# A High Precision Lane Following Control Method for an Autonomous Robot

Scientific work for obtaining the academic degree

Bachelor of Science (B.Sc.)

at the Department of Mechanical Engineering of Technical University of Munich

| | |
|---|---|
| **Supervised by** | Prof. Dr.-Ing. Markus Lienkamp<br>Jean-Michael Georg, M.Sc.<br>Chair of Automotive Technology |
| **Submitted by** | Yuxuan Xue<br>Schlierseestraße 29<br>81541 München |
| **Submitted on** | December 12, 2019 |

Chair of Automotive Technology
Department of Mechanical Engineering
Technical University of Munich

# Project description

## A High Precision Lane Following Control Method for an Autonomous Robot

Nowadays, automation technology plays an increasingly significant role in many economic sectors. In the future, autonomous robots can also be used in the road construction. These robots have to follow existing lane markings with high precision. Therefore, in this bachelor thesis, a path-following controller should be developed, which allows road marking machines to move along existing road markings with high accuracy.

The challenge is on the one hand in the missing localization of the robot in global coordinates and the resulting requirement to perform the lateral control in the vehicle coordinate system, as well as in the requirement to achieve a control accuracy of a few millimeters scale.

The following points are to be investigated by Mr. Yuxuan Xue:

- Literature research and showing the state-of-the-art
- Evaluation of various control concepts
- Development and optimization of the controller structure
- Implementation of the controller in Python as the ROS node
- A critical discussion of results

The elaboration should document the individual work steps in a clear form. The candidate undertakes to write the master thesis independently and to state the scientific sources used by him.

The submitted work remains the property of the chair as an examination document and may only be made accessible to third parties with the consent of the chair holder.

Announcement date: 12.06.2019

Submission date: 12.12.2019

Prof. Dr.-Ing. Markus Lienkamp

Jean-Michael Georg, M.Sc.

Chair of Automotive Technology
Department of Mechanical Engineering
Technical University of Munich

# Geheimhaltungsverpflichtung

Herr: **Xue, Yuxuan**

Gegenstand der Geheimhaltungsverpflichtung sind alle mündlichen, schriftlichen und digitalen Informationen und Materialien die der Unterzeichner vom Lehrstuhl oder von Dritten im Rahmen seiner Tätigkeit am Lehrstuhl erhält. Dazu zählen vor allem Daten, Simulationswerkzeuge und Programmcode sowie Informationen zu Projekten, Prototypen und Produkten.

Der Unterzeichner verpflichtet sich, alle derartigen Informationen und Unterlagen, die ihm während seiner Tätigkeit am Lehrstuhl für Fahrzeugtechnik zugänglich werden, strikt vertraulich zu behandeln.

Er verpflichtet sich insbesondere:

- derartige Informationen betriebsintern zum Zwecke der Diskussion nur dann zu verwenden, wenn ein ihm erteilter Auftrag dies erfordert,
- keine derartigen Informationen ohne die vorherige schriftliche Zustimmung des Betreuers an Dritte weiterzuleiten,
- ohne Zustimmung eines Mitarbeiters keine Fotografien, Zeichnungen oder sonstige Darstellungen von Prototypen oder technischen Unterlagen hierzu anzufertigen,
- auf Anforderung des Lehrstuhls für Fahrzeugtechnik oder unaufgefordert spätestens bei seinem Ausscheiden aus dem Lehrstuhl für Fahrzeugtechnik alle Dokumente und Datenträger, die derartige Informationen enthalten, an den Lehrstuhl für Fahrzeugtechnik zurückzugeben.

Eine besondere Sorgfalt gilt im Umgang mit digitalen Daten:

- Für den Dateiaustausch dürfen keine Dienste verwendet werden, bei denen die Daten über einen Server im Ausland geleitet oder gespeichert werden (Es dürfen nur Dienste des LRZ genutzt werden (Lehrstuhllaufwerke, Sync&Share, GigaMove).
- Vertrauliche Informationen dürfen nur in verschlüsselter Form per E-Mail versendet werden.
- Nachrichten des geschäftlichen E-Mail Kontos, die vertrauliche Informationen enthalten, dürfen nicht an einen externen E-Mail Anbieter weitergeleitet werden.
- Die Kommunikation sollte nach Möglichkeit über die (my)TUM-Mailadresse erfolgen.

Die Verpflichtung zur Geheimhaltung endet nicht mit dem Ausscheiden aus dem Lehrstuhl für Fahrzeugtechnik, sondern bleibt 5 Jahre nach dem Zeitpunkt des Ausscheidens in vollem Umfang bestehen. Die eingereichte schriftliche Ausarbeitung darf der Unterzeichner nach Bekanntgabe der Note frei veröffentlichen.

Der Unterzeichner willigt ein, dass die Inhalte seiner Studienarbeit in darauf aufbauenden Studienarbeiten und Dissertationen mit der nötigen Kennzeichnung verwendet werden dürfen.

Datum: 12. Dezember 2019

Unterschrift: ____________________

Chair of Automotive Technology
Department of Mechanical Engineering
Technical University of Munich

# Erklärung

Ich versichere hiermit, dass ich die von mir eingereichte Abschlussarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Garching, den 12. Dezember 2019

Yuxuan Xue

Chair of Automotive Technology
Department of Mechanical Engineering
Technical University of Munich

# Declaration of Consent, Open Source

Hereby I, Xue, Yuxuan, born on August 31, 1996, make the software I developed during my Bachelor Thesis available to the Institute of Automotive Technology under the terms of the license below.

Garching, December 12, 2019

Yuxuan Xue

Copyright 2019 Xue, Yuxuan

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

# Contents

**List of Abbreviations** .......... **III**

**Formula Symbols** .......... **V**

**1 Introduction** .......... **1**

**1.1 Motivation** .......... **1**

**1.2 Objective of the Thesis** .......... **2**

**1.3 Structure of the Thesis** .......... **3**

**2 State of the Art** .......... **5**

**2.1 Autonomous Driving** .......... **5**

**2.2 Modules of Autonomous Vehicles** .......... **7**

2.2.1 Hardware Module .......... 7

2.2.2 Software Module .......... 8

**2.3 Path-Tracking Control Method** .......... **9**

2.3.1 Feedback-based Control .......... 10

2.3.2 Geometry-based Control .......... 13

2.3.3 Model-based Control .......... 15

**3 Methods** .......... **19**

**3.1 Vehicle Model** .......... **19**

3.1.1 Kinematic Single-Track Model .......... 19

3.1.2 Dynamic Single-Track Model .......... 23

3.1.3 Kinematic Model Linearization .......... 24

**3.2 Tools** .......... **26**

3.2.1 ROS .......... 26

3.2.2 SILAB .......... 27

**4 Controller Design** .......... **29**

**4.1 Environment** .......... **29**

4.1.1 Path .......... 29

4.1.2 Vehicle .......... 29

4.1.3 Coordinate Transformation .......... 30

4.1.4 Minimal Distance .......... 33
4.1.5 Latency .......... 34
**4.2 Implementation of Controllers .......... 35**
4.2.1 Incremental PID Controller .......... 35
4.2.2 Stanley Controller .......... 37
4.2.3 Pure Pursuit Controller .......... 39
**4.3 Combination of Controllers .......... 41**
**4.4 Experiments in SILAB .......... 43**
**5 Result and Discussion .......... 45**
**5.1 Lateral Control Performance .......... 45**
5.1.1 Stanley Controller .......... 46
5.1.2 PID Controller .......... 49
5.1.3 Pure Pursuit Controller .......... 54
5.1.4 Adaptive Pure Pursuit Controller .......... 57
5.1.5 Combined Pure Pursuit - PID Controller .......... 59
**5.2 Real Route .......... 60**
**5.3 Conclusion .......... 63**
**6 Overview and Outlook .......... 65**
**6.1 Summary of the Thesis .......... 65**
**6.2 Future works .......... 66**
**List of Figures .......... i**
**List of Tables .......... v**
**Bibliography .......... vii**
**Appendix .......... xi**

# List of Abbreviations

| | |
|---|---|
| AD | Analog-Digital |
| CASF | Collsion Avoidance Steering Fuzzy Module |
| CMU | Carnegie Mellon University |
| DA | Digital-Analog |
| DARPA | Defense Advanced Research Projects Agency |
| DPU | Digital Processing Unit |
| FTM | Chair pf Automotive Technology |
| GUI | Graphical User Interface |
| HMI | Human-Machine-Interaction |
| ITS | Intelligent Transportation System |
| IUV | Intelligent Unmanned Vehicle |
| LQG | Linear-Quadratic-Gaussian |
| LQR | Linear-Quadratic Regulator |
| ML | Machine Learning |
| MPC | Model Predictive Control |
| OS | Operating System |
| PI | Proportional Integral |
| PID | Proportion Integration Differentiation |
| PP-PID | Pure Pursuit - Proportional Integral Differentiation |
| PV | Process Variable |
| ROS | Robot Operating System |
| SP | Set Point |
| SToRM | System for Teleoperated Road Marking |
| TSF | Target Steering Fuzzy Module |
| WIVW | Würzburg Institut for Traffic Sciences GmbH |

# Formula Symbols

| Formula Symbols | Unit | Description |
|---|---|---|
| $(\dot{-})$ | - | First Order Differential |
| $(\ddot{-})$ | - | Second Order Differential |
| $(-)_{\mathrm{f}}$ | - | Front Wheel Component |
| $(-)_{\mathrm{r}}$ | - | Rear Wheel Component |
| $\theta$ | rad | Vehicle Heading Angle |
| $\theta_{\mathrm{p}}$ | rad | Path Heading Angle |
| $\theta_{\mathrm{e}}$ | rad | Vehicle Heading Error |
| $\delta$ | rad | Vehicle Steering Angle |
| $L$ | m | Vehicle Length |
| $v_{\mathrm{x}}$ | $\frac{m}{s}$ | Vehicle Longitudinal Velocity |
| $e$ | m | Vehicle Lateral Error |
| $R$ | m | Path Curvature Radius |
| $\kappa$ | $\frac{1}{\mathrm{m}}$ | Path Curvature |
| $\omega$ | $\frac{rad}{s}$ | Vehicle lateral acceleration |
| $L_{\mathrm{d}}$ | m | Look-ahead Distance |

# 1 Introduction

Intelligent Unmanned Vehicle (IUV) are the important component of Intelligent Transportation System (ITS). The software algorithm modules of an unmanned vehicle mainly include perception, localization, prediction, routing, planning and control.

Feedback control is the basic algorithmic module of an unmanned vehicle and includes two parts: lateral control and longitudinal control. The lateral control is mainly used for the control of the steering wheel of the vehicle, while the longitudinal control is responsible for the control of the throttle and brake of the vehicle. The lateral control and longitudinal control should work together to ensure that the unmanned vehicle moves according to the predetermined reference trajectory.

Specifically, the lateral control performs tracking control according to the path, curvature and other information given by the upper layer motion planning to reduce the tracking error, ensure the stability and comfort of the vehicle.

In this work, a high precision kinematic lane following control method is proposed to solve the lateral control problem. At the end, the proposed techniques for vehicle motion control are tested in a virtual environment (SILAB).

## 1.1 Motivation

Intelligent unmanned vehicles may be the most significant innovation in the transportation since automobiles were first invented. Although intelligent unmanned cars and trucks are still in their infancy, they can fundamentally change the way how people and goods move around — with profound implications for safety, lifestyle and environment.

In general terms, the benefits of IUVs are reflected in the three following categories:

- Safety
  According to the Federal Statistical Office in Germany, around 2.4 million traffic accidents were recorded in 2012[1]. A total of 362,993 accidents involving personal injury are attributable to misconduct by the driver. By use of self-driving connected vehicles, these accidents can be avoided. The vehicles are always informed about the behavior of other vehicles by the communication with each other and therefore making the best decisions.
  In addition to the advantage of connected communication, autonomous vehicles would not suffer from humanly issues, such as getting distracted, being drunk, running red lights or texting while driving. so there is already potential for increased safety over human drivers.

- Lifestyle
  By use of a self-driving connected vehicle, passenger can effectively avoid traffic jam and congestion. Reasons for this are the lower number of traffic accidents and the more accurate real-time traffic reports.
  In addition, self-driving vehicles can help people who are not allowed to drive to enjoy the convenient mobility.
  The use of self-driving vehicles will also allow passenger to use the travel time productively. Passengers could read the morning paper or prepare the upcoming meeting on the way to work. During the travel time, the vehicles could also used as entertainment or communication centers on request.
- Economics
  Self-driving vehicles have an enormous potential to reduce emission and optimize the energy consumption. Connected autonomous vehicles are informed of the traffic conditions and they can make optimized decisions about actions, like braking and accelerating, regarding energy consumption.

In summary, intelligent unmanned vehicles have broad applications prospects and development space in many fields. Nowadays more and more universities and research institutes are investing a lot of capital and resources to carry out the research work on IUVs. With the rapid development of computer science, sensor system, automatic control and localization and positioning technology, the research on intelligent unmanned vehicles will continue to move towards feasibility and practicability. The related research results will make a great contribution to the world and civilization.

## 1.2 Objective of the Thesis

The System for Teleoperated Road Marking (SToRM) is a project of the Chair pf Automotive Technology (FTM) at TUM. The project SToRM aims to develop a road marking machine, which uses deep learning algorithms[2] to detect the lane markings and a lateral control algorithm to let the vehicle to drive itself along the lane. The road marking machine can print the road lane marking autonomously, which saves effort and time of road-workers.

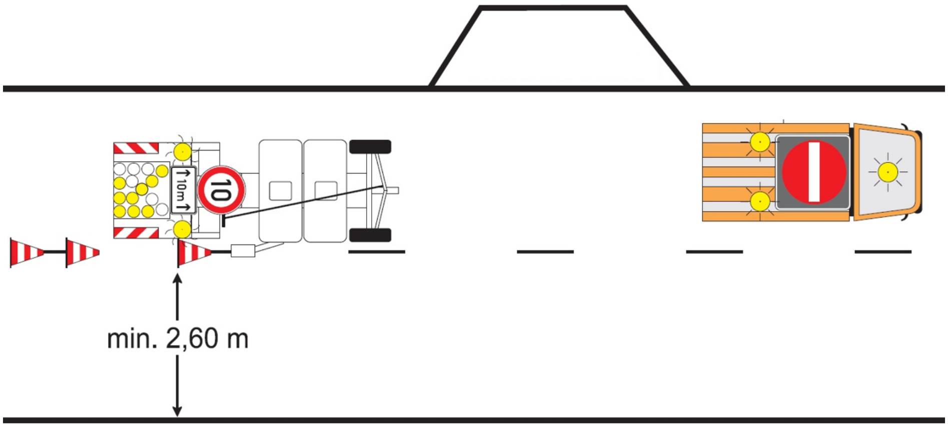


Figure 1.1: Lane marking machine in application scenario

The primary objective of this thesis is the development of lateral control algorithms which compute the steering angle automatically. The road marking machine moves slowly, so there is no need to build a dynamic model for it. The details of the kinematic model of road marking machine are in section 3.1. However, the road marking machine has to move along the lane accurately to print the lane marking. Therefore, the accuracy of the control algorithm is of significant importance in this work.

## 1.3 Structure of the Thesis

The first chapter gives a short introduction to the motivation of this work. After that, the objective and structure of the thesis are presented.

The second chapter begins with a description of the state of the art relevant to the objective of this thesis which includes an introduction into autonomous driving and its components. The project SToRM is also introduced in this chapter. Then, many state-of-the-art path following control methods are presented. In addition to the principles of control algorithms, some research states of these methods are also given.

Next, the vehicle model used in this work is introduced. The vehicle model is a nonlinear system. Therefore, the kinematic model is linearized and discussed. At the end of this chapter, the operating system and the software applied in the project are introduced.

Explaining of the simulation environment and the background are part of the fourth chapter. Next, the implementation of control methods is also presented. A method for combined controller which is designed to increase the accuracy of the control is described in the last section.

Lastly, in the final chapter, a discussion about this work, a summary and potential improvements for further works regarding the presented lateral control approach are given.

The described structure of this thesis is illustrated in figure 1.2.

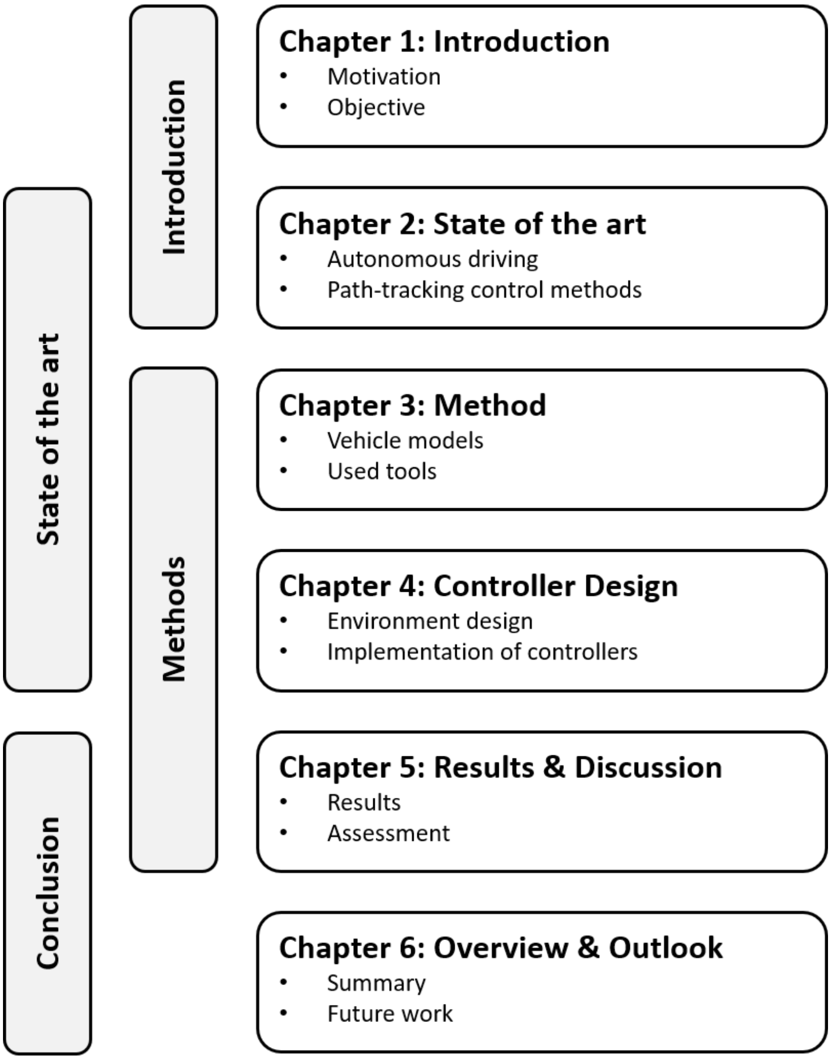


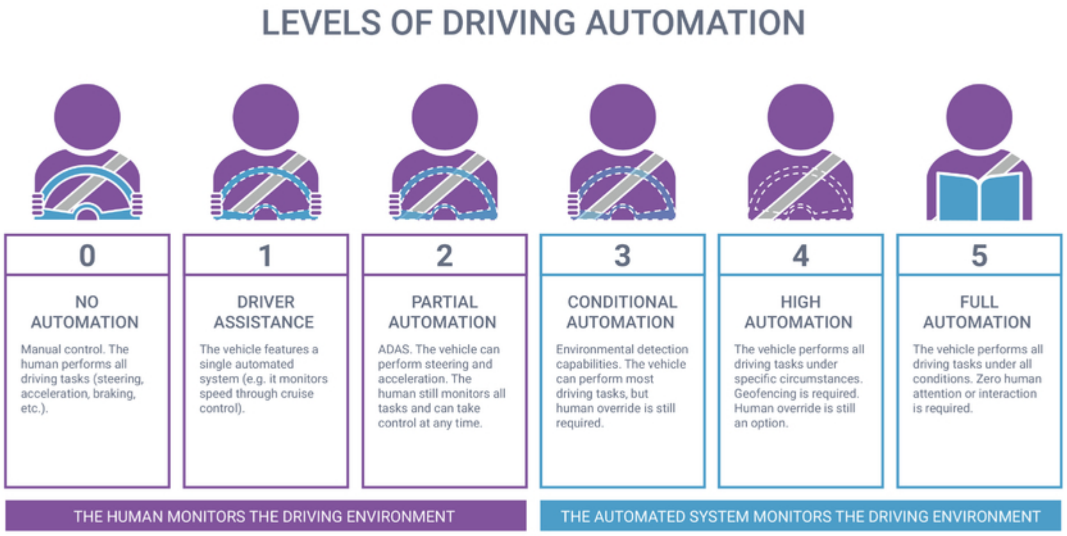
Figure 1.2: Structure of this thesis

# 2 State of the Art

Intelligent unmanned vehicles are the significant component of future intelligent transport systems and the path tracking control occupies an important position in autonomous driving technologies. This section examines the hardware and software module, which are fundamental for autonomous driving. Next, a variety of state-of-the-art lateral control methods are introduced.

## 2.1 Autonomous Driving

In recent years, there has been increasing interest in technology of autonomous driving. Nowadays, the different categories of autonomous vehicles are reflected in the six levels of automation pictured in Figure 2.1. In the case of semi-automatic vehicles (level 3 and level 4), the system assumes the task of longitudinal and lateral control. However, the human driver must supervise the system and always be prepared to completely take over the vehicle control. On the other hand, in a fully-automated vehicle, the human driver is not required to supervise the system and is allowed to perform additional activities[3].



LEVELS OF DRIVING AUTOMATION

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| NO AUTOMATION | DRIVER ASSISTANCE | PARTIAL AUTOMATION | CONDITIONAL AUTOMATION | HIGH AUTOMATION | FULL AUTOMATION |
| Manual control. The human performs all driving tasks (steering, acceleration, braking, etc.). | The vehicle features a single automated system (e.g. it monitors speed through cruise control). | ADAS. The vehicle can perform steering and acceleration. The human still monitors all tasks and can take control at any time. | Environmental detection capabilities. The vehicle can perform most driving tasks, but human override is still required. | The vehicle performs all driving tasks under specific circumstances. Geofencing is required. Human override is still an option. | The vehicle performs all driving tasks under all conditions. Zero human attention or interaction is required. |

THE HUMAN MONITORS THE DRIVING ENVIRONMENT

THE AUTOMATED SYSTEM MONITORS THE DRIVING ENVIRONMENT

Figure 2.1: The six levels of vehicle autonomy[4]

The development of autonomous driving technology has been accelerated in the past years primarily through the Defense Advanced Research Projects Agency (DARPA) challenge. The goal of this competition to let the vehicle move autonomously to the destination without any human operation.

## Research in Germany

The research of autonomous driving began in Germany in the 1980s. The Bundeswehr University Munich developed the first car-like robot VaMoRs, which can move autonomously in a closed route. After that, the Bundeswehr University Munich worked together with Daimler-Benz to start the new project PROMETHEUS and to research on the feasibility of autonomous driving. The results of the project show, the two test vehicles Vita II and VaMP are already able to move autonomously from Paris to Lille at high speed and with many lane changes[5]. Nowadays, the Bundeswehr University Munich is researching on locating the autonomous robots in buildings. This research is also very important for the development of autonomous driving.

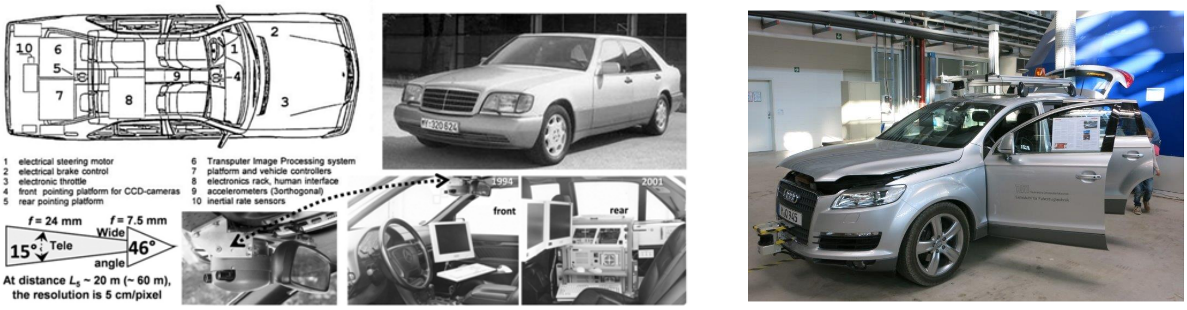

1 electrical steering motor
2 electrical brake control
3 electronic throttle
4 front pointing platform for CCD-cameras
5 rear pointing platform
6 Transputer Image Processing system
7 platform and vehicle controllers
8 electronics rack, human interface
9 accelerometers (3orthogonal)
10 inertial rate sensors
f = 24 mm
f = 7.5 mm
15°
Tele
Wide
angle
46°
At distance L5 ~ 20 m (~ 60 m), the resolution is 5 cm/pixel
1994
front
2001
rear

Figure 2.2: VaMP and Audi Q7[6]

The FTM at Technical University of Munich developed a test vehicle for the special research field — human-like cognitive perception. The test vehicle is based on an Audi Q7, which is equipped with a variety of sensors like lidar, radar and camera. The cooperative ability of vehicles was demonstrated using vehicle-vehicle communication. It is worth mentioning that the Q7 is also the test vehicle in this work.

## Research in the World

In the USA, Carnegie Mellon University (CMU) in Pittsburgh researches in the autonomous driving for decades. The test vehicle Sandstorm of CMU traveled autonomously the longest distance over all participants at the DARPA challenge 2004. CMU won the Urban challenge 2007 with test vehicle Boss, which is based on a Chevy Tahoe.

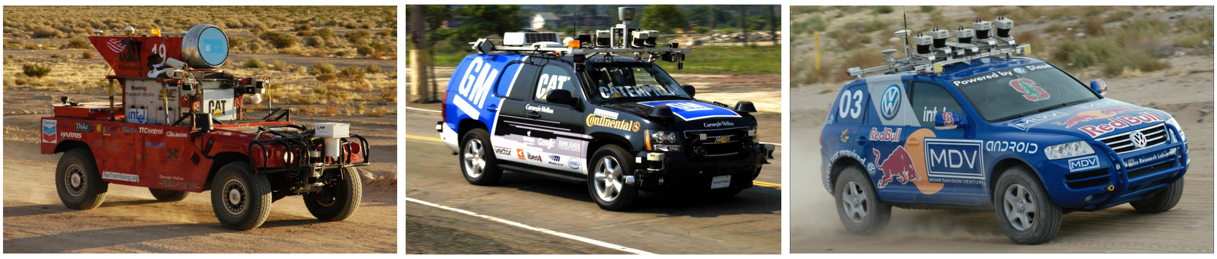


Figure 2.3: Sandstorm, Boss and Stanley[7]

The Stanford University plays has an enormous result in autonomous driving. At DARPA challenge 2005, the Stanford University won the competition with their test vehicle Stanley. The path following algorithm of Stanley is also implemented and applied in this work[6].

## 2.2 Modules of Autonomous Vehicles

This section has been divided into two parts. The first part deals with the hardware modules of IUVs and the second part of this section examines the software modules, which have received considerable critical attention over these years.

### 2.2.1 Hardware Module

The sensors of IUVs mainly include Radar, Lidar, Camera and Ultrasound. Figure 2.4 shows the different functions and range of various sensors.

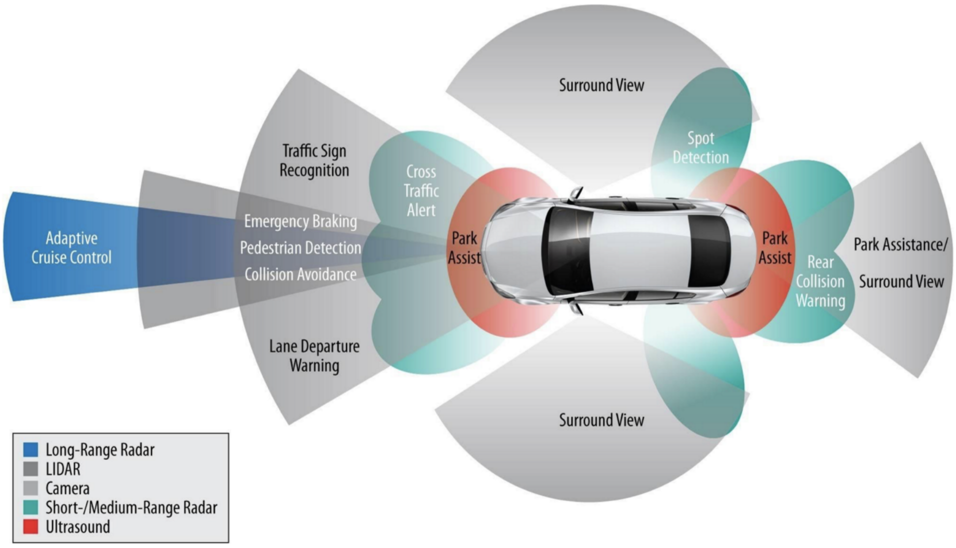


Figure 2.4: The sensors of an IUV[8]

However, each sensor has its own advantages and suitable scenarios. Table 2.1 shows the performance of these commonly used sensors in different applications scenarios. An IUV usually combines the advantages of different sensors with sensor data fusion to optimize its performance.

Table 2.1: Comparison and Evaluation of sensor system

| Sensor | Radar | Lidar | Mono Camera | Stereo Camera |
|---|---|---|---|---|
| Range | ++ | 0 | 0 | 0 |
| Opening angle | 0 | ++ | + | + |
| Accuracy of distance | + | ++ | 0 | + |
| Accuracy of velocity | ++ | / | / | / |
| Width of object | / | ++ | ++ | ++ |
| Length of object | 0 | + | + | + |
| Height of object | / | / | + | + |

In this work, because there is no need to measure the distance to other objects in the application scenario, only a camera is used to detect the line. After the lane detection, the process module in section 2.2.2 calculates the curvature and other information of the path. The control module in section 2.3 computes the steering angle in real time to let the vehicle move autonomously along the path.

### 2.2.2 Software Module

An autonomous vehicle is a highly complex integrated system, which involves many high-tech hardware and software modules. Just like a human driver, an autonomous vehicle uses different sensors to perceive other objects on the street. After data sensor fusion, a reference trajectory will be planned. The control module is responsible for the control of the throttle, brake and steering angle and performs tracking control according to the path, curvature and other sensor information. Figure 2.5 shows the software module structure of an autonomous vehicle.

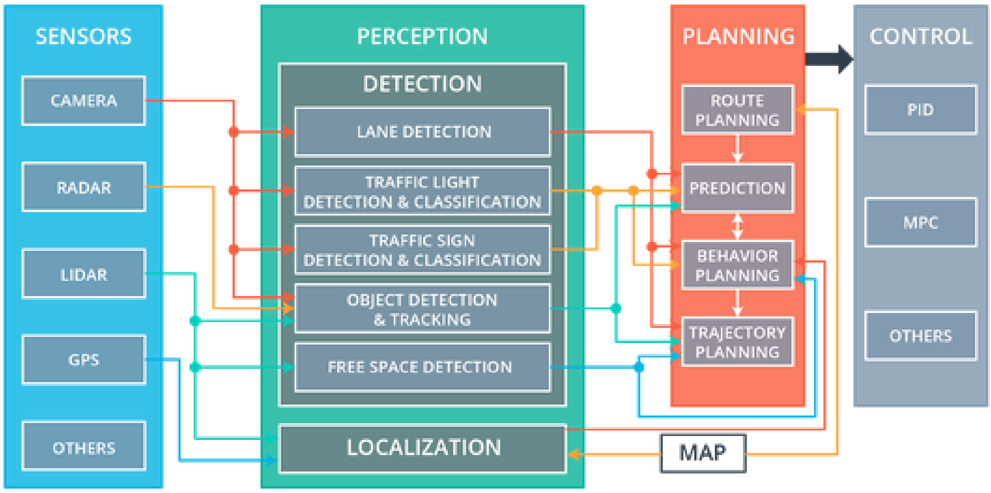


Figure 2.5: The software architecture of an autonomous vehicle[9]

### Perception and Localization

The perception layer is the sensory system of an IUV, just like human eyes and ears. Environmental perception and localization are the basic premises for IUVs. The accuracy of the environmental perception will directly affect the operation of the IUV. In a ITS, an IUV has to sense the environment around the vehicle through sensors, and then analyze and process the sensor data to extract the vehicle environmental and status information.

### Planning and Decision

The planning and decision layer is the brain of an IUV. By use of provided information from the perception layer, the planning and decision layer plans a rational path after calculation and gives the lower layer a control signal. This layer consists of two parts, they are path planning and decision module. The path planning chooses the rational path while the decision module determines how to drive along the path.

### Motion Control

Motion control is the actuator of an IUV. Details of the state-of-the-art motion control are introduced in the subsection 2.3.

## 2.3 Path-Tracking Control Method

As a key issue in the research of IUV, automatic control has an enormous research value. The main task of an automatic control module is the generation of control signals from vehicle position, vehicle pose from the planning module to let the vehicle move autonomously and track desired path quickly and accurately. Commonly, the vehicle motion control can be divided into lateral control and longitudinal control. Lateral control is the control of the vehicle steering system, performed by controlling the steering angle of the IUV to achieve the tracking of the desired path.

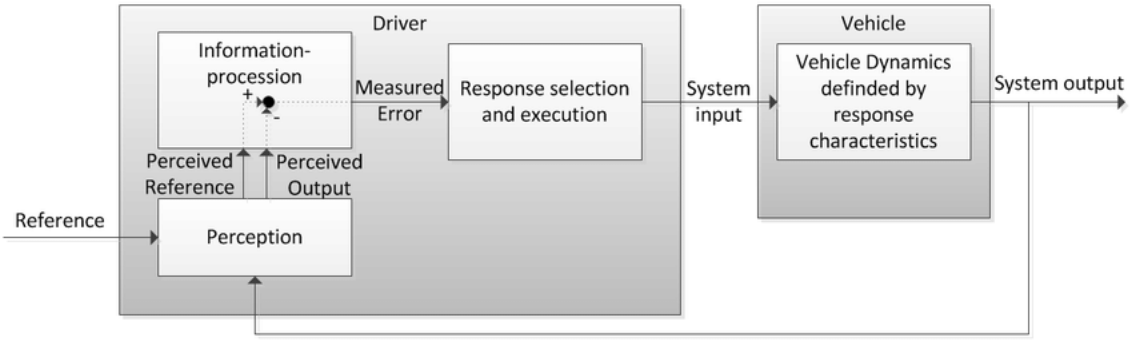


Figure 2.6: A driver-vehicle-environment control loop [10]

Figure 2.6 shows a driver-vehicle-environment control loop. Just like the driver model, autonomous driving model in Figure 2.7 also has a perception and an information procession module. A perception module helps autonomous vehicle to measure its pose and generate trajectories. According to the reference trajectory, a lateral controller can calculate a steering angle, which affects vehicle pose and position.

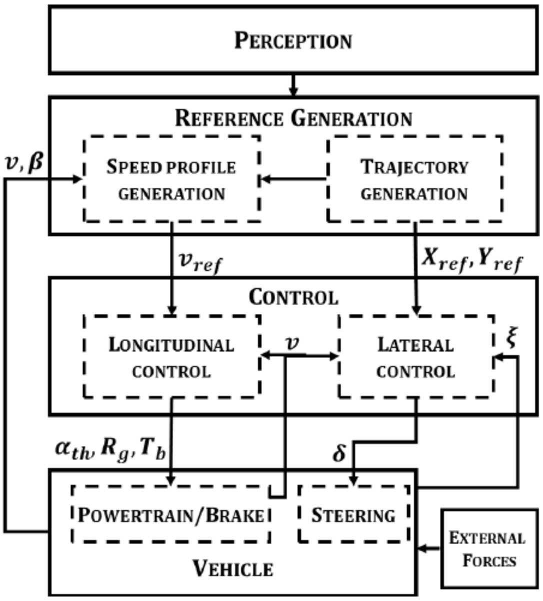


Figure 2.7: A combined lateral and longitudinal control loop[11]

The path tracking controller can be roughly separated into 3 categories: controllers based on the feedback, the geometric controllers and controllers based on models.

### 2.3.1 Feedback-based Control

Model-free control is a traditional control algorithm. The model-free controller does not take the characteristics of the vehicle model into consideration. For this reason, the model-free algorithm is less robust compared to the model-based control and cannot perform effective tracking control at high speeds.

#### PID Control

Proportion Integration Differentiation (PID) controller is a widely used linear model-free controller in engineering and industry.

As shown in the block diagram in the figure 2.8, a PID controller uses lateral error as input, which is calculated from the measurements of vehicle position and the nearest point on the desired path. The control value is the steering angle, which has a significant influence on vehicle position.

The algorithm of the PID control is uncomplicated, an advantage is that this control method does not need a vehicle model. The control parameters can be obtained via trial and error, but need significant trail work and are tremendously sensitive to the change of vehicle parameters. Above all, a PID controller is easy to implement and has great robustness[12].

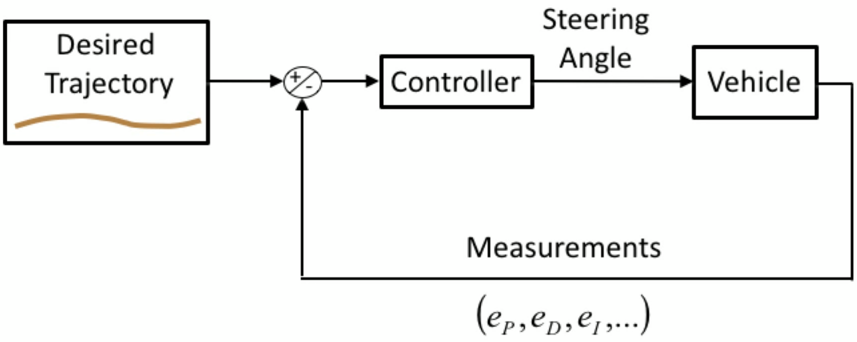


Figure 2.8: A PID block diagram of automatic steering system[13]

A PID controller calculates an error value $e(t)$ as the difference between a desired Set Point (SP) and a measured Process Variable (PV) and allies a correction based on proportional, integral and derivative terms. As shown in the block diagram 2.9, the controller attempts to minimize the error by adjustment of a control variable $u(t)$ to a new value determined by a weighted sum of the control terms.

The overall control function is

$$u(t) = k_p e(t) + k_i \int_0^t e(\tau)\,\mathrm{d}\tau + K_d \frac{\mathrm{d}e(t)}{\mathrm{d}t}, \tag{2.1}$$

where $e(t)$ is the difference between the desired setpoint and the measured process variable. For the steering control, $e(t)$ can be the lateral distance between vehicle rear axle and desired path point or the angle between the heading of the vehicle and the heading of the desired path.

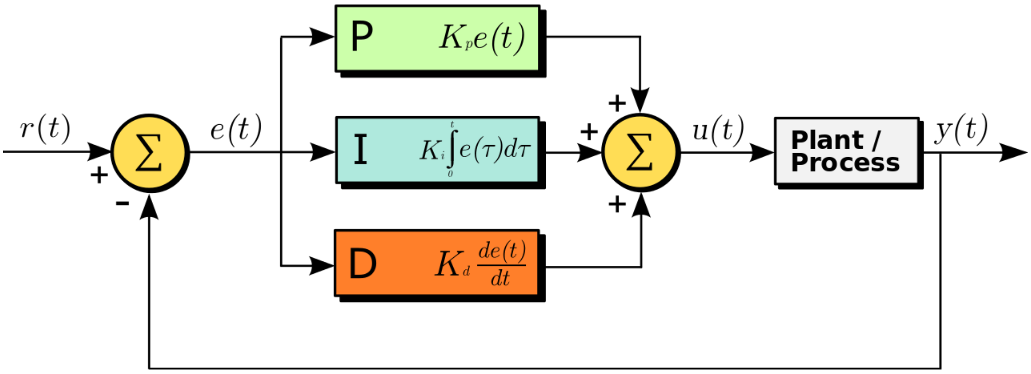


Figure 2.9: A block diagram of a PID controller in a feedback loop[14]

In the equation 5.2 and figure 2.9, the three terms have their own specification and use:

- P: outputs a value that is in direct linear proportion to the size of the error value $e(t)$. A great proportional gain factor $k_p$ causes a rapid response. However, it requires an error to generate the proportional response. If there is no error, there is no corrective response.
- I: accounts for past values of error value $e(t)$ and integrates them over time to produce I term. If there is a residual error after the application of P control, the I term seeks to eliminate the residual error by adding a control effect due to the historic cumulative value of the error.
- D: estimates of the future trend of error value $e(t)$, based on its current rate of change. It is effectively seeking to reduce the effect of the error by exerting a control influence generated by the rate of error change. The more rapid the change, the greater the controlling or dampening effect.

Furthermore, there are nowadays many state-of-the-art methods, which are applied in the traditional PID control.

In 2009, Riccardo Marino et al.[15] from the University of Rome Tor Vergata designed a nested PID steering control in vision based autonomous vehicles. The external control loop computes the yaw rate reference, using a PID control with a double integral action based on the lateral offset to filter out the disturbances on the curvature. Meanwhile, the internal control loop performs the steering control based on the yaw rate tracking error to improve the vehicle steering dynamics. An advantage of the nested PID steering control is reduction of cost. Once there is no need to measure the lateral velocity and acceleration, the cost of the relevant sensors can be saved.

In 2014, Behrooz Mashadi el al.[13] from the Iran University of Science and Technology created a driver model to apply the adaptive PID control method. They used the genetic algorithm to optimize the control parameters intelligently. The experiments proved that this driver model with adaptive PID controller has a extremely strong robustness under different road conditions.

## Stanley Control

The Stanley method, also called front wheel feedback, is the path tracking approach used by Stanford University's autonomous vehicle entry in the DARPA Grand Challenge, Stanley[16]. The Stanley controller is a complex control system, consisting of a feed-forward control based on path curvature and a PID control. Figure 2.10 illustrates the geometric relationship of the control parameters.

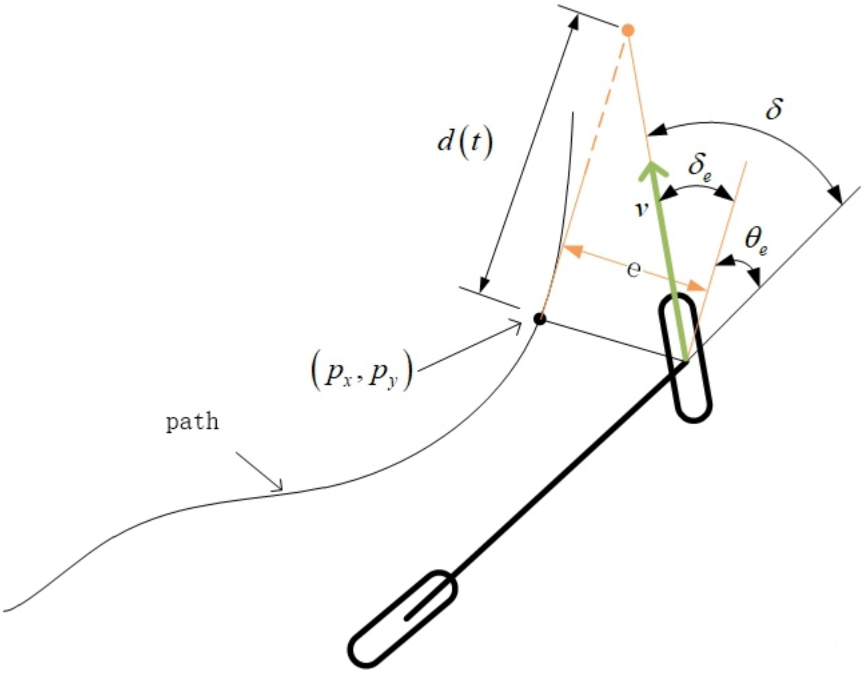


Figure 2.10: Stanley method geometry[17]

The Stanley method is a nonlinear feedback function of the cross track error $e$, measured from the center of the front axle to the nearest path point $(p_x, p_y)$ and boosts exponential convergence. Co-locating the point of control with the steered front wheel allows for an intuitive control law, where the first term simply keeps the wheels aligned with the given path by correctly the steering angle $\delta$ with the heading error

$$\theta_e = \theta - \theta_p, \tag{2.2}$$

where $\theta$ is the heading of the vehicle and $\theta_p$ is the heading of the path at $(p_x, p_y)$. The resulting steering control law is given as

$$\delta(t) = \theta_e(t) + \tan^{-1}\left(\frac{ke(t)}{v_x(t)}\right), \tag{2.3}$$

where k is a gain parameter. When $e$ is non-zero, the second term adjusts $\delta$ such that the intended trajectory intersects the path tangent from $(p_x, p_y)$ of $kv(t)$ units from the front axle. It is clear that the desired effect is achieved with this control law: As $e$ increases, the wheels are steering greater towards the path.

Above all, Stanley method is a nonlinear PID control method which represents the steering angle of the front axle with a feedback function using lateral deviation. This nonlinear controller can converge lateral deviation quickly and smoothly[18].

### 2.3.2 Geometry-based Control

The geometric controller makes use of the geometric relationship between the vehicle model and the tracked path to calculate the front wheel's steering angle, which is the most popular method for the practical autonomous vehicles because of its simple implementation and real-time calculations.[19]

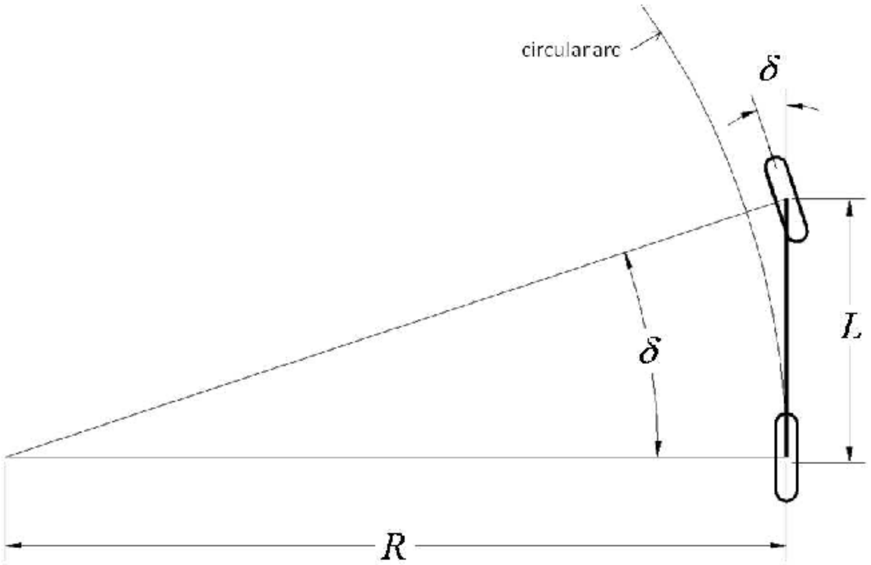


Figure 2.11: Geometric Single-Track Model[17]

A common simplification of an Ackerman steered vehicle used for geometric path tracking is the bicycle model[17]. Section 3.1.1 includes a detailed discussion of Ackerman steering and the kinematics of the single-track model. Figure 2.11 shows the simple geometric relationship:

$$\tan(\delta) = \frac{L}{R}, \tag{2.4}$$

where $\delta$ is the steering angle of the front wheel, $L$ is the distance between the front axle and rear axle (wheelbase) and $R$ is the radius of the circle that the rear axle will travel along at the given steering angle. This geometric model approximates the motion of a car reasonably well at low speeds and moderate steering angles.

### Pure Pursuit Control

The first geometric controller used in the autonomous vehicle is Pure Pursuit. Pure Pursuit was first employed to control a vehicle to follow the road center[20]. The final results show that the Pure Pursuit controller could control the vehicle to track the lane center accurately. However, its performance wholly depends on the selection of the look-ahead distance $L_d$.

The Pure Pursuit method consists of geometrically calculating the curvature of a circular arc that connects the rear axle location to a goal point on the path ahead of the vehicle.[17] The goal point is determined at a look-ahead distance $L_d$ from the current rear axle position to the desired path. The goal point $(g_x, g_y)$ is illustrated in Figure 2.12. The vehicle's steering angle $\delta$ can be determined using only the goal point location and the angle $\alpha$ between the vehicle's heading vector and the look-ahead vector.

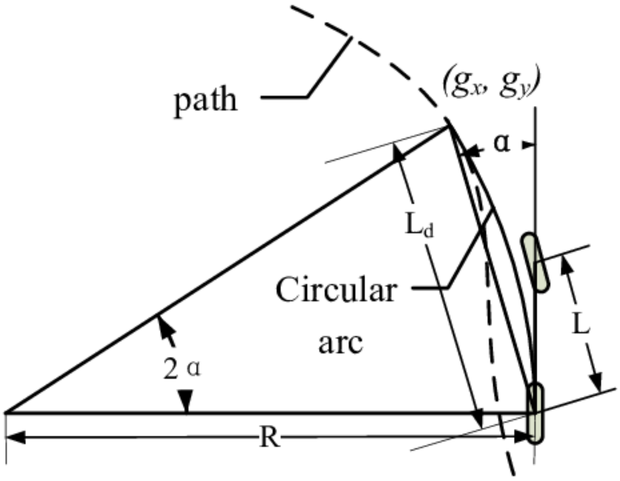


Figure 2.12: Pure Pursuit Method Geometry[17]

The curvature of the circle is given by

$$\kappa = \frac{2\sin(\alpha)}{L_d}. \tag{2.5}$$

For a vehicle speed $v_r$, the commanded heading rate is

$$\omega = \frac{2v_r\sin(\alpha)}{L_d}. \tag{2.6}$$

If the angle $\alpha$ cannot be directly computed from a camera, it can be expressed in terms of the inertial coordinate system. Consider the configuration $(x_r, y_r, \theta)^T$ and the points on the path, $(g_x.g_y)$, such that $||(g_x, g_y) - (x_r, y_r)|| = L$[21]. The $\alpha$ is given by

$$\alpha = \arctan(\frac{g_y - y_r}{g_x - x_r}) - \theta. \tag{2.7}$$

By use of the geometric single-track model of an Ackerman steered vehicle, the steering angle can be written as

$$\delta = \tan^{-1}(\kappa L). \tag{2.8}$$

Using Eq.2.4 and 2.6, the steering angle is given as

$$\delta(t) = \tan^{-1}(\frac{2L\sin(\alpha(t)}{L_d}), \tag{2.9}$$

$L$ is the wheel base and $L_d$ is the look-ahead distance.

In 2003, Rajamani et al.[22] developed a look-ahead based input output feedback linearization lateral control method for the nonlinear geometric model of vehicles. The results show that the lateral control strategy based on look-ahead can efficiently improve the tracking performance of a vehicle.

### 2.3.3 Model-based Control

The model-based controller is designed to consider a kinematic or dynamic model. Some controllers can take both of them into consideration. Simplifying the vehicle system model to a kinematic bicycle model is a common approximation used for robot motion planning, simple vehicle analysis and (for the geometric methods) deriving intuitive control laws. This section presents the control methods using kinematic and/or dynamic bicycle models. The single-track model and its state-space representation will be introduced in section 3.1.

#### Optimal Control

The theory of optimal control is aimed to operate a dynamic system at minimum cost. One of the main aspects of the method is that the solution is provided by the Linear-Quadratic Regulator (LQR). The LQR is an important solution to the Linear-Quadratic-Gaussian (LQG) problem.

The cost function is often defined as the sum of the derivation of key measurements, like the lateral distance of the vehicle from their desired value. The cost function can be expressed as

$$J = \sum_{\tau=0}^{N-1} (x_\tau^T Q x_\tau + u_\tau^T R u_\tau + 2 x_\tau^T N u_\tau) + x_N^T Q_f x_N, \tag{2.10}$$

$Q$ is a diagonal weighting matrix with an entry for each state corresponding to the performance aspects contributing to the cost function and $R$ is the weighting value corresponding to the control effort contributing to the cost function.

The optimal control sequence minimizing the performance index is given by

$$u_\tau = -F_\tau x_\tau \tag{2.11}$$

with

$$F_\tau = (R + B^T P_{\tau+1} B)^{-1} (B^T P_{\tau+1} A + N^T). \tag{2.12}$$

$P_{\tau-1}$ is determined iteratively backwards in time by the dynamic Riccati equation

$$P_{\tau-1} = A^T P_\tau A - (A^T P_\tau B + N)(R + B^T P_\tau B)^{-1} (B^T p_\tau A + N^T) + Q. \tag{2.13}$$

The solution to the dynamic Riccati equation is omitted here since it is not specific to path tracking[17]. There are also many software packages available to perform the task.

LQR is an optimal control regulator that better tracks a reference trajectory compared to the traditional controllers such as PID.

In 2013, Codeiro et al.[23] used bio-inspired LQR approach to let an off-road robotic vehicle to track the desired path. The results showed, a bio-inspired reference shaping approach can enhance the lateral tracking performance efficiently.

## Model Predictive Control

Model Predictive Control (MPC) is an advanced method of process control that is used to control a process while satisfying a set of constraints. The main advantage of MPC is the fact that it allows the current timeslot to be optimized while keeping future timeslots in account. MPC has also the ability to anticipate future events and can take control actions accordingly.

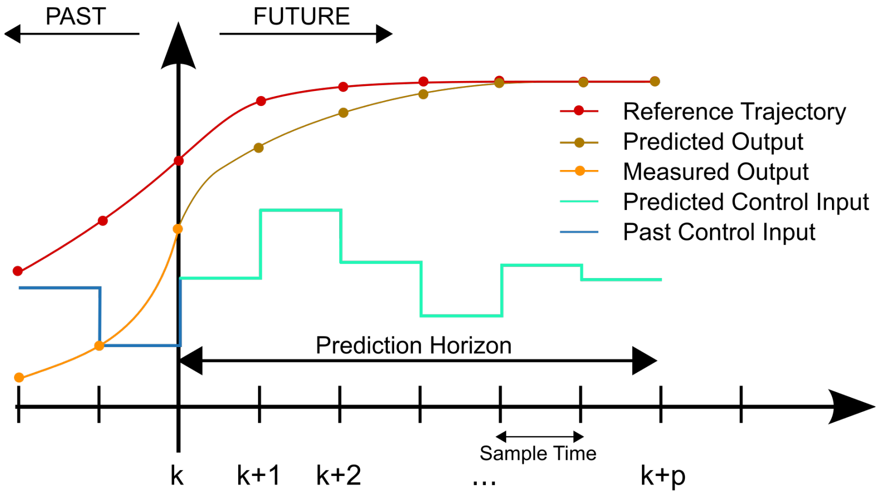


Figure 2.13: A discrete model predictive control scheme[24]

Figure 2.13 shows a scheme of the theory behind MPC. MPC is based on iterative, finite-horizon optimization of a plant model. At time $t$ the current plant state is sampled and a cost minimizing control strategy is computed for a relatively short time horizon in the future: $[t, t+T]$. Specifically, an online calculation is used to explore state trajectories that emanate from the current state and find a cost-minimizing control strategy until time $t+T$.

MPC is nowadays the most advanced control method. It is also applied by many autonomous driving research institutes and companies. Baidu (a company from China) open sourced the control module of their autonomous vehicle Apollo in GitHub[25]. The control module includes MPC and is implemented in C++.

The main differences between MPC and LQR are that LQR optimizes in a fixed time window (horizon) whereas MPC optimizes in a receding time window and that a new solution is computed often whereas LQR uses the single (optimal) solution for the whole time horizon. Therefore, MPC typically solves the optimization problem in smaller time windows than the whole horizon and hence may obtain a suboptimal solution. In 2017, Jezierski el al.[26] performed experiments to compare LQR and MPC algorithms of a inverted pendulum. The results show that the LQR algorithm is performing better in the control task with the disturbance rejection, while the MPC controller gives better results in the case of the trajectory tracking task.

In 2016, Zhijun Li et al.[27] used a MPC scheme in corporating neural-dynamic optimization to achieve trajectory tracking of nonholonomic mobile robots. Compared to the existing MPC, which requires repeatedly calculating the Hessian matrix of the Langragian and then solves a quadratic program. The results show, the computation complexity reaches $O(n^3)$, while the proposed neural-dynamic optimization contains $O(n^2)$ operations.

Additionally, there are a variety of other modern lateral control methods.

## Adaptive Control

In 2004, Netto, Chaib and Mammar[28] applied the self-tuning regulator to solve the vehicle lateral control problem. The dynamic singel-track model takes unknown wind forces and road curvature into consideration. The simulations, which are carried out with a nonlinear model, confirm the efficacy of the controller and its robustness.

In 2007, Montemerlo el al.[29] from Stanford University developed adaptive control method, which has integrated learning module to tune the control parameter. They applied the control algorithm in the test vehicle Junior. It won the 2.Prize in the DARPA challege in 2007.

## Sliding-Mode Control

In 2017, Lei Tan et al.[30] designed a sliding-mode controller for the active four-wheel steering system, which both side slip angle and yaw rate can trace the given reference trajectories. Simulation results show that, compared to the front wheel steering and active four-wheel steering vehicle based on linear quadratic control law, the proposed sliding-mode control strategy can effectively deal with disturbances, and improve handling stability of the vehicle.

## Fuzzy Control

In 2006, Mohamed et al.[31] used fuzzy logic control system to steer autonomously a two-axle vehicle. Two fuzzy modules in the steering controller are designed to meet the basic driving requirement: Target Steering Fuzzy Module (TSF) and Collsion Avoidance Steering Fuzzy Module (CASF). The first module steers the vehicle toward the target by monitoring the steering angle and the angle to the target. The objective of the second module is to avoid collisions with static and dynamic obstacles.

# 3 Methods

In this chapter, all the relevant methods and software will be introduced, which play a significant role to the development and testing of the lateral controller. In the section 3.1, the used vehicle model will be introduced. There are two vehicle models, namely kinematic bicycle model and dynamic bicycle model. In the section 3.2,

## 3.1 Vehicle Model

In this section, some commonly used models of mobility of car-like robots will be introduced. Such models are widely use in motion planning and control method to approximate a vehicle's behavior in response to control actions in relevant operating conditions[21]. A high-precision model may accurately reflect the response of the vehicle, but the added detail may lead to the algorithm problems of planning and control. This is a conflict between the accuracy of the selected vehicle model and the difficulty of the algorithm problems. This section provides an overview of general modeling, namely kinematic bicycle model and dynamic bicycle model.

### 3.1.1 Kinematic Single-Track Model

In the most basic model of practical use, a car consists of two wheels connected by a rigid link and is restricted to move in a plane. It is assumed that the wheels don't slip at the contact point, but rotate freely around the axes of rotation. The front wheel in Figure3.1 has a degree of freedom, which allows the front wheel to rotate around an axis normal to the plane of motion[32]. This is to model steering.

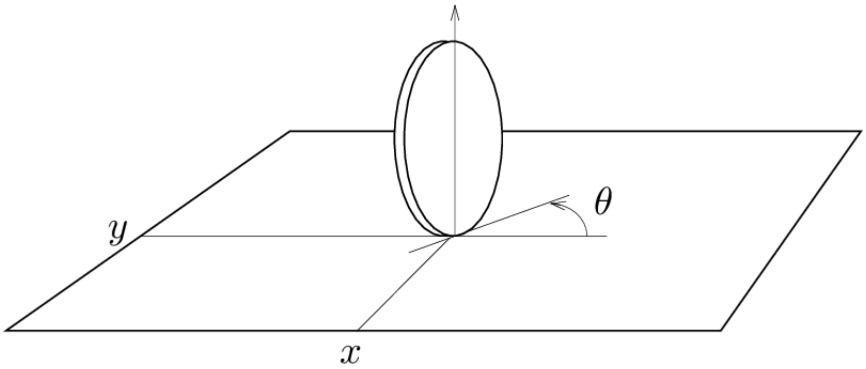


Figure 3.1: Front wheel of a car-like robot[17]

These two modeling features reflect the experience, what the most passengers have: the car is unable to move laterally without simultaneously moving forward. More formally, the limitation on maneuverability is discussed as a nonholonomic constraint[33]. The nonholonomic constraint is expressed as a differential constraint on the motion of the car.

The following is a derivation of the differential constraint in Cartesian coordinate system for the configuration. In reference to Figure3.2, the vectors $p_f$ and $p_r$ represent the location of the rear and front wheels in a stationary or inertial coordinate system with basis vectors $(\hat{e}_x, \hat{e}_y, \hat{e}_z)$.

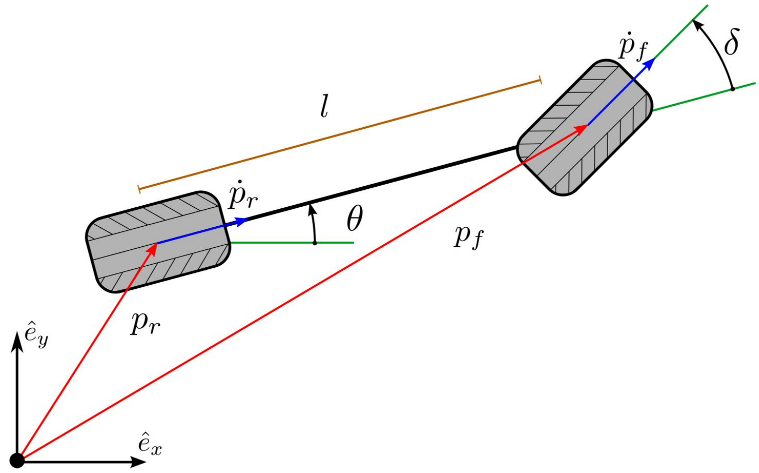


Figure 3.2: A kinematic single track model[21]

$p_f$ and $p_r$ are the contact points of the rear wheel an front wheel on the ground. $\theta$ is the angle which describes the direction that the vehicle is facing, also called vehicle heading. This is defined as the angle between $\hat{e}_x$ and $p_f$-$p_r$. Blue arrows represent the time derivatives of $p_f$ and $p_r$, are also the direction of nonholonomic constraint.

Defferential constraints will be derived for the coordinate systems from the angle $\theta$, together with the motion of one of the points $p_f$ as in [16] or $p_r$ as in [34].

To satisfy the no-slip assumption, the motion of the points $p_f$ and $p_r$ must be colinear with the vehicle orientation. Expressed as an equation, this constraint on the rear wheel is

$$(\dot{p}_r \cdot \hat{e}_y)\cos(\theta) - (\dot{p}_r \cdot \hat{e}_x)\sin(\theta) = 0, \tag{3.1}$$

and for the front wheel:

$$(\dot{p}_f \cdot \hat{e}_y)\cos(\theta+\delta) - (\dot{p}_f \cdot \hat{e}_x)\sin(\theta+\delta) = 0. \tag{3.2}$$

The constraint between front wheel and rear wheel can be expressed in

$$\begin{aligned} x_f &= x_r + l\cos(\theta), \\ y_f &= y_r + l\sin(\theta). \end{aligned} \tag{3.3}$$

The equation 3.2 can be rewritten as

$$\begin{aligned} 0 &= \frac{d(y_r + l\sin(\theta))}{dt}\cos(\theta+\delta) - \frac{d(x_r + l\cos(\theta))}{dt}\sin(\theta+\delta) \\ &= (\dot{y}_r + l\dot{\theta}\cos(\theta))\cos(\theta+\delta) - (\dot{x}_r - l\dot{\theta}\sin(\theta))\sin(\theta+\delta) \\ &= \dot{y}_r\cos(\theta+\delta) + l\dot{\theta}\cos(\delta) - \dot{x}_r\sin(\theta+\delta) \end{aligned} \tag{3.4}$$

This expression can also be rewritten in terms of the componentwise motion of each point along the basic vectors. The motion of the rear wheel along the $\hat{e}_x$-direction is $x_r := p_r \cdot \hat{e}_x$. Similarly, for $\hat{e}_y$-direction, $y_r := p_r \cdot \hat{e}_y$. The forward speed is $v_r := \dot{p}_r \cdot (p_f - p_r)/||(p_f - p_r)||$, which is the magnitude of $\dot{p}_r$ with the correct sign to indicate forward or reverse driving[21]. In terms of the scalar quantities $x_r$, $y_r$ and $\theta$, the differential constraint is

$$\begin{aligned} \dot{x}_r &= v_r \cos(\theta), \\ \dot{y}_r &= v_r \sin(\theta), \end{aligned} \tag{3.5}$$

after that, the equation 3.4 can be rewritten as

$$\begin{aligned} \dot{\theta} &= \frac{\dot{x}_r \sin(\theta+\delta) - \dot{y}_r \cos(\theta+\delta)}{l \cos\delta} \\ &= \frac{v\cos(\theta)\sin(\theta+\delta) - v\sin(\theta)\cos(\theta+\delta)}{l\cos(\delta)} \\ &= \frac{v\cos(\theta)(\sin(\theta)\cos(\delta)+\cos(\theta)\sin(\delta))}{l\cos(\delta)} \\ &- \frac{v\sin(\theta)(\cos(\theta)\cos(\delta)-\sin(\theta)\sin(\delta))}{l\cos(\delta)} \\ &= \frac{v\sin(\delta)}{l\cos(\delta)} \\ &= \frac{v\tan(\delta)}{l}. \end{aligned} \tag{3.6}$$

Therefore, the kinematic single-track model can be expressed as

$$\begin{cases} \dot{x}_r = v_r \cos(\theta) \\ \dot{y}_r = v_r \sin(\theta) \\ \dot{\theta} = \frac{v_r}{l} \tan(\delta) \end{cases} \tag{3.7}$$

Alternatively, the difficult constraint can be written in terms of the motion of $p_f$,

$$\begin{cases} \dot{x}_f = v_f \cos(\theta+\delta), \\ \dot{y}_f = v_f \sin(\theta+\delta), \\ \dot{\theta} = \frac{v_f}{l} \sin(\delta), \end{cases} \tag{3.8}$$

where the front wheel forward speed $v_f$ is used. The front wheel speed $v_f$ is related to the rear wheel speed by

$$\frac{v_r}{v_f} = \cos(\delta). \tag{3.9}$$

The planning and control problems for this model involve selecting the steering angle $\delta$ with in the mechanical limits of the vehicle $\delta \in [\delta_{min}, \delta_{max}]$, and forward speed $v_r$ within an acceptable range, $v_r \in [v_{min}, v_{max}]$.

A simplification that is sometimes utilized, is to select the heading rate $\omega$ instead of steering angle $\delta$. These quantities are related by

$$\delta = \arctan(\frac{l\omega}{v_r}), \tag{3.10}$$

simplifying the heading dynamics to

$$\dot{\theta} = \omega,\ \omega \in [\frac{v_r}{l}\tan(\delta_{min}), \frac{v_r}{l}\tan(\delta_{max})]. \tag{3.11}$$

In this situation, the model is sometimes referred to as the unicycle model since it can be derived by considering the motion of a single wheel.

The kinematic models are suitable for planning paths at low speeds (like the application scenario in this work) where inertial effects are small in comparison to the limitations on mobility imposed by the no-slip assumption. A major drawback of this model is that it permits instantaneous steering angle changes which can be problematic if the motion planning module generates solutions with such instantaneous changes[21].

In addition to the limit on the steering angle, the steering rate can therefore be limited: $v_\delta \in [\dot{\delta}_{min}, \dot{\delta}_{max}]$. The same problem can arise with the car's speed $v_r$ and can be resolved in the same way. The drawback to this technique is the increased dimension of the model which can complicate motion planning and control problems.

The choice of coordinate system is not limited to using one of the wheel locations as a position coordinate. For models derived using principles from classical mechanics it can be convenient to use the center of mass as the position coordinate as in [35], [36], or the center of oscillation as in [37], [38].

### 3.1.2 Dynamic Single-Track Model

When the acceleration of the vehicle is sufficiently large, the no-slip assumption between tire and ground is invalid. For this reason, the kinematic single-track model is no more suitable and the dynamic single-track model is developed, which considers the inertial effects. As it shows in figure 3.3, $\omega_f$ and $\omega_r$ are relative angular velocities of the wheels with respect to the vehicle.

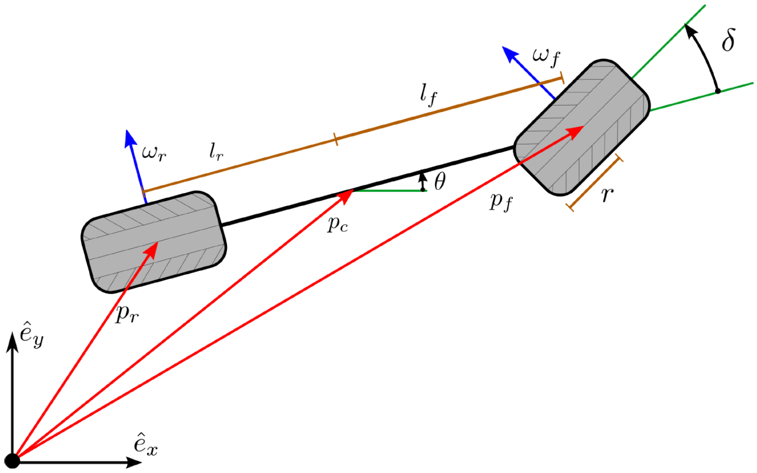


Figure 3.3: A dynamic single-track model[21]

In the dynamic single-track model, the acceleration is proportional to the force generated by the ground on the tires. Taking $p_c$ to be the vehicles center of mass, the motion of the vehicle is governed by

$$m\ddot{p}_c = F_f + F_r, \tag{3.12}$$

and

$$I_{zz}\ddot{\theta} = (p_c - p_f) \times F_f + (p_c - p_r) \times F_r, \tag{3.13}$$

where $F_r$ and $F_f$ are the forces applied to the vehicle by the ground through the ground-tire interaction, m is the total mass of the vehicle and $I_{zz}$ is the polar moment of inertia in the $\hat{e}_z$ direction about the center of mass.

The models discussed in this section are suitable for the motion planning and control tasks in following text. What differs dynamic single-track model from kinematic single-track model is that the no-slip assumption is no more valid, which occurs in steering with high speed. In this work, the vehicles moves with a very low velocity, the inertial effect can therefore be neglected. That means, only the kinematic single-track model will be applied in this work.

### 3.1.3 Kinematic Model Linearization

The kinematic vehicle system is a nonlinear control system. The differential equation of rear wheel from 3.7 can be linearized as

$$\dot{z} = f(z,u) = A^{*}z + B^{*}u \tag{3.14}$$

with the state vector and input vector

$$z = \begin{bmatrix} x \\ y \\ v \\ \theta \end{bmatrix} \quad and \quad u = \begin{bmatrix} a \\ \delta \end{bmatrix}, \tag{3.15}$$

where the inputs are the longitudinal acceleration and the steering angle, outputs are the vehicle position, the vehicle velocity and the yaw angle. The matrix $A^{*}$ can be expressed as

$$\begin{aligned} A^{*} &= \begin{bmatrix} \frac{\partial}{\partial x} v\cos(\theta) & \frac{\partial}{\partial y} v\cos(\theta) & \frac{\partial}{\partial v} v\cos(\theta) & \frac{\partial}{\partial \theta} v\cos(\theta) \\ \frac{\partial}{\partial x} v\sin(\theta) & \frac{\partial}{\partial y} v\sin(\theta) & \frac{\partial}{\partial v} v\sin(\theta) & \frac{\partial}{\partial \theta} v\sin(\theta) \\ \frac{\partial}{\partial x} a & \frac{\partial}{\partial y} a & \frac{\partial}{\partial v} a & \frac{\partial}{\partial \theta} a \\ \frac{\partial}{\partial x} \frac{v\tan(\delta)}{l} & \frac{\partial}{\partial y} \frac{v\tan(\delta)}{l} & \frac{\partial}{\partial v} \frac{v\tan(\delta)}{l} & \frac{\partial}{\partial \theta} \frac{v\tan(\delta)}{l} \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 & \cos(\bar{\theta}) & -v\sin(\bar{\theta}) \\ 0 & 0 & \sin(\bar{\theta}) & v\cos(\bar{\theta}) \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{\tan(\bar{\delta})}{l} & 0 \end{bmatrix} \end{aligned} \tag{3.16}$$

and $B^{*}$ can be expressed as

$$\begin{aligned} B^{*} &= \begin{bmatrix} \frac{\partial}{\partial a} v\cos(\theta) & \frac{\partial}{\partial \delta} v\cos(\theta) \\ \frac{\partial}{\partial a} v\sin(\theta) & \frac{\partial}{\partial a} v\sin(\theta) \\ \frac{\partial}{\partial a} a & \frac{\partial}{\partial a} a \\ \frac{\partial}{\partial a} \frac{v\tan(\delta)}{l} & \frac{\partial}{\partial a} \frac{v\tan(\delta)}{l} \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & \frac{\bar{v}}{l\cos^{2}(\bar{\delta})} \end{bmatrix}. \end{aligned} \tag{3.17}$$

Because the computer can only process the discrete signals, after the Forward Euler discretization with sampling time dt the discrete-time model is

$$z_{k+1} = z_k + f(z_k, u_k)\,\mathrm{d}t. \tag{3.18}$$

Using first degree Taylor expansion around $\bar{z}$ and $\bar{u}$, then the model is

$$\begin{aligned} z_{k+1} &= z_k + (f(\bar{z},\bar{u}) + A^{*}z_k + B^{*}u_k - A^{*}\bar{z} - B^{*}\bar{u})\,\mathrm{d}t \\ z_{k+1} &= (I + dtA^{*})z_k + (dtB^{*})u_k + (f(\bar{z},\bar{u}) - A^{*}\bar{z} - B^{*}\bar{u})\,\mathrm{d}t. \end{aligned} \tag{3.19}$$

The discretized linearized model can be rewritten in normal form as

$$z_{k+1} = Az_k + Bu_k + C, \tag{3.20}$$

where

$$\begin{aligned} A &= (I + \mathrm{d}tA^*) \\ &= \begin{bmatrix} 1 & 0 & \cos(\bar{\theta})\,\mathrm{d}t & -\bar{v}\sin(\bar{\theta})\,\mathrm{d}t \\ 0 & 1 & \sin(\bar{\theta})\,\mathrm{d}t & \bar{v}\cos(\bar{\theta})\,\mathrm{d}t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{\tan(\bar{\delta}))}{l}\,\mathrm{d}t & 1 \end{bmatrix}, \end{aligned} \tag{3.21}$$

$$\begin{aligned} B &= \mathrm{d}tB^* \\ &= \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \mathrm{d}t & 0 \\ 0 & \frac{\bar{v}}{l\cos^2(\delta)}\,\mathrm{d}t \end{bmatrix}, \end{aligned} \tag{3.22}$$

$$\begin{aligned} C &= (f(\bar{z},\bar{u}) - A^*\bar{z} - B^*\bar{u})\,\mathrm{d}t \\ &= dt(\begin{bmatrix} \bar{v}\cos(\bar{\theta}) \\ \bar{v}\sin(\bar{\theta}) \\ \bar{a} \\ \frac{\bar{v}\tan(\bar{\delta})}{l} \end{bmatrix} - \begin{bmatrix} \bar{v}\cos(\bar{\theta}) - \bar{v}\sin(\bar{\theta})\bar{\theta} \\ \bar{v}\sin(\bar{\theta}) + \bar{v}\cos(\bar{\theta})\bar{\theta} \\ 0 \\ \frac{\bar{v}\tan(\bar{\delta})}{l} \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ \bar{a} \\ \frac{\bar{v}\bar{\delta}}{l\cos^2(\bar{\delta})} \end{bmatrix}) \\ &= \begin{bmatrix} \bar{v}\sin(\bar{\theta})\bar{\theta}\,\mathrm{d}t \\ -\bar{v}\cos(\bar{\theta})\bar{\theta}\,\mathrm{d}t \\ 0 \\ -\frac{\bar{v}\bar{\delta}}{l\cos^2(\bar{\delta})}\,\mathrm{d}t \end{bmatrix}. \end{aligned} \tag{3.23}$$

Now the vehicle can be represented as a linearized state-space model with matrices A, B and C. Because it is discrete, it can now also be processed in computer.

## 3.2 Tools

An objective of this section was to introduce the software and method, which are used in this work. The Controller receives data from the camera, after which it can calculate a steering angle and send it to the vehicle. The mechanism of how the units transfer the data is based on a Robot Operating System (ROS). The software in the loop ran in SILAB.

### 3.2.1 ROS

ROS is an Operating System (OS) in concept because it provides all the services such as hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management.

ROS is designed to be a loosely coupled system where a process is called a node and every node should be responsible for one task. Nodes communicate with each other using messages that pass via logical channels called topics. Each node can send or get data from the other node using the publish/subscribe model.

As shown in the figure 3.4, the wheel odometer measures odom and sends it to the path planner. The path planner receives odom and then computes an appropriate velocity. After that, the path planner sends the velocity to the motor controller. The wheel odometer, path planner and motor controller are nodes.

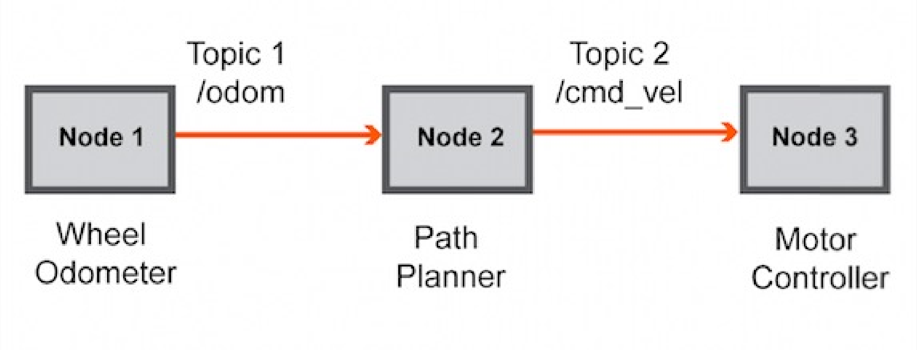


Figure 3.4: Nodes and topics in ROS[39]

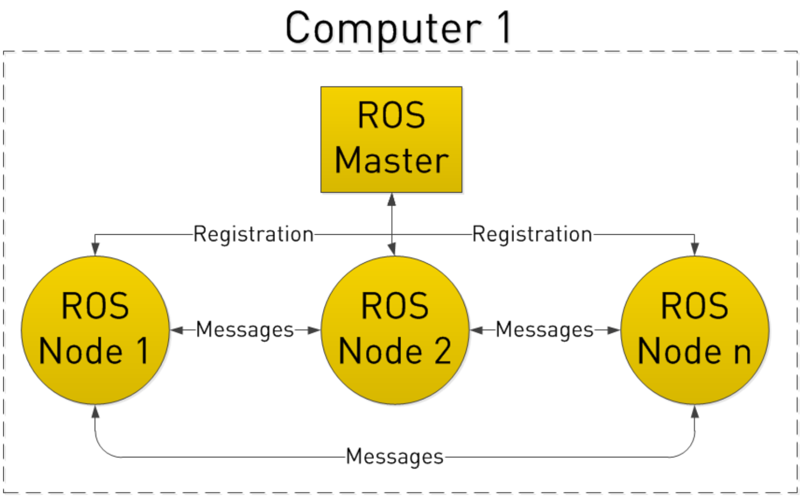


Figure 3.5: Master and nodes in ROS[40]

In order to manage this loosely-coupled environment, there is a Master in ROS which is responsible for name registration and lookup for the rest of the system. Without the Master, nodes would not be able to find each other or exchange messages. The relationships between master, nodes, and messages are described in figure 3.5.

Messages are structures of data filled with pieces of information by nodes. Nodes exchange them using called topics. Then nodes either publish topics or subscribe to them. As the example in the figure 3.9 shows, the camera node state will publish a topic called */image_data*. Both of the other nodes register that they are subscribed to the topic */image_data*.

However, there is a disadvantage of the publish / subscribe model. While it is a very flexible communication paradigm, its many-to-many one-way transport is not appropriate for request / reply interactions, which are often required in a distributed system. Request / reply is done via a service, which is defined by a pair of messages: one for the request and one for the reply. A providing ROS node offers a service under a string name, and a client calls the service by sending the request message and awaiting the reply.

The example is in the figure 3.8. With the implementation of service, the image processing node first requests the topic */image_data*, the camera node gathers data from the camera, and then sends the reply. It realizes that the nodes can gather data from other nodes at specific time, namely only when the data is requested.

### 3.2.2 SILAB

The simulation software SILAB introduced by the Würzburg Institut for Traffic Sciences GmbH (WIVW) is used for the driving simulation study. SILAB offers a reproducible vehicle and traffic environment with the possibility exporting vehicle states and data for further research. The figure 3.6 shows a simulated urban scenario in SILAB.

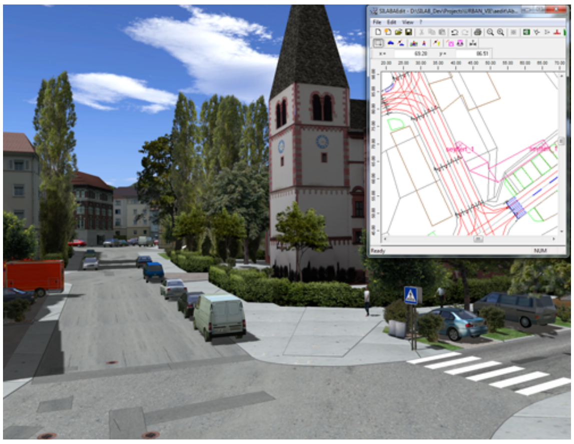


Figure 3.6: A simulation scenario of in SILAB

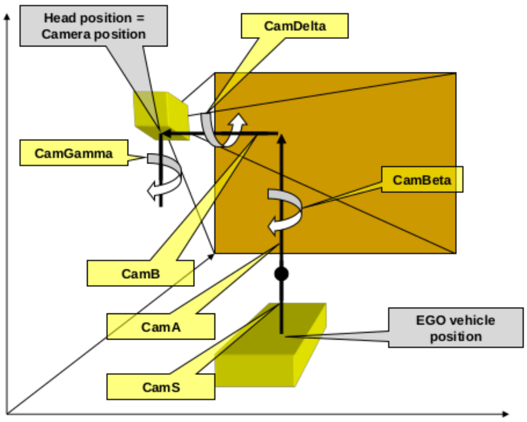


Figure 3.7: camera positioning in SILAB

The communication between SILAB and the Interface is achieved by a Digital Processing Unit (DPU), called *DPUTOFInterface*. The DPU transmits control commands and vehicle data between SILAB and the Interface.

The camera positioning plays a significant role in this work because it greatly affects the lane detection. In SILAB, it is possible to simulate every camera position and heading by modifying the 6 parameters in the file *DPUSGE_TOF.inc*. The meaning of these parameters are described in the figure 3.7.

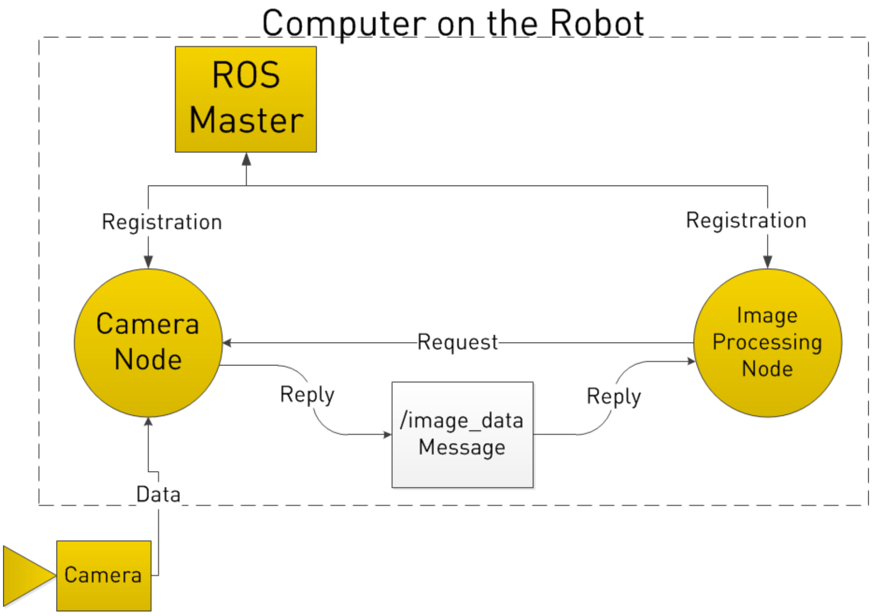


Figure 3.8: Services in ROS[40]

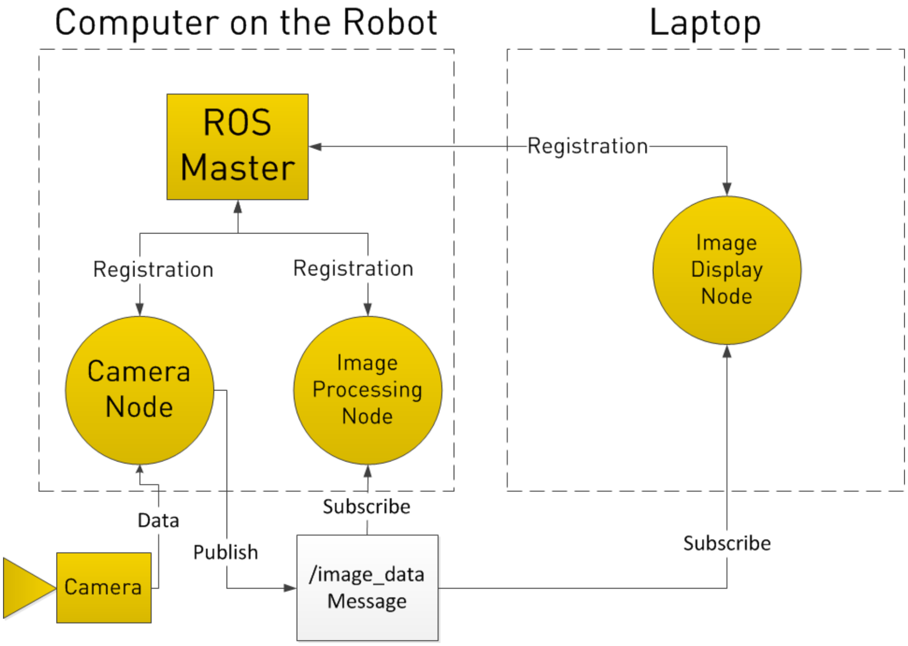


Figure 3.9: Publishing and subscribing in ROS[40]

# 4 Controller Design

This chapter contains the building of simulation environments and the design of controllers. In the first section, the simulation environment is introduced. The control algorithm is implemented in the simulation environment and the controller gains are tuned based on the control performance. Next, some modern methods for combined controller are introduced.

## 4.1 Environment

This section mainly includes how to build a simulation environment for an autonomous robot. The path and the trajectory of robots are visualized at the end of this section. An important property of autonomous robots, the coordinate transformation is also introduced in this section. A simulation of latency is presented in the last subsection.

### 4.1.1 Path

The simulation environment is implemented in Python. The path is expressed as discrete points. In the visualization, the points are connected by lines. Therefore, the path is shown in figure 4.3 as a continuous curve. In the following example, there are 201 points in total over 50$m$ to express the sinusoidal path.

```python
ax = np.linspace(0, 50, 201)
ay = [10 * math.sin(ix / 10) for ix in ax]
```

### 4.1.2 Vehicle

The function *update* updates the vehicle state every $\mathrm{d}t$. The vehicle heading angle $\theta$ affects along with the vehicle velocity the horizontal and vertical position. The steering angle affects the vehicle heading angle, the mathematical formula of which is given in the section 3.1.1.

It is worth mentioning that the input of the function state and delta, are respectively the vehicle state before the execution of this function and the steering angle, which is computed by the control algorithm. The output is the vehicle state after the execution of the function.

```python
def update(state, delta):

    state.x = state.x + state.v * math.cos(state.psi) * dt
    state.y = state.y + state.v * math.sin(state.psi) * dt
    state.theta = state.theta + state.v/L * math.tan(delta) * dt
    state.v = state.v

    return state
```

### 4.1.3 Coordinate Transformation

Autonomous robots make sensor observations usually in their ego-centric frame. Figure 4.1 shows the detected lanes in vehicle ego-centric frame. In this project, there is no high definition map or GPS available to localize the vehicle in the world coordinates. Therefore, the lateral control also has to be computed in the ego-centric system.

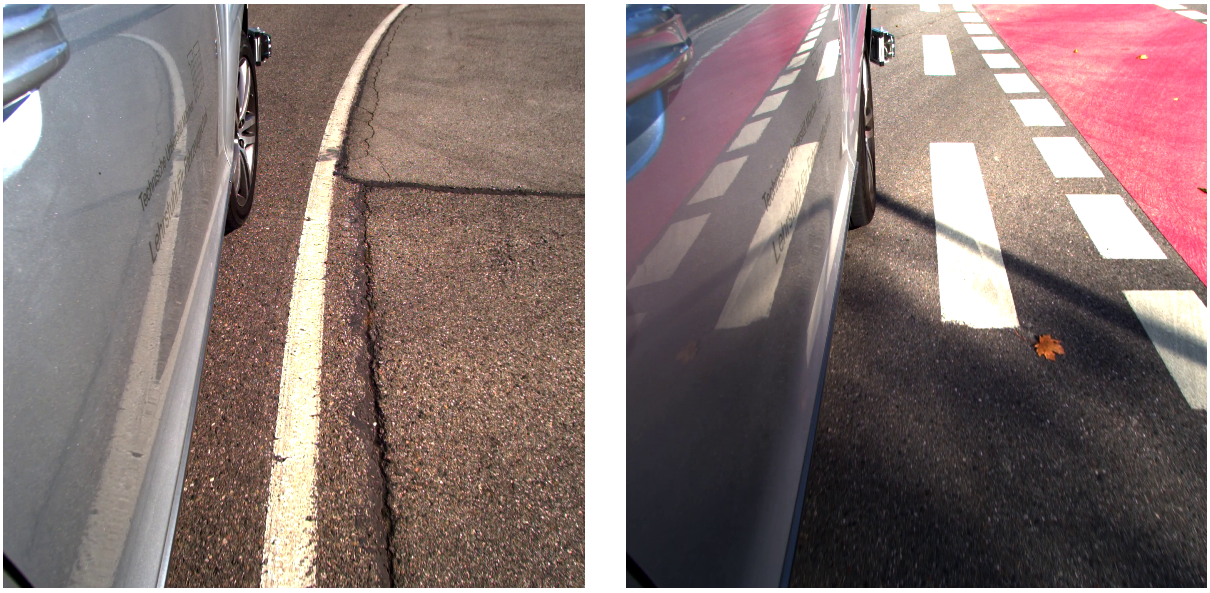


Figure 4.1: Detected lanes in the ego-centric frame

For this reason, the coordinate transformation is necessary for the development of a controller. At first, the index of the nearest path point is determined. This process is implemented in a function named *calc_nearest_index*:

```python
def calc_nearest_index(state, path points):

    dx = [state.x - icx for icx in cx]
    dy = [state.y - icy for icy in cy]

    d = [idx ** 2 + idy ** 2 for (idx, idy) in zip(dx, dy)]

    # get the index of minimal distance in all of distance
    i = d.index(min(d))

    return i
```

This function calculates the distances between every path point and the vehicle position and returns the index of minimal distance. The more compact the path points are, the more accurate the nearest path point relative to the vehicle position is.

Figure 4.2 shows the principle of the 2 dimensional coordinate transformation in a cartesian coordinate system. The black coordinate $x, y$ is the higher-order object coordinate system, frequently called the world coordinate system. For a rotation by an angle $\theta$ counterclockwise around the origin, the function form is $x' = x\cos(\theta) + y\sin(\theta)$ and $y' = -x\sin(\theta) + y\cos(\theta)$. It is simple to rewrite it in a rotation matrix:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}. \tag{4.1}$$

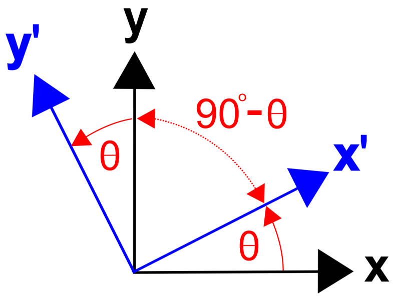


Figure 4.2: A cartesian coordinate transformation in 2 dimension

For transforming the coordinate system, it is necessary to gather $n$ points after the nearest path points. The vehicle camera range decides the n. For example, the distance between 2 discrete path points is $0.1m$. If the required preview length in the simulation is $4m$, then n should be 40 to gather all of the path points $4m$ in front of the vehicle.

Figure 4.3 shows the total path for the vehicle in the world coordinate system. As shown in figure 4.4 and 4.5, the vehicle has $4m$ preview length. As explained before, $[x, y]$ in figure 4.4 is path point position in world coordinates and $[x', y']$ in figure 4.5 is the path point position in the vehicle ego-centric frame. It is worth mentioning that the positive x-direction is the vehicle heading and the origin refers to the middle point of the vehicle rear wheels. It is obvious that the vehicle should be steered right in this example.

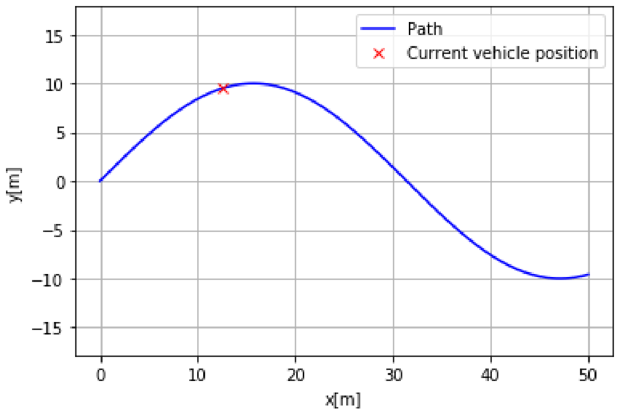


Figure 4.3: The global path in the world coordinate system

The path points, which were in the world coordinate system on $[x, y]$, are now in the vehicle ego-centric frame on $[x', y']$. The function poly_fit is implemented to transform the coordinate system to ego-centric frame and computes the path to a 3 degree polynomial.

```
def poly_fit(i, cx, cy):

    # use 40 Point(about 4m) to polyfit a curve
    x = [cx[ia] - state.x for ia in range(i,i+n)]
    y = [cy[ia] - state.y for ia in range(i,i+n)]

    x_dot = [(ix * cos(state.theta)) + (iy * sin(state.theta))
  for (ix, iy) in zip(x, y)]
    y_dot = [(-ix * sin(state.theta)) + (iy * np.cos(state.theta)
) for (ix, iy) in zip(x, y)]

```

```
    polynom = np.polyfit(fx, fy, 3)

    return polynom
```

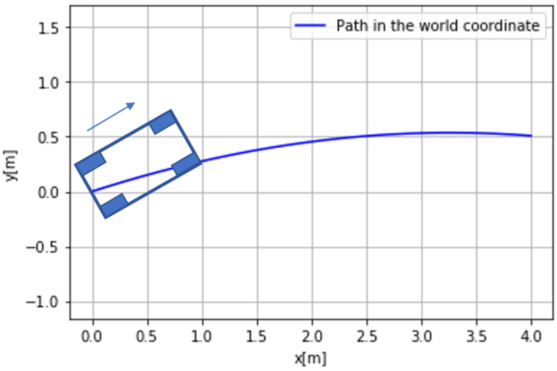


Figure 4.4: The local path in the world coordinate system

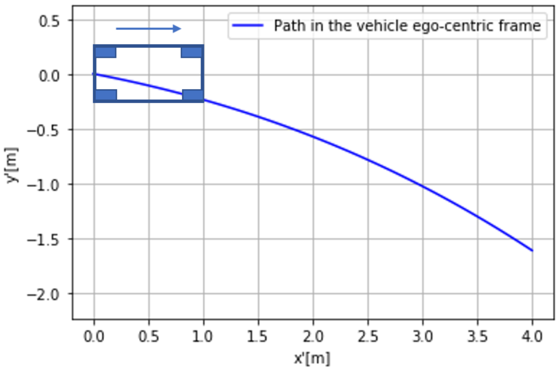


Figure 4.5: The local path in the ego-centric frame

Using the functions in this subsection, it is possible to simulate any path and transform it into ego-centric frame, which is very helpful to develop a lateral controller.

### 4.1.4 Minimal Distance

For a discrete system, the minimal distance which is computed via $d = \sqrt{(x-x_p)^2+(y-y_p)^2}$ is not accurate. $(x,y)$ is the vehicle position and $(x_p,y_p)$ is the nearest path point in this discrete system. It is necessary to define a separate function to compute the minimal distance.

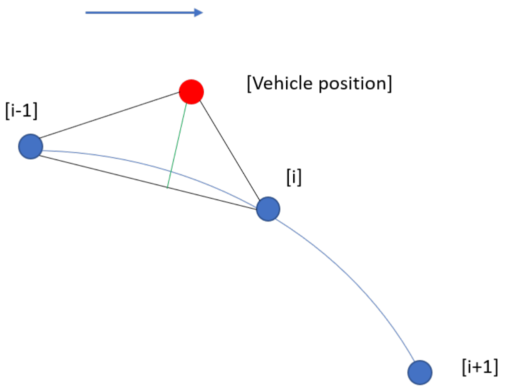


Figure 4.6: Vehicle position between [i-1] and [i]

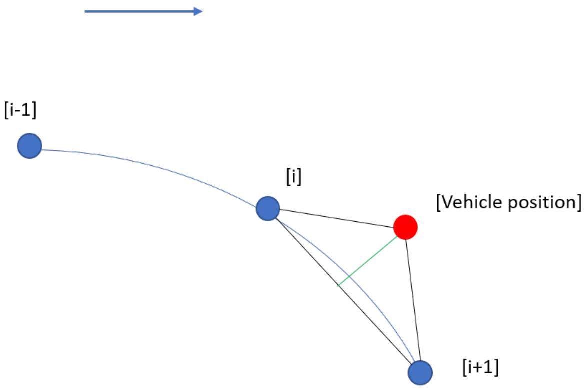


Figure 4.7: Vehicle position between [i] and [i+1]

Figure 4.6 and 4.7 shows the principle of this algorithm. First of all, is is determined whether the vehicle is in front or behind the nearest discrete path point. It is not complicated to compute the 3 side lengths of the triangle in figures if all the point positions are already known. By use of Heron's formula, the area of the triangle can be calculated:

$$A = \sqrt{s(s-a)(s-b)(s-c)}, \tag{4.2}$$

where $a,b,c$ are the side lengths of this triangle, $s$ is the semi-perimeter of the angle and can be calculated by:

$$s = \frac{a+b+c}{2}. \tag{4.3}$$

Once the area of the triangle is known, the improved minimal distance, namely the height of this triangle (green lines in the figure 4.6 and 4.7) is easy to calculate by:

$$h = \frac{2A}{c}. \tag{4.4}$$

The implementation in Python is:

```
    if bx[i]>0:
        a = math.sqrt(bx[i]**2 + by[i]**2)
        b = math.sqrt(bx[i-1]**2 + by[i-1]**2)
        c = math.sqrt((bx[i]-bx[i-1])**2 + (by[i]-by[i-1])**2)
        s = (a+b+c) / 2.0
        area = math.sqrt(s * (s-a) * (s-b) * (s-c))
        error = 2 * area /c

    elif bx[i]<0:
        a = math.sqrt(bx[i]**2 + by[i]**2)
        b = math.sqrt(bx[i+1]**2 + by[i+1]**2)
        c = math.sqrt((bx[i+1]-bx[i])**2 + (by[i+1]-by[i])**2)
        s = (a+b+c) / 2.0
        area = math.sqrt(s * (s-a) * (s-b) * (s-c))
        error = 2 * area /c
```

### 4.1.5 Latency

Latency is a time delay between phenomena and observation of some physical change in the system. Latency is physically a consequence of the limited velocity with which any physical interaction can propagate. The magnitude of this velocity is always less than or equal to the speed of light. Therefore, every physical system with any physical separation (distance) between cause and effect will experience some sort of latency, regardless of the nature of stimulation.

Clearly, there is a latency between the lane detection module and the steering module. It is useful to simulate the latency in the simulation environment for controller development. If the lateral controller performs well in the simulation with latency, it is rational to assume that the controller also performs well in the real world, where latency is nonnegligible.

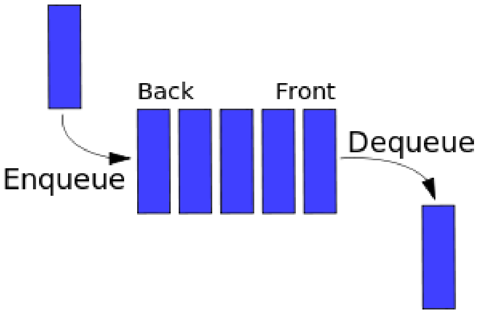


Figure 4.8: First in-first out in the queueing theory

The simulation of latency is based on the queueing theory. First of all, a queue with a size of n is initialized to store the potential steering angles. n is determined by latency divided by the time step. The simulation of latency follows the first in-first out (FIFO) principle, which is shown in figure 4.8. Each time a steering angle is calculated and is added to the queue. Values are only popped from the queue when it is full. Therefore, the vehicle steering unit only uses the steering angle calculated $n \cdot dt$ seconds prior. This introduces latency between lane detection module and vehicle steering unit.

```
import queue
from collections import deque
#define the size of queue
domains = [0] * int(queue_size)
lenkradwinkel_queue = deque(domains)

def latency(state, path points):

    lenkradwinkel_future = path_tracking_controller(state, path
  points)
    lenkradwinkel_use = lenkradwinkel_queue.pop()
    lenkradwinkel_queue.appendleft(lenkradwinkel_future)

    return lenkradwinkel_use
```

The function *latency* is the implementation of system latency. The steering angle which is computes based on the actual vehicle state is appended on the left site of the queue. At the same time, the steering angle on the right side of the queue is dequeued and applied to the vehicle. The results shown in section 5 demonstrates that the simulation method can simulate latency correctly.

## 4.2 Implementation of Controllers

This section mainly includes the implementation of 3 lateral control methods. The 3 control methods are PID, Stanley and Pure Pursuit. The principle and the state-of-the-art of these path-tracking control methods were introduced in section 2.3. The simulation and visualization are based on the environment described in the section 4.1.

### 4.2.1 Incremental PID Controller

The principle and the base form of a PID controller was introduced in section 2.3.1. However, a computer can only compute a discrete signal. Therefore, the PID algorithm must be discretized to be implemented.

The resulting digital signal $y(t_k)$ is use in the control function to calculate the value of the control signal $u(t_k)$. Figure 4.9 shows how a control loop be implemented. As shown in the figure, the control signal is scaled and sent to the Digital-Analog (DA) Converter, where it is held constant during the present time step, followed by the control action of the plant. The computer registers the process measurement signal via an Analog-Digital (AD) converter. The AD converter produces a numerical value which represents the measurement.

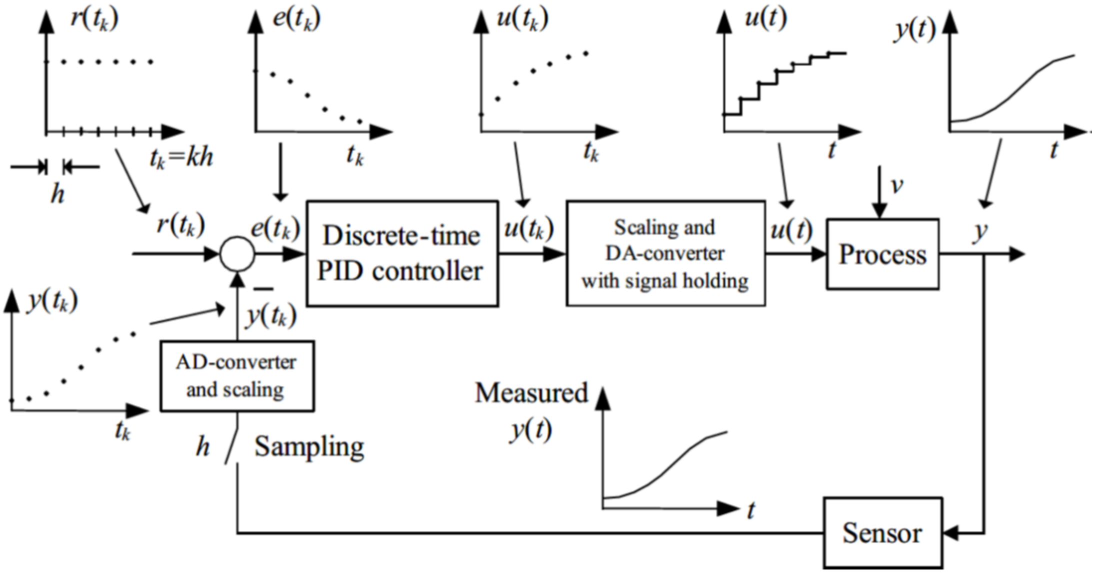


Figure 4.9: Control loop implemented with a computer[41]

There are two prevailing PID forms:

- Position PID algorithm: the output of the controller is the absolute value of the control signal.
  The component form of a discrete Position PID is

$$u(k) = K_p e(k) + K_i \sum e(k) + K_d[e(k) - e(k-1)], \tag{4.5}$$

  which needs to store all previous errors $e_1, e_2, ..., e_{k-1}$.
- Incremental PID algorithm: the output of the controller represents the increments of the control signal. It defines the incremental control value

$$\Delta u_k = u_k - u_{k-1}. \tag{4.6}$$

The form of discrete incremental PID is

$$\Delta u_k = K_p[e(k) - e(k-1)] + K_i e(k) + K_d[e(k) - 2e(k-1) + e(k-2)]. \tag{4.7}$$

This algorithm does not need to store all previous errors but only the 3 most recent errors $e_{k-2}, e_{k-1}, e_k$.

Error $e_k$ describes the distance between the actual vehicle state and the nearest path point. It is computed with the function *closest_error* which is presented in subsection 4.1.4. To develop a position PID controller, the $e_k$ is applied in equation 4.5. Alternatively, the $e_k$ is applied in equation 4.7 to develop an incremental PID controller.

```
def pid_controller(state, path points, error_array, delta_array):

    error = closest_error(state,path points)

    k_e = len(error_array)
    j_e = len(delta_array)

    e_ke = error
    e_k1e = error_array[k_e-2]
    e_k2e = error_array[k_e-3]

    delta_k1_e = delta_array[j_e-1]

    delta_delta_e = kp * (e_ke - e_k1e) + ki * e_ke + kd * (e_ke
   - 2.0*e_k1e + e_k2e)
    delta_e = delta_k1_e + delta_delta_e

    if delta_e>math.pi/4.0:
        delta = math.pi/4.0
    elif delta_e<-math.pi/4.0:
        delta = -math.pi/4.0
    else:
        delta = delta_e

    return delta
```

Function *pid_controller* is the implementation of an incremental PID controller. *error_array* is a list where all previous errors are stored. The incremental PID control algorithm computes a change of steering angle. It is necessary to store the last steering angle to calculate the actual steering angle. For this reason, a list called *delta_array* is implemented. Clearly, there is a maximal and minimal steering angle and $\delta_e$ is checked whether is beyond the limitation of vehicle constraint. It implemented at the end of the function.

Now, a incremental PID controller is fully implemented. The results and the discussion are in section 5.

### 4.2.2 Stanley Controller

The Stanley control method helped Stanford University to win the DARPA challenge in 2007. The principle of Stanley control is introduced in section 2.3.1. The base formula of Stanley control is

$$\delta(t) = \theta_e(t) + \tan^{-1}(\frac{ke(t)}{v_x(t)}), \tag{4.8}$$

where $\theta_e(t)$ is the angle difference between the vehicle heading and the heading of the nearest path point, $e(t)$ is the lateral error and $v_x(t)$ is the longitudinal velocity. The calculation of $e(t)$ is introduced in former sections and the longitudinal velocity is predefined by users.

To compute $\theta_e(t)$, it is necessary to import a derivative function to calculate the tangent of the path on the nearest point. The function *poly_fit*, first introduced in the section 4.1.3, aims to fit the discrete points curve to compute the polynomial parameters. Next, a continuous curve is reconstructed with these polynomial parameters and the heading of the path on the nearest point can be computed. Because it is already in the vehicle ego-centric frame, the tangent of the path is already the heading error. This algorithm is implemented in the function *calc_theta_e*.

```
from scipy.misch import derivative

def calc_theta_e(state, path points):

    polynom parameter  = poly_fit(state, path points)

    def f(x):
        return p3 * (x ** 3) + p2 * (x ** 2) + p1 * x + p0

    d1y_dx1 = derivative(f, nearest path point x value, dx=1e-6)

    theta_e = math.atan2(d1y_dx1, 1.0)

    return theta_e
```

However, the directions of $\theta_e(t)$ and $e(t)$ are confusing factors. They have to be clearly predefined. As shown in figure 4.10, the magnitude of the scalar value e is illustrated in red. As illustrated $e(t) > 0$, and for the case where the car is to the left of the path, $e(t) < 0$. Besides, as illustrated $\theta_e(t) < 0$, and for the case where the car heading is to the left of the path, $\theta_e(t) > 0$.

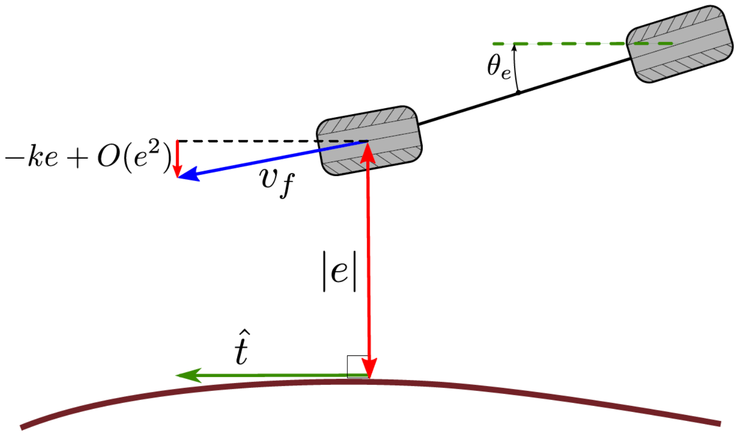


Figure 4.10: $\theta_e(t)$ and $e(t)$ in Stanley control[21]

After the calculation of $\theta_e(t)$ and $e(t)$, it is not complicated to implement equation 4.8 as function *stanley_control*.

```
def stanley_control(state, path points):

    th_e = calc_theta_e(state, path points)

    #longitudinal velocity predefined
    v = state.v

    delta = th_e + math.atan2(k*-e/v, 1.0)

    return delta
```

Now, a Stanley controller is implemented. The results and discussion are a part of section 5.

### 4.2.3 Pure Pursuit Controller

The principle of the Pure Pursuit control method is already introduced in section 2.3.3. The feedback controllers, which compute the steering angle via lateral and heading errors, compute the steering angle in the world coordinate and the vehicle ego-centric system in the same way. However, unlike the feedback controllers like the PID and the Stanley controller, the Pure Pursuit controller has a totally different implementation in the vehicle ego-centric frame.

The key issue of the Pure Pursuit algorithm is angle $\alpha$. The steering angle is given as

$$\delta(t) = \tan^{-1}\left(\frac{2L\sin(\alpha(t))}{L_d}\right), \tag{4.9}$$

where an angle $\alpha$ is required. The angle $\alpha$ in the world coordinate system is given by

$$\alpha = \arctan\left(\frac{g_y - y_r}{g_x - x_r}\right) - \theta, \tag{4.10}$$

where the $(g_x, g_y)$ is the goal point on the path in the world coordinate system and the $(x_r, y_r)$ and $\theta$ are the vehicle rear position and heading angle in world coordinates, respectively. However, there is no position or angle in the vehicle ego-centric frame. Therefore, there is a another method to calculate the angle $\alpha$. This method is introduced in the following text.

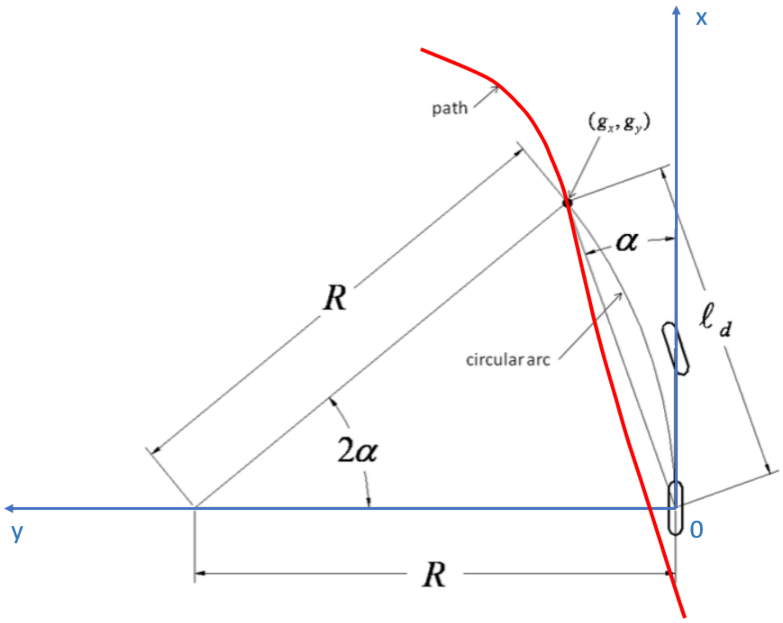


Figure 4.11: A Pure Pursuit geometry in the vehicle ego-centric frame

Figure 4.11 shows a Pure Pursuit geometry in the vehicle ego-centric frame. The detected lane in this figure is illustrated in red. A Pure Pursuit controller in the vehicle ego-centric frame uses the polynomial parameters as the inputs.

A simplified geometry of the path is shown in the figure 4.12. The coordinate system is rotated 90 degrees clockwise to keep close to the common practice, which is helpful for understanding. As shown in this figure, the length of the side in orange is $L_d\sin(\alpha)$ and the length of the side in yellow is $L_d\cos(\alpha)$.

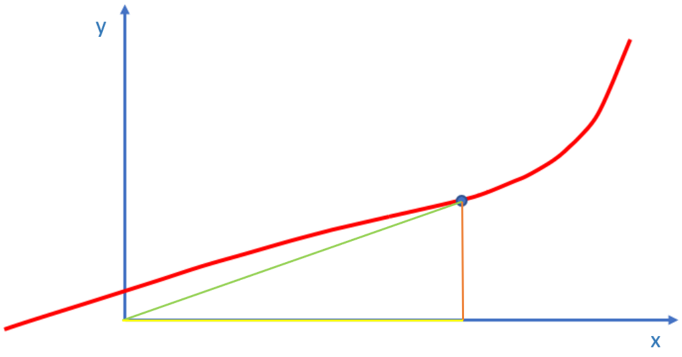


Figure 4.12: A simplified path geometry after rotation

It is not difficult to define the angle $\alpha$ in such a coordinate system. To calculate the angle $\alpha$, it is helpful to implement

$$x = L_d cos(\alpha) \tag{4.11}$$

into function

$$p_3x^3 + p_2x^2 + p_1x + p_0 = L_d sin(\alpha), \tag{4.12}$$

where $p_3, p_2, p_1, p_0$ are the polynomial parameters of the illustrated curve in figure 4.12. Therefore, only x is unknown in this equation. By use of the *fsolve* function from *scipy.optimize*, it is not complicated to solve for $\alpha$.

The implementation of the Pure Pursuit algorithm is

```python
from scipy.optimize import fsolve

def pure_pursuit_controller(state, path points):

    polynom = poly_Abschnitt(state, path points)

    def f(x):
        return ((ld*math.cos(x))**3)*p_3+((ld*math.cos(x))**2)*
    p_2+(ld*math.cos(x))*p_1+p_0-ld*math.sin(x)

    alpha = fsolve(f, 0)

    delta = math.atan2(2.0 * long * math.sin(alpha) / ld, 1.0)

    return delta
```

The results and evaluation of Pure Pursuit are presented in section 5.

## 4.3 Combination of Controllers

The Pure Pursuit controller is widely used to solve the path-tracking problem for the autonomous vehicle. However, its performance depends highly on the look-ahead distance. Besides, the geometric relationship does not take the impacts of the tracking error into consideration. This section introduces a method of combined controllers. The geometric controller Pure Pursuit is used as the basic controller and a position Proportional Integral (PI) controller is used to improve the lateral control performance. Figure 4.13 shows the structure of a combined Pure Pursuit - PI controller. At the end of the combining process, a low-pass filter is added to smooth the control performance.

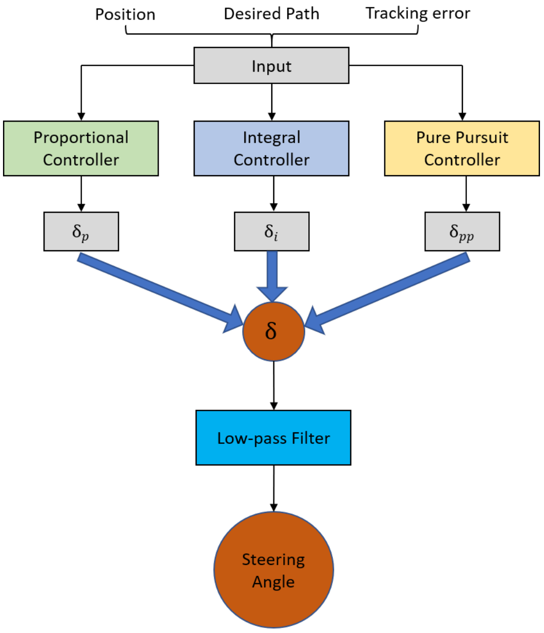


Figure 4.13: The Structure of a combined Pure Pursuit - PI controller[19]

The role of the P-controller is to add a feedback mechanism to the steering angle prediction of Pure Pursuit to reflect the tracking error. The control law of the P controller is

$$\delta_p = K_p e_{la}, \tag{4.13}$$

where the look-ahead error $e_{la}$ is a combination of both lateral error e and heading error $\theta_e$. The look-ahead error, namely the minimal distance, is illustrated in green in figure 4.14.

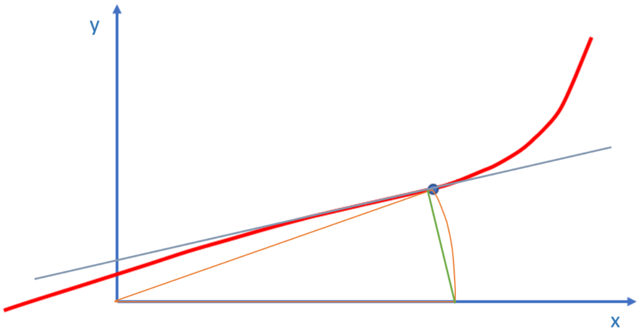


Figure 4.14: A geometry relationship of look-ahead error $e_{la}$

The role of the I-controller is to reduce the steady-state error. In this method, a position I-controller is used to integrate the derivations from the tracked path over a period time. It is given as

$$\delta_i = K_i \sum e. \tag{4.14}$$

A Low-pass filter (LPF) can reduce the amplitude of a sudden change in a certain period of time. Actually, the sudden change of the autonomous vehicle may happen under many situations, like an error in perception, the location and so on. The filter is presented as

$$\delta = \sum_{0}^{l-2} \delta_{previous}\omega_{previous} + \delta_{current}\omega_{current}, \tag{4.15}$$

where $\omega_{previous}$ is the weight of the previous steering angle, $\omega_{current}$ is the current steering angle's weight and $l$ is the length of the filter window, which is given as

$$\omega_{previous} = \frac{1-\omega_{current}}{l-1}. \tag{4.16}$$

$\delta_{current}$ is the current computed output of the steering angle. The LPF can synthesize the previous steering angle output, smooth the final steering angle and calculate $\delta$.The final steering angle output can be given as

$$\delta(t) = LPF(K_{pp}\delta_{pp} + \delta_p + \delta_i), \tag{4.17}$$

where $K_{pp}$ is the control parameter of the Pure Pursuit controller that needs to be adjusted according to the actual situation. $\delta_p$ and $\delta_i$ are determined via equation 4.13 and 4.14. The sum of $K_{pp}$, $K_p$ and $K_i$ is equal to one because these three controllers are mutually restricted.

The results and comparison of separate controllers and combined controller are presented in section 5.

## 4.4 Experiments in SILAB

Two computers work together to run path detection and tracking experiments in SILAB. PC_1 runs a Windows OS because SILAB only works on computers running Windows OS. PC_2 runs Ubuntu OS and ROS is integrated into Ubuntu OS. A program in PC_1 named *ScreenStreamer144* is executed to send the SILAB window from PC_1 to PC_2. A launch file called *storm.launch* runs in ROS to receive the SILAB window from PC_1.

Figure 4.15 shows a ROS graph including all involving nodes and topics. As shown in the figure, the received video from the window compute is divided in images with a fixed frequency. A lane detection node in ROS uses the images as input and computes the polynomial of curve of the latest lane and publishes it as a message. Thanks to ROS, a lateral controller node subscribes the message and uses it to calculate the steering angle. Finally, the computed steering angle is sent to SILAB. Here, the system is a closed loop.

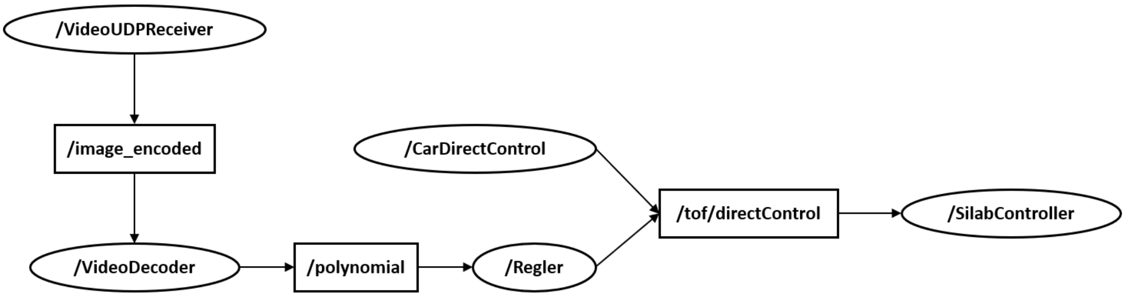


Figure 4.15: A graph including nodes and topics in ROS

Figure 4.16 shows a visualization of the lane detection process. The lane detection algorithm was implemented by Bomelburg-Zacharias[2], who works for project SToRM. The original picture in SILAB is on the upper left corner of the figure. After 6-step lane detection process, a polynomial in the current vehicle ego-centric frame is determined and visualized as a curve on the lower right corner of the figure.

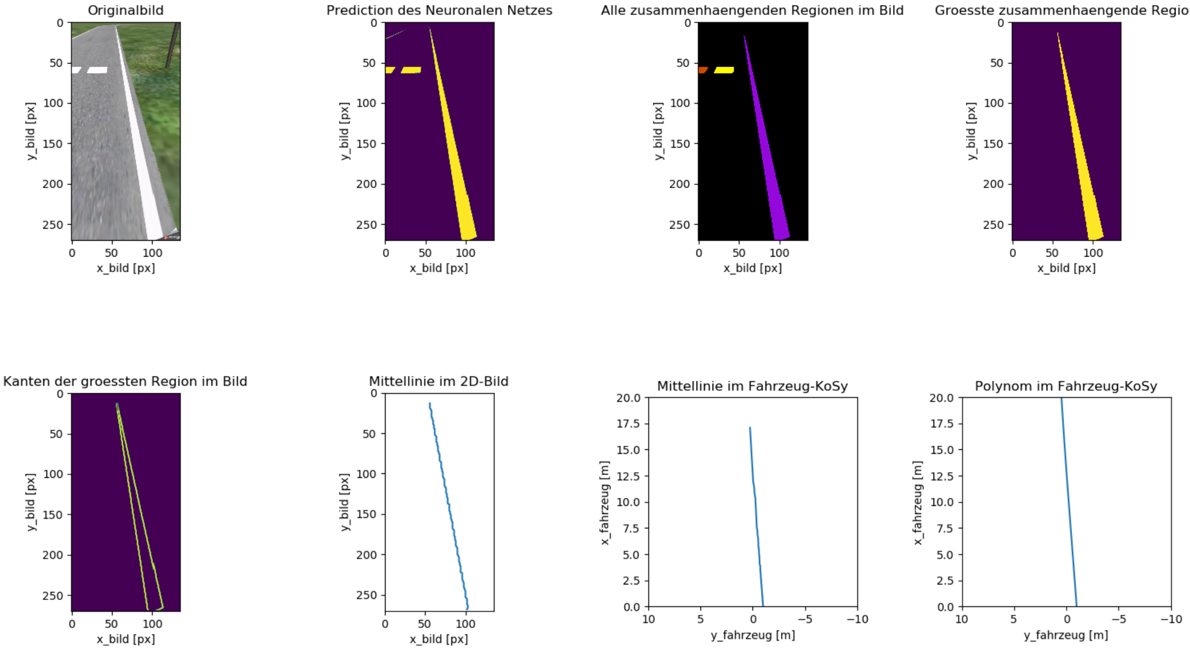
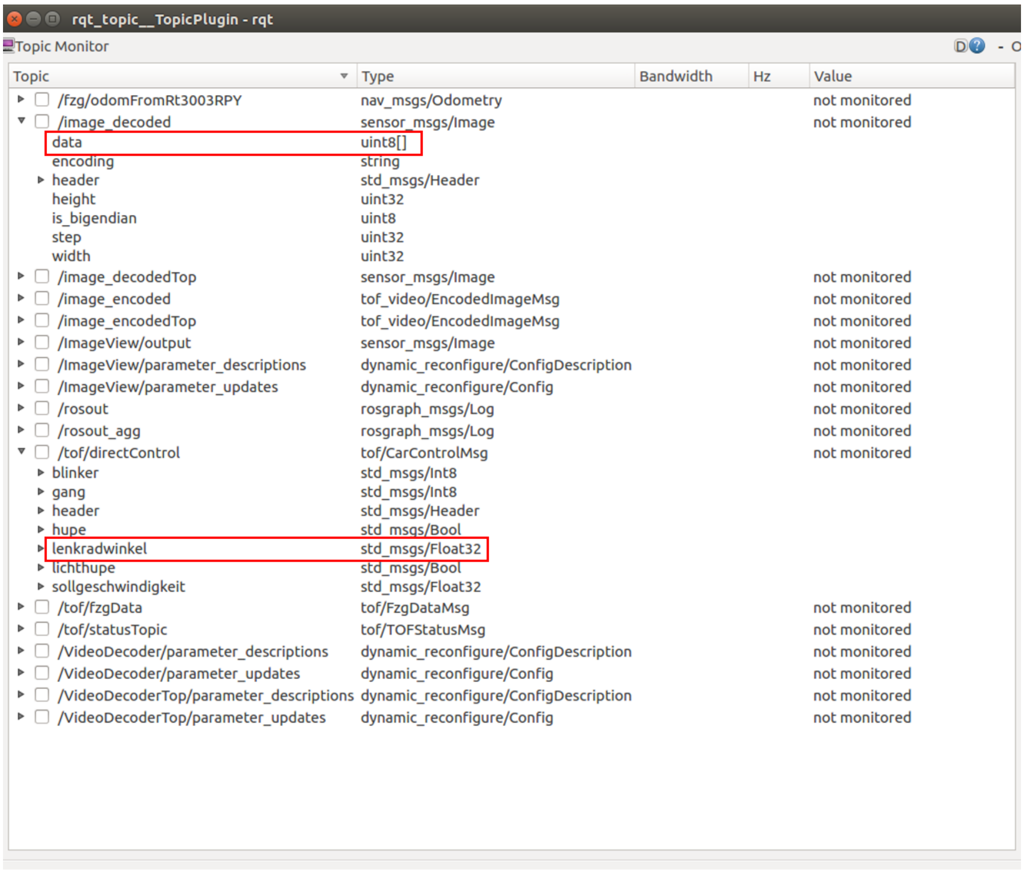


Figure 4.16: Process of lane marking detection

ROS provides many tool packages, which are very helpful in observing and checking the system. For example, figure 4.15 is a ROS graph created by the *rqt* package. *rqt* is a Qt-based framework Graphical User Interface (GUI) development for ROS. There is a another important function of the *rqt* package: *rqt_topic*. *rqt_topic* provides a GUI plugin for displaying debug information about ROS topics including publishers, subscribers, publishing rate and ROS messages.



| Topic | Type | Bandwidth | Hz | Value |
|---|---|---|---|---|
| ▸ ☐ /fzg/odomFromRt3003RPY | nav_msgs/Odometry | | | not monitored |
| ▾ ☐ /image_decoded | sensor_msgs/Image | | | not monitored |
| data | uint8[] | | | |
| encoding | string | | | |
| ▸ header | std_msgs/Header | | | |
| height | uint32 | | | |
| is_bigendian | uint8 | | | |
| step | uint32 | | | |
| width | uint32 | | | |
| ▸ ☐ /image_decodedTop | sensor_msgs/Image | | | not monitored |
| ▸ ☐ /image_encoded | tof_video/EncodedImageMsg | | | not monitored |
| ▸ ☐ /image_encodedTop | tof_video/EncodedImageMsg | | | not monitored |
| ▸ ☐ /ImageView/output | sensor_msgs/Image | | | not monitored |
| ▸ ☐ /ImageView/parameter_descriptions | dynamic_reconfigure/ConfigDescription | | | not monitored |
| ▸ ☐ /ImageView/parameter_updates | dynamic_reconfigure/Config | | | not monitored |
| ▸ ☐ /rosout | rosgraph_msgs/Log | | | not monitored |
| ▸ ☐ /rosout_agg | rosgraph_msgs/Log | | | not monitored |
| ▾ ☐ /tof/directControl | tof/CarControlMsg | | | not monitored |
| ▸ blinker | std_msgs/Int8 | | | |
| ▸ gang | std_msgs/Int8 | | | |
| ▸ header | std_msgs/Header | | | |
| ▸ hupe | std_msgs/Bool | | | |
| ▸ lenkradwinkel | std_msgs/Float32 | | | |
| ▸ lichthupe | std_msgs/Bool | | | |
| ▸ sollgeschwindigkeit | std_msgs/Float32 | | | |
| ▸ ☐ /tof/fzgData | tof/FzgDataMsg | | | not monitored |
| ▸ ☐ /tof/statusTopic | tof/TOFStatusMsg | | | not monitored |
| ▸ ☐ /VideoDecoder/parameter_descriptions | dynamic_reconfigure/ConfigDescription | | | not monitored |
| ▸ ☐ /VideoDecoder/parameter_updates | dynamic_reconfigure/Config | | | not monitored |
| ▸ ☐ /VideoDecoderTop/parameter_descriptions | dynamic_reconfigure/ConfigDescription | | | not monitored |
| ▸ ☐ /VideoDecoderTop/parameter_updates | dynamic_reconfigure/Config | | | not monitored |

Figure 4.17: A rqt_topic window in the experiment in SILAB

Figure 4.17 shows the *rqt_topic* window in this experiment. The often used topics can be found here. For example, the polynomial of curve is stored in *data* in branch */image_decoded*. The value and the frequency is also given in this window. Additionally, the *lenkradwinkel* (steering angle) plays also a significant role in experiments.

# 5 Result and Discussion

This chapter mainly includes the results and discussions of the implemented controllers. Section 5.1 shows the lateral control performance of PID, Stanley and Pure Pursuit. An adaptive Pure Pursuit is introduced in section 5.1.4. It has an adaptive look-ahead distance, which is computed a priori via the path curvature. A comparison of modern lateral controllers is located in the last section.

## 5.1 Lateral Control Performance

In this section, the lateral control performance of all 3 controllers implemented in chapter 4 is discussed. The results are computed based on the simulation environment introduced in section 4.1. All used vehicle parameters are given in table 5.1.

Table 5.1: Used vehicle parameters in experiments

| Vehicle Parameters | | | | |
|---|---|---|---|---|
| Reference path | Wheelbase | Velocity | Steering limit | Initial configuration |
| $(x_s(s), y_s(s)) = (s, 10 \cdot \sin(s/10))$ | $2.82m$ | $2\frac{m}{s}$ | $\lvert\delta\rvert \leq \frac{\pi}{4}$ | $(x_r(s), y_r(s)) = (0, 0)$ |
| $(x_s(s), y_s(s)) = (s, 2 \cdot \arctan(s) + 3.1)$ | | | | |
| $(x_s(s), y_s(s)) = (s, -(\kappa_r^2 - s^2)^{1/2}) + \kappa_r$ | | | | |
| Real route in Garching near Munich | | | | |

The first 2 paths in table 5.1 are illustrated in figure 5.1 and 5.2, the red rectangle represents vehicle. As shown in the 2 figures, path 1 is a long route with enormous curvature change, path 2 is a route but with a lane changing. Reference path 3 is first introduced in subsection 5.1.4

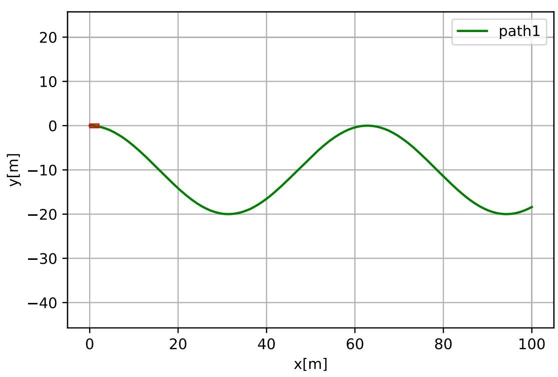


Figure 5.1: Reference path 1

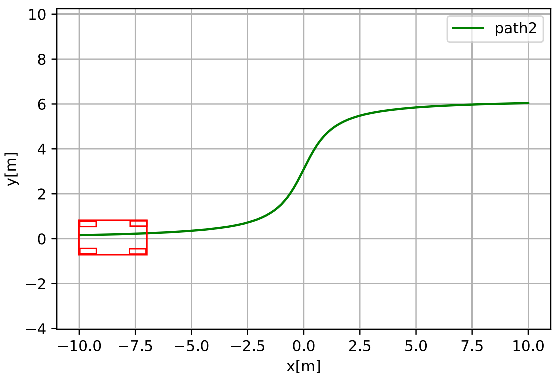


Figure 5.2: Reference path 2

In following subsections, the lateral control performance of controllers in Path 1 is shown. The results of the performance of other controllers in path 2 are given in the appendix.

### 5.1.1 Stanley Controller

This section shows the lateral control performance of the Stanley controller with different gains $k$. The principle of the Stanley controller was introduced in 4.2.2. The base formula is given as

$$\delta(t) = \theta_e(t) + \tan^{-1}\left(\frac{ke(t)}{v_x(t)}\right), \tag{5.1}$$

where $k$ is the only modifiable parameter. Figure 5.3 shows the relatioship between the vehicle steering angle and the path curvature. Figure 5.4 shows the lateral error and the heading error of the vehicle with a Stanley controller having $k = 0.8$ in reference path 1. The figures show clearly that the path curvature affects the lateral control performance greatly.

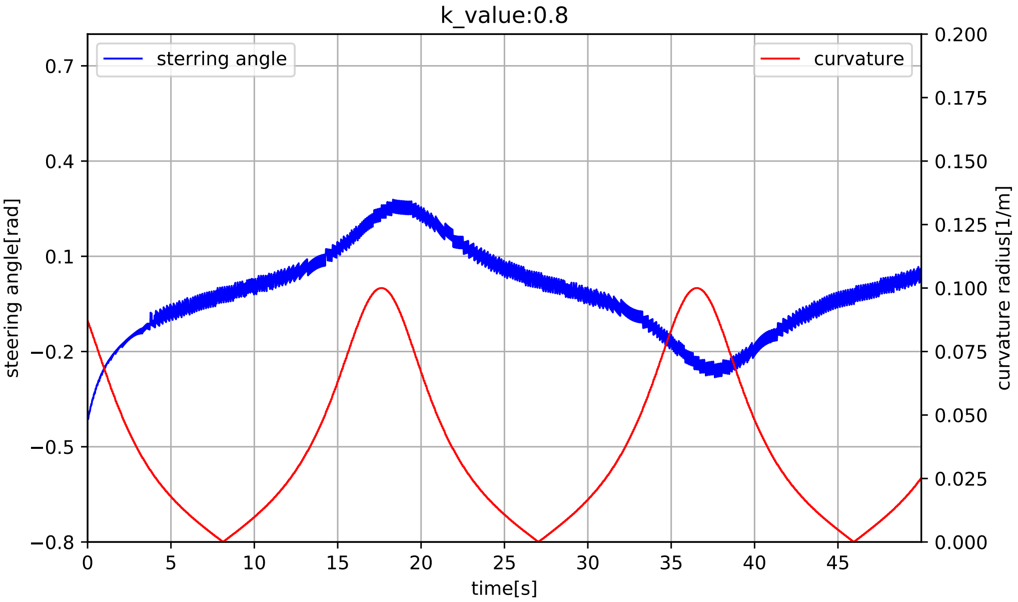


Figure 5.3: Stanley with $k = 0.8$, steering angle and curvature of the path

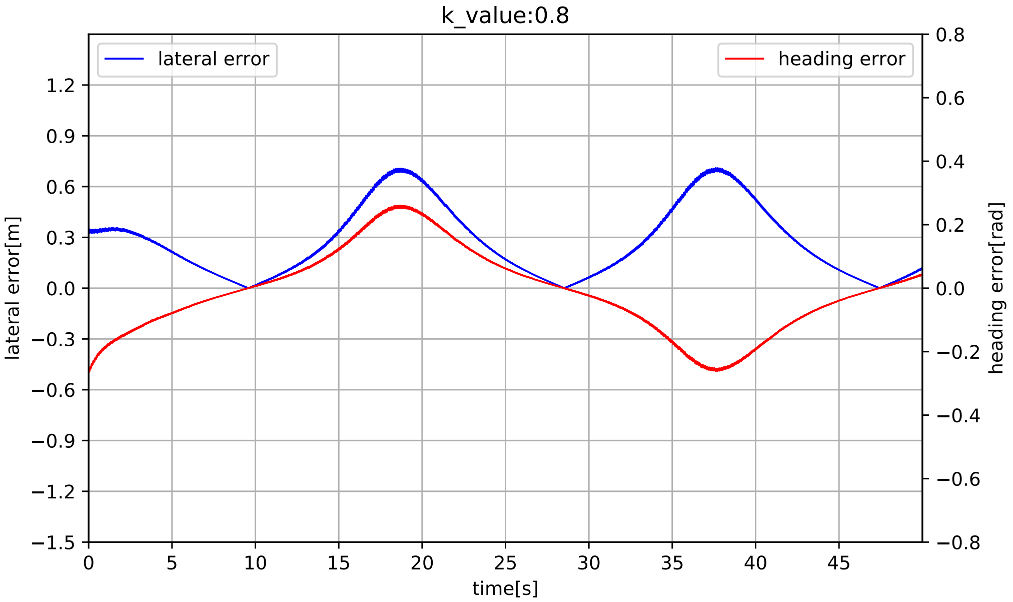


Figure 5.4: Stanley with $k = 0.8$, lateral error and heading error

Next, the lateral control performance of the Stanley controllers with different gains is compared. The objective of the comparison is to determine the affection of gain $k$ on a Stanley Controller.

Figure 5.5, 5.6, 5.7 and 5.8 show the average lateral error and the maximal lateral error of the vehicle using the Stanley controller. The figures lead us to the conclusion that the gain $k$ of a lateral controller affects the lateral control performance greatly.

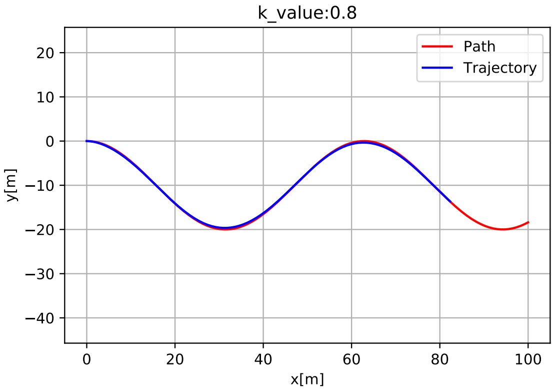


Figure 5.5: $k = 0.8$, average lateral error 0.32$m$

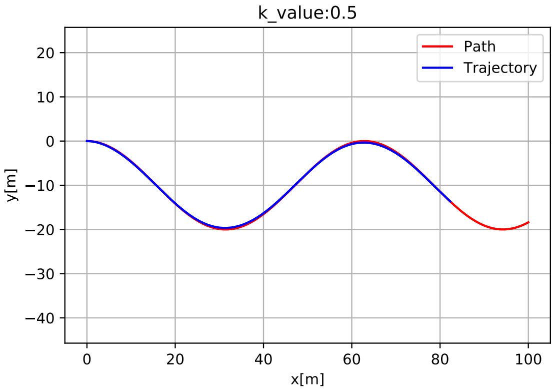


Figure 5.6: $k = 0.5$, average lateral error 0.38$m$

Comparing figure 5.5 with figure 5.6, the average lateral error increases by 14% when the gain $k$ drops from 0.8 to 0.5. The same relationship is reflected in the maximal lateral error, which increases by 5%. More average lateral errors by different gains are given in table 5.2.

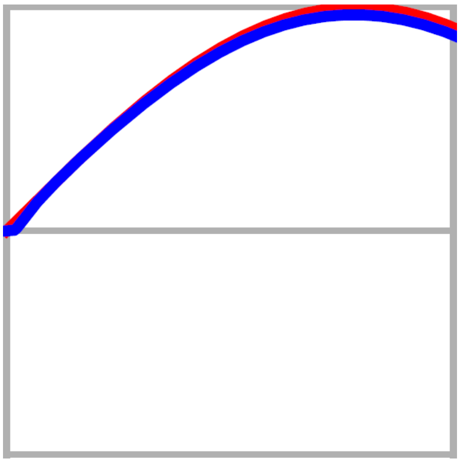


Figure 5.7: $k = 0.8$, maximal lateral error 0.71$m$

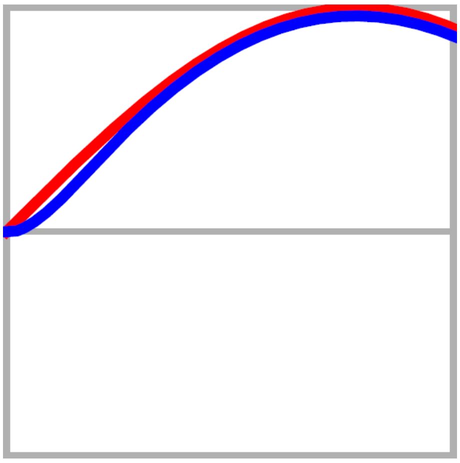


Figure 5.8: $k = 0.5$, average lateral error 0.74$m$

Figures 5.7 and 5.8 show that a greater gain leads to a quicker reaction of the lateral error. Both of the figures show the detailed information of Figures 5.5 and 5.6. It is clear that the Stanley controller with $k = 0.8$ reacts more quickly than the controller with $k = 0.5$. It can be explained with the basic form of the Stanley controller in equation 5.1: gain $k$ only affects the lateral error term. That means a greater gain leads to a greater error term. Therefore, the calculated steering angle is greater and the vehicle reacts more quickly to the lateral error.

To prove the hypothesis, it is necessary to design a Stanley controller with $k = 0$. This Stanley controller has only heading-errors feedback, which means it cannot react on any lateral errors. The result is shown in figure 5.9. Clearly, the vehicle moves along the form of the path, but always with a parallel offset. The cause of this phenomena is that the Stanley controller has no lateral error feedback and cannot react to the lateral error. Therefore, the hypothesis is proved: the greater the gain $k$, the better the control performance.

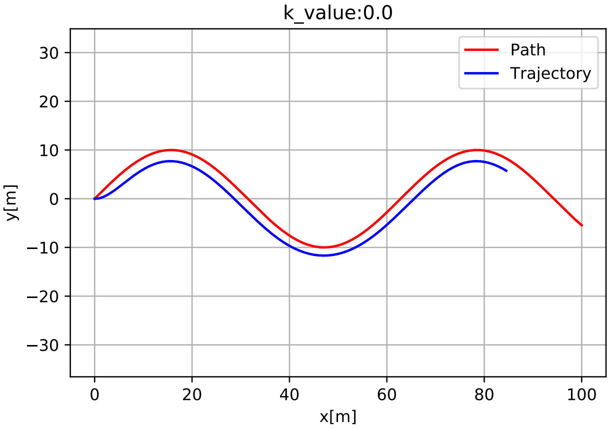


Figure 5.9: $k = 0$,only heading error feedback, without lateral error feedback

Table 5.2 shows the average and maximal lateral errors of Stanley controllers with potential gains in reference path 1. The relationship of lateral control performance and gains is not difficult to find: the greater the gain, the better the controller performs. Figure 5.10 shows the relationship more intuitively. Additionally, the system applying Stanley controller with $k \geq 0.9$ is not stable, the vehicle cannot follow the path with controllers with these gains.

Table 5.2: Comparison of Stanley controllers with different gains

| Gain $k$ | Average lateral error | Maximal lateral error | Variance |
|---|---|---|---|
| 0.0 | 1.98$m$ | 2.71$m$ | 0.190$m^2$ |
| 0.1 | 0.60$m$ | 1.64$m$ | 0.269$m^2$ |
| 0.2 | 0.49$m$ | 1.25$m$ | 0.138$m^2$ |
| 0.3 | 0.44$m$ | 1.04$m$ | 0.090$m^2$ |
| 0.4 | 0.40$m$ | 0.88$m$ | 0.068$m^2$ |
| 0.5 | 0.38$m$ | 0.74$m$ | 0.056$m^2$ |
| 0.6 | 0.36$m$ | 0.71$m$ | 0.050$m^2$ |
| 0.7 | 0.35$m$ | 0.71$m$ | 0.048$m^2$ |
| 0.8 | 0.32$m$ | 0.70$m$ | 0.054$m^2$ |
| 0.9 | 47.09$m$ | 96.914$m$ | 821.977$m^2$ |

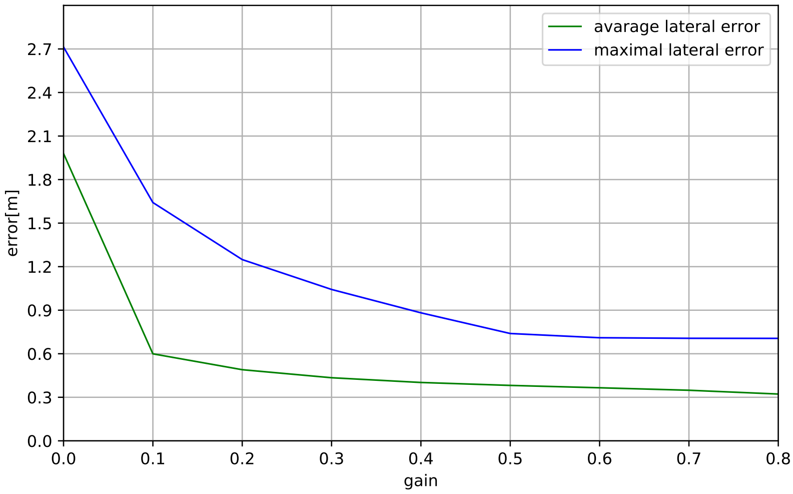


Figure 5.10: The relationship between gains and lateral errors of Stanley controllers

### 5.1.2 PID Controller

This section includes the performance and discussion of the PID controller with different inputs and gains. The principle and implementation was introduced in section 4.2.1. The basic formula of an incremental PID controller is given as

$$\begin{aligned}\Delta\delta &= K_p[e(k)-e(k-1)]+K_ie(k)+K_d[e(k)-2e(k-1)+e(k-2)],\\ \delta_k &= \delta_{k-1}+\Delta\delta_k,\end{aligned} \tag{5.2}$$

where $u_k$ is the computed steering angle and $e$ is either the lateral or heading error. There are three gains in the PID controller. Therefore, the modification of $K_p$, $K_i$ and $K_d$ is the major task of the controller tuning process.

#### Lateral error as input

The PID controller can use lateral error of vehicles as the input $e$. Figure 5.11 and 5.12 shows the trajectory of vehicle using PID controller with 2 different gain-set. The *gain-set 1* is $K_p = 1.0$, $K_i = 0.1$, $K_d = 1.0$ and the *gain-set 2* is $K_p = 3.0$, $K_i = 0.1$, $K_d = 5.0$.

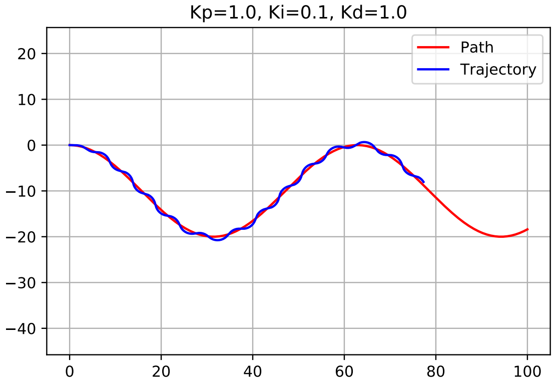


Figure 5.11: PID using lateral error with *gain-set 1*

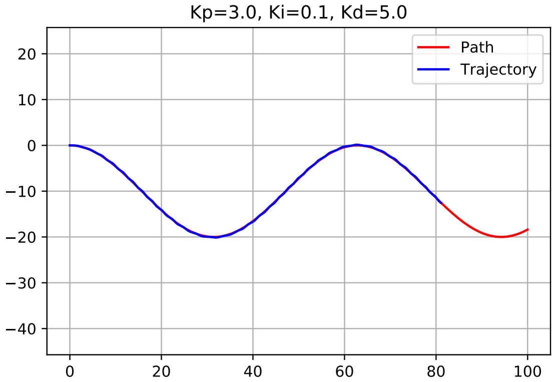


Figure 5.12: PID using lateral error with *gain-set 2*

As shown in the 2 figures, the PID controller using lateral error as input with the *gain-set 2* has already great lateral control performance. Table 5.3 expresses average and maximal lateral error, variance as well.

Table 5.3: Comparison of PID controllers using lateral errors with different gain-sets

| Gain-set | Average lateral error | Maximal lateral error | Variance |
|---|---|---|---|
| *Gain-set 1* | -0.013$m$ | 0.81$m$ | 0.185$m^2$ |
| *Gain-set 2* | -0.004$m$ | 0.15$m$ | 0.005$m^2$ |

Comparing figure 5.12 with 5.11, the PID controller has greater $K_p$ and $K_d$. As explained in section 2.3.1, a great $K_p$ causes a rapid response and a great $K_d$ reduces the effect of the error based on the rate of error change. For this reason, the trajectory in figure 5.12 is smoother than the trajectory in figure 5.11.

However, the controller performs not as expected. Figure 5.13 shows the lateral and heading error of the vehicle with the PID controller using the lateral error as input with the *gain-set 2*. The figure shows clearly that the lateral errors are already small values, mainly confined to $[-0.1, 0.1]$, but there is no indication that the lateral errors are a convergence. As shown in the figure, the lateral errors are permanent oscillation. Therefore, the system with PID only using the lateral error is not stable.

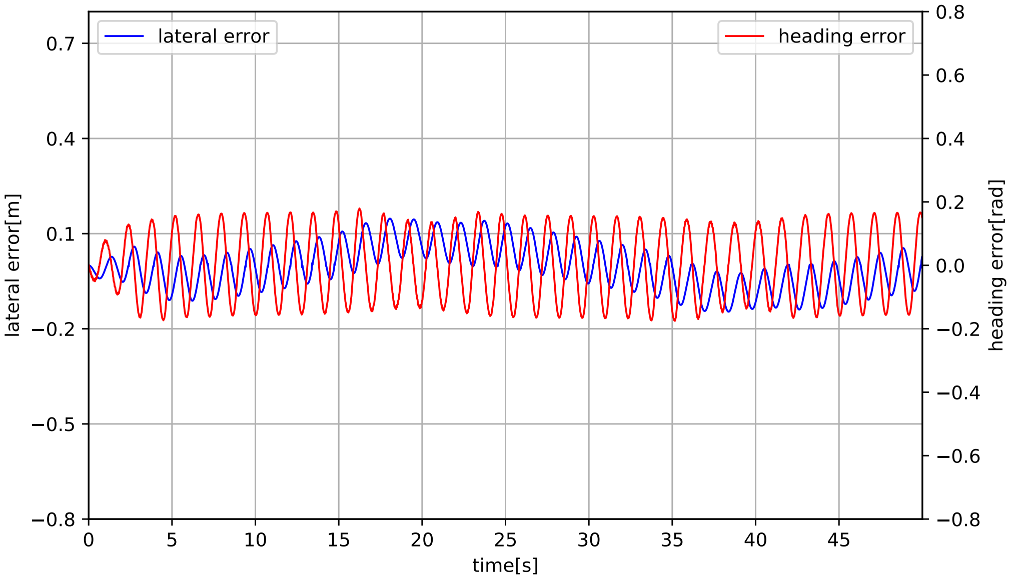


Figure 5.13: PID with *gain-set 2*, lateral and heading error

## Heading error as input

Additionally, the PID controller can also use heading error of vehicles as the input $e$. Figure 5.14 and 5.15 shows the trajectory of vehicle using PID controller based on heading errors with 2 different gain-set. The *gain-set 3* is $K_p = 2.0$, $K_i = 0.0$, $K_d = 2.0$ and the *gain-set 4* is $K_p = 2.0$, $K_i = 0.2$, $K_d = 2.0$.

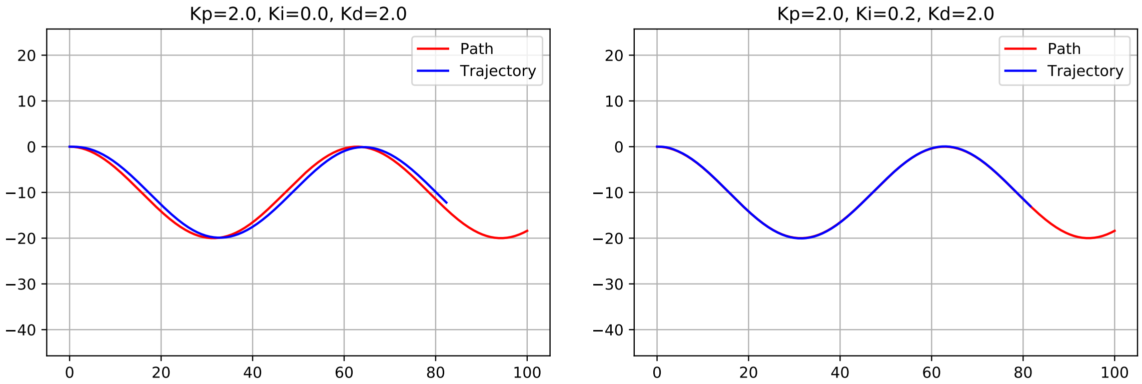


Figure 5.14: PID using heading error with *gain-set 3* Figure 5.15: PID using heading error with *gain-set 4*

Figure 5.15 shows that the PID controller using the heading error as the input has a significant control performance. Table 5.4 shows more detailed information on the control performance.

Table 5.4: Comparison of PID controllers using heading errors with different gain-sets

| Gain-set | Average lateral error | Maximal lateral error | Variance |
|---|---|---|---|
| *Gain-set 3* | -0.189$m$ | 1.10$m$ | 0.692$m^2$ |
| *Gain-set 4* | -0.007$m$ | 0.06$m$ | 0.001$m^2$ |

Comparing figure 5.15 with 5.14, the PID controller has greater $K_i$. As explained before, the I-item guaranteed the stationary accuracy of a system. For this reason, the vehicle in figure 5.15 moves more accurate along the path than the vehicle in figure 5.14.

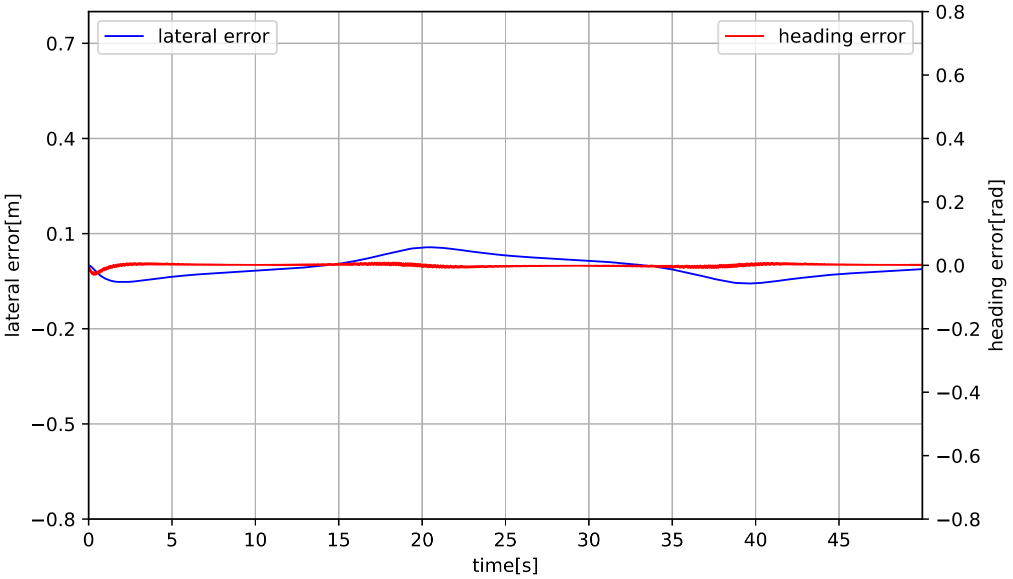


Figure 5.16: PID with *gain-set 4*, lateral and heading error

Figure 5.16 shows the lateral and heading error of the vehicle with the PID controller using the heading error as the input with the *gain-set 4*. Compared to figure 5.13, the PID controller using heading error has a better performance. However, it performs still not well facing the great curvatures.

## Lateral and Heading error as input

Figure 5.13 shows that the PID controller using lateral errors does not take the heading error into consideration, the vehicle has always heading errors. However, as shown in figure 5.16, the vehicle has almost no heading errors, but it has the greatest lateral error by the greatest curvature. Therefore, it is helpful to combine the advantages of two inputs of controllers. In this section, the PID controller using lateral and heading errors is introduced.

$$\delta = K_e \delta_e + K_{th} \delta_{th}, \tag{5.3}$$

$\delta_e$ and $\delta_{th}$ are computed as

$$\begin{aligned} \delta_e(k) &= \delta(k-1) + \Delta\delta_e(k), \\ \delta_{th}(k) &= \delta(k-1) + \Delta\delta_{th}(k). \end{aligned} \tag{5.4}$$

It is worth mentioning that by the computation of $\delta_e$ and $\delta_{th}$, the increments are directly added to former steering angle $\delta(k-1)$, not to $\delta_e(k-1)$ or $\delta th(k-1)$. Two gains $K_e$ and $K_t h$ in equation 5.3 can modify the participation's degree of $\delta_e$ and $\delta_{th}$, which can be helpful in adjusting to the multiple scenarios.

Figure 5.17 shows the trajectory of the vehicle using the PID controller using lateral and heading errors as inputs. As shown in th figure, the PID controller has good performance. Additionally, the example gain-set is given above the figure. More performance measurements are expressed in table 5.5.

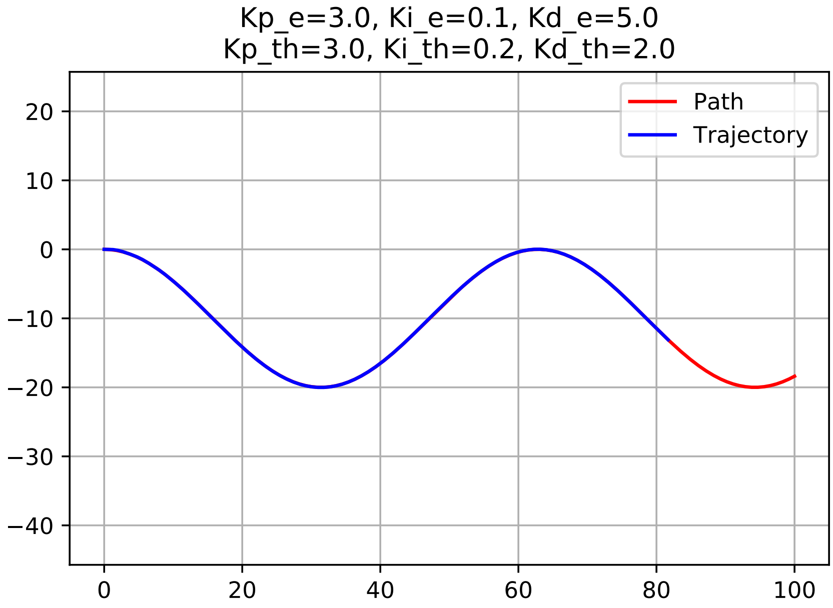


Figure 5.17: Trajectory of the vehicle with the PID controller using lateral and heading errors

Figure 5.18 shows the relationship between the vehicle steering angle and the path curvature. Same as Figure 5.3, the path curvature affects the steering angle most greatly.

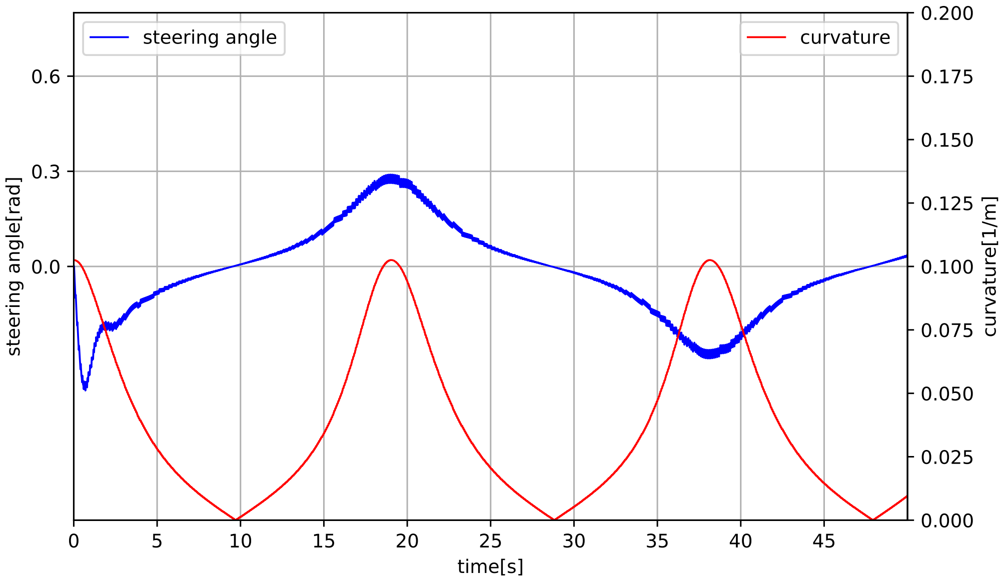


Figure 5.18: PID using lateral and heading errors, steering angle and curvature of the path

Figure 5.19 shows the lateral control performance of the PID controller. As shown in the figure, the PID controller has good performance and the system is stable and stationary accurate. Compared to Figure 5.16, the PID controller using combined errors has clearly less maximal lateral error and better control performance.

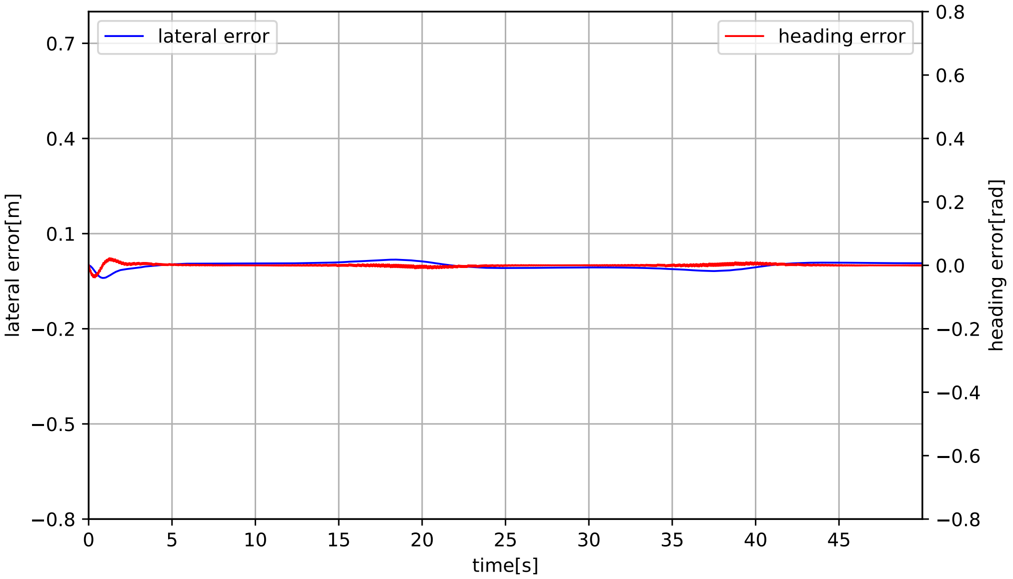


Figure 5.19: PID using lateral and heading errors, lateral control performance

Table 5.5 shows the performance measurements of the PID controller with different gains. As shown in the table, the PID controller using combined errors performs better than the PID controller using single error and the Stanley controller.

The PID controller using combined errors has totally 8 modifiable gains. In this work, the nested *for-loops* is used to find the optimal gain-set. There are other parameter tuning methods, which is introduced in section 6.2.

Table 5.5: Comparison of PID controllers using combined errors with different gains

| $K_e$ | $K_{th}$ | $Kp_e$ | $Ki_e$ | $Kd_e$ | $Kp_{th}$ | $Ki_{th}$ | $Kd_{th}$ | Average lateral error | Maximal lateral error | Variance |
|---|---|---|---|---|---|---|---|---|---|---|
| 0.5 | 0.5 | 2 | 0.2 | 2 | 2 | 0.2 | 2 | -0.0006$m$ | 0.1654$m$ | 0.0084$m^2$ |
| 0.5 | 0.5 | 2 | 0.2 | 2 | 2 | 0.2 | 3 | $2.9e^{-05}m$ | 0.1684$m$ | 0.0086$m^2$ |
| 0.5 | 0.5 | 2 | 0.2 | 2 | 2 | 0.2 | 4 | 0.0006$m$ | 0.1704$m$ | 0.0087$m^2$ |
| 0.5 | 0.5 | 2 | 0.2 | 2 | 2 | 0.3 | 2 | -0.0015$m$ | 0.1022$m$ | 0.0033$m^2$ |
| 0.5 | 0.5 | 2 | 0.2 | 2 | 2 | 0.3 | 3 | -0.0015$m$ | 0.1045$m$ | 0.0035$m^2$ |
| 0.5 | 0.5 | 2 | 0.2 | 2 | 2 | 0.3 | 4 | -0.0012$m$ | 0.1058$m$ | 0.0036$m^2$ |
| 0.5 | 0.5 | 2 | 0.2 | 2 | 2 | 0.4 | 2 | -0.0002$m$ | 0.04108$m$ | 0.0004$m^2$ |
| 0.5 | 0.5 | 2 | 0.2 | 2 | 2 | 0.4 | 3 | -0.0003$m$ | 0.0414$m$ | 0.0004$m^2$ |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

Here, the results and discussions of PID controllers are presented. As shown in the results, the PID controller performs a good path-tracking performance. Additionally, this performance also measures up to the expectations of accuracy of lateral controllers.

### 5.1.3 Pure Pursuit Controller

This section includes the results and discussions about the performance of the Pure Pursuit controller. The basic formula of the Pure Pursuit Controller is given as

$$\delta(t) = \tan^{-1}\left(\frac{2L\sin(\alpha(t))}{L_d}\right), \tag{5.5}$$

where look-ahead distance $L_d$ is the only modifiable parameter. Therefore, the $L_d$ property is the main tuning property for the controller. The $L_d$ describes how far along the path the vehicle should look from the current position to compute the angular velocity commands.

As shown in figure 5.20, the effect of changing this parameter can change how the vehicle tracks the path. There are two major goals: regaining the path and maintaining the path. In order to quickly regain the path between way points, a small $L_d$ will let the vehicle move quickly towards the path. However, as can be seen in the figure below, the vehicle overshoots the desired path and oscillates along it. In order to reduce the oscillations along the path, a larger $L_d$ should be chosen. However, this might result in larger curvatures near the corners.

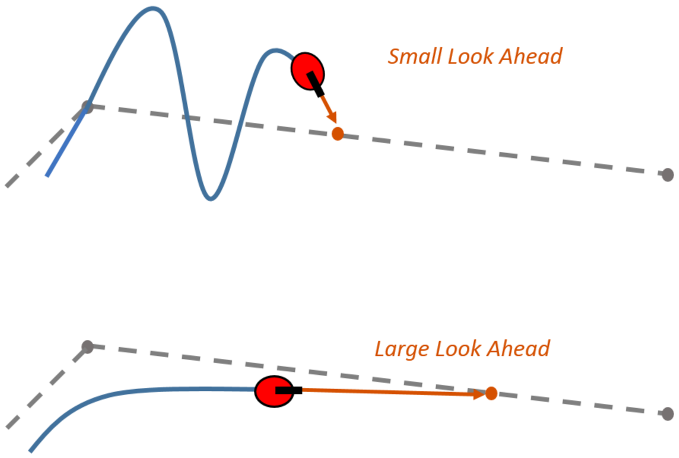


Figure 5.20: Pure Pursuit controller, affect of look-ahead distance $L_d$

Figures 5.21 and 5.22 show the lateral control performance of the Pure Pursuit controllers with different look-ahead distances. In principle, the smaller the look-ahead distance, the better the path-tracking performance. Here, the Pure Pursuit controller with $L_d = 3m$ performs better than the Pure Pursuit controller with $L_d = 10m$.

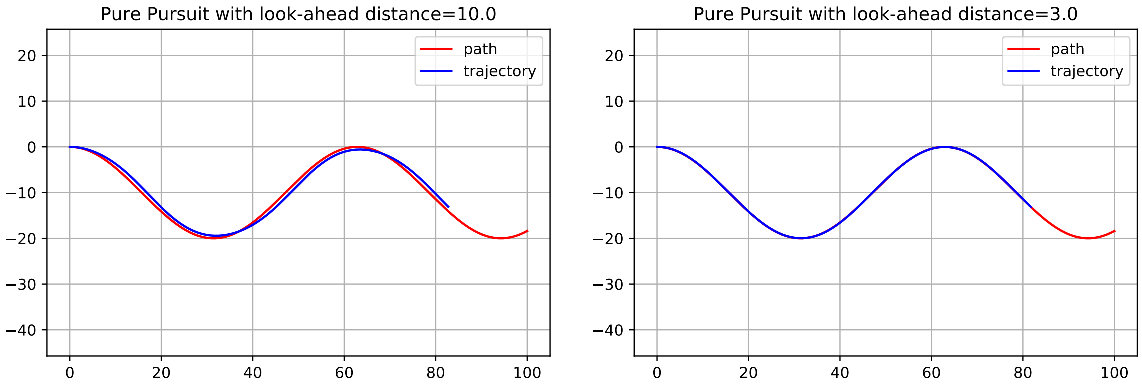


Figure 5.21: Pure Pursuit controller with $L_d = 10m$

Figure 5.22: Pure Pursuit controller with $L_d = 3m$

Figure 5.23 shows the relationship between the vehicle steering angle and the path curvature. Similar to the Stanley and PID controller, the Pure Pursuit controller computes a large steering angles for a large great path curvatures. The path curvature affects the steering angle most greatly.

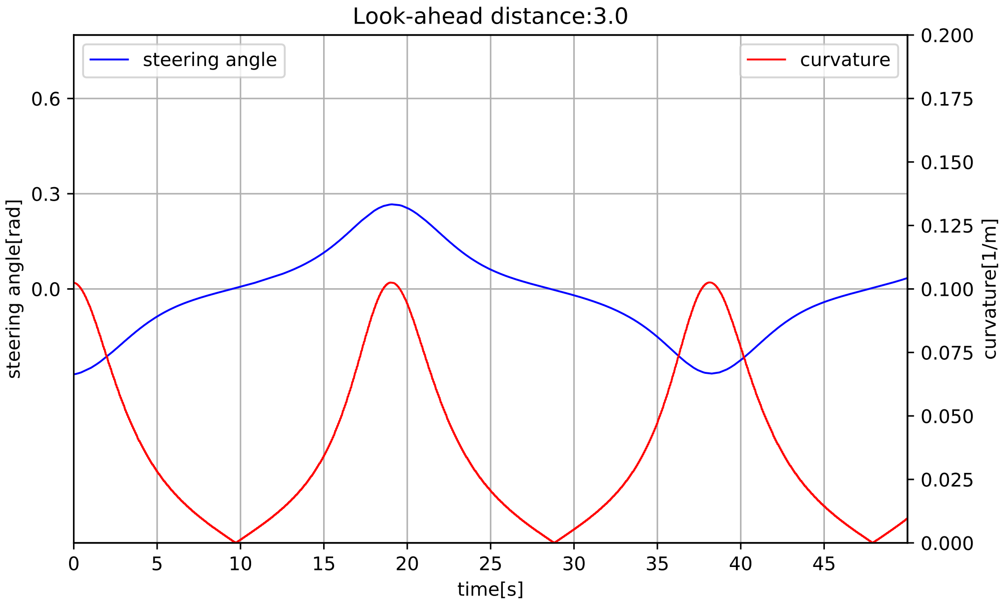


Figure 5.23: Pure Pursuit with $L_d = 3.0$, steering angle and curvature of the path

Figure 5.24 shows the lateral control performance of the Pure Pursuit controller using $L_d = 3m$. As shown in the figure, the Pure Pursuit controller has a good control performance. The Pure Pursuit controller has a smaller lateral error compared to the Stanley controller. However, compared to figure 5.19, the Pure Pursuit controller does not perform better than the PID controller using combined errors.

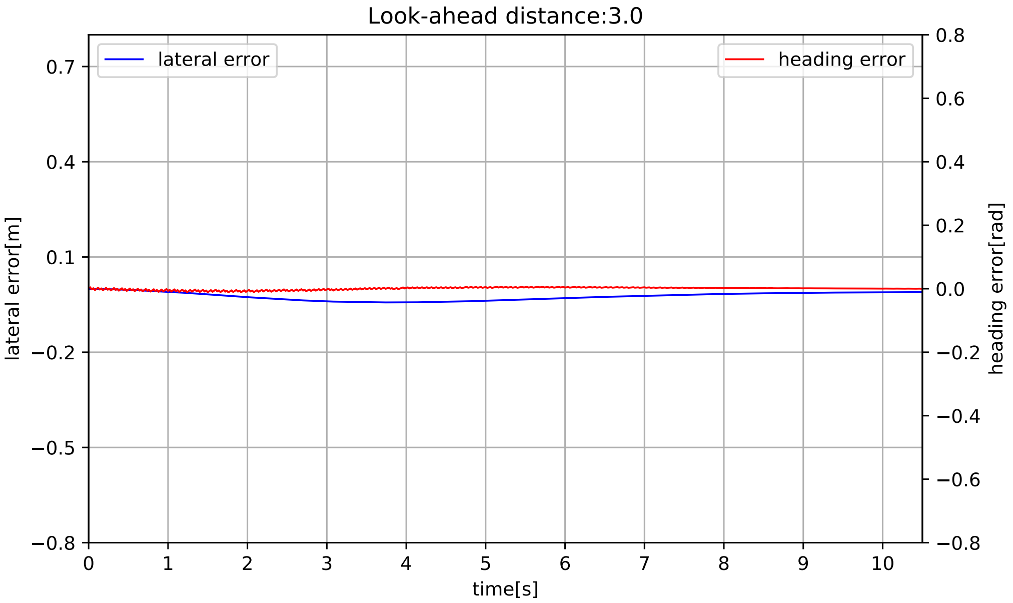


Figure 5.24: Pure Pursuit with $L_d = 3.0m$, lateral and heading errors

Table 5.6 shows the relationship between the lateral control performance and the look-ahead distance of the Pure Pursuit controller. Clearly, the Pure Pursuit controller has a better performance when it has a smaller look-ahead distance. Figure 5.25 is the visualization of the table 5.6. It shows the relationship more intuitively.

Table 5.6: Comparison of Pure Pursuit controllers using different look-ahead distances

| $L_d$ | Average lateral error | Maximal lateral error | Variance |
|---|---|---|---|
| 2.0$m$ | 0.005$m$ | 0.019$m$ | $1.467e^{-5}m^2$ |
| 2.5$m$ | 0.011$m$ | 0.027$m$ | $4.031e^{-5}m^2$ |
| 3.0$m$ | 0.095$m$ | 0.041$m$ | $10.2e^{-5}m^2$ |
| 3.5$m$ | 0.029$m$ | 0.062$m$ | 0.0002$m^2$ |
| 4.0$m$ | 0.044$m$ | 0.088$m$ | 0.0004$m^2$ |
| 4.5$m$ | 0.063$m$ | 0.120$m$ | 0.0008$m^2$ |
| 5.0$m$ | 0.085$m$ | 0.158$m$ | 0.0014$m^2$ |
| 5.5$m$ | 0.112$m$ | 0.202$m$ | 0.0023$m^2$ |
| 6.0$m$ | 0.144$m$ | 0.251$m$ | 0.0036$m^2$ |
| 6.5$m$ | 0.179$m$ | 0.120$m$ | 0.0053$m^2$ |
| 7.0$m$ | 0.220$m$ | 0.367$m$ | 0.0076$m^2$ |
| 7.5$m$ | 0.265$m$ | 0.433$m$ | 0.0107$m^2$ |
| 8.0$m$ | 0.316$m$ | 0.504$m$ | 0.0150$m^2$ |
| 8.5$m$ | 0.371$m$ | 0.581$m$ | 0.0206$m^2$ |
| 9.0$m$ | 0.432$m$ | 0.664$m$ | 0.0280$m^2$ |
| 9.5$m$ | 0.497$m$ | 0.751$m$ | 0.0375$m^2$ |
| 10.0$m$ | 0.568$m$ | 0.844$m$ | 0.0497$m^2$ |
| 10.5$m$ | 0.643$m$ | 0.943$m$ | 0.0651$m^2$ |
| 11.0$m$ | 0.723$m$ | 1.046$m$ | 0.0842$m^2$ |

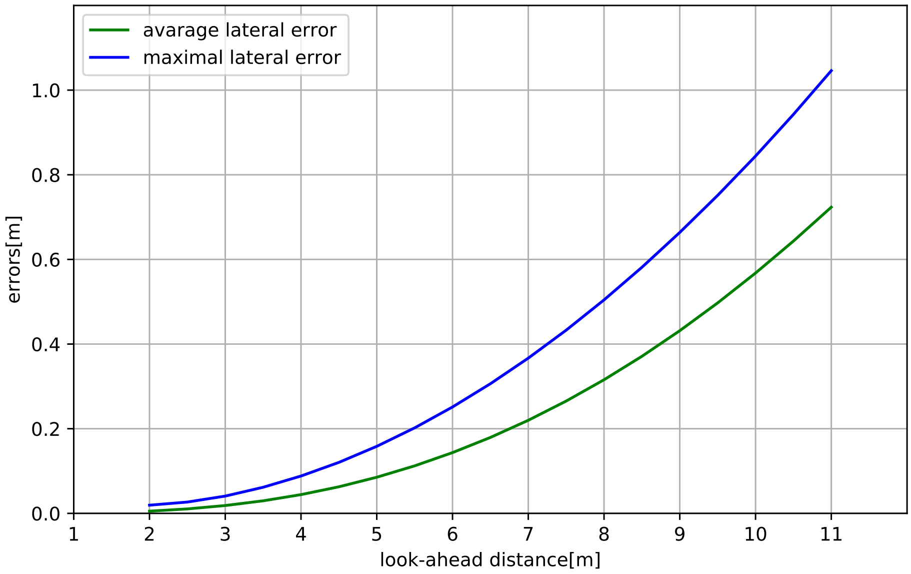


Figure 5.25: The relationship between $L_d$ and lateral errors of Pure Pursuit controllers

The results of the experiments show clearly that the look-ahead distance and the path curvature affects the performance of the Pure Pursuit controller greatly. Therefore, using a adaptive look-ahead distance computed from the current path curvature, might increase the lateral control performance and decrease the variance of lateral errors. The Pure Pursuit controller using adaptive look-ahead distance is introduced in the next section.

### 5.1.4 Adaptive Pure Pursuit Controller

As explained in the last section, the Pure Pursuit controller using an adaptive look-ahead distance might have a better control performance. As shown in the figures in the last section, the lateral errors and steering angles are greatly influenced by the path curvature and the look-ahead distance. Therefore, it is necessary to determine the relationship between the lateral errors, the path curvatures and the look-ahead distances. Next, the optimal look-ahead distance can be computed from the current path curvature and the desired lateral offset.

By use of path 3 in table 5.2, the vehicle can move along the path having constant curvatures. Using two-level nested *for-loops*, the three values $L_d$, $K_r$ and $e$ are respectively measured and stored. These values build a curved surface shown in figure 5.26 in the three dimensional space. As shown in the figure, the greater the curvature and the look-ahead distance, the greater the lateral error.

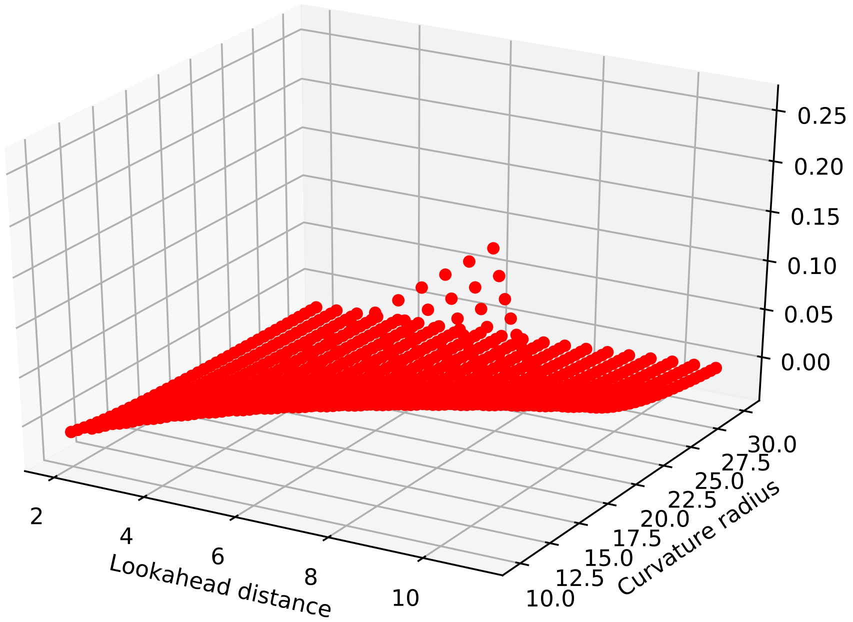


Figure 5.26: Relationship between lateral errors, path curvatures and look-ahead distances

Furthermore, this curved surface can be fitted. The iterative curved surface fitting algorithm can be found in the Appendix. The fitted curved surface is given as

$$e = k_1 L_d + k_2 K_r + k_3, \tag{5.6}$$

and the adaptive look-ahead distance is expressed as

$$L_d = \frac{k_2 K_r + k_2 - e}{k_1}, \tag{5.7}$$

where the gains $k_1$, $k_2$ and $k_3$ are a priori determined before the vehicle goes into service. The look-ahead distance should be computed every time before it is applied in the Pure Pursuit control algorithm.

The determined gains of the vehicle in this experiment are given in table 5.7.

Table 5.7: Gains of the adaptive look-ahead distance

| $k_1$ | $k_2$ | $k_3$ |
|---|---|---|
| 0.00857346438284748 | -0.004910809672486602 | 0.0555105524581532 |

Figure 5.27 shows the lateral control performance of the Pure Pursuit controller with the respective adaptive look-ahead distance. Compared to the pure pursuit controller with $L_d = 3$ in figure 5.24, the Pure Pursuit controller with the adaptive look-ahead distance does not have a better performance. Clearly, an adaptive look-ahead distance is not better than an appropriate constant look-ahead distance.

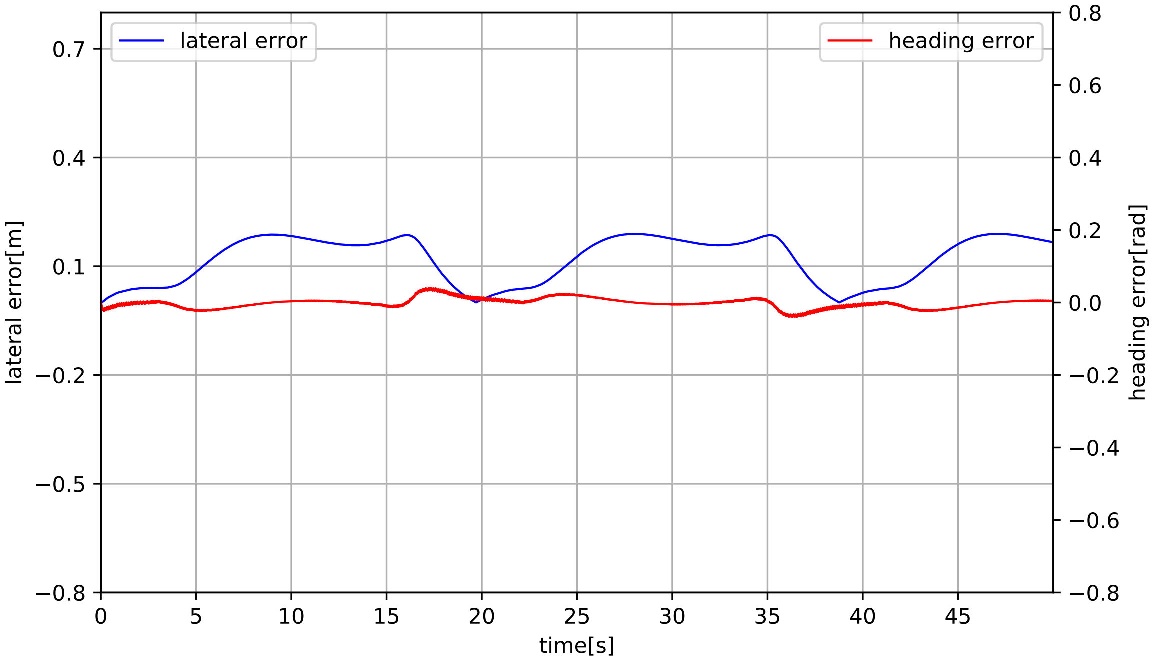


Figure 5.27: Pure Pursuit controller with adaptive $L_d$, lateral and heading errors

Additionally, the Pure Pursuit controller with the adaptive look-ahead distance is unrealistic. It is inconvenient to determine the gains $k_1$, $k_2$ and $k_3$. The gains can also change and be incorrect because of a tiny change of the working environment.

### 5.1.5 Combined Pure Pursuit - PID Controller

As explained in section 4.3, combining the Pure Pursuit and the PID controller might increase the control performance. The combined controller is called Pure Pursuit - Proportional Integral Differentiation (PP-PID) controller. The principle of combined controllers is introduced in section 4.3, the steering angle is given as

$$\delta = K_{PP}\delta_{PP} + K_{PID}\delta_{PID}, \tag{5.8}$$

where $\delta_{PP}$ is the steering angle computed by the Pure Pursuit controller introduced in section 5.1.3, $\delta_{PID}$ is the steering angle computed by the PID controller introduced in section 5.1.2. It is worth mentioning that $K_{PP}$ and $K_{PID}$ are the weights used to control the each controller.

There are a total 11 modifiable gains and weights of the PP-PID controller. The applied gains in these experiments are given in table 5.8.

Table 5.8: Gains of the Pure Pursuit-PID controller

| $K_e$ | $K_{th}$ | $Kp_e$ | $Ki_e$ | $Kd_e$ | $Kp_{th}$ | $Ki_{th}$ | $Kd_{th}$ | $L_d$ | $K_{PP}$ | $K_{PID}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0.5 | 0.5 | 3.0 | 0.1 | 5.0 | 3.0 | 0.2 | 2.0 | 3.0$m$ | 0.5 | 0.5 |

As shown in figure 5.28, the vehicle using the PP-PID controller has almost no lateral and heading error. It can accurately move along the desired path. The maximal error is $0.026m$ for curvature radius $10m$ and the average lateral error is $-0.0048m$, which meet the requirement of lateral controllers.

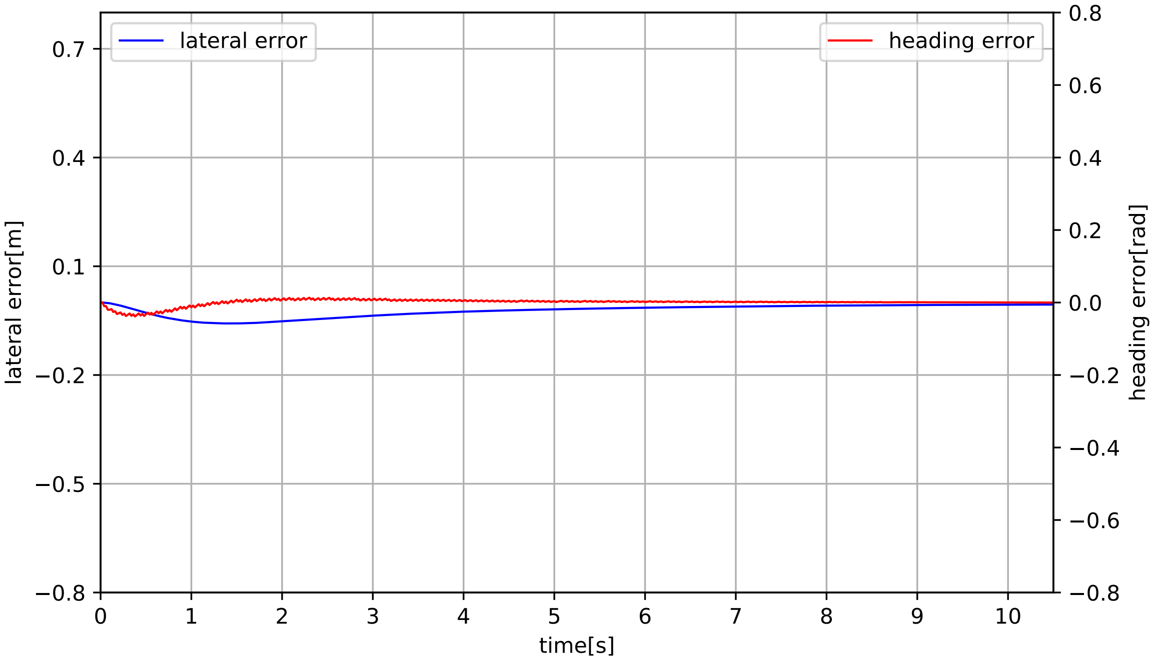


Figure 5.28: PP-PID controller, lateral and heading errors

Compared to the PID controller using the same gains as in figure 5.19, the PP-PID controller has no oscillation at the start of the experiment. This is the advantage of the Pure Pursuit controller. Additionally, the Pure Pursuit controller shown in figure 5.24 can never exactly follow direct paths between way points, it always has positive lateral errors. However, Use of the property of the PID controller can compensate for it effectively. Therefore, the PP-PID controller has almost no lateral error.

## 5.2 Real Route

The path 1 is sinusoidal, which has the enormous curvature change and other path information. However, a sinusoidal path is not enough representative. To check the plausibility, the controller implemented must also track a real route as reference path. Therefore, a real route is chosen in Google Maps. As shown in figure 5.29, the route chosen is in Garching near Munich, where the TUM Garching campus is located.

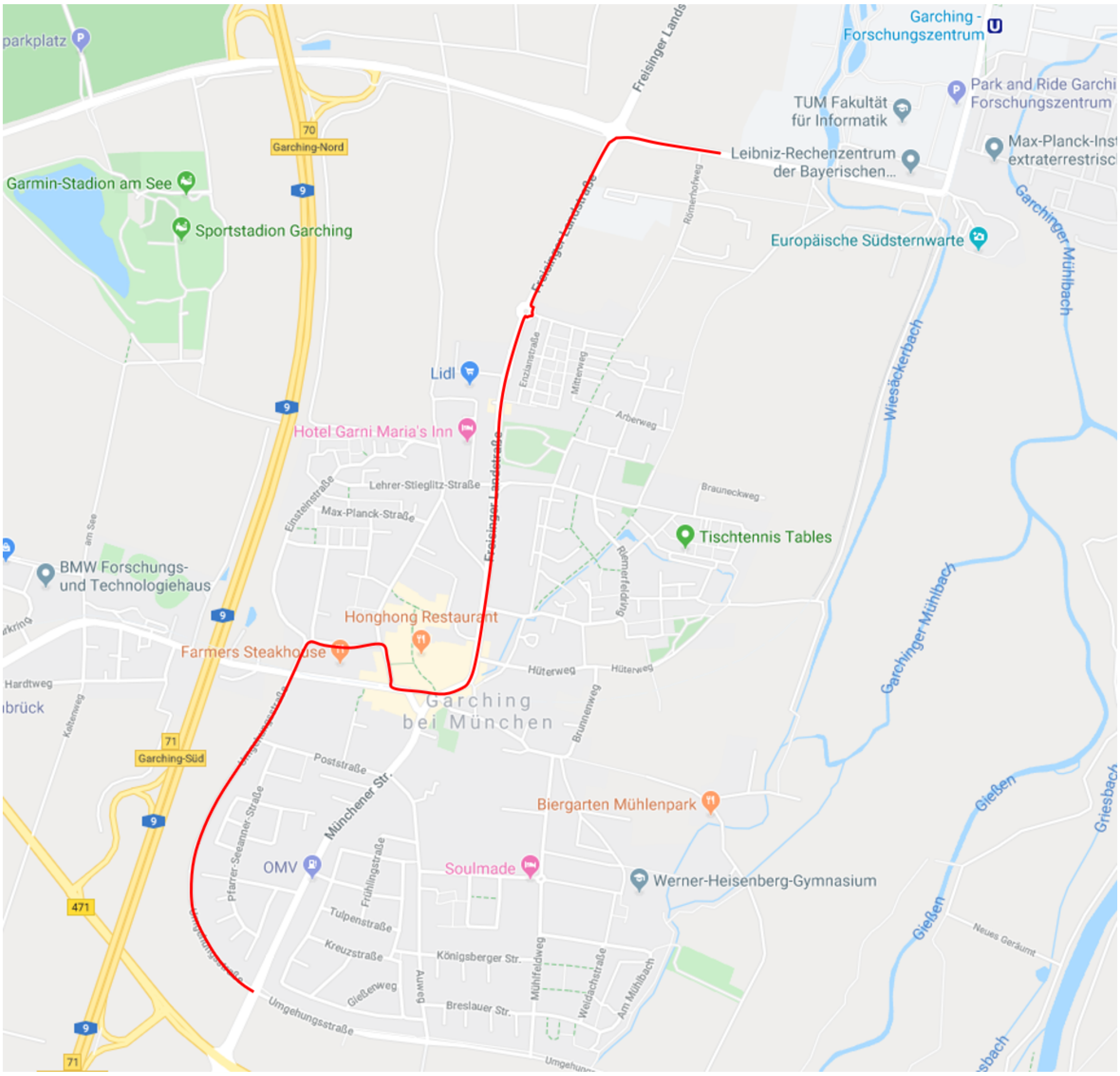


Figure 5.29: Real route in Garching near Munich

The route is in total $3.46km$ long. It includes country road, roundabout traffic, route with great curvature and many other representative information. By use of the special software at FTM, the route can be discretized in more than 17000 path points. The position of these path points are given by the software too. The distance between two path points is $0.1m$, which is greater as expected. Use of minimal distance method introduced in section 4.1.4 can effectively reduce the error and increase the control accuracy.

Figure 5.30 presents the reconstructed real route in the simulation. Figure 5.31 shows the performance of the PP-PID controller on the real route. Generally, the PP-PID controller has a good performance in the path-following maneuver.

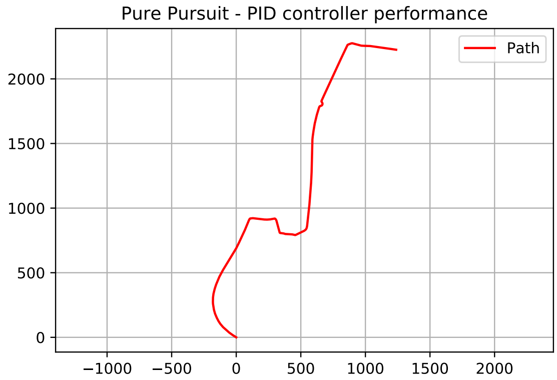


Figure 5.30: Real route path

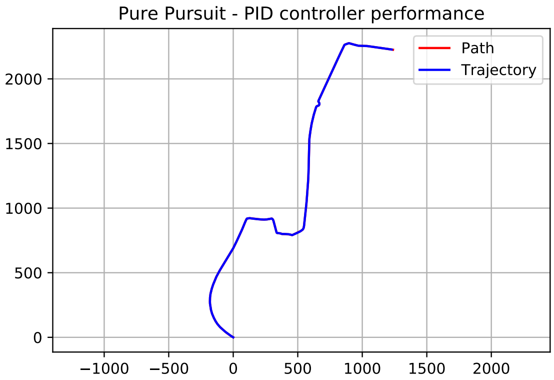


Figure 5.31: Real route PP-PID performance

As shown in figure 5.32, the PP-PID controller performs well in path following maneuver. It has almost no lateral error greater than $0.1m$ and heading error greater than $0.1$ rad. The maximal lateral error is $0.0966m$ and the average error is $0.0037m$. Variance is 0.0002, which meets the requirement too.

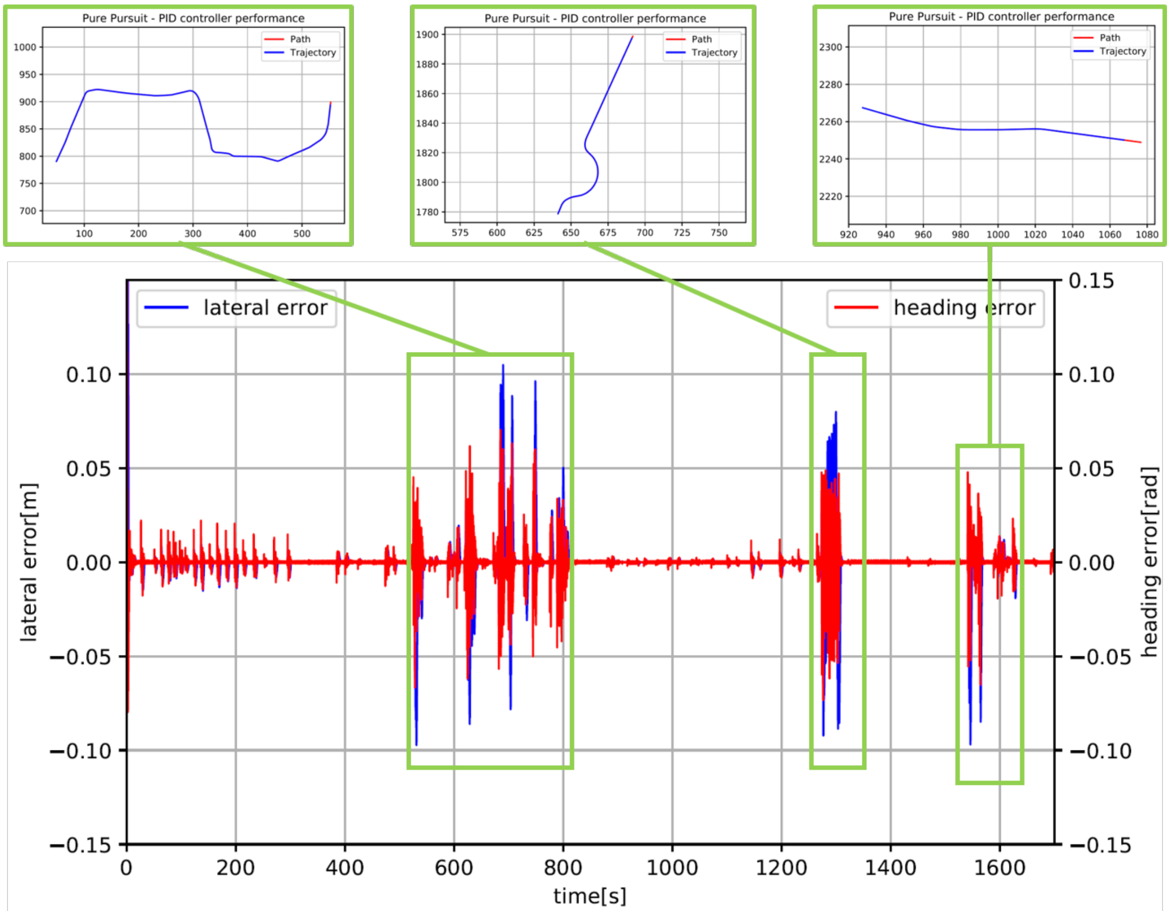


Figure 5.32: PP-PID on the Real Route, lateral and heading errors

The relationship between the steering angle and the path curvature is shown in figure 5.33. The path curvature is still the biggest influence factor for the steering angle of a PP-PID controller.

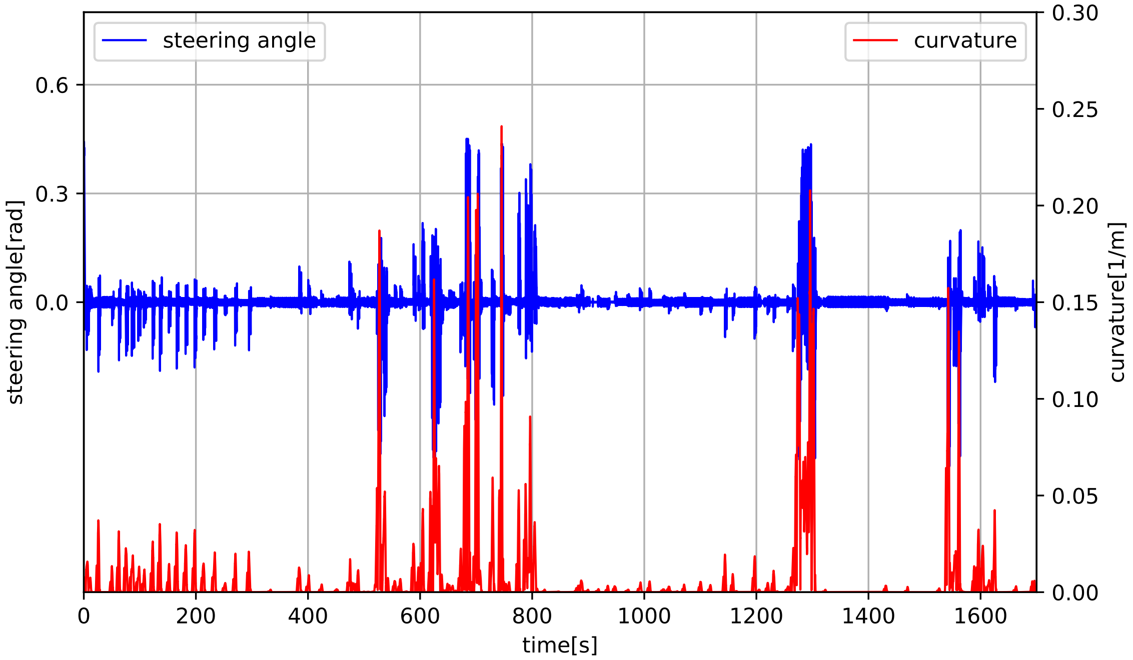


Figure 5.33: PP-PID on the Real Route, steering angle and path curvature

The results in this section proves that the PP-PID controller can let the vehicle move along the real route with high accuracy. The lateral errors and heading errors keep in the low level, which meets the requirement of the path-following controller.

## 5.3 Conclusion

The performance of all controllers implemented in this thesis are presented in the previous section. This section provides an overview and a comparison of these controllers.

Tabel A.1 presents the controller types and corresponding lateral errors and variances. The smaller the maximal and lateral errors, the better the control performance. Additionally, the variance is a significant performance measurement. Variance is the expectation of the squared deviation of a random variable from its mean. Here, the smaller the variance, the more stable the control system.

Table 5.9: Comparison of the performance of different controllers

| Overview of the lateral errors of different controllers | | | |
|---|---|---|---|
| Controller | Average lateral error | Maximal lateral error | Variance |
| Stanley | 0.332$m$ | 0.705$m$ | 1.467$e^{-5}m^2$ |
| PID using lateral error | 0.0036$m$ | 0.1524$m$ | 0.0050$m^2$ |
| PID using heading error | 0.0065$m$ | 0.0575$m$ | 0.0010$m^2$ |
| PID using combined errors | -0.0004$m$ | 0.0175$m$ | 9.999$e^{-5}m^2$ |
| Pure Pursuit | 0.0052$m$ | 0.0193$m$ | 1.467$e^{-5}m^2$ |
| Adaptive Pure Pursuit | 0.1191$m$ | 0.1895$m$ | 0.0041$m^2$ |
| Pure Pursuit - PID | -0.0048$m$ | 0.0258$m$ | 0.0001$m^2$ |

As shown in the table, the PID controller using combined errors has a better performance than the PP-PID controller. However, the PID controller using combined errors shown in figure 5.19 displays an oscillation at the begin of the experiment. Such oscillation cannot happen in the vision-based control system. Because if the vehicle begins to oscillate, the camera might not be able to record the lane marking anymore. Therefore, the PP-PID controller is finally chosen as the controller applied in the service.

It is worth mentioning that the parameters and gains of above controllers have not been optimally tuned. Because of the significant amount of the control parameters, the tuning of the gains are a herculean challenge. By use of Machine Learning techniques such as reinforcement learning, the optimal control parameters can be automatically found[42].

Additionally, the controllers implemented in this thesis are based on the kinetic single-track model introduced in section 3.1.1. Use of the controllers based on the dynamic model introduced in section 3.1.2, such as MPC, might increase the lateral control performance[43].

# 6 Overview and Outlook

In this chapter, also the last chapter of the thesis, a short summary and some continuing concepts and work regarding the presented lateral control methods are given. Section 6.1 is a summary of the thesis, it includes my personal perspective based on six months working experience on state-of-the-art lateral control technology. Section 6.2 includes some modern control strategies which are due to limited time and lack of prior knowledge have not been researched and implemented in the project.

## 6.1 Summary of the Thesis

With the rapid developments of computer science, robotics, automatic control, and artificial intelligence, autonomous and teleoperated driving has attracted more and more focus in recent years. Autonomous and teleoperated driving entails substantial advantages for industrial, public, and private transport. Its success is therefore in general interest.

The objective of this thesis is to develop a lateral controller which can compute the steering angle automatically and let the vehicle move along the planned path accurately. This work presented an algorithmic solution to tackle this issue.

Before getting into details of the lateral control method, first it is necessary to delimit the method from related topics and tasks. This work does not include image processing and lane detection approaches, since another student in project SToRM is searching for solutions in these areas simultaneously[2]. Because a lane detection module is available, the curve and its fitted polynomial of lanes were chosen as inputs for the lateral controller.

An accurate vehicle model can improve the lateral control performance substantially. There are two vehicle models available in this work, kinematic model and dynamic model. Basically, a dynamic model takes the slip of the tires into consideration, that's the reason why it can describe a vehicle motion more accurately. However, the slip of tires can be neglected if a vehicle moves slowly. It is also necessary to linearize the vehicle model, because the vehicle model is a nonlinear system.

In the thesis, there are a variety of lateral controllers introduced. There are three controllers implemented: Pure Pursuit controller, Stanley controller and PID controller. The control parameters and gains can be modified via the control performance, namely the lateral error. It is also helpful to combine controllers to improve the accuracy.

All vehicles without rear wheel steering suffer from a nonholonomic constraint. In the thesis, the PID controller using lateral error did not perform as expected. The nonholonomic constraint of the vehicle may be the mostly responsible.

## 6.2 Future works

The lateral control method presented in this thesis was planned, designed, and implemented within the six-month limitation of a bachelor thesis. It is worth mentioning that additional works are required to improve the performance of controllers and human machine interface. The future works from the developer's perspective mainly include

- Model Predictive Control (MPC)
  An basic introduction is already given in the thesis. However, there was not enough time to design and implement a MPC for this project. Clearly, a well-tuned MPC is one of the best performed lateral controllers. It has not only good contol performance, but also robustness in comparison with Stanley. It is worth investing time and effort on MPC.
- Machine Learning (ML):
  ML can be used to tune controllers. In this work, *for-loops* and *nested for-loops* are used to find the best look-ahead distance for Pure Pursuit and the best gain for Stanley and PID. By use of the ML approach, it is possible to compute the relationship between the local curvature of paths and the look-ahead distance much more accurately than when computed by *for-loops*. Look-ahead distance is the most significant property of Pure Pursuit. Once the adaptive Look-ahead distance is correctly chosen, the Pure Pursuit can also show good performance.

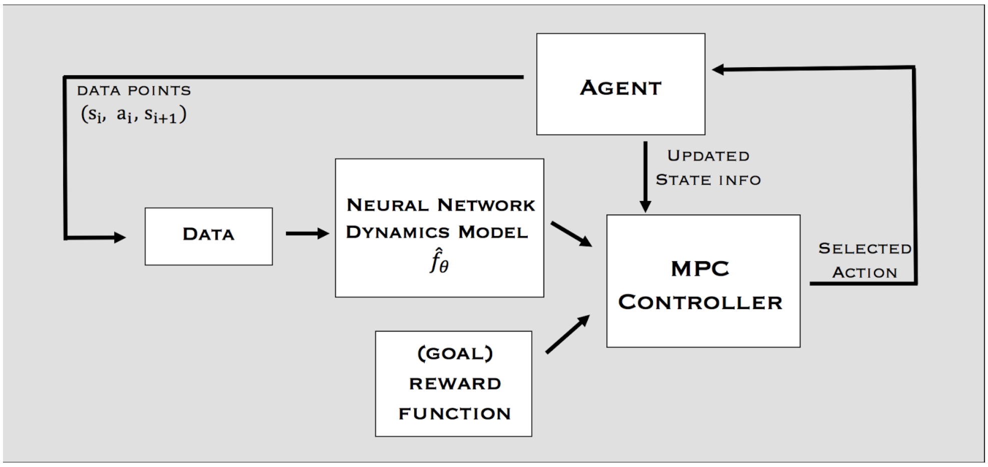


Figure 6.1: A control loop of model-based reinforcement learning algorithm[44]

- Human-Machine-Interaction (HMI):
  There are several controllers implemented in this work. Basically, users can only make changes in the ROS subscriber node to switch the controller types. It is also complicated if users want to modify the control parameters. It would be better when a graphical user interface (GUI) is available to modify controllers. A GUI is familiar to many people and intuitively clear. It can improve the human-machine-interaction efficiently.

# List of Figures

Figure 1.1: Lane marking machine in application scenario .... 2
Figure 1.2: Structure of this thesis .... 4
Figure 2.1: The six levels of vehicle autonomy[4] .... 5
Figure 2.2: VaMP and Audi Q7[6] .... 6
Figure 2.3: Sandstorm, Boss and Stanley[7] .... 6
Figure 2.4: The sensors of an IUV[8] .... 7
Figure 2.5: The software architecture of an autonomous vehicle[9] .... 8
Figure 2.6: A driver-vehicle-environment control loop [10] .... 9
Figure 2.7: A combined lateral and longitudinal control loop[11] .... 9
Figure 2.8: A PID block diagram of automatic steering system[13] .... 10
Figure 2.9: A block diagram of a PID controller in a feedback loop[14] .... 11
Figure 2.10: Stanley method geometry[17] .... 12
Figure 2.11: Geometric Single-Track Model[17] .... 13
Figure 2.12: Pure Pursuit Method Geometry[17] .... 14
Figure 2.13: A discrete model predictive control scheme[24] .... 16
Figure 3.1: Front wheel of a car-like robot[17] .... 19
Figure 3.2: A kinematic single track model[21] .... 20
Figure 3.3: A dynamic single-track model[21] .... 23
Figure 3.4: Nodes and topics in ROS[39] .... 26
Figure 3.5: Master and nodes in ROS[40] .... 26
Figure 3.6: A simulation scenario of in SILAB .... 27
Figure 3.7: camera positioning in SILAB .... 27
Figure 3.8: Services in ROS[40] .... 28
Figure 3.9: Publishing and subscribing in ROS[40] .... 28
Figure 4.1: Detected lanes in the ego-centric frame .... 30
Figure 4.2: A cartesian coordinate transformation in 2 dimension .... 31
Figure 4.3: The global path in the world coordinate system .... 31
Figure 4.4: The local path in the world coordinate system .... 32

Figure 4.5: The local path in the ego-centric frame ... 32

Figure 4.6: Vehicle position between [i-1] and [i] ... 33

Figure 4.7: Vehicle position between [i] and [i+1] ... 33

Figure 4.8: First in-first out in the queueing theory ... 34

Figure 4.9: Control loop implemented with a computer[41] ... 35

Figure 4.10: $\theta_e(t)$ and $e(t)$ in Stanley control[21] ... 37

Figure 4.11: A Pure Pursuit geometry in the vehicle ego-centric frame ... 39

Figure 4.12: A simplified path geometry after rotation ... 40

Figure 4.13: The Structure of a combined Pure Pursuit - PI controller[19] ... 41

Figure 4.14: A geometry relationship of look-ahead error $e_{la}$ ... 41

Figure 4.15: A graph including nodes and topics in ROS ... 43

Figure 4.16: Process of lane marking detection ... 43

Figure 4.17: A rqt_topic window in the experiment in SILAB ... 44

Figure 5.1: Reference path 1 ... 45

Figure 5.2: Reference path 2 ... 45

Figure 5.3: Stanley with $k = 0.8$, steering angle and curvature of the path ... 46

Figure 5.4: Stanley with $k = 0.8$, lateral error and heading error ... 46

Figure 5.5: $k = 0.8$, average lateral error 0.32$m$ ... 47

Figure 5.6: $k = 0.5$, average lateral error 0.38$m$ ... 47

Figure 5.7: $k = 0.8$, maximal lateral error 0.71$m$ ... 47

Figure 5.8: $k = 0.5$, average lateral error 0.74$m$ ... 47

Figure 5.9: $k = 0$,only heading error feedback, without lateral error feedback ... 48

Figure 5.10: The relationship between gains and lateral errors of Stanley controllers ... 48

Figure 5.11: PID using lateral error with *gain-set 1* ... 49

Figure 5.12: PID using lateral error with *gain-set 2* ... 49

Figure 5.13: PID with *gain-set 2*, lateral and heading error ... 50

Figure 5.14: PID using heading error with *gain-set 3* ... 50

Figure 5.15: PID using heading error with *gain-set 4* ... 50

Figure 5.16: PID with *gain-set 4*, lateral and heading error ... 51

Figure 5.17: Trajectory of the vehicle with the PID controller using lateral and heading errors ... 52

Figure 5.18: PID using lateral and heading errors, steering angle and curvature of the path ... 52

Figure 5.19: PID using lateral and heading errors, lateral control performance ... 53

Figure 5.20: Pure Pursuit controller, affect of look-ahead distance $L_d$ ... 54

Figure 5.21: Pure Pursuit controller with $L_d = 10m$ ........ 54
Figure 5.22: Pure Pursuit controller with $L_d = 3m$ ........ 54
Figure 5.23: Pure Pursuit with $L_d = 3.0$, steering angle and curvature of the path ........ 55
Figure 5.24: Pure Pursuit with $L_d = 3.0m$, lateral and heading errors ........ 55
Figure 5.25: The relationship between $L_d$ and lateral errors of Pure Pursuit controllers 56
Figure 5.26: Relationship between lateral errors, path curvatures and look-ahead distances ........ 57
Figure 5.27: Pure Pursuit controller with adaptive $L_d$, lateral and heading errors ........ 58
Figure 5.28: PP-PID controller, lateral and heading errors ........ 59
Figure 5.29: Real route in Garching near Munich ........ 60
Figure 5.30: Real route path ........ 61
Figure 5.31: Real route PP-PID performance ........ 61
Figure 5.32: PP-PID on the Real Route, lateral and heading errors ........ 61
Figure 5.33: PP-PID on the Real Route, steering angle and path curvature ........ 62
Figure 6.1: A control loop of model-based reinforcement learning algorithm[44] ........ 66
Figure A.1: Stanley controller performance ........ xv
Figure A.2: Pure Pursuit controller performance ........ xv
Figure A.3: PID controller performance ........ xv
Figure A.4: PP-PID controller performance ........ xv

# List of Tables

Table 2.1: Comparison and Evaluation of sensor system ........ 7
Table 5.1: Used vehicle parameters in experiments ........ 45
Table 5.2: Comparison of Stanley controllers with different gains ........ 48
Table 5.3: Comparison of PID controllers using lateral errors with different gain-sets ... 49
Table 5.4: Comparison of PID controllers using heading errors with different gain-sets 50
Table 5.5: Comparison of PID controllers using combined errors with different gains ... 53
Table 5.6: Comparison of Pure Pursuit controllers using different look-ahead distances 56
Table 5.7: Gains of the adaptive look-ahead distance ........ 58
Table 5.8: Gains of the Pure Pursuit-PID controller ........ 59
Table 5.9: Comparison of the performance of different controllers ........ 63
Table A.1: Comparison of the performance of different controllers in path 2........ xv

# Bibliography

[1] Statistisches Bundesamt. „*Traffic accidents in Germany*,“ 2012. Available: https://www.destatis.de/EN/Themes/Society-Environment/Traffic-Accidents/_node.html.

[2] Bömelburg-Zacharias, „Development and Implementation of a Deep Learning Algorithm for Detection of Lane Markings,“ Master Thesis, Chair of Automotive Technology, Technical University of Munich, Munich, 2019.

[3] T. Tang, P. Vetter, S. Finkl, K. Figel and M. Lienkamp, „Teleoperated Road Vehicles – The "Free Corridor" as a Safety Strategy Approach,“ *Applied Mechanics and Materials*, vol. 490-491, pp. 1399–1409, 2014, DOI: 10.4028/www.scientific.net/AMM.490-491.1399.

[4] Synopsys. „*The six levels of vehicle autonomy*,“ 2019. Available: https://www.synopsys.com/automotive/autonomous-driving-levels.html.

[5] R. Behringer and N. Muller, „Autonomous road vehicle guidance from autobahnen to narrow curves,“ *IEEE Transactions on Robotics and Automation*, vol. 14, no. 5, pp. 810–815, 1998, DOI: 10.1109/70.720356.

[6] Sebastian Gnatzig, „Trajectory based teleoperation of Trajectory based teleoperation of Road vehicles based on a Shared Control approach,“ PhD thesis, Chair of Automotive Technology, Technical University of Munich, Munich, 2015.

[7] C. Urmson and W. Whittaker, „Self-Driving Cars and the Urban Challenge,“ *IEEE Intelligent Systems*, vol. 23, no. 2, pp. 66–68, 2008, DOI: 10.1109/MIS.2008.34.

[8] Neurohive, *Sensor scheme of an autonomous vehicle, showing the functionality of each.*, 2019. Available: https://neurohive.io/en/state-of-the-art/self-driving-cars/.

[9] Udacity. „*Architecture of Self-Driving Cars*,“ 2019. Available: https://atul.fyi/post/2017/01/03/how-self-driving-cars-work/.

[10] C. Gold, „Modeling of Take-Over Performance in Highly Automated Vehicle Guidance,“ PhD thesis.

[11] R. Attia, R. Orjuela and M. Basset, „Combined longitudinal and lateral control for automated vehicle guidance,“ *Vehicle System Dynamics*, vol. 52, no. 2, pp. 261–279, 2014, DOI: 10.1080/00423114.2013.874563.

[12] D. E. Rivera, M. Morari and S. Skogestad, „Internal model control: PID controller design,“ *Industrial & Engineering Chemistry Process Design and Development*, vol. 25, no. 1, pp. 252–265, 1986, DOI: 10.1021/i200032a041.

[13] B. Mashadi, M. Mahmoudi-Kaleybar, P. Ahmadizadeh and A. Oveisi, „A path-following driver/vehicle model with optimized lateral dynamic controller,“ *Latin American Journal of Solids and Structures*, vol. 11, no. 4, pp. 613–630, 2014, DOI: 10.1590/S1679-78252014000400004.

[14] Nicis Digital. „*Shema of the Proportional-Integral-Derivative (PID) Controller*,“ 2019. Available: https://nicisdigital.wordpress.com/2011/06/27/proportional-integral-derivative-pid-controller/.

[15] R. Marino, S. Scalzi, G. Orlando and M. Netto, „A nested PID steering control for lane keeping in vision based autonomous vehicles,“ in *Proceedings of the 2009 American Control Conference*, 2009, pp. 2885–2890, ISBN: 978-1-4244-4523-3. DOI: 10.1109/ACC.2009.5160343.

[16] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian and P. Mahoney, „Stanley: The robot that won the DARPA Grand Challenge,“ *Journal of Field Robotics*, vol. 23, no. 9, pp. 661–692, 2006, DOI: 10.1002/rob.20147.

[17] J. Snider, „Automatic Steering Methods for Autonomous Automobile Path Tracking,“ 2011.

[18] L. B. Cremean, T. B. Foote, J. H. Gillula, G. H. Hines, D. Kogan, K. L. Kriechbaum, J. C. Lamb, J. Leibs, L. Lindzey, C. E. Rasmussen, A. D. Stewart, J. W. Burdick and R. M. Murray, „Alice: An information-rich autonomous vehicle for high-speed desert navigation,“ *Journal of Field Robotics*, vol. 23, no. 9, pp. 777–810, 2006, DOI: 10.1002/rob.20135.

[19] Y. Chen, Y. Shan, L. Chen, K. Huang and D. Cao, „Optimization of Pure Pursuit Controller based on PID Controller and Low-pass Filter,“ in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, 11/4/2018 - 11/7/2018, pp. 3294–3299, ISBN: 978-1-7281-0321-1. DOI: 10.1109/ITSC.2018.8569416.

[20] R. Wallace, A. Stentz, C. Thorpe, H. Maravec, W. Whittaker and T. Kanade, *First Results in Robot Road-Following*, 1985.

[21] B. Paden, M. Cap, S. Z. Yong, D. Yershov and E. Frazzoli, „A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles,“ *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 33–55, 2016, DOI: 10.1109/TIV.2016.2578706.

[22] R. Rajamani, C. Zhu and L. Alexander, „Lateral control of a backward driven front-steering vehicle,“ *Control Engineering Practice*, vol. 11, no. 5, pp. 531–540, 2003, DOI: 10.1016/S0967-0661(02)00143-0.

[23] R. A. Cordeiro, J. R. Azinheira, E. C. de Paiva and S. S. Bueno, „Dynamic modeling and bio-inspired LQR approach for off-road robotic vehicle path tracking,“ in *2013 16th International Conference on Advanced Robotics (ICAR)*, 2013, pp. 1–6, ISBN: 978-1-4799-2722-7. DOI: 10.1109/ICAR.2013.6766549.

[24] X. Yang, G. Liu, A. Li and van Dai, „A Predictive Power Control Strategy for DFIGs Based on a Wind Energy Converter System,“ *Energies*, vol. 10, no. 8, p. 1098, 2017, DOI: 10.3390/en10081098.

[25] Baidu. „*Apollo: an open source autonomous driving platform*,“ 2019. Available: https://github.com/ApolloAuto/apollo.

[26] A. Jezierski, J. Mozaryn and D. Suski, „A Comparison of LQR and MPC Control Algorithms of an Inverted Pendulum,“ in *Trends in advanced intelligent control, optimization and automation* (Advances in Intelligent Systems and Computing). vol. 577, W. Mitkowski, J. Kacprzyk, K. Oprzędkiewicz and P. Skruch, ed. New York NY: Springer Berlin Heidelberg, 2017, pp. 65–76, ISBN: 978-3-319-60698-9. DOI: 10.1007/978-3-319-60699-6_8.

[27] Z. Li, J. Deng, R. Lu, Y. Xu, J. Bai and C.-Y. Su, „Trajectory-Tracking Control of Mobile Robot Systems Incorporating Neural-Dynamic Optimized Model Predictive Approach,“ *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 46, no. 6, pp. 740–749, 2016, DOI: 10.1109/TSMC.2015.2465352.

[28] M. S. Netto, S. Chaib and S. Mammar, „Lateral adaptive control for vehicle lane keeping,“ in *2004 American control conference*, 2004, 2693–2698 vol.3, ISBN: 0-7803-8335-4. DOI: 10.23919/ACC.2004.1383872.

[29] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, D. Johnston, S. Klumpp, D. Langer, A. Levandowski, J. Levinson, J. Marcil, D. Orenstein, J. Paefgen, I. Penny, A. Petrovskaya, M. Pflueger, G. Stanek, D. Stavens, A. Vogt and S. Thrun, „Junior: The Stanford entry in the Urban Challenge,“ *Journal of Field Robotics*, vol. 25, no. 9, pp. 569–597, 2008, DOI: 10.1002/rob.20258.

[30] L. Tan, S. Yu, Y. Guo and H. Chen, „Sliding-mode control of four wheel steering systems,“ in *2017 IEEE International Conference on Mechatronics and Automation*, 2017, pp. 1250–1255, ISBN: 978-1-5090-6758-9. DOI: 10.1109/ICMA.2017.8015996.

[31] M. B., L. Z. and N. E., „A Fuzzy Logic Controller for Autonomous Wheeled Vehicles,“ in *Supervisory Control for Turnover Prevention of a Teleoperated Mobile Agent with a Terrain-Prediction Sensor Module*, J. Buchli, ed. INTECH Open Access Publisher, 2006, ISBN: 3-86611-284-X. DOI: 10.5772/4721.

[32] A. de Luca, G. Oriolo and C. Samson, „Feedback control of a nonholonomic car-like robot,“ in *Robot Motion Planning and Control* (Lecture Notes in Control and Information Sciences). vol. 229, M. Thoma and J.-P. Laumond, ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 171–253, ISBN: 978-3-540-76219-5. DOI: 10.1007/bfb0036073.

[33] R. M. Murray and S. S. Sastry, „Nonholonomic motion planning: steering using sinusoids,“ *IEEE Transactions on Automatic Control*, vol. 38, no. 5, pp. 700–716, 1993, DOI: 10.1109/9.277235.

[34] Y. Kuwata, S. Karaman, J. Teo, E. Frazzoli, J. P. How and G. Fiore, „Real-Time Motion Planning With Applications to Autonomous Urban Driving,“ *IEEE Transactions on Control Systems Technology*, vol. 17, no. 5, pp. 1105–1118, 2009, DOI: 10.1109/TCST.2008.2012116.

[35] N. Peric, I. Petrović and Z. Butkovic, *ISIE 2005: Proceedings of the IEEE international symposium on industrial electronics 2005 / Editors: Nedjeljko Peric, Ivan Petrovic and Zeljko Butkovic*, 2005, ISBN: 0780387384. Available: http://ieeexplore.ieee.org/Xplore/home.jsp%20BLDSS.

[36] A. Rucco, G. Notarstefano and J. Hauser, „Computing minimum lap-time trajectories for a single-track car with load transfer,“ in *2012 IEEE 51st Annual Conference on Decision and Control*, 2013, pp. 6321–6326, ISBN: 978-1-4673-2066-5. DOI: 10.1109/CDC.2012.6426265.

[37] J. h. Jeon, R. V. Cowlagi, S. C. Peters, S. Karaman, E. Frazzoli, P. Tsiotras and K. Iagnemma, „Optimal motion planning with the half-car dynamical model for autonomous high-speed driving,“ in *2013 American Control Conference (ACC*, 2013, pp. 188–193, ISBN: 978-1-4799-0177-7. DOI: 10.1109/ACC.2013.6579835.

[38] S. C. Peters, E. Frazzoli and K. Iagnemma, „Differential flatness of a front-steered vehicle with tire force control," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 298–304, ISBN: 978-1-61284-456-5. DOI: 10.1109/IROS.2011.6094800.

[39] yahya Tawill. „*An Introduction to Robot Operating System*," 2019. Available: https://www.allaboutcircuits.com/technical-articles/an-introduction-to-robot-operating-system-ros/.

[40] Clearpath Robotics. „*ROS 101: Intro to the Robot Operating System*," 2019. Available: https://robohub.org/ros-101-intro-to-the-robot-operating-system/.

[41] N. Wang, J. Wang, Z. Li, X. Tang and D. Hou, „Fractional-Order PID Control Strategy on Hydraulic-Loading System of Typical Electromechanical Platform," *Sensors (Basel, Switzerland)*, vol. 18, no. 9, 2018, DOI: 10.3390/s18093024.

[42] M. Jun and M. G. Safonov, „Automatic PID tuning: an application of unfalsified control," in *Proceedings of the 1999 IEEE international symposium on computer aided control system design*, 1999, pp. 328–333, ISBN: 0-7803-5500-8. DOI: 10.1109/CACSD.1999.808669.

[43] P. Falcone, F. Borrelli, J. Asgari, H. E. Tseng and D. Hrovat, „Predictive Active Steering Control for Autonomous Vehicle Systems," *IEEE Transactions on Control Systems Technology*, vol. 15, no. 3, pp. 566–580, 2007, DOI: 10.1109/TCST.2007.894653.

[44] G. K. Anusha Nagabandi. „*Overview of our model-based reinforcement learning algorithm.*" 2019. Available: https://bair.berkeley.edu/blog/2017/11/30/model-based-rl/.

# Appendix

**A Chapter Anhang** ........ **xiii**
- **A.1 Algorithm** ........ **xiii**
  - A.1.1 Curvature and Curvature Radius ........ xiii
  - A.1.2 Iterative Curved Surface Fitting ........ xiv
- **A.2 Controller Performance** ........ **xv**

# A Chapter Anhang

## A.1 Algorithm

In this section, some algorithmic methods used in this thesis is introduced.

### A.1.1 Curvature and Curvature Radius

The curvature and curvature radius is given as

$$\kappa = \frac{\mathrm{d}\phi}{\mathrm{d}s} = \frac{\frac{\mathrm{d}\phi}{\mathrm{d}x}}{\frac{\mathrm{d}s}{\mathrm{d}x}} = \frac{\frac{\mathrm{d}^2 x}{\mathrm{d}x^2}}{(1+(\frac{\mathrm{d}y}{\mathrm{d}x})^2)^{\frac{3}{2}}}, \tag{A.1}$$

$$\kappa_r = \frac{1}{\kappa}. \tag{A.2}$$

```python
def calc_curvature_theta(state, cx, cy):

    polynom, diff = poly_Abschnitt(state, cx, cy)

    def f(x):
        return polynom[0] * (x ** 3) + polynom[1] * (x ** 2) + polynom[2] * x + polynom[3]

    d1y_dx1 = derivative(f, diff, dx=1e-6, n=1)
    d2y_dx2 = derivative(f, diff, dx=1e-6, n=2)

    curvature = abs(d2y_dx2)/((1+d1y_dx1**2)**(3/2))

    theta = math.atan2(d1y_dx1, 1.0)

    if d2y_dx2 !=0:
        curvature_radius = abs(((1 + d1y_dx1 ** 2) ** 1.5) / d2y_dx2)
    else:
        curvature_radius = 0


    return curvature, theta, curvature_radius
```

### A.1.2 Iterative Curved Surface Fitting

```
x1 = array of lookahead distance
x2 = array of curvature radius
x3 = array of lateral error

point = []

for i in range(0,len(x1)):
    point.append([x1[i],x2[i],x3[i]])

ISum = 0.0
X1Sum = 0.0
X2Sum = 0.0
X1_2Sum = 0.0
X1X2Sum = 0.0
X2_2Sum = 0.0
YSum = 0.0
X1YSum = 0.0
X2YSum = 0.0

for i in range(0,len(point)):

 x1i=point[i][0]
 x2i=point[i][1]
 yi=point[i][2]

 ISum = ISum+1
 X1Sum = X1Sum+x1i
 X2Sum = X2Sum+x2i
 X1_2Sum = X1_2Sum+x1i**2
 X1X2Sum = X1X2Sum+x1i*x2i
 X2_2Sum = X2_2Sum+x2i**2
 YSum = YSum+yi
 X1YSum = X1YSum+x1i*yi
 X2YSum = X2YSum+x2i*yi

m1=[[ISum,X1Sum,X2Sum],[X1Sum,X1_2Sum,X1X2Sum],[X2Sum,X1X2Sum,X2_2Sum]]
mat1 = np.matrix(m1)
m2=[[YSum],[X1YSum],[X2YSum]]
mat2 = np.matrix(m2)
_mat1 =mat1.getI()
mat3 = _mat1*mat2

m3=mat3.tolist()
a0 = m3[0][0]
a1 = m3[1][0]
a2 = m3[2][0]

y = a0+a1*x1+a2*x2
```

## A.2 Controller Performance

This section presents the performance of different controllers in path 2, namely a path with lane changing course.

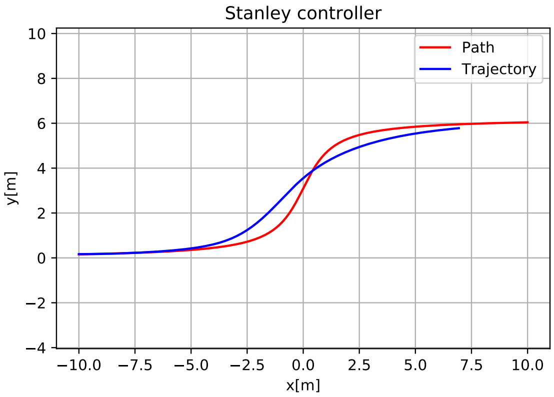


Figure A.1: Stanley controller performance

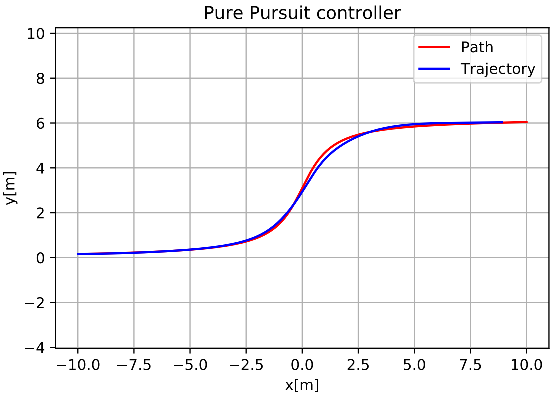


Figure A.2: Pure Pursuit controller performance

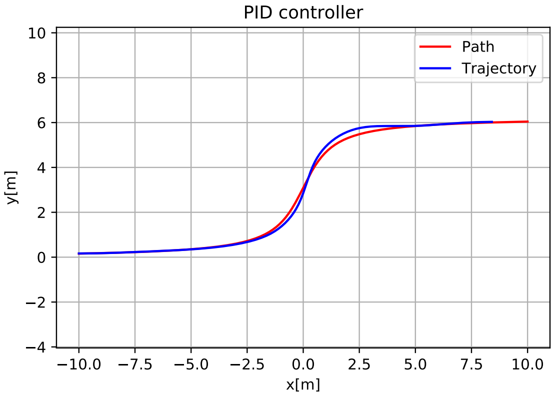


Figure A.3: PID controller performance

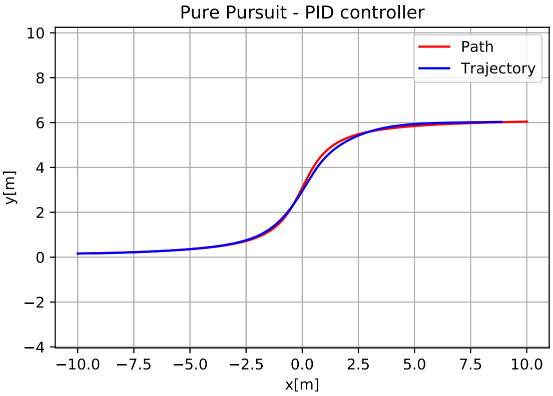


Figure A.4: PP-PID controller performance

Table A.1: Comparison of the performance of different controllers in path 2

Overview of the lateral errors of different controllers

| Controller | Average lateral error | Maximal lateral error | Variance |
|---|---|---|---|
| Stanley | 0.585$m$ | 1.492$m$ | 0.1829$m^2$ |
| PID | 0.071$m$ | 0.271$m$ | 0.0112$m^2$ |
| Pure Pursuit | 0.056$m$ | 0.205$m$ | 0.0063$m^2$ |
| Pure Pursuit - PID | 0.049$m$ | 0.179$m$ | 0.0049$m^2$ |

As shown in figures above, the Stanley controller has the worst performance. The Pure Pursuit controller and PP-PID controller has both better performance than the PID controller. The table gives the statistic of errors, which is more accurate. It shows that the PP-PID controller has more accuracy than the Pure Pursuit controller in the path-following maneuver in a lane changing course.