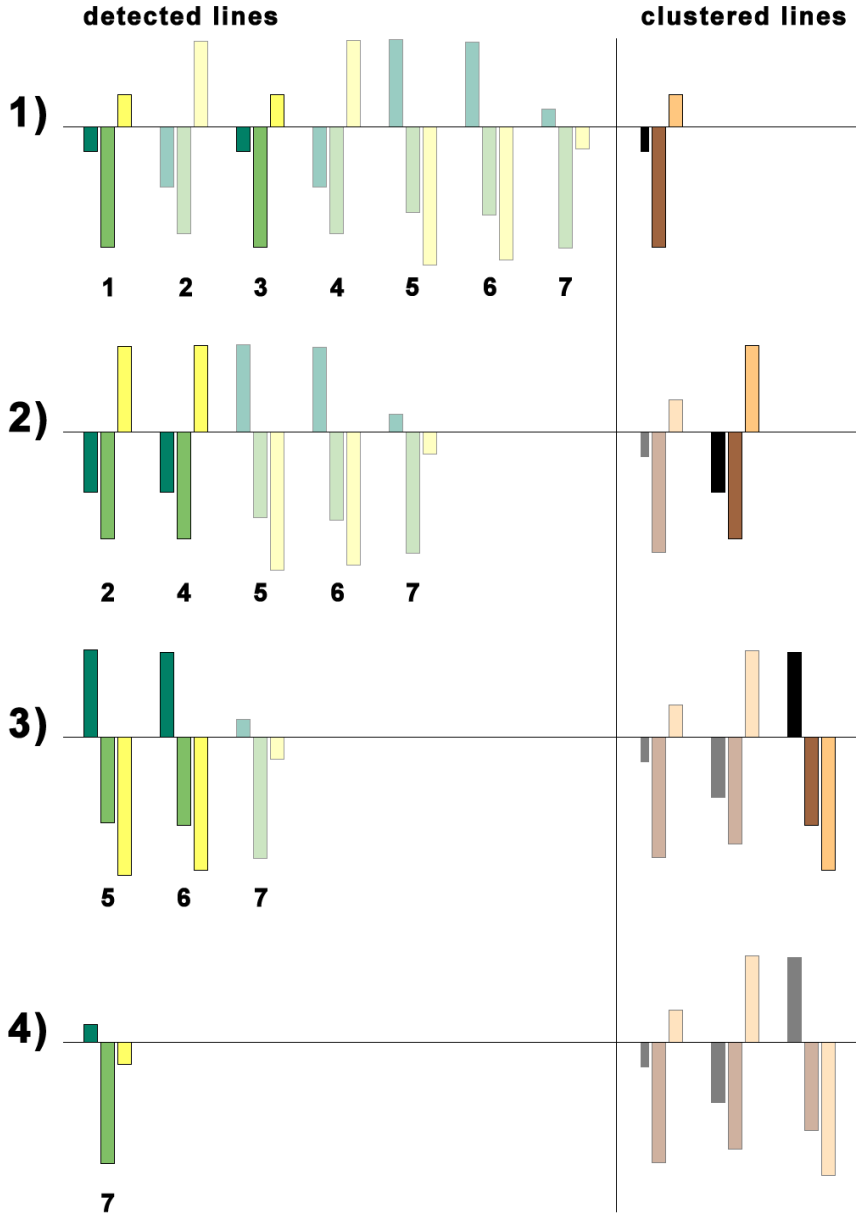


Figure 6.9: Visualization of the clustering steps passed through by the clustering algorithm

6.6 Dynamic Calculation of Vanishing Point

The vanishing point of the recorded road image always lies in a special area, but changes dynamically depending on the course of the road. To get better results when evaluating the currently found line parameters (see 6.4), the vanishing point is recalculated in every frame.
The detected lines in the previous frame are taken to perform the calculation. With a least square optimization the intersection of the lines becomes the new vanishing point. For computing an intersection at least two lines must have been found, otherwise the previous vanishing point is kept.

Assuming that n lines have been detected and they do not intersect exactly in one point, a central point is the best approximation for the vanishing point. I use a least square optimization, which minimizes the sum of the square of distances from all the lines to the central point. Mathematically a line is represented by the three parameters $\begin{bmatrix} a & b & c \end{bmatrix}$. For every point (x, y) on the line the equation

$$a \cdot x + b \cdot y + c = 0$$

holds true. In vector notation this becomes

$$\begin{pmatrix} a & b & c \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = 0.$$

Note that $\begin{pmatrix} x & y & 1 \end{pmatrix}^T$ can be interpreted as a representation of the point in homogeneous coordinates. With n lines, n parameters sets must be taken into consideration, leading to the matrix vector notation

$$\begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ \vdots & \vdots & \vdots \\ a_n & b_n & c_n \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \mathbf{0}. \tag{6.11}$$

$\begin{pmatrix} x & y & 1 \end{pmatrix}^T$ represents the vanishing point, as it is the only point which lies on all of the lines and therefore is situated in the intersection of the considered lines. Equation (6.11) can be rewritten as

$$\begin{pmatrix} a_1 & b_1 \\ a_2 & b_2 \\ \vdots & \vdots \\ a_n & b_n \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = - \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix}, \tag{6.12}$$

leading to the final optimization equation. Expressing the equation as $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$, $\mathbf{x}$ can be determined directly via the inverse of $\mathbf{A}$, if $\mathbf{A}$ is a 2×2 matrix. Otherwise the set of linear equations is overdetermined and $\mathbf{x}$ must be calculated by minimizing the norm of (6.12) via

$$\arg\min_{\mathbf{x}} \left\| \mathbf{A} \cdot \mathbf{x} - \mathbf{b} \right\|_2,$$

leading to the well known solution

$$\mathbf{x} = \mathbf{A}^{+} \cdot \mathbf{b} = \left(\mathbf{A}^{T} \cdot \mathbf{A}\right)^{-1} \cdot \mathbf{A}^{T} \cdot \mathbf{b},$$

where $\mathbf{x}$ is is new vanishing point with the coordinates $\begin{pmatrix} x \\ y \end{pmatrix}$ and $\mathbf{A}^{+}$ is the *Moore-Penrose pseudoinverse.* It is easy to show that this calculation also holds true for $\mathbf{A}$ being of size 2×2. With the inserted parameters of the previously found lines my final equation for the vanishing point $\mathbf{vp}$ is

$$\mathbf{vp} = \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a_1 & b_1 \\ a_2 & b_2 \\ \vdots & \vdots \\ a_n & b_n \end{pmatrix}^{+} \cdot \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix}.$$

As $\mathbf{A}$ has only two columns the term $\left(\mathbf{A}^{T} \cdot \mathbf{A}\right)$ always leads to a 2×2 matrix, hence reducing the computational effort for the inversion to a minimum.

To prevent a too intense movement of the vanishing point, the vanishing point of the previous frame is incorporated as well. The equation for the final smoothed vanishing point is

$$\mathbf{vp} = \frac{2}{3} \cdot \mathbf{vp} + \frac{1}{3} \cdot \mathbf{vp}_{\text{prev}}.$$

6.7 Review and Results

All the discussed steps of my line detection algorithm are shown in Figure 6.10 for a better understanding of the relation between the different steps. The outcome of the sequence of operations is a set of parameters describing the detected lines. These lines found are in combination with the new edge filtered video frame the input parameters for the following run of the algorithm. For initialization purposes or if no lines have been found, the line parameter input is empty.

For the example used throughout this paper the lines found and the calculated vanishing point are illustrated in Figure 6.11. It can be seen that the lane markings are detected perfectly and all outliers and incorrectly found lines during this process have been discarded correctly. Note that only the lower half of the video frame is used for the algorithm, as the upper part does not contain any lane markings. Therefore, it is not necessary for the calculation, but shown here for a better demonstration of the road scene.

Figure 6.12 and 6.13 illustrate several road scenes and the detected lines. For the former the algorithm was implemented in Matlab and video frames had a resolution of 320x120 pixel. The latter shows results obtained from the implemented algorithm on the mobile phone. Note that the resolution of the images for this reason was 160x60 pixel.

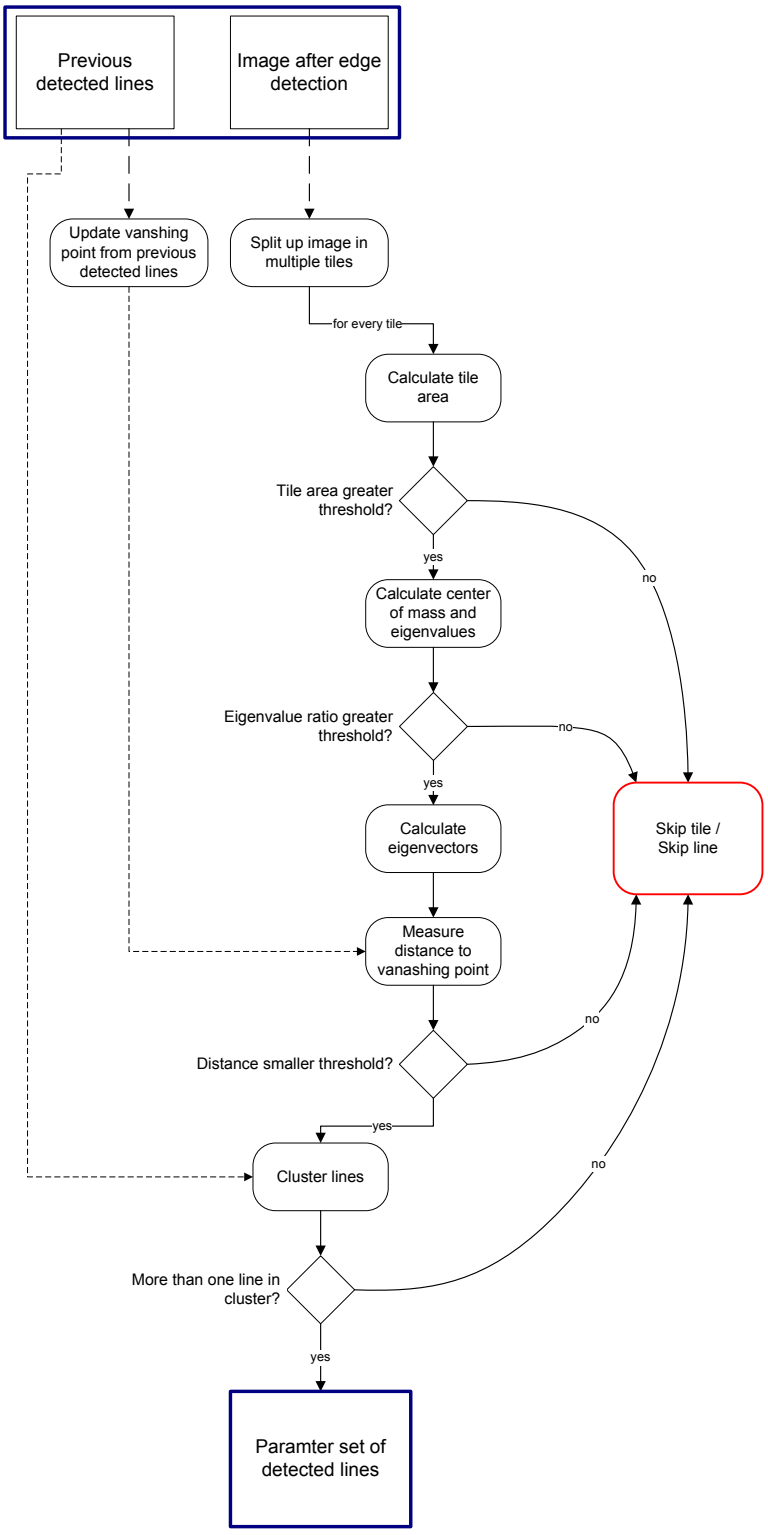


Figure 6.10: Sequence of operations

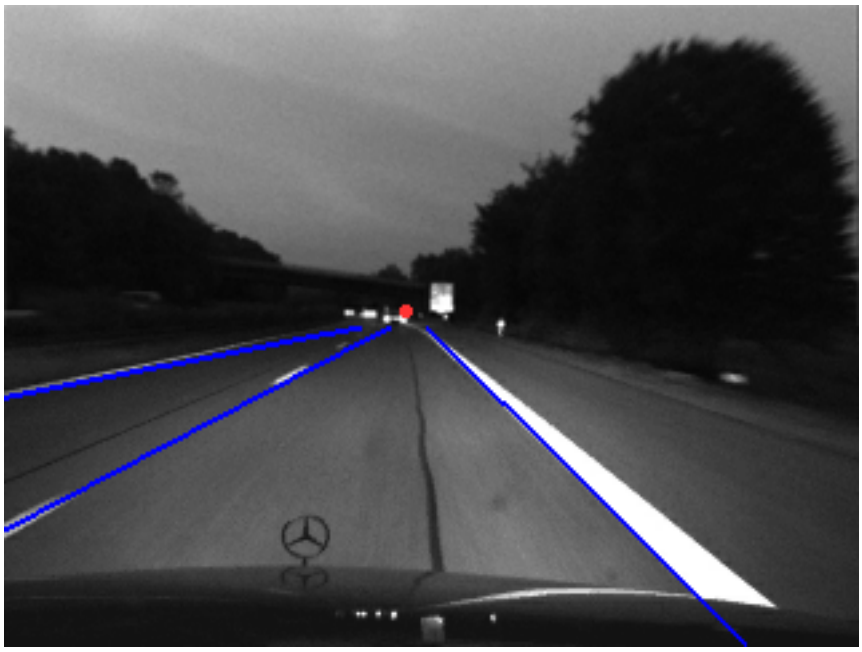


Figure 6.11: Detected lines in example image

Figure 6.12: Results for the implementation in Matlab with a resolution of 320x120 pixels.

Figure 6.13: Results for the implementation in C++ for the mobile phone with a resolution of 160x60 pixels.

Chapter 7

Implementation of Lane Detection

The implementation of the previously described lane marking detection algorithm has been split up into two iterations. For the development of the algorithm, the numerical computing environment Matlab [33] was used. This was advisable, as Matlab offers an appealing and easy to use development environment. Programming errors occur less often compared to an implementation in C++. As a result a faster development of the algorithm was possible combined with the opportunity to give more different approaches a change and to run a higher number of tests. In addition Matlab provides an image processing toolbox, which makes it convenient to test different edge filtering techniques.

Nevertheless the implementation in C++ is still important, because the goal was to use integer operations only. This is because the target platform on the mobile phone provides no floating point unit (FPU). Instead, such floating point operations are emulated witch the normal central processing unit (CPU), which increases the computation effort enormously. Therefore, the attention was to describe values as fixed point numbers in integer representation, when required.

An explanation of the different developed Matlab scripts can be found in appendix B.2. The implementation design on the mobile phone is discussed in the following and its output is presented.

7.1 Mobile Phone Implementation

The implementation in C++ addresses two requirements. Firstly, it has to run smoothly on the mobile phone. Secondly, it has to be designed in a generic way so that it can be used on any other platform supporting a C++ compiler. This is necessary, as a further goal is to implement a lane detection application on the next generation model of the embedded controller. This controller has an embedded Linux distribution as operating system. Therefore, the long-term aim is to port the computer vision library OpenCV to this platform. I have examined, if major functions of OpenCV can be ported to the mobile

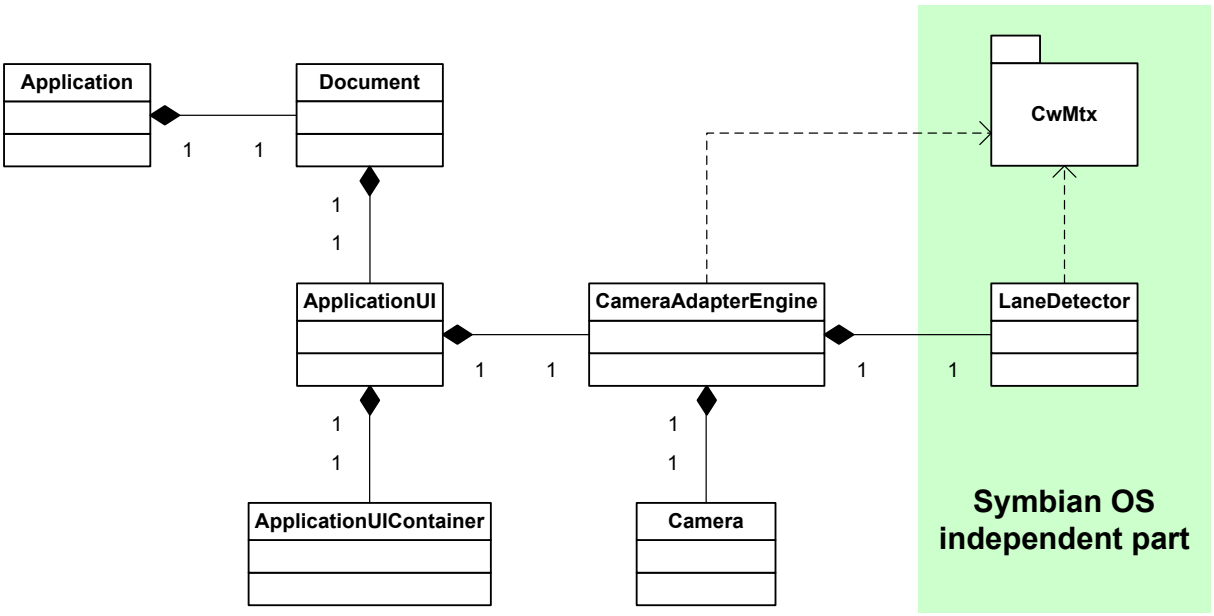


Figure 7.1: The application design with separation in platform dependent and platform independent parts.

phone. Because the library uses platform specific commands to obtain for example single clock ticks, it would be required to write a software layer between Symbian and OpenCV, which offer these interfaces. One solution could be to use the cross platform multimedia library SDL [30], where a Symbian port is in development. As this could be the task for another research project, I had to go for a different solution. After some investigations I decided to write the core functions for image processing myself, as no adequate library was available which would also run on Symbian.

Because the GUI programming for the mobile phone is very restricted and absolutely platform dependent, only the algorithm for detecting lanes has been implemented abstractly. For the representation of image and matrix calculations I used an open source library for matrix and vector math, called `CwMtx` [15]. The library provides classes for matrix and vector operations that are used extensively in engineering and science problems. Because the library itself is small and just the header files with the methods really used for calculations can be included and compiled, the program size keeps small as well. This is important as memory is rare on the mobile device. Moreover, the library does not make use of the C++ STL, which is absolutely essential, as Symbian does not provide an appropriate support. To address the restrictions of Symbian, a few methods using C++ input-output operations via `iostream` have been modified to use the standard C functions included in `stdio.h`.

To address the different goals of the implementation I chose the design illustrated in figure 7.1.

On the left side all the classes are restricted to Symbian. They implement the interface to the Symbian / Series 60 framework and provide methods for operating the mobile phone's

camera as well as for user interaction and a graphical output. On the right side is the independent class LaneDetector, which comprises the algorithm described so far.

To consider a fixed point representation of small values I multiplied them with a constant factor before computation and adjusted the result later on. For example the line parameters a and b as mentioned in 6.3 are obtained from the calculated eigenvectors, which are normalized to length 1. Therefore, both parameters lie in the range of -1 to 1, which is not suitable for my implementation. The internal representation has been adjusted that both parameters are 100 times greater leading to a normalized length of 100. With a strict consideration of scaling in the program code like the one mentioned, the result is the same as using floating point representation but with a strong speed-up on the device. For the calculation of eigenvalues the computation of a square root is required. Because the C library `math.h` implements this function with a `double` value, only here a floating point representation is used.
The interface between the class `LaneDetector` and an enclosing application is provided via the use of the `CwMtx` library. Thus for using the lane detector every image has to be converted in this format.

A characterization of the developed classes and their functionality is given in Appendix B.2. An even more detailed description can be found in the online documentation on the enclosed CD.

7.2 Output

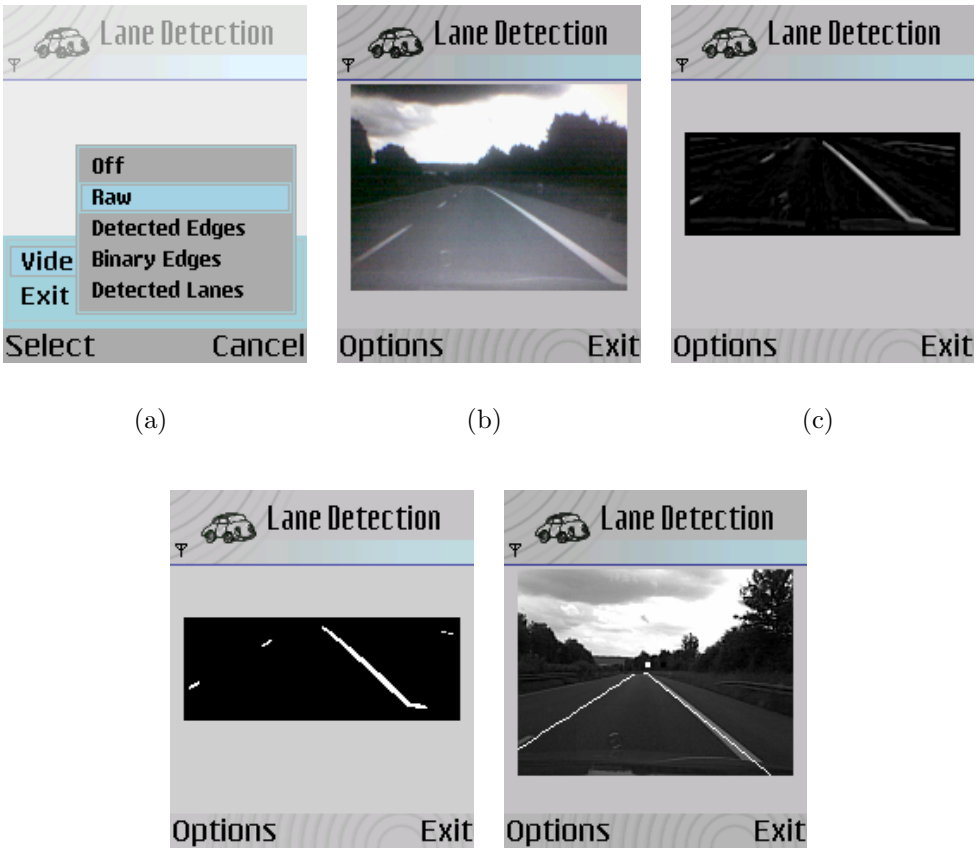
The mobile phone application for lane marking detection works quite satisfactory. However, there are some restrictions.

The implementation in Matlab operates with an input image size of 320x240 pixels. I have also tested the implementation on the mobile phone with this resolution, but had to figure out that the computational effort was too high. The final application works with half the resolution of 160x120 pixels. This means that only an image of size 160x60 pixels is used for detecting lane markings, because just the lower part contains interesting information about lane markings and is therefore used.

This results in a tile size of 10x10 pixels, which has to be sufficient for calculating the eigenvalues and eigenvectors. Compared to the implementation in Matlab, the result is less accurate. This is especially noticeable in the attempt to detect dashed lane markings. Because of the small size, eigenvectors may vary more and the distance for clustering lines belonging together is too large. An increase of the threshold for the distance calculation in the parameter space does not solve the problem, because then lines not belonging together would be clustered. This results in a worse output and is therefore not applicable.

In Figure 7.2 screenshots from the different possible views are displayed. Image (a) shows the applications menu, where the different working and display modes can be chosen. The following images (b) to (e) illustrate the different outputs on the display.

Nevertheless the result of lane detection on the mobile phone is remarkable. The application performed perfectly in a long time test without any errors or memory overflows. However, I found that the application uses quite a lot battery power. After one day of testing and developing the application, the battery was low again. This is because the back light of the phone was always turned on. On the other hand a mobile device normally remains in a suspended mode using almost no power and allows a standby time of about one week. As my application has a high computational effort for a mobile phone, the power consumption is automatically a lot higher. A professional usage of such a system in a car would, therefore, require additional external supply of electrical power.



(a) (b) (c)

(d) (e)

Figure 7.2: Screenshots from the application performing the lane marking detection algorithm.
(a) Applications menu
(b) Direct video input
(c) Detected edges in the road scene
(d) Binary image of the edge detected image
(e) Detected lane markings drawn in the original frame

Chapter 8

Conclusion

This thesis presents an approach for robot control and lane detection with mobile phones. An application to remotely control an EyeBot controller has been developed. The attention was put on a clear design and an abstract implementation of the connection specific parts. As a result, the application is reusable for other possible connections, e.g. a serial communication.
The chosen Bluetooth link performed well. It is possible to easily connect to devices in range and control them via the mobile phones buttons. The application was designed to operate with several remote controllers at a time.
However, there appeared some limitations. The Nokia 6260 mobile phone is not capable of establishing more than one Bluetooth link with the available API. Problems occurred with receiving a continuous large stream of data. I figured out that the mobile phone was not fast enough to read out the connection buffer. Furthermore, displaying text and graphics at the phone's screen implicates a high computational effort. As a result, individual images sent from the EyeBot to the mobile phone can be handled, but not a continuous image stream.

Developing applications for Symbian OS and the Nokia S60 platform is a challenging task. Symbian is an open platform for the mass-market covering a variety of different mobile phones. With the consequence of several limitations and restrictions concerning program design. But with proper use of the coding conventions and APIs, Symbian offers a lot possibilities for further development and the mobile phone can be used as an embedded system alternative.
For small projects a development in C++ for Symbian OS is not recommended in my opinion. However, there is an open source project with the intention to provide a Phyton interpreter for the Nokia S60 platform [9]. Support for Bluetooth and a camera API are already included so far. This is promising for the future, as it tends to enable rapid application development and the ability to create stand-alone S60 applications written in Python.

This thesis evaluated several approaches for lane marking detection and investigated an efficient, low computational algorithm, which was finally implemented on the mobile phone.

To take the specific orientation of lane markings in a recorded image into consideration, an edge detection based on steerable filters was investigated. Only the inner side of lane markings was detected to prevent a costly clustering of detected lines belonging together. The finally chosen separable steerable filter was based on the first Gaussian derivative. As expected the output was more accurate compared to the simple Sobel filter, however it required about 30% more calculation time.
For the extraction of line parameters the edge image was separated into several tiles. A separation in 16×6 tiles performed best. Each tile was evaluated in terms of its variance by performing an principal component analysis of the covariance matrix.
To discard outliers not representing a line in reality, four conditions were defined which had to be met. Hence, the image moment and eigenvalue ratio had to lie above investigated thresholds. The distance of the detected lines to the vanishing point and the number of possible clustered lines were evaluated to reject false lines.
A dynamic vanishing point update was performed by a least square optimization. It was shown that the included inversion was just applied to a 2×2 matrix, thus not leading to an increase in computational effort.

The lane detection algorithm was implemented independent from the Symbian platform using standard C++ and an additional matrix library to run on any platform. All computations were performed with a fixed point integer representation. This was advisable, as embedded devices usually do not include a floating point unit.

The algorithm works reliably to detect solid lane markings. Dashed lane markings are detected most of the time. Possibilities for further improvements of the detection rate could be considered.
Optimization of edge filtering would automatically lead to better results. An adaptive threshold combined with non-maxima suppression would increase the accuracy of the binary image. As a result the orientation of eigenvectors would exactly match the lane marking direction.
Enhancemet of the detection of dashed lines could be achieved by involvement of already detected lines over a longer period of time in the algorithm. With the additional information an estimation of possible lines can be stated, even if there is no line detected in the current iteration.
Of course all these adaptations lead to a higher computational effort. Indeed, the mobile phone application already faces the restriction that the image size was set to 160x60 pixels to run smoothly, compared to twice the size of 320x120 on the PC. The processing speed may be improved by using a smaller filter size for edge detection or by using an optimized image processing or matrix library, rather than using hand-written code. Finally newer mobile phones will offer a more powerful processor.
In any case a tradeoff between accuracy and processing speed has to be made to find the optimal performance.

Appendix A

Mathematical Background

A.1 Edge Detection

A problem encountered is the definition what an edge in an image really is. The usual approach, which is also used in this paper is to define edges by means of discontinuities in the image signal, which is similar to a steep intensity gradient. The method used for detecting this discontinuities often becomes one of finding local maxima in the first derivative of the image signal, or zero-crossing in the second derivative as shown in figure A.1.

To calculate the intensity gradient of an image with intensity function $f(x,\, y)$, the first derivative in both directions has to be calculated. Thus the gradient is a vector composed of partial derivatives

$$\nabla f(x,\, y) = \left(\frac{\partial f}{\partial x},\, \frac{\partial f}{\partial y}\right) = (f_x,\, f_y)\,.$$

The gradient vector points in the direction of most rapid change in intensity and its direction is given by

$$\Theta(x,\, y) = \arctan\left(\frac{f_y}{f_x}\right). \tag{A.1}$$

Related to edge detection in an image, the edge strength is given by the gradient magnitude

$$M(x,\, y) = \|\nabla f(x,\, y)\| = \sqrt{f_x^2 + f_y^2}. \tag{A.2}$$

For the second derivative, the Laplace operator is used on the image. It is a scalar composed of second order partial derivatives

$$\nabla^2 f(x,\, y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}.$$

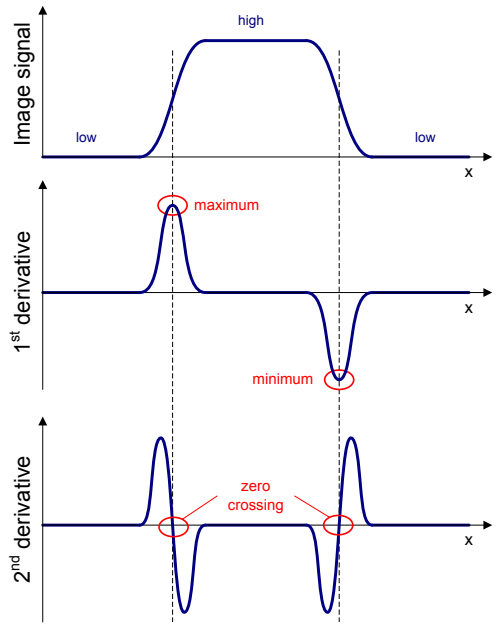


Figure A.1: First and second derivative of one dimensional image signal (slice through image)

In discrete images, partial derivatives are approximated by finite differences

$$\frac{\partial f}{\partial x}[x,\, y] \approx f[x+1,\, y] - f[x,\, y]\,.$$

Thus it is obvious to calculate the gradient by simply taking the difference of intensity values between adjacent pixels. The easiest way to this is to convolute the image with a discrete filter, like $[-1,\, 1]$. Note, that with this convolution mask it is not clear, where to save the result, because it is not possible to place it between the pixels. To overcome this problem, it is preferable to use a filter mask with an odd number of entries, hence the result can be assigned to the middle pixel.

By approximating the second derivative via

$$\begin{aligned}\frac{\partial^2 f}{\partial x^2}[x,\, y] &= \frac{\partial f}{\partial x}[x+1,\, y] - \frac{\partial f}{\partial x}[x,\, y] \\ &\approx f[x+2,\, y] - 2f[x+1,\, y] + f[x,\, y]\,,\end{aligned}$$

an odd filter mask like $[1,\, -2,\, 1]$ is calculated automatically.

An example convolution mask for the approximated gradient in x direction would be the *Prewitt* operator

-1	0	1
-1	0	1
-1	0	1

It is the simplest and fastest operator. Its mask for the y direction is obtained by a 90° rotation of the given one. Note that for every operator the sum over all entries has to be zero, since it should not react to flat regions in the image.

Once the derivative is calculated the next stage is to apply a threshold to determine where the result suggests and edge to be present and saving the result as a binary image. Setting the threshold at a low level allows to detect more lines, but the outcome gets increasingly sensitive to noise. On the other hand setting the threshold at a high level may result in missing important edges or in segmented lines. Thus choosing the right threshold is an important step to get a suitable image representing the detected edges.

A.2 Steerable Filters

An edge located at different orientations in an image can be detected by splitting the image into orientation sub-bands obtained by basis filters. The final filter, generally donated as two dimensional function $f(x, y)$ can then be written as linear sum of rotated versions of itself. Hence $f(x, y)$ is said to be steerable if it can be expressed at an arbitrary rotation θ with

$$f^{\theta}(x, y) = \sum_{j=1}^{M} k_j(\theta) \cdot f^{\theta_j}(x, y). \tag{A.3}$$

$f^{\theta_j}(x, y)$ is a rotated version of $f(x, y)$ at angle θ_j and $k_j(\theta)$ for $1 \leq j \leq M$ are the interpolation functions as well dependent from θ. In the following analysis it will be shown what functions $f(x, y)$ can satisfy (A.3), how many terms M are needed and what the interpolation functions $k_j(\theta)$ are.

In polar coordinates with $r = \sqrt{x^2 + y^2}$ and $\phi = \arg(x, y)$ for function f can be expanded into the Fourier series

$$f(r, \phi) = \sum_{n=-N}^{N} a_n(r) \cdot e^{\imath n\phi}. \tag{A.4}$$

Substituting terms in equation (A.3) by their Fourier decomposition according to (A.4) results in the relation

$$\sum_{n=-N}^{N} a_n(r) \cdot e^{\imath n\theta} = \sum_{j=1}^{M} k_j(\theta) \cdot \sum_{n=-N}^{N} a_n(r) \cdot e^{\imath n\theta_j} \tag{A.5}$$

The exponent is not any more dependent on ϕ, but on the steering and basis angles θ and θ_j, because both sides have been projected onto the complex exponential $e^{\imath n\phi}$. Extracting the sum over n on the right side of (A.5) leads to

$$a_n(r) \cdot e^{\imath n\theta} = \sum_{j=1}^{M} k_j(\theta) \cdot a_n(r) \cdot e^{\imath n\theta_j}, -N \leq n \leq N.$$

If $a_n(r) \neq 0$ both sides are divided by $a_n(r)$, otherwise the constraint for that specific n is removed for the set. The equations for $-n$ and n are redundant, therefore it is possible

to consider just positive frequencies int the range of $0 \leq n \leq N$. The resulting

$$e^{\imath n\theta} = \sum_{j=1}^{M} k_j(\theta) \cdot e^{\imath n\theta_j}, 0 \leq n \leq N \, and \, a_n(r) \neq 0$$

can be written is matrix vector notation like

$$\begin{pmatrix} 1 \\ e^{\imath\theta} \\ \vdots \\ e^{\imath N\theta} \end{pmatrix} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ e^{\imath\theta_1} & e^{\imath\theta_2} & \cdots & e^{\imath\theta_M} \\ \vdots & \vdots & \ddots & \vdots \\ e^{\imath N\theta_1} & e^{\imath N\theta_2} & \cdots & e^{\imath N\theta_M} \end{pmatrix} \cdot \begin{pmatrix} k_1(\theta) \\ k_2(\theta) \\ \vdots \\ k_M(\theta) \end{pmatrix}. \tag{A.6}$$

Theorem: The steering condition (A.3) holds for functions expandable as Fourier series if and only if there exists a set of interpolation functions $k_j(\theta)$ satisfying the matrix equation (A.6).

As mentioned before, if for any $a_n(r) = 0$ with $-N \leq n \leq N$, then the corresponding n^{th} row will be removed. Therefore the number T of nonzero Fourier coefficients $a_n(r)$ gives the minimum number of basis functions required for steering condition. That is that $M \geq T$ in (A.6). A detailed derivation of this conclusion id given in [10].

The resulting T basis function orientations θ_j must be chosen in order to provide that the columns of matrix in equation (A.6) are linearly independent. To achieve this requirement, it is sufficient to take angles between 0 and π. Moreover for reasons of symmetry and robustness against noise they should be spaced equally in the domain. All functions, which are bandlimited in angular frequency are steerable, hence a finite Fourier series can be expressed. In practice the most useful functions are those which require a small number of basis filters.

Appendix B

Implementation Description

B.1 Communication Implementation

User interface specific classes

Class name	*Description*
`RemoteBTApp`	This is the application class, predetermined by the S60 application design. It creates the applications document.
`RemoteBTDocument`	Describes the applications document. The class is also predetermined by the S60 application structure for GUI applications. Its purpose is to create the application UI.
`RemotBTAppUI`	The class for the application user interface provides support for user input handling, the Symbian own control architecture and for view management. Moreover, `RemotBTAppUI`encloses a connection to the socket server. This is necessary to be able to create further connections to a remote device via a socket architecture. It also creates the `MainView` and a `RemoteSessionList`, where all currently active `RemoteSessions` are stored.
`MainView`	`MainView` represents the start screen and handles user inputs as well as activation and deactivation of this view. It creates the `MainViewContainer` to enable updates of the view.
`MainViewContainer`	`MainViewContainer` includes all component controls of the main view and updates them on changes. As a result it handles occurring control events. A session can be created or closed by selecting the corresponding list box entry, the class provides functions for `RemoteSession` construction and deletion. However, if a remote session is instantiated successfully the ownership is transferred to the `RemoteSessionList` of the application engine.

Class name	*Description*
`RemoteView`	The `RemoteView` simulates the EyeBot controller LCD. It displays exactly the same data. For this reason it includes functions which execute the commands extracted from the received messages. To be able to also update data in the background, if the view is not active the class contains a class Display, which stores the content of the display.
`RemoteViewContainer`	`RemoteViewContainer`updates the `RemoteView` on changes. It reads data to show from `Display` and writes them to the remote view. For this reason there are several functions to update text and graphics independent and only the parts, which have really been changed. Further, it converts positions in the coordinate system of the EyeBot LCD into the coordinate system on the mobile phone's screen.

Engine specific classes

Class name	*Description*
`RemoteSession`	`RemoteSession`is the main class for every new created session to a EyeBot controller. It includes the finite state machine mentioned in 4.1.4.
`RemoteSessionList`	`RemoteSessionList`subsumes all currently active `RemoteSession` objects and manages them. It offers methods for adding and deleting sessions. It is also an active object to enable a `RemoteSession` to delete itself. For this purpose a session can call `RemoteSessionList::DeleteSession()` and the session will be marked to be deleted next. With the next call of `RunL()` of the active object `RemoteSessionList` the session will be finally destroyed, thus allowing the `RemoteSession` to go into a save state before destruction.
`Message`	`Message`represents the message protocol structure, which is used for communication. It offers methods for creating and modifying a message.
`Parser`	The `Parser` parses the received message and executes the appropriate extracted command immediately. If it looses synchronization the beginning of the next start message signalized via the unique start bit is awaited. Hence data from the current erroneous received message is lost.
`Display`	`Display`stores the current content of the EyeBot's LCD to write it to the mobile phones screen. It is also updated if the according remote session view is not in focus. As a result the view is always up to date, when in focus. Directly writing to the mobile phones screen would not make this feature possible.
`Soundplayer`	`Soundplayer`enables to play-back a defined sound file on the mobile phone.

Connection specific classes

Class name	*Description*
`ConnectionEngine`	`ConnectionEngine`is an abstract calls an defines all the necessary functions to operate with a connection to a remote controller. These include establishing the connection, sending and receiving data an closing the connection.
`BTConnection`	This class is inherited from `ConnectionEngine`. Hence it implements all the virtual functions for the specific Bluetooth link.
`BTDiscoverer`	`BTDiscoverer`is called from `BTDiscoverer` and offers functions for device inquiry, device selection and a possible check for available Bluetooth services at the remote controller. As all EyeBot controller offer the needed serial connection service for a remote controller, this functionality is implemented exemplary only.
`BTDeviceParam`	Class to store device specific parameter, as device address, name and port number. The major device and service class according to the Bluetooth standard are also included.

B.2 Lane Detection Implementation

Matlab Files

This section discusses the different developed Matlab scripts to get the lane marking detection running. The files can be found on the enclosed CD in the directory `\matlab`. In the following is the purpose of the main files explained, which have been created

File name	*Description*
extract_lines.m	`function [param, vanishingpoint] = extract_lines(binaryImg, oldparam, oldvp)` This function contains the line detection and clustering algorithm as previously explained. Input parameters are the binary image with the detected edges, the parameter set of the previously detected lines and the corresponding previous vanishing point. Return values are the set of currently detected lines and the undated vanishing point used for the calculation.
lane_detection.m	`function detectLanes()` This is the main Matlab file. It defines the video where the lane markings should be extracted. Further on all necessary initialization for the edge and line detection steps are done. For a better comparison between the result of the first and second Gaussian derivative steerable filter, both tyes can be tested and. Note that for extracting lines, only the 1^{st}derivative makes sense, because with the 2^{nd} derivative filter both inner and outer lane marking boundaries are detected. The edge filtering with a Sobel filter is as well implemented to allow a further comparison.
steerable_filter_basis.m	This script generates and displays the basis filters for the first and second Gaussian derivative.
steerable_1st_derivative.m	The script includes the generation of the filter coefficients for the separable filter as well as several illustrations and an animation of so the gained filter.
display_all_oriented_filters.m	The script finally displays the different oriented steerable filters used for the edge detection.

Mobile Phone Implementation Classes

Class name	*Description*
`CLaneDetectionApp`	This class is compulsory for a Symbian GUI application and comprises the application entry point. It creates the CLaneDetectionDocument object.
`CLaneDetectionDocument`	This class is compulsory for a Symbian GUI application. It represents the model of the Model-View-Controller concept and creates the application UI on start up.
`CLaneDetectionAppUi`	This class is compulsory for a Symbian GUI application. It is the view in the MVC concept and constructs the application container as well as the camera adapter engine. Moreover the application user interface is responsible for handling user inputs as keystrokes and menu selections.
`CLaneDetectionContainer`	This class is compulsory for a Symbian GUI application and implements the controller of the MVC concept. It is appropriate for the dynamic handling of components on the view.
`MCaemraListener`	The camera listener is an abstract class for the communication between `Camera` and `CCameraAdapterEngine`. It specifies only one method `FrameReady()`, which is called by `Camera`, when a recorded video frame is ready for further processing.
`CCameraAdapterEngine`	`CCameraAdapterEngine` is inherited from `MCaemraListener`. Hence the adapter engine provides an interface to the Camera object. It implements the abstract virtual method `FrameReady()` and thus handles the further processing of the recorded image. For this reason the class has an object of `LaneDetector` as member, which is called to perform the lane marking extraction. In combination to this it handles the display of the video frame on the screen as well as . To be able to address the mobile phones camera a `Camera` object is created on instantiation. Further, a number of convenient functions are offered to control the `Camera` object and hence the mobile phone's camera. Moreover, the class provides methods for manipulating the screen display. This is used to show the recorded image directly or with detected edges, as binary image and with found lane markings. Finally the class includes two methods for increasing and decreasing the image brightness. This is helpful to generate a balanced recording, where the following edge detection can work on.

Class name	*Description*
`Camera`	This class provides an direct access to the mobile phones camera and therefore inherits from `MCameraOvserver`, a Symbian intern abstract class with methods that are called in combination with asynchronous request to the phones camera. For example the class provides methods, which specify the behavior of the application in situations like power on, image ready and reservation and release of the camera. The Symbian provided class `CCamera` finally is instantiated to take control over the hardware camera.
`LaneDetector`	This class contains the whole power of the lane detection algorithm. It just offers one public function to start the algorithm. Internally there are several functions called iteratively to perform and edge detection with steerable filter, to generate a binary image, to find lines in the binary image, to cluster found lines and finally to draw the found lines back into the image frame. The class makes extensive use of the `CwMtx` library. I have implemented the functions in a way that most matrices needed for computations are members of the class and therefore just created on instantiation of the object. This result in an optimized memory allocation, because most variables are reused.

Appendix C

Project CD

Bibliography

[1] Bellino, M., Y.L. de Meneses, P. Ryser and J. Jacot: *Lane detection algorithm for an onboard camera*. Proc. SPIE, 5663:102–111, 2004. 7, 8, 9

[2] Bräunl, Thomas.: *EyeBot: a family of autonomous mobile robots*. Neural Information Processing, 1999. Proceedings. ICONIP'99. 6th International Conference on, 2, 1999. 5

[3] DaimlerChrysler: *Innovations in the commercial vehicle - moving down the road towards the fail-safe truck*. Website. http://www.daimlerchrysler.com/dccom/0-5-75472-1-77526-1-0-0-0-0-0-36-7165-0-0-0-0-0-0-0.html. 5, 6

[4] Dellaert, F., D. Pomerlau and C. Thorpe: *Model-based car tracking integrated with a road-follower*. Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on, 3, 1998. 7

[5] Dickmanns, ED and BD Mysliwetz: *Recursive 3-D road and relative ego-state recognition*. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 14(2):199–213, 1992. 7

[6] Forsyth, D.A. and J. Ponce: *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference, 2002. 9, 10

[7] Forum Nokia: *Symbian Developer Library Documentation*, 2006. http://www.symbian.com/Developer/techlib/v70sdocs/doc_source. iii, 14, 22

[8] Forum Nokia, Nokia Corporation: *Nokia 6260 Mobile Phone*. Website. http://forum.nokia.com/devices/6260. 4

[9] Forum Nokia, Nokia Corporation: *Python for S60*. Website. http://opensource.nokia.com/projects/pythonfors60/. 75

[10] Freeman, W.T. and E.H. Adelson: *The design and use of steerable filters*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 13(9):891–906, 1991. 38, 41, 42, 80

[11] Harrison, R. and P. Northam: *Symbian OS C++ for Mobile Phones*. John Wiley & Sons, Inc. New York, NY, USA, 2003. 15

[12] Kang, D.J. and M.H. Jung: *Road lane segmentation using dynamic programming for active safety vehicles*. Pattern Recognition Letters, 24(16):3177–3185, 2003. 10

[13] Kaske, A., D. Wolf and R. Husson: *Lane boundary detection using statistical criteria*. International Conference on Quality by Artificial Vision, QCAV9, pages 28–30, 1997. 7, 9

[14] Kreucher, C. and S. Lakshmanan: *LANA: a lane extraction algorithm that uses frequency domainfeatures*. Robotics and Automation, IEEE Transactions on, 15(2):343–350, 1999. 7, 9

[15] Kuiper, Harry: *The CwMtx library for matrix, vector and quaternion math*. Website. http://www.xs4all.nl/~hkuiper/cwmtx/cwmtx.html. 70

[16] McCall, JC and MM Trivedi: *An integrated, robust approach to lane marking detection and lane tracking*. Intelligent Vehicles Symposium, 2004 IEEE, pages 533–537, 2004. 7, 9, 38, 39, 41

[17] MCCALL, J.C. and M.M. TRIVEDI: *Video-based lane estimation and tracking for driver assistance: Survey, system, and evaluation*. IEEE Transactions on intelligent transportation systems, 7(1):20–37, 2006. iii, 7, 9, 38, 39

[18] McCall, J.C., D. Wipf, M.M. Trivedi and B. Rao: *Lane Change Intent Analysis Using Robust Operators and Sparse Bayesian Learning*. To Appear: IEEE International Workshop on Machine Vision for Intelligent Vehicles in Conjunction with IEEE International Conference on Computer Vision and Pattern Recognition, 2005. 7, 9, 38, 39

[19] Mc Donald, J.B., J. Franz and R. Shorten: *Application of the Hough Transform to Lane Detection in Motorway Driving Scenarios*. Proc. of the Irish Signals and Systems Conference, 2001. 7, 9

[20] Nokia Corporation: *Nokia Connectivity Framework 1.2*. Website. http://forum.nokia.com/info/sw.nokia.com/id/8b7443ac-66ee-4d47-9f25-1da664b23c9c/Nokia_Connectivity_Framework_1_2.html. 18

[21] Nokia Corporation: *Symbian OS: Acive Objects And The Active Scheduler*. 2004. 18

[22] Nokia Corporation: *Symbian OS: Application Framework Handbook*. 2005. iii, 15

[23] Nokia Corporation: *Symbian OS: Desinging Bluetooth Applications in C++*. 2005. iii, 21

[24] Nokia Corporation: *Symbian OS: Threads Programming*. 2005. 18

[25] Nokia Corporation: *Symbian OS Basics Workbook*. 2.0, 2006. iii, 14

[26] Otsu, N.: *A threshold selection method from gray level*. IEEE Transactions on Systems, Man, and Cybernetics, 9(1):62–66, 1979. 48

[27] Petitt, J.D. and T. Bräunl: *A Framework for Cognitive Agents*. International Journal of Control, Automation, and Systems, 1(1):229–235, 2003. 5

[28] Robotics & Automation Lab, The University of Western Australia: *The EyeBot Controller*. Website. http://robotics.ee.uwa.edu.au/eyebot. 3

[29] Schmitz, C. and D.I.E. FH: *Wireless Networks for Mobile Robots*. 2005. 5

[30] SDL project: *Simple Directmedia Layer*. Website. http://www.libsdl.org. 70

[31] Siemens: *Der Blick ins Gehirn des Fahrers*, 2002. iii, 6

[32] Taylor, CJ, J. Malik and J. Weber: *A real-time approach to stereopsis and lane-finding*. Intelligent Vehicles Symposium, 1996., Proceedings of the 1996 IEEE, pages 207–212, 1996. 7, 9

[33] The MathWorks: *Matlab product webpage*. Website. http://www.mathworks.com/products/matlab/. 69

[34] The University of Western Australia: *Robotics & Automation Lab*. Website. http://robotics.ee.uwa.edu.au. 1

[35] Wang, Y., D. Shen and E.K. Teoh: *Lane detection using spline model*. Pattern Recognition Letters, 21(8):677–689, 2000. 7, 8

[36] Wang, Y., E.K. Teoh and D. Shen: *Lane detection and tracking using B-Snake*. Image and Vision Computing, 22(4):269–280, 2004. 7, 8

[37] Wilke, P.: *Flexible wireless communication network for mobile robot agents Peter Wilke, Thomas Bräunl The Authors*. Industrial Robot: An International Journal, 28(3):220–232, 2001. 5

[38] YIM, Y.U. and S.Y. OH: *Three-feature based automatic lane detection algorithm(TFALDA) for autonomous driving*. IEEE Transactions on intelligent transportation systems, 4(4):219–225, 2003. 7, 9