Camera-based lane detection for automated driving of a model car

Bahri Enis Demirtel

Master's Thesis – 02 November 2017

Supervisor: Dr.-Ing. Eric Lenz

TECHNISCHE UNIVERSITÄT DARMSTADT

REGELUNGSTECHNIK UND MECHATRONIK *rtm*

Aufgabenstellung

Im Rahmen des Projektseminars Echtzeitsysteme werden von Studenten Themen aus dem Bereich des autonomen Fahrens bearbeitet. Bisher haben sich die Fahrzeuge dabei frei im Raum bewegt, und eine Orientierung stützte sich im Wesentlichen auf Wände, die über Ultraschallsensoren und 2D- sowie 3D-Kameras erkannt werden.

Um realistischere Fahrsituationen zu behandeln, soll in Zukunft auf eine Fahrt innerhalb von markierten Fahrspuren umgestellt werden. Als mittelfristiges Ziel ist die Teilnahme studentischer Gruppen am Carolo-Cup zu nennen. Die Spezifikation der Fahrspurmarkierungen ist daher den Regeln zum Carolo-Cup zu entnehmen.

Basisziel dieser Arbeit ist die Implementierung einer geeigneten Methode, die anhand von Kameradaten die aktuelle Fahrspur und die Nachbarfahrspur erkennt. Dabei ist der Verlauf der Fahrspur in einer geeigneten mathematischen Beschreibung anzugeben, aus der sich die Breite der Fahrspur, deren Krümmung und Krümmungsänderung über den Weg bestimmen lässt.

Damit die Daten sinnvoll weiterverarbeitet werden können, ist eine genügend kleine Abtastzeit zu erreichen, und die Totzeit zwischen der Bilderfassung und der Ausgabe der Ergebnisse darf nicht zu hoch werden. Zudem sollte bei der in einem Zeitschritt bestimmten Fahrspur angegeben werden, wie sich diese in Relation zur der im Schritt davor bestimmten Fahrspur verhält, um mit einem ortsfesten Koordinatensystem rechnen zu können.

Diese Erkennung soll ausreichend robust sein, dass diese in Räumen sicher funktioniert. D. h. es müssen die typischerweise zu erwartenden Lichtbedingungen berücksichtigt werden.

Eine einfache Fahrzeugführung ist ebenfalls Bestandteil der Arbeit und notwendig, um die Fahrspurerkennung validieren zu können. Diese ist jedoch nicht Schwerpunkt der Arbeit. Die Fahrzeugführung soll ein flüssiges, nicht zu langsames Fahren innerhalb der erkannten Fahrspur ermöglichen.

Die Verwendung bestehender Lösungen ist möglich und wird bei entsprechenden guten Ergebnissen auch positiv bewertet. Damit soll aber eine entsprechende Erweiterung wie

- das Erkennen von Kreuzungen,
- das Erreichen einer gewissen Robustheit gegenüber fehlerhaften (d. h. unterbrochenen) Markierungen (Hierbei sollte sich an den Regeln des Carolo-Cups orientiert werden.) und
- das Erkennen von Schildern

einhergehen.

Wenn begründet, können weitere bzw. andere Kameras zur Verfügung gestellt werden.

Beginn:	02 May 2017
Ende:	02 November 2017
Seminar:	20 November 2017

Prof. Dr.-Ing. Ulrich Konigorski

Dr.-Ing. Eric Lenz

Technische Universität Darmstadt
Institut für Automatisierungstechnik und Mechatronik
Fachgebiet Regelungstechnik und Mechatronik
Prof. Dr.-Ing. Ulrich Konigorski

Landgraf-Georg-Straße 4
64283 Darmstadt
Telefon 06151/16-25200
www.rtm.tu-darmstadt.de

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 02 November 2017

Bahri Enis Demirtel

Contents

1 Introduction

Automobiles have been an essential part of modern life for the better part of a century and have, for just as long, been a source of injury and death due to accidents. According to data from Federal Statistical Office in Wiesbaden, in 2016, roughly 2.6 million road traffic accidents occured in Germany and because of these accidents, 3,206 people died. When compared with 2015, although the number of deaths due to traffic accidents decreased by 7.3%, the number of road traffic accidents increased by 2.7% in 2016.[1]

The number one cause of traffic accidents is human error. Like all industries, the automotive industry continues to change and develop rapidly. In a couple of years, there will be more autonomous cars on the road and these cars will eventually replace human drivers. Because of this reason, a large portion of road traffic accidents will be eliminated. This is not the only advantage of autonomous cars. Thanks to autonomous cars, people in traffic will experience less stress, and will also have more time for other things. While driving, the people will be able to work, eat, read and even sleep. But it is also not so easy to build such reliable cars. Because of this reason, nowadays, one of the biggest research areas in the automotive industry is autonomous cars. This research area includes many different fields. Some of these fields are: Car-2-Car/Car-2-X communication, lane detection, sign recognition, object detection, path planning, and so on. In this master's thesis, some lane detection methods, their implementations, advantages, and disadvantages will be discussed.

As in all industries, technology in the automotive industry is continuing to develop day by day. For example, the number of sensors, and their corresponding features, is increasing exponentially. One such sensor is the color camera. To begin with, in the automotive industry, cameras were used only to assist drivers in parking and reversing.

Nowadays, however, one of the main functions of color cameras is lane detection, in both autonomous cars and in cars equipped with a lane departure warning system. In this master's thesis, the lanes will be detected and then formulated mathematically.

The results of this master's thesis will be utilized and expanded upon by the students who will participate in the Echtzeitsysteme Projektseminar at the Technical University of Darmstadt. One of the aims of this seminar is to attend the Carolo-Cup organized annually by the Technical University of Braunschweig. Because of that, the width, the curvature, and the changes of the curvature of the track used in this master's thesis are the same as those belonging to the track used in the Carolo-Cup. In a real-life situation, there are of course oftentimes more factors that can hinder lane detection, including shadows cast by trees, buildings, and other structures; sunlight directly entering the lens of the camera and similarly less-than-ideal lighting conditions; dirt and debris on the road surface; and so on.

Therefore, the lanes of the track must be detected in a sufficiently short amount of time and there should be no dead time between lane detection and mathematical formulation. Lane detection must also be sufficiently robust, so that it should not be disrupted by less-than-ideal lighting conditions.

1.1 Carolo-Cup

The Carolo-Cup is a student competition that provides teams of students with an opportunity to design and implement automated remote control cars within the confines of a realistic application scenario. Essentially, the cup serves as opportunity for students to showcase their abilities to find and implement innovative algorithmic solutions for both vehicle control and environmental perception.

The teams of students coming from universities all over the world present their findings to a panel of jurors composed of academics as well as experts from the private sector.

Each team of students is tasked with designing and implementing a 1:10 concept of an automated vehicle that is both cost and energy efficient.

In the competition, several driving tasks have to be completed as quickly and as accurately as possible.

The team must also present their concept at the competition.

Prior to 2017, challenges of the competition itself included avoiding obstacles and stopping at intersections. However, in this year, with the aim of providing an even more realistic urban setting, the challenge of recognizing and obeying traffic signs was added.

1.2 Problem Statement and Objective

Autonomous driving is a topic currently being actively researched. Research on autonomous driving can be conducted in two fundamental areas: lane detection and lane guidance. With regard to lane detection, there are different scientific techniques that can be utilized, according to the literature, all with their own advantages and disadvantages under different conditions. For example, some techniques are suitable for straight lines, but not for curves. Others are suitable for curves as well but do not function well under certain light conditions. Others still are quite robust and suitable for curves, yet are computationally intensive (resulting in a video feed with significant gaps).

In this master's thesis, the aim is to research and implement the most appropriate and effective method for use in the Carolo-Cup.

1.3 Structure of Thesis

In Chapter 2, the fundamentals of lane detection are explained. All methods utilized in this thesis, along with their respective justifications, are also explained in this chapter. Some methods are also compared with regard to their advantages and disadvantages.

In Chapter 3, the state of the art will be discussed. The other possible solutions for lane detection will also be explored here and their advantages and disadvantages will be compared.

In Chapter 4, the steps of implementation are explained. The components can be divided broadly into the properties of the track, the hardware of the model car, and the software libraries and programs to be utilized. In the software section, all cases will be explained in detail.In this chapter, the program flow will also be explained in detail.

In Chapter 5, the results of the methods utilized will be compared. The computing time of all phases in this thesis will be presented and discussed. Also, all parameters utilized and their effects on this thesis will be also presented and discussed. In this chapter, the researcher will attempt to find an answer to the question, 'How can computing time be reduced?'.

In Chapter 6, all results of this master thesis will be presented and possible improvements and/or enhancements will be discussed.

2 Fundamentals

In this chapter, the fundamentals of lane detection will be explained. All methods, which are used in this thesis, will be theoretically focused and the use of their functions in software will be also explained. Also, some advantages and disadvantages of methods will be discussed.

2.1 Properties of Track at Carolo-Cup

The Carolo-Cup is an annual competition at the Technical University of Braunschweig which are attended by students. Every year the track and some properties of the competition are changing. For example, in the competitions until 2017 there was no traffic sign, but starting in 2017 there are also some traffic signs, speed limit zones, blocked areas and crosswalks for pedestrian. Because of this, in the competitions until 2017, there was only one way to understand who had the right of way. If there is a stop line on the road in front of an intersection, it means the car has to wait until the intersection is free. In the competitions starting from 2017, there are different types of intersections: They are 'Intersections with stop lines', 'Intersections with give-way lines', 'Intersections with priority to right', 'Enforced crossing direction - give-way condition', 'Enforced crossing direction - right of way condition'. Except 'Intersections with priority to right', they all have traffic signs stating who has priority. If there is a no traffic sign, it means the right side always has priority.[2]

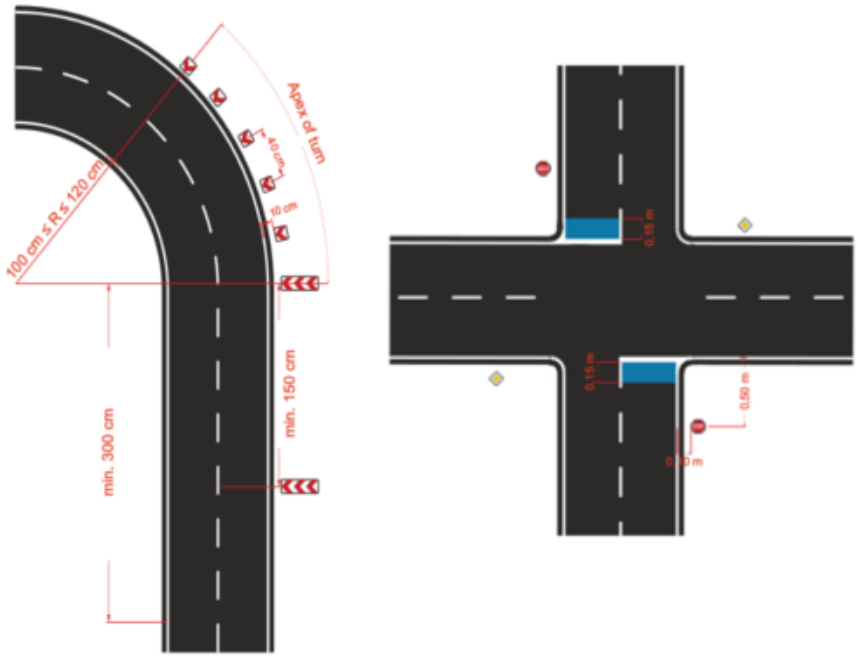


Figure 2.1.: Left: Markings for sharp turns at Carolo-Cup. Right: Intersections with stop lines at Carolo-Cup[2]

2.2 Inverse Perspective Mapping

Inverse Perspective Mapping(IPM) is an algorithm which is able to obtain accurate bird's-eye view images from forward looking cameras. With the IPM algorithm, each image pixel is remapped, and a new array of pixels is created where the lines in perspective are transformed into straight lines and objects are distorted. IPM is one of the most used methods in lane detection. In lane detection, IPM ensures that the lanes are shown vertical and parallel to each other. On the other hand, because of the re-mapping of pixels, IPM is a computationally expensive method. Because of this reason, in some cases in this master's thesis, rather than remapping all pixels of the images, only the pixels relevant to the lane and accordingly, the fitted curve, were remapped. Thanks to this, in some cases, a lot of computing time was saved.

In order to use the IPM method, the intrinsic and extrinsic parameters of camera are necessary to process images for coordinate transformation and calibration.

- **Intrinsic Parameters :** Intrinsic parameters are camera-specific. It includes information of the focal length (f_x, f_y) and optical centers (c_x, c_y). It is also called a camera matrix. Although the intrinsic parameters are camera-specific, once the camera is calibrated, the modified intrinsic parameters can be stored for future purposes. It is expressed as a 3x3 matrix:

$$\text{camera matrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

In order to find the parameters of the camera matrix, the camera must be calibrated. In order to do this, there is a node in **R**obotic **O**peratic **S**ystem(ROS) which is programmed in the Python programming language. In the tutorial for camera calibration**??** an 8x6 checkerboard with 108mm squares is used. The following command must be used for the calibrating the camera.

rosrun camera_calibration cameracalibrator.py —size 8x6 —square 0.108 image:=/usb_cam/image_raw camera:=/usb_cam

As seen in the above command, the number and the size of the checkerboard's squares must be written in the command and then the camera starts automatically after the command is run. This opens up the calibration window which then highlights the checkerboard. The checkerboard must be moved around in front of the camera. During this process, the camera takes some measurements from the checkerboard. When enough data is collected, the *CALIBRATE* button is highlighted. After this button is clicked, the camera matrix is shown and simultaneously saved as the instrinsic parameters in the camera.

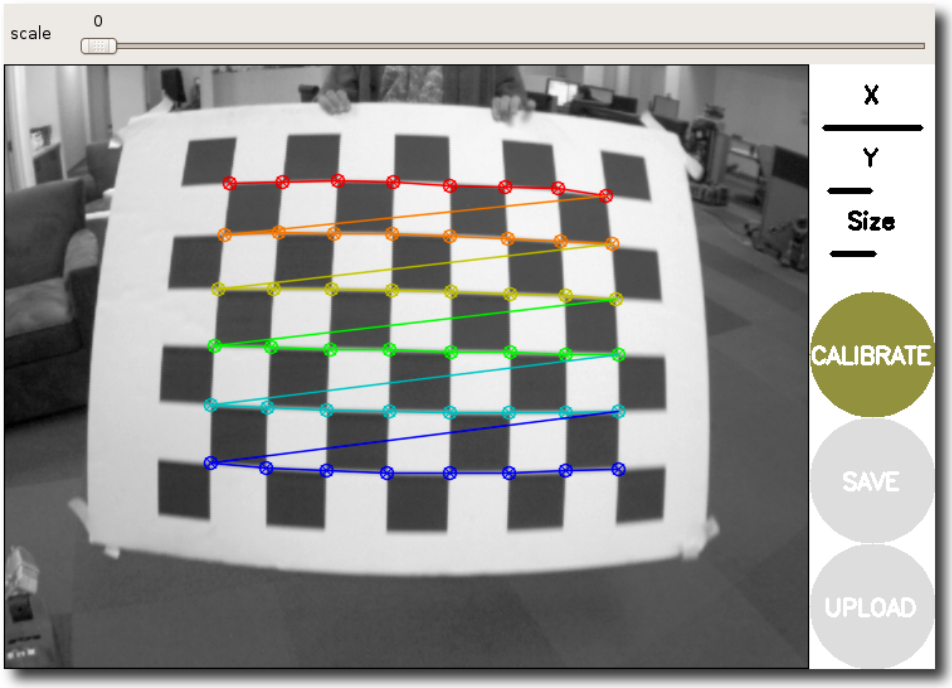


Figure 2.2.: Calibration of Camera[3]

For this master's thesis, the camera was calibrated and the parameters of the camera matrix can be found in the Table A.1

- **Extrinsic Parameters :** Extrinsic parameters are dependent on the camera position. The parameters are H and θ. H is the distance between the camera and ground. θ is the camera tilt angle. These values must be measured because they have to be used for Inverse Perspective Mapping. The extrinsic parameters of the camera can be found in the Table A.1.

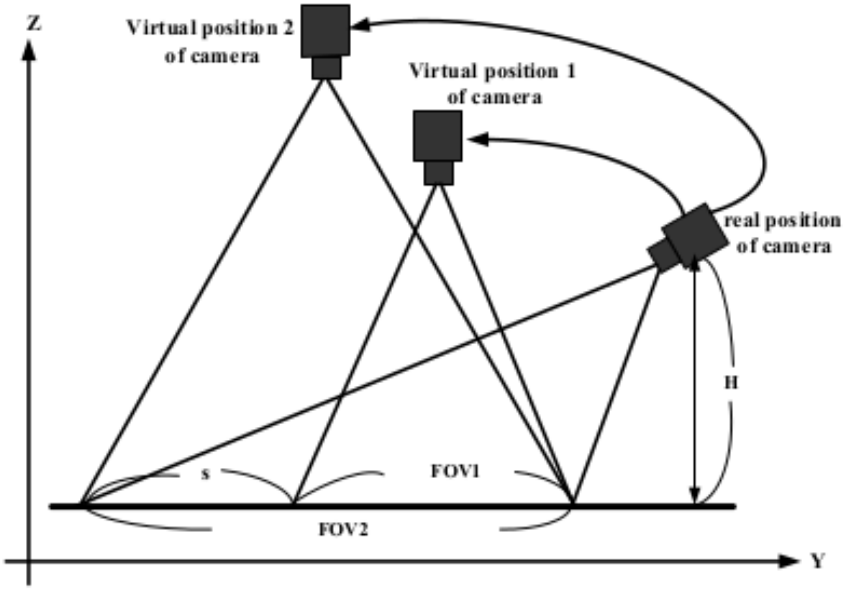


Figure 2.3.: Related Positions of the Camera [4]

As seen at Figure 2.3, the camera on the car has field of view 2 (FOV2) at the real position of the camera but in this case, the view is not a bird's-eye view, so if the same field of view is to be observed from a bird's-eye view, IPM will virtually change the position to Virtual Position 2 of the camera. In this case, although the camera is at its real position, it will appear as though it is at Virtual Position 2. Sometimes, however, obtaining the view from Position 2 is not desirable, because if the field of view is very large, then Position 2 is also very high. In this project, if the lanes observed from a position high above the ground, the lanes will appear small, and thus lane detection becomes more difficult and less accurate. To solve this problem, the upper portion of the image can be cropped, so the camera will have a smaller field of view. In this case, although the camera is at its real position, it will appear as though it is at Virtual Position 1.

In order to apply IPM algorithm, the image coordinates must also be changed. Below, the steps of IPM calculations from the paper of [4] will be detailed.

In order to calculate the destination image coordinates (x*,y*), the input image coordinates (x,y), the height of the camera from the ground (H), the focal length of the camera (f), and the tilt angle of the camera (θ) are all necessary and are used in the following formula[4]:

$$x^* = H\frac{xsin\theta+fcos\theta}{-ycos\theta+fsin\theta}\ ;\ y^* = H\frac{ysin\theta+fcos\theta}{-ycos\theta+fsin\theta}$$

In this equation, the transformed component values of x^* and y^* may be less than or equal to zero. Because of this reason, a constant d is defined as[4] :

$$\mathrm{d} = \left|\frac{H(sin\theta+fcos\theta)}{fsin\theta-cos\theta}\right| + 1$$

This means that the coordinate point in the original source image has been mapped into the point of the destination image coordinate system. Below there is the proposed equation[4] :

$$x^* = H\frac{xsin\theta+fcos\theta}{-ycos\theta+fsin\theta} + \mathrm{d}\ ,\ y^* = H\frac{ysin\theta+fcos\theta}{-ycos\theta+fsin\theta} + \mathrm{d}\ ,\ \text{where d} = \left|\frac{H(sin\theta+fcos\theta)}{fsin\theta-cos\theta}\right| + 1$$

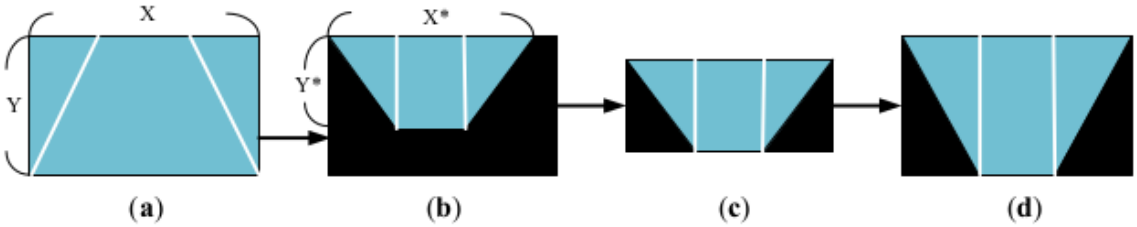


Figure 2.4.: Procedures of IPM[4]

2.3 Edge Detection

Edge detectors are essential parts of most computer vision systems. Edge detectors dramatically decrease the amount of data to be processed and extract the useful parts of images. They work by detecting discontinuities in brightness. It is usually done by applying a 2-D filter, which is designed in such a way that it is only sensitive to the large gradients present in the image, returning zero values for all monotonous regions. There is a wide variety of specifically composed operators, each of them showing the best results when used with some particular types of edges. For instance, the different orientations of the edges, as well as their geometry, structure, and thickness should correspond with operator characteristics in order for the detection to be reliable. Another problem is the image noise, which in many cases could be confused with the local gradient and thus detected as an edge. Because of that, there is no universally efficient operator, and for each particular case it should be specifically configured, and some pre-editing must be used in order to achieve the best detection quality. In this project, the edge detector is used to detect the lanes and to exclude other unnecessary information from images. There are several different methods suitable for lane edge detection, and all of them can be grouped into two categories. They are :

1. **Gradient method :** This method searches for the maximum and minimum in the first derivative of the image and with that, the edges can be found. From the color intensity

perspective, any edge has either a rise or fall, which gives a local spike after applying the derivative operator, thus highlighting the edge location in the image. For this method, the first order derivative filter must be used. For example : Sobel-Operator, which is described in detail in Section 2.3.1.

2. **Laplacian method :** This method searches for the zero crossing in the second derivative of the image. This is an alternative for locating local spikes, as the second derivative is equal to zero when the first is at minimum or maximum. For example : Canny edge detector, which is described in detail in Section 2.3.2.

According to [5], there are three steps of the edge detection algorithm. They are :

- **Filtering :** For edge detection, it is required to use a suitable smoothing filter. The filters sharpen the edges and ignore unnecessary information. It is often utilized to improve the functioning of an edge detector against noise. The more filtering is applied, however, the greater the loss of edge strength. The most common implementation of a smoothing filter used for image preprocessing is Gaussian blur, which is described in detail in Section 2.4
- **Enhancement :** To be able to better detect edges, changes in the intensity in the area surrounding a point must be determined. Pixels in which a significant change in intensity occurs are emphasized by enhancement, which is usually applied by calculating the gradient magnitude.
- **Detection :** Though many points in an image have a nonzero value for the gradient, not all of these points are actually edges. Because only points with strong edge content are desired, a method must be applied to determine which points are actual edge points. Thresholding is often utilized to do so.

Well known edge detection filters are :

- Sobel-Operator
- Canny Edge Detector
- Laplacian-Filter
- Prewitt-Operator

2.3.1 Sobel Operator

The Sobel Operator, sometimes called the Sobel filter or Sobel-Feldmann operator is one of the most used edge detectors in image processing and computer vision. The Sobel Operator uses vertical and horizontal masks. These masks used are odd-numbered square matrices and they are generally 3x3 matrices. Approximations of the derivatives for the horizontal changes and for the vertical changes are calculated by the operator by using two 3x3 kernels and convolving them with the original image. If A is defined as the source, if G_x is an image which contains the

horizontal derivative approximations at each point, and if G_y is an imagine which contains the vertical derivative approximations at each point, then the calculations are[6]:

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * A \quad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A$$

where * here denotes the 2-dimensional signal processing convolution operation.

Since the Sobel kernels can be decomposed as the products of an averaging and a differentiation kernel, they compute the gradient with smoothing. For example, G_x can be written as:

$$\begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} +1 & 0 & -1 \end{bmatrix}$$

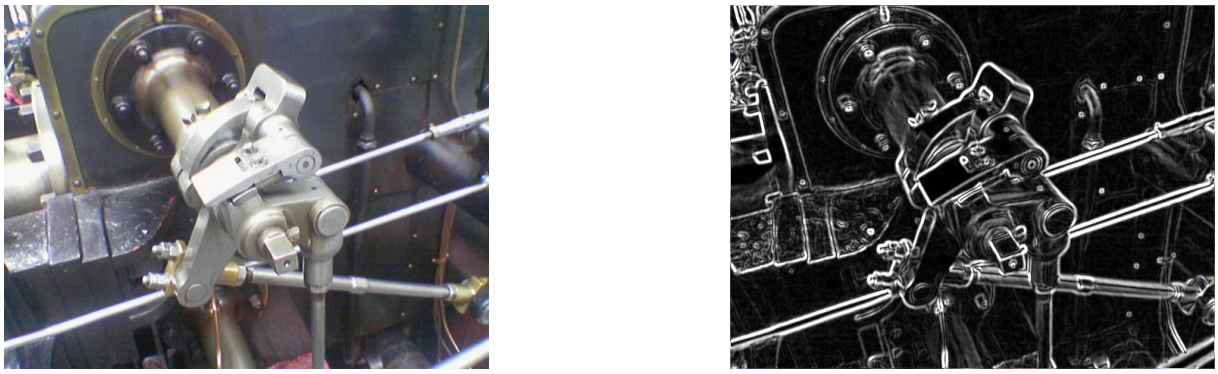
The x-coordinate is defined here as increasing in the 'right'-direction, and the y-coordinate is defined as increasing in the 'down'-direction. At each point in the image, the resulting gradient approximations can be combined to give the gradient magnitude, using[6]:

$$\mathrm{G} = \sqrt{G_x^2 + G_y^2}$$

Using this information, we can also calculate the gradient's direction[6]:

$$\theta = atan\left(\frac{G_y}{G_x}\right)$$

where, for example, θ is 0 for a vertical edge which is lighter on the right side.



(a) Original Image **(b)** Sobel Operator applied Image

Figure 2.5.: Sobel Operator[6]

The Sobel operator's main advantages are due to its simplicity. One is that the Sobel operator provides an approximation of the gradient magnitude. Another advantage of the Sobel operator

is that it can detect edges and their orientations. In this cross operator, the detection of the edges and their orientations is considered to be simple because of the approximation of the gradient magnitude. The Sobel operator does have some disadvantages, however. For example, it is sensitive to noise. The magnitude of the edges degrades as the level of noise present in image increases. As the magnitude of the edges degrades, the accuracy of the Sobel operator also diminishes. Overall, the Sobel operator cannot accurately detect edges when edges are thin or smooth.

In this master's thesis, the Sobel Operator function in OpenCV was used. Before the Sobel operator was used, in order to reduce the noise, the 'GaussianBlur' function was used. As can perhaps be inferred from the name, the 'GaussianBlur' function blurs the image using a Gaussian filter. In order to apply the 'GaussianBlur' function, the following command must be run.

```
void GaussianBlur(cv::Mat src, cv::Mat dst, cv::Size ksize, double sigmaX,
            double sigmaY=0, int borderType=BORDER_DEFAULT )
```

The parameters of the function will be described in detail.[7]

- **src :** Input image, which can have any number of channels but the depth of which must be one of the following: CV_8U, CV_16U, CV_16S, CV_32F and CV_64F.
- **dst :** Output image, which must be the same size and type as the input image.
- **ksize :** Gaussian kernel size. This size shows the width and height of the Gaussian kernel. These sizes must not be the same value and the values must be odd and positive.
- **sigmaX :** Gaussian kernel standard deviation in the X direction.
- **sigmaY :** Gaussian kernel standard deviation in the Y direction.
- **borderType :** Pixel extrapolation method.

Guassian Blur will be explained in the Section 2.4.

After the Gaussian Blur is applied, the picture must be converted from color to grayscale. For that, there is a small function in OpenCV. With the following command, a color picture can be converted easily to grayscale.

```
void cvtColor(cv::Mat src, cv::Mat dst, int code, int dstCn=0 )
```

The parameters of the function will be described in detail.

- **src :** Input image
- **dst :** Output image which is the same size and depth as the input image.
- **code :** Color space conversion code(here used CV_BGR2GRAY).
- **dstCn :** Number of channels in the destination image.

Before the Sobel operator is used, the Gaussian Filter must be applied and the image must be converted to gray scale. After this, the image is ready for Sobel operator to be applied. In order to calculate the 'derivatives' in the x and y directions, the following command must be run twice because gradient X and gradient Y must be calculated separately.

```
void Sobel(cv::Mat src, cv::Mat dst, int ddepth, int dx, int dy, int ksize=3,
       double scale=1, double delta=0, int borderType=BORDER_DEFAULT )
```

The parameters of the function will be described in detail.

- **src :** Input image.
- **dst :** Output image which is in the same size and depth as the input image.
- **ddepth :** The depth of the output image.
- **xorder :** Order of the derivative x.
- **yorder :** Order of the derivative y.
- **ksize :** Size of the extended Sobel kernel; it must be 1, 3, 5, or 7.
- **scale, delta and borderType :** Optional values. In this project, the default values were used.

The last step of the application of the Sobel operator is approximating the gradient by adding both directional gradients. In the previous step, the gradients of the x and y coordinates were calculated separately. With following command, the weighted sum of these two gradients must be calculated.[8]

```
void addWeighted(cv::Mat src1, double alpha, cv::Mat src2, double beta,
                 double gamma, cv::Mat dst, int dtype=-1)
```

The parameters of the function will be described in detail.

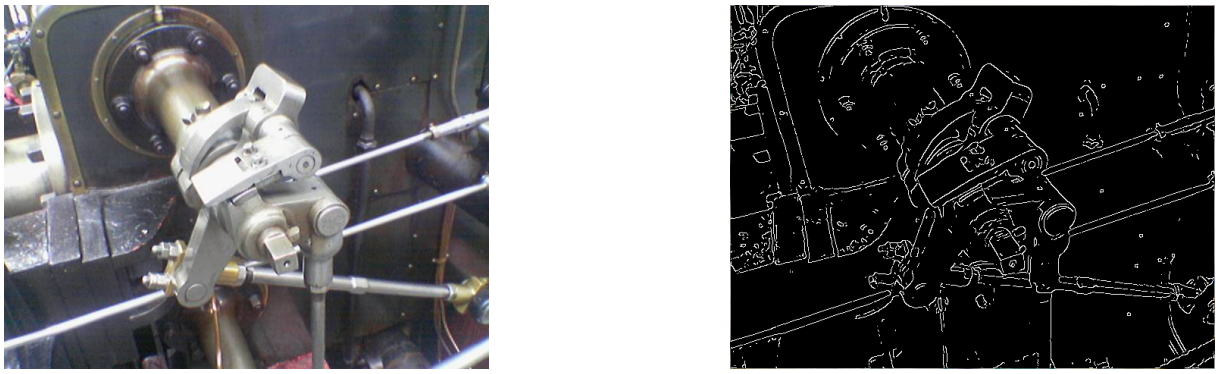
- **src1 :** First input array.
- **alpha :** Weight of the first array elements.
- **src2 :** Second input array. This array must have the same size and channel number of the first input array.
- **beta :** Weight of the second array elements.
- **dst :** Output array, which has the same size and channel number of the the input arrays.
- **gamma :** Scalar added to each sum.
- **dtype :** Optional depth of the output array.

2.3.2 Canny Edge Detector

Another popular edge detector is the Canny Edge Detector, which was developed in 1986 and was named after its developer, John F. Canny. The Canny Edge Detector is a multi-stage algorithm and it can be analyzed in five different stages.[9]

1. **Noise Reduction :** To get stable lane detection results, we have to reduce/remove all noise from frames. Lane detection without filtering out the noise can cause false detection. Gaussian filter is used for removing noise in the frames. Gaussian filter blurs images and removes detail and noise. The size of Gaussian filter kernel must be (2k+1)x(2k+1). It is important to choose the size of Gaussien filter because if the size of kernel is larger, detector's sensitivity to noise is lower but on the other hand, with the increase in size of the Gaussian filter kernel, the localization error in the edge detection will also increase slightly. [10]
2. **Finding Intensity Gradient of the Image :** Essentially, the Canny algorithm locates edges in image where the grayscale intensity changes most starkly. In order to find these areas, the gradients of the image must be determined. In order to determine the gradients at each pixel in the smoothed image, the Sobel operator is applied. The Sobel operator has already been thoroughly discussed in section 2.3.1.
3. **Non-maximum Suppression :** Non-maximum suppression is an edge thinning technique which is used as an intermediate step in many computer vision algorithms. The image is scanned along the image gradient direction, and pixels that are not part of the local maxima are set to zero. This way, all image information that is not part of the local maxima is effectively suppressed.
4. **Double Thresholding :** The edge pixels remaining after applying non-maximum suppression provide a more accurate depiction of real edges in an image. Despite this, there are still some remaining edge pixels resulting from noise and color variation. Therefore, it is necessary to filter out edge pixels with a weak gradient value while preserving edge pixels with a high gradient value. In order to do this, high and low threshold values must be selected. Edge pixels are marked as strong edge pixels when gradient values are higher than the high threshold value. They are marked as weak edge pixels when gradient values are lower than the high threshold value and higher than the low threshold value. They are suppressed when their values are lower than the low threshold value. The two threshold values are determined empirically and are dependent on the content of a given image.
5. **Hysteresis Thresholding :** Hysteresis Thresholding is the last part of Canny Edge Detector. Until this step, strong edge pixels are extracted from the true edges but there are also some weak edge pixels, some of them are extracted from true edges and some of them are extracted from some noise. So the weak edge pixels which are extracted from true edge, should be strong edge pixels and the weak edge pixels which are extracted from noises

must be removed. If there is a weak edge pixel, eight neighbor pixels of that weak edge pixel are checked, and if at least one of these neighbor pixels is a strong edge pixel, these weak edge pixels remain as edges in the final image.



(a) Original Image **(b)** Canny Edge Detector applied Image

Figure 2.6.: Canny Edge Detector[10]

The main advantage of Canny Edge Detector is enhancement of the signal with respect to the noise ratio. This is done via the non-maxima suppression method, as it results in one pixel wide ridges as the output. The other advantage is better detection of edges, especially in a noisy state, by applying the thresholding method. The effectiveness of the Canny method is affected by adjustable parameters. Small filters are desirable for the detection of small, sharp lines, since they cause fewer instances of blurring. Large filters are desirable for detecting larger, smoother edges. However, they also cause higher instances of blurring. The main disadvantage of the Canny edge detector is that it is time consuming, due to its complex computation.

The Canny Edge Detector is also a function in OpenCV. As is the case with the Sobel Operator, before the Canny Edge Detector is used, a filter should be applied (for example, the Gaussian Filter) and the image must be converted to gray scale. With the following command, the Canny Edge Detector can be run in OpenCV.[11]

```
void Canny(cv::Mat image, cv::Mat edges, double threshold1, double
    threshold2, int apertureSize=3, bool L2gradient=false)
```

The parameters of the function will be described in detail.[11]

- **image :** Input image, which has to have an 8-bit single channel.
- **edges :** Output image, which is the same size and type.
- **threshold1 :** First threshold of the hysteresis procedure.
- **threshold2 :** Second threshold of the hysteresis procedure.
- **apertureSize :** Size of the extended Sobel kernel.
- **L2gradient :** A flag for image gradient magnitude.

In this master's thesis the Sobel Operator was utilized. The first reason for using this algorithm rather than the Canny Edge Detection algorithm is its performance. The latter uses the second derivative, which is more computationally expensive and therefore takes more time. For the purpose of lane detection, it is crucial to spend as little computation time per frame as possible, because longer durations may result in a substantial delay between the image capture and the car driving module's reaction to a new lane position.

2.4 Gaussian Blur

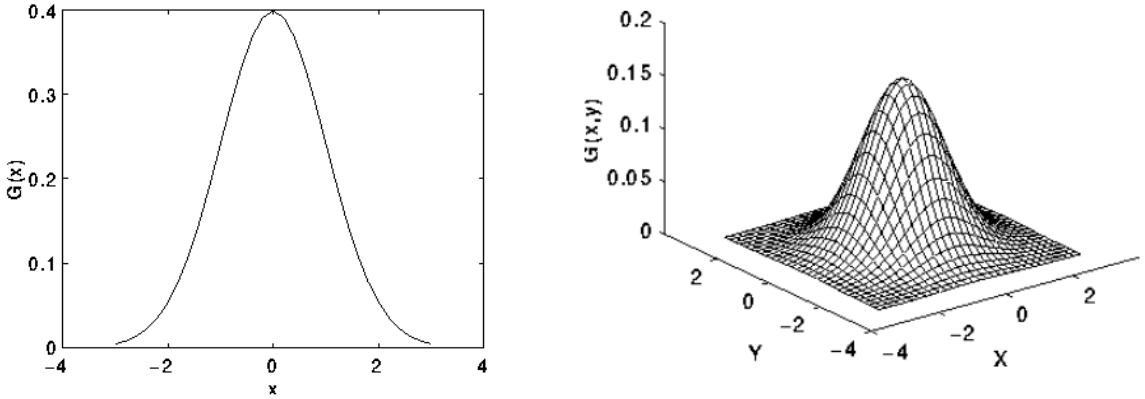
The Gaussian Blur is a non-uniform low pass filter which is used to blur images. For smoothing/blurring the images, there are also some other filters, like Normalized Box Filter, Median Filter, Bilateral Filter, as well as others, but the Gaussian Blur is the most useful one, although it is not the fastest. This filter is widely used in image processing as one part of the preprocessing stage because it reduces image noise and detail.

The Gaussian Blur uses a Gaussian function in order to calculate the transformation to apply to each pixel in the image. The equation of a Gaussian function in one dimension has the following form[7]:

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

where σ is the standard deviation of the distribution. The equation of a Gaussian function in two dimensions is the product of two such Gaussians, one in each dimension[7]:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



(a) 1-D Gaussian distribution **(b)** 2-D Gaussian distribution

Figure 2.7.: Gaussian distributions with mean (0,0) and σ=1[7]

The Gaussian filter works by using the 2D distribution as a point-spread function. This is achieved by convolving the 2D Gaussian distribution function with the image. A discrete approximation to the Gaussian function needs to be produced. This theoretically requires an infinitely large convolution kernel, as the Gaussian distribution is non-zero everywhere. Fortunatly, the distribution has approached very close to zero at about three standard deviations from the mean. 99% of the distribution falls within three standard deviations. This means that normally, the kernel size can be limited to contain only values within three standard devations of the mean.

The distribution is assumed to have a mean of zero. The continuous Gaussian functions need to be discretized to store it as discrete pixels. An integer valued 5 by 5 convolution kernel approximating a Gaussian with a σ of 1 is shown[7] :

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

As seen in the convolution kernel shown above, the kernel coefficients diminish with increasing distance from the kernel's center and the central pixels have a higher weighting than those on the periphery. Also, the Gaussian kernel coefficients depend on the value of σ. If the value of σ is larger, it produces a wider peak, meaning that the images are blurred more.

2.5 Hough-Transformation

In order to isolate features of a particular shape in an image, a method called the Hough Transformation can be utilized. The Hough Transformation which is universally used today was further developed by Richard O. Duda and Peter E. Hart in 1972, although a more rudimentary version had been patented by Paul Hough in 1962.[12] In computer vision, it is often necessary to detect simple edges like straight lines, curves, and ellipses. As a result, this technique is used often in computer vision and image processing. The Hough Transformation is used mostly after an edge detection algorithm. There are actually variations of the Hough Transformation. In this master's thesis, two variations of the Hough Transformation were used. They are:

- **Standard Hough Line Transformation :** This type of Hough Transformation is used mostly for detecting straight lines. How the Standard Hough Transformation works will be explained in detail and as a result, why it is more suitable for detecting straight lines will become more clear.
- **Probabilistic Hough Transformation :** This type of Hough Transformation is suitable for both straight lines and curves, so in this master's thesis, the Probabilistic Hough Transfor-

mation was used for detecting lanes. How the Probabilistic Hough Transformation works will also be explained.

2.5.1 Standard Hough Line Transformation

The Standard Hough Line Transformation, which is also called the Linear Hough Transformation, is used mostly for detecting straight lines. There are many different formulas to represent a line segment analytically. A convenient formula for representing a line in an Image(Cartesian) space is:

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

In this equation, ρ is the length of a normal from the origin to the line and θ is the angle between this normal and the x-axis.

Standard Hough Transformation algorithm consists of the following steps[13]:

Algorithm 1 HT for detecting lines on the $\theta - \rho$ parameterization

1: **procedure** DETECT LINES
   Input: Input image I with dimensions I_w,I_h, Hough space dimensions H_ρ,H_θ.
   Output: Detected lines $L = \{(\theta_1,\rho_1),...\}$
2: $H(\overline{\rho},\overline{\theta}) \longleftarrow 0$, $\forall\overline{\rho} \in \{1,...,H_\rho\},\overline{\theta} \in \{1,...,H_\theta\}$
3: **for all** $x \in \{1,...,I_w\},y \in \{1,...,I_h\}$ **do**
4: **if** $I(x,y)$ **is edge then**
5: **increment** $H(\overline{\rho}(\overline{\theta},x,y),\overline{\theta}),\forall\overline{\theta} \in \{1,...H_\theta\}$
6: **end if**
7: **end for**
8: $L = \{(\theta(\overline{\theta}),\rho(\overline{\rho}))|\overline{\rho} \in \{1,...,H_\rho\} \wedge \overline{\theta} \in \{1,...,H_\theta\} \wedge$ at $(\overline{\rho},\overline{\theta})$ is a high max. in $H\}$
9: **end procedure**

An edge detection algorithm is applied to the image with dimensions I_w, I_h, which provides the binary data used in Step 4 to determine whether or not a pixel belongs to an edge. Steps 3 - 7 convert detected edges to the Hough space H and accumulate them. The following function is used to calculate $\overline{\rho}$ for all detected edges that pass through the point (x,y) at angle $\overline{\theta}$[13]:

$$\overline{\rho}(\overline{\theta},x,y) = \left[\frac{H_\rho\left((y - \frac{I_h}{2})sin(\frac{\pi}{H_\theta}\overline{\theta}) + (x - \frac{I_w}{2})cos(\frac{\pi}{H_\theta}\overline{\theta})\right)}{\sqrt{I_w^2 + I_h^2}}\right]$$

In Step 8, the high maximum (the local maximum exceeding a given threshold) of all accumulated lines in Hough space H is determined. It gives $\overline{\rho}$ and $\overline{\theta}$ as detected line parameters in the Hough space. After that Hough coordinates $\overline{\rho}$ and $\overline{\theta}$ are transformed back into ρ and θ using the following formulas[13]:

$$\rho(\overline{\rho}) = \frac{\sqrt{I_w^2 + I_h^2}}{H_\rho}\left(\overline{\rho} - \frac{H_\rho}{2}\right), \qquad \theta(\overline{\theta}) = \frac{\pi}{H_\theta}\overline{\theta}$$

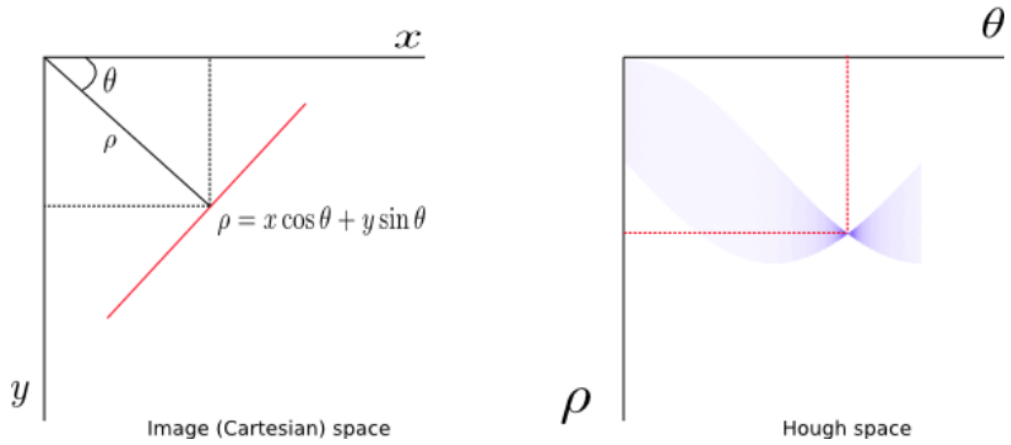


Figure 2.8.: Standard Hough Line Transformation[14]

In the left side of Figure 2.8, a line is shown in an Image (Cartesian) space and on the right side of same Figure, the same line is shown in its transformed state in the Hough space. A point in the Image (Cartesian) space is transformed into a sinusoidal curve in the Hough space. A line can be defined as a set of points, so a line is transformed into a set of sinusoids intersecting at a point in the Hough space. In this case, detecting points in the Hough space also means detecting the lines in the Cartesian space. After detecting points in the Hough Space, the points must be inversely transformed in order to get the corresponding lines in the Cartesian space.

Standard Hough Line Transformation is widely used technique in curve detection. Despite this, it does have two important downsides. Firstly, for each nonzero pixel in an image, the parameters for both the existing curve as well as the redundant ones are obtained during the 'voting procedure'. Secondly, the more accuracy that is needed, the higher the parameter resolution must be defined. As a result, there is often a large storage requirement and low speed for real applications. Probabilistic Hough Transformation was derived in order to counter these problems.

In this master's thesis, in order to detect the Hough lines, the function of *cv2.HoughLines()* in OpenCV was used. The function is:

```
void HoughLines(cv::Mat image, cv::Mat lines, double ρ, double θ, int
                threshold, double srn=0, double stn=0)
```

The parameters of the function will be described in detail.[15]

- **image :** Output of the edge detector. It should be a grayscale image (although in fact it is a binary one).

- **lines :** A vector that will store the parameters (ρ, θ) of the detected lines. At the end of the Standard Hough Line Transformation, only the ρ and θ values will be returned, so only straight lines can be drawn in OpenCV.
- ρ **:** The resolution of the parameter ρ in pixels. In this master's thesis, **1 pixel** was used.
- θ **:** The resolution of the parameter θ in radians. In this master's thesis, **180 degree**(CV_PI) was used.
- **threshold :** The minimum number of intersections to 'detect' a line.
- **srn and stn :** Both of these parameters are for the multi-scale Hough transformation. srn is a divisor for the distance resolution ρ and stn is a divisor for the distance resolution θ. Default values of both parameters are zero.

2.5.2 Probabilistic Hough-Transformation

Probabilistic Hough Transformation, also called Randomized Hough Transformation, is a variant of the Standard Hough Transformation. Hough Transformation is computationally expensive. Probabilistic Hough Transformation is an optimization of Standard Hough Transformation which does not analyze all points in the image. Instead, it looks at only a random subset of points sufficient for line detection. In order to apply Probabilistic Hough Transformation, the threshold must be decreased. Probabilistic Hough Transformation is commonly used to detect curves (straight line, circle, elipse,etc.)

Essentially, the point of the Hough Transformation is to implement a 'voting procedure' for all potential curves in the image. Curves that actually found in the image will then have relatively high 'voting scores' when the algorithm terminates. Probabilistic Hough Transformation differs from Standard Hough Line Transformation by trying to avoid a computationally expensive 'voting process' for every nonzero pixel in the image. It does this by taking advantage of the geometric properties of analytical curves, which allows it to improve efficiency as well as reduce the storage requirement of the original algorithm.

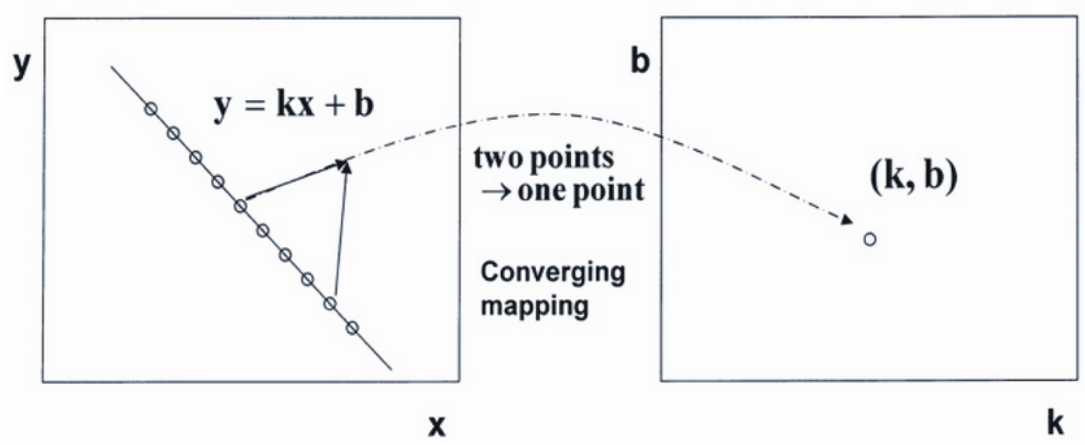


Figure 2.9.: Probabilistic Hough Transformation

Line detection using Probabilistic Hough Transformation is similar to standard Hough Transformation, but instead of using all edge points of the image, some subset is randomly selected, which is then used to determine a single unique parameter space point for every pair of points from the subset (see Figure 2.9).

As soon as the amount of points that are mapped to the same parameter in Hough space exceeds a certain threshold, the line is confirmed to be present in the original image, i.e. the line is detected. It is then removed from the input array so that the other lines are detected faster. The algorithm terminates either after all the lines are detected and there is no more input data, or the maximum limit of iterations is reached. Because of this approach, the required computational power and memory consumption are much less when compared with Standard Hough Transformation.

Probabilistic Hough Transformation algorithm consists of the following steps:

1. Initialise the set of points K representing the detected edges of the input image
2. Randomly select two points from the set K
3. Apply the mapping into Hough space and determine the parameter point of the straight line represented by two selected points
4. Acumulate parameter points in the Hough space. If the amount of input points mapping to the same Hough space parameter exceed a given threshold, consider this parameter as a parameter for detected line and remove it from the input data, otherwise go to step 2, if the maximum number of loops is not reached.
5. Repeat until all input data is processed.

The following function from OpenCV library implements aforementioned algorithm:

```
void HoughLinesP(cv::Mat image, cv::Mat lines, double ρ, double θ, int
        threshold, double minLinLength=0, double maxLineGap=0)
```

The parameters of the function will be described in detail.[15]

- **image :** Output of the edge detector. It should be a grayscale image (although in fact it is a binary one).
- **lines :** A vector that will store the parameters (x_{start}, y_{start}, x_{end}, y_{end}) of the detected lines.

  At the end of the Probabilistic Hough Transformation, the end points of detected lines will be returned. With the Probabilistic Hough Transformation in OpenCV, straight lines can be detected. In addition, the small straight lines in other shapes can be combined, and with this technique, all shapes can be detected.

In this master's thesis, all lanes were detected with Probabilistic Hough Transformation. However, instead of Hough Lines, the points which were returned as the result of Probabilistic Hough Transformation were used.

- ρ **:** The resolution of the parameter ρ in pixels. In this master's thesis, **1 pixel** was used.
- θ **:** The resolution of the parameter θ in radians. In this master's thesis, **1 degree**(CV_PI/180) was used.
- **threshold :** The minimum number of intersections to 'detect' a line.
- **minLinLength :** The minimum number of points that can form a line. Lines with less than this number of points are disregarded.
- **maxLineGap :** The maximum gap between two points to be considered in the same line.

2.6 K-Nearest Neighbors Algorithm

K-Nearest Neighbor(KNN) is an non-parametric lazy learning algorithm. The non-parametric technique means that it doesn't make any assumptions on the underlaying data distribution. In the definition of KNN, the term 'lazy learning algorithm' is used. It means it doesn't use the data training points to do any generalization. In other words, there is no explicit training phase or it is very minimal. It also means that the training phase is pretty fast. Most of the lazy algorithms – especially KNN – make decisions based on the entire training data set. On the other hand, KNN is one of the top 10 data mining algorithms[16].

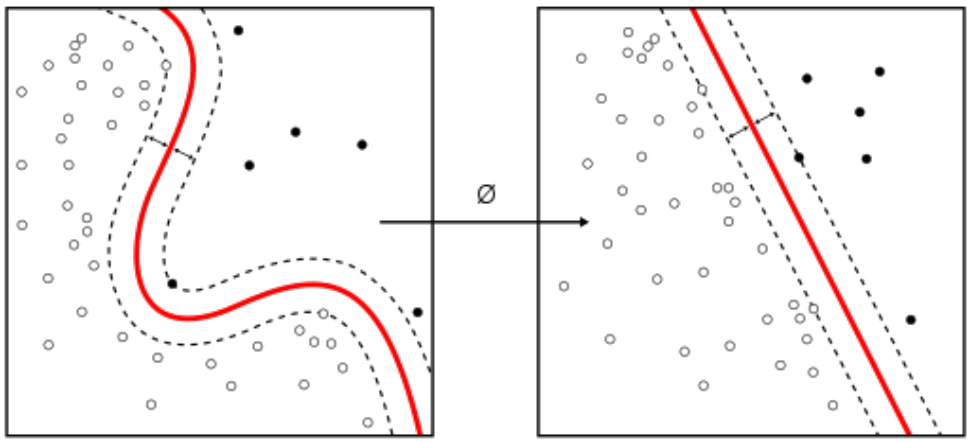


Figure 2.10.: K-Nearest-Neighbors Algorithm[17]

KNN makes predictions through the direct use of the training dataset. Predictions are made for a new instance (x) by searching the entire training set for the K most similar instances (the neighbors) and summarizing the output variable for those K instances. For regression this could be the mean output variable. In classification, this could be the most common (mode) class value. In order to determine which of the K instances in the training dataset are the closest to

a new input, various distance measure methods are used. There are many different distance measure methods, but some important ones are:

- **Euclidean Distance :** It is one of the most popular methods for distance measure. In this method, the square root of the sum of the squared differences between a new point and an existing point is calculated.

$$\textbf{Euclidean } \sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

- **Manhattan Distance :** It is called also City Block Distance. It calculates the distance between real vectors using the sum of their absolute difference.

$$\textbf{Manhattan } \sum_{i=1}^{k}\left|x_i - y_i\right|$$

- **Minkowski Distance :** The Minkowski distance is a metric in a normed vector space which can be considered as a generalization of both the Euclidean distance and the Manhattan distance.

$$\textbf{Minkowski } \left(\sum_{i=1}^{k}\left(\left|x_i - y_i\right|\right)^q\right)^{\frac{1}{q}}$$

There are also some other distance measure methods like Tanimoto, Jaccard, Mahalanobis, and cosine distance. It must be decided which distance measure method should be used according the properties of the dataset. For example, if the input variables are similar in type (e.g. all measured widths and heights), the Euclidean distance measure method is good. However, if the input variables are not similar in type (such as age, gender, height, etc.), the Manhattan distance measure method is better than the Euclidean measure distance method.

The K-Nearest Neighbors Algorithm has advantages and disadvantages. According to [18],the main advantages of KNN are simplicity, effectiveness, intuitiveness and competitive classification performance in many domains. On the other hand, KNN can have poor run-time performance when the training set is large. It is very sensitive to irrelevant or redundant features because all features contribute to the similarity and thus to the classification. The computation cost is also quite high because we need to compute distance of each query instance to all training samples.

K-Nearest Neighbors Algorithm is also a function in OpenCV. The following command shows the usage of the KNN function in OpenCV.

```
void flann::Index_<T>::knnSearch(vector<T>& query, vector<int>& indices,
          vector<float>& dists, int knn, SearchParams& params)
```

The parameters of the function will be described in detail.[19]

- **query :** The query point.
- **indices :** The indices of the KNN are found in this vector.

- **dists :** The distances of the KNN are found in this vector.
- **knn :** Number of nearest neighbors to search for.
- **params :** There are some different optional parameters, which can be used in this function.

2.7 Curve Fitting

Curve fitting is used to find the 'best fit' line or curve for a series of data points. Curve fitting results in the production of mostly mathematical equations which can be used to locate points anywhere along the curve. They are several different types of curve fitting. Some of them are[20]: linear, exponential, polynomial, exponential, power, logarithmic, etc. In this master thesis, polynomial curve fitting was used. Polynomial curve fitting differs from order of the polynomial. Polynomial curve fittings are called different names depending on their orders. First order polynomial curve fittings are called linear regression; second order, quadratic regression; and third order, cubic regression.

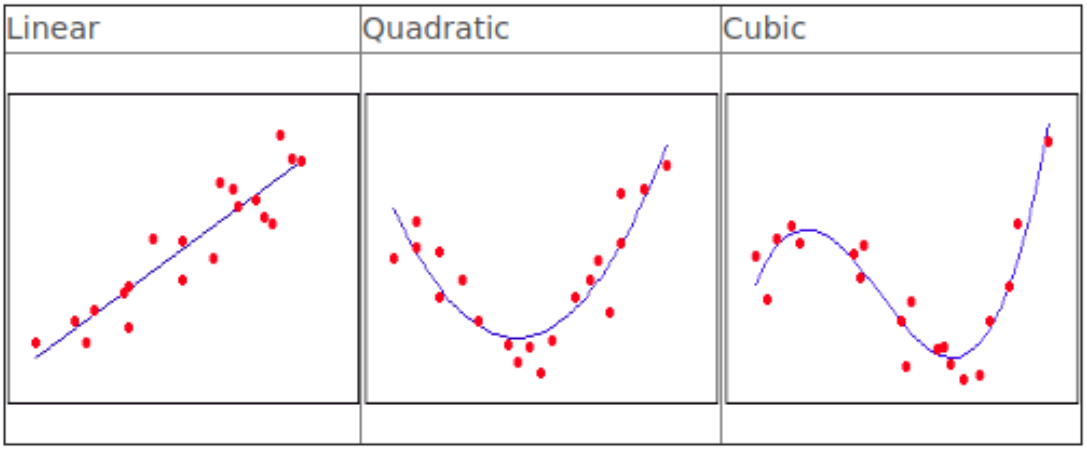


Figure 2.11.: Types of Polynomial Curve Fitting[21]

In this master's thesis, curve fitting is used for getting the best mathematical descriptions of lanes. Curve fitting uses the Hough points which appear on the lanes as input, and give a mathematical equation as output. First order polynomial curve fitting is more suitable for defining straight lines and second order polynomial curve fitting is more suitable for defining curves. There are both straight and curved lanes, so first order polynomial curve fitting is not sufficient for our project. Accordingly, in this master's thesis, second order polynomial curve fitting is used.

There are many different methods for curve fitting. One of the most famous methods is the least squares method, which is also the method utilized in this master's thesis. Another method which can be used instead of least squares algorithm is the interpolation algorithm. As in the curve fitting method, there are different types of interpolation. Some of them are: linear interpolation,

polynomial interpolation, and cubic-spline interpolation. In the interpolation method, the data points are connected to each other. If it is linear interpolation, the data points are connected to each other with a straight line, but if it is cubic-spline interpolation, the data points are connected to each other with a third-degree polynomial. Curve fitting and interpolation each have their own advantages and disadvantages. For example, in curve fitting, the fitted curve does not always pass through the original data points, but in interpolation, the generated curve always does so. On the other hand, if the data set is very large, curve fitting can give the 'best fit' even in low-order polynomials. In this master's thesis, because there are so many data points from the lanes, the curve fitting method is more suitable when compared with interpolation. In addition, if there is noise that occurs far away from the lanes, because of the large size of the data set, this noise is ignored and does not change the fitted curve. In Figure 2.12, there is a fitted curve and interpolation for a small data set.

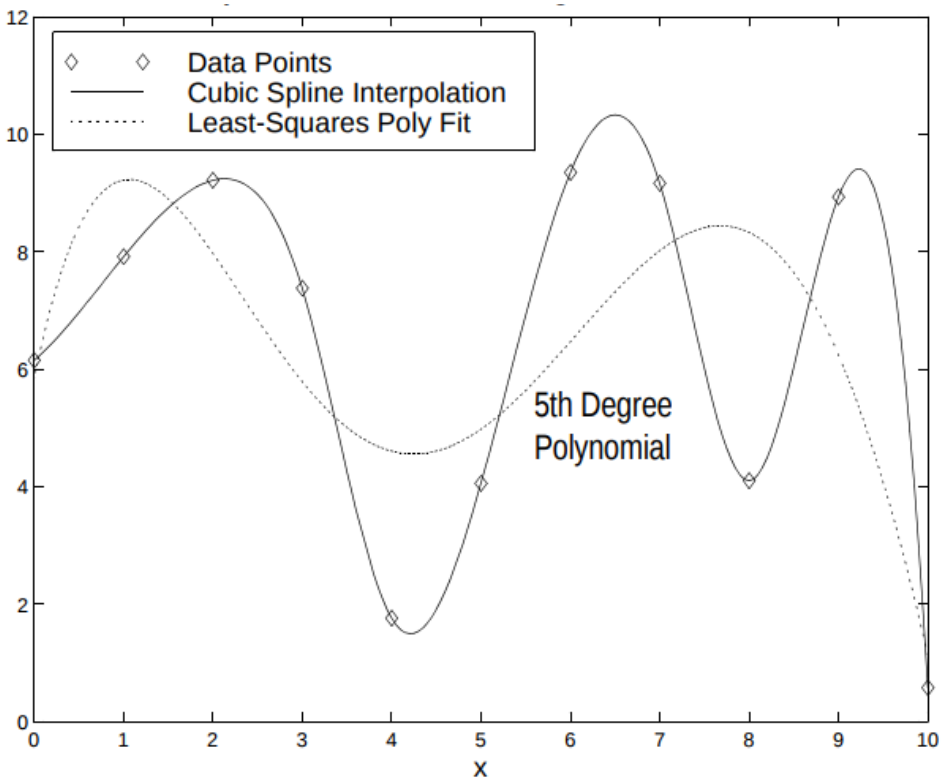


Figure 2.12.: Interpolation and Curve Fitting to Random Numbers[22]

Next, how a polynomial curve fitting is generated using the least squares method will be described.

A data set mainly expresses the relationship between variables with an equation which is generally represented with a k^{th} order polynomial. The general description of k^{th} is :

$$y = a_k x^k + ... + a_1 x + a_0 + \epsilon$$

The general polynomial regression model with the error ϵ typically provides an estimate rather than an implicit value of the dataset for any given value of x. A data set which has N data points

can be described with the maximum order of the polynomial which is k = N - 1, but the lowest possible order of the polynomial is generally chosen.

The aim of the least squares' method is to minimize the variance between dataset values and estimated values from the polynomial equation.

In order to determine the coefficients of the polynomial regression model $(a_k, a_{k-1}, ..., a_1)$ must be solved the following linear equations.

$$\begin{bmatrix} N & \sum_{i=1}^{N} x_i & \dots & \sum_{i=1}^{N} x_i^k \\ \sum_{i=1}^{N} x_i & \sum_{i=1}^{N} x_i^2 & \dots & \sum_{i=1}^{N} x_i^{k+1} \\ \vdots & \vdots & \vdots & \vdots \\ \sum_{i=1}^{N} x_i^k & \sum_{i=1}^{N} x_i^{k+1} & \dots & \sum_{i=1}^{N} x_i^{2k} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_k \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} y_i \\ \sum_{i=1}^{N} x_i y_i \\ \vdots \\ \sum_{i=1}^{N} x_i^k y_i \end{bmatrix}$$

The equations in the stardard form of Ma = b can be solved with many different methods. In this master's thesis, *Cramer's Rule* was used to solve for the polynomial coefficients of curve fitting. With the help of the determinants of the square matrix, M can be solved for in any linear system of equations in order to find the coefficients. Each of the coefficients a_k may be determined using the following equation[23]:

$$a_k = \frac{det(M_i)}{det(M)}$$

In order to find the coefficients of polynomial curve fitting, we have to use the equation shown above, so the determinate of M_i must be divided by the determinate of M. Above, the equation Ma = b was shown, so the determinate of the M matrix can be calculated. However, for the determinate of M_i matrix, the M matrix must be modified. To find the M_i matrix, the i^{th} column must be replaced with the column vector b, which was used in the equation Ma = b. For example, if the M_0 matrix is to be determined, the M matrix must be modified like at the following[23] :

$$M = \begin{bmatrix} N & \sum_{i=1}^{N} x_i & \dots & \sum_{i=1}^{N} x_i^k \\ \sum_{i=1}^{N} x_i & \sum_{i=1}^{N} x_i^2 & \dots & \sum_{i=1}^{N} x_i^{k+1} \\ \vdots & \vdots & \vdots & \vdots \\ \sum_{i=1}^{N} x_i^k & \sum_{i=1}^{N} x_i^{k+1} & \dots & \sum_{i=1}^{N} x_i^{2k} \end{bmatrix} \quad M_0 = \begin{bmatrix} \sum_{i=1}^{N} y_i & \sum_{i=1}^{N} x_i & \dots & \sum_{i=1}^{N} x_i^k \\ \sum_{i=1}^{N} x_i y_i & \sum_{i=1}^{N} x_i^2 & \dots & \sum_{i=1}^{N} x_i^{k+1} \\ \vdots & \vdots & \vdots & \vdots \\ \sum_{i=1}^{N} x_i^k y_i & \sum_{i=1}^{N} x_i^{k+1} & \dots & \sum_{i=1}^{N} x_i^{2k} \end{bmatrix}$$

In this master's thesis, the 2^{nd} order polynomial equation is used, so only the values of $a_0, a_1 and a_2$ must be calculated. To understand curve fitting better, we are going to develop the 2^{nd} order polynomial curve fit for the given dataset.

x	-3	-2	-1	-0.2	1	3
y	0.9	0.8	0.4	0.2	0.1	0

$$M = \begin{bmatrix} 6 & -2.2 & 24.04 \\ -2.2 & 24.04 & -8.008 \\ 24.04 & -8.008 & 180.0016 \end{bmatrix} \quad M_0 = \begin{bmatrix} 2.4 & -2.2 & 24.04 \\ -4.64 & 24.04 & -8.008 \\ 11.808 & -8.008 & 180.0016 \end{bmatrix}$$

$$M_1 = \begin{bmatrix} 6 & 2.4 & 24.04 \\ -2.2 & -4.64 & -8.008 \\ 24.04 & 11.808 & 180.0016 \end{bmatrix} \quad M_2 = \begin{bmatrix} 6 & -2.2 & 2.4 \\ -2.2 & 24.04 & -4.64 \\ 24.04 & -8.008 & 11.808 \end{bmatrix}$$

$$a_0 = \frac{det(M_0)}{det(M)} \Longrightarrow a_0 = \frac{2671.1962}{11661.2736} = 0.2291$$

$$a_1 = \frac{det(M_1)}{det(M)} \Longrightarrow a_1 = \frac{-1898.4602}{11661.2736} = -0.1628$$

$$a_2 = \frac{det(M_2)}{det(M)} \Longrightarrow a_2 = \frac{323.7632}{11661.2736} = 0.0278$$

In this case, the fitted curve function is :

$$y = 0.0278x^2 - 0.1628x + 0.2291$$

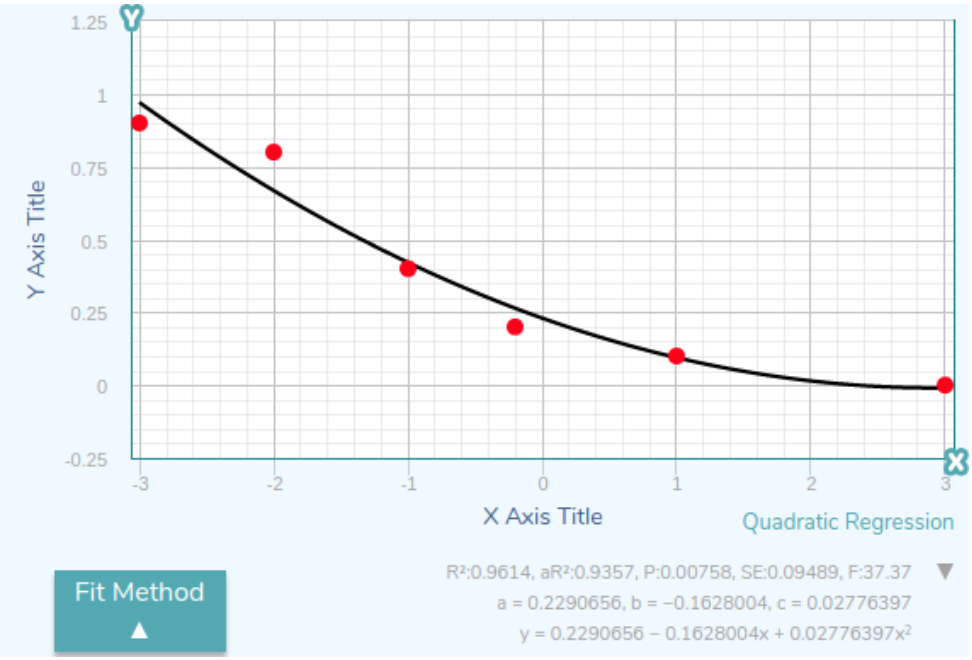


Figure 2.13.: Curve Fitting[24]

3 Related Works

In the automotive industry, one area of particular research interest is autonomous driving, and within this area, lane detection an important component being actively researched. As can be attested by the various projects carried out and scientific papers written on the subject, many different methods can be used in lane detection, each with its own advantages and disadvantages. In terms of disadvantages, some are only able to detect straight lines, for example, while others are not robust in rainy weather or in different light conditions. Some papers also compare the results of different lane detection projects. One of them is from Ammu M Kumar and Philomina Simon from India, which compares some lane detection projects conducted between 2003 and 2014.[25]

According to the literature, one of the methods used most often for lane detection is the 'Hough Transformation'. But there are also some other methods like 'Haar like features', 'Random sample consensus (RANSAC)' or 'Artificial Neural Network(ANN)'. In this chapter, some different methods will be described in order to provide an overview of the methods used in lane detection.

One important algorithm for lane detection was developed by Mohamed Aly, and was also used by Nicolas Acero Sepulveda, who did his bachelor's thesis in the area of lane detection at the Technical University of Darmstadt in 2016.[26] In the beginning of this method, with the aid of Inverse Perspective Mapping(IPM) algorithm, the view is changed from the camera view to the top view. By doing so, the perspective effect is avoided. For the top view frame, the Gaussian and threshold filters are used to filter noise. This method is not very different from Method 2 used in this master's thesis. However, in Aly's algorithm the Random sample consensus (RANSAC) algorithm was used for lane detection, while in this master's thesis, the Probabilistic Hough Transformation was used.

In another method[27] for lane detection, the 'Haar like features' algorithm was used to obtain potential lane points. According to authors, the processing time of this method is approximately 0.12 ms and the detection rate is 90.16%. It is one of the fastest algorithms for lane detection. In this method, because of the perspective effect, the lanes appear diagonal, so a diagonal directional filter called the 'steerable filter' is used, and then the 'Haar like features' algorithm is applied. After this process, the maximum responses are obtained, which are the left and right lanes.

Another robust method[28], which was developed at the Technical University of Braunschweig, was tested successfully in a competition for autonomous automobiles in 2007 called the 'DARPA Urban Challenge'. The 'DARPA Urban Challenge' is different than Carolo-Cup, because the Carolo-Cup is for a 1:10 concept of an automated vehicle, but the DARPA Urban Challenge is for real vehicles. In this project, the IPM technique was used and instead of the RGB color space,

the HSV color space was used, because this format is much more robust under different light conditions. In this project, like in Aly's project, the RANSAC algorithm was used. According the paper written for this project, just ten frames per second can be processed and this project needs to have a modern graphics card.

After considering approaches mentioned above, the algorithmic parts that proved to be efficient in the process of lane detection were selected for evaluation in this research. For preprocessing phase, Sobel Operator and Canny Edge Detector are two techniques that showed best results in terms of the quality of initial detection. Standard Hough Transformation and Probabilistic Hough Transformation are used most commonly as a core algorithm for distinguishing the road marking lines. Comparison of these methods' performance, sustainability, and quality of detection, as well as the final choice justification, can be found in Chapter 5.

4 Implementation

In this chapter, the implementation part of this master's thesis will be described. The test track, which was built before by another student; the hardware part of the model automobile, which was used in this master's thesis; and finally, the software part which was programmed for detecting the lanes will all be explained in detail.

In the Software section in 4.4, all methods programmed and utilized during this master's thesis will be described. In order to better demonstrate the process, pictures of each step, as well as block diagrams of each method, will be shown.

4.1 Test Track

As previously mentioned, the medium-term goal of this master thesis is attending the Carolo-Cup at Braunschweig University, so the test track was prepared according to the Carolo-Cup criteria by Nicolas Acero Sepulveda, who also did his bachelor's thesis with this model automobile. For this test track, two black PVC floor carpets were used and on these floor carpets, the lanes of the track were made by using white electrical tape. The straight part of the track was made on one of these PVC floor carpets and the curved part of track was made on the second PVC floor carpet. The straight part of the track is approximately 2 meters long and the curve radius of the curved part of test track is approximately 1 meter. This curve is the tightest curve at Carolo-Cup, so with this, the test track can be tested in the worst case scenario. In the Carolo-Cup competition, the track is much larger; however, for the purposes of this master thesis, a larger test track in not needed. In Figure 4.1, the test track used in the testing of this master's thesis is shown.

Figure 4.1.: Test Track

4.2 Hardware

While working on a software project, though it is of course important to write an efficient software, it is also important to choose the right hardware. In this project, the model automobile was already prepared for a project course at the Technical University of Darmstadt. However, for this project, it was necessary to use another camera for lane detection, as the original was not sufficient for this purpose. In this chapter, the hardware component of this project and the changes made to the model automobile will be described.

4.2.1 Model Automobile

During the course of this master's thesis, a model automobile which was prepared for the Projectseminar Echtzeitsysteme at Technical University of Darmstadt was used. The chassis, steering mechanism, power train, and engine control were derived from the model-building of a Japanese company, Tamiya. The maximum velocity of the model automobile is approximately[26] 1 m/s and the minimum steering radius is around[26] 90 cm.

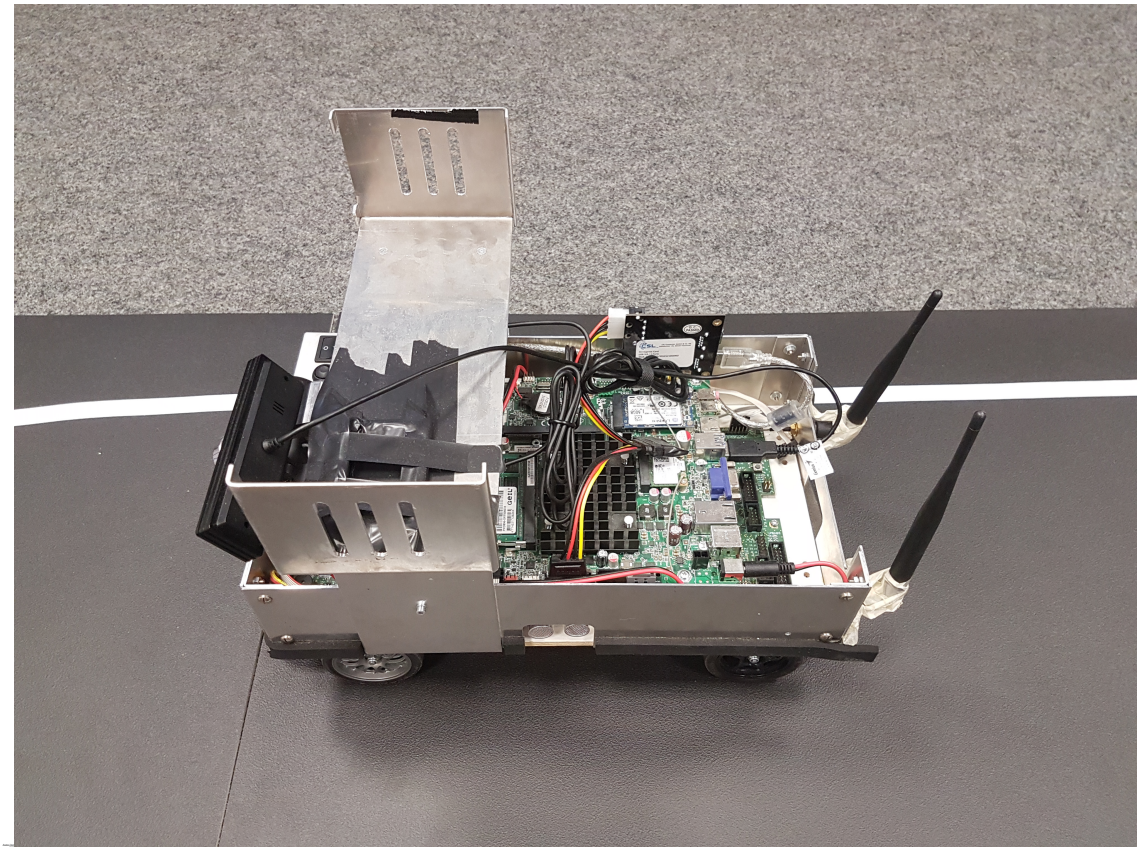


Figure 4.2.: Model Auto

4.2.2 Microcontroller and Main Board

In this model automobile, there is a microcontroller and a main board. The microcontroller is used for controlling steering and receiving the measurements from ultrasonic sensors and hall effect sensors. The 16-bit microcontroller is from MB96300 series from Fujitsu company.

The main board on the model car is from PD10BI-MT ThinMini-ITX series from the MiTAC company. This main board communicates with the microcontroller over an UART interface through USB connection. On this main board, according to the website of the MiTAC company[29], there is an Intel Celeron J1900 Quad core processor with integrated graphics. Furthermore, there is an 8 GB DDR3-1600 RAM and 1Gbit/s Ethernet, VGA, HDMI, USB 2.0/3.0, SATA ports and an Intel Dual Band Wireless AC 7260 Network adapter, which is connected to two external WLAN antennas. A 60 GB Kingston SSD-Harddisk is connected over an integrated PCI-Express Port. A 3200 mAh Li-Fe battery is used as a power supply.

4.2.3 Camera

The camera is one of the main components of lane detection and accordingly, autonomous driving. For this thesis, I had to research the most suitable camera because all cameras have different properties.

At the beginning of the Projectseminar Echtzeitsysteme, the Logitech C270 HD Webcam was being used. The resolution of the camera is 1280x960 pixels and the Frame per Second (FPS) value is 30 Hertz (Hz) at a 640x480 pixel resolution. The field of View (FOV) is just 60 degrees. The problem with this camera is that if there is a curve, the camera cannot see all of the lanes, and thus is not very suitable for lane detection. When I started my master's thesis, there was a Kinect v2 camera on the model car. The Kinect v2 camera was developed by Microsoft and released in 2013. This camera has a depth sensor with a resolution of 512x424 pixels and its FOV is 70x60 degrees. The FPS value is 30 Hz at a 512x424 pixel resolution. This camera also has a color camera with resolution of 1920x1080 pixels and a FOV of 84.1x53.8 degrees. The FPS value is 30 Hz at a 1920x1080 pixel resolution. This camera had two main disadvantages for this master's thesis. The first disadvantage is the FOV value of camera. This value is better than the value of Logitech C270 camera but it is still not enough for curve lane detection. The second main disadvantage is the location of the color camera. The color camera of this camera is not in the middle of camera, but on the right. This is a disadvantage for us because when there are curves going left as opposed to right, the camera is unable to see the left and even perhaps the middle lane of the track. Thus, this is problematic for lane detection.

Due to these reasons, it was necessary to select a camera which has a sufficiently high FOV value. After doing research, I decided that the Genius Widecam F100 camera is the best choice for this master's thesis because this camera has a FOV value of 120 degrees and it can also be used with the Linux Operating System. The resolution of this camera is 1920x1080 pixels and the FOV value is 120 degrees. The FPS is 30 Hz at a 1920x1080 pixel resolution. With this camera, it is possible to detect most if not all lanes, including when there are curves.

Figure 4.3.: Genius 120-degree Ultra Wide Angle Full HD Conference Webcam(WideCam F100)

4.3 Response Time of the System

Before the lane detection project was started, it was necessary to conduct an experiment with the system. The aim of this experiment was to measure the response time of the entire system. In order to measure it, the camera had to detect something and then the software had to set an I/O pin on the main board. The amount of time this process took was in fact the response time being searched for.

For this experiment, a camera, a function generator, an LED, and an oscilloscope were needed. A circuit with a power supply and an LED were designed. The LED was placed in a small box with a camera, and one probe of an oscilloscope was connected to the LED while its other probe was connected to an I/O Pin on the mainboard, which could be set with software.

Before the results were measured, the frequency of the camera was set to 10 Frames per Second(FPS) and the power supply was set to 3.28 Hz square wave. This means that the LED is supposed to blink approximately every 305 ms.

After the supply was turned on, the LED started to blink. For this experiment, a small firmware able to detect the color red was written in order to detect when the red LED lit up.

The LED and a camera were placed together in a box and the box was closed, and so the inside of the box was totally dark. When the red LED lit up, the camera detected it and the firmware set an I/O pin on the main board. For this experiment, two different cameras (Logitech Webcam C270 model and Logitech Webcam C905 model) were used and the results were very similar.

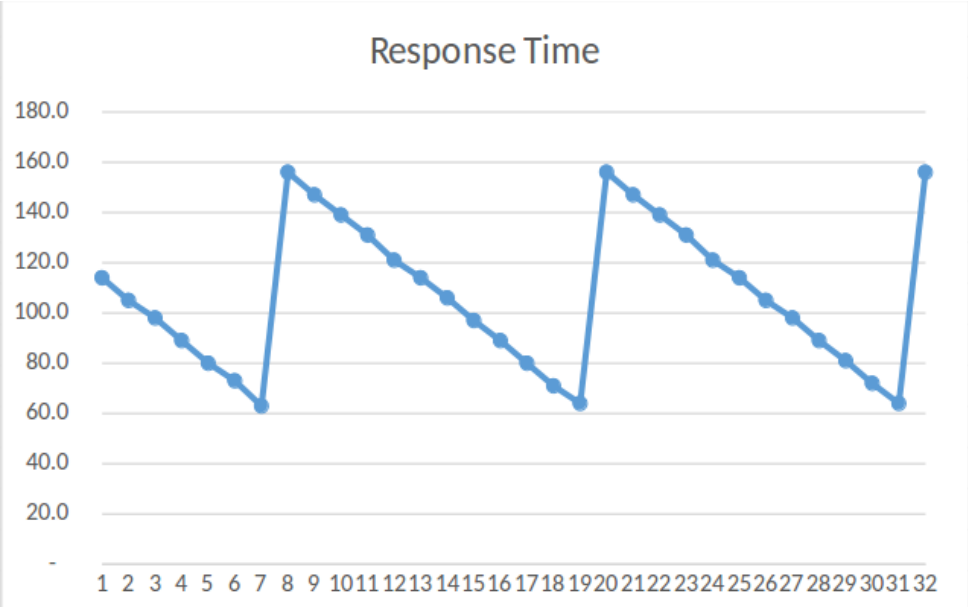


Figure 4.4.: Response Time of the System

As in Figure 4.3 seen, the maximum response time of the system was 156 ms, the minimum response time of the system was 63 ms, and the average time was 106.6 ms.

This Figure shows that the frames are steadily sent by the camera. In addition, if the LED lights up and is immediatly followed by new frame, then the light is detected. If the LED goes dim and is followed by a new frame, then the response time of the system is minimum, which is measured at 63 ms. On the other hand, if the frame comes and then immediately the LED changes its status (on/off), the new status of the LED is not detected until the new frame is received. The FPS value of the camera is ten, which means that the system must wait nearly 100 ms for the new frame and after that, the response time of the system must taken into account, so the maximum response time of this total system is around 156 ms. This means that the FPS value of the camera is important for response time of the entire system. Because it is desirable to increase the FPS value of the camera, decreasing the computing times of lane detection methods is also desirable, as this would allow for an increase in the FPS value.

4.4 Software

In this chapter, the software algorithms defined in this master's thesis will be focused on. With the aid of program flow charts and explanations of all their steps, the algorithms will themselves be explained in detail. In order to find the best solution, five different source code versions were generated. For all these source codes, the computing times were calculated and compared in terms of which solution can detect the lanes better. In the following pages, there are detailed explanations of the methods utilized. The development environment and the software utilized in this master's thesis will be also described.

4.4.1 Development Environment and Related Software

As also mentioned at subsection 4.2.2, in this project the previously introduced main board was utilized. One of the compact and fast versions of the Linux 16.04 operating system, *Lubuntu* was installed in this main board.

The version *Kinetic* of ROS was used for implementation of this master's thesis. ROS is the abbreviation of **R**obotic **O**perating **S**ystem, which is a flexible framework for writing robot software (i.e. collection of tools and libraries for robot software development). On the ROS wiki page[30], ROS is defined as an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.

For using prepaid image processing functions, an open-source computer vision and machine learning software library called OpenCV was used. According to the OpenCV website[31], there are more than 2500 optimized algorithms in the OpenCV library and OpenCV has a user community of more than 47 thousand people.

ROS can be programmed with Python, C++ or Lisp programming languages and OpenCV can be programmed with Python or C++ programming languages. In this master's thesis, C++ was used.

Because the monitor cannot always be connected to a PC via a cable for practical reasons, in order to see if the algorithm can detect the lanes, the on-board PC must be connected to a PC remotely. For that, there is a software used which can connect two different computers which are run by the Linux operating system. As the first step, a wireless network must be created in one of these computers and the other computer must be connected to this wireless network. Then the IP address of the model car must be determined or a static IP address must be given to the model automobile. As the last step, 'Remmina Remote Desktop Client' must be used in the PC and the IP address of the model automobile (Server) must be given. After that, the model automobile and the PC are connected to each other. In this software, the quality of screen sharing and the speed can be adjusted.

4.4.2 Preprocessing

There are many different possibilities for lane detection algorithms. Of course, each has its own advantages and disadvantages. In this master's thesis, some methods were defined, and in this chapter, these methods will be explained in detail.

In all of these methods, some processes are common, and this is called the preprocessing phase. In Figure 4.5, the steps of the preprocessing part are shown.

At the beginning, the frames are obtained from the camera via ROS-Topic. ROS uses different image formats than OpenCV, which uses the image format Matrix(Mat). The frame obtained via ROS-Topic must be converted from the ROS image data type to a Mat object. In order to convert the frame, a ROS-Package *cvbridge*[32] was used. cvbridge converts the ROS image format to a Mat Object, which is the OpenCV Image Format.

Mat is a class which has two parts. The parts are a matrix header and a pointer to the matrix containing the pixel values. The matrix header contains information like the size of matrix, storing method, etc. It always has a fixed size but the size of matrix is variable from image to image.

There are many methods which can store the pixel values to the Mat object. In this case, the color space and the data type utilized can be chosen. For gray images, it is easy to choose the color space because there are just two colors : black and white. By changing the density of colors(black and white), it is possible to create many shades of gray. There are more methods for color images. Color images generally have three or four channels. These three channels are used for RGB color values. The RGB colors are based on the colors red, green, and blue, which can all be detected by the human eye. For the transparency of a color, a fourth channel called alpha(A) can be used.

There are also another color formats, which have some advantages[33].

- The RGB format is quite similar to the human eye, but the OpenCV display system uses the BGR format, which uses another order of colors.
- The HSV and the HLS formats are more natural ways to display colors. They decompose colors into their hue, saturation, and value/luminance components. Another advantage of the HSV and the HLS formats is that they are less sensitive to the light conditions of the input image.
- In JPEG image formats the YCrCb format is used.
- If the distance of a given color to another color is to be measured, CIE L*a*b* format is more suitable than others.

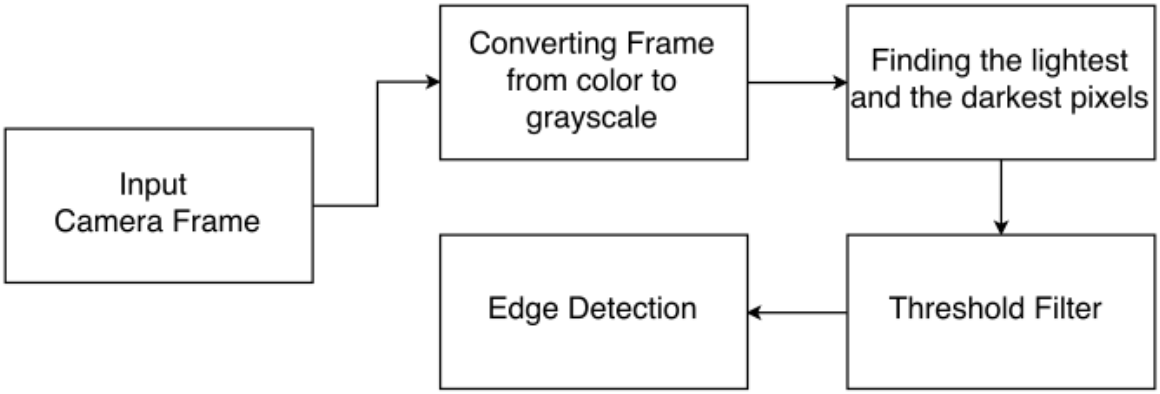


Figure 4.5.: Block Diagram of the Preprocessing Phase

After the frame from Camera via ROS-Topic is received, the color frame has to be converted to a grayscale frame. For detection lanes, Hough Transformation is used and for Hough Transformation, the grayscale format of the input image is needed. Converting a frame from BGR format to grayscale format has some advantages, the main advantage being the processing time. Normally color frame matrix content has three or four channels, but grayscale frame matrix content has just one channel, so grayscale frame matrix size is much smaller compared to color frame matrix size. Because of this, the image processing time is much less for the grayscale format compared to BGR format. In order to convert the BGR formatted frame to a grayscale formatted frame, the *cvtColor* function from OpenCV is used.

For stable lane detection, the light conditions must be considered. Because of this, after converting the frame with BGR format to grayscale format, the lightest and darkest pixels have to be searched for. In order to find the lightest and the darkest pixels, the following function in OpenCV was used. This function will be explained in detail.[8]

```
void minMaxLoc(InputArray src, double* minVal, double* maxVal=0, Point*
        minLoc=0, Point* maxLoc=0, InputArray mask=noArray())
```

- **src :** Input single channel array.
- **minVal :** Pointer to the returned minimum value.
- **maxVal :** Pointer to the returned maximum value.
- **minLoc :** Pointer to the returned minimum location.
- **maxLoc :** Pointer to the returned maximum location.
- **mask :** Optional mask used to select a sub-array.

After finding the lightest and darkest pixels, it is possible to estimate the lighting conditions. These values are then used in the next step. After this processing, a filter is applied to all frames, transforming images into binary images by transforming each pixel according to whether it is inside or outside a specified range. The user chooses a threshold value to process. If a pixel is greater than this value, it is assigned an 'inside' value; otherwise, it is assigned an 'outside' value. Depending on the lightest and darkest pixel values, the threshold value changes. Through this dynamic parameter, the lanes are more able to be more clearly detected and noise can be cancelled more successfully. For this thresholding operation, the following function in OpenCV was used. This function will be explained in detail[34].

```
double threshold(InputArray src, OutputArray dst, double thresh, double
                                    maxval, int type)
```

- **src :** Input single channel array, which is 8-bit or 32-bit floating point.
- **dst:** Output array which has the same size and type as src.
- **thresh :** Threshold value.
- **maxval :** Maximum value to use with the THRESH_BINARY and THRESH_BINARY_INV thresholding types.
- **type :** Thresholding type, which can be THRESH_BINARY, THRESH_BINARY_INV, THRESH_TRUNC, THRESH_TOZERO and THRESH_TOZERO_INV.

After the threshold filter, an edge detection filter must be applied. In this master's thesis, the Sobel-Operator is used. This is explained in Chapter 2 in detail.

This preprocessing part is common for all cases but after this process, the cases diverge.

4.4.3 Method 1 : Hough Transformation + Rectangle Method + Curve Fitting + IPM

The algorithms and their orders which was used in this method, will be explained step by step. At Figure 4.6, the block diagram of Method 1 is shown.

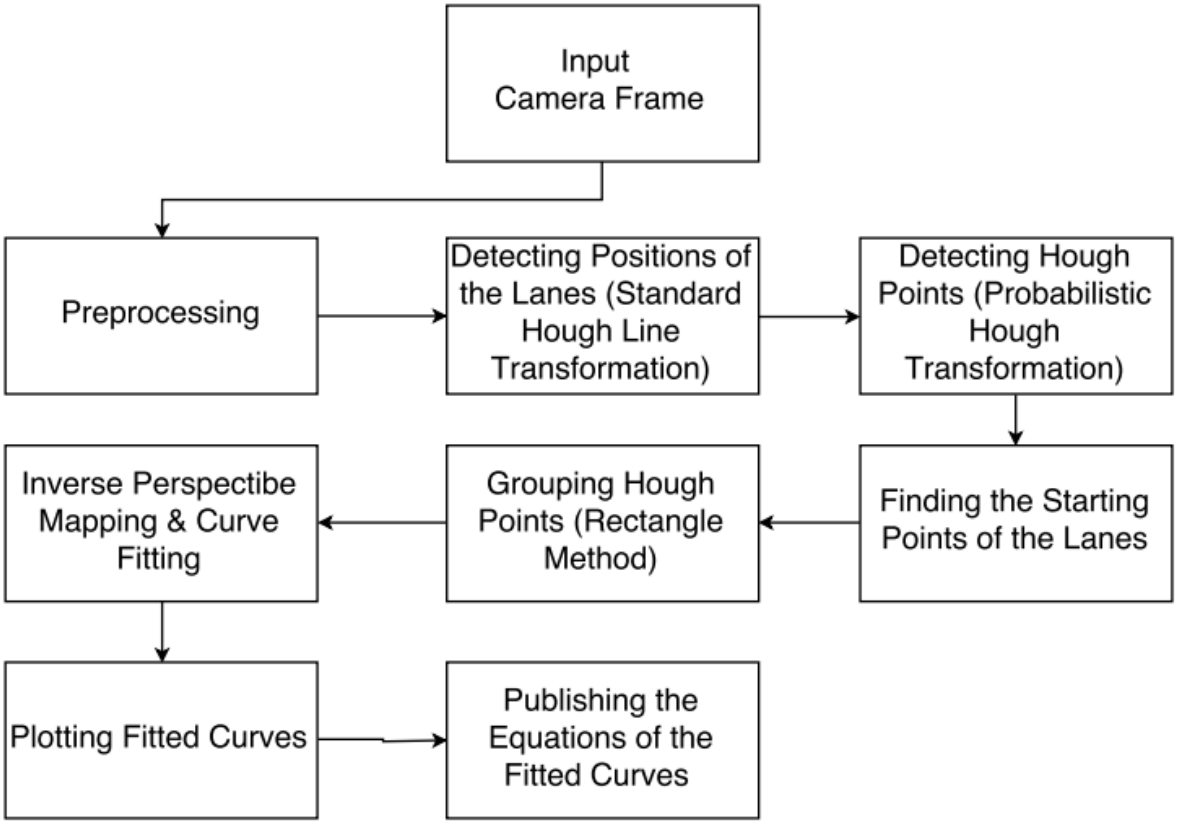


Figure 4.6.: Block Diagram of Method 1

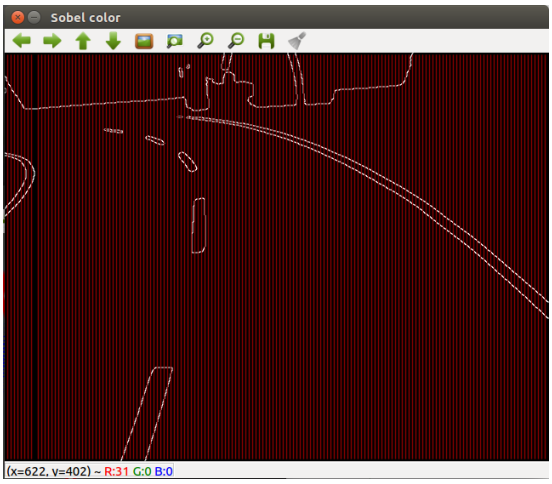
Step 1 : As with all methods, the first step is the preprocessing phase. When the preprocessing phase is over, it must be determined which pixel columns the lanes lie between.

Step 2 : Secondly, the positions of the lanes should be found. In order to find the positions of the lanes, the Standard Hough Line Transformation is used. But the Standard Hough Line Transformation should not utilize the entire frame; rather, the frame is cropped. There is an advantage to cropping the frame, which will be explained in detail.

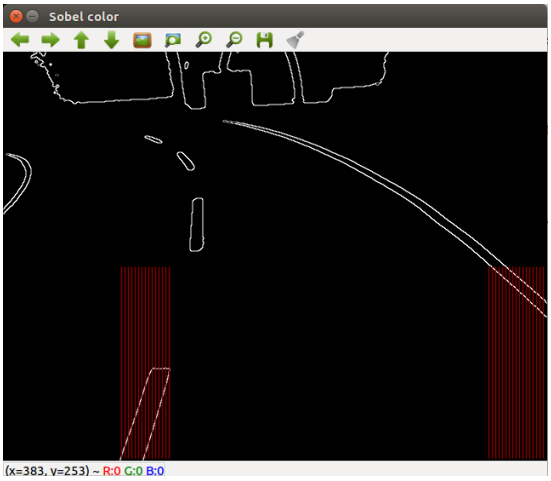
In this project, the Standard Hough Line Transformation is used differently than usual. Normally, the Standard Hough Line Transformation detects the straight lines but here, it checks if there are two edges in the same pixel column of the frame. In other words, if there is a lane, there are two edges of the lane in the frame so if there is a lane, there are also Hough Transformation lines. For the Standard Hough Line Transformation, the function of OpenCV is used. As previously explained in Section 2.5.1, in order to use this function in OpenCV, some parameters must be set. One of these parameters was the 'threshold', which decides the minimum number of the points to detect a line. In a lane, there are two edges, so the Standard Hough lines must be drawn if two points in the same vertical are detected. These two points are only searched for vertically, so another parameter in the Standard Hough Line Transformation function from OpenCV is used for the angle, which has to check the points in that angle. Because only vertical edges are checked, this angle value, named 'θ', is set to 'CV_PI(180 degree)'.

The frames obtained from the camera are at a resolution of 640x480 pixels, which is the default value. The resolution of the camera can be increased and decreased by adjusting its settings. The original height of the frame was 640 pixels in this method, but as previously mentioned, the frame was cropped. As seen in Figure 4.7a, the camera detects the edges that do not belong to the track. When the frame is not cropped, the detection of irrelevant edges would result in the undesired production of Standard Hough Transformation lines (shown in red). Thus, the Hough Transformation is only used for the lower 230 pixels, if more than 230 pixels are used for generating the Standard Hough Transformation lines.

As seen in Figure 4.7b, the red lines are shown only in the last 230 pixels of frame height and do not cover the irrelevant parts of the frame. If there is a lane, the red lines are very close to each other, but if there is no lane, there is a distance of at least 50 pixels between the red lines. In Figure 4.7b, there are two different groups of red lines. In each group, there is a lane. Thanks to the Standard Hough Line Transformation, it is known which lanes are between which pixel columns. The group of Standard Hough Transformation lines on the right indicates the right lane and the group of Standard Hough Transformation lines on the left indicates the middle lane.



(a) Standard Hough Transformation without Cropping Image

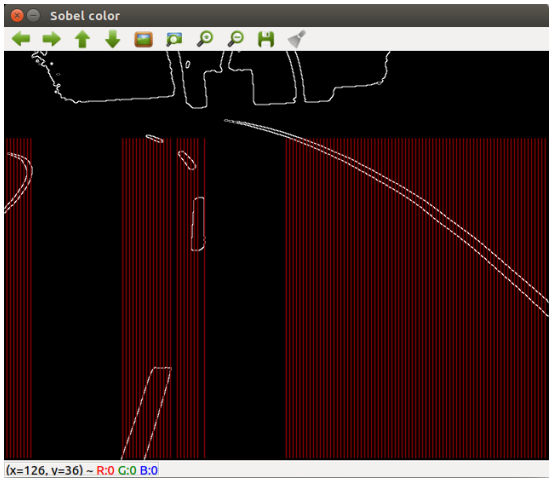


(b) Standard Hough Transformation with Cropping Image

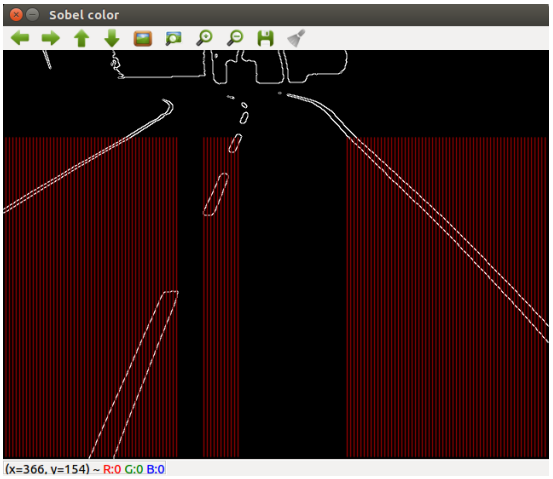
Figure 4.7.: Detecting Lane Positions

As previously explained, the positions of the two lanes (right and middle) were detected. However, even if more than the lower 230 pixels of the frame are used, the positions of the all lanes can be detected (if all three lanes can be seen in the frame), but in this case, another problem occurs. As seen in Figure 4.8a, the Standard Hough Transformation is implemented for the lower 380 pixels of the frame. As shown in the same Figure, if there is a curve, everything works fine and the positions of the all lanes can be detected. However, if there is a straight lane like in Figure 4.8b, the position of the lanes can not be detected correctly. The beginning of the middle lane is grouped with the left lane, so it causes lane detection problems. In other words,

there are again three different groups for three different lanes but each group of red lines fails to separate the lanes clearly from each other. As a result, the correct starting points of the middle and left lanes are not able to be found. In order for the lanes to be detected clearly, the Standard Hough Line Transformation must be used for only the lower 240 pixels of the frame. At the end of this step, the positions of the middle and right lanes are detected.



(a) Standard Hough Transformation in the Curve Lane



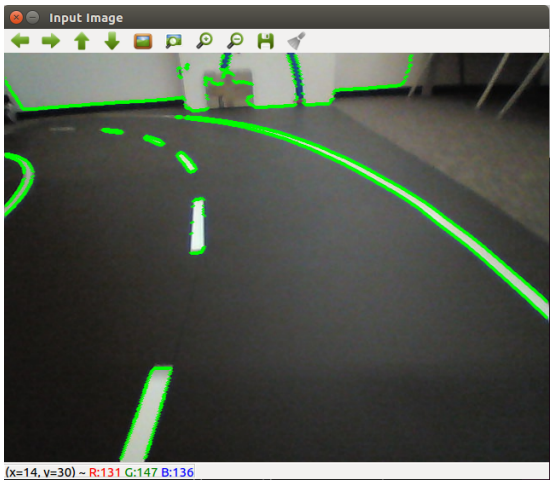
(b) Standard Hough Transformation in the Straight Lane

Figure 4.8.: Standard Hough Transformation Detection Problem

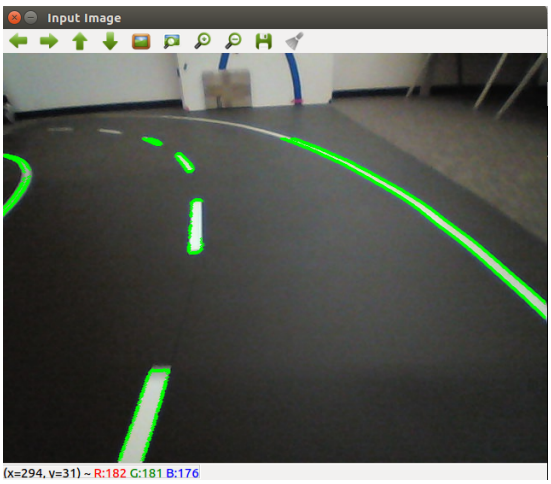
Step 3 : After the positions of the middle and right lanes are detected, the Probabilistic Hough Transformation must be implemented. Probabilistic Hough Transformation finds the pixels, which are on the edges of the lanes. The Probabilistic Hough Transformation is a bit different than Standard Hough Line Transformation because Standard Hough Transformation is more suitable for straight lanes. However, if there is a curve, the Standard Hough Transformation is not able to detect lanes very well. As previously mentioned in Section 2.5.1, a line can be represented as y = mx+c, or in parametric form, as ρ = x cos θ + y sin θ, where ρ is the perpendicular distance from origin to the line, and θ is the angle formed by this perpendicular line and the horizontal axis measured counter-clockwise. However, the process differs in the case of the Probabilistic Hough Transformation. A line is represented by two or more points. If the Probabilistic Hough Transformation finds at least two points from the same line, it represents the two end points of these lines.

In this project, the function for the Probabilistic Hough Transformation from OpenCV is used. It detects lines, but all lanes consist of small lines, even if they are curved. In this project, these small lines on the lanes are detected. These small lines have start and end points, so instead of the drawing these small lines, the start and end points of these small lines are shown. All of these points (pixels) are on the lanes. As previously explained in Section 2.5.2, the parameters must be set for Probabilistic Hough Transformation. Here, the value 'ρ' is the resolution in pixels. In this project, this value is set as two and it means that for every two pixels, lines

are searched for. Another value in this function is 'θ', which is the resolution in radians. In the Standard Hough Transformation, this value is chosen as 180 degrees, which means only vertical lines can be detected. However, in the Probabilistic Hough Transformation, this value is set as one pixel, which is the minimum value that can be chosen. The 'threshold' value decides the minimum number of points in the same line in order to detect a line. This value used in this project is two because it is enough to detect a line if there are only start and end points in a line. Another parameter in this function is 'minLinLength', which is the minimum length of the line. In this project, Probabilistic Hough Transformation is used to detect small lines in the lanes, so this value must be very small. In this project, two is selected as this value, because if there are two pixels next to each other in the same lane, these pixels must be detected. The last parameter of the Probabilistic Hough Transformation function is 'maxLineGap', which allows for the detection of lines which are farther away from each other than the value selected for this parameter. In order to detect a large number of Hough Points, which is needed for stable line detection, the value selected is two, meaning that the Hough Points farther than two pixels away from each other are detected. All of these parameters can be changed and affect the computing time.



(a) Original Image with Probabilistic Hough Transformation without Cropping



(b) Original Image with Probabilistic Hough Transformation with Cropping

Figure 4.9.: Probabilistic Hough Transformation Points

In Figure 4.9a and Figure 4.9b, the Probabilistic Hough Transformation Points (green pixels) are shown. The Probabilistic Hough Transformation detected a large number of pixels. By changing the parameters of the Probabilistic Hough Transformation found in OpenCV, fewer Hough Points on the lanes are detected, but detecting too few Hough points can also cause some problems for lane detection. On the other hand, detecting a large number of Hough Points results in more computing time, which is undesirable. So in this case, the parameters of Probabilistic Hough Transformation function in OpenCV must be optimized. Thanks to optimization, the best solution (less computing time and good lane detection) is found. As previously mentioned,

detecting a larger number of Hough Points increases the computing time. Due to this reason, the Probabilistic Hough Transformation is not used in the entire frame. As seen in Figure 4.9a, in the upper pixels of the frame, there are many Probabilistic Hough Points which are not relevant to the lane, so the upper 100 pixels of the frame are cropped before the Probabilistic Hough Transformation is implemented. In Figure 4.9b, the Probabilistic Hough Points are just on the lines. When the upper 100 pixels are cropped, the algorithm is much faster when compared to leaving using the entire frame.

Step 4 : It is now known in which pixel columns the middle and right lanes lie (Step 2), and the points (pixels) on the lanes are also shown thanks to Probabilistic Hough Transformation (Step 3), and after these two steps, the starting points (pixels) of the lanes can be found. In order to do that, the Hough Points for each group of Hough lines found in Step 2 are compared with each other, and then the lowest pixels in the columns should be found. If the camera can see three lanes, then the different starting points for the three different lanes must be found. This means that this process must be done for all lanes which can be seen in the frame. As seen in Step 2, the position of the left lane is not detected, although the camera is able to see the left lane. Because the left lane is not seen in the lower 230 pixels, the position of the left lane is not detected. In this case, the Hough Points, which lie to the left of the middle lane, are compared with each other and then the lower pixel is found. It is the starting point (pixel) of the left lane.

Step 5 : The next step in this method is getting the Hough Points which are relevant to the lanes. Now, for each lane, the Hough Points must also be grouped. In order to get the Hough Points relevant to the lanes, the 'rectangle method' is used. In the ectangle method, a rectangle is drawn for each lane at the starting point found in Step 4, and the coordinates of all of the Hough Points in that rectangle are saved in a vector. The highest Hough Points in the rectangle must then be found. From that point, another rectangle must be drawn, but the sizes of the rectangles become progressively smaller, because the objects seem smaller the further away they are from the camera. The sizes of the rectangles are also similar for left and right lanes, but there is an exception with regard to the middle lane. The middle lane has dashed lines, so the rectangles in the middle lane must be bigger than in the left and right lanes. As seen in Figure 4.10 (rectangles shown in blue), with this method, the noise and the Hough Points not relevant to the lanes are removed. For the 'rectangle method', a function is defined in the source code. The inputs of this function are: the input image, in which also the Probabilistic Hough Transformation was used; the starting point of the lane; the vector of the Hough Points; and the lane for which this function is used must also be defined. In which lane the rectangles are to be drawn must also be known, because as previously mentioned, due to the dashed lines in the middle lane, the size of the rectangles in the middle lane is bigger when compared to the rectangles in the left and right lanes.

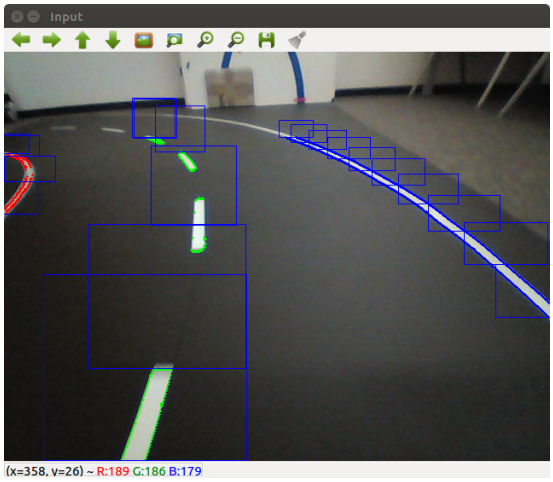


Figure 4.10.: Rectangle Method

Step 6 : The next step of this method is curve fitting. Curve Fitting is an algorithm which gives a mathematical description and this mathematical description provides the best fit for a series of data points. But in this master's thesis, the curve fitting to be performed is modified. In this curve fitting, the coordinates of Hough Points are used but the x and y coordinates of these Hough Points are swapped, because the y-axes of these coordinates have more range than the x-axes. As a result, this swap raises the stability of the curve fitting.

After using the Hough Points as input, three different mathematical equations are produced. One of these equations is for the left lane, another is for the middle lane, and the last equation is for the right lane. In order to produce these equations, naturally, only the relevant Hough points are used. For example, for left lane curve equation, only the Hough Points from the left lane are used. For the curve fitting, a function is implemented in the source code. One of the inputs is the input image. Another input of the function is the vector of the Hough points which are relevant for that lane. In other words, if the curve fitting function is used for the right lane, then the Hough points relevant to right lane must be used. These points are grouped in Step 5 with the rectangle method. If it is desired for the fitted curves to be shown in the frame, in order to differentiate the fitted curves from each other, colors can be chosen. As a result, the color value is also another input in this function. Available colors of fitted curves are red, green, and blue. Other colors can also be used. The last input value of this function is 'IPM'. In order to track the lanes, the equations of the fitted curves must be given from the bird's-eye perspective so the Inverse Perspective Mapping (IPM) algorithm must be applied. For IPM, a function from OpenCV, which is called 'findHomography', was used. Thanks to this function, all Probabilistic Hough Points on the lanes can be converted to the bird's-eye perspective. It is also possible to convert all pixels from the camera perspective to bird's-eye perspective, but in that case, it is also necessary to convert 640x480 pixels, which means 307200 pixels in total. Because this is computationally expensive, only the Hough Points relevant to the lanes are converted. Much fewer pixels are converted when compared to converting all pixels in the frame. The IPM algorithm is very computationally expensive in and of itself, so converting

fewer pixels makes this method more efficient. As a result, the inputs of the fitted curves are the transformed Hough Points, which are converted from the camera perspective to the bird's-eye perspective.The output equations of the fitted curves are therefore also for the bird's-eye view.

Step 7 : The next step of this method is plotting the fitted curves produced by the curve fitting function. In this case, the fitted curve starts from the first pixel and continues to the 480th pixel vertically and the output values of the equation for each pixel in the column are found. For each line, the fitted curves are plotted in different colors. To be sure if the lanes are correctly detected, the input image is also transformed and the fitted curves are plotted in Figure 4.11a.

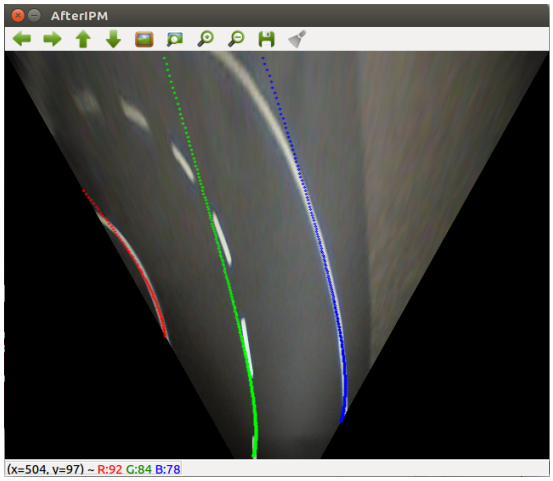


Figure 4.11.: Curve Fitting

Step 8 : The last step of this method is publishing the coefficients of the equations of the fitted curves from the bird's-eye view. In order to activate the automobile for autonomous driving, only the equations of the fitted curves are needed. For this step, a ROS Topic must be created. The frequency of the rostopics can be adjusted.

4.4.4 Method 1(b) : Resize + Hough Transformation + Rectangle Method + Curve Fitting + IPM

This method is very similar to Method 1 in Section 4.4.3. The only difference between these two methods is that in the Method 1, the resolution of the frame is 640x480 pixels and in this method, the resolution of the frame is four times smaller, with a resolution of 320x240 pixels. In order to reduce the resolution of the frame, the width and height values in pixels must be sent to the camera before it is started. While reducing the resolution of the frame has some advantages, it also has some disadvantages. These will be explained in the Chapter 5. The output frame of this method is shown in Figure 4.12.

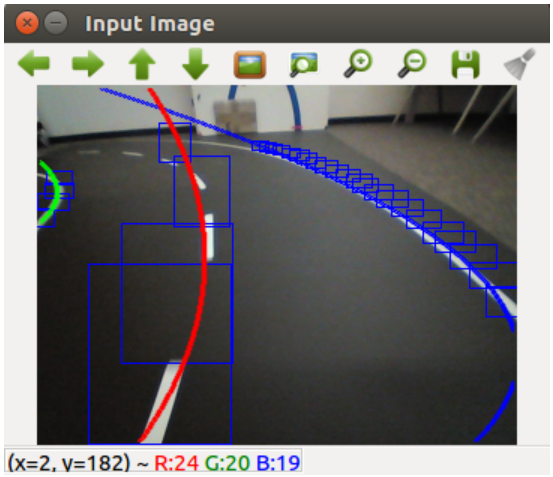


Figure 4.12.: Output Image(320x240 pixels resolution)

4.4.5 Method 2 : IPM + Hough Transformation + KNN + Curve Fitting

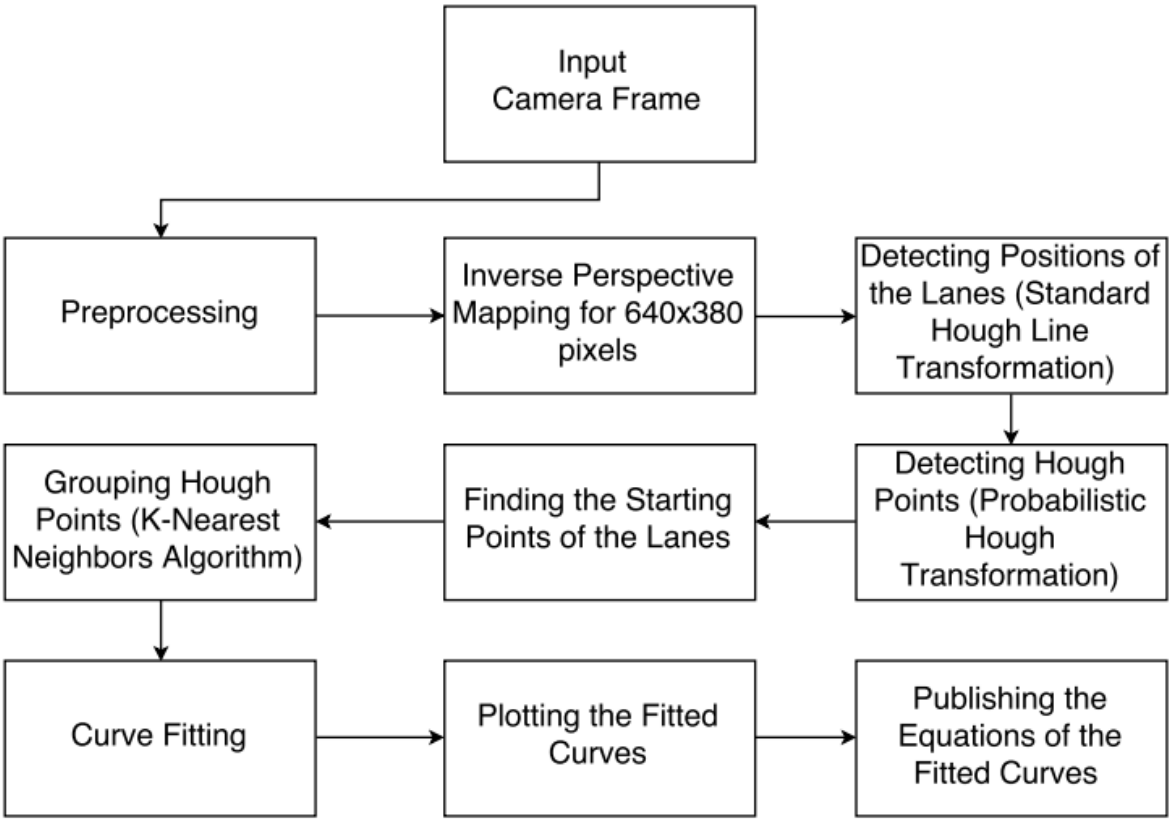


Figure 4.13.: Block Diagram of Method 2

Step 1 : As previously mentioned, the preprocessing part is common for all methods. So in this method, the preprocessing part was also the first step.

Step 2 : In this method, the frames are taken by the camera, then the upper 100 pixels of the frame are cropped, otherwise, the lanes appear far away from the bird's-eye perspective and thus appear smaller (more info in the Section 2.2). Then the cropped frames are converted from the camera perspective to the bird's-eye view perspective. In Section 4.4.3 (Method 1), only the curve fitting pixels were converted from the camera perspective to the top of the track perspective. The time needed to convert 640x380 pixels (243200 pixels in total) is a bit more when compared to Method 1. The original frame, which can be seen in Figure 4.14a, is converted to the frame which can be seen in Figure 4.14b by OpenCV 'findHomography' function.

(a) Original Image **(b)** A Image with Inverse Perspective Mapping

Figure 4.14.: Inverse Perspective Mapping

Step 3 : In Method 1, the Standard Hough Transformation is applied to only the lower 230 pixels. In this methood, it sufficient to apply the Standard Hough Line Transformation to only the last 200 pixels of the frame, because these 200 pixels are able to cover all lanes in the frame. In Figure 4.15a, all lanes can be distinguished thanks to the Hough lines.

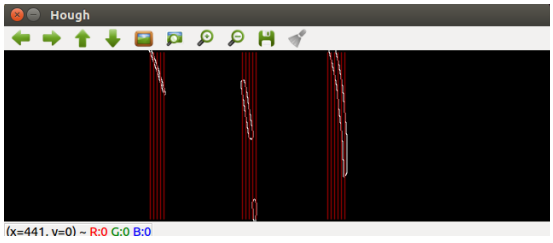


Figure 4.15.: Standard Hough Line Transformation after IPM Method

Step 4 : After finding which pixel columns the lanes lie between in Step 3, the Probabilistic Hough Transformation was used. Thanks to Probabilistic Hough Transformation, the Hough Points appear on the lanes. As mentioned in Method 1 in Section 4.4.3, there are some parameters in Probabilistic Hough Transformations, so the number of Hough Points can be descreased

or increased. Decreasing the number of Hough Points also decreases the computing time but decreasing the number of Hough Points by too much can decrease the accuracy of lane detection. Therefore, the parameters must be set in an optimal way.

Step 5 : We have already found out which pixel columns the lanes lie between. The Hough Points must now be grouped according to lanes. If the camera can see all three lanes, then there must also be three different groups of Hough Points. However, if there is a curve, the camera can see just the right and middle lane, so the Hough Points must be grouped into two groups in this case. For each lane, the starting points of the lanes must be found. For that, all Hough Points in a group must be compared with each other. The Hough Points which are the closest to the bottom of the frame are the starting points.

Step 6 : The next step of this method is the k-nearest neighbors algorithm (KNN). KNN is a learning algorithm. Here KNN is used instead of the rectangle method used in Method 1. The KNN algorithm divides the frames into some equal-sized pieces. The nearest neighbor points are found in each piece. This process begins from the starting points of the lanes which were found in Step 5. The KNN functions are used from OpenCV. There are some parameters in these functions. For example, we can set the number of neighbor points to be found. The parameters must be set optimally, meaning the project must work fast and efficiently.

Step 7 : After finding the points from KNN in Step 6, which are to be used as the input values of the curve fitting function, then the function of the curve fitting is applied. The advantage of the KNN algorithm is that it does not return so many Hough Points, which are the input values of the curve fitting, when compared to rectangle method. So here, the curve fitting function produces the mathematical equations of the lane much faster when compared to Method 1.

Step 8 : After the equations of the fitted curves for each of the lanes are calculated, these equations have to be plotted like in Method 1. Here, the x and y coordinates of the Hough Points were also swapped in order to get more stable curves. For each of the vertical pixels, the results of all of the equations were plotted in the frame.

Step 9 : The last step of this method is also to publish the coefficients of the equations of the fitted curves of each of the lanes which can be seen by the camera.

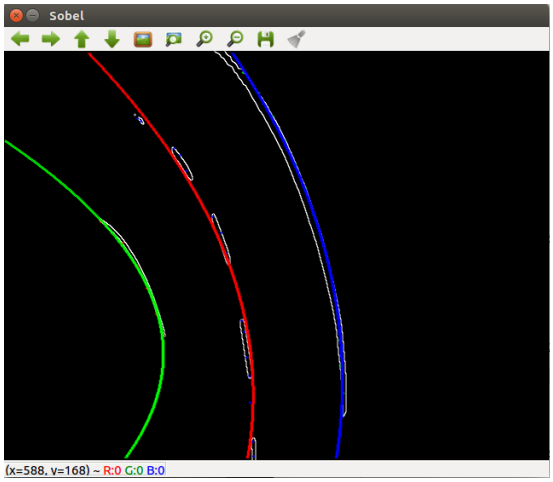


Figure 4.16.: Output Image

4.4.6 Method 2(b) : Resize + IPM + Hough Transformation + KNN + Curve Fitting

This method is very similar to Method 2, which was described in Section 4.4.5. In the Method 2, the resolution of the frame is 640x480 pixels and in this method, the resolution of the frame is 320x240 pixels. In Chapter 5, the effects of the reducing the resolution of the frame will be described. The output frame of this method is shown in Figure 4.17.

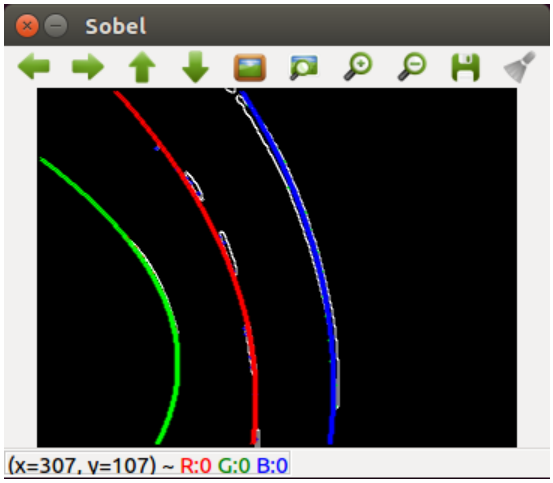


Figure 4.17.: Output Image(320x240 pixels resolution)

4.4.7 Method 3 : IPM + Hough Transformation + Rectangle Method + Curve Fitting

This method is a composite method of the Method 1 in Section 4.4.3 and Method 2 in Section 4.4.5. As in Method 2, at the beginning of this method, the preprocessing part is applied, and then Inverse Perspective Mapping for all pixels is implemented. After that, for 200x640 pixels, the Standard Hough Transformation is used and then for all pixels Probabilistic Hough Transformation is implemented. Until this point, the implementation is completely identical to

Method 2. In Method 2, however, at the point in the process where the k-nearest neighbours (KNN) method is used, the rectangle method used in Method 1 in Section 4.4.3 is applied. There is, however, a difference between the rectangle method utilized here and the rectangle method from Method 1. In Method 1, the size of rectangles decreased progressively. However, in this method, the size of the rectangles is always the same because the Inverse Perspective Method(IPM) is used at the beginning. As a result, the lanes are shown from the top of track. The size of the rectangles in the middle lane is bigger than the size of the rectangles in the left and right lanes, however. Because of the dashed lines, it was necessary to use larger rectangles in the middle lane.

After the rectangle method is used, the curve fitting method is used. As with the other methods, the x and y coordinates of Probabilistic Hough Points are swapped after the rectangle method is applied. Thanks to this swap, the curve fitting is more accurate. After producing three different equations for three different lanes, the output values for all pixel columns are calculated and plotted in the frame. In Figure 4.18, the fitted curves are shown.

The last step of this method is as in the others: to publish the equations of the fitted curves produced from the lanes. These fitted curves are published with Ros Topics.

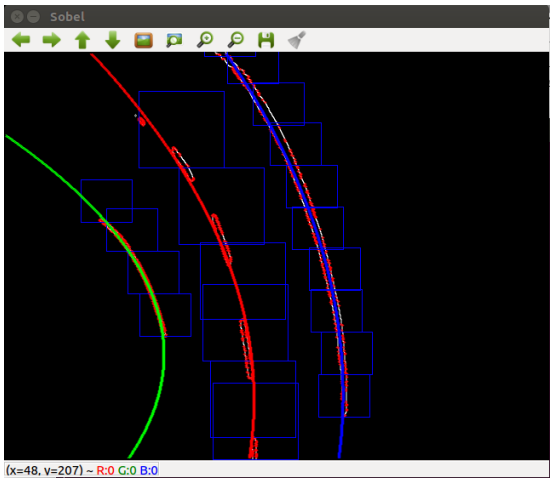


Figure 4.18.: Output Image

5 Evaluation and Discussion

In this chapter, the algorithms for each method used in this master's thesis will be evaluated. In addition, the parameters and their effects on computing time and the reliability of lane detection will be observed. The methods will be also compared with each other, with the aim of finding the most efficient and reliable method. In another section of this chapter, the average computation times for each of the algorithms as well as for the total lane detection process in each method will also be measured. At the end of this chapter, the problems which can occur during lane detection will also be defined.

5.1 Average Computing Time

In total, five different methods for lane detection were implemented. Each method has a different average computing time as well as a different level of reliability in terms of lane detection. In this chapter, all of the methods, including their respective algorithms, will be described. The computing times of said algorithms, as well as how changing the parameters of the algorithms affects their computing times, will also be discussed.

In the following table, the computing times for all algorithms, as well as their average computing times, are displayed. Due to limited space, abbreviations were used in place of the following terms:

- **PRE :** Preprocessing
- **SHT :** Standard Hough Transformation
- **FLL :** Finding the Locations of the Lanes
- **PHT :** Propabilistic Hough Transformation
- **REC :** Rectangle Algorithm
- **CF :** Curve Fitting and changed perspective of the fitted curves from the camera side to the top side
- **PUB :** Publishing the coefficients of the fitted curves
- **ACT :** Average Computing Time

Table 5.1.: Computing Times of Method 1

PRE	SHT	FLL	PHT	REC	CF	PUB	ACT
20.15 ms	0.81 ms	2 μs	23.42 ms	0.83 ms	4.54 ms	19 μs	49.77 ms

As also shown in Table 5.1, the longest computing times belong to 'Preprocessing phase' and 'Probabilistic Hough Transformation'. These values can be decreased by changing the param-

eters of Probabilistic Hough Transformation or by removing some steps in the 'Preprocessing phase'. In both cases, however, doing so can also result in undesirable effects. For example, in this project, there is a dynamic thresholding filter utilized during the 'Preprocessing phase'. A function finds the lightest pixel in the the frame and then the threshold value of this filter is changed dynamically according the value of the lightest pixel. Using such a filter results in a faster computing time, but it would also result in less reliability in changing light conditions. With regard to changing the parameters of the Probabilistic Hough Transformation, though it is possible to reduce its computing time, fewer Hough Points on the lanes will be detected. If too few Hough Points are detected, then the stability of lane detection will also suffer.

The advantage of this method is, it is the fastest method, which gets the frames with 640x480 pixels resolution, but the disadvantage of this method is, it is hard to detect the lanes, which are far away from the camera. Because the lanes, which are far away from the camera, look smaller compare to the lanes which are close to the camera.

Method 1(b) : The resolution of the frame from the camera is decreased in this method. The process in this method is nearly identical to the process in Method 1, but the resolution of the frame is four times smaller. Due to the presence of fewer pixels in the frames, the computing time is much less when compared to Method 1.

Table 5.2.: Computing Times of Method 1(b)

PRE	SHT	FLL	PHT	REC	CF	PUB	ACT
5.26 ms	0.34 ms	1 μs	11.60 ms	0.28 ms	3.49 ms	17 μs	20.99 ms

The computing time is the biggest advantage of this method, but there are also some disadvantages of decreasing the resolution of the frames. The size of the lanes is also smaller in this method when compared to Method 1, so the reliability of the lane detection is a bit worse and is also more sensitive to the changing light conditions.

This method should be used if a higher FPS value of the camera is desired. Because of the lower computing time, it is possible to raise the FPS value. This method is also useful if the processor is not powerful enough for a high computing effort.

Method 2 : In the Method 1, the Inverse Perspective Method was used only for the fitted curves. But in this method, the IPM algorithm was used at the beginning. In the following table, the computing times for each algorithm, as well as the average computing time for this method, are shown. Terms used in Method 2, which were not used in Method 1, and their respective abbreviations are as follows:

- **IPM :** Inverse Perspective Mapping Algorithm
- **KNN :** K-Nearest Neighbors Algorithm

As shown in the table above, the average computing time in Method 2 is much higher than in Method 1. One of the biggest reasons for the higher computing time is IPM. In Method 1, the

Table 5.3.: Computing Times of Method 2

IPM	PRE	SHT	FLL	PHT	FSP	KNN	CF	PUB	ACT
15.73 ms	15.85 ms	1.16 ms	5 μs	17.51 ms	11 μs	3.02 ms	3.96 ms	20 μs	57.27 ms

IPM algorithm was used only for the pixels relevant to the fitted curves, but in Method 2, the IPM algorithm was used for the entire frame. In other words, in Method 2, the IPM algorithm is used for many more pixels than in Method 1, resulting in a higher computing time. Another difference between Method 1 and Method 2 is the use of the KNN and the rectangle algorithms, respectively. The KNN algorithm used in the Method 2 is much faster than Rectangle algorithm used in the Method 1.

The main advantage of this method is that, thanks to the IPM algorithm, there is no perspective effect. In the other words, the size of the lanes does not appear smaller even if they are further away from the camera. The main disadvantage of this method is the computing time. Because of the IPM algorithm, the computing time is higher than in Method 1.

Method 2(b) : This is also a method in which the resolution of the frames is decreased. In this method, the input resolution of the frame is 320x240 pixels directly at the beginning resulting in a resolution that is four times less when compared with Method 2. As a result, the computing time is also less than in Method 2.

Table 5.4.: Computing Times of Method 2(b)

IPM	PRE	SHT	FLL	PHT	FSP	KNN	CF	PUB	ACT
4.61 ms	3.66 ms	0.21 ms	2 μs	5.43 ms	6 μs	1.95 ms	2.09 ms	17 μs	17.99 ms

The advantages and disadvantages of descreasing the frame resolution are already mentioned in the Method 1.b description.

Method 3 : This method is identical to Method 2, except for one difference. Instead of the K-Nearest Neighbors Algorithm, the Rectangle Algorithm is used. As a result, the terms and their respective abbreviations used in the following table are also identical to those used in Method 2, with the exception of REC (previously defined in Method 1), which replaces KNN:

Table 5.5.: Computing Times of Method 3

IPM	PRE	SHT	FLL	PHT	FSP	REC	CF	PUB	ACT
19.36 ms	18.57 ms	1.35 ms	5 μs	17.33 ms	17 μs	3.50 ms	5.12 ms	20 μs	65.27 ms

With this method, the KNN algorithm and Rectangle algorithm can be compared to each other, because with the exception of when the algorithms are utilized in their respective methods, the results are quite similar. The results show that the KNN algorithm is faster when compared to

the Rectangle algorithm. The main advantage of this method is again due to the IPM algorithm, but its high computing time is also the biggest disadvantage of this method.

5.2 Lane Detection Quality

High computational performance is essential for lane detection, however this parameter alone is not sufficient for the algorithm to be considered successful. Another important criterion is the quality of lane detection. Each of the proposed methods has some advantages and disadvantages from this perspective. For instance, Inverse Perspective Mapping (IPM) algorithm adds more stability to the methods where it is used. That happens because IPM algorithm shows the lane from the bird's eye perspective, so the sizes of the lanes are the same. In Method 1, however, the IPM algorithm was not used, so the lanes' geometry changes depending on the proximity to the camera. That is why when the rectangle method is applied, the size of the rectangles should be adjusted according to the y-coordinate, otherwise the Probabilistic Hough Points from the different lines could be mixed up with each other. Using IPM in Method 3 allows such adjustments to be omitted.

In addition, there are two different algorithms, which are used to group the Probabilistic Hough Points: k-Nearest Neighbors algorithm and the rectangle method. As mentioned in the previous section, KNN algorithm is faster compared to the rectangle method, however in many cases the direction of the curve must be detected to ensure that while choosing the nearest neighbors for the middle line, the Probabilistic Hough Points from the left or the right line are not taken into consideration. Rectangle method is immune to this problem, because the rectangles adjust their coordinates automatically based on the direction of the curves.

Based on these results, the best quality of the lane detection is achieved by combining IPM algorithm and the rectangle method, which is implemented in Method 3.

6 Conclusion

The aim of this master's thesis is to implement a method which can detect lanes in an indoor setting, which is necessary for the Carolo-Cup. In addition, the computing time required for lane detection should not be very high, and lane detection must remain stable in the expected light conditions.

In order to achieve this objective, many projects on lane detection were researched and their advantages and disadvantages were compared. In the literature, there are many different methods for lane detection and all of them have advantages and disadvantages, which are dependent on the application scenario as well as other factors (detecting only straight lanes or straight and curved lanes, the power of the CPU, the required FPS value, and so on). In this case, five different methods are implemented. In 3 of the 5 methods, the Inverse Perspective Mapping (IPM) algorithm is used. The IPM algorithm diminishes the view effect but also has a high computing time. Two of the 5 methods show that the lane detection algorithm can be also implemented without the IPM algorithm. In all methods, the preprocessing parts (edge detection, threshold filter, and so on) are similar. In all methods, in order to detect the pixels on the lanes, the Hough Transformation is used. At the end of the Hough Transformation implementation, the pixels which are close to each other have to be found and for this task, two different algorithms are used: the K-Nearest Neighbors algorithm and the Rectangle algorithm. After all pixels are grouped according to each lane on which they lie, the curve fitting algorithm is used for all lanes in all methods.

After measuring the computing times for all algorithms, it is observed that for the preprocessing part, the IPM algorithm and Probabilistic Hough Transformation are the most computationally expensive algorithms. In some methods, the computing times of these algorithms are decreased. In Methods 1 and 1(b), the IPM algorithm is implemented only with regard to the fitted curves, and thus only for some of the pixels in the frame, saving a lot of computing time for the IPM algorithm as a result. It is also possible to compare the computing times with regard to grouping the Hough Points after the Probabilistic Hough Transformation is used. Two different algorithms can be used to group the Hough Points: the K-Nearest Neighbors Algorithm and the Rectangle Method. The K-Nearest Neighbors Algorithm requires less computing time. The last factor with regard to computing time to be discussed is the resolution of the frames. If the resolution of the frames is decreased, the number of the pixels which have to be worked on also decreases, so the processes take less time.

In conclusion, as an outcome of this work, three algorithmic approaches to road lane detection are constructed and tested in a real environment. All of them fit into desired timing requirements and show stable and precise detection results in various light conditions, with Method 3

being the most successful in terms of both quality and performance. Proposed algorithms can be used as a base for autonomous car operation during the Carolo-Cup.

Bibliography

[1] Kraski, Gerhard: *More traffic accidents but fewer fatalities than ever before in 2016*. url = https://www.destatis.de/EN/PressServices/Press/pr/2017/07/PE17_230_46241.html, Juli 2017.

[2] Braunschweig, Technische Universität: *Carolo-Cup Regulations 2017*. url = https://wiki.ifr.ing.tu-bs.de/carolocup/system/files/Regelwerk_2017_20161016_en_0.pdf, October 2016.

[3] Allevato, Adam: *How to Calibrate a Monocular Camera*. url = http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration, April 2017.

[4] Lin, Chien-Chuan and Ming-Shi Wang: *A Vision Based Top-View Transformation Model for a Vehicle Parking Assistant*. Sensors, Department of Engineering Science, National Cheng Kung University Taiwan, No.1, University Road,Tainan City 701, Taiwan, 2012.

[5] Ramesh Jain, Rangachar Kasturi, Brian G. Schunck: *Machine Vision*. McGraw-Hill, 1995.

[6] Kim, Daniel: *Sobel Operator and Canny Edge Detector*. url = http://www.egr.msu.edu/classes/ece480/capstone/fall13/group04/docs/danapp.pdf, 2013.

[7] Team, OpenCV Dev: *Image Filtering*. url = https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=gaussianblurgaussianblur, October 2017.

[8] Team, OpenCV Dev: *Operations on Arrays*. url = https://docs.opencv.org/2.4/modules/core/doc/operations_on_arrays.htmloperations-on-arrays, October 2017.

[9] Technology Delhi, Indian Institute of: *Canny Edge Detector*. url = http://www.cse.iitd.ernet.in/pkalra/col783/canny.pdf, March 2009.

[10] Wikipedia: *Canny Edge Detector*. url = https://en.wikipedia.org/wiki/Canny_edge_detector, May 2017.

[11] Team, OpenCV Dev: *Canny Edge Detector*. url = https://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=canny, October 2017.

[12] Duda, Richard O. and Peter E. Hart: *Use of the Hough Transformation To Detect Lines and Curves in Pictures*. Stanford Research Institute, Menlo Park, California, 15, January 1972.

[13] Radovan Josth, Marketa Dubska, Adam Herout and Jiri Havel: *Real-Time Line Detection Using Accelerated High-Resolution Hough Transform*. Brno University of Technology, Faculty

of Information Technology Brno, Czech Republic.

[14] Sharma, Nabin: *Linear Hough Transform Using Python*. url = https://nabinsharma.wordpress.com/2012/12/26/linear-hough-transform-using-python/, December 2012.

[15] Team, Opencv Dev: *Hough Line Transform*. url = https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html, October 2017.

[16] Xindong Wu, Vipin Kumar, J. Ross Quinlan Joydeep Ghosh Qiang Yang Hiroshi Motoda Geoffrey J. McLachlan Angus Ng Bing Liu Philip S. Yu Zhi-Hua Zhou Michael Steinbach David J. Hand Dan Steinberg: *Top 10 algorithms in data mining*. Springer-Verlag London Limited 2007, 2007.

[17] Wikipedia: *k-nearest neighbors algorithm*. url = https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm, August 2017.

[18] Sadegh Bafandeh Imandoust, Mohammad Bolandraftar: *Application of K-Nearest Neighbor (KNN) Approach for Predicting Economic Events: Theoretical Background*. S B Imandoust et al. Int. Journal of Engineering Research and Applications, 3:605–610, October 2013.

[19] Team, OpenCV Dev: *Fast Approximate Nearest Neighbor Search*. url = https://docs.opencv.org/2.4/modules/flann/doc/flann_fast_approximate_nearest_neighbor_search.html, October 2017.

[20] *The KaleidaGraph Guide to Curve Fitting*. url = http://www.synergy.com/Tools/curvefitting.pdf.

[21] Frost, Jim: *Curve Fitting with Linear and Nonlinear Regression*. url = http://blog.minitab.com/blog/adventures-in-statistics-2/curve-fitting-with-linear-and-nonlinear-regression, August 2013.

[22] Wickert, M.: *Interpolation and Curve Fitting*. url = http://www.eas.uccs.edu/ mwickert/ece1010/lecture_notes/1010n6a.PDF, Juli 2000.

[23] *FITTING OF A POLYNOMIAL USING LEAST SQUARES METHOD*. url = https://neutrium.net/mathematics/least-squares-fitting-of-a-polynomial/.

[24] MyCurveFit: *MyCurveFit*. url = https://mycurvefit.com/, 2017.

[25] Kumar, Ammu M and Philomina Simon: *REVIEW OF LANE DETECTION AND TRACKING ALGORITHMS IN ADVANCED DRIVER ASSISTANCE SYSTEM*. Proc.IEEE Intell. Veh. Symp., 4, Juni 2008.

[26] Sepulveda, Nicolas Acero: *Fahrbahnerkennung und automatisierte Fahrbahnführung anhand einer Farbkamera mit einem Modellfahrzeug*. Bachelorsthesis, Technical University of Darmstadt, 2016.

[27] Heechul Jung, Junggon Min and Junmo Kim: *An Efficient Lane Detection Algorithm For*

Lane Departure Detection. IEEE Intelligent Vehicles Symposium (IV), June 2013.

[28] Christian Lipski, Björn Scholz, Kai Berger Christian Linz Timo Stich: *A Fast and Robust Approach to Lane Marking Detection and Lane Tracking*. IEEE Southwest Symposium on, 2008.

[29] Corporation, MiTAC Computing Technology: *About*. url = http://client.mitac.com/products-embedded-board-PD10BI.html, 2016.

[30] Dirk, Thomas: *What is ROS*. url = http://wiki.ros.org/ROS/Introduction, May 2014.

[31] team, OpenCV: *About*. url = https://opencv.org/about.html, 2017.

[32] Wiki, ROS: *cv bridge ROS Wiki*. url = wiki.ros.org/cv_bridge, December 2013.

[33] team opencv dev: *Mat The Basic Image Container*. url = http://docs.opencv.org/2.4/doc/tutorials/core/mat_the_basic_image _container/mat_the_basic_image_container.html, October 2017.

[34] Team, OpenCV Dev: *Basic Thresholding Operations*. url = https://docs.opencv.org/2.4/doc/tutorials/imgproc/threshold/threshold.html, October 2017.

A Appendix

A.1 Default Values of Parameters

Parameter	Identification	Standard Value
f_x	x-value of local length	318.503
f_y	y-value of local length	318.266
c_x	x-value of optical center	320.129
c_y	y-value of optical center	208.651
H	The height of the camera from the ground	245 mm.
θ	Tilt angle of the camera	35.8 Grad