Entwurfsmethodik
Institut für Informatik

GOETHE
UNIVERSITÄT
FRANKFURT AM MAIN

Alex Hunziker

Autonomous Steering of an Electric Bicycle Based on Sensor Fusion Using Model Predictive Control

Master Thesis

Entwurfsmethodik
Institut für Informatik

GOETHE UNIVERSITÄT
FRANKFURT AM MAIN

Master Thesis

Institute of Computer Science
Faculty of Computer Science and Mathematics

Autonomous Steering of an Electric Bicycle Based on Sensor Fusion Using Model Predictive Control

Alex Hunziker
Matriculation Number: 6766786

Examiners:
Prof. Dr. L. Hedrich
Prof. Dr. M. Minor

Supervisor:
M. Sc. Ahmad Tarraf

Date of Submission:
19. December 2019

Frankfurt am Main, December 19, 2019

Statutory Declaration

I herewith declare that I have composed the present thesis myself and without use of any other than the cited sources and aids.

Frankfurt am Main, December 19, 2019

Alex Hunziker

Eidesstattliche Versicherung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Hilfsmittel angefertigt habe.

Frankfurt am Main, December 19, 2019

Alex Hunziker

Task definition

Mobility is currently undergoing fundamental change. The rise in artificial intelligence and increased computing power makes it possible to develop autonomous vehicles. It is expected that fully autonomous cars will be commercially available in the near future. At the same time, increased awareness of environmental problems such as climate change has made environmentally friendly means of transport more attractive. In addition, the sharing economy has made using vehicles on demand possible.

In this context, this thesis aims to develop a control system for autonomous electric bicycles. By controlling the steering and brakes of the bicycle as well as the engine, the module should be able to navigate from a starting point to a destination while avoiding collisions with obstacles. The balancing of the bicycle is beyond the scope of this work and will be addressed during the construction of the vehicle.

A tangible hardware implementation with different sensors will be built to achieve the stated goal. The data obtained from the sensors and processed images are then fused together to estimate the state of the vehicle. Adaptive Model predictive control (MPC) should be used to predict the future state of the vehicle based on the current one, and to determine the appropriate control signals at a given time.

The steering and control module shall be designed to be suitable for the use on a bicycle. This means that it should have a small form factor, low power consumption, be lightweight and consist of relatively inexpensive components. Since the system needs to respond quickly to its environment, all software components need an optimal control and data path. As measurements may fail or data obtained may be ambiguous (e.g. if the lane on a camera image is not detected with sufficient certainty), the system should be robust and have fallbacks mechanisms to ensure higher reliability.

All components of the developed system/software should first be validated individually and the functionality of the controller needs to be evaluated by simulations. Thereafter, the integration and performance of the entire system should then be tested.

Contents

List of Figures

List of Tables

1 Introduction

For some years now, autonomous vehicles have been receiving extensive coverage in research, the industry and the media. Currently, various car manufacturers and technology companies are working on the development of the first commercially available fully autonomous car [72, p. 105]. Autonomous vehicles are expected to increase road safety as they are less prone to errors than human drivers. In addition, they have the potential to increase passenger comfort, create more time for other activities, reduce congestion, and some sources claim that they may be more environmentally friendly to drive [56].

Another recent development in society is the rise of the sharing economy. This trend can be observed across various industries. Among other things, there has been a significant growth in the availability and use of shared cars, bicycles and recently eScooters [29]. Instead of buying, people are increasingly renting vehicles on demand. One of the challenges of these sharing services is to place vehicles in suitable locations for potential users [38, p. 3].

A third trend is sustainability and environmental friendliness. People are increasingly aware of the importance of environmental problems such as climate change. The promotion and use of more environmentally friendly means of transport has positive effects on emissions and thus a positive environmental impact [50, p. 10].

The Goethe University is currently working on combining these three topics by developing a prototype of a bicycle sharing service where the user can order a bicycle to a specific location. One of the fleet's autonomous bicycles is then to be driven to the desired location, where the customer can conveniently start using it. The solution will thus improve the comfort of sharing bicycles and promote this environmentally friendly mode of transport.

In order to develop an autonomous bicycle, several components are required. The aim of this thesis is to develop a control and steering module for such a vehicle. The purpose of the developed unit is to autonomously navigate the vehicle from a starting point to a destination. It should control the steering, throttle and brakes of the vehicle. The control unit must meet certain criteria; it should be relatively small, consume little power and provide fallback mechanisms in case of measurement errors in order to increase reliability. The balancing of the vehicle is is beyond the scope of this thesis.

The steering and control module was constructed using a number of different sensors such as ultrasonic and GPS sensors, a lidar module and a camera for the roadside recognition. In addition, an acceleration sensor was added for future improvements of the module. The

collected sensor signals are fused together to provide a comprehensive and more reliable measurement of vehicle's state. Based on the measured and the target state, MPC is used to find appropriate control signals for steering, acceleration and braking of the vehicle.

While autonomous vehicles are a very active research topic from an academic and industrial point of view, most studies have focused exclusively on autonomous cars. Autonomous bicycles are much less the focus of research. Some papers are available which propose control systems to balance autonomous bicycles [77] [6]. A few publications used Model Predictive Control for this task [35] [57]. Other articles also consider route tracking for such vehicles [41] [40]. Kim and Yamakita proposed an MPC based bicycle robot that can track a route [42]. In addition, the controller proposed by Yavin addresses collision avoidance for autonomous bicycles as well [76]. The papers of Zhao et al. [78] and Stasinopoulos et al. [70] focus on obstacle avoidance for bicycles based on distance measurements using custom laser range and 3D lidar sensors. Lane detection is a very active research topic and many publications can also be found on roadside detection that use various techniques for different scenarios [30][55].

No papers were found that use computer vision based roadside detection for the navigation of autonomous bicycles. Moreover, none of the published papers found uses MPC with this combination of sensors as data sources to control autonomous bicycles. In addition, some of the work mentioned above is purely simulative. This thesis thus has a different focus than previous publications and the system design is novel.

1.1 Structure of the Thesis

The thesis first covers some fundamental subjects that are important for the implemented system. The chapter begins by presenting the general idea and the state of the art of autonomous driving. Then computer vision and some concepts of this domain that are relevant for this work are described. In the next sections, sensor fusion and Dijkstra's shortest path algorithm are covered. Subsequently, control systems in general and Model Predictive Control as a specific approach are discussed. The chapter then concludes with the single-track model upon which the proposed state space models for the controllers are based.

In the following chapter System Development the implemented system is presented. First, an overview of the hardware and the software architecture is provided. Afterwards, the individual sensors used and the processing of their signals is discussed. This includes a subsection describing the implemented image processing module. Then, a section describes how the route planning was implemented. In the following section, the model for simulating and predicting the vehicle dynamics as well as the actual implementation of the MPC controllers are presented. Finally, it is shown how all modules are integrated into the overall system and how the connection to the test vehicle is established.

After all components and their interaction in the overall system have been described, the next chapter documents the validation of the system. Firstly, the individual sensors are tested and the results evaluated. Secondly, the road detection module is validated with a number of different scenarios. In a third section of this chapter, the route planning module is evaluated. Subsequently, the developed controllers are tested with different simulation scenarios. Finally, the control loop is closed using a test vehicle. This makes it possible to fully test the system's ability to control the vehicle, avoid collisions and track a route.

The last chapter summarizes the results of the thesis and proposes some future work.

2 Fundamentals

This chapter presents the fundamentals relevant for the development of the system at hand. The topics are covered only to the extent necessary in order to gain a basic understanding for the following chapters.

2.1 Autonomous Driving

Autonomous Vehicle (AV)s are vehicles that are able to sense their environment and navigate in it with little or no human intervention [72, p. 105].

Autonomous vehicles have numerous advantages [24]. It is assumed that AVs are less likely to cause accidents and thus increase road safety and can save lives. A 2017 study analyzed Google's autonomous testing cars and found that these vehicles indeed had fewer accidents per million miles driven than the average humanly controlled car [73, p. 57]. This, along with a better response time of AVs compared to classic vehicles, can also reduce traffic jams. In addition, autonomous cars will also increase passenger comfort by providing free time for the former driver to do something else.

Another potential application of AVs are shared vehicles. A vehicle could be ordered to a certain place and drive there autonomously. This increases the comfort of such services considerably [24].

AVs use a variety of sensors, such as lidar, radar, GPS and acceleration sensors to analyze their surroundings [72, p. 105]. Based on the collected sensory information, sophisticated control systems identify suitable navigation paths, recognize traffic signs and avoid obstacles [19, p. 7].

A system for classifying the degree of automation into six different levels (from fully manual to fully automated systems) was developed by the Society of Automotive Engineers (SAE) [33, p. 17]. Rather than defining specific vehicle capabilities, this classification is based on the extent to which driver intervention and attention is required. These definitions refer primarily to cars, but can also be applied to other vehicles. The six automation levels are shown in Figure 2.1.

A level 0 vehicle may be able to issue warnings and intervene in certain situations, but the control is not taken over by the system. An example of a Level 0 technology would

Figure 2.1: SAE Automation Levels [46]

be the anti-lock braking system (ABS). [33, p. 21]. One level higher, at level 1, the system can take over partial control of the vehicle. It may take over steering or speed control, but not both at the same time. Examples of Level 1 systems are cruise control or parking assistance systems where the driver still controls the speed manually [33, p. 19]. In Level 2 systems, the automated system can take full control of the vehicle according to the SAE definition. However, the driver must remain alert at all times and be able to intervene and retake control. Most commercially available self driving cars currently fall into this category [7]. Level 3 systems, in contrast, do not always require the driver's attention. However, the vehicle may request intervention by the driver if necessary and the driver must then take control within a specified period of time. This restriction is lifted for Level 4 systems. In these systems, vehicles can drive completely autonomously within certain areas. The system must be able to safely abort the journey if necessary under any circumstances [33, p. 19]. Finally, Level 5 systems are able to drive completely without human intervention. Steering wheels or similar controls are optional in this type of system [33, p. 19].

Although the definitions of the levels are somewhat ambiguous and some companies use creative formulations for their marketing, the current state can be summarized as follows; There are a number of Level 2 vehicles from different manufacturers on the market. Despite some claims by the manufacturer, Tesla's current autopilot is strictly speaking only a Level 2 system [7]. Some technology and automotive companies claim to have the Level 3 "eyes-off " technology ready or almost ready. Audi's traffic jam pilot, introduced in 2018, is considered to have been the first real Level 3 system [5, p. 2]. Some companies, including the Alphabet subsidiary Waymo, are currently testing fully autonomous taxis in certain cities. These vehicles are classified as level 4. However, they are not yet available to the broad public [5, p. 3].

The technical development and testing of fully autonomous vehicles is in full swing. Among the remaining problems to be solved are e.g. rough weather conditions and unexpected behavior of other road participants [1]. Besides the technical challenges, there are also regulatory, legal and cyber security challenges that need to be mitigated until fully autonomous Level 5 vehicles can be made available [72, p. 119].

2.2 Computer Vision

Computer Vision is the branch of computer science that teaches machines to see. It is concerned with the theory, design and implementation of algorithms that can process visual data such as images or videos in order to recognize objects and gain a comprehensive understanding of the visual source [23]. The ultimate goal is, from an engineering point of view, to design autonomous systems that can perform or exceed the tasks that the human visual system can perform [31]. Computer vision deals with the acquisition, processing, analysis and understanding of digital images and the extraction of data from the real world. Understanding in this context means transforming visual information into numerical or symbolic information on the basis of which other processes can take appropriate action [43]. Recently, computer vision has become increasingly popular due to its use in combination with machine learning, especially deep learning techniques. The application of learning algorithms, such as Convolutional Neural Network (CNN)s, on computer vision data sets has led to remarkable advances in this area. [47, p. 3367].

Some of the typical tasks in Computer Vision (CV) are recognition, motion analysis, scene reconstruction, and image restoration [15]. Recognition is about determining whether an image contains an object, feature, or property. There are several recognition-based tasks such as pose estimation, character recognition, face recognition, or searching for image content. Motion analysis tasks are mainly concerned with estimating the motion (speed) of points on the image, objects in the 3D scene, or the camera. The goal of scene reconstruction is to calculate a 3D model based on visual data. Finally, image restoration tasks aim to remove noise from images by using filters or advanced approaches that usually include a step to analyze the local structures of the image before filtering.

Computer Vision has a wide range of applications. Examples for CV systems are the automatic inspection of goods in manufacturing, gesture recognition for human-computer interactions or medical image analysis. CV systems can also be used for image databases or the navigation of autonomous vehicles [71, p. 16].

Although implementations of CV systems vary greatly depending on the application, most image processing systems typically involve six tasks. Firstly, the image has to be captured by one or more sensors, which can not only be cameras, but also radar, ultrasound or other devices. Typically, the result is an image based on light intensity in one or more spectral bands. However, this is not necessarily the case and the image can instead be composed of other physical properties. Secondly, the captured image data undergoes some form of pre-processing where the data is enhanced in order to be processed later. Examples of pre-processing are noise reduction, contrast enhancement, or correction of the image coordinate system [71, p. 87]. Thirdly, characteristics such as edges or local points of interest are extracted from the image data. In a fourth step, the image is segmented and areas that are relevant for further processing are selected. The resulting smaller data set is

then processed. Objects are recognized and classified for example, or certain parameters such as the size of an object are estimated. Based on the results of the previous step, in a sixth step a decision can be made by the system [15]. What kind of decision this is depends strongly on the application.

The following subchapters describe some concepts of Computer Vision that are relevant for the developed system.

2.2.1 Color Representations

A color representation is required for the processing of digital image data. The most widespread model is the red, green, blue (RGB) representation. It is an additive color model in which each pixel is broken down into the three primary colors red, green and blue.

To form a color with RGB, three light beams (one red, one green and one blue) have to be superimposed. Each of the three beams is called a component, and can have an arbitrary intensity[69, p. 1275]. This is how most computer screens produce their colors, which is one reason for the prevalence of this model. When the three basic components are plotted in a 3D space, a color cube is obtained (see 2.2).

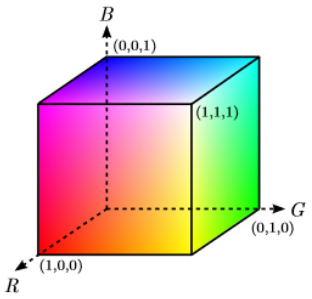

Figure 2.2: RGB Cube [3, p. 24]

Hue, saturation, lightness (HSL) is an alternative color representation. In contrast to RGB it is commonly represented as a cylinder. In this model, the colors of each hue (pure spectral colors) are arranged in a circle. While the neutral colors are located on the center axis, saturation increases as one moves away from the center. The neutral colors range from black at the bottom of the axis to white at the top. One of the main advantages of using HSL over RGB is that it is more intuitive and more similar to the way humans perceive color. A discussion of the advantages and disadvantages of this representation can be found in a paper of Chavollas [12, p. 6]. The HSL representation is often used in CV for object recognition, edge detection and other applications. The reason for this is that the relevant features are often easier to recognize in the HSL color space compared to

RGB. This is because all components in the latter are correlated with the light impacting the object and therefore also with each other [13, p. 2260]. Another reason why HSL is widely used in CV is that the transformation from and to RGB values is efficient.

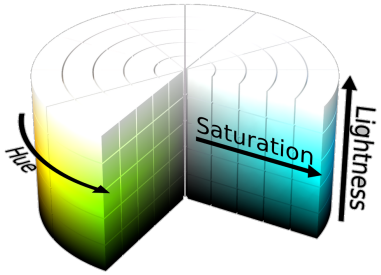

Figure 2.3: HSL Cylinder[2, p. 6]

2.2.2 Noise reduction

A fundamental challenge in computer vision, and signal processing in general, is noise. Noise can originate from various sources, both from the sensor itself and from the environment, and is difficult to avoid in practice [9, p. 4]. Denoising images is the attempt to remove the noise from the image signal and to estimate the original image. However, reducing the noise of images is associated with certain costs, including the computational effort and possible loss of details. Restoring the original image is critical for a number of computer vision applications, such as visual tracking, image segmentation and recognition [9, p. 109].

Various approaches are used to eliminate noise from images. Their effectiveness varies depending on the type of noise. Gaussian filters are one approach that can be used. They work by applying a smoothing mask that alters the image by assimilating neighboring pixels [15, p. 41]. There are, however, many more approaches for denoising available. Nowadays it is also possible to use deep learning algorithms for this task [26, p. 5].

A method that is fast and achieves good results is called non-local means[10, p. 208]. The basic idea of the algorithm is to replace the color of a pixel with the average of the colors of similar pixels. However, pixels similar to a certain pixel do not have to be in its immediate neighborhood. Therefore, a larger part of the image is searched for pixels similar to the pixel to be denoised. Contrary to the name, the approach is semi-local. Mathematically speaking, in the discrete case, this means that the resulting image u is computed at pixel p for each channel i based on the original image v as follows [10, p. 209]:

$$u_i(p) = \frac{1}{C(p)} \sum_{q \in B(p,r)} v_i(q) w(p,q), \qquad C(p) = \sum_{q \in B(p,r)} w(p,q) \tag{2.1}$$

Figure 2.4: Example for non-local means denoising [10, p. 210]

$B(p, r)$ is the neighborhood of p with radius r and $C(p)$ is a normalization factor. $w(p, q)$ is a Gaussian weight function that determines the weight of a particular original pixel in the neighborhood based on how similar it is to p [10, p. 210].

2.2.3 Perspective Transformation

The transformation of the perspective of a picture (also called homography) is a geometric image transformation. While the content of the picture is not changed by this process, the pixel grid is deformed and then mapped to the target. Essentially, perspective transformations can be imagined as rotations and translations of a camera in 3D space that is used to view the 2D image [9, p. 163].

A perspective transformation is defined by a 3x3 matrix M that satisfies the following condition for the source trapezoid coordinates (x_i, y_i) and the target coordinates (x'_i, y'_i) [45, p. 383]:

$$\begin{bmatrix} t_i x'_i \\ t_i y'_i \\ t_i \end{bmatrix} = M \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}, \qquad i = 0, 1, 2, 3 \tag{2.2}$$

The transformation can then be calculated by assigning a pixel of the source image to each pixel of the target image [45, p. 385]:

$$\mathrm{dst}(x, y) = \mathrm{src}(\frac{M_{11}x + M_{12}y + M_{13}}{M_{31}x + M_{32}y + M_{33}}, \frac{M_{21}x + M_{22}y + M_{23}}{M_{31}x + M_{32}y + M_{33}}) \tag{2.3}$$

Perspective transformations can transform trapezoidal images into other trapezoids by changing the geometry of the source image while interpolating missing pixels. It is therefore possible to select a trapezoidal Region of Interesst (ROI) within an image and transform it into another trapezoid or, as a special case, into a rectangle. If the selected ROI is considered or presumed to be a rectangle in the real world, the transformation of it into a rectangle gives a bird's eye view of it [45, p. 385]. An example is shown in Figure 2.5.

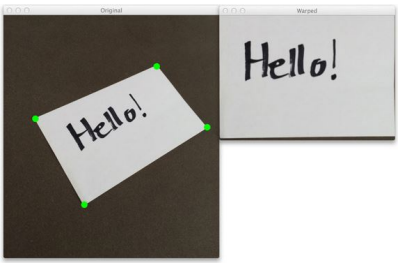

Figure 2.5: Example of a perspective transformation [64]

2.2.4 Edge Detection

Edge detection is an essential feature of many CV applications. Edge detection incorporates a variety of methods designed to identify points or lines within an image where the color or brightness values change abruptly [48]. The discontinuities found in this way are called edges. The goal is usually to divide the image into different areas. It can be shown that the such edges often correspond to discontinuities in depth, surface orientation or material properties [48]. In non-trivial images, a number of problems can occur, such as edges that do not correspond to any interesting property of the image, or fragmentation, which means that the edge curves are missing some segments which were not recognized by the algorithm.

There are many approaches to detecting edges. Most of them work by searching for maxima in edge strength and are usually based on first-order derivative expressions or on the zero points of second-order derivative expressions [79, p. 537]. In the following, the Canny edge detector (with a Sobel filter) is briefly described as it is used in the software developed.

The Canny edge detector is a multi-level algorithm to detect a wide range of edges in images. Although it was developed in the early stages of image processing, it is still widely used because it provides relatively good results and has a low computational complexity. Its biggest disadvantage, however, is its sensitivity to noise. [63, p. 577].

The first step of the Canny edge detector is thus to denoise the image. This is usually done by applying a Gaussian filter [49, p. 51].

In a second step, the intensity and direction of the gradients of the image are determined. For this purpose, two convolution Sobel masks are applied to each pixel of the image, one for the x direction and the other for the y direction [27, p. 1579]. The filters used are:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.4}$$

The gradient magnitude G and the direction Θ are calculated as follows:

$$G = \sqrt{G_x^2 + G_y^2}, \qquad \Theta = \arctan \frac{G_y}{G_x} - \frac{3\pi}{4} \tag{2.5}$$

Based on the gradient direction and magnitude, a non-maximum suppression is applied. The aim of this technique is to find the largest gradient locally and to suppress all other gradients by setting them to zero. Only the most intense edge candidates are preserved [49, p. 52].

In a final step called hysteresis, two thresholds are applied [18]. If an edge candidate pixel has a lower value than the lower threshold, it is not considered to be an edge. When the value is above the higher threshold, it is accepted as an edge. Finally, if the value is between the two thresholds, the pixel will also be considered to belong to an edge if it is connected to a pixel that is above the higher threshold.

Figure 2.6: Canny edge detector example (right) original image (left) [28, p. 205]

The result of the Canny edge detector is a binary image with the same proportions as the original image (an example is shown in Figure 2.6).

2.3 Sensor Fusion

When a system collects data from multiple sensors, the process by which this data is meaningfully combined is called sensor fusion. Essentially, the purpose of sensor fusion is to translate the sensory inputs into a more reliable and complete estimation of the system state that can be used for the intended purpose [34, p. 108].

In many cases, the sensors contained in a system use different measuring principles and have different capabilities (heterogeneous sensors). The information collected by the sensors of a system can be either complementary or redundant [34, p. 108]. Redundant signals can originate from several sensors of the same type or from different sensors. They are used to improve the accuracy of measurements or to provide backup in the event of a sensor failure or faulty measurement. When multiple sensor signals are combined into

an estimated value, often statistical methods such as the central limit theorem or Kalman filters are used [21].

Sensor fusion plays an important role in autonomous vehicles. The available sensor data about the location, the environment and the state of the vehicle (e.g. speed, steering angle, ...) have to be combined in order to make decisions about steering and speed control signals.

2.4 Dijkstra's Shortest Path First algorithm

Dijkstra's shortest path first algorithm is a greedy graph algorithm developed by Edsger W. Dijkstra. It is used to find the shortest path in a weighted graph between two nodes [14, p. 658] (without negative weighting).

For each node (V) of a graph, all possible ways to reach it from adjacent nodes and to reach other adjacent nodes from it are marked with corresponding edges (E). All nodes are placed into the set of unvisited nodes and the shortest known way to reach them is indicated. Initially, the length of the shortest path for all nodes in the set is infinite, with the exception of the initial node, which has a path length of zero. The node with the shortest path is then visited and the path from there to each neighboring node is calculated. If the calculated total path length to reach a node is shorter than the current shortest known path to reach it, then the value and path for that node are updated. The currently visited node is thereafter removed from the set of unvisited nodes. Then the node with the shortest path that has not yet been visited is visited and the above steps are repeated. Once the destination node is reached, the minimum distance to reach it and the path leading to it will are known [17, p. 270].

The worst-case performance of the algorithm is:

$$O(|E| + |V|log|V|) \tag{2.6}$$

This is asymptotically the fastest algorithm to determine the shortest path for generic directed graphs [14, p. 663].

2.5 Control Systems

A dynamic system is a system whose behavior changes over time, usually in response to external stimulation [36]. A control system, in turn, is a dynamic system that essentially manages or regulates the behavior of another dynamic system via control loops (see Section 2.5.1). Given a target value (input, also called Setpoint (SP)), the aim of a control system

is to adjust the actual value (output) accordingly [59, p. 2]. The difference between these two values is called the error.

Whilst a control system can be a SISO (Single Input, Single Output) system, most systems have multiple variables and are therefore MIMO (Multiple Input, Multiple Output) systems. Control systems come in many forms, ranging from very simple applications such as thermostats to very complex ones such as missile control systems [36, p. 36]. The following subchapters cover some of the concepts relevant for this work.

2.5.1 Open and Closed Loop Systems

Control systems can be broadly divided into open and closed loop systems [36, p. 2].

In open control loops, the control action is determined only on the basis of the SP (see figure 2.7). The actual value of the controlled variable is not measured and thus has no influence on the control signal.

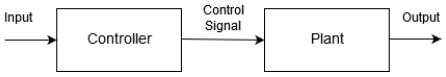

Figure 2.7: Open-loop control system (based on [32, p. 5])

While open loops are easy to design and implement, disturbances and other external influences can lead to inadequate control signals and outputs. The use of feedback, on the other hand, can lead to instabilities in a system and may cause oscillations [36, p. 3].

Closed control loops measure the actual values and take control measures to reduce the error between the desired and the actual values. A closed loop system typically consists of four components: the comparator, the controller, the plant and the sensor. Figure 2.8 shows the layout of the components in a typical control loop.

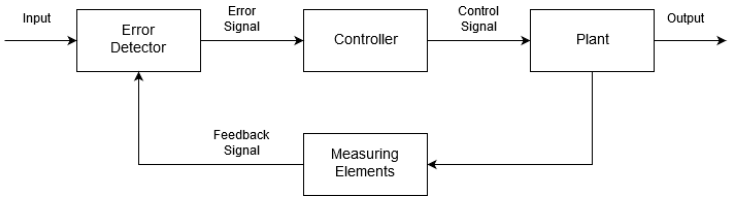

Figure 2.8: Closed-loop control system (based on [32, p. 7])

Sensors are responsible for measuring the state, respectively the actual value. The measured value is then fed into the comparator together with the desired value. The error signal results from the comparison of the reference value with the measured output. Based on the error, the controller must then decide on the system reaction, i.e. find out how the system must react in order to minimize the error. Although not necessarily the case, the

controller is usually a piece of software. The control signal (also called actuating signal) is then used to control the system [32, p. 7]. The plant is the system under control, and does not have to be a plant in the conventional sense.

The design and implementation of a closed loop system is more complex than an open one. By measuring the actual state, however, the controller can handle model inaccuracies and disturbances.

2.5.2 State-Space Representation

In order to understand the behavior of a dynamic system, a mathematical description of it is required. The state-space representation is a mathematical model for such systems that have inputs and outputs.

The state of a system consists of state variables that describe the system and evolves over time. For each of the n variables of the state, a first-order differential equation is developed to predict the subsequent state based on its current value and the current action [75]. The state vector should at any given time t_0 contain all information about the state of a system.

If the system is linear with p inputs, q outputs and n states, it can be written in the following state-space representation [75, p. 3]:

$$\dot{x}(t) = A(t)x(t) + B(t)u(t) \tag{2.7}$$

$$y(t) = C(t)x(t) + D(t)u(t) \tag{2.8}$$

Where $x(t)$ an $\mathbb{R}^n$ vector that contains the state of the system. $y(t)$ is an $\mathbb{R}^q$ output vector, which consists of the set of output variables. $u(t)$ is the set of $\mathbb{R}^p$ control signals sent to the system. A is called the state matrix, which indicates how the current state affects the change in state. B is a $n \times q$ matrix that describes how the input variables affect the state change. Finally, C and D are the $q \times n$ and $q \times p$ matrices, which describe the transformations of the state and control vector to obtain the output variables. Even if the matrices can be time variant, they are often time invariant.

For use in conjunction with MPC (see Section 2.5.3), the state-space models have to be converted from a continuous form to their discrete counterparts. There are several ways to discretize state-space models. The method used in this thesis was proposed by R. DeCarlo [16, p. 216]. It allows the calculation of the discrete representation as follows, where A_d and B_d are the discretized matrixes:

$$e^{\begin{bmatrix} A & B \\ 0 & 0 \end{bmatrix}^T} = \begin{bmatrix} A_d & B_d \\ 0 & I \end{bmatrix} \tag{2.9}$$

State-space models do not have to be linear systems. The more general form of state-space models that can also be used for nonlinear functions is:

$$\dot{x}(t) = f(t, x(t), u(t)) \tag{2.10}$$

$$y(t) = h(t, x(t), u(t)) \tag{2.11}$$

Equivalent to the linear case, the first equation describes the change of state, and the second is the output equation. Since only linear state-space models are used in this thesis, the nonlinear case is not discussed further.

2.5.3 Model Predictive Control

MPC is an advanced method for controlling a closed loop dynamic system. It is powerful due to its ability to optimize the control signals for the current time slot by taking into account the plant's predicted states in a certain number of future time slots. In addition, MPC can solve the optimization task while simultaneously satisfying a number of input, control signal, and output constraints [58, p. 139]. It is widely used across various domains.

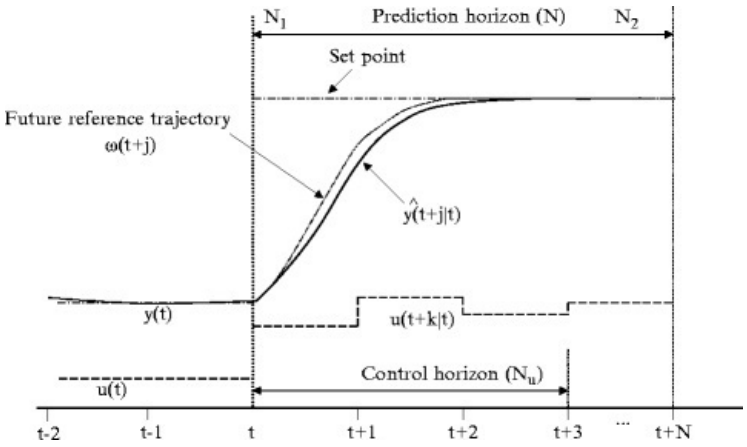

Figure 2.9: Schematic SISO MPC example [53]

The basic idea of MPC is to predict the future behavior of the system over a defined time period called the prediction horizon [53]. At any time of the control horizon, which is shorter or equal to the prediction horizon, the optimal controls and the simulated system output are determined. This is done by minimizing a cost function that measures deviations of the output and control variables from the specified setpoints under the constraints defined. The cost function is minimized for the entire prediction horizon, while the controls are only changed during the control horizon. After that, it is assumed that the controls will remain constant. A graphical example of a SISO MPC system can be found in the

Figure 2.9. The control signals computed for the first time slot are then applied to the plant and the prediction is repeated with the resulting state in the next time interval [58].

Mathematically, the following optimization problem is solved at each time step t:

$$u = arg\min_{u} J(x(t), u) \tag{2.12}$$

$$s.t. \quad J(x(t), u) = \sum_{k=t}^{N+t} ||x(k)||_Q + ||u(k)||_R \qquad x(k) \in X \quad u(k) \in U \tag{2.13}$$

In this equation, Q is the cost of the deviation of the output variables, while R represents the cost of the deviation of the control variables. X is the set of state vectors that meet the constraints for the model and U is the set of allowed controls [58].

In order to be able to predict the future state of the plant, a model of it must be available for the development of the control system. While most complex processes are non-linear, they can often be assumed to be linear over a small range. Linearized plant models can be used for MPC, since the feedback mechanism is able to compensate for the prediction errors of the linear plant model.

Besides the linear MPC there are other versions like the nonlinear MPC and the adaptive MPC. The latter is discussed in the following Subsection.

2.5.4 Adaptive Model Predictive Control

In classical MPC controllers, a linear model of the plant is used to make predictions about the future state of the system. While these predictions are almost never exact in practice, the controller can often be tuned to achieve solid performance [4, p. 6-2]. However, if the plant is strongly nonlinear or the parameters vary considerably over time, a single linear model is no longer sufficient to obtain satisfactory predictions.

Adaptive MPC addresses this problem by using multiple linear plant models to improve the predictions [4, p. 6-3]. Each of the models can be used for a subrange of the operating range of the controller. These models, all with different parameters, are either generated on the fly or can be pre-calculated offline. At each time step, the most suitable model is selected based on the current state of the system. It is then used for the entire prediction horizon of the current optimization. The underlying assumption is that the plant can be approximated by a linear system in a narrow range. If this is not the case, adaptive MPC will not be the appropriate solution.

2.6 Linear Single-Track Model

Single-track models allow to obtain meaningful results in the simulation of vehicle dynamics while being relatively simple and having few parameters. They exist in both linear and non-linear versions. Hereafter the original linear model developed by Riekert and Schurk [62] is discussed.

The linear single-track model provides an approximate description of the lateral vehicle dynamics [65, p. 243]. Certain assumptions are necessary to obtain a simple model [65, p. 243]. It is assumed that the mass of the vehicle is concentrated in the center of mass. For four-wheeled vehicles the front and rear wheels are combined in the middle of the axis. Potential tire chamber angles are not taken into account, nor are the lateral inclination or rolling movements of the vehicle. Moreover, the forces of the vehicle mass on both axis are assumed to be constant. In the models basic form the speed of the vehicle is kept constant as well.

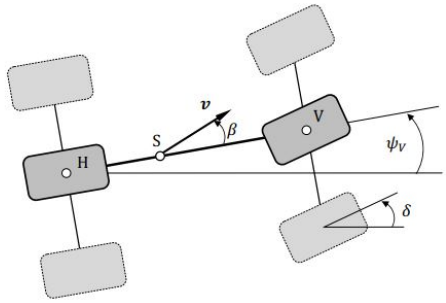

Figure 2.10: Linear single-track model [65, p. 243]

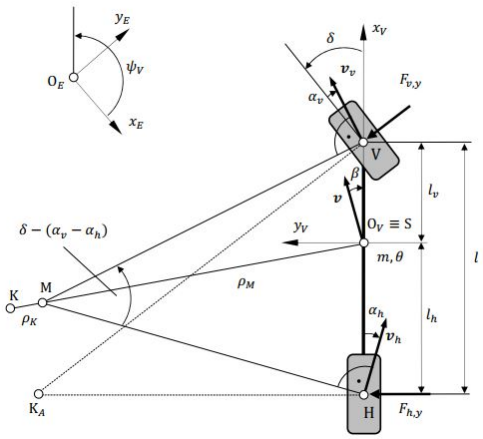

Figure 2.11: Mathematical description of the single-track model [65, p. 245]

Under these constraints above, the linear single-track model can be derived as follows [65,

p. 245]. The velocity of the vehicle in any direction within a coordinate system attached to the center of mass of the vehicle $K_v = \{S, x_v, y_v, y_v, z_v\}$ is given by the following equation, with S being the center of mass and β being the slip angle:

$$v_v = \begin{bmatrix} v\cos\beta \\ v\sin\beta \\ 0 \end{bmatrix} \tag{2.14}$$

Based on this, the acceleration in the mass center S can be described using the yaw rate $\dot{\psi}$:

$$v_a = \begin{bmatrix} -v(\dot{\psi}_v + \dot{\beta})\sin\beta \\ v(\dot{\psi}_v + \dot{\beta})\cos\beta \\ 0 \end{bmatrix} \tag{2.15}$$

The lateral force F_y is now assumed to be linearly dependent on the lateral slip of the tires α with cornering stiffness c_a.

$$F_y = c_a\alpha \tag{2.16}$$

Based on these equations, the momentum in the y-direction and the angular momentum around the vertical axis can be written based on the steering angle δ and the lateral forces for both tires:

$$mv(\dot{\psi}_v + \dot{\beta})\cos\beta = cos\delta F_{f,y} + F_{r,y} \tag{2.17}$$

$$\theta\ddot{\psi}_v = F_{f,y}\cos\delta l_f - F_{r,y}l_r \tag{2.18}$$

Now the variables for the transverse forces have to be replaced by the corresponding equations for the two tires (see 2.16), addditionally α is also replaced:

$$F_{f,y} = c_{a,f}(\delta - \beta - l_f\frac{\dot{\psi}_v}{v}) \tag{2.19}$$

$$F_{r,y} = c_{a,r}(-\beta + l_r\frac{\dot{\psi}_v}{v}) \tag{2.20}$$

After the substitution of the forces, the equations obtained previously can be rewritten, assuming that both β and θ have relatively small absolute values, to obtain the single-track equations [65, p. 249]:

$$\ddot{\psi} = -\frac{c_{a,f} l_f^2 + c_{a,r} l_r}{\theta v}\dot{\psi} - \frac{c_{a,f} l_f - c_{a,r} l_r}{\theta}\beta + \frac{c_{a,f} l_f}{\theta}\delta \tag{2.21}$$

$$\dot{\beta} = -1 - \frac{c_{a,f} l_f - c_{a,r} l_r}{m v^2}\dot{\psi} - \frac{c_{a,f} + c_{a,r}}{m v}\beta + \frac{c_{a,f}}{m v}\delta \tag{2.22}$$

The only movement capabilities considered in the model are the yaw angle ψ and the side slip angle β. The slip angle is the difference between the moving direction of the mass center and the longitudinal axis. This model is idealized, yet allows satisfactory results up to a certain lateral acceleration. For cars, this value is about $4m/s^2$ [65, p. 244].

3 System Development

3.1 Hardware Overview

Several constraints were considered when selecting the hardware components. Since the system is to be installed on a bicycle, it must be compact and lightweight in order to be mounted on such a vehicle without being disturbing. The energy consumption of the device used should also be kept low in order not to unnecessarily drain the batteries. Finally, the components should be relatively inexpensive yet powerful and capable of computing appropriate system responses within a useful period of time.

The core of the system is a Raspberry Pi[1]. It is connected to all the sensors and receives their signals. In addition, it is be responsible for all necessary calculations, including the computer vision algorithm, sensor fusion and the MPC. Finally, it is also connected to the vehicle in order to transmit the steering, acceleration and braking impulses. The Raspberry Pi was chosen due to its small form factor and energy consumption, its versatility (as it runs on standard Linux) and its computing power. Since some of the tasks, namely image processing and the model predictive control, require relatively powerful hardware, the latest Raspberry Pi 4 Model B with 2GB RAM and a Quad Core 1.5GHz CPU is used.

The following sensors are installed:

- Grove - Ultrasonic Ranger (Seeed Technology)
- Camera for Raspberry Pi 5MP (Anpro)
- NEO-6M GPS Module (u-blox)
- Lidar Sensor X4 (YDLIDAR)
- MPU-6050 Gyroscope and Accelerometer (PEMENOL)

These sensors and their integration into the system are described in chapter 3.3.

Two power banks are used. The first one delivers a maximum capacity of two amperes and powers the Raspberry Pi, which in turn supplies all sensors with power. The second battery has a maximum output of one ampere and is connected to the lidar sensor, as the power consumption of the sensor during its start-up often exceeds what the Raspberry Pi can provide.

[1] https://www.raspberrypi.org/, last accessed 12/03/2019

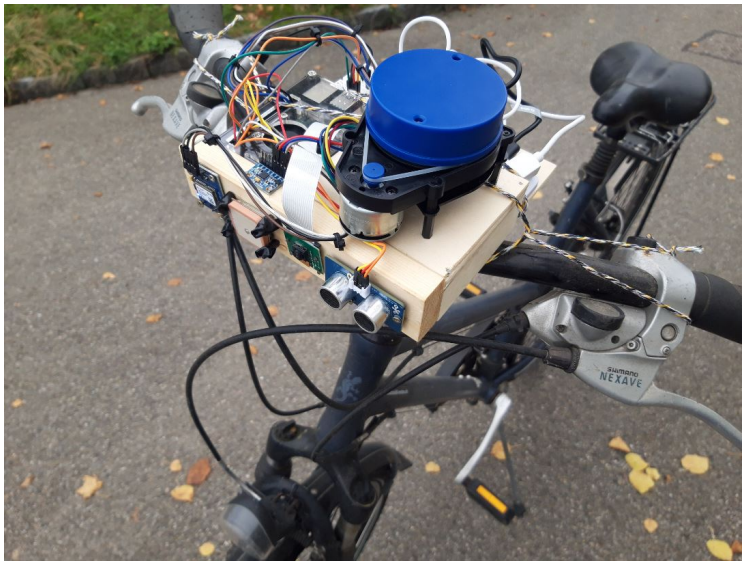

Figure 3.1: Hardware mounted on a bicycle

The system is developed for an electronic bicycle. A vehicle compatible with the system at hand is being developed by the university, but is not yet available. Therefore the system cannot be validated with the targeted vehicle. Instead, some tests concerning the sensors (see Chapter 4) are performed in which the system is mounted on a classical bicycle. Moreover, the system has been mounted on a test vehicle developed by Simon Grasemann [25] to validate the closed-loop behavior of the control system (see section 4.5). Pictures of the developed system in both configurations are shown in Figures 3.1 and 3.2.

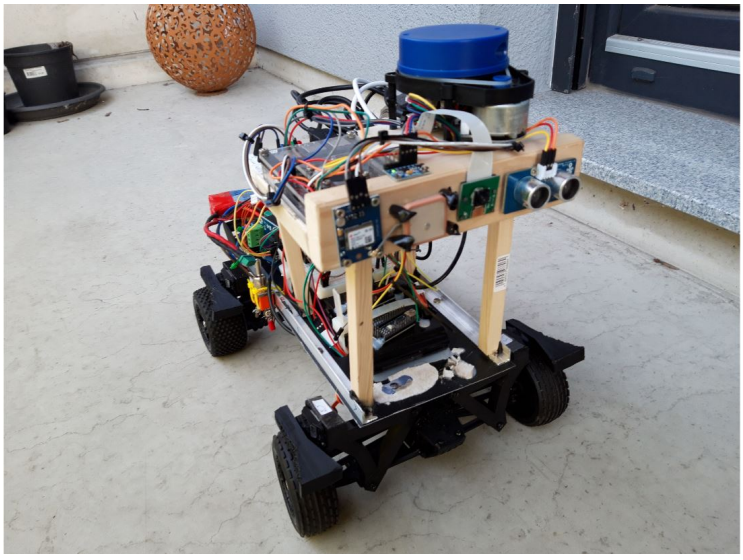

Figure 3.2: Hardware mounted on the test vehicle

The GPS module, the ultrasonic sensor, the accelerometer and the control LEDs are connected via the board level General-purpose input/output (GPIO) pins of the Raspberry Pi. GPIOs are digital pins that can be used as either input or output and whose behavior can be set at runtime [61, p. 9]. The lidar sensor is connected via Universal Serial Bus (USB) and the camera via the dedicated Camera Serial Interface (CSI) of the Raspberry Pi. The connection to the test vehicle is realized via a USB connection between the Raspberry pi and the Arduino of the test device. The latter is connected to the engine, steering and braking equipment.

3.2 Software Overview and Architecture

The implemented autonomous control software has several tasks to handle. It should read the measured values of all sensors and keep track of the valid measurements and their time stamps. Then the software has to combine these data, determine the appropriate MPC model for the prediction and use it to simulate the change of the vehicle state and ultimately to find and send the appropriate control signals.

When deciding on an architecture for a software system, one should choose an architecture that is easily extensible, reliable and maintainable [60]. The software should be modular, which means that different parts of the software should be encapsulated and abstracted from each other. This reduces complexity and increases flexibility [60]. Following this guideline, the system developed for this project is divided into several modules, with each sensor, the MPC and the route planning being separate components that are not dependent on other parts of the software. A graph summarizing the architecture can be found in Figure 3.3.

The subordinate sensor modules never take control of the entire program. Instead, the overarching coordination module is responsible for controlling all other components and requests information from them when needed. Although Python does not offer interfaces as other object-oriented languages do, the sensor modules have been developed to provide the same methods. This can be seen as a soft form of having them implement a defined interface. An abstract template according to which all sensors are written is shown below.

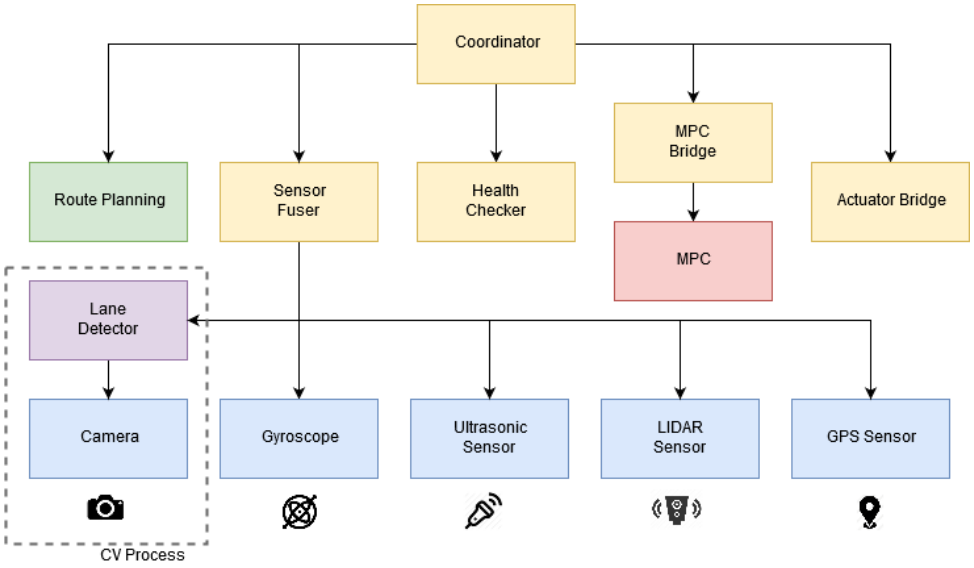

Figure 3.3: Architecture of software components

Listing 3.1: Abstract sensor class

```
class Sensor(object):
  # Sensor wide parameters/constants
  ...

  def __init__(self, parameters):
    # Initialize sensor here
    ...
    self.measured_value = None
    self.stop = False
    measure_thread = threading.Thread(target=self.measurement_loop)
    measure_thread.start()

  def measurement_loop(self):
    while not self.stop:
      # Do measurement and sanitize result
      ...
      self.measured_value = sanitized_value

  def stop_measuring(self):
    self.stop = True

  def get_data(self):
    return self.measured_value

  ...
```

The software is mainly implemented in Python and C++. The choice of languages was primarily based on the availability of suitable libraries. While the ultrasound sensor, the GPS module, the route planning and the coordinator module are written in Python, the lidar sensor module and the MPC models are implemented using C++. MATLAB, Simulink and MathWork's Model Predictive Control Toolbox[2] were used for the development of the MPC models due to the advanced development and optimization tools they provide for such controllers. The interface between the Python modules and the C++ parts was realized with ctypes[3]. The developed software consists of 53% Python (> 2000 lines), 37% C++ (> 1400 lines) and 7% MATLAB (Simulink files are not included in this count). T he remaining code consists of Bash scripts and Make files.

For the implementation it is vital that the different modules can be executed in parallel and that a delay in one module has no effect on other parts of the software. Therefore, each of the sensor modules runs in its own thread and stores its measurements individually. In addition, the camera and road detection modules are extracted into a separate process (using the Multiprocessing library) to improve parallel execution. The coordinator then calls the getter methods of the sensor classes to read the measurements and fuses them into the current system state.

The limited computing power of the hardware has to be taken into account during the implementation. Since the response time of the system has to be kept short, the algorithms used and implemented need to be fast and have adequate computational complexity. All modules have been developed with this constraint in mind, particularly the CV module.

3.3 Sensors

3.3.1 Ultrasonic Sensor

Ultrasonic sensors are a cost-effective way to measure distances in one direction. They essentially consist of two components, a transmitter and a receiver [39, p. 71]. The transmitter emits high-frequency sound waves in the ultrasonic spectrum (above 18kHz). These waves are reflected by potential nearby obstacles, and the reflected waves are then detected by the receiving sensor.

Since the waves travel at the speed of sound, the distance can be easily computed by measuring the time difference between emitting and receiving the sound signal. The following formula is used where c is assumed to be 343m/s, which is approximately the speed of sound in the air at 20° Celsius [8]:

[2]https://de.mathworks.com/products/mpc.html, last accessed 11/15/2019
[3]https://docs.python.org/3/library/ctypes.html, last accessed 11/15/2019

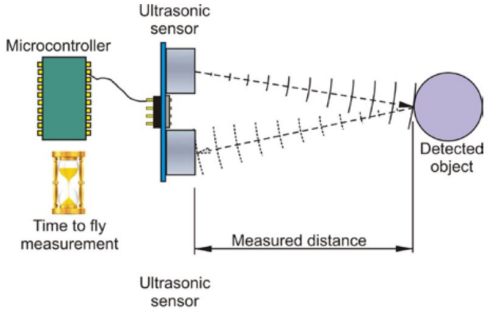

Figure 3.4: Ultrasonic sensor working principle [39, p. 71]

$$d = \frac{c * \Delta t}{2} \tag{3.1}$$

Ultrasonic sensors can detect some obstacles that cannot be detected by optical sensors (e.g. shiny surfaces). However, they are sensitive with respect to the angle of the obstacle's surface [39, p. 73]. Compared to lidar, the measurement angle is fairly large and the range rather short.

In this project, an ultrasonic sensor was used to supplement the distance measurement of the lidar sensor in order to increase the probability of detecting a nearby obstacle and avoiding collisions. The hardware used is the second generation Grove Ultrasonic Ranger manufactured by Seeed Technology. It emits ultrasonic waves at 40kHz and has a measurement range of about 2-400cm with a resolution of 1cm and a measurement angle of around 15° [67]. The sensor is connected via the GPIO pins of the Raspberry Pi. With the RPi.GPIO package, the measurements can be triggered and signals can be received directly[4].

3.3.2 Camera

In order to recognize the lanes or roadsides, computer vision techniques (see section 3.4) are used. Therefore, a camera module is needed to obtain live footage that can be processed on the fly.

For this project the "Anpro Camera for Raspberry Pi" is used. It is an HD Camera that can take pictures with a maximum resolution of 5 megapixels. The camera was specifically designed for the Raspberry Pi and can therefore be connected via the CSI port[61, p. 11] of the device.

[4]https://sourceforge.net/projects/raspberry-gpio-python/, last accessed 12/04/2019
[5]http://anpro-tek.com/anpro-kamera-fur-raspberry-pi-3-modell-b-plus-5-megapixel-kamera-mit-2-ersatz-flexkabel/, last accessed 11/18/2019

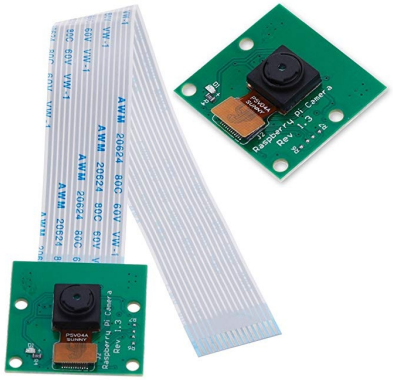

Figure 3.5: Anpro Camera for Raspberry Pi[5]

The module that was written to retrieve images from the camera uses the PiCamera package to communicate with the it[6]. In order to keep image processing fast, the resolution of the captured pictures is reduced to 600x800 pixels.

3.3.3 Satnav Sensor

An important feature for an AV is to have information about its position. For this purpose a Global Positioning System (GPS) sensor is used.

The sensor uses satellite navigation to determine its own spatial position. A network of satellites transmits time signals to the receivers, such as the built-in sensor, from which the latter can calculate the current latitude, longitude and altitude. This sensor is a passive receiver and can therefore be used independently of the availability of an internet connection. A minimum of four satellite signals are required to determine the current position. Satnav sensors can calculate their position within an error margin of a few meters [54].

The GPS module NEO-6M from u-blox is used for the developed system. The manufacturer claims that the sensor can measure the spatial position with an accuracy of $2.5m$. The accuracy of the speed and heading direction is $0.1m/s$ respectively $0.5°$ for velocities above $3.5m/s$ [74, p. 6]. The GPS Module is connected via the GPIO pins of Raspberry Pi. Universal Asynchronous Receiver/Transmitter (UART) is used for asynchronous serial communication between the two devices. In UART data is sent in a serial digital data stream with a fixed frame; A start bit is followed by five to nine data bits, an optional parity bit and a stop bit.

During the implementation of the system, the target vehicle for it was not yet available. For this reason, additional data, which will be obtained by sensors on the bicycle itself

[6]https://picamera.readthedocs.io, last accessed 12/04/2019

in the target solution, is retrieved from the GPS sensor instead. This additional data consist of the yaw rate and the speed. Although the GPS module is able to provide measurements for these variables, they are inherently inaccurate due to the limitations of satellite navigation technology. This has to be taken into account when evaluating the system.

3.3.4 Lidar Sensor

Light Detection and Ranging (Lidar) sensors, as well as ultrasonic sensors (see 3.3.1), can be used to measure distances. However, instead of using ultrasonic waves, they use optical laser light. The basic idea is the same, a transmitter emits a signal that is reflected by the obstacle and the returning signal is measured by a receiver. The distance is then calculated based on the time difference between the two events and/or the wavelength differences [11, p. 3]. The Lidar technology is used for a variety of purposes ranging from measuring the topography of landscapes and civil engineering applications to autonomous driving.

Lidar sensors can detect a wide variety of materials and provide superior resolution compared to ultrasonic sensors due to their narrow laser beam. They can be operated with either ultraviolet, visible or infrared light. One measurement only indicates the distance to a certain point in the space, but the sensor can be rotated or the signal can be deflected by a mirror system. In combination with measuring in rapid succession, 2D or even 3D distance data of the surrounding environment can be acquired [66, p. 429].

The sensor used for this project is the YDLIDAR X4[7] (see Figure 3.6). It was chosen mainly due to its competitive price. The sensor of the YDLIDAR rotates about 7 times and performs about 5000 measurements per second. This makes it possible to create a 2D map of the surrounding obstacles as shown in the Figure 3.7. The YDLIDAR X4 covers a range of up to 12 meters, which is sufficient for the expected speeds of up to $4m/s$. According to the manufacturer, the measurement error should typically be less than 1% of the actual distance [68].

Even though the sensor could be operated directly via its serial ports (UART), an adapter board is used to connect to the device via USB. Since the power consumption of the sensor motor exceeds what the Raspberry Pi can provide during the sensor's start-up, the sensor is supplied with power separately. The software that controls the Lidar sensor is written in C++ based on the YDLIDAR SDK[8]. The module is compiled into a shared object and called by the python coordinator via ctypes. The developed sensor module does not send all measurements to the coordinator, but provides a public method to obtain the distances for any given angle.

[7] http://www.ydlidar.com/product/X4, last accessed 12/04/2019
[8] https://github.com/yangfuyuan/ydlidar_sdk, last accessed 12/04/2019

Figure 3.6: YDLIDAR X4 [68]

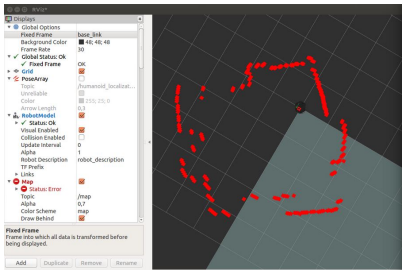

Figure 3.7: Lidar Result visualized by YDLIDAR Development Kit [68]

Due to its ability to detect other materials than the ultrasonic sensor, provide 360° information, and its longer range, the developed system uses a Lidar sensor in addition to the ultrasonic sensor. It will measure obstacles in the direction of travel and also helps to avoid steering into obstacles.

3.4 Lane and Roadside Detection

The lane and roadside detection module is based on CV and is used to determine the current curve radius of the road and the lateral position of the bicycle on it. With these two values, the MPC controller is able to follow the road. It is important to accurately capture the roadsides to calculate these values. Two types of road recognition have been implemented, the basic concept for both is to fit a second order polynomial to reflect the roads shape [37]. One mode aims to detect white lines on the road and can be used on larger roads where such markings are present. The second mode is aimed at small roads without white lanes. Instead, it detects the edge of the asphalt. Only the second mode is currently active in the system, as it is more suitable for the area in which the device is intended to be tested. The procedure is similar for both modes, so although the following description applies mainly to the second mode, it generally covers the first mode as well.

This module is implemented in Python. It uses the OpenCV[9] library, which provides over 2500 algorithms for computer vision and machine learning, is free and open source.

[9]https://opencv.org/, last accessed 11/17/2019

OpenCV itself is written in C++, but includes a Python wrapper and can therefore conveniently be called from Python scripts. In addition, numpy[10] is used for the multidimensional array representation of the digital image data and some mathematical algorithms. For improved parallel performance, the CV-related scripts run in their own dedicated process.

The camera module (see 3.3.2) serves as input device for the image data. It is configured to take pictures with a resolution of 800x600px. This size allows fast processing while the resolution is still high enough to extract the desired features. In order to increase the number of images processed per second, the class that controls the camera runs on a different thread than the rest of the image processing. This allows to capture and cache a new image while the analysis of the current one is still ongoing. Thus the two modules can work independently and the waiting time is reduced.

During pre-processing, the image is first converted from the RGB color space to the HSL color space, since the components of the latter allow better recognition of relevant features (see 2.2.1). In addition, the image is denoised using the non-local means algorithm (see 2.2.2). Since the image resolution is relatively low and in order to reduce the computation time, the size of the search window is set to merely 11px. While this reduces the noise reduction quality, the results are still satisfactory. In addition, this measure almost halves the time needed to analyze a frame.

In a third step, Canny edge detection (see 2.2.4) is applied to the light channel of the image, since the edge of the road should be clearly visible in this color component. The saturation component of the asphalt will typically be lower than that of the grass next to the road. Therefore, a binary image is generated that contains only pixels above a defined saturation threshold. The two resulting binary images are then combined additively. An example is shown in Figure 3.8.

Figure 3.8: Roadside detection on straight street without lanes

[10]https://numpy.org/, last retrieved 11/17/2019

The ROI is where the street and its edges are supposed to be. In order to calculate the desired values from the image data, the perspective is warped so that the image is seen from a bird's eye view. For this purpose, the transformation matrix is first calculated based on the region of interest, and then the image is transformed. This is achieved by using the OpenCV getPerspectiveTransform() and warpPerspective() functions respectively.

On the warped image, a sliding window algorithm is applied in order to identify the points relevant for the curve calculation. Starting at the bottom of the image, a search window is centered horizontally based on all the nonzero pixels found in the relevant half of the ROI. Within the window, all these pixels are considered when fitting the curve later on. Then the window is moved up by its height, whereas the new center of the window is the average horizontal position of the lane or edge pixels in the current window. This process is repeated until the top of the image is reached. Now a second order polynomial is fitted on all the nonzero pixels found in the search windows by using numpy's polyfit() function. A threshold for the minimum number of pixels found in the windows is applied before calculating the curve; if not enough edge or lane pixels are detected, no curve is calculated because the lane or the roadside could not be clearly identified in the previous steps. Figure 3.9 shows the lanes found in an example frame as well as the sliding windows.

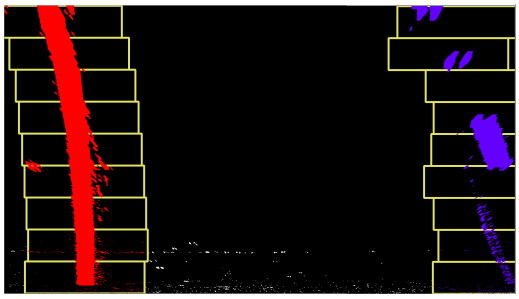

Figure 3.9: Warped image showing the sliding windows and detected lanes

Based on the fitted polynomial and knowledge about the dimensions of the ROI in the real world, the curvature of the road and the vehicle position relative to the edge or lanes is calculated. The dimensions of the ROI in the real world were determined experimentally and are assumed to be constant, which requires that the camera's height and angle remain constant relative to the ground. This assumption will not always be correct due to the potential inclination of the vehicle and thus the camera angle. It needs to be tested with the target vehicle whether the calculated results are satisfactory despite this simplification or whether adjustments are necessary. Examples for calculated lanes and roadsides can be found in Figures 3.10 and 3.11.

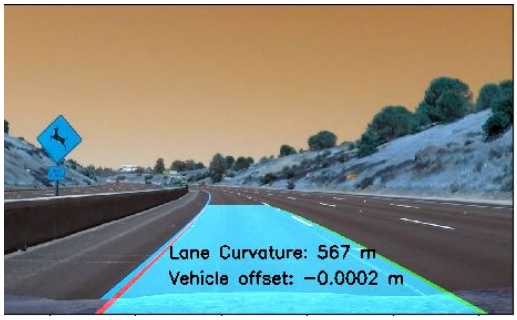

Figure 3.10: Detected lanes of a curved road

Figure 3.11: Detected edge of a straight road

3.5 Route Planning

The bicycle should be able to navigate autonomously from a starting point to a destination. Therefore, a route planning module has been developed within the scope of this thesis.

First, the route planning module loads the data about all possible paths and creates an abstract graph of it. Weights are assigned to all the edges based on their geographical length. Each node is assigned a hexadecimal identifier. Now the shortest path between a specified start and end point is calculated using the shortest path algorithm of Dijkstra (see 2.4). It is assumed that the vehicle is at the coordinates of the starting point when the journey begins. The start and destination point must be specified by the user with the corresponding hexadecimal identifiers. A user friendly interface is a possible improvement for the future.

For this project the street and trail network of the Campus Westend of the Goethe-University Frankfurt is preloaded. The data were generated by Omer-Ibrahim Erduran as a part of his master thesis [22]. Figure 3.12 shows the visualization of this path network on the map of the campus.

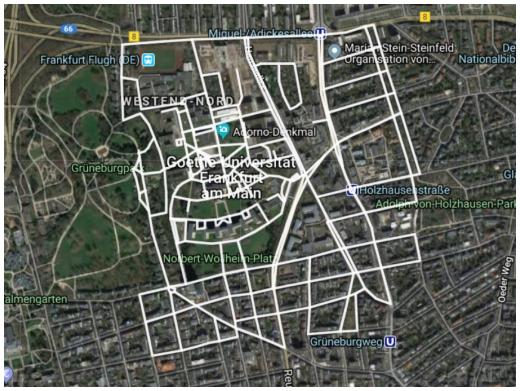

Figure 3.12: Preloaded path network graph [22, p. 47]

3.6 Model Predictive Control

The aim of the system is to control the steering and speed of an electric bicycle autonomously. This is achieved by using model predictive control. For this purpose, an adequate representation of the controlled vehicle is required. The linear single-track model, as described in Section 2.6, serves as the basis for the development of the state space models that are presented in the following. The vehicle parameters used are derived in Subsection 3.6.1.

The single-track equations have been extended with additional state variables to generate models suitable for the task at hand. Most notably, the single-track model has been extended to allow varying velocities over time. This is important because the controller is expected to accelerate and decelerate autonomously, and therefore appropriate actuating signals must be computed by the MPC controller. For this purpose, a new state variable is introduced that represents the speed of the bicycle. In addition, control variables for the throttle and brakes are added to the model. Since the engine properties of the target device are still unknown, a simple linear relationship between the throttle respectively the brake signals and the vehicle speed is assumed.

The velocity may change over time, and thus also the parameters of the single-track model equations describing the lateral dynamics of the vehicle (see Section 2.6). When the actual velocity deviates too much from the velocity parameter used in these equations, the model yields poor results because it is no longer an accurate description of the controlled system. One solution for this is the use of adaptive MPC. While the plant model is still linear, the controller takes the non-linearity of the underlying system into account by using different linearized models depending on the state of the system, as described in the Section 2.5.4. For this project, this means that when the vehicle speed changes, the plant model for predicting the future states is potentially updated with new parameters to better represent the behavior of the bicycle at the current speed.

3.6.1 Vehicle Parameters

The single-track model has some vehicle-specific parameters. Since the target hardware does not yet exist, realistic parameters need to be estimated in order to obtain meaningful simulation results for the upcoming target vehicle.

It is assumed that the mass and dimensions of the vehicle to be controlled will resemble a typical electronic bicycle. For the simulation the technical data of a typical e-bike[11] is used. The mass of the vehicle without a driver is 22kg. Assuming the center of gravity is at the saddle, the distance between it and the front wheel is 1.1m (l_f) and 0.7m (l_r) for the rear wheel respectively.

The yaw moment of inertia can be approximated with the following formula [51, p. 35]:

$$I = m\frac{a^2 + b^2}{12} \tag{3.2}$$

Here, m is the mass of the vehicle and a and b are the length and width respectively. The width of a bicycle is small and is therefore considered negligible. The length used for the calculation is 1.8m. Thus the resulting approximate yaw moment of inertia is $5.94kg \times m^2$.

The stiffness of the front and rear tire have to be approximated as well. Assuming the center of the mass is at the saddle, the vertical forces on each wheel (m_r and m_f) can be calculated as follows.

$$m_r = \frac{m * l_f}{l_f + l_r} \tag{3.3}$$

$$m_f = m - m_r \tag{3.4}$$

This results in vertical forces of 130N for the rear and 85N for the front tire. Based on the typical ratio between normalized lateral forces and slip angles for bicycles [20, p. 1375] as well as the calculated vertical forces, the stiffness of the front tire is estimated to be about $900N/rad$ and that of the rear tire about $1370N/rad$.

In order to model the velocity change over time, the longitudinal dynamics described in Mellodge's book [52] can be used. If the inclination angle of the road is assumed to be zero and the force exerted on the wheels is the rolling resistance F_r, the change in speed is approximately:

$$\dot{v} = \frac{-F_d - F_r}{m} \tag{3.5}$$

In this equation F_d can be calculated based on the drag coefficient C_d, the mass density of the air ρ and the cross-sectional area of the vehicle front A [52]:

$$F_d = \frac{1}{2}C_d\rho A v^2 \tag{3.6}$$

[11] ETROPOLIS Neo 29er 50W 36V 12Ah

Although the velocity change over time also depends nonlinearly on the current velocity, the corresponding parameter in the state space models is currently not updated. However, this is easy to change and should indeed be done once the relevant parameters of the target vehicle are known.

3.6.2 State Space Models

The vehicle should be able to navigate with the information obtained about the road by the CV module, and as a fallback with GPS data only. Since the states to be optimized differ in these two cases, two different MPC models have been developed. Moreover, the models are available in two versions each, one for the case where an obstacle is detected and one for the case where no obstacle is present. The reason for this distinction is that if there is no obstacle, the time to collision is not applicable and should not be accounted for. If, however, an obstacle is present, it should be considered with a large weight in the cost function of the MPC, since a collision is to be avoided whenever possible. Consequently, four state space models are required. The control signals for all models are the steering angle δ, the throttle a, and the brakes b. s_a and s_b are the factors that represent the intensity of acceleration and breaking effects.

The first model is used in situations where no reliable data is available from the CV module and no obstacle is nearby. In this case, the controller optimizes the direction in which the bicycle rides. Attempts to optimize for the next GPS coordinates directly led to poor results because their change with respect to the yaw angle is strongly nonlinear and therefore not suitable for a linear MPC controller. Therefore, the direction between the next target point and the current vehicle position is calculated and used as SP for the yaw angle. The resulting state space model thus yields four variables: the yaw angle ψ, yaw rate $\dot{\psi}$, side slip β, and velocity v. Its equation is as follows:

$$\begin{bmatrix} \dot{\psi} \\ \ddot{\psi} \\ \dot{\beta} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{c_r l_r^2 + c_f l_f^2}{Iv} & \frac{c_r l_r - c_f l_f}{I} & 0 \\ 0 & -1 - \frac{c_f l_f - c_r l_r}{mv^2} & -\frac{c_f + c_r}{mv} & 0 \\ 0 & 0 & 0 & \frac{-F_d - F_r}{mv} \end{bmatrix} x(t) + \begin{bmatrix} 0 & 0 & 0 \\ \frac{c_f l_f}{I} & 0 & 0 \\ \frac{-c_f}{mv} & 0 & 0 \\ 0 & s_a & s_b \end{bmatrix} y(t) \tag{3.7}$$

If there is an obstacle nearby, the model has to be adjusted slightly. The distance that should be maintained from the obstacle depends on the velocity of the vehicle. Thereby collisions can be avoided reliably, without having to maintain unnecessarily large distances to the obstacles. Therefore, instead of the distance d to the obstacle, the time to impact $\frac{d}{v} = \zeta$ is added to the state space model. It is therefore possible to determine how many seconds of safety distance should remain between the bicycle and the obstacle. The resulting state space model is:

$$\begin{bmatrix} \dot{\psi} \\ \ddot{\psi} \\ \dot{\beta} \\ \dot{v} \\ \dot{\zeta} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & -\frac{c_r l_r^2 + c_f l_f^2}{Iv} & \frac{c_r l_r - c_f l_f}{I} & 0 & 0 \\ 0 & -1 - \frac{c_f l_f - c_r l_r}{mv^2} & -\frac{c_f + c_r}{mv} & 0 & 0 \\ 0 & 0 & 0 & \frac{-F_d - F_r}{mv} & 0 \\ 0 & 0 & 0 & \frac{-1}{v} & 0 \end{bmatrix} x(t) + \begin{bmatrix} 0 & 0 & 0 \\ \frac{c_f l_f}{I} & 0 & 0 \\ \frac{-c_f}{mv} & 0 & 0 \\ 0 & s_a & s_b \\ 0 & 0 & 0 \end{bmatrix} y(t) \tag{3.8}$$

Since v will decrease during the braking phase that is be expected when an obstacle appears, the model typically underestimates ζ. In practice, this means that the computed braking impulses are stronger and the value of ζ is less likely to fall below the setpoint.

However, if data on the lanes or roadsides is available, the controller should aim to drive along the road. For this purpose, the lateral position of the vehicle on the road and the curvature of the street are calculated. These values are then converted into state variables, namely the offset from the ideal lateral position on the road O_y and the desired yaw rate $\ddot{\psi}$. The latter is calculated based on the road curvature and the current velocity. Additionally, for technical reasons, the target yaw rate $\dot{\psi}_t$ has to be included in the state as well, since its value is used to update the lateral offset during the prediction. The remaining states originate from the single-track model, analogous to the two models above. The model obtained when visual information about the road is available is:

$$\begin{bmatrix} \dot{O}_y \\ \dot{\psi} \\ \ddot{\psi} \\ \dot{\beta} \\ \ddot{\psi}_t \\ \dot{v} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -v & 0 & v & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -\frac{c_r l_r^2 + c_f l_f^2}{Iv} & \frac{c_r l_r - c_f l_f}{I} & 0 & 0 \\ 0 & 0 & -1 - \frac{c_f l_f - c_r l_r}{mv^2} & -\frac{c_f + c_r}{mv} & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{-F_d - F_r}{mv} \end{bmatrix} x(t) + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{c_f l_f}{I} & 0 & 0 & 0 \\ \frac{-c_f}{mv} & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & s_a & s_b \end{bmatrix} y(t) \tag{3.9}$$

When data about the road is available and an obstacle is detected, the state space model shown in Equation 3.9 is extended in the same way as in the above case:

$$\begin{bmatrix} \dot{O}_y \\ \dot{\psi} \\ \ddot{\psi} \\ \dot{\beta} \\ \ddot{\psi}_t \\ \dot{v} \\ \dot{\zeta} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -v & 0 & v & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{c_r l_r^2 + c_f l_f^2}{Iv} & \frac{c_r l_r - c_f l_f}{I} & 0 & 0 & 0 \\ 0 & 0 & -1 - \frac{c_f l_f - c_r l_r}{mv^2} & -\frac{c_f + c_r}{mv} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{-F_d - F_r}{mv} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{-1}{v} & 0 \end{bmatrix} x(t) + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{c_f l_f}{I} & 0 & 0 & 0 \\ \frac{-c_f}{mv} & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & s_a & s_b \\ 0 & 0 & 0 & 0 \end{bmatrix} y(t) \tag{3.10}$$

3.6.3 Weights, Setpoints and Optimizations

As described in Section 2.5.3, MPC minimizes a cost function to determine the next controls signals. Therefore costs have to be assigned to the deviations of the state and control variables from their setpoints. For each of the four controllers, the costs assigned were determined based on testing different scenarios. In general, the time to impact, if available, has been assigned the highest weight in order to avoid collisions whenever possible. If the roadside is detected by the CV module, the offset from the lateral target position is weighted with the second highest value, followed by the yaw rate and the velocity. The remaining state variables obtain a weight of zero. If the vehicle has to rely solely on the GPS data, the second highest weight is assigned to the calculated heading direction followed by the vehicle speed. The other state variables are weighted with zero.

The deviations of the manipulated variables from zero are weighted as well in all models. In the case of the steering angle, the aim is to avoid unnecessarily large steering motions, which in turn would necessitate future corrections. Non-zero weights for the brake and throttle signals, on the other hand, prevent their simultaneous use and also reduce harsh control impulses. A maximum steering angle can be set to avoid too narrow curves. In the implementation the maximum steering angle is set to $0.35\texttt{rad}$ for the simulation, and $0.2\texttt{rad}$ for the test vehicle.

While some reference values for the controllers are constant others need to be calculated. For GPS-based models, the target direction is calculated based on the current vehicle position and the next target coordinates of the determined route. The weights for the yaw rate and the side slip are zero, therefore the setpoint value is irrelevant and thus set to zero. The target speed is calculated based on a given maximum cruise speed, the distance to the next route coordinates, the destination coordinates and the difference between the current and desired heading direction. This allows the controller to stop the vehicle completely at the end of the journey and slow it down before curves. It also slows the vehicle down before intermediate route points, as they often imply an imminent change of the heading direction. The target time to impact is set to $2.5s$ when an obstacle is detected. In other words, the safety distance to obstacles is set to 2.5 times the current vehicle speed.

For the models that use image processing data, the yaw angle and the side slip references are set to zero, since their weights are also zero. The desired distance from the roadside is set to a constant value, in the current implementation this value is $0.7m$. The target yaw rate is calculated based on the assumed curvature of the road and the current speed. Finally, the target velocity and time to collision are calculated analog to the above case.

In addition to the specified weights and calculated target values, further measures are taken to improve the system behavior. Firstly, the measured direction of travel and yaw rate are ignored if the vehicle speed is zero or very low, as the measurements are not reliable in this case. Once the vehicle reaches a threshold speed that is currently set

to $0.5m/s$, these values are taken into consideration again as the measurements become more reliable with increasing velocities. Secondly, an emergency break condition has been implemented which stops the vehicle immediately, regardless of the MPC output, if an obstacle is coming too close or if the available data is not sufficient to effectively navigate the vehicle. In addition, the time to impact for detected obstacles is reduced to zero if the distance to them is less than one meter. Thirdly, the 360-degree distance measurements of the lidar sensor are used to prevent the vehicle from being steered sideways into an obstacle. Lastly, the system is configured to never allow negative velocities as reference values.

3.6.4 Development and Implementation

The MPC controllers were developed using the Model Predictive Control Toolbox distributed by MathWorks[12]. This toolbox allows to develop controllers based on a state space model by specifying the system constraints and weights. The QP solver used by MATLAB to tackle the optimization problem is described in the application documentation [4, p. 2-21]. With Simulink, different scenarios were tested and the controllers optimized. This software setup was chosen because the toolbox allows rapid development and testing and provides immediate graphical feedback. A screenshot of the MPC Toolbox and a test scenario in Simulink are shown in the Figures 3.13 and 3.14 respectively.

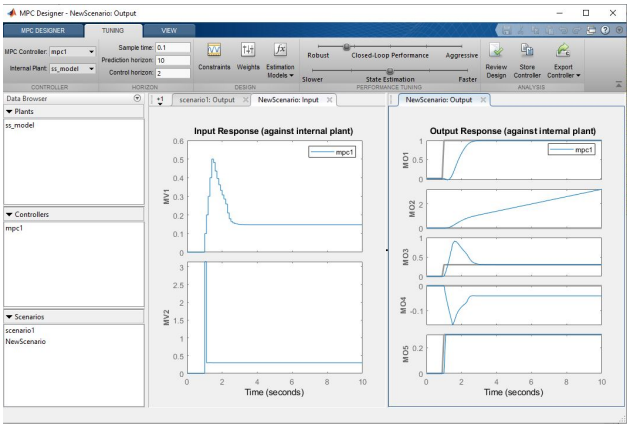

Figure 3.13: Model Predictive Control Toolbox (User Interface) [4]

For the use in the overall system, the controllers have ben reimplemented based on dlib[13] however. Dlib is a C++ multi-purpose library. In addition to machine learning, graph theory and image processing algorithms, it also contains various tools for creating control systems. The included MPC class implements the algorithm proposed by Koegel and

[12]https://de.mathworks.com/products/mpc.html, last accessed 11/19/2019
[13]http://dlib.net/, last accessed 12/07/2019

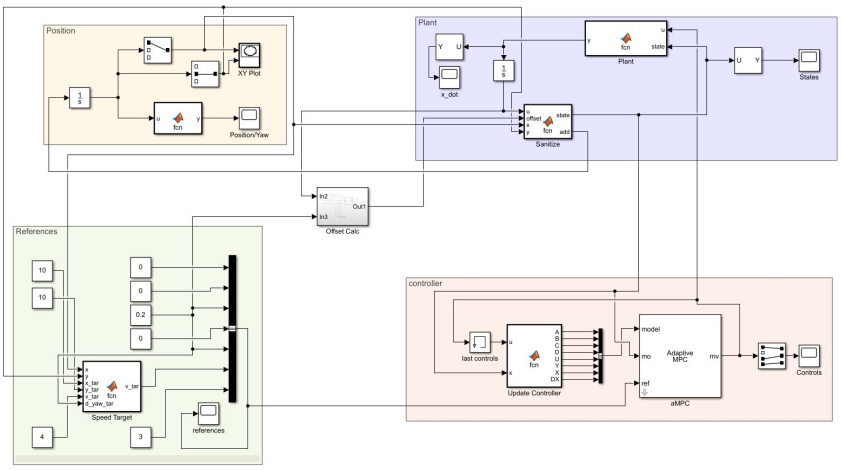

Figure 3.14: Test scenario for controller in Simulink

Findeisen [44, p. 1362]. The optimization problem of the quadratic cost function is solved by a fast gradient descent method, which also considers the constraints. Similar to MATLAB's Model Predictive Control Toolbox, dlib allows the instantiation of a controller object based on a state space model, the constraints and the weights. Dlib was chosen for the implementation because it is fast, open source and provides a straightforward Application programming interface (API). This facilitated the integration with the other software components.

However, it should be noted that dlib's MPC class has certain limitations compared to the MPC Toolbox. First, there is no option to set output constraints. As there are no mandatory output restrictions within any of the proposed models, this limitation is not an issue. Second, dlib does not natively support adaptive MPC. Nevertheless, this functionality can be emulated by creating multiple MPC objects for different parameters and then selecting the correct one at runtime.

3.7 System Integration

In order to obtain a useful system, the modules described above must work together and be interconnected. The module, which is responsible for the coordination of all other modules, is called Coordinator in this implementation. Its first task is to initialize all sensor modules, start the road detection sub-process, create all MPC objects and initialize the object responsible for the connection to the actual vehicle. The Coordinator is also responsible for starting the PathManager, which returns a list of all coordinates belonging to the planned route.

Each sensor then in turn creates its own thread in which the respective measuring loop is

started. These loops perform measurements until they are stopped by the Coordinator. Furthermore, each sensor module offers a getter method for its measured values.

The Coordinator collects the data of all sensors, the results of the road detection module, potentially updates the next route coordinate and combines all the data into a state object. This object contains all the raw data needed by the MPC controllers.

Since the data collected by the sensors are complementary and not redundant, apart from the distance measurements, they can be combined straight forwardly. However, it is necessary to check whether the measurement values are meaningful and up-to-date. This task is performed by a class called HealthChecker. Light-emitting diode (LED)s are used to indicate whether all required data is available and reasonably recent. The shorter of the two measured distances is accepted if both the ultrasonic and the lidar measurements are valid. If only one distance sensor provides a meaningful measurement, its value is accepted. If important data is missing, the vehicle is stopped and the system waits until the data situation becomes acceptable again.

Based on the data gathered, the appropriate MPC controller is selected by a class in the coordinator called MPCBridge. This class is also responsible for converting the measurements into the units required by the controllers. This means that latitude and longitude information is converted to meters, with the reference point being the next target coordinates, and that degrees are transformed to radians. In addition, this class performs most of the optimizations described in Section 3.6.3. Moreover, it is responsible for formatting the measurement data in such data structures that they can be used by the selected controller.

The Coordinator then requests the next control signals from the selected MPC object. The received steering, engine and brake control signals are subsequently sent to the vehicle. A shortened and simplified code snippet of the Coordinator class is shown below.

Listing 3.2: Shortened and simplified coordinator code

```
class Coordinator(object):
  ...

  def __init__(self, ...):
    ...
    self.path_manager = PathManager(...)
    self.mpc_bridge = MPCBridge(...)
    self.actuator_bridge = ActuatorBridge(...)
    self.active = True

  def start_trip(self, start, destination, ...):
    ...
    # Instantiate SensorFuser here.
    # It in turn will instantiate/start the individual Sensors
    self.sensor_fuser = SensorFuser(...)
```

```
16      self.path_manager.retrieve_path(start, destination)
17      coordinator_thread = threading.Thread(target=self.main_loop)
18      coordinator_thread.start()
19
20    def main_loop(self):
21      while self.active:
22        ...
23        state = self.sensor_fuser.retrieve_updates()
24        ...
25        state.next_target = self.path_manager.get_next()
26        self.health_checker.check(state)
27        impulses = self.mpc_bridge.request_step(state)
28        self.actuator_bridge.send(impulses)
29        ...
30      self.sensor_fuser.stop()
31
32    ...
33
34  if __name__ == "__main__":
35    ...
36    coordinator = Coordinator(...)
37    coordinator.start_trip(start, destination, ...)
38    ...
```

As mentioned in Section 3.1, a test vehicle is used for the validation because the target hardware does not yet exist. The steering, the brakes and the engine are operated by an Arduino Nano that is part of the test vehicle and to which the control unit is connected via USB. The developed steering and control unit sends UTF-8 encoded strings containing the actuating signals to the Arduino. These strings are always six bytes long; A start character, followed by two bytes for the steering angle, one byte each for the throttle and brakes, and one termination character. The neutral position for the steering angle is 50 and 0 for the throttle and brakes respectively. PySerial[14] is used to transmit the signals via the serial USB connection.

The software interface of the Arduino developed by Grasemann [25, p. 15] was modified in order to be able to receive all three pieces of information at once. This significantly improves the vehicle's responsiveness to control signals.

[14] https://github.com/pyserial/pyserial, last accessed 11/18/19

4 Validation

4.1 Sensor Testing

The first step of the validation is to check the correctness and accuracy of the measurements of the individual sensors. The following subsections describe the results obtained.

4.1.1 Ultrasonic Sensor

The ultrasonic sensor is used for distance measurements. Based on the manufacturer's specifications, it is expected that distances in the range of 2cm to approximately 4m can be measured.

First, the sensor was tested under good conditions in a quiet environment, at an angle of 90° to a hard obstacle with a smooth surface. The results can be found in the Table 4.1. It can be concluded that the measurements are reliable up to about $4 - 4.5m$ and that the standard deviation is less than 1cm in all cases. This is very accurate for the purposes intended.

Distance (m)	Standard Deviation (cm)	Invalid Measurements
1.01	0.1	0%
2.01	0.1	0%
2.98	0.2	0%
4.03	0.3	0%
4.57	0.4	8%
~ 5.0	n/a	100%

Table 4.1: Ultrasonic sensor accuracy

In order to test the real world capabilities, different materials were tested from different distances. The materials examined were wood, concrete, a car (i.e. metal and glass) and bushes. All these obstacles were measured at different distances of one to four meters. The raw measurement data can be found on the mass storage medium attached to this thesis.

In these realistic scenarios, the range is almost always less than four meters. Obstacles close to that distance are only detected under ideal conditions. The ultrasonic waves were,

as expected, much better reflected by hard surfaces than by soft ones. While the soft leaves of shrubs were only detected reliably from a distance of $1.5m$ (detection rate 89%), a hard concrete wall was still recognized at a distance of more than 4 meters in 85% of the cases. The range for the detection of cars and wood is between 2 and 3 meters with success rates of 95% and 3% for the car and 69% and 29% for the wooden surface. Wherever a distance was measured, the accuracy was always good.

The sensor module currently performs about 8.5 measurements per second, which is sufficient for the purposes of this project. This rate could easily be increased as there are currently scheduled waiting times between measurements. Given the maximum speed of the vehicle and the relatively short range of the device, it can be concluded that an ultrasonic sensor alone is not sufficient to detect obstacles in front of the vehicle.

4.1.2 Lidar

In addition to the ultrasonic sensor, a rotating lidar is used for measuring distances. The device should be able to detect distances of up to 12m and provide up to 5000 measurements per second. Since only the points in front of the vehicle are of interest, the other half of the measurements is discarded, resulting in a maximum of 2500 measurements per second.

Analogous to the ultrasonic sensor, the lidar was first tested under good conditions to measure its accuracy. For this sensor, this means in an environment without sunlight. The Table 4.2 shows the results of these measurements. While the deviations are relatively small and the failure rates low, the sensor seems to overestimate the distances proportional to the actual distance values. This could either be due to slightly inaccurate measurement directions or due to general sensor inaccuracies. However, the sensor precision is still sufficient for the purposes of the system. The sensor rotates at approximately 8.5Hz, which means that 8 measurements per second can be performed in any direction. In the tests carried out, approximately 240 measurements per rotation were made, resulting in slightly more than 2000 measurements per second.

The capability of the sensor also depends on the material by which the laser beam is reflected. It was possible to reliably measure the distances to a black car up to 4m, a hedge up to 7m, a glass surface up to 4m and a brick wall up to 14m during the test. However, the lighting conditions have the greatest influence on the sensor performance. When the sensor is used in direct sunlight, its range decreases drastically, sometimes to less than 2 meters. A lower sensor performance is to be expected in sunlight, as also mentioned in the user manual [68, p. 14]. Even though all lidar sensors are sensitive to ambient light to some extent, a more powerful signal transmitter may mitigate this problem.

Distance (m)			
Actual	Measured	Standard Deviation (cm)	Invalid Measurements
3.0	3.05	0.2	0%
5.0	5.08	0.4	0%
8.0	8.21	1.1	0%
10.0	10.19	4.5	0%
12.0	12.25	7.0	2.4%
14.0	14.35	8.1	16.8%

Table 4.2: Lidar sensor accuracy

When the lidar sensor is not used in direct sunlight, it can perform distance measurements with a longer range than the ultrasonic sensor. Due to its 360-degree measurement capability, it is useful for avoiding steering into obstacles. By combining the two distance sensors, a more reliable distance estimation can be obtained.

4.1.3 GPS sensor

In order to evaluate the accuracy of the GPS sensor, three scenarios were tested. Firstly, stationary position measurements were performed to verify the accuracy of the calculated position. Secondly, the sensor was moved at a constant speed along a straight line to measure yaw angle, yaw rate, and speed deviations. Finally, the sensor was moved at constant speed in a circle to evaluate the accuracy of the yaw rate. The tests were performed outdoors as this is necessary for good GPS reception.

For the first test, the device was placed in various stationary positions. The measured standard deviations of the coordinates were 0.5m for the latitude and 1.0m for the longitude. This is within the range of the advertised accuracy. The velocity had a mean deviation from the true value (0m/s) of 0.15m/s, while the yaw rate yielded, as expected, arbitrary values.

The second test was conducted while moving at constant speeds along a straight line. It was performed at three different velocities, the results are shown in Table 4.3. The obtained coordinates of one of these test runs are shown in Figure 4.1. The velocity, heading direction and yaw rate are all calculated based on the position. Since there are already inaccuracies in the latter, it is not surprising that all three derived values contain a significant proportion of noise. This may prove to be a major issue, especially when following a lane. In this case, accurate yaw rate measurements are vital. The accuracy of the heading direction increases with speed, as expected, since the differences between the coordinates increase and the signal-to-noise ratio thus improves.

Measured	Standard Deviation		
Velocity (m/s)	Velocity (m/s)	Heading Direction	Yaw Rate
1.2	0.26	6.4°[1]	n/a[2]
2.1	0.27	6.4°	4.1°
3.7	0.39	2.8°	4.7°

Table 4.3: GPS measurements at constant velocities

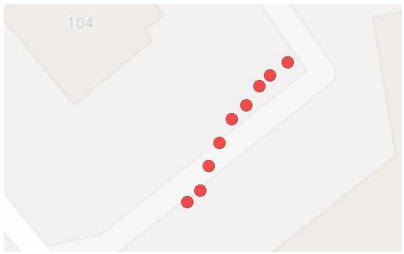

Figure 4.1: Accuracy of obtained GPS coordinates [3]

The sensor returns about three measurements per second. However, the actual number of measurements seems to be smaller, since the same values are typically received several times in a row. The effective temporal resolution is thus low. This is suboptimal considering how dependent the system is on this data.

It can be stated that the unit provides measurements with an adequate accuracy given the limitations of the underlying technology. However, especially the yaw angle and the yaw rate have a rather low precision with regard to the system task. Therefore, an accelerometer has been built into the system which can be used in the future to obtain better estimates of the yaw rate. At low speeds, the heading direction and yaw angle measurements mostly consist of noise. Consequently, these values are ignored by the controller at low velocities. The speed measurement could be significantly improved by calculating this value from the wheel revolutions of the vehicle. This feature should be available in the target vehicle.

4.2 Roadside Detection

In order to test the road detection module, the hardware was attached to a bicycle in order to ensure a fixed camera angle at a constant height. Since the target vehicle is not yet available, the module was tested with a regular bicycle. In the current setup, the module

[1] For the valid 46% of measurements
[2] Results obtained contain mostly noise
[3] Map data: https://www.openstreetmap.org/search?query=#map=18/50.14312/8.69939, last accessed 12/11/2019

is capable of capturing and analyzing about three frames per second. Various test images have been taken, which will be analyzed in the following. As only the roadside detection is relevant for the current implementation, only this mode will be discussed below. Since each frame is analyzed separately, the pictures are also examined individually in the following tests. When conducting a test with a continuous stream of frames, the CPU utilization was constantly at about 220% and the memory consumption was about 140MB (roughly 8% of the total RAM of the Pi).

The roadside detection mode should be able to identify the edge of a road and calculate its curvature as well as the offset of the vehicle from its ideal lateral position on the road (see section 3.4).

4.2.1 Straight Roads

At first, it is examined whether the module is able to detect edges of straight roads. Figure 4.2 contains three example images, all of which show straight roads with curbstones, followed by grass. The vehicle is placed at an appropriate distance to the roadside. The ideal result would be an infinitely large curve radius (a perfectly straight edge) and a small lateral offset.

Figure 4.2: Straight road detection

As can be seen, all three roads are recognized correctly, the calculated radii are relatively large and the offsets are small. Deviations in the offsets are probably due to imperfect placements of the vehicle while taking the test images, while the different radii are due to inaccuracies in the edge detection. However, it can be stated that in all three test images the results are satisfactory. The roadsides are detected even though the grass in

the first picture does not reach the road and there are road damages in the second picture. Furthermore, the first picture contains shadows and a bicycle close to the roadside. These factors have not influenced the algorithm performance.

4.2.2 Curved Roads

In addition to straight roads, roads with curves in both directions should be recognized. Since the algorithm used utilizes a second order polynomial to approximate the road curvature, this should be possible.

Figure 4.3 shows four such roads. Two with a left curve and two with a right curve. In addition, each curve is also shown on a satellite image. The cyan circle represents the vehicle's position, while the magenta circle represents the calculated curvature. Ideally, the detected curve would follow the lane on the camera image exactly, and the calculated curvature would match the actual road curvature in front of the vehicle.

It can be seen that the curves are detected in all four cases. Looking at the resulting curves drawn on the original pictures, it can be seen that the curve directions and positions are mostly correct, except for the third image where the curvature seems to have been overestimated. However, when looking at the satellite images and the superimposed circles, it can be observed that the calculated curve radius is actually too large in the third case. For the other three examples, the curve radius obtained seems to be reasonable. The passing car on the last picture does not affect the algorithm negatively.

In the filtered binary images (middle column) it can be seen that in the second and third pictures the curbstones were partly recognized as part of the road and partly not. This is not ideal, since such partial recognitions may have a considerable impact on the curve calculation.

4.2.3 Lateral Position

In addition to the road curvature, the lateral position of the bicycle on the road is also of interest. The software should be able to detect whether the vehicle has more or less distance to the roadside as it should. To validate this, the vehicle was placed three times at the same position, only once shifted to the left and once to the right. The results are shown in Figure 4.4.

The software is, in this test case, able to determine whether the vehicle is too far to the left (positive offset values) or too far to the right (negative offset values). The lane was recognized as more or less straight in all three images. It should be noted that the curbstones are again partially recognized as part of the street and that the lower left corner

[4]Source satellite images: https://www.google.com/maps, last accessed 12/11/2019

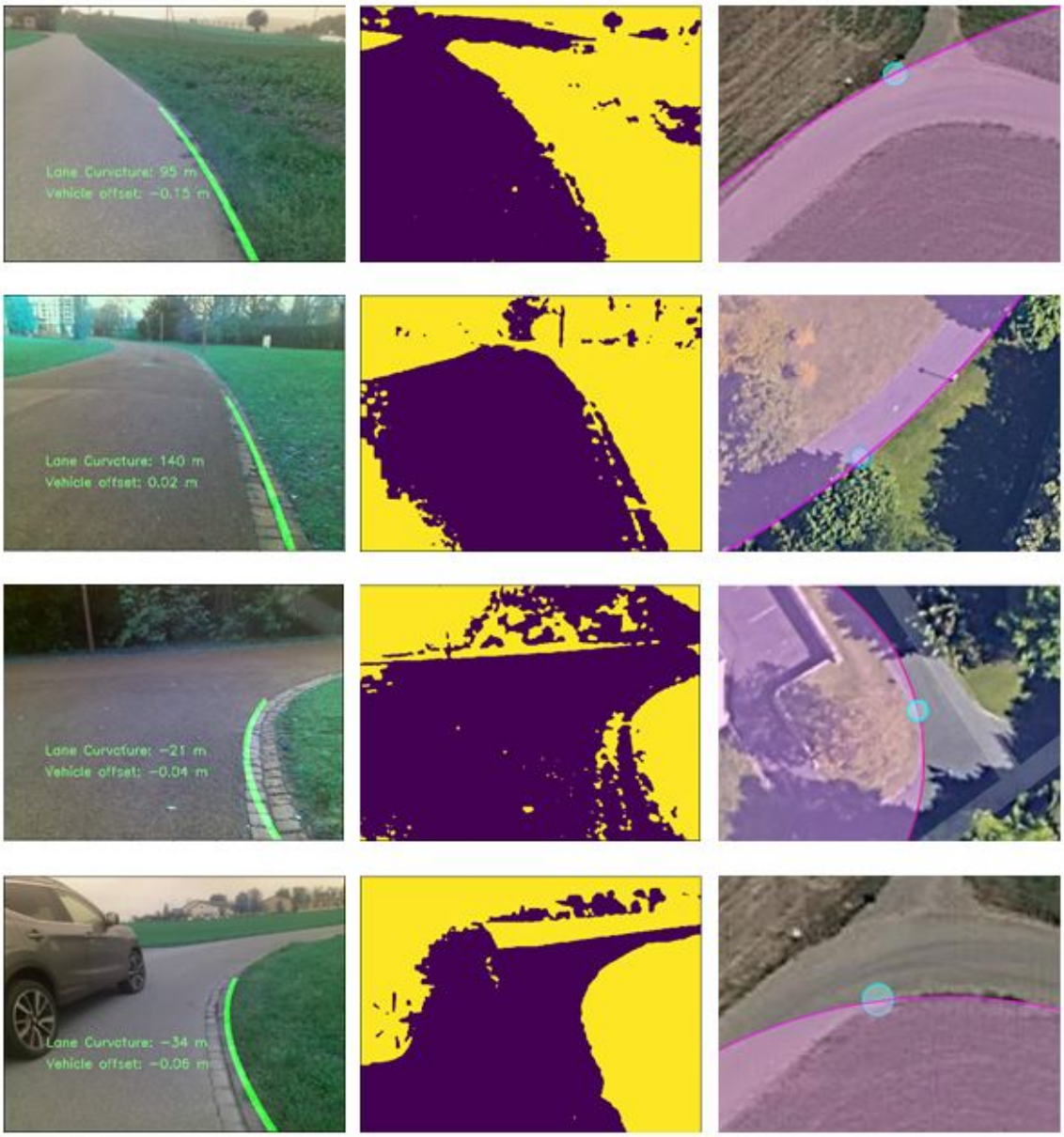

Figure 4.3: Right and left curve detection[4]

on the first image is not recognized as part it. Since the corner in question is not taken into account for the edge detection, the result is still acceptable, though. In general, it can be concluded that the lateral offset of the vehicle can be determined by the current approach. Larger offsets would, however, not be detectable due to the relatively narrow viewing angle of the camera.

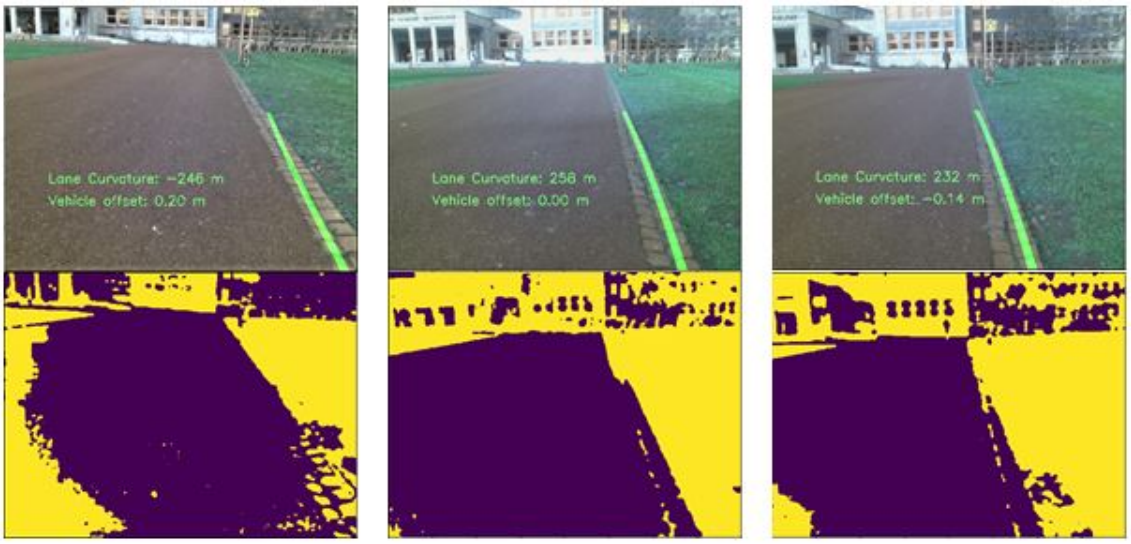

Figure 4.4: Offset to specified lateral position

4.2.4 Camera Rotation

The above scenarios were based on the assumption that the camera has no lateral inclination. This is also a constraint under which the software was developed in general. In reality, however, a bicycle may lean sideways while driving, and therefore it is interesting to see how the module behaves under these circumstances. For this test a picture of a straight road was digitally rotated by 10° to the left (right picture) and to the right (left picture). By using the same footage as basis for all three images, the effect of the camera rotation can be examined isolated. The results are shown in Figure 4.5.

On the basis of the results, three observations can be made. First, the roadside is accurately detected in all three cases. Second, the reported curvature of the road is largely unaffected by the image transformation, although a slight curve is computed in the left image. Third, the calculated lateral offset changes due to the rotation, but not to the same extent in both directions. If the camera is tilted, this means that the whole vehicle is inclined as well and thus the wheels also have a different lateral position. The direction of the expected wheel position change and the direction of the offset change are the same. Whether or not they match in a meaningful way has yet to be tested and confirmed once the target hardware is available.

4.2.5 Rejection

It is not only necessary that the module is able to identify roadsides if they are present, but it is also important that it does not report garbage values when no roadside is detected. Figure 4.6 shows three scenarios where no data should be returned.

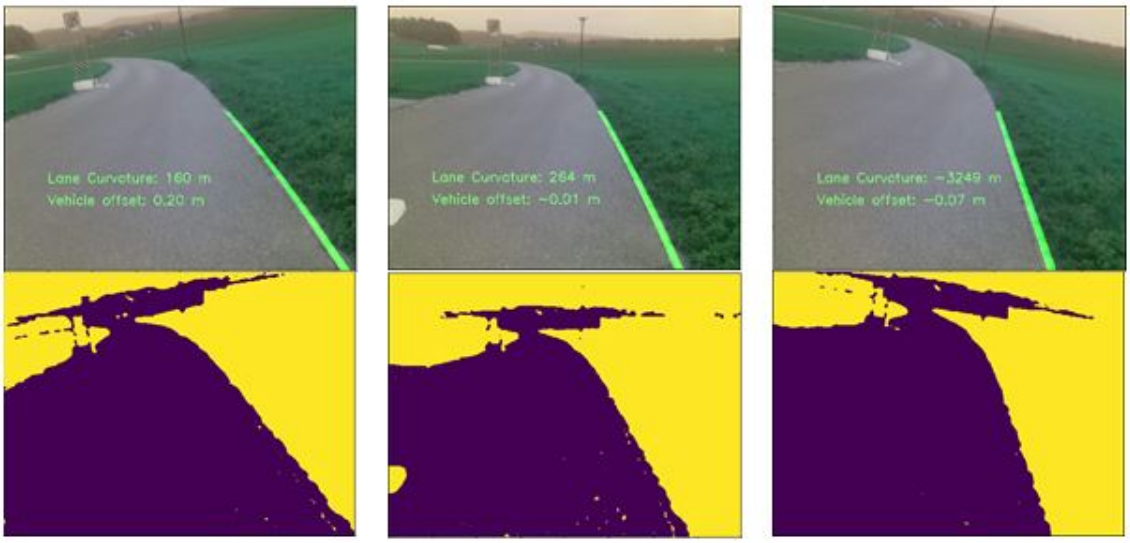

Figure 4.5: Rotated camera footage

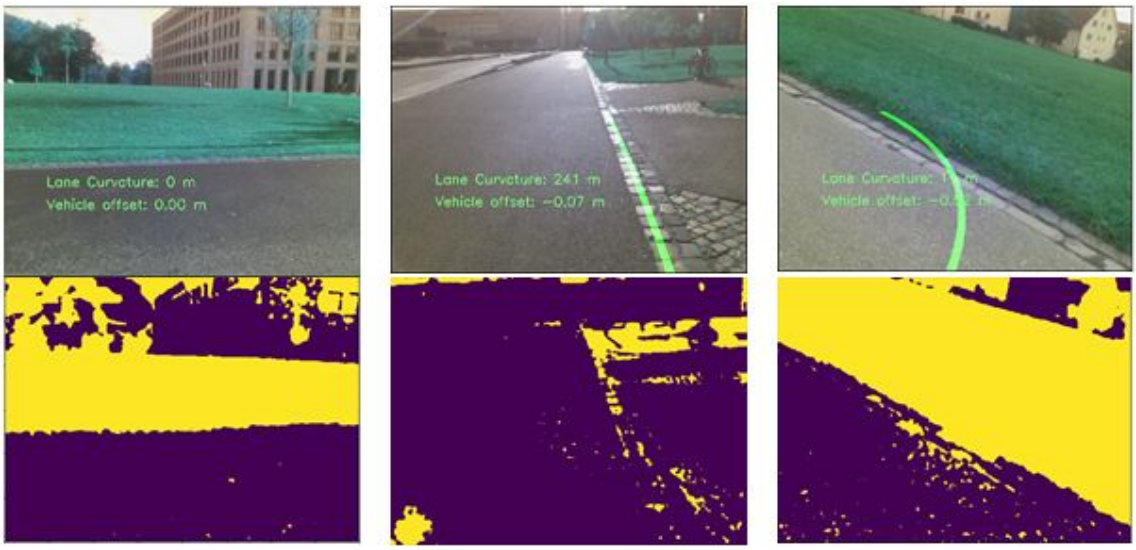

Figure 4.6: Roadside detection, rejection test

The first test picture is indeed rejected by the algorithm and no curve is returned. This is because no significant edge is found in the first two search windows (as described in 3.4).

The second picture shows a junction, with curbstones running through. While the recognition of curbstones as part of the road can be useful for the other use cases mentioned above, here this should not be the case. Otherwise the vehicle is not permitted to turn right onto the other road (since following the lane has higher priority than navigating to a GPS signal) even if the calculated route would require this.

Finally, the third image should not be used to determine the roadside, since the actual edge begins outside of the ROI in the lower right corner. Therefore, the curve calculation cannot be carried out in a meaningful way. However, since in this picture the saturation values for some parts of the asphalt are quite large, the first two search windows are not discarded and a curvature is calculated. Although the resulting curvature would coincidentally guide the vehicle along a reasonable path, this result is undesirable because the computer vision module is supposed to detect roadsides, which it did not do here.

In summary, the module is able to avoid returning some invalid roadsides, but the rejection of false positives is not as robust as it should be.

4.2.6 Robustness and Limitations

The above sections show that the implemented road detection module is capable of identifying and processing straight and curved roadsides, calculating the lateral position of the vehicle on the road, and is partially robust against lateral inclination of the camera. However, there are a number of limitations to consider.

The edge detection of this module is only written and tested for asphalted roads surrounded by grass. While it may work with materials other than grass, this has not been tested. Furthermore, the performance of the edge detection may be affected by weather conditions (e.g. rain), seasons (e.g. snow, fallen leaves on the road) and lighting conditions. As shown in Subsection 4.2.5, invalid frames may be rejected under certain circumstances, but performance is suboptimal in this respect.

The curve calculation requires a well recognized edge in order to work properly. If the edge of the road is not clearly recognized, the resulting curve may be inaccurate. In addition, there the an assumption behind the algorithm that the vehicle already roughly follows the street. If this is not the case, the module is not capable of working properly.

Lastly, the camera used has a rather small viewing angle. A camera with a wider viewing angle would be better suited for this purpose as it could cope with larger lateral offsets.

Despite these limitations, the module can detect straight and curved roads and fairly accurately determine the lateral position of the vehicle. It makes sense to use this information, when available, for navigation, rather than relying solely on GPS signals. Once the target vehicle is available, it has to be tested whether the data provided by this module is accurate and frequent enough to navigate the vehicle properly on this basis.

4.3 Route Planning

The aim of the route planning module is first to find a route between the start and destination and second to take the shortest path possible. The module was tested by

randomly selecting twenty start and destination points in the graph and calculating the route between them. The results were then evaluated to see if they satisfy the above criteria.

When assessing the results, it was assumed that all paths and roads can be used in either direction, as this information is not available in the source data. While most routes have been calculated as expected, some seem to contain smaller detours. Two root causes have been identified for these. First, some paths and connections are missing in the graph. Second, some edges have start and end coordinates that are slightly inaccurate, which prevents the algorithm from recognizing them as being part of the intersection they belong to. On one hand, these problems could be solved by improving the raw data that makes up the graph. On the other hand, it would also be possible to treat the nodes of the graph not as single point coordinates, but as circles with a certain radius. This could solve the second issue as well. In all test cases, however, a valid route from the start to the destination was found. Figure 4.7 shows the waypoints of one of the twenty test cases on a map.

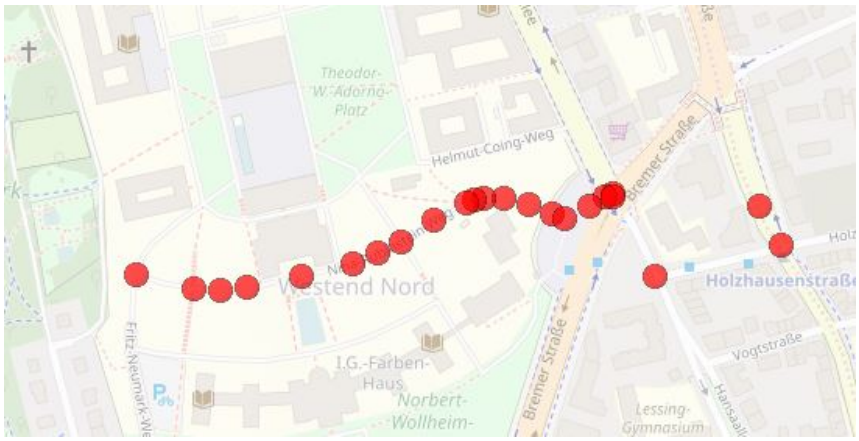

Figure 4.7: Example route plotted on a map[5]

Since the only issues found are caused by inaccuracies in the sample data used, it can be presumed that the route planning module works as intended.

4.4 MPC Simulations

The controllers created were tested on the basis of various scenarios. In the following, a representative example is presented for each base scenario to show the general capabilities

[5]Map data: https://www.openstreetmap.org/search?query=#map=17/50.12666/8.66915, last accessed 12/11/2019

of the controllers. The first scenario tests whether the controller is able to reach predetermined coordinates and stop the vehicle there. The second scenario then tests whether the controller can correct a lateral offset and follow a defined curvature afterwards (i.e. maintain a constant yaw rate if the speed is constant). Thirdly, it is checked whether the control system can avoid colliding with an obstacle by braking. Finally, a more comprehensive test aims to show that the controller is able to track a complete route and provide the necessary steering, throttle and brake signals.

4.4.1 Navigate to Position

The vehicle is placed at the coordinates (0, 0) with an initial speed of $2m/s$ and a heading angle of 0°. Then the destination coordinates are set to (-10, -10). It is expected that the control system will take a sensible path to the specified destination and bring the vehicle to a complete stop on arrival. Furthermore, the control signals should not oscillate too much and the brakes and throttle should never be applied simultaneously.

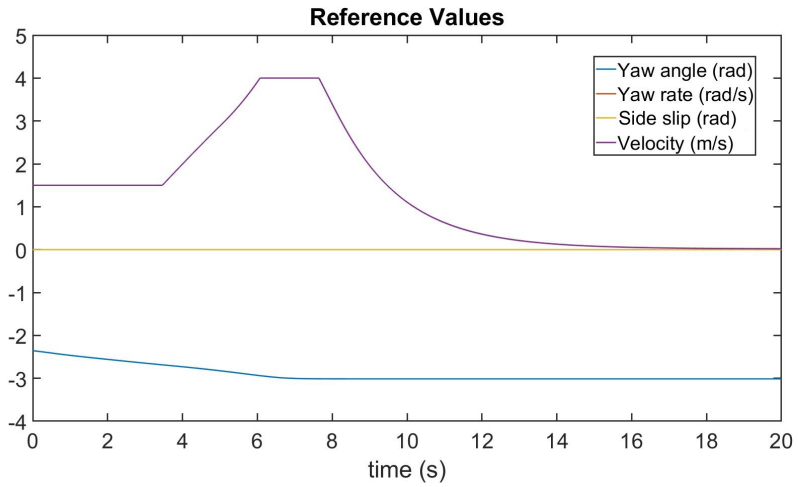

Figure 4.8: Navigation test, reference values

The calculated setpoints are shown in Figure 4.8. Figure 4.9 contains a plot of the simulated states and Figure 4.10 shows the obtained control signals.

Based on Figure 4.11, which shows the calculated vehicle position over the simulation period, it can be concluded that the controller is able to navigate on a rather direct path to the specified position and stop there.

The control signals are adequate with respect to the above conditions, but the change in steering angle may be considered to be too rough. Unfortunately, the Model Predictive Control Toolbox does not allow to limit the rate of change of the control signals in adaptive MPC controllers. Even though the option is available in the settings, it has no influence

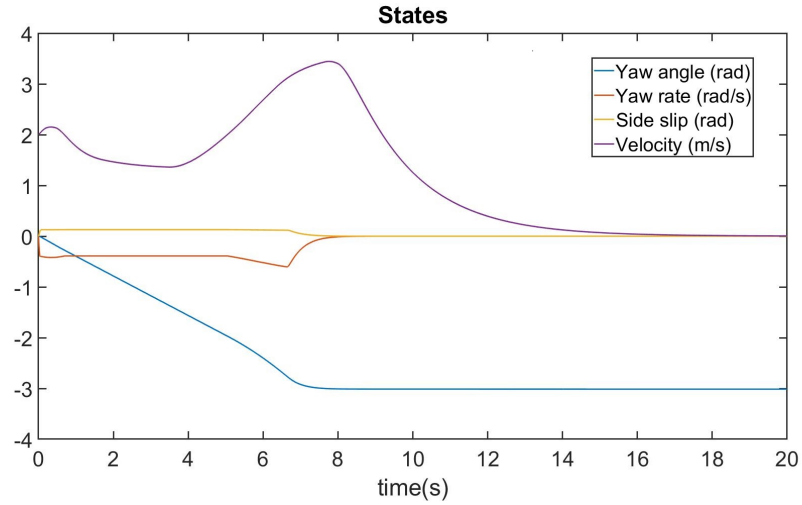

Figure 4.9: Navigation test, simulated states

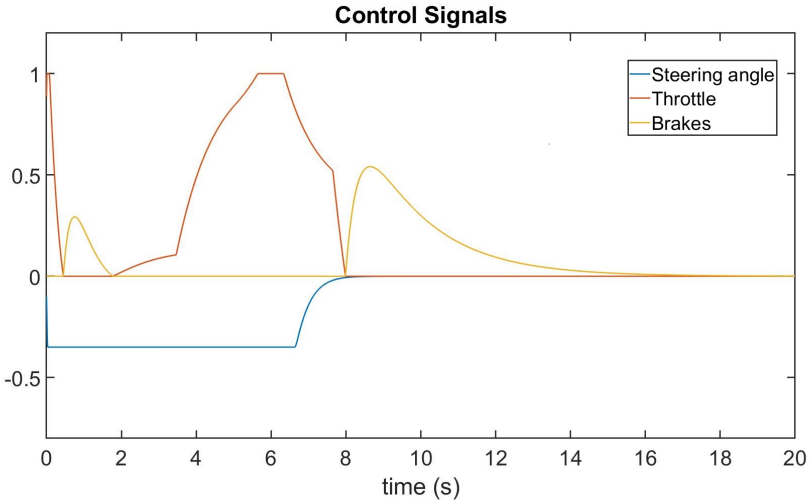

Figure 4.10: Navigation test, control signals

on the simulation and the functionality thus seems to be missing. Nevertheless, should the steering become smoother, this could be achieved by increasing the weight of the steering angle, as described in section 3.6.3.

4.4.2 Follow the Road

As described in Section 3.6.2, the models used differ substantially depending on whether a roadside has been detected or not. While the above test considers the case in which this information is not available, this scenario tests the MPC controller which is relying on this information. Given a target yaw rate and a lateral offset, it is tested whether

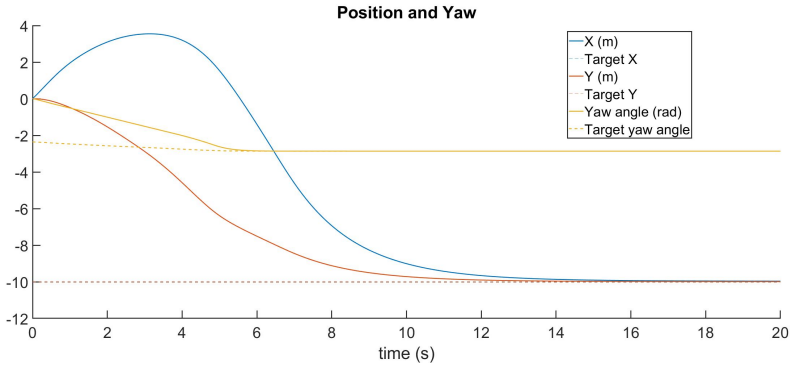

Figure 4.11: Navigation test, calculated vehicle position

the controller is capable of approximating the yaw rate and correcting the offset. At a constant speed, this should result in a circular trajectory of the vehicle.

The initial speed is set to $2m/s$ and the initial yaw angle is set to zero. The target yaw rate is kept constant at $0.2\texttt{rad}$ and the lateral offset is initialized with a relatively large value of $-0.7m$. The target value for the offset is zero.

All SPs remain constant over the simulation period with the values described above. The dummy variable containing the target yaw rate (see 3.6.2) has been omitted in the following figures because its value does not contribute to the evaluation of the control performance. The control signals are shown in Figure 4.13 and the simulated states in 4.12 respectively. This results in the simulated vehicle positions over time, as shown in Figure 4.14.

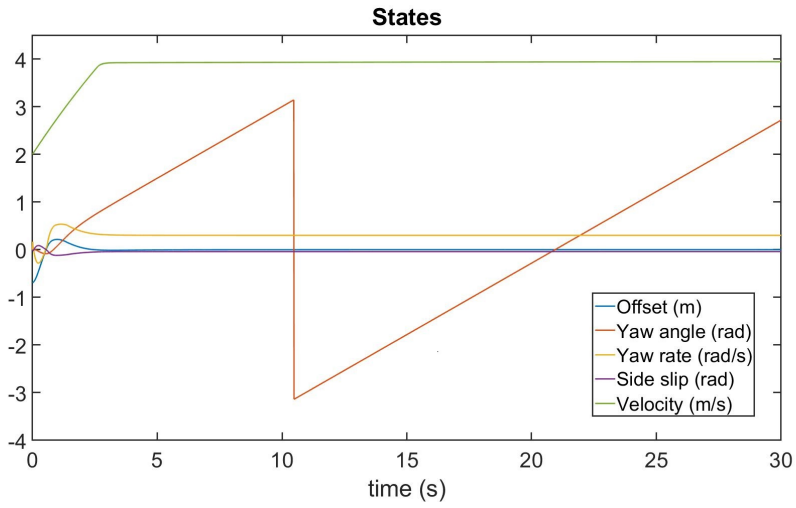

Figure 4.12: Follow the curve, simulated states

It can be observed that the controller is able to correct the initial offset by the steering impulses issued within the first seconds. Furthermore, it is able to assume the target yaw

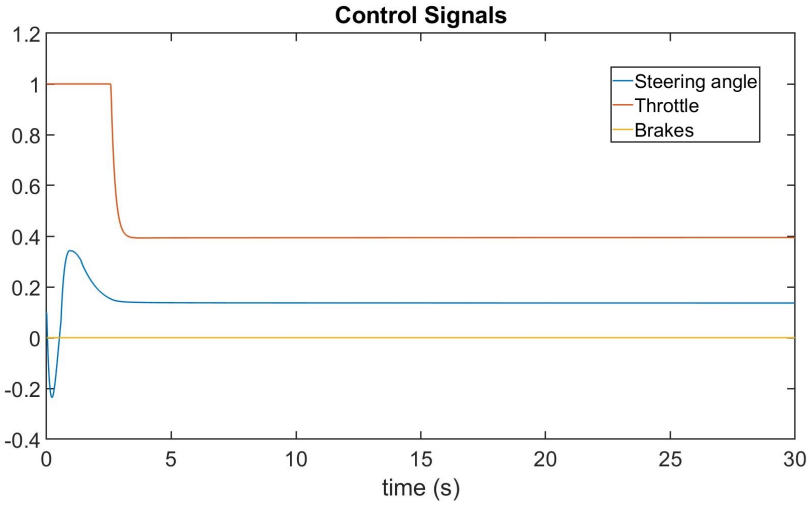

Figure 4.13: Follow the curve, control signals

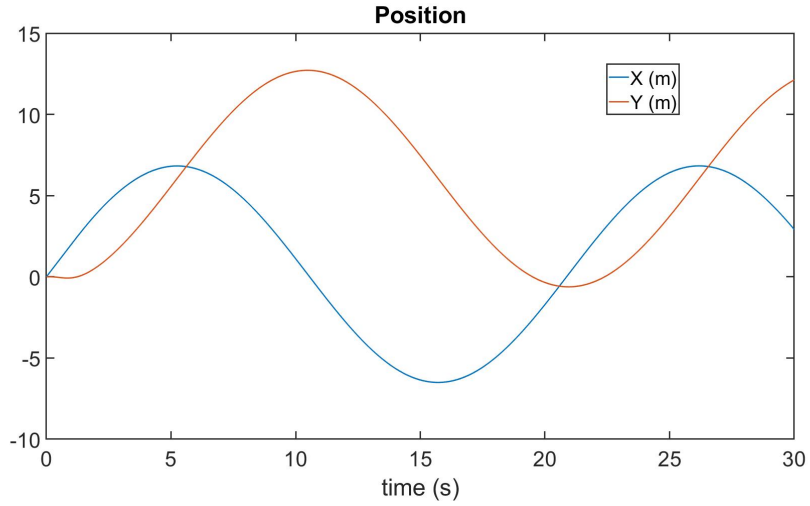

Figure 4.14: Follow the curve, vehicle position

rate later on. The changes of the steering angle are smoother than in the test above.

4.4.3 Obstacle Avoidance

It is vital that the controller can avoid collisions. Since the implemented collision avoidance strategy is to simply apply the brakes, the test case simulates an obstacle at specified coordinates and tests whether the controller can stop before reaching these.

For this scenario, the vehicle is placed at position $(0, 0)$ with an initial speed of $4m/s$ and a heading direction of $45°$. The obstacle is $9m$ away from the vehicle at the time of

detection ($t = 0$). Therefore, the time to collision is initially $2.25s$. In both modes, with and without information from the road detection module, obstacle avoidance is supported and works in the same way. The former mode is used for this test. The target yaw rate is set to zero, therefore the simulated path is supposed to be a straight line.

Ideally, the obstacle should already be detected when it is 4 seconds or farther away from the bicycle. However, the controller also needs to be able to avoid a collision if the obstacle is detected later, as is the case in this scenario. The test is successful if the vehicle stops completely before the impact occurs and if the yaw rate is close to the setpoint value. The target value for the time to impact will not be obtained. This is because if the distance is less than 1m, this variable is set to zero in order to stop the vehicle completely at a reasonable distance from the obstacle (see section 3.6.3).

Figure 4.15 shows the simulated states for this scenario, while Figure 4.16 shows the control signals. As can be seen, the deceleration respectively braking of the vehicle occurs relatively smoothly as the obstacle approaches, and eventually the vehicle stops. As previously mentioned, it can be seen that at a certain point the time to impact is set to zero because the distance to the obstacle is less than 1m. While the heading direction is kept nearly constant in the beginning, the steering angle increases once the vehicle has almost completely stopped. This has also a certain influence on the simulated yaw angle. While this is undesirable behavior, the sensors would in reality detect a lateral deviation which in turn would result in corrective steering. In addition, the steering becomes irrelevant as soon as the vehicle comes to a standstill.

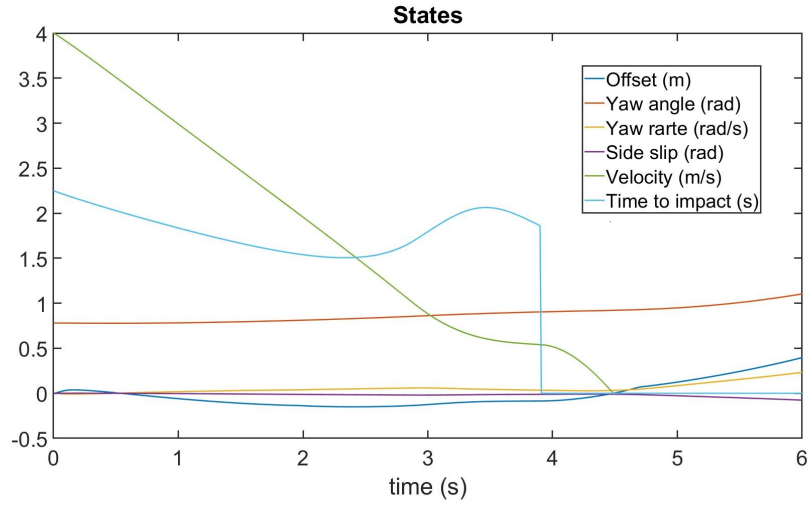

Figure 4.15: Collision avoidance, simulated states

Figure 4.17 shows the calculated positions of the vehicle. In this scenario, the obstacle is located approximately at the position (6.3, 6.3). The vehicle thus stops about $0.9m$ in front of the obstacle.

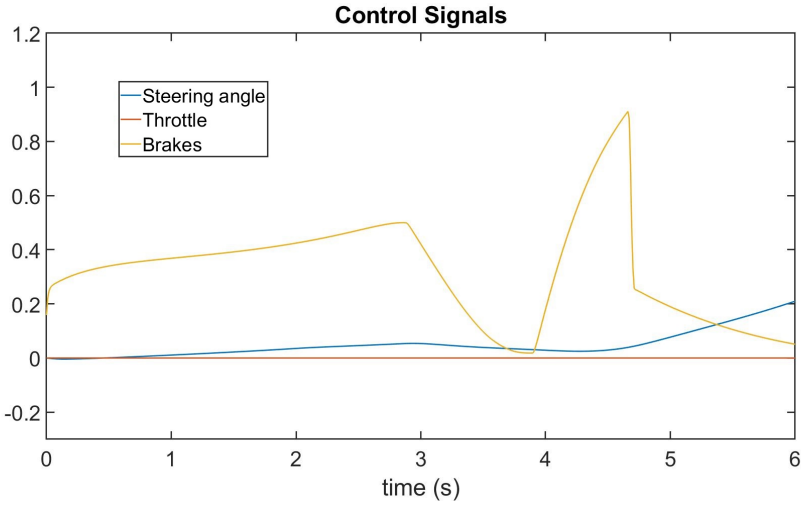

Figure 4.16: Collision avoidance, control signals

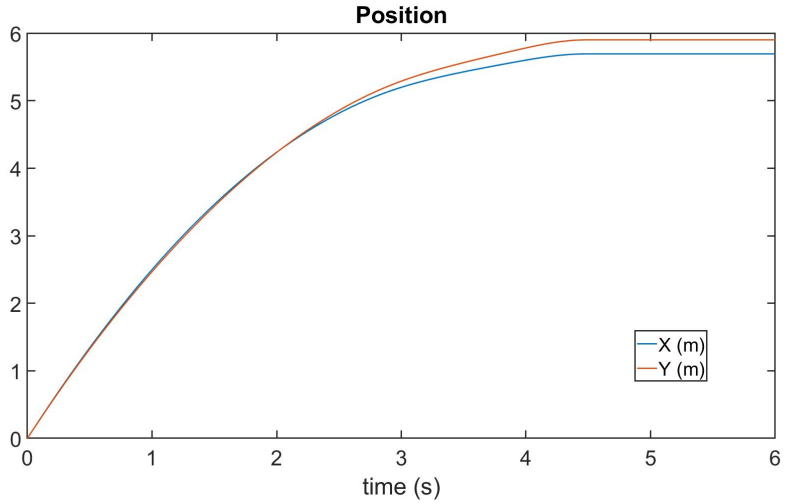

Figure 4.17: Collision avoidance, calculated position

While the above scenario was performed with the controller that uses lane information, similar results are achieved with the GPS only controller.

4.4.4 Route Tracking

In the previous subsections it was shown that the controllers are able to steer the vehicle to given coordinates, to keep the vehicle at a given distance from the edge of a curved road and to stop when an obstacle is detected. However, it is also important that a complete path can be tracked by the controller. Furthermore, the vehicle should reduce its speed before corners and stop completely at the end of the track. For this test the maximum

allowed steering angle was set to $\frac{\pi}{4}$. The route is determined by its waypoints and the desired velocity is computed based on the waypoints and the current vehicle position. All target values are plotted in Figure 4.18.

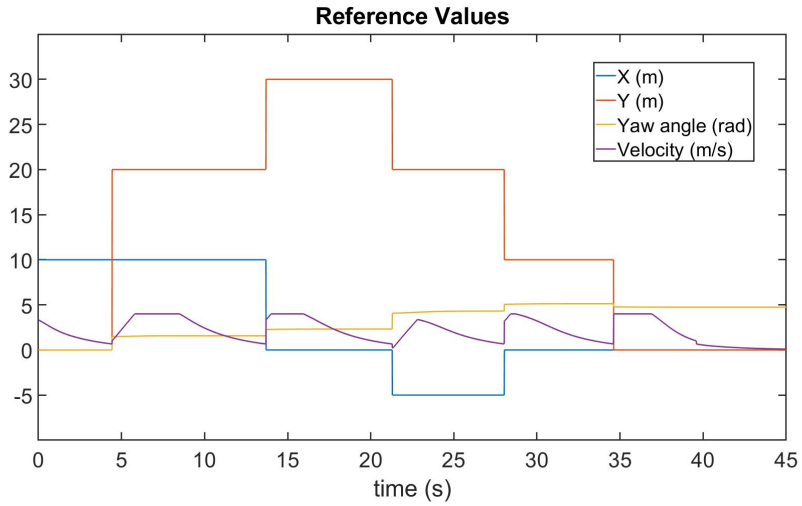

Figure 4.18: Route tracking, reference values

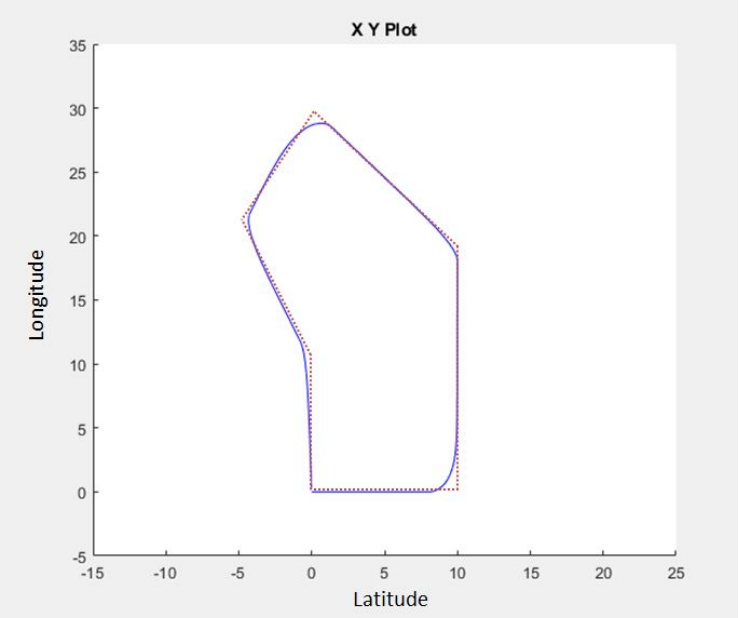

Figure 4.19: Ideal path (red) versus simulated path (blue)

Figures 4.19 and 4.20 show that the controller is able to follow the optimal path accurately. The deviations arise from the steering radius of the bicycle. It can also be seen that the speed of the vehicle is reduced before each corner or waypoint and that the vehicle stops completely at the end of the defined route.

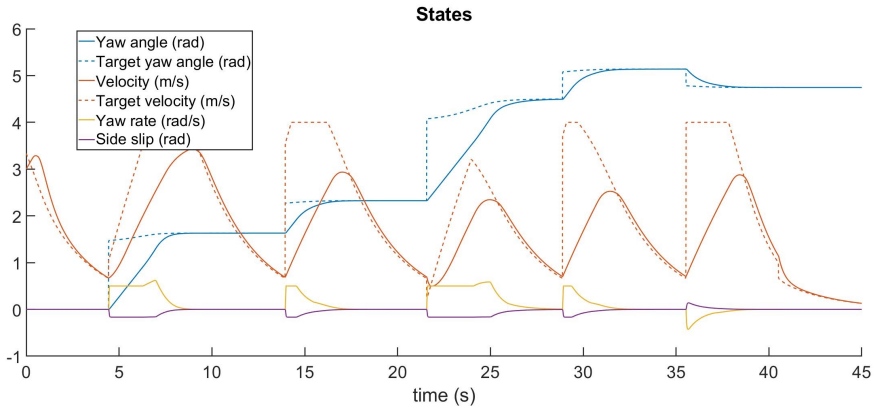

Figure 4.20: Route tracking, simulated states

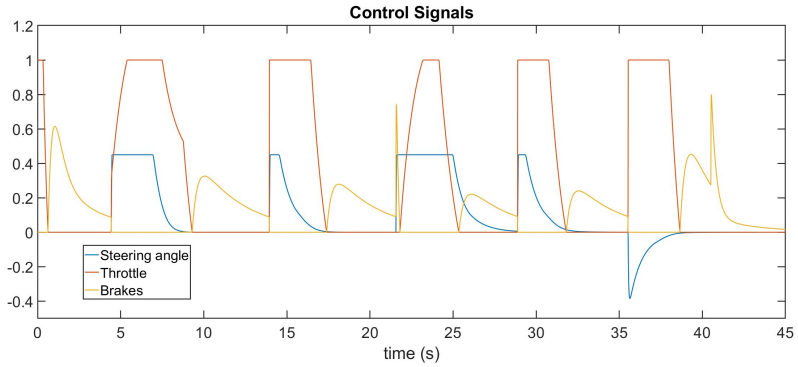

Figure 4.21: Route tracking, control signals

As illustrated in Figure 4.19, the deviations of the simulated path from the ideal path are small. To achieve this, the controller has been set to be quite aggressive. The steering angle thus changes relatively quickly (see 4.21) and the speed is reduced significantly before each turn. It has to be tested with the target bicycle if this configuration is desirable. If necessary, the steering impulses can be softened, for example by increasing the weight of large steering angles. In this case, however, the path will be traced less accurately.

4.5 Closed Loop Testing

As already mentioned in section 3.1, the electric bicycle for which the developed system is intended does not yet exist and will only be built in future projects. However, to test the closed loop behavior of the system, it was mounted on a test vehicle.

With this setup it is possible to assess the capabilities of the system in general. However, it is more to be seen as a proof of concept, since the test hardware is obviously not a bicycle. Therefore, certain restrictions apply for this setup. First, the track detection

module has to be deactivated. Due to the size of the vehicle, the camera is too close to the ground and thus cannot capture a sufficiently large section of the road to provide meaningful results. Second, the parameters for the models used by the controllers (see 3.6.1) are adjusted to bicycles. Hence, the control signals may not be optimal. Third, the vehicle's software interface is not capable of reliably receiving as many signals per second as the control module can transmit. Therefore, the signal rate had to be reduced. Since the vehicle has a considerably smaller turning radius than a bicycle, both the maximum steering angle and the weights of the control signals have been altered to achieve better results in the following tests.

Some quirks were found regarding the test vehicle. Sometimes the steering servo turned sharply to the left, regardless of the current input signals. In addition, the steering has a play of a few degrees, which means that the set steering angle does not necessarily correspond exactly to the actual one. Since there are no sensors to measure the actual steering angle, this has a negative effect on the system accuracy. Finally, there is currently no interface to receive the current speed from the vehicle and therefore for the following tests the velocity had to be calculated based on the GPS positions.

Validation with this test vehicle nevertheless yields useful results. It is possible to validate the integration of the various software components. Namely it can be determined whether the correct MPC controllers are selected, whether they return appropriate actuating signals and whether the connection to the vehicle is working properly. Moreover, it is possible to assess whether the system can autonomously navigate along a track and avoid collisions.

4.5.1 Integration and Capabilities

For the system to function properly, the components tested in the above sections need to work together in a meaningful way and the system as a whole must be able to control the vehicle. This was evaluated on the basis of tests with the test vehicle and the subsequent analysis of the collected log files.

The coordinator module first initializes all the required objects and the MPC controllers. Then all measurement loops are started. The log files collected from the test scenarios show that the data can be fused together for both valid and invalid sensor measurements. The system is therefore robust against nonvalid measurements.

The aggregated data must then be transformed before it can be used by the controllers. It can be seen from the analyzed test cases that the conversion of coordinates into meters as well as the calculation of the actual and target heading direction, yaw rate and target velocity yield reasonable results. The software is also able to select the appropriate controller depending on whether information about the road is acquired by the CV module and whether an obstacle is identified nearby. Depending on the velocity, the corresponding state space parameters are selected. No model is used if not enough data is available.

Based on the aggregated data, the MPC controllers are able to determine control signals that are then properly sent to the vehicle. The vehicle is able to accelerate, brake and steer in the correct direction based on the MPC output. In addition, when approaching a waypoint, the target speed decreases as expected and the next waypoint is loaded.

The calculation time of one iteration varies depending on which model is used and whether Python performs garbage collection at a particular time. For the current implementation, an average of about 5 control signals per second are observed. The developed control module would be able to calculate about 10 such signals per second without the limitations imposed by the test vehicle. When running the software, the CPU usage of the Raspberry Pi varies between 25% and 300%, depending on whether the image processing module is deactivated and whether an obstacle is detected.

The basic prerequisites for autonomous navigation of the vehicle are that the control module is capable of collecting, aggregating and processing the required data and of controlling the vehicle. Since these requirements are met, the entire system can now be tested. The next subsection checks whether the collision avoidance works properly. The last subsection discusses the vehicles capability to autonomously navigate along a real test track.

4.5.2 Collision Avoidance

The control system should prevent the vehicle from colliding with obstacles. The performance depends of course on the detection of the obstacles, see Section 4.1. To test the system's capability in this respect, a route was defined that was blocked by an obstacle (i.e. a brick wall).

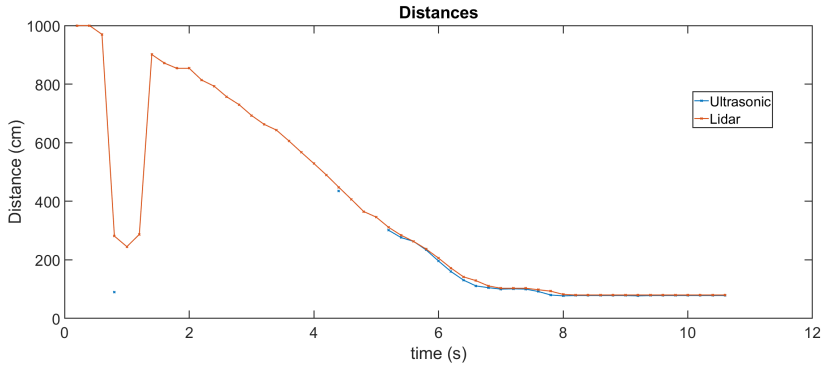

Figure 4.22: Measured distances to the obstacle

The test was performed several times. In all cases, the controller was able to stop the vehicle before the impact occurred. The data obtained during one of the test runs is examined here exemplarily.

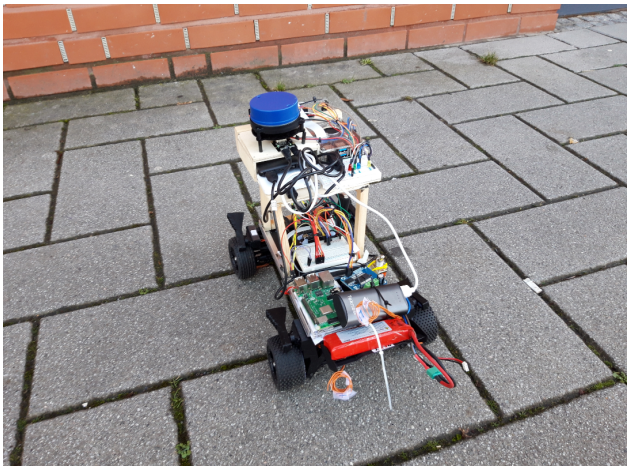

Figure 4.23: Stopped vehicle in front of brick wall

Figure 4.22 shows the measured distance over time as the vehicle approaches the obstacle. It can be seen, that in this test run the ultrasonic and lidar measurements were almost identical, the main difference being the lower range of the ultrasonic sensor. Around the first second, the measured distance dropped significantly, probably caused by a fairly large bump in the road. Since the sensors are mounted relatively low above the ground, the bump may have caused them to identify part of the road as an obstacle. It can also be seen that the vehicle comes to a standstill at a distance of approximately $80cm$ from the obstacle. When looking at all the test runs carried out, there is a certain variability in this number.

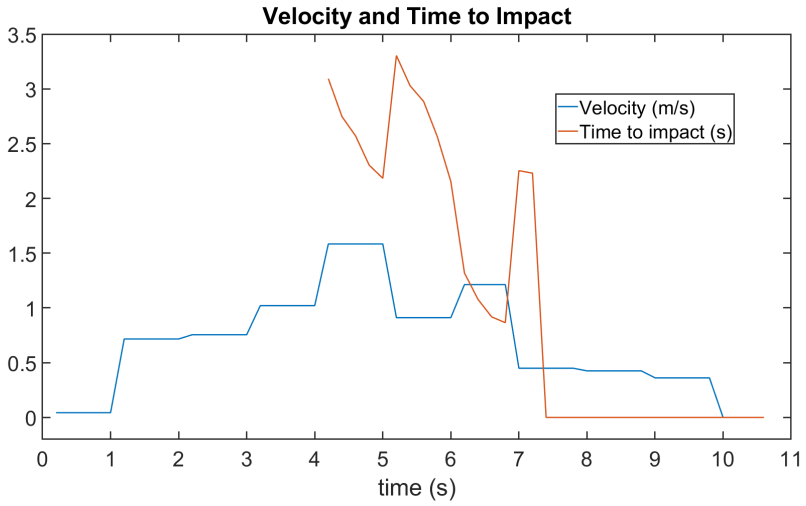

Figure 4.24: Measured speed and time to impact

The metric used by the controller to avoid collisions is the time to impact (see 3.6.2).

It is only calculated when an obstacle is nearby and potentially set to zero whenever the distance to the obstacle is less than one meter (see 3.6.3). The values computed in the examined test run are plotted in Figure 4.24. Due to the low temporal resolution and accuracy of the velocity measurements, the calculated metric is somewhat unstable and has a reduced accuracy. Here the limitations of the speed measurements are clearly observable; although the vehicle has stopped around the eighth second, the speed measured by the GPS module is not reduced to zero until two seconds later. Despite this measurement limitation, the system is able to stop the vehicle properly before the impact happens.

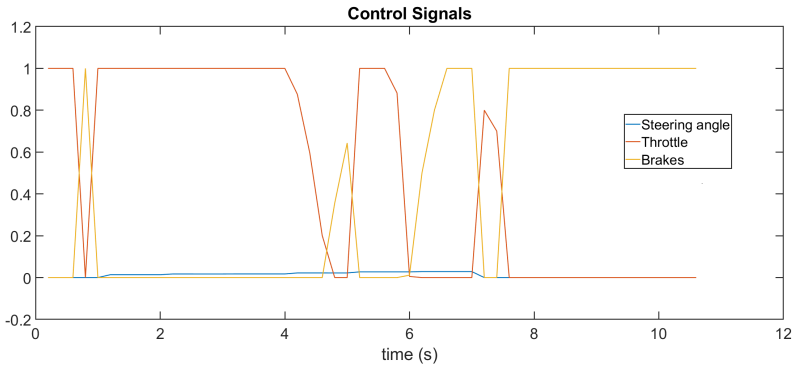

Figure 4.25: Collision avoidance, control signals

Improving the responsiveness of the controller and the smoothness of the control signals (see Figure 4.25) requires both more frequent and more accurate data on the vehicle velocity. This can be achieved by a sensor that measures the speed directly on the basis of the wheel rotation. More frequent control signals may also improve the results. This is possible with the current implementation; the distance measurements are already performed with a higher temporal resolution and the controller is able, as stated above, to double the number of actuating signals computed per second. However, as the vehicle cannot process more signals, increasing the number of iterations per second is currently useless.

4.5.3 Route Tracking

Based on specified start and destination coordinates, a route was calculated which had to be followed by the vehicle in this final test. For the reason stated above, the road detection module was disabled for this test. The test track was completed several times, some of the log files can be found on the attached CD. The data presented herein are taken from one of these runs, namely $fb\_run7.log$.

Figure 4.26 shows the waypoints in blue, and the logged GPS positions in red. It can be seen that the vehicle is capable of tracking the route based on its waypoints. The path driven seems to be fairly accurate, considering that the precision and temporal resolution

of the GPS measurements are limited and that the actual steering angle is not observable. Between waypoint two and three the steering impulse was too hard, which was corrected by the controller later on. Figure 4.27 shows the latitudes and longitudes measured while driving. In addition, the setpoints are displayed as dotted lines. The data shows that the system is capable of approaching the desired values. If the distance to the current target point falls below a certain threshold value, the system switches to the next target coordinates.

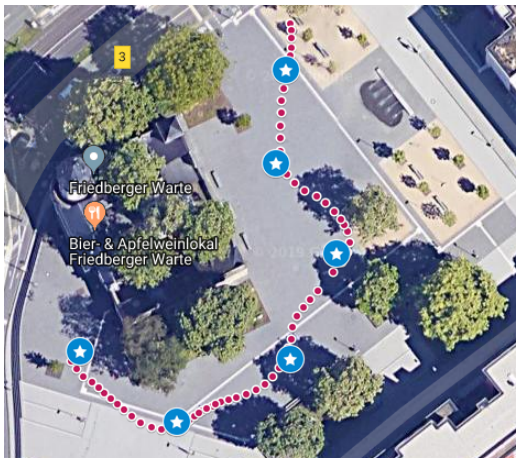

Figure 4.26: Waypoints (blue) and actual route (red)[6]

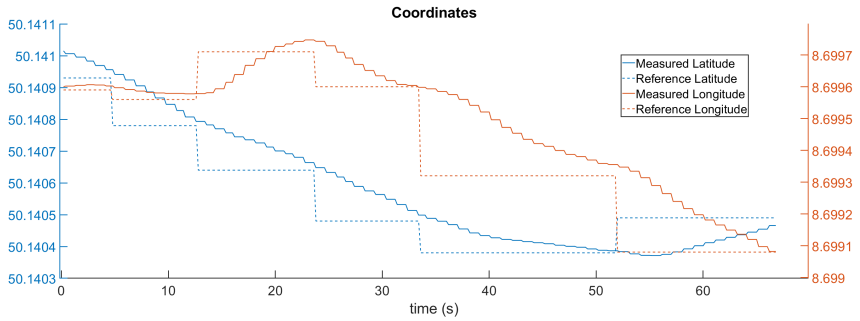

Figure 4.27: Route tracking, target and measured coordinates

Figure 4.28 shows the reference values passed on to the controller. The measured position and the next waypoint are transformed into the desired yaw angle. The target speed is sometimes temporarily reduced in order to prepare for potential upcoming curves. If an obstacle had been detected by either of the two distance sensors, the target time to impact would have been set to $2.5s$. However, no obstacle was detected during this particular test run. The other reference values are set to zero.

[6]Created with Google MyMaps, 15/12/2019
https://drive.google.com/open?id=1DTAhvFw76j82aQrYRsO7SMaoaAF7QAW2&usp=sharing

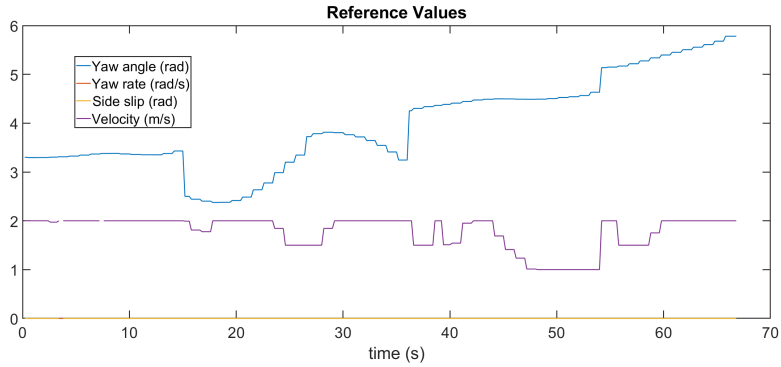

Figure 4.28: Route tracking, calculated target states

The measured states are shown in Figure 4.29. They seem to have a rough temporal resolution due to the limited number of GPS measurements. In addition, the GPS sensor smoothes the returned velocities, resulting in measured values that differ from actual ones. In general, however, it can be seen that the controller is able to manipulate the states in order to reach the setpoints. This can also be seen in Figure 4.30, which plots both desired and measured values for the velocity and the heading direction.

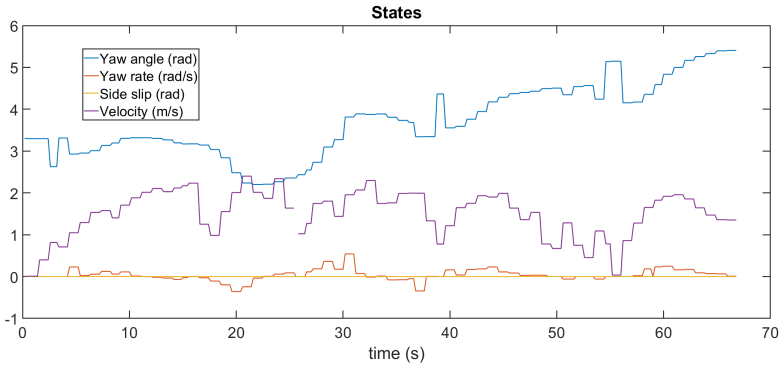

Figure 4.29: Route tracking, measured states

The actuating signals sent to the vehicle are capable of controlling it in a meaningful way. The test vehicle has a fast acceleration and the brakes are very effective. Even at low braking intensities it quickly comes to a standstill. Furthermore, the vehicle has a high rolling resistance and slows down rapidly if the throttle is not used. This, together with the rough temporal resolution of certain variables, results in the slightly hectic control signals for the throttle and brakes as shown in Figure 4.31. The steering signal is more steady. Generally, the system is able to react appropriately to the measured states.

The tests carried out show that the system is able to calculate the target states and to control the vehicle in order to reach them. Thus it is able to follow a track autonomously. However, some limitations have been identified and the system performance varies from

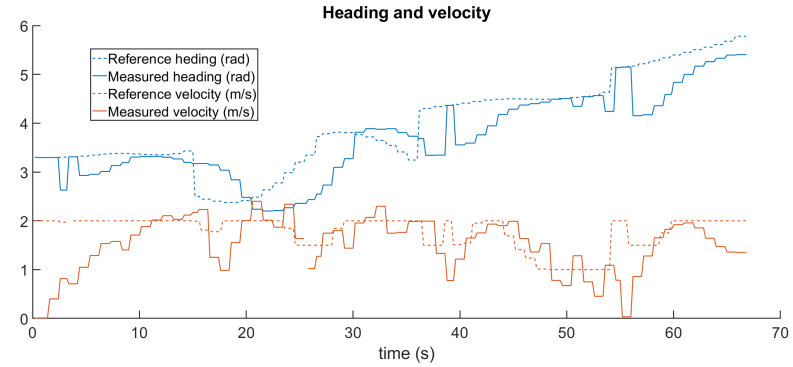

Figure 4.30: Route tracking, heading and velocity

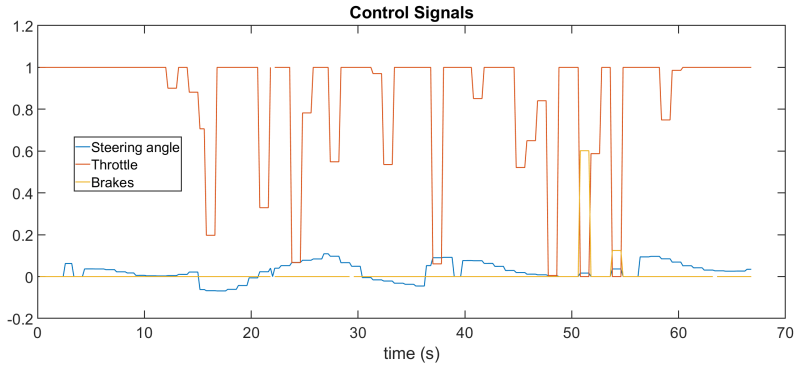

Figure 4.31: Route tracking, controls signals sent to vehicle

run to run. The results could be significantly improved by integrating the accelerometer into the system and by adding sensors to the vehicle to measure the actual steering angle and velocity.

5 Conclusion

In this thesis a control and steering module for an autonomous bicycle was developed. Based on sensor fusion and model predictive control, the module is able to trace routes autonomously.

The system is developed to run on a Raspberry Pi. An ultrasonic sensor and a 2D Lidar sensor are used for distance measurements. The vehicle's position is determined by using GPS signals. Additionally, a camera is used to capture pictures for the roadside detection. An accelerometer was added and connected, but not yet integrated into the system.

In order to recognize the road and the position of the vehicle on it, CV techniques are used. The captured images are denoised using the non-local means algorithm, and Canny edge detection is performed in the HSL color space. The ROI of the image is then selected and a perspective transformation is applied. Thereafter a sliding window algorithm selects the edges belonging to the roadside and a second order polynomial is fitted to the selected data. Based on this, the road curvature and the lateral position of the vehicle on the road are calculated. The validation shows that the implemented software is able to detect straight and curved roads as well as the vehicle's lateral offset. Some limitations, for example regarding weather and low-contrast lighting conditions, have to be taken into account when using this algorithm. Furthermore, the reliability of rejecting frames on which no roadside could be detected should be improved.

A route planning module was implemented to navigate the vehicle from the start to the destination coordinates. This is done by creating an abstract graph of the roads and using Dijkstra's algorithm to determine the shortest path.

Four MPC controllers were implemented to control the movements of the vehicle. They are based on state space equations derived from the linear single-track vehicle model. This relatively straightforward model makes it possible to predict the vehicle behavior and is efficient to compute. Each controller was built with different parameters for different vehicle speeds to account for the non-linearity of the system. The controllers simulate the future states of the system at each timeslot and select appropriate control signals for steering, throttle and brakes. Using multiple models and controllers makes the system more resilient, as it is able to react properly in situations where not all data is available.

In this thesis, all the components of the steering and control module were individually validated. It was established that the each individual component works as expected and

certain constraints and accuracy limits were identified. Finally, the closed loop capabilities of the system were assessed using a test vehicle. Despite some limitations imposed by this setup, it was shown that the control module is indeed capable of autonomously navigating a vehicle and avoiding collisions.

5.1 Future Work

While the developed system already has many features and capabilities, several improvements are still to be made. The limitations resulting from the setup with the test vehicle are presumably solved by the actual development of the targeted bicycle, but there are also various possibilities to further improve the control module itself.

The derivation of the yaw rate from the measured positions, on which the current implementation relies, is rather inaccurate. This can be mitigated by integrating the already connected accelerometer into the system. The precision of the measured yaw rate could thereby be significantly improved. Although there are already two different distance sensors, the obstacle detection is not yet optimal. The use of additional ultrasonic sensors, a more powerful lidar that works better in sunlight or other sensors (e.g. radar) could further enhance the detection of obstacles. It is also important that once the target hardware is available, the actual steering angle and speed are measured directly by the bicycle. This will allow a much better estimation of the system state.

Once the target hardware is available, the camera can be placed at a height at which its pictures cover a useful area, and the roadside detection module can thus be activated. After activation, its performance needs to be tested in the integrated system. It would probably make sense to replace the camera with one that has a wider viewing angle. This would allow to detect roadsides further to the side from the current position of the vehicle, as well as strong right curves. In addition, the computer vision module should become more robust, avoid false positives more reliably and take the potential sideway leaning of the vehicle into account. Using a convolutional neural network to detect the roadside instead of the current edge detection techniques may improve the robustness and accuracy of the results. The instance segmentation approach for lane detection proposed by Neven et al. [55] may be a promising candidate for this.

The single-track model used to predict the system behavior is simplistic and comes with a number of constraints. It may be replaced by a more sophisticated model, such as a model that takes rolling motions into account. This may allow to predict the future states of the vehicle more accurately. However, as with all proposed extensions and improvements, the limited computing power has to be kept in mind and it has to be ensured that the system continues to be able to react rapidly.

It will be interesting to assess the performance of the control module on the real target

bicycle. In this thesis, a flexible and expandable steering and control module was developed which could lay the foundation for the development of an autonomous bicycle.

Bibliography

[1] Driverless cars are stuck in a jam. *The Economist*, Oct 2019.

[2] A. Althwaini. *Aerial Image segmentation.* PhD thesis, Laurentian University, 08 2016.

[3] J. M. Barrios. *Content-Based Video Copy Detection.* PhD thesis, University of Chile, 11 2013.

[4] A. Bemporad, M. Morari, and N. L. Ricker. Model predictive control toolbox user's guide. *The mathworks*, 2019.

[5] J. Betz, A. Wischnewski, A. Heilmeier, F. Nobis, T. Stahl, L. Hermansdorfer, B. Lohmann, and M. Lienkamp. What can we learn from autonomous level-5 motorsport? In *9th International Munich Chassis Symposium 2018*, pages 123–146. Springer, 2019.

[6] A. Beznos, A. Formal'Sky, E. Gurfinkel, D. Jicharev, A. Lensky, K. Savitsky, and L. Tchesalin. Control of autonomous motion of two-wheel bicycle with gyroscopic stabilisation. In *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No. 98CH36146)*, volume 3, pages 2670–2675. IEEE, 1998.

[7] R. Bishop. What tesla's grab of deepscale is all about. *Forbes*, Oct 2019.

[8] D. A. Bohn. Environmental effects on the speed of sound. In *Audio Engineering Society Convention 83*. Audio Engineering Society, 1987.

[9] G. Bradski and A. Kaehler. *Learning OpenCV: Computer vision with the OpenCV library.* " O'Reilly Media, Inc.", 2008.

[10] A. Buades, B. Coll, and J.-M. Morel. Non-local means denoising. *Image Processing On Line*, 1:208–212, 2011.

[11] J. Carter, K. Schmid, K. Waters, L. Betzhold, B. Hadley, R. Mataosky, and J. Halleran. *Lidar 101: An Introduction to Lidar Technology, Data, and Applications.* National Oceanic and Atmospheric Administration (NOAA), 2012.

[12] E. Chavolla, D. Zaldivar, E. Cuevas, and M. A. Perez. Color spaces advantages and disadvantages in image color clustering segmentation. In *Advances in Soft Computing and Machine Learning in Image Processing*, pages 3–22. Springer, 2018.

[13] H.-D. Cheng, X. H. Jiang, Y. Sun, and J. Wang. Color image segmentation: advances and prospects. *Pattern recognition*, 34(12):2259–2281, 2001.

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, 3rd Edition*. The MIT Press, 2009.

[15] E. R. Davies. *Computer and machine vision: theory, algorithms, practicalities*. Academic Press, 2012.

[16] R. A. DeCarlo. *Linear systems: A state variable approach with numerical implementation*. Prentice-Hall, Inc., 1989.

[17] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[18] L. Ding and A. Goshtasby. On the canny edge detector. *Pattern Recognition*, 34(3):721–725, 2001.

[19] J. Dokic, B. Müller, and G. Meyer. European roadmap smart systems for automated driving. *European Technology Platform on Smart Systems Integration*, page 39, 2015.

[20] A. Dressel and A. Rahman. Measuring sideslip and camber characteristics of bicycle tyres. *Vehicle system dynamics*, 50(8):1365–1378, 2012.

[21] G. A. Einicke. *Smoothing, Filtering and Prediction: Estimating the Past, Present and Future Second Edition*. InTech, 2019.

[22] O.-I. Erduran. Energieschonende platzierung von elektrofahrzeugen mit methoden des maschinellen lernens. unpublished master thesis, 2019.

[23] G. M. Farinella, S. Battiato, and R. Cipolla. *Advanced topics in computer vision*. Springer, 2013.

[24] P. Goldin. 10 advantages of autonomous vehicles, February 2018. itsdigest.com [Online; posted 25-Oct-2019].

[25] S. Grasemann. Entwurf und simulation eines teilautonomen fahrzeugmodells mit assistenzsystemen. unpublished bachelor thesis, 2019.

[26] S. Gu and R. Timofte. A brief review of image denoising algorithms and beyond. *Springer series on Challenges in Machine Learning*, 1, 2019.

[27] S. Gupta and S. G. Mazumdar. Sobel edge detection algorithm. *International journal of computer science and management Research*, 2(2):1578–1583, 2013.

[28] A. Hechri, R. Hmida, and M. Abdellatif. Robust road lanes and traffic signs recognition for driver assistance system. *International Journal of Computational Science and Engineering*, 10:202–209, 01 2015.

[29] A. Herrmann-Fankhaenel. How to take advantage of online platforms like the sharing economy does. In *Co-Creation*, pages 77–88. Springer, 2019.

[30] A. B. Hillel, R. Lerner, D. Levi, and G. Raz. Recent progress in road and lane detection: a survey. *Machine vision and applications*, 25(3):727–745, 2014.

[31] T. Huang. Computer vision: Evolution and promise. *1996 CERN SCHOOL OF COMPUTING*, page 21, 1996.

[32] J. M. Hughes. *Real world instrumentation with Python: automated data acquisition and control systems.* " O'Reilly Media, Inc.", 2010.

[33] S. international. Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles. *SAE International,(J3018)*, 2018.

[34] M. Kam, X. Zhu, and P. Kalata. Sensor fusion for mobile robot navigation. *Proceedings of the IEEE*, 85(1):108–119, 1997.

[35] K. Kanjanawanishkul. Lqr and mpc controller design and comparison for a stationary self-balancing bicycle robot with a reaction wheel. *Kybernetika*, 51(1):173–191, 2015.

[36] R. M. M. Karl Johan Aström. *Feedback Systems, An Introduction for Scientists and Engineers.* Princeton University Press, 2010.

[37] R. Keenan. Advanced lane finding. `https://github.com/udacity/CarND-Advanced-Lane-Lines`, last accessed 11/18/19, 2017.

[38] A. G. Kek, R. L. Cheu, and M. L. Chor. Relocation simulation model for multiple-station shared-use vehicle systems. *Transportation research record*, 1986(1):81–88, 2006.

[39] M. Kelemen, I. Virgala, T. Kelemenová, L. Miková, P. Frankovskỳ, T. Lipták, and M. Lörinc. Distance measurement via using of ultrasonic sensor. *Journal of automation and control*, 3(3):71–74, 2015.

[40] L. Keo and Y. Masaki. Trajectory control for an autonomous bicycle with balancer. In *2008 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pages 676–681. IEEE, 2008.

[41] L. Keo and M. Yamakita. Control of an autonomous electric bicycle with both steering and balancer controls. *Advanced Robotics*, 25(1-2):1–22, 2011.

[42] Y. Kim and M. Yamakita. Model predictive path-following for bike robot. In *The SICE Annual Conference 2013*, pages 1592–1597. IEEE, 2013.

[43] R. Klette. *Concise computer vision.* Springer, 2014.

[44] M. Koegel and R. Findeisen. A fast gradient method for embedded linear predictive control. *IFAC Proceedings Volumes*, 44(1):1362–1367, 2011.

[45] R. Laganiere. Compositing a bird's eye view mosaic. In *Proc. Conf. Vision Interface*, pages 382–387. Citeseer, 2000.

[46] Level 5 Design Pty Ltd. Connected and autonomous vehicles. `https://www.level5design.com.au/connected-autonomous-vehicles.html`, last accessed on 11/18/19, 2019.

[47] M. Liang and X. Hu. Recurrent convolutional neural network for object recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3367–3375, 2015.

[48] T. lindeberg. Edge detection. In M. Hazewinkel, editor, *Encyclopaedia of Mathematics*. Springer, 2001.

[49] X. Ma, B. Li, Y. Zhang, and M. Yan. The canny edge detection and its improvement. In *International Conference on Artificial Intelligence and Computational Intelligence*, pages 50–58. Springer, 2012.

[50] E. Maibach, L. Steg, and J. Anable. Promoting physical activity and reducing climate change: Opportunities to replace short car trips with active transportation. *Preventive medicine*, 49(4):326–327, 2009.

[51] B. G. McHenry and R. Mc Henry. Accident reconstruction. In *Southeast Coast Collision Conference*. McHenry Software, Inc., 2008.

[52] P. Mellodge. *A Practical Approach to Dynamical Systems for Engineers*. Woodhead Publishing, 2015.

[53] G. A. Munoz-Hernandez, D. I. Jones, et al. *Modelling and controlling hydropower plants*. Springer Science and Business Media, 2012.

[54] U. Nations. Current and planned global and regional navigation satellite systems and satellite-based augmentations systems. In *Proceedings of ICG*, pages 15–40, 2010.

[55] D. Neven, B. De Brabandere, S. Georgoulis, M. Proesmans, and L. Van Gool. Towards end-to-end lane detection: an instance segmentation approach. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 286–291. IEEE, 2018.

[56] T. A. S. Nielsen and S. Haustein. On sceptics and enthusiasts: What are the expectations towards self-driving cars? *Transport policy*, 66:49–55, 2018.

[57] Y. Noda, T. Sumioka, and M. Yamakita. An application of fast mpc for bike robot. In *2012 Proceedings of SICE Annual Conference (SICE)*, pages 540–545. IEEE, 2012.

[58] P. Orukpe. Model predictive control fundamentals. *Nigerian Journal of Technology*, 31(2):139–148, 2012.

[59] F. Owen. *Control Systems Engineering*. California Polytechnic State University, May 2012.

[60] W. Pree and E. Gamma. *Design patterns for object-oriented software development*, volume 183. Addison-wesley Reading, MA, 1995.

[61] Raspberry Pi (Trading) Ltd. *Raspberry Pi 4 Model B Datasheet*, June 2019.

[62] P. Riekert and T.-E. Schunck. Zur fahrmechanik des gummibereiften kraftfahrzeugs. *Ingenieur-Archiv*, 11(3):210–224, 1940.

[63] W. Rong, Z. Li, W. Zhang, and L. Sun. An improved canny edge detection algorithm. In *2014 IEEE International Conference on Mechatronics and Automation*, pages 577–582. IEEE, 2014.

[64] A. Rosebrock. 4 point opencv getperspective transform example. `https://www.pyimagesearch.com/2014/08/25/4-point-opencv-getperspective-transform-example`, last accessed 11/08/2019, 2014.

[65] D. Schramm, M. Hiller, and R. Bardini. Einspurmodelle. In *Modellbildung und Simulation der Dynamik von Kraftfahrzeugen*, pages 225–254. Springer, 2018.

[66] B. Schwarz. Lidar: Mapping the world in 3d. *Nature Photonics*, 4(7):429, 2010.

[67] Seeed Technology Co.,Ltd. *Ceramic Ultrasonic Sensor Specification*, July 2017.

[68] Shenzhen Yuedeng Technology Co. Ltd. *YDLIDAR X4 User Manual*, January 2019.

[69] V. S. Soon, C. Y. Lam, and E. Phaklen. Reconfigurable hardware acceleration of rgb to hsl converter. In *Advanced Computer and Communication Engineering Technology*, pages 1275–1282. Springer, 2016.

[70] S. Stasinopoulos, M. Zhao, and Y. Zhong. Human behavior inspired obstacle avoidance & road surface quality detection for autonomous bicycles. In *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 2121–2126. IEEE, 2015.

[71] R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2011.

[72] A. Taeihagh and H. S. M. Lim. Governing autonomous vehicles: emerging responses for safety, liability, privacy, cybersecurity, and industry risks. *Transport reviews*, 39(1):103–128, 2019.

[73] E. R. Teoh and D. G. Kidd. Rage against the machine? google's self-driving cars versus human drivers. *Journal of safety research*, 63:57–60, 2017.

[74] u-blox Holding AG. *NEO-6u-blox 6 GPS Modules Data Sheet*, 2018.

[75] R. L. Williams, D. A. Lawrence, et al. *Linear state-space control systems*. Wiley Online Library, 2007.

[76] Y. Yavin. Point-to-point and collision avoidance control of the motion of an autonomous bicycle. *Computers & Mathematics with Applications*, 50(10-12):1525–1542, 2005.

[77] H. Yetkin, S. Kalouche, M. Vernier, G. Colvin, K. Redmill, and U. Ozguner. Gyroscopic stabilization of an unmanned bicycle. In *2014 American Control Conference*, pages 4549–4554. IEEE, 2014.

[78] M. Zhao, S. Stasinopoulos, and Y. Yu. Obstacle detection and avoidance for autonomous bicycles. In *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, pages 1310–1315. IEEE, 2017.

[79] D. Ziou, S. Tabbone, et al. Edge detection techniques-an overview. *Pattern Recognition and Image Analysis C/C of Raspoznavaniye Obrazov I Analiz Izobrazhenii*, 8:537–559, 1998.

List of Abbreviations

GPIO	General-purpose input/output
MPC	Model predictive control
SP	Setpoint
CSI	Camera Serial Interface
AV	Autonomous Vehicle
SAE	Society of Automotive Engineers
CV	Computer Vision
GPS	Global Positioning System
UART	Universal Asynchronous Receiver/Transmitter
ROI	Region of Interesst
Lidar	Light Detection and Ranging
HSL	Hue, saturation, lightness
RGB	red, green, blue
LED	Light-emitting diode
ABS	anti-lock braking system
CNN	Convolutional Neural Network
API	Application programming interface
USB	Universal Serial Bus