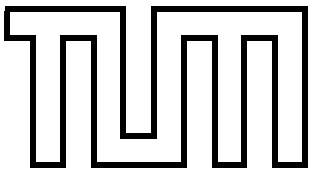
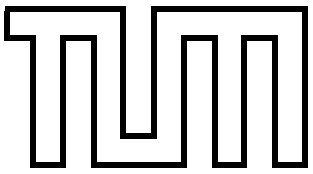


FAKULTÄT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's thesis in Computer Science

Development and Construction of an Autonomous Car with Low Cost Components

Roger Rösch



FAKULTÄT FÜR INFORMATIK

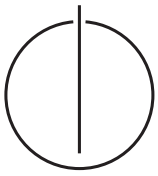
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's thesis in Computer Science

Development and Construction of an Autonomous Car with Low Cost Components

Entwicklung und Konstruktion eines autonomen Fahrzeugs mit kostengünstigen Komponenten

Author:	Roger Rösch
Supervisor:	Prof. Dr.-Ing. habil. Alois Knoll
Advisors:	Dr.sc.nat. Kai Huang
	Biao Hu
Date:	May 17, 2016



I assure the single-handed composition of this master's thesis only supported by declared resources.

Alling, May 17, 2016 Roger Rösch

Abstract

In recent years development boards with processors based on the ARM architecture, which are as powerful as personal computers were less than a decade ago, have been released. This enables building autonomous vehicles with very low cost. In this thesis a project to convert a remote controlled car into an autonomous one is presented. The car has been equipped with processors, webcams and ultrasonic sensors to allow autonomous driving. The goal was to enable the car to drive on an empty road towards a target location and to follow instructions of road signs and traffic lights on the course. Different sensors, platforms and algorithms which could have been used are discussed. Limitations of low cost hardware are examined and possible solutions presented.

Zusammenfassung

In den letzten Jahren wurden Entwicklungsboards mit Prozessoren basierend auf der ARM Architektur auf den Markt gebracht, die ähnlich leistungsstark sind wie Heimrechner vor einem Jahrzehnt. Dies ermöglicht es, autonome Fahrzeuge sehr kostengünstig zu bauen. In dieser Arbeit wird ein Projekt beschrieben, das zum Ziel hat, ein ferngesteuertes Auto in ein autonomes umzuwandeln. Das Fahrzeug wurde mit Prozessoren, Webcams und Untraschallsensoren ausgerüstet, um autonomes Fahren zu ermöglichen. Das Ziel war, das Fahrzeug entlang leerer Straßen zu einem Zielort fahren, sowie unterwegs Verkehrzeichen und Ampelsignale korrekt befolgen zu lassen. Verschiedene weitere Sensoren, Plattformen und Algorithmen, deren Nutzung möglich gewesen wäre, werden besprochen. Die Einschränkungen kostengünstiger Hardware und mögliche Lösungskonzepte hierzu werden präsentiert.

Acknowledgements

I would like to thank Kai Huang and Biao Hu who allowed me to work on such an interesting topic and were always ready to answer my questions and discuss suggestions while allowing me to choose my own focus for this thesis. Further thanks goes to the students who participated in the WS 2015/16 course "Lab course: Hardware/Software Co-Design with a LEGO car" and decided to support the autonomous car project. Especially Christoph Griesbeck, Christoph Winter, Ioannis Stratakos and Laurie Marceau who always took part and continuously improved their code. Valentin Novak, participating via the Erasmus Programme, provided great help, gave valuable suggestions, presented solutions for hardware problems and gave me the opportunity to always talk about the state of the project. Without them the project would be in a far earlier state. I am especially indebted to my parents who contributed by letting us utilize the resources of their company and by supporting me to the best of their abilities. I appreciated the help of my proof readers, my brother Richard Rösch, Mia Klee and others already named here, who occupied me for days with their suggestions. Special thanks goes to Ellen Papelitzky who always managed to bring a smile to my face, even when going through her proof reading remarks. Finally I would like to express my gratitude towards the developers of PureBasic and the various open source projects which supported me during the creation of this thesis.

Contents

Abbreviations

ARGO	Autonomous car project of University of Parma
ARM	Advanced RISC Maschine, processor architecture
CPU	Central Processing Unit
CSI	Camera Serial Interface
ECU	Electronic Control Unit
eMMC	Embedded MultiMediaCard, computer storage technology
FPGA	Field-Programmable Gate Array, a type of integrated circuit
fps	Frames Per Second
GIMP	GNU Image Manipulation Program
GPIO	General-Purpose Input/Output (pins)
GPS	Global Positioning System
H.264	MPEG-4 Part 10, Advanced Video Coding, video codec
HSV	Hue Saturation Value colorspace
I^2C	Inter-Integrated Circuit, bus protocol
IMU	Inertial Measurement Unit
KB	KiloByte
MB	MegaByte
MJPEG	Motion JPEG, video codec
OCR	Optical Character Recognition
OSRM	Open Source Routing Machine
PID	Proportional-Integral-Derivative (controller)
PWM	Pulse Width Modulation, method for encoding information
RAM	Random-Access Memory, fast and volatile memory
RGB	Red Green Blue colorspace
ROI	Region Of Interest
RRT	Rapidly exploring Random Trees, path finding algorithm
SD	Secure Digital, non-volatile memory card standard
SLAM	Simultaneous Localization and Mapping
USB	Universal Serial Bus, interface standard

1 Introduction

1.1 Motivation

The idea to build autonomous cars has a long history. Already in 1930 Miles J. Breuer published a story named "Paradise and Iron" [1][2] where he mentions several autonomous operating vehicles, among them a self driving car. Some even attribute the idea to create an autonomous car to Leonardo da Vinci (1452-1519), who had sketched a car made with springs which can be wound up and programmed to drive along a certain path [3]. So it is not surprising that research on this topic started as soon as basic resources were available. An example for an early autonomously operating car is the result of the ARGO Project initiated by Professor Alberto Broggi. A common car was equipped with two cameras, electrical motors to control the steering wheel and a personal computer to control the car. In 1998 it drove 2000 kilometers along the Italian highways and operated fully autonomously for over 90% of the tour [4]. The computer for this system only had 32 megabytes of RAM and a 200-MHz Pentium Pro processor. In recent years ARM development boards with a much higher performance have become available, one of the most well known ones is the Raspberry Pi. Finding comparable benchmarks between these CPUs is difficult, but tests by the popular benchmark site Phoronix show that the second revision of the Raspberry Pi can even compete with a processor from 2004 [5]. While the cameras used by Alberto Broggi were priced at 100 dollar each and the processor of the computer alone costs several hundred dollars [6], the Raspberry Pi costs a fraction of it and cheap webcams can be bought for prices as low as a few euro. So today it should be possible to equip a vehicle with low cost components and convert it into an autonomous one. This opens up many possibilities for use: Search and rescue operations, toys, self built home service robots and warehouse automation are examples where affordable and therefore also replaceable autonomous vehicles are of interest.

1.2 Structure of this Thesis

This document is separated in the following parts: In section 1 "Introduction" the motivation for this thesis is given and necessary definitions are made. Section 2 "Project" gives an overview of the current state of a project where such a low cost solution is attempted. The main focus in section 3 "Alternative Algorithms, Sensors and Platforms" is on alternative solutions and possible improvements to methods used. This was chosen as central theme as there are many possible alterations to our initial design that came up during construction and implementation, but could not be implemented due to time constraints. Finally at the end of section 3 lessons learned are mentioned and recommendations for further work on the project and similar projects are presented.

In the appendix detailed information about setting up different components is retained to help people who have to work with similar tools.

1.3 Definition of Low Cost

Low cost components is a vague term. An automated 3D scanner may be labeled as low cost at 3.500 dollars when compared to models that cost more than 240.000 dollars [7]. In this thesis the aim is to use individual components that cost less then 100 euro, while lower prices are preferred. Cheap components often come with significant disadvantages compared to their higher priced counterparts. Where this is the case, these disadvantages will be highlighted.

1.4 Prices

As the focus of this thesis are low cost components, prices of sensors and other parts will be mentioned. To obtain these prices multiple websites are compared against each other. The cheapest price found will be used, rounded up to the next multiple of 5 euro. Components which cost less then 5 euro will be rounded to the next multiple of 50 euro cents. The websites queried for prices are ebay.com, amazon.com and aliexpress.com. Prices which require the purchase of high volumes of components were discarded. These choices were made to assess the cost for use in small scale projects and because these websites offer a broad choice of available equipment.

2 Project

2.1 Project Participants and Schedule

The project was started in September 2015 under supervision of Kai Huang and Biao Hu. In October Valentin Novak joined the team. He focused mainly on the FPGA and construction of attachments for the car. Later on he started working on localization. In mid October several students from the "Lab course: Hardware/Software Co-Design with a LEGO car" joined the team. They worked on different subtasks. Students who contributed significantly to the project and their respective subtasks are: Christoph Griesbeck and Christoph Winter, working on traffic sign detection, Ioannis Stratakos, working on traffic light detection, Laurie Marceau, working on stereo vision and Fabian Schurig, working on lane detection. Starting with the summer semester 2016 the project was handed over to Lu Cheng and a new group of students in the summer semester lab course.

2.2 Project Goals

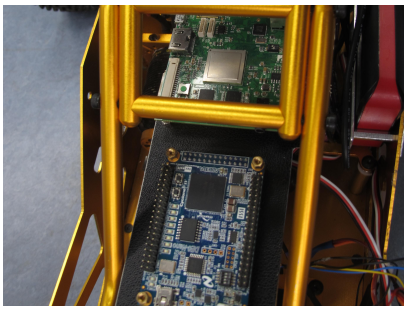
After students of the lab course joined the project, several goals were selected.

- Use of low cost components
- Localization and navigation
- Lane recognition
- Compliance with traffic signs and traffic lights
- Obstacle avoidance

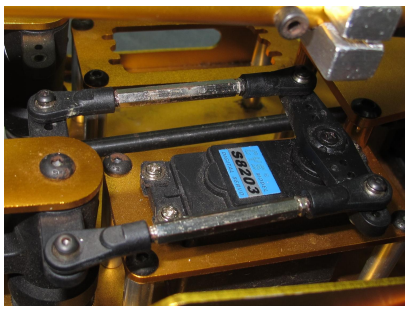
These goals were chosen ambitious deliberately, because it was known that in the following semester other students would continue the project.

2.3 Equipment Provided

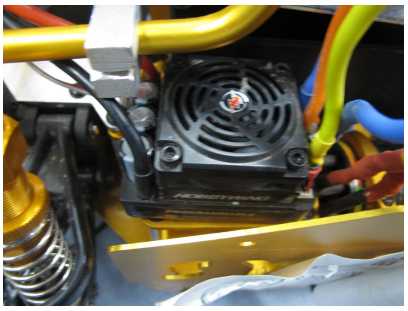
Kai Huang provided a remote controlled car of the type SY-BLS5 equipped with additional metal bars on top. The car came with a remote control and no manual. The car is driven by a brushless motor, controlled by an EZRUN-150A-PRO brushless speed controller. The steering is controlled by two servo motors of type CYS S8203. The team was supplied with an FPGA DE0-NANO board from Terasic and a quadcore i.MX6 development board named SABRE Lite from Boundary Devices. Also provided were two ultrasonic sensors, type KS103 from Kingsin Technologies, for obstacle detection and two webcams from Etron Technology Inc. as well as one Quickcam Pro 9000 from Logitec. This equipment is shown in Figure 1.



(a) SABRE Lite i.MX6
FPGA DE0-NANO



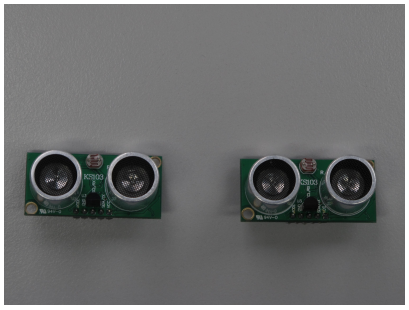
(b) CYS S8203 Servo



(c) EZRUN-150A-PRO brushless
speed controller

(d) Quickcam Pro 9000

(e) Remote control



(f) KS103 ultrasonic sensors

Figure 1: Equipment initially provided for the project. The car in its initial state is not shown due to the fact that the attachments made are difficult to remove. Also missing are the Eltron webcams used for the stereo camera as they were dismantled for this purpose. Their boards can be seen in Figure 2.

2.4 FPGA Motor Control

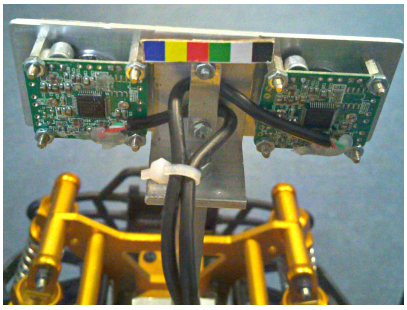
To allow control of the motors with the FPGA we examined the existing cables. Manuals for the different parts were difficult to acquire, so we used an oscilloscope and a multimeter to examine the car while it is operated by remote control. This allowed us to identify the roles of different wires and to design a solution to operate them via Pulse Width Modulation (PWM) signals generated from the development boards or the FPGA. Details on the parameters for the SY-BLS5 can be found in the appendix.

2.5 Computer Vision

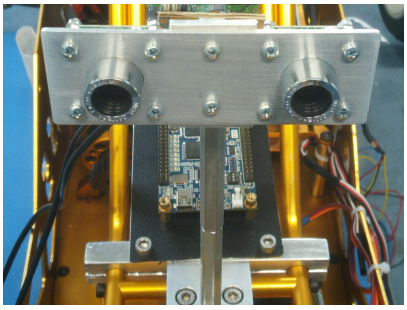
To allow the car to recognize objects such as traffic signs or traffic lights, we made use of the webcams supplied to us. During testing we replaced the Quickcam Pro, not only because it does not fit the low cost target with its price of currently around 100 euro, but also because it is not suited for the environment we used it in. This webcam is designed for use in video conferences and similar environments. In our vehicle the advantages of the model turned into disadvantages: The automatic focus could not cope with quickly changing distances and objects were often out of focus during driving. Furthermore it sends compressed data, allowing higher resolutions while staying under the USB 2.0 Bandwith limit of 35 MB/s [8]. Decompressing the video produces additional processor load and the high resolution is not necessary. We replaced it with a PlayStation Eye webcam. This model has a price below 10 euro, was created for capturing quick movements and does not use image compression [9]. For image processing we utilized the open source library OpenCV [10]. OpenCV contains a wide variety of image processing functions, which would have been impossible to implement in the given timeframe. It also supports OpenCL acceleration. Help on compiling OpenCV can be found in the appendix under Appendix C "OpenCV". Detailed information on algorithms can be obtained from the students who worked on them (see subsection 2.1 "Project Participants and Schedule"). Image processing is computationally extensive and thus needs to be well optimized on systems with a weak CPU like the Raspberry Pi 2 or the i.MX6.

2.5.1 Obstacle Detection

To allow the car to detect obstacles in front of it, a binocular stereo camera was built from the two Etron cameras. Figure 2 shows the assembled camera. Both webcams are on the same level and face towards the same direction, as this simplifies disparity calculation and thus puts less burden on the processor. A function provided by OpenCV is used to calculate the disparity map, that is passed as an input to the pathfinding algorithm to avoid detected obstacles. Difficulties were encountered during implementation of stereo recognition as it was not possible to synchronize



(a) Back



(b) Front

Figure 2: Stereo camera built from two webcams. On the back of the camera a color calibration stripe is attached which is in the third cameras field of view, allowing it to react quicker to illumination changes.

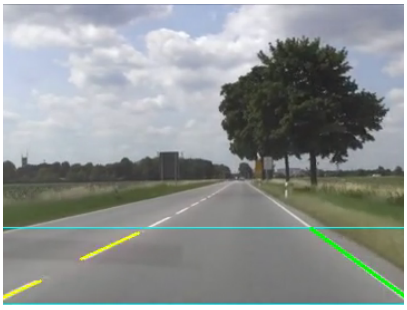
both cameras. OpenCV stores images received from the webcam in a ring buffer. To get current images a thread for each camera was created, which queries images continuously. While this significantly improved results, there was still a difference of up to one frame between the images, equal to 1/30th of a second, the inverse value of the maximum framerate supported by the cameras. This results in huge errors at high speeds and in curves. Solutions to this are presented in 3.2.2 Stereo Vision.

2.5.2 Traffic Light Detection

Because of the small size of our car compared to a regular one, the front cameras may miss elevated signs or traffic lights. So we added a third camera, the PlayStation Eye, on top of the car, pointing slightly upwards. To detect Traffic lights a number of simple image processing steps is run on the Image. First it is converted to HSV color space. In this color space hue is a separate value and the image can be filtered by color. This allows to distinguish traffic light colors. Morphologic and blur operations close holes created by noise in the image. A circle detection algorithm is run and the results are verified in regard to their circular shape and continuous color to obtain the final results. To exclude tail lights from leading vehicles and similar candidates for false positives results outside of a Region Of Interest (ROI) are discarded.

This detection algorithm performed well under sufficiently lighted conditions, but under difficult lighting the accuracy dropped. An example of detection can be seen in Figure 3a. As a solution to this we attached a color calibration stripe at the bottom center of the cameras field of view. This allows the algorithm to pick reference colors from the stripe and adjust its threshold parameters for filtering the HSV converted image.

(a) Traffic lights detected



(b) Lane markings detected

Figure 3: Examples of traffic light and lane detection. On the left image a correctly identified traffic light and a false positive (only detected for one frame) is shown. On the right image the detected lane marking is marked in green, the yellow lines are possible candidates which have been discarded because they could not be verified against previous frames due to the dashed nature of the line.

In addition this allowed to compensate another flaw of common webcams, automatic brightness adjustment. Usually this helps to cope with alternating light conditions, but under certain circumstances the adjustment fails, for example if there is a big bright or dark surface in the image like shown in Figure 4b. This was solved by reading the RGB value of yellow on the stripe. If its blue component exceeded a certain value, brightness was turned down and if its red and green parts fell below a certain threshold, brightness was increased. This greatly improved results.

A big problem still remaining are false positives like the circular red shape inside a STOP sign's P. This could be solved by looking for shapes outside of the circle.

2.5.3 Traffic Sign Recognition

Traffic sign recognition uses a similar approach to traffic light detection. The image is first converted to HSV color space. After filtering by color and applying Canny Edge Detection [11], shapes are detected via a function of OpenCV and possible signs are selected by checking them for certain characteristics regarding shape and color. In case of a speed sign the algorithm attempts to match the number against stored template images to determine the number inside the sign. Only speed limits up to the cars maximum speed of 80 km/h are recognized. While detection of some signs, like STOP or yield signs, works very good, differentiation of the speed signs is currently unreliable. This may be improvable by using an OCR library or by utilizing a better template matching algorithm.

A possible speedup could be achieved by converting the image to HSV only once and passing it to both, traffic light and traffic sign recognition.

(a) Correct brightness

(b) False brightness

Figure 4: Error in automatic brigthness adjustment of a webcam. A big black object causes loss of detail in the surrounding elements. Both images have the same background and illumination

2.5.4 Lane Recognition

Lane recognition was implemented, but not thoroughly tested yet. The processing steps are kept as simple as possible to increase throughput. One camera of the stereo camera setup was used as input because it has the ground in view. The ROI is restricted to areas below the horizon. At first the image is converted into grayscale and Canny Edge Detection [11] is run. The in OpenCV integrated algorithm [12] to find contours is called, returning edges which are connected. Results are passed to the Ramer–Douglas–Peucker algorithm [13][14]. This simplifies detected contours and allows to filter out small lines, reducing noise. The remaining lines are compared against lines from the last iterations in a verification step and returned if they match. This approach is very fast and works well on straight roads, but has serious limitations regarding the shape of lane markings. For example curved lanes are not detected if they bend too much, as they do not qualify as lines. Dashed lane markings are also not detected at certain speeds as the white dash in the reference data may be located at the place of a space between dashes in the current frame, resulting in the line being discarded in the verification step. This problem is demonstrated in Figure 3b. It could be solved by considering the speed of the car and moving recognized lines accordingly when comparing them. Another problem arises when driving around corners, where a new environment moves into view of the camera in a few frames and detected lanes can not be verified because no past information is available.

2.6 Localization and Navigation

This part was not finished till the project was handed over to the next team. However an GPS receiver and an inertial measurement unit (IMU) were purchased and it was planned to combine these with a Kalman filter. This would allow to determine the accurate location and heading of the

vehicle [15]. Using this data and vector maps supplied by the Openstreetmap project [16] a path could be computed, along which the car has to drive. For this several libraries are available [17].

2.6.1 Local Navigation with Tentacles

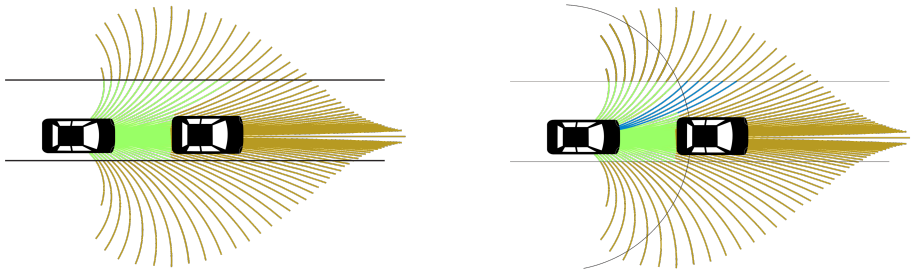
For local navigation it was planned to implement an algorithm derived from "Driving with Tentacles: Integral Structures for Sensing and Motion" [18]. A simulation was created to visualize the algorithm and first steps towards implementing it for the project were done. The approach is to precompute a number of possible paths the vehicle can take, depending on the speed of the car. This has advantages if computational resources are limited. Creating the precomputed paths (Tentacles) has to be done only once. The path which comes closest to the desired target can be selected and executed. This target can be the point at which the global navigation gives the next instruction. As the paths are evaluated several times per second, the car never fully executes a path.

2.6.2 Obstacle Avoidance

For obstacle avoidance with tentacles the car only has to check each of the paths for collisions with objects it has detected with its sensors. If objects are stored in an array, for example an image, relative to the car, each check only requires a conditional check. If the resolution in which individual tentacles are stored is within reasonable limits, regarding available computational resources, all tentacles of a set can be checked very fast. Additionally the crash distance is computed for each tentacle, the distance at which the collision with an object would occur along the tentacle. This allows the car to drive along narrow, curved paths or on a narrow road with a vehicle in front of it by selecting the tentacle with the highest crash distance. This distance has to be higher than a certain limit below which the car would be unable to brake. Figure 5 shows this solution. If a tentacle with a detected crash has to be selected, this is also an indicator for the car to slow down. Furthermore tentacles which are in the middle of the lane can be preferred so the vehicle stays within its current lane.

For emergency braking the ultrasonic sensors are utilized. If an object is detected closer than a certain threshold, the car stops. This functionality is implemented in the FPGA so that the response is not slowed down or influenced by communication with the other devices.

The sensors which we had available gave reliable measurements at up to 6 meters in experiments, 5.5 are specified in their data sheet [19]. Assuming the speed is limited to about 20 km/h, the stopping distance on asphalt is 3m at worst [20]. If 30 measurements per second are taken, this adds another 0.2m which the vehicle travels in the worst case. So ultrasonic sensors can



(a) All colliding tentacles discarded (b) Save crash distance defined

Figure 5: Situation where all tentacles collide with objects. In case all colliding tentacles are discarded (a), all tentacles are classified as undrivable. If a speed dependent limit for the distance to an unavoidable collision is set (black semicircle), the vehicle can choose among some tentacles (blue). Dependent on the distance to the collision the vehicle can slow down. Image of car taken from Wikipedia [21]. The car sizes are exaggerated. Images based on graphic from "Driving with Tentacles: Integral Structures for Sensing and Motion" [18]

prevent damage to a vehicle at low speeds. At higher speeds, however, they can not provide any significant protection and alternate sensors which can trigger emergency baking must be used.

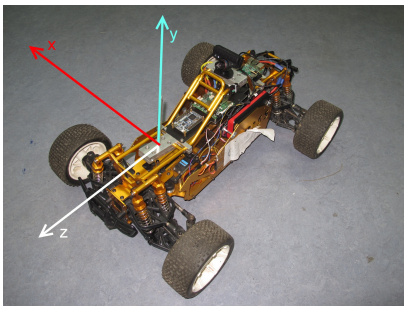
2.6.3 Algorithm Changes

In the original paper [18] each tentacle is computed as a circular arc. This may not be accurate because the car does not drive along a perfect circle. It takes a considerable amount of time for the servo motors used in our project to change the angle of the front wheels, a coarse measurement estimates the time at 0.5 seconds to change from steering fully to one side to steering fully to the other side. If the car is traveling at 20 km/h and tentacles are checked every 0.1 seconds, the real car position deviates approximately 30 cm from the estimated one, assuming the selected tentacle steers fully right and the car is currently steering fully left. So it is more accurate to let the steering angle change slowly over time.

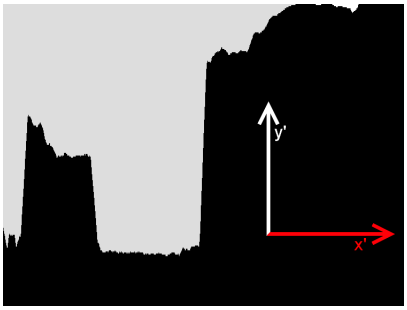
In the paper a set of tentacles for each speed is calculated. Because of the slow change of the steering angle it may also be necessary to compute individual sets for the same speed and different current steering angles. Each set only requires about 50 to 200 KB of memory, depending on tentacle resolution and number, therefore many can be stored.

2.6.4 Top View Interpretation of Disparity Map

The disparity map generated from the stereo camera can be used as an input to the tentacle algorithm. Images from both individual cameras composing the stereo camera are passed to the



(a) Coordinate axes on car



(b) Coordinate axes on topview

Figure 6: Conversation to top view visualized on images of the car and a sample topview.

disparity map calculation, generating a disparity map with a resolution equal to the resolution of the individual cameras in the stereo camera construction. A 3 axis coordinate system, shown in Figure 6a, is assumed, with the z axis pointing forwards from the cars point of view, the x axis perpendicular to the side of the car and the y axis perpendicular upwards. In this case the pixel coordinates of the disparity map x and y correspond, with perspective induced errors, to the x and y axis while the value of the pixels corresponds to the z axis. The y axis is discarded by taking the maximum of pixel values on each column of the disparity map, leaving only a top view. This is done because the logic would have to factor in the scaling of objects with distance and the cars height to calculate if it can pass below objects. This way the car will avoid obstacles which it would not have hit because of its height, like a branch, but the path finding can be restricted to two dimensions. In the next step the information is converted into an image where the axes x' and y' correspond to pixel coordinates, the color black marks "no obstacle", white a potential obstacle. This image can then be interpreted by the tentacle algorithm and the vehicle can be steered accordingly. The result is visualized in Figure 7.

Additional steps have to be executed to increase accuracy. The camera coordinate system is different from the coordinate system centered on the car. This results from the viewing angle and perspective. To correct this, inverse perspective mapping [22] with parameters obtained from earlier camera calibration can be used. Additionally disparity maps which are generated with the fast OpenCV block matching algorithm tend to be noisy. This was solved by averaging values between columns when generating the top view.

2.6.5 Limitations

This approach is very fast and simple to implement, but it also comes with disadvantages which depend on the environment the vehicle operates in. Because no map is generated, in contrast to Simultaneous Localization and Mapping (SLAM) algorithms, the vehicle can drive into dead ends.

(a) Input disparity map (b) Top view with tentacles

Figure 7: Example of top view generation from disparity map. A sample tentacle set is shown to visualize its application. Only paths curving to the right are collision free.

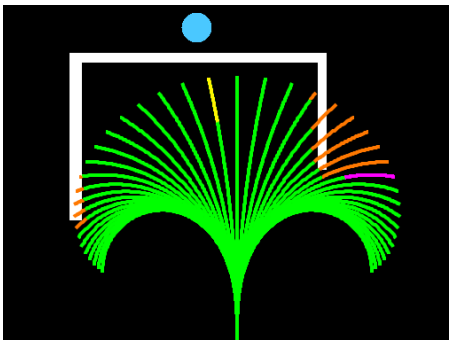


Figure 8: Example where the tentacle approach generates an invalid solution. A blue circle marks the target. The obstacle is white. The yellow marked tentacle will be selected, letting the car drive into a dead end, while the magenta tentacle would be the best choice to reach the target.

An example for this is shown in Figure 8. This could be solved by developing a SLAM solution with tentacles, which will likely increase the processor load significantly, or by implementing a backtracking solution. Dead end streets can be avoided by the large scale navigation solution. Obstacles like the one shown in Figure 8, however, are unlikely to occur on roads, as most objects like other vehicles and persons have an approximately convex shape. This is evident from the ARGO experiment mentioned in the introduction. The ARGO car drove a high percentage of a 2000 kilometer journey autonomously and did not perform mapping [4][23].

2.7 Results

At the end of the semester we successfully demonstrated the different computing platforms and algorithms working together. As localization was not completed, the car was mounted on a

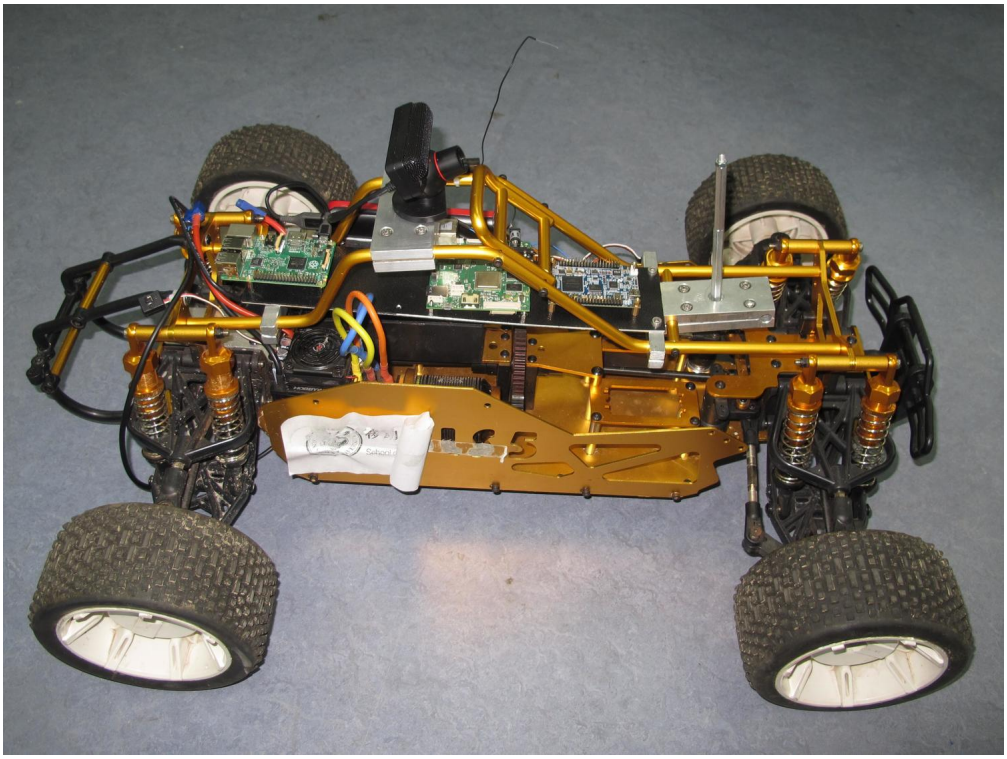


Figure 9: The car in its final setup. On top the Playstation Eye webcam used for sign detection. Below the three boards can be seen. From left to right: Raspberry Pi 2, i.MX6 SABRE LITE, FPGA DE0-NANO. The Rasperry Pi is supplied by an EC Technology Powerbank for mobile phones, visible as red line behind the Eye webcam. The rod in front serves as mount for the stereo camera. To keep other components visible ultrasonic sensors and the cables connecting the boards and motors are not attached.

box with the wheels moving freely. A server was started on the i.MX6 SABRE Lite with the Raspberry Pi 2 connecting to it. On the Raspberry Pi traffic light and sign recognition was run. If certain traffic signs or lights were detected, the Raspberry Pi sent a signal to the server. The i.MX6 SABRE Lite then used this information to generate steering commands that were sent to the FPGA and finally forwarded to the motors in PWM form. For example on detecting a red light the car stopped. The algorithms ran fast enough to process 30 frames per second, which is sufficient for moderate speeds like the 20 km/h that we had chosen as limit, but too low for the high speeds the car can potentially reach. Images from the stereo camera were saved. The algorithms of the students were completed to the stages indicated above. Additionally both boards were shown to be ready for use and code was created to call algorithms in parallel, query the cameras as concurrently as possible and to allow saving of image sequences recorded by the car while driving. These image sequences can be used as input to further improve algorithms.

3 Alternative Algorithms, Sensors and Platforms

During the project it became apparent that many parts or algorithms used were non ideal solutions for the task at hand. Therefore this part of the thesis will highlight alternative solutions which may have performed better. It will also describe available sensors and parts in detail in regard to their use for autonomous vehicles.

3.1 Electronic Control Units

To control the vehicle one or more Electronic Control Units (ECUs) are needed to execute algorithms and send correct signals to the different actuators and sensors. The choice of these ECUs is very important, as it determines the available programming environment, the available interfaces, which can be used to attach sensors or actuators, and the computational resources which are available for processing algorithms.

3.1.1 ARM Development Boards

In the project two development boards, the i.MX6 SABRE Lite and the Raspberry Pi 2 Model B, were used. In recent years several such platforms have been released at prices well below 100 euro. Such ARM based boards currently have a processing power comparable to personal computer processors released approximately 10 years prior to their release [5] and they are often equipped with a broad range of interfaces, including interfaces which are commonly found on personal computers like USB as well as interfaces for generic electronics like GPIO pins. All components required for the operation of the board are usually on one printed circuit board with the only exceptions of input devices and power supply. Many such devices require adding storage in form of a MicroSD [24], while others make use of an integrated eMMC storage [25].

These boards have multiple advantages if used in a low cost autonomous vehicle. If a MicroSD is used for storage, the acquired data is likely to survive harsh environments and may even survive crashes which destroy the vehicle. Some MicroSDs can withstand accelerations of up to 500g and complete immersion in water [26]. If the MicroSD is also used as storage for the operating system, a damaged board can be replaced in seconds. The ARM processors used on most of the boards also work well with passive cooling, so there are no moving parts. Through the different interfaces a broad range of hardware can be attached, ranging from commercial USB devices like webcams to sensors which utilize the I^2C bus. Additionally their power consumption is very low compared to traditional computers, enabling long runtimes for battery powered platforms [27]. Some boards are powered via USB, allowing the use of portable batteries, designed for smartphone recharging, as energy source.

Such boards often allow to install a broad variety of operating systems, giving the user the ability to choose one, that fulfills his requirements for the task at hand. For example on a Raspberry Pi many different Linux systems like the Debian based Raspbian can be installed, as well as Windows 10 IoT and freeRTOS.

They also possess serious disadvantages in comparison to current personal computers. The reduced processing power causes restrictions which will be a main focus in the following parts of this document. Also driver and manufacturer support varies widely. Often the community has to resolve upcoming issues or problems with drivers only working for certain operating systems as is evident from lengthy forum discussions [28]. Therefore it is very important to research this before choosing a board. Lastly modern computers often possess graphics cards or even dedicated accelerator cards which can significantly boost performance for tasks like image processing. While many boards are equipped with GPUs, drivers for these are often only targeted at mobile devices, focusing on 3D acceleration and lacking support for parallel computing languages like Cuda and OpenCL. For example the i.MX6 platform, that was available for the project, has only partial OpenCL Support [29].

3.1.2 FPGAs

Field-Programmable Gate Arrays (FPGAs) are circuits which can be programmed to behave like arbitrary circuits or computing platforms. This makes them very versatile and powerful, allowing to use them efficiently for any task with many similar steps. Their performance can not be compared with CPUs like ARM and x86 processors because these are targeted at executing their instruction set very efficiently and mostly sequentially, while FPGAs can compute many tasks in parallel. FPGAs are in this respect similar to GPUs. Tasks for which FPGAs are especially suited include accelerated image processing and general digital signal processing. They often have many available interface pins which allow to connect them to many different devices. Other uses of FPGA like fast prototyping of circuits are mostly irrelevant in creating low cost solutions for autonomous driving.

An FPGA is especially suited to generate signals which are very sensitive to timing issues due to the possibility to dedicate a certain part of its computational resources entirely towards generating a signal. PWM signals serve here as sample for any signal. On a processor with a normal operating system, generating a PWM signal poses two problems. At first generating a reasonably stable PWM signal requires more computational resources because the operating system has to do many context switches per second to the PWM generating process. Additionally it is not possible to generate a clean PWM signal at high frequencies in modern multitasking operating systems, because other processes may delay the context switch to the PWM generating thread, producing errors in the signal. This can be solved by using a real-time operating system.

Using an expensive FPGA solely for generating a PWM signal has become obsolete due to cheap devices being available which have hardware PWM generators. For example it is possible to achieve a stable PWM signal with a Raspberry Pi by utilizing the DMA controller of its Broadcom chip [30], but other signal forms may still require the use of an FPGA.

Thus an FPGA may be advisable for use, if no other accelerator, like a GPU with OpenCL support is available, or stable PWM generation can not be achieved otherwise. Their prices are often higher than 100 euro, but there are FPGA boards available with prices as low as 50 euro, making them interesting candidates for coprocessors.

3.1.3 Microcontrollers

Microcontrollers have very low power requirements and are the cheapest possible ECUs. Their prices can be below 1 euro. Their processing power and storage capacity however is quite limited compared to development boards. They can be used to control autonomous vehicles with simple sensors, which return values that can be used without much processing. For example a vehicle equipped with nothing more than a few ultrasonic sensors and a GPS can be steered towards its target avoiding obstacles [31]. If sensors requiring a high amount of processing are used, additional computational resources have to be added through another ECU. Another problem of microcontrollers is missing prior work. On other platforms existing algorithms and code can be used. On microcontrollers this is often difficult because they are restricted to the use with specific programming languages, variables and functions.

Several possible uses of microcontrollers emerge from these characteristics. Microcontrollers may be used in conjunction with an FPGA as accelerator and simple control logic to steer a vehicle. Another possibility is to use microcontrollers in conjunction with an ECU which provides greater computational resources to provide a function which the other ECU can not provide. An example for this may be PWM generation for an ARM development board without hardware PWM. Microcontrollers may also offer additional benefits if they are paired with other useful devices, like the ESP8266, which provides wireless access.

3.1.4 Other Options

Personal computers (PCs) with x86 or x64 processor architecture and modular components are still too large to be used on small autonomous vehicles. Also their components often have high power consumption and generate enough heat to require active cooling. Smaller scale PCs have been appearing on the market in recent years, but their prices are still far too high for low cost applications with typical prices starting around 300 euro.

Smartphones are also a possible choice for an ECU. Current Smartphones combine powerful CPUs and GPUs together with good cameras in a very small space, although they have significant disadvantages in regard to their interfaces. Most smartphones possess only a single USB port, which may not support USB host mode, and a 3.5 mm audio jack. They can be used in conjunction with a development board, using wireless communication. Modern smartphones may support using OpenCL on their GPU [32]. Smartphones, which allow this, are currently too expensive for low cost solutions with prices typically above 250 euro. The prices, however, have been declining in the last years, and currently the cheapest OpenCL capable smartphone, using an Adreno GPU, is available at only 150 euro.

3.1.5 Using Multiple ECUs

Using more than one ECUs may increase the available computational resources, capabilities and/or interfaces, but it has to be considered if the disadvantages are worth it. Data interfaces between ECUs have limited bandwith and they often connect to interfaces where otherwise sensors could have been attached. Another concern is coordination between sensors connected to different ECUs. Interfaces introduce delays which make synchronization difficult. There are situations in which the use of additional ECUs provides big advantages. For example the two cameras of a stereo camera setup could be triggered and processed by an FPGA and then the image along with the calculated distance information could be passed to the main ECU.

3.2 Sensors

A wide variety of sensors exists, which allow a system to operate autonomously. For our project we were equipped with webcams and ultrasonic sensors which are easy to acquire at low cost. Here possible uses and characteristics are presented and alternatives mentioned where available.

3.2.1 Range sensors

Many range sensors are available, but the most commonly used ones are ultrasonic sensors. When used for distance measurement ultrasonic sensors work by emitting a sound wave and measuring its time of flight. This approach allows simple construction and low costs, but its capabilities are limited. Ultrasonic sensors generally have a broad angle, for models on the websites mentioned genereally between 15 to 60 degrees, in which they detect objects. This makes construction of detailed environment maps complicated as a detected object can be anywhere on a circular arc in front of the sensor. Also the sensors trigger on the first detected reflection, so an object which is behind another object will not be recognized. It is difficult to determine characteristics of an object or to recognize a hole where the autonomous vehicle could fit through, as the walls

around the hole reflect the sound wave [33]. The range of ultrasonic sensors is also very limited. Depending on the frequency of the generated sound pulses the sensors work reliably at distances of up to 10m [34], limited by the speed of sound. At speeds of over 40 km/h this is not sufficient to brake a vehicle in time [20]. Additionally the signal of ultrasonic sensors can be deflected by surfaces due to specular reflection on flat surfaces, so objects at an angle may be detected with wrong distances or not at all. Also foam can absorb the ultrasonic pulse [35]. The rate at which an ultrasonic sensor can take measurements is limited by the speed of sound and the maximum range to be measured. As an example, sensors of the company Senix can take up to 200 measurements per second for low distances [36].

Despite of all these disadvantages ultrasonic sensors have advantages as well. Due to the wide angular resolution a single ultrasonic sensor can detect objects inside a big circular sector in front of it. No further processing is necessary as the sensor usually returns a distance value right away. Ultrasonic sensors generally require low power. The sensors available on the websites searched for prices generally require less than 50mW to operate. The sensors can detect objects that are not detected by other sensors, for example glass by a camera.

Based on these characteristics ultrasonic sensors are fit for emergency braking at low speeds in case the other sensors failed to detect an object. It is complicated to use them for path planning because they do not provide details about detected objects. They can be used as rear sensors to allow driving backwards if other sensors would be too expensive or too complicated to process. It is possible to make a robot which uses a big array of ultrasonic sensors for simple navigation [31]. Using ultrasonic sensors for mapping is also possible, but due to the limited range the robot has to came close to obstacles to identify them and due to the wide angle at which the sensors detect objects, the maps are of poor quality or require extensive data collection [37].

Ultrasonic sensors are available at prices starting around 1 euro with detection ranges of 4 meters. At prices starting from 10 euro models are available which are waterproof or have higher ranges. Instead of soundwaves microwaves could be used, which allow higher scanning frequencies and ranges because of their higher propagation speed and lower absorption by air [38]. At low prices however no such sensors were found. Sensors measuring movement with microwaves and the Doppler effect are available, so it should be possible to manufacture such devices at low prices. They may have been missed or pushed out of the market through competition from cheaper ultrasonic sensors. Infrared distance sensors are also available. They measure distance in very narrow angles, have a typical range of up to one meter and cost around 4 euro. The low range and bad performance of infrared based sensors under influence of sunlight make their use difficult.

3.2.2 Cameras

Since the MilleMiglia Tour of the ARGO car, the price for cameras has significantly declined. Today USB webcams are available for prices less than 2 euro. These webcams are not as sophisticated as their industrial counterparts but they can nevertheless be invaluable sensors for an autonomous vehicle. A camera delivers detailed information about its field of view. From colors and positions inside the image a variety of information can be derived. It would be beyond the scope of this document to try listing each possible use, so some examples relevant for autonomous vehicles will be named and possibles issues, encountered when using low cost components, will be discussed.

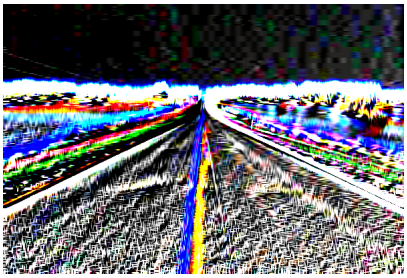
The main concern with using cameras on a low cost autonomous vehicle is the high amount of processing required to extract information out of an image captured. But in recent years FPGAs have become available at low prices and development boards have become fast enough to cope with this. There are however several techniques to speed up evaluation of images captured by the camera. First of all it is often not necessary to use the full available resolution. Webcams are available which offer resolutions higher than full HD at prices less than 5 euro. But often a resolution of 320x240 pixels is enough to extract needed features, reducing the data to be used in computation 27 times compared to full HD. Additionally high resolutions create another problem with many development boards. The interface used is often USB 2.0 which has a limited bandwidth of only around 35MB/s, multiple devices share this bandwidth in many cases. This allows only one camera recording at a resolution of 640x480 pixels and 30 frames per second (fps) as can be seen in Equation 1. If the resolution is reduced to 320x240 pixels, five webcams can be used in parallel or two with 60 fps. A high framerate is important for autonomous vehicles, especially at high speeds, to allow them to react in time to changes in their environment.

$$\begin{aligned} Width \cdot Height \cdot fps \cdot colors &= bytes/s \\ 640 \cdot 480 \cdot 30/s \cdot 3 = 27648000\, bytes/s &= 26.37\, MB/s \end{aligned} \tag{1}$$

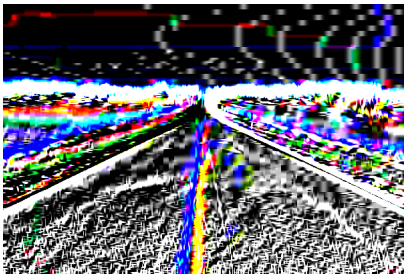
Other solutions for this problem exist. For example cameras not in use can be shut down, suited for example for a vehicle with a back and a front camera, which only uses the one in the direction of driving. Also some development boards offer other interfaces, for example the Raspberry Pi has a CSI-2 interface which has a higher maximum bandwidth, not shared with the USB devices. Additionally there exist cameras which compress captured images. While this allows a greater throughput it has serious disadvantages [39]. For further processing the image data has to be decompressed, causing additional load for the processor. Also the algorithms used in these cameras are implementing a lossy compression, pleasant for the human eye. The artifacts introduced are barely visible for humans but may make image processing difficult. Effects similar

(a) Original image

(b) Image compressed as low quality jpeg



(c) Original image with neon edge filter from GIMP applied



(d) Compressed image with neon edge filter from GIMP applied

Figure 10: Artifacts introduced by compression visualized by applying an image filter. While differences in the compressed image are barely visible to the human eye, the results of the image filter are vastly different. Effects are exaggerated by choice of the filter and compression quality to make them clearly visible.

to the one shown in Figure 10d may reduce robustness of image processing done on the acquired image. For example block artifacts are introduced by the codecs MJPEG and H.264, common for webcams which use compression. These blocks may show up in edge detection.

When choosing a camera it is advisable to use a webcam which is targeted at motion detection instead of serving for video conferences. While most webcams return at most 30 fps the PlayStation Eye webcam we used for our project allows up to 187 fps at a resolution of 320x240 pixels. It does not use compression for its images and allows control of many camera parameters.

The choice of image processing algorithms is equally important. Recently there have been great advances in image processing and especially object recognition with deep learning, an approach where high-level abstractions of data are modelled by multiple layers of processing. These algorithms require great computational resources which are not available for low cost solutions. There have been first steps to solve this problem by optimizing deep learning algorithms, but in general these approaches are not suited for the task at hand [40]. It is better to use simple techniques which can also return very good results if correctly adapted to the inputs used. This allows fast reactions to detected environment changes and leaves computational resources for

Figure 11: Image from Figure 10 with applied Sobel operator

other parts of the vehicles logic. The use of image convolutions can deliver important intermediate results for many applications. Convolutions apply a simple filter to an image by sliding a matrix over the image, the so called kernel, multiplying each pixel covered by the kernel with the value stored within the covering cell, adding them together and storing the value in the center pixel. The kernel is rotated by 180 degrees before sliding it over the image. This is shown in Equation 2 with I being the image matrix, k the kernel matrix and n being the size of the kernel matrix. For example the simple matrix shown in Equation 3, called Sobel operator, allows to highlight edges in an image for further processing steps [41]. The results are shown in Figure 11. If multiple of these kernels are applied after each other they can instead be combined into a single kernel which is applied to the image in a single step, greatly reducing the processing time. Another advantage of these algorithms is that they are easily parallelizable. The image can be divided into arbitrary parts and processing can be done independent. This makes them very suitable for parallel computing, so FPGAs and GPUs with OpenCL support can be utilized. Many other fast processing steps exist, generally it is good to use parallelizable algorithms and take cache behavior into consideration.

$$I^*(x,y) = \sum_{i=1}^{n}\sum_{j=1}^{n} I(x-i, y-j)k(i,j) \tag{2}$$

$$k = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{3}$$

Computation of image processing algorithms can sometimes be accelerated by using image pyramids. This means precomputing a set of images by scaling the input image to different sizes. Features to be examined can then be searched in lower resolution images at first and

possible matches can be examined closer in higher resolution versions. This pyramid together with other intermediate steps can be shared between different detection algorithms. For example in our project the HSV conversion result can be used for both, the traffic sign and the traffic light detection.

If objects have to be recognized in an image, this can also be done by using so called feature extractors. These are algorithms which identify features with characteristics likely to be found in another image even if the viewpoint of the camera changes. To fulfill this, these features should be robust against noise or (motion) blur, should only occupy a small area of the image, be invariant in regard to geometric changes and distinctive for a certain feature. One descriptor fulfilling these requirements to a limited degree is the Harris corner detector [42][43]. This descriptor can be applied to the object in question and can be compared against matches in an image. Then a correspondence between the results can be found. There exist many variations to this approach with different feature extractors, additional smoothing steps and different matching algorithms. Most of them however require high computational resources targeting at the best possible detection quality, but if simple feature descriptors are used in combination with image pyramids good performance can be reached [44].

Another approach to object detection is template matching. Here the object is not described by characteristics, but by one or more samples of its appearance. These samples are moved over the image and matches are recorded. Slight modifications allow to match rotated versions and versions in different lighting conditions. Image pyramids and break conditions early in the template matching process are very important to speed up the process. The advantage of this method is, that it allows very exact localization of the provided templates in the image [45]. Template matching has been continuously improved over the years to solve specific tasks.

There are also many other fast image processing algorithms, and creativity can be very useful to adapt them to the task at hand. For example in the ARGO project the image was examined for symmetries to detect cars in front of the vehicle. They also used edge detection to search for symmetries in other domains. Very important is the use of prior knowledge to reduce the amount of data to be processed. For example for lane detection a ROI can be specified so that only the part of the picture below the horizon is analyzed. Also results from one algorithm can be used to restrict the ROI of another step, but doing this prevents these algorithms from being run in parallel.

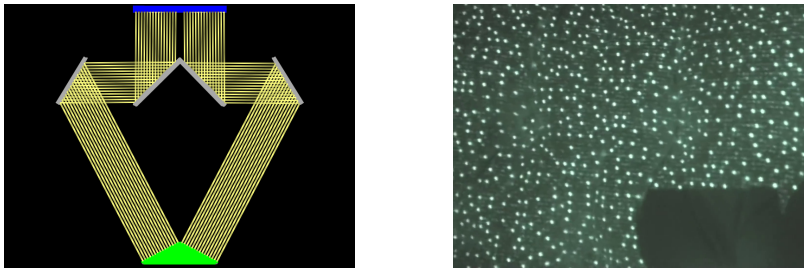
Stereo Vision

Very useful for navigating is information about the distance of objects in an image. For flat roads this can be derived by calibrating the camera and applying inverse perspective mapping [22]. Also other information can be used. The distance to vehicles on a street can be estimated by

their respective position on the road or by assuming an average width of vehicles and comparing against the width of a detected vehicle. Furthermore it is possible to estimate distance from an image sequence [46]. But this is a complex operation not suitable for slow CPUs. More reliable information about distance can be obtained from a stereo camera system, in which two cameras are used in parallel. Similar to human vision information about distance can be extracted from the displacement of an object between the view of both cameras. This information can be calculated for each pixel, generating a so called disparity map. Computing disparity can be difficult for homogeneous areas with no distinctive features. Also it is a complex operation to align the images first. This can be solved by ensuring that both cameras are mounted in parallel on the same plane. Features in one image appear then on the same line in the image of the second camera. From the change in pixel position the distance can be calculated, the greater the difference the closer the object. OpenCV includes an implementation of a very fast block matching algorithm to perform stereo matching. This algorithm utilizes image pyramids to increase performance and also returns disparity for homogeneous areas [47].

Such stereo cameras are available at prices starting around 60 euro currently, these cheap models are not fit for use in a moving vehicle because of low framerates. Stereo camera systems can however be built with simple tools and by using two webcams of the same brand. We made an attempt to this in our project and encountered issues with synchronization between the cameras. Synchronization is very important for this approach. If the two images passed to the disparity map calculation have a time difference, not only the distance, but also the movement speed of objects, influences the disparity, causing errors in distance calculation. Webcams usually send a stream of images after initializing them, not allowing to pause it to synchronize two cameras. Initializing two webcams at exactly the same time also proved to be difficult in our experiments. One solution to this may be using cameras which can be triggered, but these are usually used in industrial environments and expensive. But webcam boards may possess pins which allow synchronization by trigger signals if webcams are wired together. One example where this is possible is the PlayStation Eye camera mentioned earlier [48]. Because such functionality is not important for operating webcams in regard to their common uses this is however rarely documented in manuals.

Another very promising approach is the use of mirrors and prisms. Planar mirrors and simple prisms are available from many sources. These optics can be used to project images from two different viewpoints towards a single image sensor, mapping each viewpoint to half of the sensor. This idea is very old. Theodore Brown has built such a system, nicknamed stereoscopic transmitter, back in 1894. It is visualized in Figure 12a. Such systems are easy to build and do not require any synchronization as both images are captured by the same sensor. Disadvantages are distortions and chromatic aberrations caused by some of the possible optic systems and the resolution of the image sensor is halved. Distortion can be corrected by prior camera calibration, but the necessary



(a) Single lens with mirrors (b) Kinect IR pattern

Figure 12: Two possible stereo camera approaches illustrated. On the left with mirrors (gray) which direct light from different viewing angles of an object (green) onto the two halves of an image sensor (blue). On the right a pattern of a structured light system, in this case a Kinect, captured with an infrared camera. The change of the pattern on the bent surface can be observed. The dark area is the shadow of the camera.

calculations increase the computational complexity. The image resolution can be increased. As mentioned before, it is not always useful to utilize the full resolution of a webcam, so instead of creating a stereo camera with two webcams recording at a resolution of 320x240 pixels a single webcam recording at 640x480 pixels can be used. Many such systems have been successfully tested [49].

Stereo vision can also be achieved using active stereo vision systems. One of the solutions which can be acquired for less than 100 euro is the Kinect device. This system uses a single infrared camera together with an infrared projector. A structured infrared pattern is projected onto the scene, and the camera is calibrated in regard to the pattern on an object with a known distance. This method is therefore often named structured light stereo vision. Objects then cause changes in the pattern and from these the distance to an object can be derived. A section of the pattern generated by a Kinect can be seen in Figure 12b. This system works reliable, but can not be used in sunlight or even light shade because the infrared pattern can not be detected anymore [50].

Disregarding the interference from sunlight the use of active sensors is generally problematic when designing autonomous vehicles. If multiple vehicles are operating in the same area their active systems may interfere with each other. For example the infrared patterns from two Kinect devices could overlap, and, without adaption of the algorithms, the devices would fail to extract depth information from the images.

3.3 Algorithms

3.3.1 Localization

Outdoor localization has become very affordable. GPS modules can be purchased for less than 10 euro. Inertial measurement units (IMU) for 5 euro or for 10 euro together with a compass sensor and a barometer. An IMU and a GPS can be combined, for example via a Kalman filter, to obtain position information updated at around 50 Hz. In combination with the compass sensor providing an initial calibration for the heading accurate results can be obtained [15].

While this already provides reliable results, additional sensors can be used to further increase the accuracy or to compensate in case of a lost GPS signal. Odometric sensors recording the rotation of wheels are available for 10 euro. The barometer in IMUs equipped with it may be used as indicator for the current height.

Indoors the localization problem is much more complex. Simultaneous Localization and Mapping (SLAM) provides good positioning but requires high computational resources [51]. Many studies have been done to solve this problem and a broad variety of solutions has emerged. Particle filter based approaches have been particularly successful at making SLAM computationally feasible for slow CPUs. One algorithm named FastSLAM has served as inspiration for many derived works which have continuously improved the algorithm. Other approaches that use wireless networks as reference for distance measurement are possible. Wireless routers or even cellular network base stations can be used as reference for locating a vehicle. Ultrasonic sensors and input from cameras may also be used [52].

Mobile modems are available at prices starting from 10 euro, but to achieve a good localization 4G networks are required where the prices for modems start at 25 euro. It is unknown to the author if these modems are fit for such localization purposes, further research regarding this is required.

Wireless modules are available at prices starting below 1 euro. For 2 euro the ESP8266 Module can be acquired which can also serve as auxiliary ECU and has a growing community which supports development efforts [53].

3.3.2 Path Planning

Path planning is a very critical area for low cost vehicles because here the limited computational resources have great influence on the solutions. The tentacle path planning mentioned in section 2.6 seems to be very promising. Under different conditions other algorithms may become of interest. For example if mapping was performed, the map can be segmented into a grid and an optimal path can be found using the A* algorithm [54]. Because of the restricted computational resources and memory tradeoffs have to be made. Usually this is simple in path planning because

the accuracy can be decreased in favor of faster computing times. The number of precomputed paths for the tentacle algorithm, or the resolution of the grid laid over the map for A* can be reduced. In many algorithms heuristic approaches can improve performance. And sometimes, depending on the environment, it may be better, if the vehicle makes a non optimal decision for the movement direction instead of standing still. Another group of algorithms called Rapidly exploring Random Trees (RRT) generates a tree like structure in the search space till the target area is connected to the start. These algorithms tend to be computationally demanding, but can be greatly accelerated by using heuristics [55].

Many path finding algorithms have the disadvantage, that once a path has been found, the vehicle has to compute a trajectory following that path additionally. This can be easily avoided for tentacle and RRT algorithms by using already optimized trajectories as individual tentacles or parts of the tree. Depending on the situation, a simple PID controller can be used to follow a lane.

For path planning across long distances available libraries can be used. Open source projects exist which allow fast and efficient routing across long distances, often based on the OpenStreetMap project. Notable is the OSRM project [56] which has impressive performance [57], but to maintain this performance precomputed graphs are stored alongside with the map data, significantly increasing storage requirements [58]. Another notable project is the Navit car navigation engine. This project is specifically targeted at low end devices and requires only little memory, storage and computational resources. It is fast enough to run smoothly with a graphical interface on a smartphone with only 128 MB of RAM and a 400 MHz ARM CPU. The routing engine of Navit is currently tied to the user interface so it can not be used as effortlessly as OSRM [59].

If an Internet connection is available, also one of the many routing services can be used. Often small volume requests are free of charge and the computational resources on the vehicle can be saved for other tasks [60].

3.3.3 Control Logic

The choice of control logic depends on the preferences of the team working with the car. The approaches are very different and too numerous to be discussed in this thesis. However it is again important to take the limited computational resources into account. Frameworks may be beneficial by taking care of steps for which otherwise own solutions have to be found, but they can introduce too much overhead, consuming computational resources and memory for code that is not used in the project. Additionally frameworks may have certain limitations which are easily reached because the number of available solutions in choice of sensors and algorithms is vast. Finally frameworks may be difficult to port to devices with a non standard operation system or no operating system at all like an FPGA or microprocessor. Fuzzy logic decisions in cases where appropriate should be used to speed up decisions.

3.4 Conclusion

Creating a low cost autonomous vehicle poses many additional challenges compared to the use of higher priced parts. But it is certainly possible and opens up many possibilities where expensive solutions can not be used. During the project important lessons have been learned. The choice of sensors and the ECU is extremely critical. Not because it is not possible to reach the goal with other options, but because setting up the components, interfacing them with each other and optimizing code so, that it is fast enough to run on the provided ECUs can cause massive delays in the project. The choice of the correct components can not only be made from their technical specifications alone. It is equally important to research the manufacturer and driver support and it is very beneficial if there is a community behind a device which develops solutions and tutorials to support development. Even though the components we used caused many issues we successfully demonstrated that low costs components are capable to perform real time image processing for tasks relevant for autonomous cars. With the gathered experience, which has been passed on to the next team in form of code, documentation and advise, it should be possible to make the first test in truly autonomous mode soon. Today the choice of computing devices has been made much easier. The Raspberry Pi 3 has a very powerful CPU which should be more than enough to handle autonomous driving along with object detection. Additionally ARM based boards with support of the OpenCL Full Profile [61] are appearing. This will allow algorithms of much higher complexity to be run. I hope the ideas and concepts presented in this thesis support interested people as well as the team continuing our project in creating fascinating new systems.

References

[1] M. Breuer and M. Page, *The Man with the Strange Head and Other Early Science Fiction Stories*, ser. Bison frontiers of imagination. University of Nebraska Press, 2008, pp. 44–66. [Online]: https://books.google.de/books?id=tqbvavvzZCcC

[2] Paradise and iron | isfdb.
[Online]: http://www.isfdb.org/cgi-bin/title.cgi?1286314/ (Accessed 2016-04-15).

[3] Leonardo da Vinci's 1478 self-propelled car | howstuffworks.com.
[Online]: http://auto.howstuffworks.com/da-vinci-car1.htm (Accessed 2016-04-15).

[4] A. Broggi, M. Bertozzi, and A. Fascioli, "Argo and the MilleMiglia in Automatico tour," *Intelligent Systems and their Applications, IEEE*, vol. 14, no. 1, pp. 55–64, 1999.

[5] Performance-per-watt & how the Raspberry Pi 2 + Pi Zero compare to old NetBurst CPUs - Phoronix.
[Online]: https://www.phoronix.com/scan.php?page=article&item=raspberry-pi-burst&num=6 (Accessed 2016-04-18).

[6] Chronology of microprocessors (1996-1997).
[Online]: http://processortimeline.info/proc1996.htm (Accessed 2016-04-18).

[7] J. Straub and S. Kerlin, "Development of a large, low-cost, instant 3d scanner," vol. 2, no. 2, pp. 76–95.

[8] Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, and Philips, "Universal serial bus specification revision 2.0."
[Online]: http://sdphca.ucsd.edu/Lab_Equip_Manuals/usb_20.pdf

[9] PLAYSTATION EYE - q+a | three speech: Semi-official PlayStation blog.
[Online]: http://threespeech.com/blog/2007/04/playstation-eye-qa/ (Accessed 2016-04-24).

[10] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, 2008.

[11] J. Canny, "A computational approach to edge detection," no. 6, pp. 679–698.
[Online]: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4767851 (Accessed 2016-05-03).

[12] S. Suzuki and K. be, "Topological structural analysis of digitized binary images by border following," *Computer Vision, Graphics, and Image Processing*, vol. 30, no. 1, pp. 32 – 46, 1985.
[Online]: http://www.sciencedirect.com/science/article/pii/0734189X85900167

[13] U. Ramer, "An iterative procedure for the polygonal approximation of plane curves," vol. 1, no. 3, pp. 244–256.
[Online]: http://www.sciencedirect.com/science/article/pii/S0146664X72800170 (Accessed 2016-05-14).

[14] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 10, no. 2, pp. 112–122, 1973.

[15] P. Zhang, J. Gu, E. E. Milios, and P. Huynh, "Navigation with imu/gps/digital compass with unscented kalman filter," in *IEEE International Conference Mechatronics and Automation, 2005*, vol. 3, 2005, pp. 1497–1502.

[16] OpenStreetMap.
[Online]: https://www.openstreetmap.org/ (Accessed 2016-04-24).

[17] Routing - OpenStreetMap wiki.
[Online]: https://wiki.openstreetmap.org/wiki/Routing#Developers (Accessed 2016-04-25).

[18] F. Von Hundelshausen, M. Himmelsbach, F. Hecker, A. Mueller, and H.-J. Wuensche, "Driving with tentacles: Integral structures for sensing and motion," *Journal of Field Robotics*, vol. 25, no. 9, pp. 640–673, 2008.

[19] *KS101B/KS103/KS103S Ultra Sonic Range Finder*, Kingsin Technologies Co., Ltd.

[20] Bremsweg und Anhalteweg | Unfallanalyse Berlin.
[Online]: http://www.unfallanalyse.de/beratung/rechner.html (Accessed 2016-04-14).

[21] Hedaja, "Icon eines Autos von oben gesehen."
[Online]: https://commons.wikimedia.org/wiki/File:Car_icon_top.svg (Accessed 2016-05-15).

[22] A. M. Muad, A. Hussain, S. A. Samad, M. M. Mustaffa, and B. Y. Majlis, "Implementation of inverse perspective mapping algorithm for the development of an automatic lane tracking system," in *TENCON 2004. 2004 IEEE Region 10 Conference*, vol. A, Nov 2004, pp. 207–210.

[23] A. Broggi, M. Bertozzi, A. Fascioli, C. G. L. Bianco, and A. Piazzi, "The ARGO autonomous vehicle's vision and control systems," vol. 3, no. 4, pp. 409–441. [Online]: https://air.unipr.it/retrieve/handle/11381/1449465/1262/ijics.pdf (Accessed 2016-04-17).

[24] M. Maksimović, V. Vujović, N. Davidović, V. Milošević, and B. Perišić, "Raspberry Pi as Internet of things hardware: performances and constraints," *design issues*, vol. 3, p. 8, 2014.

[25] Odroid C2 hardware [odroid wiki]. [Online]: http://odroid.com/dokuwiki/doku.php?id=en:c2_hardware (Accessed 2016-04-20).

[26] SanDisk proof. [Online]: https://www.sandisk.com/about/proof (Accessed 2016-04-21).

[27] Arduino Uno vs BeagleBone vs Raspberry Pi power consumption. [Online]: http://makezine.com/2013/04/15/arduino-uno-vs-beaglebone-vs-raspberry-pi/ (Accessed 2016-04-21).

[28] Mali gpu driver - device drivers gpu - orangepi. [Online]: http://www.orangepi.org/orangepibbsen/forum.php?mod=viewthread&tid=53&extra=&page=1 (Accessed 2016-04-21).

[29] OpenCL on Sabre iMX6Q. [Online]: https://community.freescale.com/thread/305019 (Accessed 2013-03-12).

[30] PWM on the Raspberry Pi pi-blaster | github. [Online]: https://github.com/sarfata/pi-blaster/ (Accessed 2016-04-12).

[31] U. Farooq, M. Amar, E. ul Haq, M. U. Asad, and H. M. Atiq, "Microcontroller based neural network controlled low cost autonomous vehicle." IEEE, pp. 96–100. [Online]: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5460762 (Accessed 2016-05-02).

[32] V. Hindriksen. OpenCL support on recent Android smartphones. [Online]: https://streamcomputing.eu/blog/2014-06-30/opencl-support-recent-android-smartphones/ (Accessed 2016-05-03).

[33] N. Kelly, "A guide to ultrasonic sensor set up and testing instructions, limitations, and sample applications," 2009.

[34] D. P. Massa, "Features-choosing an ultrasonic sensor for proximity or distance measurement part 1: Acoustic considerations-the first step toward identifying the right proximity sensor for your application is," *Sensors-the Journal of Applied Sensing Technology*, vol. 16, no. 2, pp. 34–37, 1999.

[35] Pwm on the raspberry pi pi-blaster | github.
[Online]: https://senix.com/frequently-asked-questions/ (Accessed 2016-04-13).

[36] Frequently asked questions about ultrasonic sensors | Senix.
[Online]: https://senix.com/frequently-asked-questions/ (Accessed 2016-04-14).

[37] W. D. Rencken, "Concurrent localisation and map building for mobile robots using ultrasonic sensors," in *Intelligent Robots and Systems '93, IROS '93. Proceedings of the 1993 IEEE/RSJ International Conference on*, vol. 3, pp. 2192–2197.

[38] G. Durgin, T. S. Rappaport, and H. Xu, "Measurements and models for radio path loss and penetration loss in and around homes and trees at 5.85 ghz," *IEEE Transactions on Communications*, vol. 46, no. 11, pp. 1484–1496, Nov 1998.

[39] AlexP. PS3Eye - not your typical webcam.
[Online]: https://codelaboratories.com/research/view/ps3eye-not-your-typical-webcam (Accessed 2016-05-04).

[40] O. Dürr, Y. Pauchard, D. Browarnik, R. Axthelm, and M. Loeser, "Deep learning on a raspberry pi for real time face recognition," in *Eurographics (Posters)*, pp. 11–12.
[Online]: https://www.researchgate.net/profile/Oliver_Duerr/publication/279537625_Deep_Learning_on_a_Raspberry_Pi_for_Real_Time_Face_Recognition/links/5596886c08ae5d8f3931b5c5.pdf (Accessed 2016-05-04).

[41] M. Sharifi, M. Fathy, and M. T. Mahmoudi, "A classified and comparative study of edge detection algorithms," in *Information Technology: Coding and Computing, 2002. Proceedings. International Conference on*, pp. 117–120.

[42] D. G. Lowe, "Object recognition from local scale-invariant features," in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, vol. 2. Ieee, pp. 1150–1157.
[Online]: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=790410 (Accessed 2016-05-05).

[43] K. G. Derpanis, "The Harris corner detector," *York University*, 2004.

[44] P. Azad, T. Asfour, and R. Dillmann, "Combining harris interest points and the SIFT descriptor for fast scale-invariant object recognition," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*. IEEE, pp. 4275–4280. [Online]: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5354611 (Accessed 2016-05-05).

[45] S. L. Tanimoto, "Template matching in pyramids," vol. 16, no. 4, pp. 356 – 369. [Online]: http://www.sciencedirect.com/science/article/pii/0146664X81900460

[46] L. Matthies, R. Szeliski, and T. Kanade, "Kalman filter-based algorithms for estimating depth from image sequences," in *Multisensor Fusion for Computer Vision*. Springer, 1993, pp. 87–130.

[47] A. Koschan, V. Rodehorst, and K. Spiller, "Color stereo vision using hierarchical block matching and active color illumination," in *Pattern Recognition, 1996., Proceedings of the 13th International Conference on*, vol. 1. IEEE, pp. 835–839. [Online]: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=546141 (Accessed 2016-05-05).

[48] Playstation Eye: Synchronization - NUI group community forums. [Online]: http://nuigroup.com/forums/viewthread/12445/#66879 (Accessed 2016-05-05).

[49] G. Somanath, M. V. Rohith, and C. Kambhamettu, *Advances in Visual Computing: 6th International Symposium, ISVC 2010, Las Vegas, NV, USA, November 29 – December 1, 2010, Proceedings, Part II*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 170–181. [Online]: http://dx.doi.org/10.1007/978-3-642-17274-8_17

[50] J. Suarez and R. R. Murphy, "Using the Kinect for search and rescue robotics," in *Safety, Security, and Rescue Robotics (SSRR), 2012 IEEE International Symposium on*, Nov 2012, pp. 1–2.

[51] D. Törnqvist, T. B. Schön, R. Karlsson, and F. Gustafsson, "Particle filter SLAM with high dimensional vehicle model," vol. 55, no. 4, pp. 249–266.

[52] D. Dardari, P. Closas, and P. M. Djurić, "Indoor tracking: Theory, methods, and technologies," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 4, pp. 1263–1278, April 2015.

[53] Everything ESP8266. [Online]: http://www.esp8266.com/index.php?sid=eb26510ee8c09534e1d275eb22e7a225 (Accessed 2016-05-08).

[54] P. E. Hart, N. J. Nilsson, and B. Raphael, “Correction to "a formal basis for the heuristic determination of minimum cost paths",” *SIGART Bull.*, no. 37, pp. 28–29, Dec. 1972. [Online]: http://doi.acm.org/10.1145/1056777.1056779

[55] J. Bruce and M. Veloso, “Real-time randomized path planning for robot navigation,” in *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, vol. 3, 2002, pp. 2383–2388.

[56] Project OSRM.
[Online]: http://project-osrm.org/ (Accessed 2016-05-15).

[57] P. Neis. A comparison of several routing-engines – which one is the fastest? : Neis one!
[Online]: http://neis-one.org/2011/07/comparison-routing/ (Accessed 2016-05-09).

[58] Navit bugtracker ticket #957 (osrm).
[Online]: http://trac.navit-project.org/ticket/957 (Accessed 2016-05-09).

[59] Navit - car navigation system.
[Online]: http://www.navit-project.org/ (Accessed 2016-04-25).

[60] Routing/online routers - OpenStreetMap wiki.
[Online]: https://wiki.openstreetmap.org/wiki/Routing/online_routers (Accessed 2016-05-09).

[61] en:xu3_hardware [odroid wiki].
[Online]: http://odroid.com/dokuwiki/doku.php?id=en:xu3_hardware&s[]=opencl (Accessed 2016-05-16).

[62] N. Pinckney, “Pulse-width modulation for microcontroller servo control,” *IEEE Potentials*, vol. 25, no. 1, pp. 27–29, 2006.

[63] *USER MANUAL, EZRUN Series Brushless Speed Controller*, Hobbywing.

[64] Community. (2015) i.MX6x SABRE Lite SPI Flash Recovery.
[Online]: https://www.eewiki.net/display/linuxonarm/i.MX6x+SABRE+Lite+SPI+Flash+Recovery (Accessed 2016-04-18).

[65] Community. (2015) i.MX6x SABRE Lite.
[Online]: https://www.eewiki.net/display/linuxonarm/i.MX6+SABRE+Lite (Accessed 2016-04-18).

[66] OpenCL embedded.
[Online]: https://community.freescale.com/thread/305019#comment-366131
(Accessed 2015-08-03).

[67] Rtl8188 access point install script.
[Online]: http://blog.sip2serve.com/post/48899893167/rtl8188-access-point-install-script
(Accessed 2015-01-01).

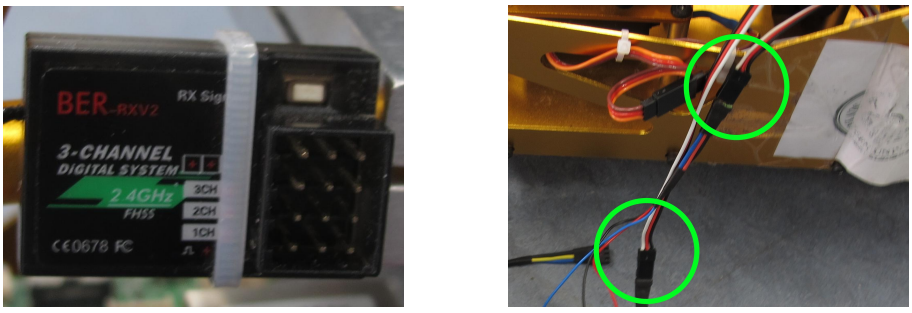
Appendix

A Motor and Servo Motor Control

We determined the cables to the remote control receiver shown in Figure 13 carried the following signals: Black is connected to ground. Red provides +6V, the white cables carry Pulse Width Modulation (PWM) signals to control the driving and servo motors. These have an active voltage level of 3.3V and a period of 20 milliseconds. The motors react to the signals as shown in table 1.

For the drive motor:		For the servo motors:	
state	duty cycle duration	state	duty cycle duration
full power backwards	0.98ms	full left	0.9ms
not powered	1.46ms	straight	1.5ms
full power forwards	2ms	full right	2.1ms

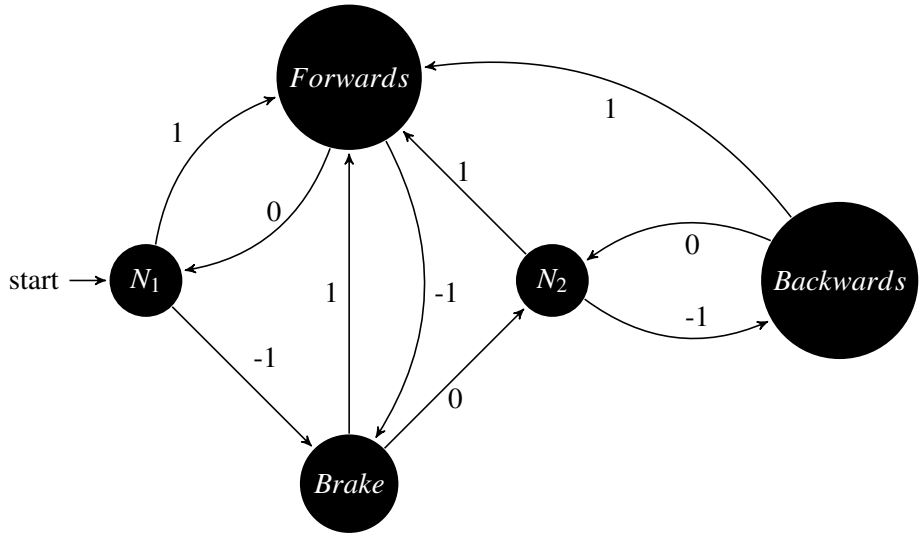
Table 1: Duration of duty cycles for servo and drive motor control. Period is 20ms. Values like these are common for servo motors[62]



(a) Receiver (b) Cables

Figure 13: Receiver (a) for remote control signals. The cable ends marked with green circles in (b) have to be attached in rows 3 and 4 with white facing left if the car has to be controlled by remote. Here they are attached to a self built adapter for the FPGA pins.

The drive motor is behind a control unit and and signals are sent to the control unit. The control unit follows a state machine. The possible state transitions are depicted in figure 14. Please note that during the initialization of the PWM the signal needs to indicate a neutral position, otherwise the input is ignored. It is possible to change the control algorithm of the controller as described in its manual [63] so that it reacts directly to the input signals, but this would mean loosing the ability to brake.



1 = Signal indicating forward motion
0 = Signal indicating no movement
-1 = Signal indicating backward motion

Figure 14: Possible state transitions with the motor control unit.
N_1 and N_2 indicate neutral positions

B Development Boards

The SABRE Lite i.MX6 board caused many setbacks in development. The bootloader was configured in a way which prevented nearly all ways to boot the board and documentation was only partially available, distributed over many different websites.

At first the SD slot had to be used for booting. The SD had to have a FAT32 partition on it. A current U-Boot for the device was compiled as described here: `https://boundarydevices.com/u-boot-on-i-mx6/`. As cross compiler it is recommended to use arm-linux-gnueabihf with the command "export CROSS_COMPILE=arm-linux-gnueabihf" executed before compiling U-Boot. The flashing procedure is described on an eewiki site [64].

On another site on eewiki [65] a step-by-step guide is presented to compile the Linux kernel for the i.MX6. Using a current kernel is important because there have been improvements on i.MX6 USB support. However some changes to the procedures desribed in the guide were neccessary. The first partition had to be formatted with FAT32, not ext2. In the U-Boot environment a variable had to be changed and saved "setenv script 6x_bootscript" and "savenev". A Debian image was used as filesystem.

B.1 Serial Communication

To utilize serial communication on the i.MX6 the boot parameters for the kernel have to be altered, so kernel messages are not sent over the serial port. This is done by changing the line "console=ttymxc1,115200n8" to "console=ttymxc0,115200n8" in the file uEnv.txt on the boot partition. Alternatively a connection from the second serial port on the SABRE Lite board would have been possible.

Serial communication was achieved via C++. A basic sample was provided by members of the LEGO car group and modified to suppress print statements. Functions were added or changed to allow timeouts, starting and stopping characters, and flushing of the serial port.

B.2 Wireless

Attempts were made to configure the i.MX6 to open a wireless hotspot or connect to a network using an Edimax EW-7811UN dongle. This proved to be difficult. Only by restarting the networking several times a connection to an unencrypted network could be established. Problems resulted from poor USB support of the i.MX6. It was decided that this connection is to unreliable.

B.3 Partial OpenCL Support

OpenCL support can be installed on the i.MX6 Sabre Lite via a package named gpu-viv-bin-mx6q provided by the manufacturer. However no OpenCV program using OpenCL could be run due to the fact that only partial OpenCL support for the i.MX6 is available, and there are no plans to change this [29]. OpenCV requires support of the OpenCL full profile [66].

B.4 Raspberry Pi 2

Because the powerful GPU of the i.MX6 board can not be utilized with OpenCV and rewriting basic image processing algorithms to utilize the OpenCL subset would be beyond the scope of the project, we decided to use a Raspberry Pi 2 as auxiliary CPU. In our benchmarks the i.MX6 Quadcore achieved only 1.2 times the performance of the Raspberry Pi 2. Using a Raspberry Pi has the additional advantage of utilizing the great driver support available. It can be used as hotspot together with an Edimax EW-7811UN dongle [67], which allows to gather data, see the status of the car while driving it via remote or while it is driving on its own and allows to update and debug code on the device without attaching cables. The Raspberry Pi also has four USB Ports which allow to attach more devices and because of its Micro USB power input it can be conveniently powered via one of the many available USB battery packs. Raspbian was chosen

as operating System for the Raspberry Pi. It is based on Debian which allows to use similar commands as on the i.MX6 and therefore reduces effort for system maintenance.

B.5 Ethernet Communication

To exchange information between the Raspberry Pi and the SABRE Lite board, we decided to use Ethernet because of its reliability and speed. Ethernet communication was achieved utilizing the Linux socket API. Wrapper functions were written to send a data structure containing relevant information from the Raspberry Pi to the i.MX6. The i.MX6 can then use the values for decision making and send commands via serial to the FPGA Board.

C OpenCV

C.1 Compiling OpenCV

Compiling OpenCV requires many development libraries. A chain of commands was worked out to compile OpenCV on the SABRE Lite, the Raspberry Pi and on PCs of project participants. A Debian based system is required, but steps should be similar on other Linux based operating systems.

The necessary commands can be seen in Listing 1. Note that the compilation time of OpenCV can be in the order of several hours on low end devices like ARM development boards.

C.2 Compiling Codes with OpenCV

The easiest solution we found to compile Code with OpenCV is the tool CMake. Inside the source code directory a file named CMakeLists.txt has to be created with the contents shown in Listing 2, replacing {Executable_Name} and {Sourcecode} with their corresponding names. For example AutonomousVehicleControl and AutonomousVehicleControl.cpp.

The code can then be compiled with the commands shown in Listing 3. CMake automatically searches for required libraries and passes them to the compiler for linking.

Listing 1 CMake

```
# Download OpenCV code
git clone --depth 1 https://github.com/Itseez/opencv/
# Execute the following line if you want to
# include the OpenCV contribution libraries.
# Including, for example, stereo image processing
git clone --depth 1 https://github.com/Itseez/opencv_contrib.git
cd opencv
mkdir release
cd release
# install necessary development libraries and tools
apt-get install python3-dev cmake build-essential pkg-config python3-
    numpy libjpeg-dev libpng12-dev libtiff5-dev libjasper-dev
    libavcodec-dev libavformat-dev libswscale-dev libv4l-dev libtbb-
    dev libfaac-dev libmp3lame-dev libopencore-amrnb-dev libopencore-
    amrwb-dev libtheora-dev libvorbis-dev libxvidcore-dev x264 v4l-
    utils doxygen unzip libeigen3-dev libqt4-dev
# be careful to adapt the -D OPENCV_EXTRA_MODULES_PATH={path}
# or to remove it,
# depending on where and if you downloaded the modules.
cmake -Wno-dev -D OPENCV_EXTRA_MODULES_PATH=../../opencv_contrib/
    modules -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/
    local -D WITH_TBB=ON -D WITH_V4L=ON -D INSTALL_C_EXAMPLES=ON -D
    INSTALL_PYTHON_EXAMPLES=ON -D BUILD_EXAMPLES=ON -D WITH_OPENGL=ON
    -D WITH_EIGEN=ON -D BUILD_DOCS=ON -D BUILD_TESTS=OFF -D
    BUILD_PERF_TESTS=OFF -D WITH_QT=ON ..
make -j 4
# Instead of the following command you can use "sudo checkinstall"
# to allow a clean and easy uninstallation as well as generating a
# debian package which can be used on other computers with similar
# CPU architecture and operating system.
sudo make install
```

Listing 2 Sample CMakeLists.txt file

```
cmake_minimum_required(VERSION 2.8)

project( {ProjectName} )
find_package( OpenCV REQUIRED )
set(THREADS_PREFER_PTHREAD_FLAG ON)
find_package(Threads REQUIRED)

add_definitions(-std=c++11)


add_executable( {Executable_Name} {Sourcecode} )

target_link_libraries( {Executable_Name} ${OpenCV_LIBS} Threads::
    Threads )
set(CMAKE_BUILD_TYPE Release)
```

Listing 3 Steps to compile a cmake project

```
mkdir release
cd release
cmake ..
# Above steps are only needed once.
# Repeat the cmake execution if you change the CMakeLists.txt
# or if you install a missing library.
# In this case remove contents of the release directory beforehand.

# Final compilation
make -j 4
# Run the generated executable
./{Executable_Name}
```

D Block Diagram

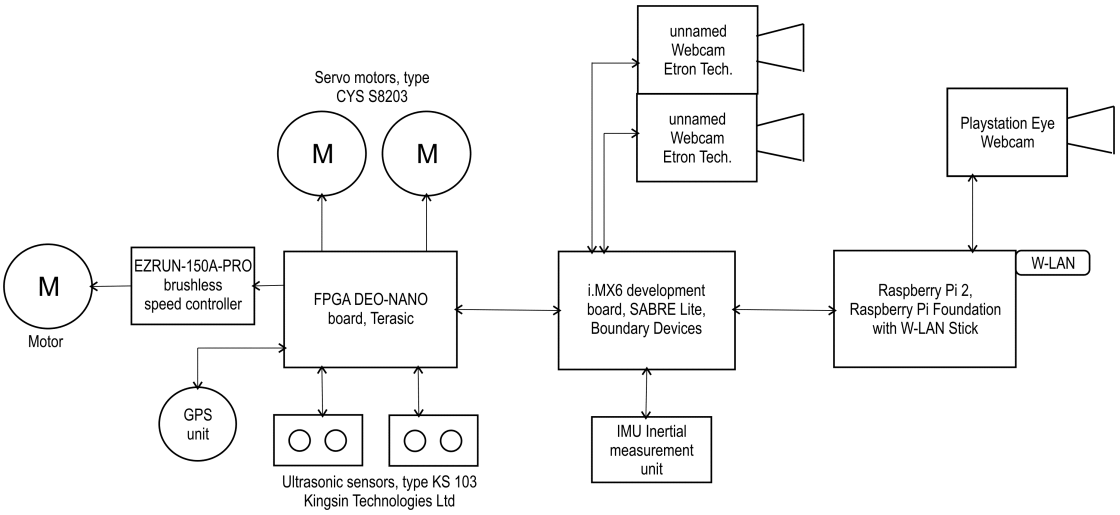


Figure 15: Block diagram of components. Arrows indicate the flow of instructions and information.

E Resources

A number of recordings and source codes were created for this project. You may contact the author under roger.roesch[at]gmail.com to obtain these free of charge.

- Sourcecode of the project
- A program to visualize tentacle pathfinding
- Recordings of the stereo camera on streets and facing obstacles
- Recordings of the top camera