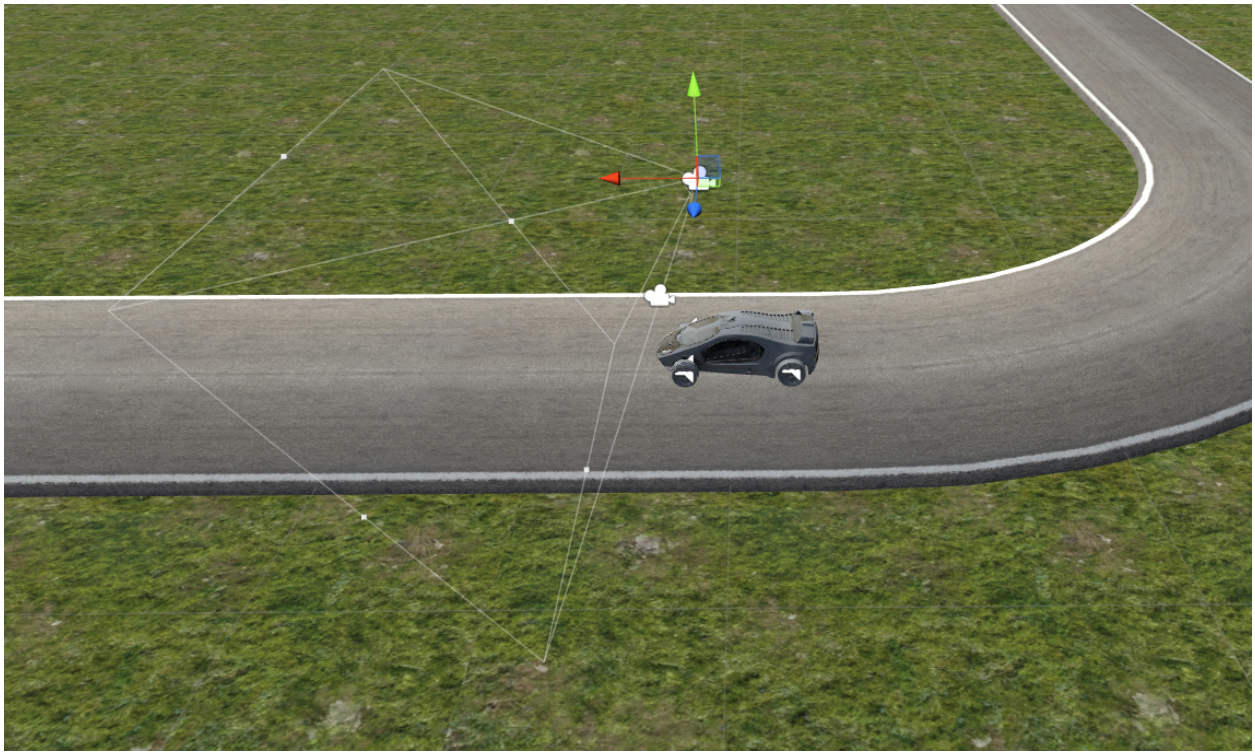
CHALMERS
UNIVERSITY OF TECHNOLOGY



A model based approach to lane detection and lane positioning using OpenCV

Bachelor's thesis in Computer Science

Daniel Posch and Jesper Rask

Department of Computer Science
CHALMERS UNIVERSITY OF TECHNOLOGY AND UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

Bachelor's thesis 2017:17

A model based approach to lane detection and lane positioning using OpenCV

Daniel Posch and Jesper Rask

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2017

A model based approach to lane detection and lane positioning using OpenCV
Daniel Posch and Jesper Rask

Supervisor: Erland Holmström, Department of Computer Science
Examiner: Peter Lundin, Departmanet of Computer Science

Bachelor's Thesis 2017:17
Department of Computer Science
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by [Chalmers University of Technology]
Gothenburg, Sweden 2017

Sammanfattning

Syftet med denna avhandling var att implementera och utveckla ett program för att kunna finna väglinger samt beräkna en förslagen rutt mellan dem, som sedan kommer att spela en viktig roll vid skapandet av en autonom miniatyrbil. I synnerhet tillhandahåller denna avhandling de första stegen för bildbehandlingen, en algoritm för vägfils-detektering och en kommunikationsmodell för att kommunicera mellan olika programspråk.

Studien har haft två huvudmål. Det första målet har varit att undersöka lämpliga algoritmer för att effektivt detektera specifik information från en bild, för att sedan berkäna en förslagen lämplig position i vägbanan, baserat på den utvunna datan. Det andra målet har varit att bestämma vilka specifikationer som krävs av kameran, för att tillfredställa den utvalda algoritmen.

Vår implementation har främst inspirerats av algoritmerna CHEVP och B-snake, men har även utformat sig med hjälp av andra modeller vars gemensamma syfte är att positionera sig inom vägfilen. Implementationen har främst utvärderats i Udacity:s spelsimulator, men också med hjälp av en modifierad radiostyrd bil. Vårt program har visat sig fungera bra i spelsimulatorn under olika förhållande såsom suddiga väglinjer, hög hastighet samt störningar. Tester på en bana med miniatyrbilen har utförts med varierade resultat.

Nyckelord: Autonoma fordon, OpenCV, vägfilspositionering, CHEVP, B-snake

A model based approach to lane detection and lane positioning using OpenCV
Daniel Posch and Jesper Rask
Department of Computer Science
Chalmers University of Technology

Abstract

The aim of this thesis was to implement and develop a computer vision based method that would play a important part in the implementation of an autonomous RC car. In particular this thesis provides the initial steps of image pre-processing, an algorithm for lane detection, position identification and a communication model.

The study was preformed with two main goals. The first goal consists of an investigation of a suitable algorithm to efficiently detect specific information from an image, to be able to act in a way based on the extracted data. The second goal was to determine what camera specifications are needed for the chosen algorithm.

Using different approaches of algorithms for detecting a path between lanes, this thesis present an implementation of the B-snake model. The program is evaluated in the Udacity game simulator, as well as on real hardware with challenging benchmarks such as lanes with hard curvature, high speed and noisy environments. The program performed well in different environments together with a limited speed. However lanes with curvature which exceeds 25 degrees has to be further developed in the future.

Keywords: Autonomous vehicles, OpenCV, lane detection, lane keeping, CHEVP, B-snake

Acknowledgements

This thesis owes its results to the help and support of many people. First of all, we would like to thank Alixander Ansari, our supervisor from Sigma Technology who proposed and were responsible for this challenging project. He provided the project group with an optimum working environment, a good mindset and essential comments. We would also like to thank our supervisor Erland Holmström for much appreciated help and support. Finally, we owe special gratitude to the project team we have been working with at Sigma Technology, Albin Falk, David Granquist, Vishnu Shyam, Paul Lazar, Andreas Jacobsson and Anders Sundkvist.

Daniel Posch and Jesper Rask, Gothenburg, June 2017

Contents

1 Introduction **1**

2 Theory **3**
2.1 Lane detection . . . 3
2.1.1 Canny/Hough Estimation of Vanishing Points algorithm . . . 3
2.1.1.1 Sectioning of the frame . . . 4
2.1.1.2 Canny edge detection . . . 4
2.1.1.3 Hough Line Transform . . . 4
2.1.1.4 Vanishing points . . . 6
2.1.1.5 Voting process . . . 7
2.1.1.6 Estimation the mid-line of road . . . 8
2.1.1.7 Drawbacks of CHEVP algorithm . . . 9
2.1.2 Kalman filter . . . 9
2.2 CUDA . . . 9

3 Method **11**
3.1 Organisation . . . 11
3.1.1 Project management . . . 11
3.2 Hardware . . . 12
3.3 Software . . . 14

4 Implementation **15**
4.0.1 Communication with the main control server . . . 15
4.1 Analysis . . . 16

5 Results, Discussion **19**
5.1 Result . . . 19
5.2 Discussion . . . 22
5.2.1 Sustainable future . . . 23

6 Conclusion **25**

7 Appendix **27**

Bibliography **29**

1
Introduction

Background The Autonomous vehicles industry are continuously evolving as companies are still competing to be the first to develop a completely autonomous vehicle. Today's implementations for the autonomous vehicle are using a variety of techniques to sense the surroundings. One of them is to utilize "Computer Vision", by implement a variety of cameras and sensors, whose purpose is to provide different algorithms with a detailed description of the surroundings.

Our project will be an element of a more extensive project, whose purpose is to create a prototype of an autonomous vehicle with a narrow AI. In order to achieve this we need to provide the vehicle with an awareness of the surroundings. To create an artificial computer vision, one tries to imitate functions that the human vision provides. By analyzing digital frequencies in detail from images, the computer can create a visual understanding and act accordingly.

Purpose The purpose of this project is to investigate different lane keeping models and implement the most suitable with the input of a RGB camera. The lane keeping algorithm should be able to find curved lanes, be able to communicate with other program's as well as give an estimated direction angle to make the car position itself between two lines. The goal is to create a narrow AI that's capable of recreate the function of an human's vision. The obtained stream of frames from the camera is analyzed and processed, with the intention of extracting certain information of the surroundings. The information should be used as an input to the lane detection algorithm, to create an estimated direction angle.

Delimitation The main goal of the project is to investigate existing solutions regarding lane detection and lane keeping and implement an own solution that should be able to detect road lines on slightly curved roads or straight roads, and be able to calculate an estimated direction angle. The lane keeping algorithm will only manage straight and curved lines with good conditions such as marked lanes, beneficial light and limited noise.

Goals for this thesis

- Investigate a suitable algorithm for lane keeping based on our initial goals.
- Determine which camera that is suitable for capturing data while moving.
- Determine how should the main neural network and the lane detection algorithm, written in different languages efficiently communicate in real time.

2

Theory

This chapter present the background of lane fitting and different communication models that this thesis studies, including the general algorithms, software structure and the disadvantages. Furthermore this chapter provides a review of the previous work where computer vision techniques are integrated in today's technology. This chapter will also set a scope for the theoretical framework used throughout the project.

2.1 Lane detection

Developing an autonomous driving system requires a robust lane detection algorithm and is one fundamental task for the autonomous vehicle. It is critical that the lane detection algorithm is robust against different noises, such as changing light conditions, other objects and road curvature [1]. The process of the lane keeping algorithm is shown in fig 2.1. When the algorithm receives the input image, it get gray scaled and broken down into parts to make it easier to detect edges in the image. The first steps of the algorithm is the Canny/Hough Estimation of Vanishing Points (CHEVP), which consists of the following procedure. The image will be pre-treated such as gray scaled and blurred, making the next operations more efficient and also to reduce different noise. Moreover the edges in the image will be extracted by the Canny Edge algorithm described in section 2.1.1.2. The pre-treated image will be the input to the Hough Line transform described in section 2.1.1.3. The Hough Line Transform will extract the actual features from the image and the output will be filtered. The lines extracted from Hough line transform will be a part of the next procedure of this algorithm, the voting process.

2.1.1 Canny/Hough Estimation of Vanishing Points algorithm

As a result of perspective projection, the road lines should intersect at an arbitrary point above the picture, something which is called vanishing points which is described in section 2.1.1.4. The first vanishing point are used as the middle of the road estimation for each section [2]. CHEVP is an algorithm developed to detect the most parallel lines in a vertical direction of a frame. These parallel lines are used to find the intersecting points.

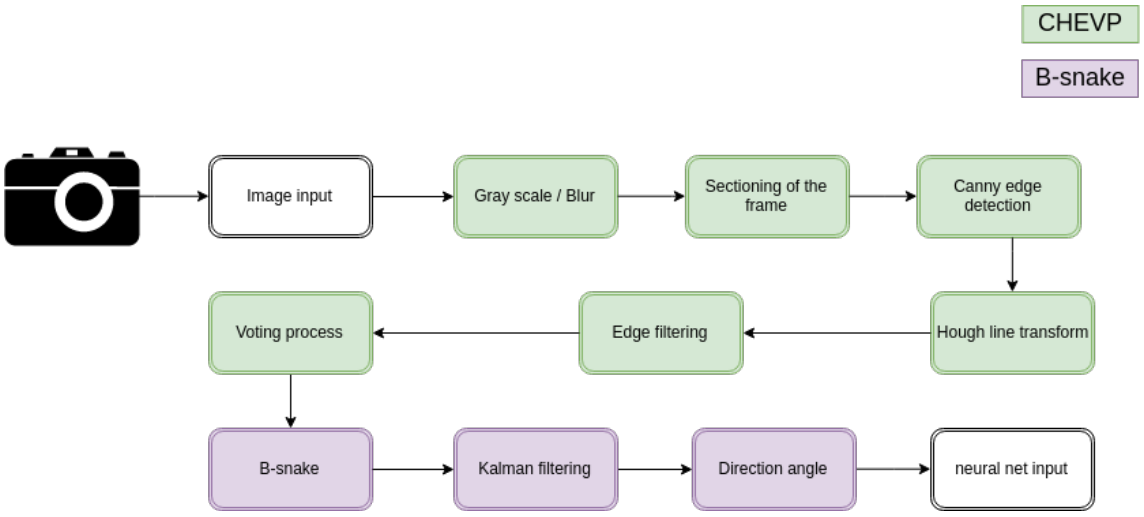


Figure 2.1: Lane detection and positioning process

2.1.1.1 Sectioning of the frame

Sectioning of the frame is something that is widely used in CHEVP for lane detection. The sectioning of the frame will not only reduce the processing power needed for the algorithm, but is essential for estimating the middle point of the road for each section of the frame. The amount of sections vary, depending on the purpose of the algorithm. The different sections are defined as one part of the input frame, and the computation is carried out on the different sections instead of the complete frame, which will result in a more effective algorithm.

2.1.1.2 Canny edge detection

John Canny presented in 1983 his "Canny Edge Detector" whose purpose is to sort out lines by analyzing step changes in a signal that contains additive white Gaussian noise. [3] . The Gaussian filter is translated by the principal of projection from a two dimensional edge [1] to an one dimensional step [2]. The step change is used as an input to the Gaussian function, which locates the minima and maxima values, although only the maxima values are used, since those are representing the step edges[3]. The edges are then found using the maxima values and compare the difference in pixel values next to the actual pixel. When an edge is found the pixels giving the edge are given a value of 127 and the rest of the pixels in the frame that are processed are given a value of 0 [4].

2.1.1.3 Hough Line Transform

The most elementary form of Hough Transform was presented in 1962, and was designed to detect lines and arcs in photographs.The technique is used to present the mathematical form of a geometric shape using it's boundary points.[5] There

[1] Two dimensional edge is a clear difference in pixel values between two neighbouring pixels. See figure 2.2

[2] A one dimensional step refers to a step change in a step funcion

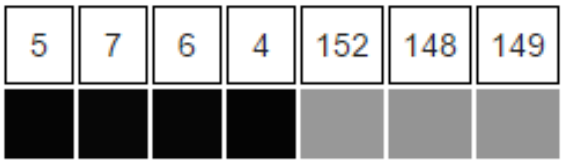


Figure 2.2: An edge is found between pixel 4 with value of 4 and pixel 5 with value of 152

are two forms of Hough Transform, where the purpose is the same but the analytic description of a feature differs. The fundamentals of the Classic Hough Transform requires that the features of an shape can be presented in a parametic form. This type of technique is most commonly used to detect lines, ellipses, and circles. The generalized Hough Transform is a further developed algorithm, used to detect different arbitrary objects, using the principle of template matching [6].

This is the implementation provided by the OpenCV libraries, and can be used to find straight lines, circles. A line is represented by the straight line equation

$$y = m * x + c$$

and in the parametic form as

$$\rho = xcos(\theta) + ysin(\theta)$$

In the parametic form ρ is defined as the lines distance from the origin, and the θ is the inclination related to the normal. This equation defines the transformation between the (x, y) space and the (ρ, θ) space, shown in figure 2.3. Therefore any line can be transformed in the terms of (ρ and θ) [5].

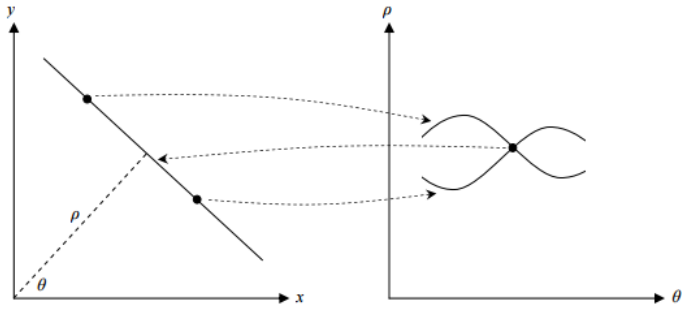


Figure 2.3: the (x, y) space and the (ρ, θ) space (D. Antolovic, "Review of the Hough Transform Method, With an Implementa- tion of the Fast Hough Variant for Line Detection.")

The points in the image space are defined as curves in the parameter space, and the intersection of the sine curves are represented by the line in the (x, y) space. The Hough Transform technique is using a voting process by creating a 2D accumulator which consists of the two parameters (ρ and θ). The procedure iterates over all the

pixels in the image, and votes for each (ρ and θ) cell if there is enough evidence of a straight line at this particular pixel. After iterating, the cells with the local maxima in the accumulator are most likely to be straight lines, and the distance from origin will be the voted ρ, while the angle will be the θ [7].

2.1.1.4 Vanishing points

The first step of determine the vanishing points is to find the most parallel edge points in a line of the section. The most parallel edge points are gained by voting for the intersections of any pair of lines in each section and the set of lines that receives the most votes are used as the the vanishing point in that particular section [2].

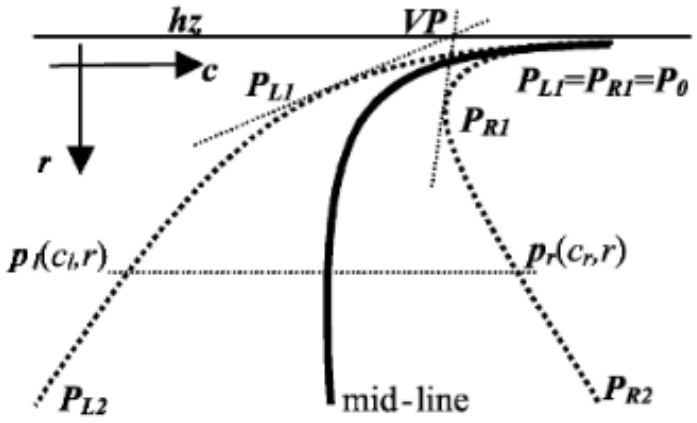


Figure 2.4: Important points in the calculation of vanishing points (Y. Wang, et.al "Lane detection and tracking using B- Snake")

To be able to calculate the vanishing point for each section of a frame, x and y values from the intersecting points of the lines gained by the Canny Edge and Hough Line are used to calculate an intersecting point in a horizontal plane. In figure 2.4 the 2 points P_{R1} and P_{L1} are the points with most votes as most parallel points who are then used to calculate the intersecting point VP. [2] The formula used to calculate the vanishing points for each section can be estimated by:

$$x_{vp} = \frac{c_{L1} - c_{R1}}{m_{L1} - m_{R1}}$$

$$y_{vp} = m_{R1} * x_{vp} + c_{R1}$$

where

$$VP = \{x_{vp}, y_{vp}\}$$

This operation is done for every section of the image. The vanishing point of each section is representing the point closest to the horizon, together with most votes as two parallel lines.

2.1.1.5 Voting process

After sectioning the frame and the Hough transform are applied for each section of the frame, the voting process is held, to accommodate the change in the road. The following procedure is being carried out on the different segments. The Hough transform will expose every detected line from the image. The detected lines are paired with each and every other line detected in the current segment. The paired lines intersection point is participating in a voting process, in order to find the vanishing point for each section with the strongest support. The voting process is weighted by the sum of paired lines, and the votes are summed for detecting a candidate for the vanishing line [2].

2.1.1.6 Estimation the mid-line of road

The set of lines vanishing points are getting determined from are assumed to be the road lines in the picture. If a section with no road lines are processed, the vanishing point estimation will go on to the next section of the frame until the road lines are found [2]. The points which are gained from the equations in section 2.1.1.4 can be used to find the middle points of the road in each section. The x_{vp} can be used as the middle point (x_m) and the section height as the y_m-value. That way the points get the in the middle of the row, and the car have some time to adjust its position[2].

Figure 2.5 show how the two lines L_1 and L_2 are found in section one by the algorithm in section 2.1.1.4 and two parallel points P_{l2} and P_{r2} are found by voting for the most parallel pair of points [2]. The found points P_{l2} and P_{r2} are then used and VP is estimated by the equation in section 2.1.1.4 to later be used in getting the middle point for that section $P_{mx}(c_{mx}, r_{mx})$ where $m = middlepoints$ and $x = sectionnumber$.

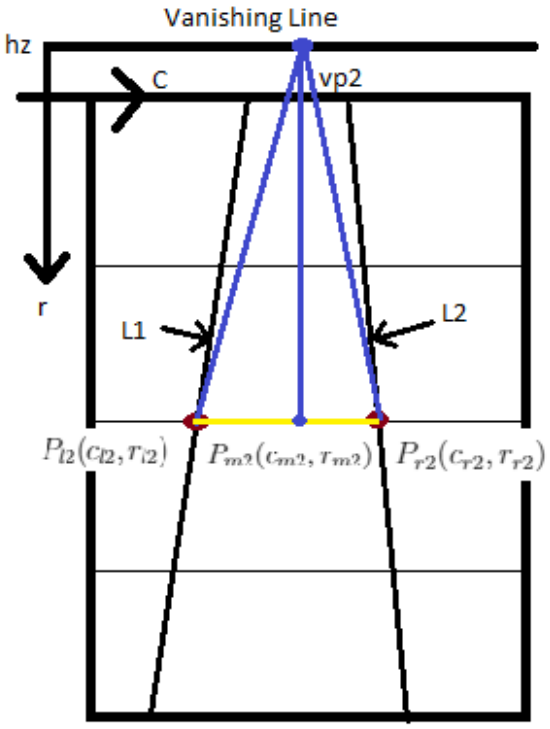


Figure 2.5: Finding midpoints in every section

After middle points for all sections of the image are found the B-snake can be drawn. To create the B-snake, all the estimated middle points for each section are connected [2].

2.1.1.7 Drawbacks of CHEVP algorithm

CHEVP is in most areas a stable algorithm to apply for lane keeping. However, the use of straight line parameters in Hough space, does not handle hard curvature. [2] Something which is solved in Generalized Hough's transform where arc's, circles etc. can be found as well [5].

2.1.2 Kalman filter

The Kalman filter is a tool utilized to estimate the values of a process, in order to minimize the mean squared error. The filter consists of a set of mathematical equations that clean up the data by take the past, present and future states in account to project the best estimated value from noisy data [8].
To simplify the Kalman filter algorithm, the state can be estimated with:

$$X_k = K_k * Z_k + (1 - K_k) * K_{k-1}$$

The X_k represent the current estimation and it is calculated by taking the kalman gain K_k and the measured value Z_k, together with the previous estimation of the signal. The only unknown value in this equation is the Kalman gain K_k, which is calculated for each consequent state [9].

2.2 CUDA

In recent years the development and interest in neural networks have been increasing rapidly. [10] The most likely reason for the change in interest came out in 2007 when Nvidia realeased their new CUDA (Compute Unified Device Architecture) platform[11]. With the release of CUDA in their GPU's (Graphical Processing Unit) the possibility of massively parallel processing with support of thousands of active threads [12]. To make use of the computing power in a GPU the need of a programming model that can match the level of parallelism of the GPU is necessary, something that CUDA provide. The reason why CUDA is more efficient is that it uses kernels. The kernel are executed by a grid of thread blocks in the GPU which have the ability to collaborate with each other [12]. CUDA has played a big part in recent years development in computer vision and lane detection algorithms since the frames are mostly processed by the GPU.

3
Method

3.1 Organisation

This project has been one part of a bigger project, to build an autonomous RC car which made the initial planning bigger than expected, since parts of the project is not included in the thesis. The group has included both product development students who has provided us with guidelines of what different scenarios the car was supposed to manage. The other parts of the group are two mechatronic students as well as two other computer science students, who has taken care of building the car, steering algorithms as well as decision making.

3.1.1 Project management

As mentioned one team of this project was working as the project management team. The management team had the discipline of controlling, planning and merge the different teams towards a common goal. The project had a defined beginning and end, and was constrained with unique goals and a budget. One fundamental task for the product management team was to define functions for each and every team, ranked by the importance to meet the end objectives.

Scrum Throughout the project we followed the agile methodology Scrum, which is a control process that facilitates the structuring of complex software development. In the beginning of the project we listed the requirements (User cases) of the car, which where supposed to work as business needs in the following procedure. The requirements where then ranked by level of importance in consideration of making the project move forward. The requirements were then further divided into backlog items. Moreover, the backlog items where estimated in work hours.

The work were carried out in two week iterations, called sprints in the agile methodology. A sprint is a set of period of time, were specific backlog items has to be accomplished. The sprint begins with a planning meeting, where the project group decides what different backlog items that should be completed during the sprint. Using this methodology the development team acquire an overview of the workflow. During the sprint, each day starts with a stand up meeting with the purpose of discuss problems and brainstorm solutions. In the end of each sprint a retrospective meeting occured, which were supposed to be relatively short. During this meeting

the different team members had the opportunity to forward their thoughts about the current sprint, such as good and bad thoughts about the previous sprint, and also how it could be improved for the coming sprint.

Trello Trello is a visual organisation tool used in software development to easier manage projects. In this thesis Trello has been used as a scrum board to manage sprints and also to create a burn down chart to keep track of the projects progress. Trello has integrated cards that could be used with checklists, this feature was used in the project to get a better understanding of which steps where needed to accomplish the goal of that particular backlog item in the sprint.

Slack Slack was the communication service we used during the project, by reason of it fulfilled our demands of what a communication tool should include. The reasons we chose Slack as the communication tool was a consequence of how the project was structured. As mentioned in the section 1, our thesis is a part of a more extensive project, which consists of four different teams. Creating different communication channels is one of the Slack's features, which made it possible to separate the different discussions, but still maintain the flexibility of remain involved in the different teams work. In addition to the communication channels, we integrated a program together with slack that made it possible to track the worked hours for every member of the team.

3.2 Hardware

Our solution of lane keeping is a part of a larger system, whose goal is to guide an RC car autonomously. The RC car is equipped with basic components such as an engine, wheel servos and batteries. The RC car were further extended with an computer board, which were supposed to handle the steering and propulsion. The Nvidia Jetson TX2 is handling the heavy computations such as the lane keeping algorithm and the neural network for decision making.

Micro controller The micro controller which was primarily used to perform this thesis has been Nvidia's Jetson TX2, a micro controller used in various projects regarding autonomous RC cars. The TX2 where used as the main controller for the heavy computations i.e lane keeping and the neural network. The Jetson TX2 is a high-performance single board computer, designed for real time systems such as self-driving cars that requires a strong CPU and GPU. The Jetson provides a strong processing power without using high amount of energy, which is optimal for our purpose. Specifications for Jetson TX2 can be found in table 3.1.

Camera The camera position was a critical decision of our project, since many factors are dependent on the camera field of view. Car paths are often unpredictable and bumpy, while the video stream must be consistent. One of the requirements for the camera mounting was that the lanes need to be detected frequently, but also be able to detect obstacles further ahead. However, this was a difficult task due to

Name	Nvidia Jetson TX2
GPU	Nvidia Pascal, 246 CUDA cores
CPU	HMP Dual Denver 2
Memory	8 GB 128 bit LPDDR4
Storage	32 GB eMMC
OS	Ubuntu 16.04

Table 3.1: Specifications of Nvidia's Jetson TX2

the fact that we used one camera instead of multiple. The camera we used in this project was a Intel Realsense R200, which had a 70 degree field of view. However this could be considered as slightly to narrow for the purpose of the camera.

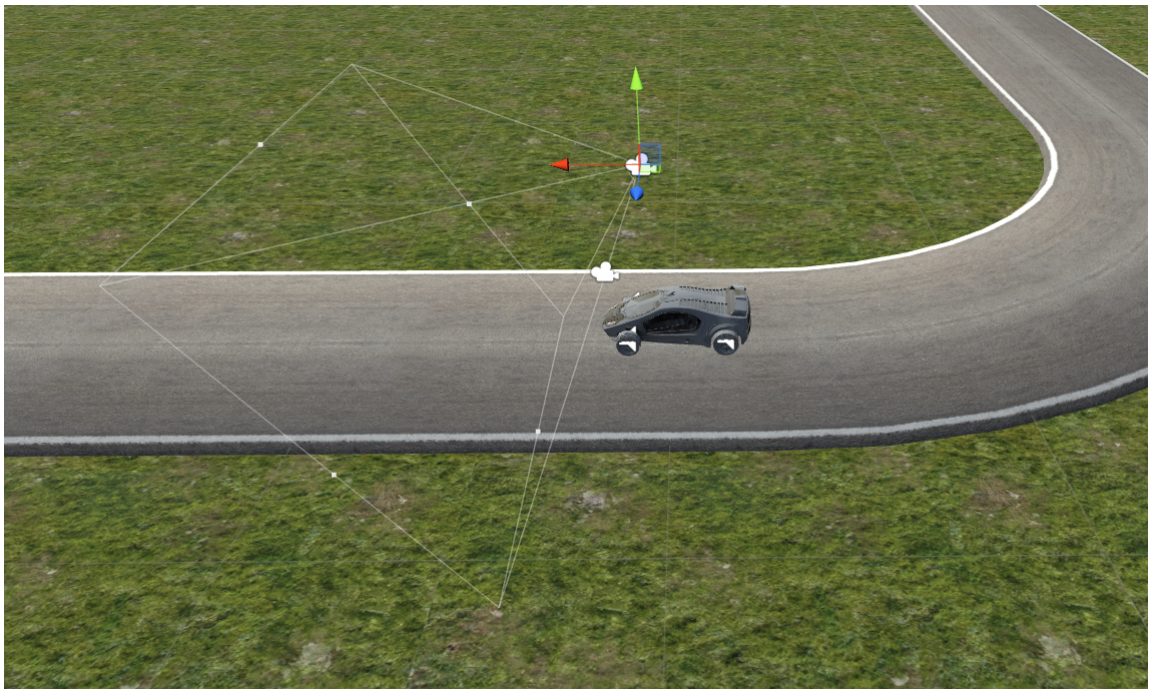


Figure 3.1: Camera field of view - simulated

Name	Intel realsense R200
Field of view	70 degree, (Cone)
Aspect ratio	16:9
Frame rate	30 fps
Active pixels	1920 x 1080
Optional	Depth sense
Range	3.5 meters

Table 3.2: Specifications of intel realsense R200

The road detection algorithm is dividing the frames into segments as described in

2.1.1.1. The idea was to make different calculations depending on how the road is curved. The lane keeping algorithm itself is structured in a way that the width and height is dynamic, but the size of the sections is adjusted manually. However the result of the algorithm is heavily dependent on the cameras field of view, and the size of the sections will affect the outcome.

3.3 Software

Conda is an open source package management system and environment management system for installing multiple versions of software packages and their dependencies and switching easily between them.

Conda All software that is written to receive images and video streams are written in Python. Since Conda is a package management system that only includes Python and packages they depend on, this software works as a package manager which makes it easier to work with different packages, which normally does not work well together. Conda has primarily been used to run programs with several different dependencies.

OpenCV OpenCV (Open Source Computer Vision Library) is a library mainly used for computer vision, machine learning and image processing. OpenCV is developed to optimize the computational performance, to provide algorithms with real-time systems as a center point. The library is adapted to provide computational efficiency for real time systems. The library is written in C++, but have bindings and wrappers in several languages.

Our implementation of the lane detection algorithm is written with OpenCV libraries. This library contain both Canny Edge and Hough's Transform as functions, which makes OpenCV an simple choice since both these functions are a part of the CHEVP algorithm described in section 2.1.1.

Pair programming Throughout the project we chose to mainly focus on pair programming. The reason behind it was that it has been a standard procedure in earlier projects. Pair programming enables us to ensure that the math in the algorithms are correct. Moreover the pair programming have been a way of realizing the problem from two perspectives rather than just a single one, which have led to a faster development and probably more efficient solutions.

4

Implementation

Due to the small duration of this project, as well as a small development team, a fast paced development was required to achieve our goal and finish the tasks regarding lane keeping which each separately is complex and somewhat unknown territory. Due to this aspects, our implementation strategy was set in the direction of implement and improve existing work, rather than start from scratch.

The implementation of this thesis was based on another project carried out in Lane Detection. Our implementation of the CHEVP algorithm was mainly based on the article by Y. Wang et.al "Lane detection and tracking using B-Snake"[2]. The implementation follows the steps that is carefully explained in the theory section 2.1

4.0.1 Communication with the main control server

The control server is developed by the other software team. One of the control server task is to read the input video stream, encode each frame into text, to make it transferable through Ethernet. Additionally the main control server created by the other software team, should send the encoded frame to the lane detection program, and wait for an answer.

Sockets ethernet TCP The communication between the different programs in this project was designed as a client-server model. The communication architecture consists of the control server working as the server, and our lane keeping program as a client. The data is transferred between the different programs through the server, using the transmission control protocol(TCP) with sockets. Our lane keeping program is defined as a client that manages the frame calculations, while the control server manages the clients and stores the data received from the clients. The control server listen to a predefined port, and is identified by it's port number and IP-address. The server is constantly listening for clients that are trying to connect via the socket, and once it find a new client candidate, the handshake procedure is carried out and the connection is established. If the handshake was successfully, data can be sent back and fourth.

Decode / Encode The server process reads the video stream from the Intel real sense camera with the intention of sending each frame to the lane positioning client. To effectively send large amount of video frames to another process in real time, we used the Base64 binary-to-text encoding schemes. The binary data is translated into a representation of an ASCII string. The main server encode the frame and

send the encoded string via TCP, and the client receive the frame represented by an ASCII string. The client process decodes the message and translates the ASCII string back to binary data, in this case a matrix.

4.1 Analysis

The purpose of this project was to investigate existing solutions and implement a lane detection system that should be able to detect road lines on slightly curved roads or straight roads, and be able to calculate an estimated direction angle. Furthermore, the algorithm should be integrated with a neural network which will not be implemented by us, together creating a simple AI that's capable of taking simple decisions based on the input image.

Lane detection models

In the beginning of our project we reviewed many different algorithms to find the most suitable for our purpose.

In the starting process of this project we looked at two approaches to achieve lane keeping. Our initial idea was to use a neural network to detect lanes and predict a direction angle for the car to aim for. The second idea was to use the OpenCV library to find lines and then calculate a B-snake for the car to follow. However, we found that using a neural network has not been as successful yet and is less documented.Therefore we decided to continued our work with OpenCV, mainly because it provided functions for Canny Edge and Hough Transform which we found the best algorithms to use in our implementation of lane keeping. We found that the B-snake model, that was an improved lane positioning algorithm compared to the earlier presented ones and had solved different noise problems, would be a good starting point. We implemented this algorithm even though we were aware of it's different disadvantages. One of the problem with the b-snake model, is that some challenging road scenarios is not yet solved, such as splitting lanes, hard lane curvature and merging lanes.

The CHEVP is robust against shadows, illumination variations and different noise, which is one problem that have characterized the earlier developed lane detection algorithms. The B-snake algorithm assume that the road have two parallel lane markings, to be able to project an intersecting point. The B-snake model describes a wider range of lane structures compared to other lane detection algorithms, due to the use of parallel knowledge of the road markings.

The input image is resized to a frame of size 640 by 360 pixels, and converted into gray scale. If we use the RGBA image, some of the image processing operation would work on a single color channel at a time, which could result in some operations being carried out on the four different image planes, which would result in a heavy loaded GPU. By gray scaling the image we ensure that the operations is only being made on

one image plane, which save 75 percent of the computation power for this operations.

The gray scaling will remove different information from the frame, that in some cases could be necessary for image processing, such as different colors. However, since our algorithm does use the color difference in the picture, the gray scaled frame works better in our user case. To detect and extract the edges, a Gaussian filter and a Canny edge detector is applied to the gray scaled image.

As a result of an improved lane detection algorithm, new problems where discovered. As mentioned above, this model are based on two parallel markings on the road. This is not a solution for countries that are using unpainted roads, and this problem is yet to be solved in the future using a different model or use a combination of models. However, the B-snake model was the most suitable choice for the user cases presented in this thesis.

Communication

As mentioned the other software team created two python programs, one neural network and also a python main server. Therefore we needed a solution to communicate between the processes in different programming languages. Several options where examined and some of them where tested. The most relevant ones where Boost:Python which creates modules from a c++ function to python that can later be used in python by importing the module. The second option where SWIG which has a similar functionality and the third where using sockets.

Boost:Python where the first option we examined, since it was one of the most common solutions for similar problems. Even though it was well documented, the dependencies in Boost:Python are not, and after a lot of configuration, we decided to shift direction to SWIG.

Similar to Boost:Python, SWIG had a lot of dependencies to configure, one of them where to make modules for every OpenCV data type and function we used, which we found it too troublesome since we are using several of them. Trying two module solutions which did not accomplish the result we where hoping for, led us to sockets. Sockets made it possible for the c++ process to communicate with the process written in python via TCP/IP using text strings. The python process encodes an incoming image into base64, which is a string format and then sends it to the receiver which in this case is the lane keeping process. When received, the lane keeping process decodes the base64 string back to image format, which can then be used by the lane keeping process. This communication can be used in both direction. After the calculations in the lane keeping process, an angle for the car to aim for will be sent via the socket back to the python process.

The basic idea was to write all the different modules for this project in C++. In the middle of the project the other software group decided to write the neural network

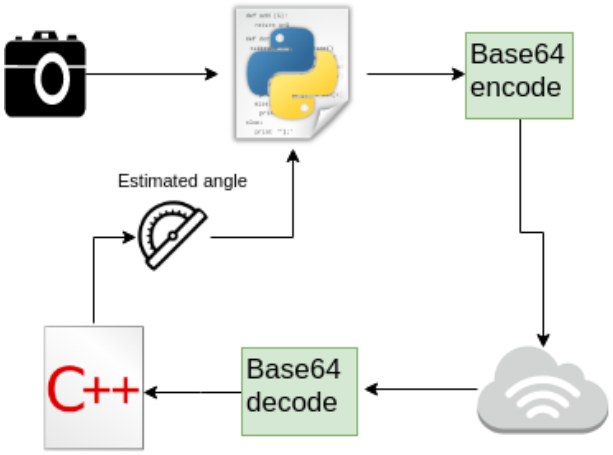


Figure 4.1: Communication scheme between the python server and the algorithm program

in the language python. As a result of the change of language we had to figure out a way to transfer and receive data in real time. The different modules are written in different processes, we needed a fast, robust and efficient solution for sending and receiving data.

The TCP protocol guarantee that every data package is received by the client, which was essential for our implementation. Using the different solutions TCP, UDS (unix domain sockets) and PIPE on a single CPU 3.3GHz gave the result.

Method	Average latency	Average throughput
TCP	6 us	253000 msg/s
UDS	2 us	1700000 msg/s
PIPE	2 us	1680000 msg/s

Table 4.1: Comparison between different Internet Protocols

5
Results, Discussion

5.1 Result

All our goals with this project was met. The lane keeping program can find lines in a vertical direction, gives a direction angle for the car to aim for, the program can communicate with the main program via sockets in a suitable speed for our purpose and a camera which can detect lanes while moving where investigated and implemented.

Lane Keeping

The lane keeping program currently works with slow velocity, this could be either because of the frame rate of the camera being too slow or the Jetson TX2 not being able to handle all the different programs running while driving. Both reasons has in the past been a bottleneck for the autonomous industry, where computational heavy algorithms has limited the development in the area of autonomous vehicles.

This project present an algorithm that is able to guide a vehicle on the road, with the use of a direction angle, calculated from the road lines. The program receives an image in base64 format and translates it to a matrix. Calculations for vanishing points, lines and middle points estimations are made on the matrix, which are used to then calculate the direction angle that makes the car stabilize in the lane which are then sent via TCP/IP in string format.

Due to Probabilistic Hough Transform mainly being used to detect straight lines and the cars turning radius being limited to 30 degrees, there is no possibility to try wider turns in reality. However the considerable bottleneck for the program is the camera, due to the low turning radius of the car. The simulations has shown that the algorithm can handle turns higher than 30 degrees, when the turning radius of the car is increased.

The RealSense R200 has a 70 degree field of view which has shown in test not to be able to detect both lines in a lane while turning which is crucial for the algorithm to work. To fix this issue we planned to get a wide angle camera just for lane keeping purposes with a 110 degree field of view, something that has been proven to work when we tried using a GoPro with a 110 degree field of view. Using a 70 degree camera has limited the turning radius to 20 degrees curves.

Figure 5.1: 70 vs 110 degree field of view

The other software team used a simulator to train their neural network, developed by Udacity. This software was essential for our test scenarios, since we were able to decide the camera placement on the car, define the amount of noise in the environment and also how the lines should be painted. A simulated frame of size 640 by 360 pixels, with minimal amount of noise is shown in the picture below.

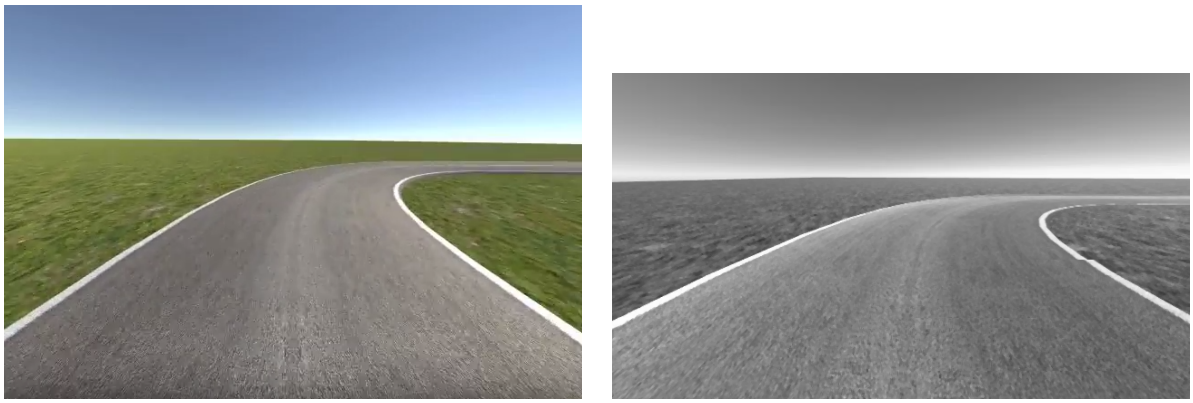


Figure 5.2: The frame given as input to the algorithm. GrayScaled, resized and blurred for less noise

The Gaussian filter removes unwanted noise, while the Canny edge detector generates a mask, that is representing the actual edges on the image. The result of this is shown in figure 5.3.

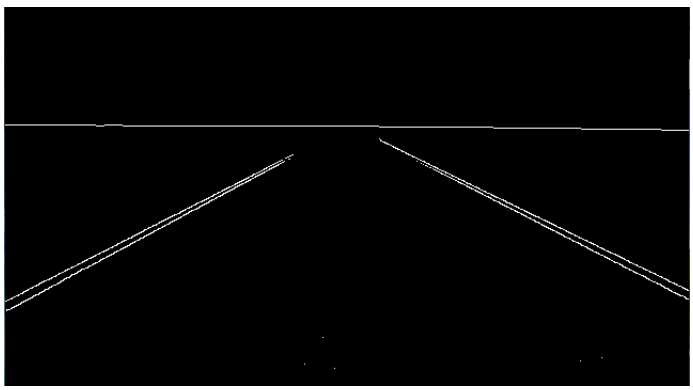


Figure 5.3: Edges found in the picture

The algorithm is applied on the pre-processed edge frame, and the outcome is the estimated direction angle. The straight arrow in the picture 5.4, is the cars actual position. The curved snake is the estimated direction, created to project the cars destination between the two white lines. We tested our algorithm on a simulated car game, but also on real hardware. However, the algorithm was only capable of driving in 10 mph and the curves could not exceed 25 degrees using the simulator. The main factor for the non-achieved results was unplanned work that had to be prioritized for the general project but also the reaction time of process that was controlling the steering.

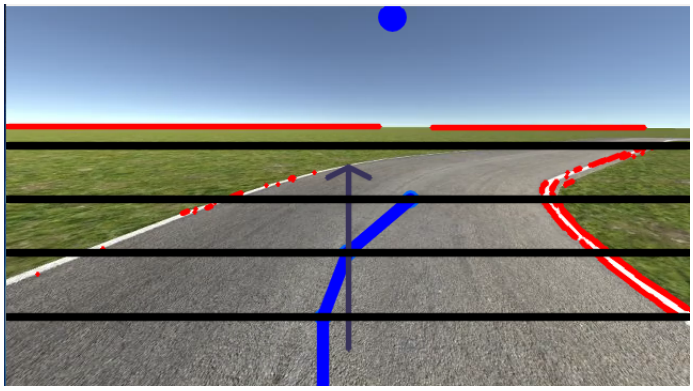


Figure 5.4: Output image

Communication using TCP

The UDS soulution is performing faster than the rest of the solutions shown in table 4.1, since it has a faster inter-process communication mechanism. The speed is increased by the knowledge of that the processes is executing on the same system, which decrease the amount of checks and routing operations. Although the performance of TCP was slower than the other methods, we chose this protocol. The performance of the TCP turned out to be more than enough for the desired result.

5.2 Discussion

This chapter will contain reflections and our conclusions of the result. We will also present our suggestions for further development. Moreover we will briefly discuss the work flow and the influence of the agile work methodology.

Companies in autonomous vehicles has switched approach from lane keeping using Hough Transform and Canny Edge as a foundation to a neural network implementation. This change could be because of the lack of result in the past years, but also because of the advancements in GPU accelerated CUDA based GPU's which brought neural network's back into the computer vision question some year's ago. The CUDA based GPU's made it possible to make calculations in computer vision based neural networks up to eight times faster.

The communication turned out to be a time consuming part of the project. The initial plan among the two different software teams, was to write the two programs in c++, to minimize the communication work. About halfway into the project, the other software team abandoned that plan for compability reasons, which created other communication problems between the different programs.

Throughout the project we focused on having virtual machines with the same dependencies to facilitate development for the software developers. This did not work as well as expected. The thought where to build projects in the virtual machines, then compile and run the code on the TX2, something that needed a software written in CUDA to compile, which we had no time to put into developing it. Therefore, we needed to continue the development at our own computers and installing dependencies for every computer used in the project, this was something that took a lot of extra time. Primarily because there where dependencies that had several steps to be completed but also some dependencies that did not work for some combinations of libraries. This situation made us develop on one computer instead of the TX2 meanwhile we solved the dependency problems, which in turn made us to have two repositories to work in, one for the TX2 and one for the computer that where able to compile the lane keeping program.

Camera

Our algorithm needs two detected lines to be able to localize itself, which was one of the fundamentals for the b-snake algorithm. Therefore the camera influences the system heavily. For further development of the lane detection algorithm, there are a few parts of the system that would improve the algorithm drastically. If we would have continued our initial though, with the use of a 110 degree field of view camera instead of one with a 70 degree field of view, the algorithm would be less vulnerable for the curved lanes. Moreover, the algorithm could be further improved by handling more complex environments. One way to achieve this is to implement the Generalized Hough transform. However the Generalized Hough Transform is noting

OpenCV libraries provides yet.

The current lane keeping program have shown to be working as the CHEVP algorithm is expected to work. However the CHEVP algorithm does only work when two lines in the road are detected which makes the program rely on that the frame always having two lines otherwise car will keep going in the same direction until two lines are found in the image. To prevent this, one could make the program follow a single line in the image, or at least have the option for a backup plan. This would make the program more reliable as a lane keeping program.

5.2.1 Sustainable future

A lane keeping algorithm does not only make our cars more efficient and safer, it could also become a matter of improving our environment. With a lane keeping algorithm one can improve the planning of the route for the car further on the road compared to an human, what speed to hold as well as which angle that is the most efficient to hold while turning.

In recent years software development in cars has accelerated immensely. This has resulted in car manufacturers hardware to become more potent. Since a lane keeping algorithm requires a lot of computing power the system need to become more efficient, both in power consumption and computing power available for the system to make the calculations properly. This could generate in more effective batteries for the systems to make them more reliable and operate longer. These improvements can later be used in batteries for cars.

6
Conclusion

A lane detection algorithm using the b-snake model together with a communication model using Ethernet has been presented. Different conclusions will be presented:

- The CHEVP algorithm has shown good results. Although if two lines does not get detected in each segment of the frame, the algorithm will not be able to calculate a estimated direction angle.
- To improve the lane keeping one could implement a solution that can create a B-snake even though only one line is detected. This would give a more stable algorithm, that could handle a field of view of 70 degrees or a lane without mid line.
- The calculations of the B-snake have proven to work well with two detected lines, and filtering false values has been managed as well as big changes in the B-snake calculations has been prevented using Kalman filter.
- The b-snake model, together with the rest of the lane detecion models, requires a detailed representation of the road. A lane detection algorithm should receive input from a camera with a wide angle field of view. Preferably, the camera should not lose any lanes while taking a sharp curve.
- The lane keeping program requires a camera with high frames rates to work properly. To manage high frame rates, high end hardware are needed to make the heavy computations.
- To communicate between programs via tcp using sockets performs adequate, assuming the amount of data is relatively small. The data arrives between the programs with almost a non-existent delay, and with no package lost. Using the UDS would increase the communication performance in general, however it would not affect the performance of the whole system markedly.
- A lane keeping system should not be reliant on a camera only, but should also be supplemented by other software and hardware which is the approach applied in today's developed autonomous vehicles.
- A cross functional team do address the problems more quickly, because the different members of each field know the most effective way of handling the resources in each part of the project. However, working in a cross-functional teams increase the difficulty in decision making and requires an organized working structure, which is in general time consuming.
- The lane positioning was achieved by using the b-snake model and performed in a satisfactory manner under good conditions, such as 2 lines detected in each frame, small noise, light curves and a speed not exceeding 3mph.

The algorithm CHEVP needs to detect two lines in the image to be able to estimate a middle of the road, a case that is not always relevant on all roads. This makes the algorithm suitable as a sub-part of a lane keeping system and not an algorithm to be used by itself.

7
Appendix

Glossary

B-Snake An esitmation of the middle of the road.

Intersection point The intersection point is the (x, y) of the two combined lines generated using Hough transform.

Image space A space is a collection of all the possible combination of values for all the parameters, within the particular space's mathematical model.

Accumulator The quantized space is referred as the accumulator space, which is divided cell from the parameter space (x, y).

Image section The input image is partitioned into a undefined numbers of horizontal sections.

References

[1] V. Yadav, "More robust lane finding using advanced computer vision techniques," 2017.

[2] Y. Wang, E. K. Teoh, and D. Shen, "Lane detection and tracking using B-Snake," *Image and Vision Computing*, 2004.

[3] J. Canny, "A YARIATIONAL APPROACH TO EDGE DETECTION," 1983.

[4] Y. Senthil Kumar, "Canny Edge Detection Implementation on TMS320C64x/64x+ Using VLIB," 2009.

[5] D. Antolovic, "Review of the Hough Transform Method, With an Implementation of the Fast Hough Variant for Line Detection."

[6] "Image Transforms - Hough Transform."

[7] "Hough Line Transform — OpenCV 3.0.0-dev documentation."

[8] L. Kleeman, "Understanding and Applying Kalman Filtering."

[9] "Bilgin's Blog | Kalman Filter For Dummies."

[10] C. S. Christos Stergiou, "Neural Networks."

[11] D. Kirk, "NVIDIA CUDA Software and GPU Parallel Computing Architecture."

[12] C. Asplund, "Object classification and localization using machine learning techniques; Designing and training models for use in limited hardware-applications."