Technische Universität Ilmenau
Fakultät für Informatik und Automatisierung
Fachgebiet Prozessoptimierung

Pontificia Universidad Católica del Perú
Escuela de Posgrado

Control of Autonomous Multibody Vehicles using Artificial Intelligence

Masterthesis in fulfilment of the requirements for the degree of

Master of Science (M.Sc.)

in Technische Kybernetik und Systemtheorie

and

Magíster

en Ingeniería de Control y Automatizacíon

Benedikt Roder

Study Program: International Double Degree

Supervisor (PUCP Lima): Dr. Antonio Manuel Morán Cárdenas

Supervisor (Co-asesor TU Ilmenau): Prof. Dr.-Ing. habil. Pu Li

This thesis was submitted on January 9th, 2020.

ET LUX IN TENEBRIS LUCET
MCMXVII

Abstract

The field of autonomous driving has been evolving rapidly within the last few years and a lot of research has been dedicated towards the control of autonomous vehicles, especially car-like ones. Due to the recent successes of artificial intelligence techniques, even more complex problems can be solved, such as the control of autonomous multibody vehicles. Multibody vehicles can accomplish transportation tasks in a faster and cheaper way compared to multiple individual mobile vehicles or robots.

But even for a human, driving a truck-trailer is a challenging task. This is because of the complex structure of the vehicle and the maneuvers that it has to perform, such as reverse parking to a loading dock. In addition, the detailed technical solution for an autonomous truck is challenging and even though many single-domain solutions are available, e.g. for pathplanning, no holistic framework exists. Also, from the control point of view, designing such a controller is a high complexity problem, which makes it a widely used benchmark.

In this thesis, a concept for a plurality of tasks is presented. In contrast to most of the existing literature, a holistic approach is developed which combines many stand-alone systems to one entire framework. The framework consists of a plurality of modules, such as modeling, pathplanning, training for neural networks, controlling, jack-knife avoidance, direction switching, simulation, visualization and testing. There are model-based and model-free control approaches and the system comprises various pathplanning methods and target types. It also accounts for noisy sensors and the simulation of whole environments.

To achieve superior performance, several modules had to be developed, redesigned and interlinked with each other. A pathplanning module with multiple available methods optimizes the desired position by also providing an efficient implementation for trajectory following. Classical approaches, such as optimal control (LQR) and model predictive control (MPC) can safely control a truck with a given model. Machine learning based approaches, such as deep reinforcement learning, are designed, implemented, trained and tested successfully. Furthermore, the switching of the driving direction is enabled by continuous analysis of a cost function to avoid collisions and improve driving behavior.

This thesis introduces a working system of all integrated modules. The system proposed can complete complex scenarios, including situations with buildings and partial trajectories. In thousands of simulations, the system using the LQR controller or the reinforcement learning agent had a success rate of $>95\,\%$ in steering a truck with one trailer, even with added noise. For the development of autonomous vehicles, the implementation of AI at scale is important. This is why a digital twin of the truck-trailer is used to simulate the full system at a much higher speed than one can collect data in real life.

ET LUX IN TENEBRIS LUCET
MCMXVII

ET LUX IN TENEBRIS LUCET
MCMXVII

Contents

List of Figures

List of Abbreviations and Symbols

Abbreviations

AI	Artificial Intelligence
DAS	Driver Assistance Systems
MATLAB	Programming Language
LQR	Linear Quadratic Regulator
MPC	Model Predictive Control
ARE	Algebraic Riccati Equation
EKF	Extended Kalman Filter
MIMO	Multiple Input Multipe Output
IDS	Instantaneous Desired State
EPDP	Extended Perpendicular Desired Position Method
LOS	Line Of Sight Method
SAT	Separating Axis Theorem
ANN	Artifical Neural Network
MLP	Multi Layer Perceptron
DBP	Dynamic Back Propagation
MSE	Mean Squared Error
SGD	Stochastic Gradient Descent
RL	Reinforcement Learning
DDPG	Deep Deterministic Policy Gradient

Symbols

$\mathbf{N}, \mathbf{R}, \mathbf{C}$	Natural, real and complex numbers
i, j, k	Counter
$f(\cdot), g(\cdot)$	Functions
z, y, a, b	Variables
$\dot{z}, z'$	Time and location derivative of x
mt	Timestep [s]
$U(a, b)$	Uniform distribution with values between a and b
$N(a, b^2)$	Normal distribution with mean a and variance b
$>$	State
$>^*$	Desired state
r	Reference, e.g. target state
u	Input, e.g. steering angle
δ	Steering angle
ϑ	Angle
v	Velocity
s	Cost value to reach target
M	Model
O_i, B	Objects, such as truck or buildings
P	Parameters
E_t	Environment at time t
Ω	Neural Network
η	learnrate
ξ	Parameter of neural network, e.g. weights
μ	Policy
$\beta, \kappa, \varepsilon$	Design parameter for the reward of the RL agent
γ, τ	Parameter for the training of the RL agent

ET LUX IN TENEBRIS LUCET
MCMXVII

1 Introduction

In the past few years, artificial intelligence technology has evolved in an exponential manner, resulting in lots of new developments whereby more complex problems that are being tackled today. The level of human involvement in various processes is declining and is being replaced or assisted by technology. This is either to reduce the execution time of a task or to handle the difficulty of controlling a system, that would enable an otherwise infeasible task to be performed. Because of this phenomenon, newly developed technologies have been introduced to many industries and to the daily life of many people.

In the field of engineering, these technological changes can be seen in almost every discipline. Since the beginning of the last century, systems have been implemented which exclusively relied on the management of human beings. Over time, these systems became more intelligent, i.e., they can function on their own without or very little human intervention. Finally, the concept of artificial intelligence (AI) was introduced, allowing systems to learn on their own, resulting in behavior not explicitly determined by the developer.

Systems using AI are able to make decisions to reach some goal. They can also take into account the environment and additional constraints. With such capabilities, autonomous robots can be created. Autonomous vehicles have been used in various fields. In unknown areas, the control system must be capable of the sensing, processing and decision-making in real-time, considering the characteristics of the system itself, the existing constraints and physical resources.

1.1 Overview

Vehicles can be classified based on their level of automation. Therefore, a set of automation levels have been defined [1]:

- **Level 0** describes a vehicle without automation. All tasks are executed by the driver.
- **Level 1** vehicles use driver assistance systems (DAS), monitored by the driver.
- **Level 2** describes partial automation. The system can execute steering and accelerating, but the human is still responsible and has to monitor the system at all times.
- **Level 3** allows the human to stop monitoring the environment. With conditional automation, the driver is only the fallback level in emergency cases.
- **Level 4** describes a vehicle that can drive on its own under certain conditions.
- **Level 5** is full automation. The system can handle all driving modes in all environments and situations. These vehicles are also called *autonomous* vehicles.

An ambitious goal is being pursued in the development of automated driving functions. As shown in Figure 1.1, the development can be divided into several tasks. The tasks shown in orange will be addressed in this thesis. Those tasks are not only for car-like vehicles, but also for autonomous trucks. Trucks are bigger, consist of more than one moving body and have to execute different activities, such as backing up to a loading dock. This is why there are even more development steps to consider.

Figure 1.1: Development of autonomous vehicles

It all begins with the sensors, that map the environment by measuring quantities, like distance to objects, yaw-rates or velocities. Taken all together, those measurements make up the perception of the environment. Next step is to understand the current situation. This includes distinguishing moving objects from buildings and the detection of false measurements. After that, a high level or maneuver planner determines which action to take next. This could be a lane change, a parking maneuver or an emergency brake. At this point, a decision is made about how the vehicle should move. After the top-level decision, the exact options, points and angles are computed by the low level or trajectory planner. Finally, controllers compute the desired steering and acceleration and execute the actions using the actuators of the vehicle.

This work deals with the low-level planning and control of autonomous multibody vehicles. Sometime high-level and low-level planning are interlinked and cannot be separated perfectly. However, it is assumed that a target position or a target trajectory is already available, and a reliable perception of the environment exists. It will be focused on the planning of desired states $>^*$ and the determination of associated control signals u^*.

1.2 Motivation and Objective

The field of autonomous driving provides many advantages in cost, efficiency and safety. A lot of research has been dedicated to the control of autonomous vehicles, especially car-like ones. Due to recent successes of artificial intelligence techniques, even more complex problems can be solved, such as the control of autonomous multibody vehicles. To enable safe and effective movement in critical situations, such as backward driving without human driver, those systems have to be controlled autonomously.

Figure 1.2: Applications of multibody vehicles [2–4]

Multibody vehicles, such as trucks with trailers or mobile robots, are used in a wide range of industries, such as warehouses, mining companies and logistics for delivery and transportation of goods, see Figure 1.2. They can accomplish transportation tasks faster and cheaper compared to multiple individual robots. The transportation capacity increases when one or more trailers are used.

Even for a human, driving a truck-trailer is challenging because of the complex structure of the vehicle and the maneuvers that it has to perform, like reverse parking to a loading dock. From the control point of view, designing a controller for truck-trailers is a problem of high complexity, which makes it a widely used benchmark.

On the one hand, there is a need for autonomous vehicles to realize the possible benefits. On the other hand, there are still many challenges to overcome until fully autonomous trucks and robots can be realized. The plurality of those challenges and approaches, such as

- Driving forward and backward and changing the driving direction automatically,
- Avoiding static and dynamic obstacles safely,
- Planning and following a path,
- Being robust against sensor failure and external influences

and the combination of those, all while avoiding the jack-knife state, sparks the need for a unified system with a more generic way of control.

In real life, in contrast to many path planning approaches, it can be useful to drive really close to objects and change the driving direction just before reaching it. This can be seen in many situations, when human drivers want to navigate or park in narrow spaces. Therefore, a way to move close to objects, but make sure there are no collisions is needed.

The overall objective of this work is to improve the capabilities of multibody vehicles for the application of autonomous driving. The goal is to develop a framework for the intelligent control of multibody vehicles, especially truck-trailer systems, using different approaches such as machine learning and neuro-control, enabling a generic approach to a plurality of scenarios. Finally, the optimization of the control behavior and design of cost functions is aspired.

With simulations in MATLAB, intelligent control strategies can be developed. Different scenarios were investigated and tested with simulations. To improve the capabilities of multibody vehicles, the following topics will be deeply examined in this work:

- Obtain an exact, but simple mathematical model for truck with one or two trailers,
- Design a software that can simulate and execute environment, controller and model,
- Propose a framework for trajectory planning and controlling with strategies for different driving modes, such as
  - Evaluate positions,
  - Switch directions and
  - Avoid obstacles,
- Design, train and improve neural networks capable of controlling the vehicle,
- Acquire, analyze and interpret data obtained from simulations and
- Analyze the impact and improvement of the designed framework as well as its performance in designed scenarios and draw conclusions.

Several systems have been proposed to solve specific problems, such as reverse parking truck-trailer vehicles or object avoidance of car-like vehicles. But in general, there are many factors to consider when building a framework for the control of autonomous multibody vehicles. Such a framework, consisting of several controllers, has to be tested in a plurality of scenarios with different trajectories that can evaluate the concept. Possible scenarios are path following, e.g. for lane change on highway or parking in narrow spaces. Methods can be compared with performance measures, such as success rate, needed time to complete a task and collision free movement.

Objective

The objective of this work is the development and evaluation of a framework for the control autonomous multibody vehicles using artificial intelligence. The framework is required to simulate an environment, implement several control approaches and provide a plurality of scenarios for testing.

1.3 Content and Outline

The content of this thesis will cover the development of a framework. It will range from literature review to modeling of the system and from controller design to testing.

First, established methods for the control of truck-trailer vehicles are presented and analyzed. This includes, but is not limited to, path planning and following and obstacle avoidance. Different approaches for the modeling of multibody vehicles will be derived and investigated.

Second, a concept for the standardization of the presented control methods as well as for the integration into the framework is designed. Different situations will be evaluated to determine optimal points to change the driving direction and to avoid collisions. Therefore, a cost function for the current state, as well as high level decision processes have to be designed. After all, the best path and control strategy can be selected for a given environment. Furthermore, new control methods, such as controller based on artificial intelligence are analyzed.

Finally, a software prototype will be used to demonstrate the concept with a control algorithm in several environments. It should be noted that the concept is not limited to a certain control algorithm, but can make use of any process to determine optimal inputs.

This work is organized as follows. Chapter 2 presents the state of the art in terms of the development of autonomous vehicles, especially models and control strategies for multibody vehicles. In Chapter 3, the behavior of multibody vehicles is investigated to obtain a mathematical model. This model is then used to simulate the dynamical system. In chapter 4, components for control, path planning and the switching of the driving direction are developed. Chapter 5 contains the training and evaluation of neural networks used to control the system. Chapter 6 presents the software that implements the concept and evaluates the resulting framework using test scenarios. Chapter 7 gives a summary and an outlook for possible improvements in future research.

How to read this document

Grey boxes include additional knowledge to understand the given information. In general, many implementation details and remarks are given throughout the document, such that the reader could actually rebuild the system. In some cases, readability and understandability is chosen over mathematical rigorousness. Notes and warnings are placed inside boxes as follows.

The methods used are mathematical modeling and analysis of the system, controller design and training of neural networks. In addition, statistical evaluation will be done by running simulations.

ET LUX IN TENEBRIS LUCET
MCMXVII

2 State of the Art

To get an overview of the latest techniques, this chapter introduces the state of the art of the modeling of multibody vehicles, pathplanning and the control of autonomous vehicles. This will support the focus of the further chapters, especially truck-trailer models (Chapter 3) and control strategies (Chapter 4), which are based on the presented models.

2.1 Modeling of Multibody Vehicles

To appropriately deal with a truck-trailer system, a model, in form of mathematical equations, is required. In reality, a plurality of different truck-trailer combinations are present, see Figure 2.1. As the model should not only cover the trailer configuration, but also the actuation, there are even more possible scenarios, as e.g. [5] proposed a driver assistance system with active trailer steering.

Figure 2.1: Visualization of different trailer configurations [6]

A truck with one trailer will be the focus in the following chapters, even though the principles can be applied to more than one trailer. The modeling is presented for an arbitrary number of trailers.

Mathematical Models

A model for a standard N-trailer vehicle is presented in [7]. This standard model consists of passive trailers (no active steering and no traction) and a truck with front axle steering and rear axle traction for forward and backward movements. In addition, it has all hitching points located on the preceding wheel axle, which simplifies the model. A model for arbitrary hitching can be found in [8]. [9] presents a generalized model with hitching as well as front and rear wheel drive.

Single-track models are the most popular ones, as they lead to a simple, but precise model at low speeds [10]. [11] models the backward movement of a truck as inverted horizontal pendulum, which already has a wide range of known control approaches. Those models can be used to design controllers for the steering angle and the velocity. A proof of controllability of multibody robots is given in [12]. Geometrical relationships of the rectangles and other shapes, used to represent the truck, the trailers and the environment, can be used to detect collisions, see [13].

As most of the simulations do not deal with possible noise in measurements or control signals, real implementations can test approaches in the real word. In many cases, a miniaturized version, e.g. 1:14, is equipped with a microcontroller, sensors and actuators. For the perception of the environment, sensors such as laser scanners [14], ultrasonic sensors and RF receivers [15] are used. For the control, electrical power steering, an electric brake system and electric engines are required.

Pathplanning

The control of mobile robots can be classified into three objectives, which are stabilization, setpoint tracking and path following [16]. So paths have to be planned for controllers to follow them [17]. A pathplanning strategy for a truck with trailers is presented in [18]. To plan the next steps in a closed-loop form, instead of the open loop approaches, such as A* or the ant colony algorithm [19], special coordinates and a controller can be used.

In [20], a trajectory is generated by segmentation into smaller elements, such as lines and curves. [21] proposes an approach with several segments for reverse parking of car-like vehicles. Minimum length pathplanning is proposed in [22], which is also used in [23] for trajectory planning and target tracking with two circle segments and a straight line.

Two methods using perpendicularity and the line of sight are described in [24], which are used to compute desired states to follow a given trajectory. In [25], a nonholonomic model is used to find kinematically feasible paths for trucks with several trailers, even in the presence of obstacles or other constraints. In [26], a controller is used for pathplanning for collision free movement of a truck with two trailers.

The presence of obstacles requires the planned paths to avoid those [27]. Force fields for collision avoidance in combination with particle swarm optimization and variable velocity are used in [28]. [29] proposes a path planner with direction changes and obstacle avoidance using a repelling spring. Pathplanning for obstacle avoidance is done in [30] with multiple stages, such as initial pathplanning, obstacle avoidance, smoothing and tracking.

2.2 Control of Autonomous Vehicles

In the following, some model-based and model-free controllers will be presented. Aside of the approach itself, it is important to ensure the stability of the controller in the closed loop. [31] proves stability for one type of controller, based on a Lyapunov approach. In [32], the stability region for certain initial conditions in the state-space is determined. In [33], the robot is controlled from last trailers perspective, to ensure its desired behavior.

Model-based Controllers

Classical approaches can be used to control the steering angle and the velocity of multi-body robots. The design is based on the mathematical model, which can be analyzed to find suitable parameters as well. [34] uses pole placement for two different controllers, steering a truck-trailer backwards. Other classical methods, as presented in [35], can be used to compute control parameters by optimization. The linear-quadratic regulator (LQR) approach computes optimal parameters for a linear system with a quadratic cost function. This approach is used e.g. in [24] to determine gains for the steering angle control. A controller cascade is developed in [36] for backing up with two trailers. A sliding mode control for automatic steering of a car-like vehicle is proposed in [37].

An extended input-output linearization is applied in [10] to control the angle between truck and trailer and trailer and trailer, respectively. [8] proposes to use an input-state linearization in case the input-output linearization does not work, because of a more complicated model. A switching controller, based on the sign of the trucks velocity, for forward and backward motion is presented in [38], enabling the system to change the controller, based on the current state. [39] designes a nonlinear H_∞-control scheme with the longitudinal velocity and the angular velocity as control input.

For similar problems, Model Predictive Control (MPC) is a robust and reliable advanced control strategy that can handle dynamic Multiple Inputs Multiple Outputs (MIMO) systems, that has been applied successfully [40]. It is widely used, as more effective than classical control methods, for the cost of higher computational effort. One of the main features of MPC is that the desired behavior as well as constraints on the system can be specified in the problem formulation [41]. A MPC implementation for the control of a mobile robot is presented in [42]. Obstacle avoidance is introduced as nonlinear constraint as part of the optimization problem.

Preview control, as introduced in [43], can be used to improve working controllers with information about future reference inputs. [44] and [45] implement preview control to extend the control for a car-like vehicle and truck-trailer, respectively.

Models are not perfect, this is why extensive testing with noisy or real world data is required to test the performance of model-based controllers.

Model-free Controllers

Another way to design a controller is to gather knowledge from other sources than a mathematical model. Fuzzy controllers are based on rules, e.g. obtained by experience of a human operator. In [46] a fuzzy controller for a truck with two trailers is shown. Based on the set of rules, it can also avoid obstacles while driving forward. A Variable Universe Based Fuzzy Controller (VUBFC), that has a variable membership function is presented in [47]. It has a better performance than common fuzzy controllers.

Fuzzy neural networks are introduced in [48] to steer a car-like vehicle based on a training process for the parameters. A discrete model with fuzzy control in proposed in [49]. Several variables are controlled in [50] with an internal virtual controller. A proof of stability is given as well. [51] describes a fuzzy implementation of a truck with one trailer and robust control for stabilization tasks.

Based on expert model knowledge, [52] develops a controller with a "safety margin" to execute an emergency halt. An expert based fuzzy logic controller is proposed in [53]. It has only a short learning phase, but it requires external experience. An expert system is another possible approach to solve forward and backward motion, even in constrained spaces [54].

Artificial intelligence (AI) can be used to control multibody vehicles. A general overview of AI can be found in [55]. The problem of a truck with trailer, backing up to a loading dock was famously used in [56] to show the effectiveness of a neural network based controller. In addition, another neural network was used for an emulator for the nonlinear dynamics to improve the systems model. A comparison between fuzzy and neuro-controller for trucks without trailer is given in [57]. A controller neural network, trained with data attained from a pathplanning program, is described in [15].

With the help of genetic algorithms, a controller for a truck with five trailers moving forward or backward while avoiding obstacles was developed in [58]. Genetic algorithms for the observation of nonlinear systems were introduced in [59]. The parameters in networks that make up a neuro-controller can also be updated by genetic algorithms [60].

Neural networks can learn from feedback by the environment. This is called reinforcement learning. With deep reinforcement learning and the Deep Deterministic Policy Gradient (DDPG) algorithm [61], an agent can learn continuous actions with continuous states. This is made possible by two neural networks, the so called *actor* and *critic* networks. In [62], deep reinforcement learning was applied to the truck-trailer problem. With this approach, many initial positions, even starting in the jack-knife position, were solved successfully.

There is a need for real life data for the training of neural networks. This data can be obtained from pathplanning algorithms from real-world implementations as done in [15] or by street observations and postprocessing, as proposed in [63].

Overview

To give an overview over the literature in terms of the control of multibody robots, Table 2.1 shows publications based on the domain and the used plant. The domain describes the approach to the problem, such as pathplanning or obstacle avoidance. The plant is the model or real implementation of trucks with a certain number of trailers.

	Truck	Truck-trailer	Truck with more trailers
Forward driving	[37]	[54]	[46], [58]
Backward driving	[44], [48]	[10], [15]*, [54]	[7], [10]
Switching directions	[22], [29]	[14]*, [38]*	
Obstacle avoidance	[29]	[30]	[26] , [46], [58]
Pathplanning	[21], [22]	[14]*, [20]*	[25]
Pathfollowing			[7], [24]**

Table 2.1: *real implementation ** with simplifications (i.e. only linefollowing)

One can see that in most of the cells, there is at least one contribution, but there is no holistic system solving the majority of problems.

ET LUX IN TENEBRIS LUCET
MCMXVII

3 Modeling and Computation

*For the control of autonomous **multibody vehicles** using artificial intelligence, a mathematical model is required, e.g. for training, simulation and visualization. For computational efficiency, this model needs to be easy to compute.*

A model is an approximation of the reality that is reduced to the properties that are relevant for the application at hand. Since in the following, the focus is on the control of position and angles, the models are adjusted accordingly. For example, the exact shape of the vehicles is neglected. In this chapter, different models for multibody vehicles are derived. There are several assumptions to make, as the model should suffice in the accuracy of describing the system, while at the same time be as simple as possible to reduce effort in computation and modeling.

However, truck-trailer mobile robots are a complex, nonlinear, unstable, underactuated and nonholonomic system difficult to control, especially when moving backwards, which have led to an intensive research work for analyzing their motion characteristics.

3.1 Assumptions

Most multibody vehicles consist of a primary body, usually called the tractor or the *truck*, and following bodies, usually called the trailers[1]. All bodies are assumed to have a rectangular shape with the same width.

Most of the models are based on robot kinematics, valid when the robot moves at low speeds without wheels-side-slipping. In this condition, the robot motion is determined only by geometrical considerations independent of masses, inertias and frictional forces. Common trucks are driven by acceleration and steering. For a low-velocity model, many effects, such as slipping of the wheels and three-dimensional behavior of the truck, can be neglected. Therefore, the acceleration will be regarded as 0, resulting in a constant velocity.

In general, there are models with on-axle and off-axle hitching. The latter accounts for the fact that some trucks have the connection to their trailers not above its axles. The greater the distance between axle and coupling, the greater is the influence of the hitching to the model. For the sake of simplicity, and as the model can be changed slightly to account for the off axle hitching, see [8], a model without hitching will be derived.

1 If there is no trailer at all, the vehicle is a car-like or single body vehicle. This is the simplest case.

In this setting, the truck will have front axle steering and rear axle traction[1]. For the control, it is assumed that full state information is instantly available, e.g. by sensor measurements, see Chapter 2.1. If not all states can be measured directly in real life applications, methods such as the Extended Kalman Filter (EKF) or 3DVar can be used [64].

In addition the following assumptions are made:

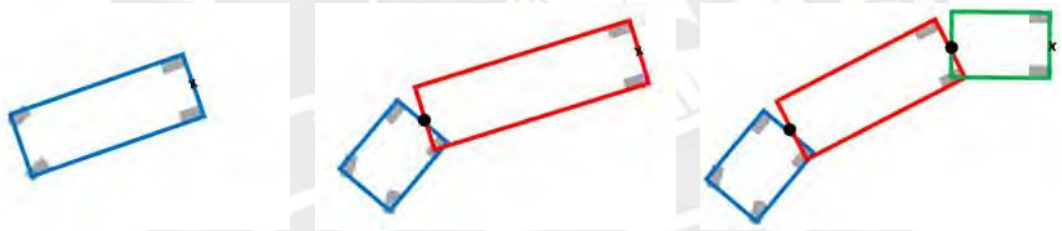
- Only two-dimensional movement in the x-y-Plane is being considered.
- Wheels and suspension are not modeled, and friction is neglected.

3.2 State Description

With constant velocity, a truck is controlled only by the steering. Therefore, the steering angle δ is the only input for a multibody vehicle. As in real life, the steering angle is limited

$$\delta_{min} \leq \delta \leq \delta_{maz} . \tag{3.1}$$

For symmetry, $\delta_{min} = -\delta_{maz}$ is assumed.



(a) Truck (b) Truck with one trailer (c) Truck with two trailers

Figure 3.1: Possible truck configurations

Considering a truck O_1 in the 2-D plane, it has coordinates (z_1, y_1) that represent the center of the rear axle and an angle ϑ_1, as can be seen in Figure 3.2. The angle ϑ_1 describes the absolute angles of the truck respective to the x-axis. The truck itself has constant dimensions, such as length $L_1 > 0$ and width w, as well as the steering angle δ. It is assumed that the steering can change instantly and therefore, no additional state is required for the steering angle, which will be an input only.

Figure 3.1b shows a truck-trailer vehicle. The truck has front steering and rear traction wheels. The trailer is passive with support rear wheels. The trailer is connected to the truck at the center of the traction axis and it is pulled or pushed by the truck when it moves forwards or backwards, respectively.

1 This configuration is arbitrary and can be changed by adjustments to the model, but it can be found in many real life examples.

For every additional trailer O_i, $i > 1$, there will be another angle ϑ_i, describing its orientation respective to the x-axis and length $L_i > 0$, resulting in its position at the rear axle (z_i, y_i). As truck and trailer or trailer and trailer, respectively, are coupled directly, there results an angle in between two following bodies $\vartheta_{ij} := \vartheta_i - \vartheta_j$, see Figure 3.3.

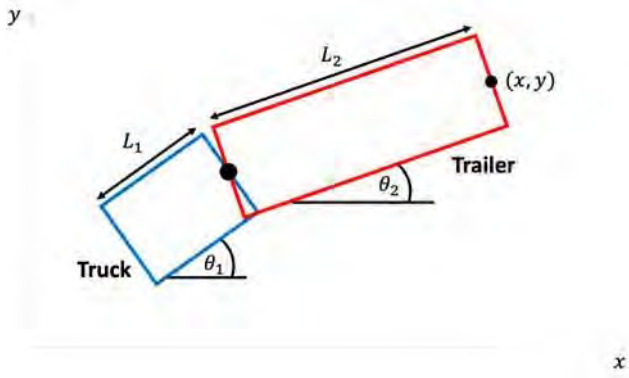


Figure 3.2: States in coordinate system for truck with one trailer

Knowing all absolute angles ϑ_i of the bodies, or one angle and all the angles ϑ_{ij} between the bodies, as well as the position of one body (z_k, y_k), $k \in \mathbf{N}$ describes the whole system. Thus, absolute angles and angles between bodies can be converted into each other:

$$\vartheta_{ij} = \vartheta_i - \vartheta_j \Rightarrow \vartheta_j = \vartheta_i - \vartheta_{ij} \tag{3.2}$$

Knowing one position leads directly to all positions by applying translational and rotatory transformations on the position using the lengths and angles. As the lengths and widths do not change over time, they are parameters instead of states.

The velocity of the truck is assumed to be constant. The velocities of the trailers may change with different angles, but it will be uniquely determined by the given states. Let the state for a truck with one trailer be given by

$$> := [z \quad y \quad \vartheta_2 \quad \vartheta_{12}]^T . \tag{3.3}$$

Consequently the state for a truck with two trailers by $> := [z \quad y \quad \vartheta_3 \quad \vartheta_{12} \quad \vartheta_{23}]^T$.

For the control of the given system, it can be useful to reduce the state representation. Therefore, a reduced state with $R \in \mathbf{R}^{r,n}$, $r < n$, $> \in \mathbf{R}^n$ and $\bar{>} \in \mathbf{R}^r$ is introduced

$$\bar{>} = R> . \tag{3.4}$$

Counterclockwise angles are positive and all angles range from $-\pi$ to π or from -180 degrees to 180 degrees, respectively.

During testing it became clear that $\delta_{max} < \frac{\pi}{2}$ is a useful constraint, as otherwise the behavior becomes instable.

The angles between bodies are limited, as trailers can never be in an angle over 180 degrees relative to the preceding body as they would collide. In fact, the angles are limited by a value far under 180 degrees, see Chapter 3.4. In addition, the absolute angles are, by convention, between -180 and 180 degrees[1]. Therefore, the state is limited

$$>_{min} \leq > \leq >_{maz} . \tag{3.5}$$

Transformation

To describe states in different coordinates, a transformation can be applied. Such transformations can shift or rotate the system. Given a rotation angle α, a velocity vector $\boldsymbol{v}$ can be transformed to the respective coordinates with a transformation matrix T:

$$\boldsymbol{v}_\alpha := T_\alpha \, \boldsymbol{v}$$

with

$$T_\alpha := \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix}$$

3.3 Model

A model for a multibody vehicle consists of a way to determine the new state after a given input u either by computing the derivative and integrating (continuous system)

$$\dot{>} = \tilde{f}(>, u) \tag{3.6}$$

with $> \in \mathbb{R}^n$, $u \in \mathbb{R}$ and $f : \mathbb{R}^{n+1} \to \mathbb{R}^n$ or computing the next state in a discrete system with timestep $mt \in \mathbb{R}$. The discrete system can be obtained by integrating over a timestep mt, see Chapter 3.5.

There are other ways to model a system, such as neural network representations, but continuous systems are considered first.

1 If the angle becomes greater than 180 degrees, it will be set to the respective value inside that region.

3.3.1 Single Track Model for one Trailer

One can assume that the left and right wheels move in a similar pattern, especially as the front wheels are steered with the same signals. The truck-trailer robot can then be modeled as chained bars as it is shown in Figure 3.3, regarding the vehicle as infinitely thin. This simplifies the modelling and results in the so-called *single-track model*.

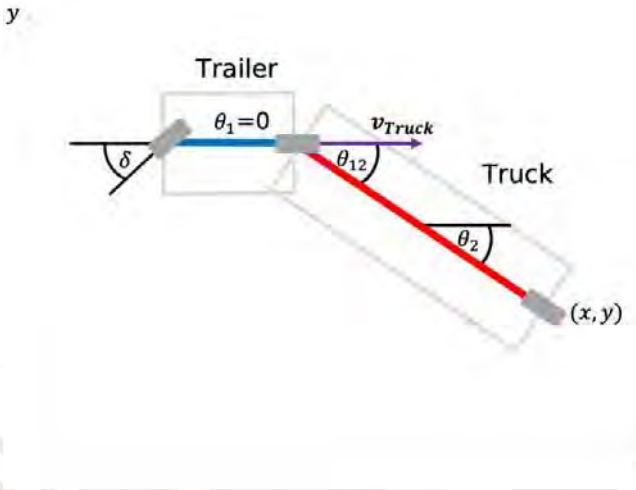


Figure 3.3: Simplified model of truck with trailer

Step 1: To start with, consider only the truck, as shown in Figure 3.1 a. With steering angle δ and positive velocities for backward movement, it is known from the literature or can easily be obtained by trigonometric identities, that

$$\dot{\vartheta}_1 = -\frac{v}{L_1}\tan(\delta). \tag{3.7}$$

Step 2: Let the coordinate system be, such that velocity vector $\boldsymbol{v}$ is parallel to the x-axis, as shown in Figure 3.3

$$\boldsymbol{v}_{Truck} = \begin{pmatrix} v \\ 0 \end{pmatrix}. \tag{3.8}$$

The x and y velocities of the truck are equal to the first and second component of $\boldsymbol{v}_{Truck}$.

The system can be rotated by any angle ϑ_1 without affecting the actual movement. This is why a simple position, i.e. $\vartheta_1 = 0$ is used.

Step 3: To obtain the model equations for the first trailer, a transformation for the angle ϑ_{12} has to be applied to obtain the resulting velocity affecting the trailer

$$\boldsymbol{v}_{Trailer} := \begin{pmatrix} v_{Trailer,1} \\ v_{Trailer,2} \end{pmatrix} = T_{\vartheta_{12}} \cdot v_{Truck} = T_{\vartheta_{12}} \cdot \begin{pmatrix} v \\ 0 \end{pmatrix} = \begin{pmatrix} v\cos(\vartheta_{12}) \\ v\sin(\vartheta_{12}) \end{pmatrix}. \tag{3.9}$$

Step 4a: The x component of $\boldsymbol{v}_{Trailer}$ is responsible for the movement of trailer. Therefore, the velocity in x and y direction of the trailer can be obtained by a rotation with the respective angle

$$\begin{bmatrix} \dot{z} \\ \dot{y} \end{bmatrix} = T_{\vartheta_2} \cdot \begin{bmatrix} v_{Trailer,1} \\ 0 \end{bmatrix} = \begin{bmatrix} v\cos(\vartheta_{12})\cos(\vartheta_2) \\ v\cos(\vartheta_{12})\sin(\vartheta_2) \end{bmatrix} \tag{3.10}$$

Step 4b: The y component of $\boldsymbol{v}_{Trailer}$ is responsible for the rotation of trailer. As the rotating trailer describes an arc with the identity $m\vartheta_2 \cdot L_2 = -ms$, the following limits hold:[1]

$$\lim_{t\to 0} \frac{m\vartheta_2}{t} = \dot{\vartheta}_2 \quad \text{and} \quad \lim_{t\to 0} \frac{ms}{t} = v_{Trailer,2} \tag{3.11}$$

resulting in

$$\dot{\vartheta}_2 = -\frac{v_{Trailer,2}}{L_2} = -\frac{v}{L_2}\sin(\vartheta_{12}). \tag{3.12}$$

Step 5: By definition, $\dot{\vartheta}_{12}$ can be computed from the derivatives of ϑ_1 and ϑ_2:

$$\dot{\vartheta}_{12} := \dot{\vartheta}_1 - \dot{\vartheta}_2 = -\frac{v}{L_1}\tan(\delta) + \frac{v}{L_2}\sin(\vartheta_{12}). \tag{3.13}$$

This leads to the final truck-trailer robot model, which is given by the following equations:[2]

$$\begin{aligned} \dot{z} &= v\cos(\vartheta_{12})\cos(\vartheta_2) \\ \dot{y} &= v\cos(\vartheta_{12})\sin(\vartheta_2) \\ \dot{\vartheta}_2 &= -\frac{v}{L_2}\sin(\vartheta_{12}) \\ \dot{\vartheta}_{12} &= \frac{v}{L_2}\sin(\vartheta_{12}) - \frac{v}{L_1}\tan(\delta). \end{aligned} \tag{3.14}$$

In the following, the state is reduced to x, such that $x := [z_1\ z_2\ z_3]^T = [y\ \vartheta_2\ \vartheta_{12}]^T$ with $u := \tan(\delta)$, resulting in a model of the form $\dot{x} = \tilde{f}(x, u) = f(x) + g(u)$. Not including the x-coordinate in the state vector simplifies the controller design process significantly. For the LQR controller for example, only three control parameters are required and no unnecessary information is used. At the same time, the robots navigation capabilities are not impacted, as the robot will reach any x-coordinate eventually.

The missing information on the x-position will be used to determine the switching between controllers, see Chapter 4.3. This way, it is possible to reach any desired x-position.

1 The minus sign comes from the orientation of the angles (*counter clockwise is positive*).
2 An algorithm to obtain the model equations for an arbitrary number of trailers is given in the next section.

3.3.2 General Single-Track Model

The model with $N \in \mathbb{N}$ bodies and $N + 1$ trailers can be derived by using the steps given in Section 3.3.1 and repeating steps 3–5 for all trailers. A visualization of the model with two trailers is given in Figure 3.1c.

Let in the following $i = 1, \ldots, N - 1$ and $j := i + 1$ describe two bodies that will be considered in a given iteration.

Step 3*: Given the velocity vector of the first body $\boldsymbol{v}_i$ and the full state of the second body $\boldsymbol{x}_{Full,j} = [z_j \quad y_j \quad \vartheta_j \quad \vartheta_{ij}]$ and its length L_j, the new velocity vector can be determined by coordinate transformation:

$$\boldsymbol{v}_j := \begin{matrix} v_{j,1} \\ \\ v_{j,2} \end{matrix} = T_{\vartheta_{ij}} \cdot \boldsymbol{v}_i \,. \tag{3.15}$$

Step 4a*: The resulting velocities in x and y direction can then be obtained using the x-component $v_{j,1}$ of the velocity:

$$\begin{matrix} \dot{z}_j \\ \dot{y}_j \end{matrix} = T_{\vartheta_j} \cdot \begin{matrix} v_{j,1} \\ 0 \end{matrix} = \begin{matrix} v \prod_{k=1}^{j} \cos(\vartheta_{k\,k+1}) \cos(\vartheta_j) \\ v \prod_{k=1}^{j} \cos(\vartheta_{k\,k+1}) \sin(\vartheta_j) \end{matrix} \tag{3.16}$$

Step 4b*: The resulting angular velocity of O_i can be computed by the y-component of the velocity $v_{j,2}$:

$$\dot{\vartheta}_j = \frac{v_{j,2}}{L_j} \,. \tag{3.17}$$

Step 5*: The angular velocity between the first and the second body can readily be determined by the angular velocities of the individual bodies:

$$\dot{\vartheta}_{ij} = \dot{\vartheta}_i - \dot{\vartheta}_j \,. \tag{3.18}$$

Using this general formulation of the model, the two-trailer model can be obtained in a straightforward way. The result is:

$$\begin{aligned} \dot{x} &= v \cos(\vartheta_{12}) \cos(\vartheta_{23}) \cos(\vartheta_3) \\ \dot{y} &= v \cos(\vartheta_{12}) \cos(\vartheta_{23}) \sin(\vartheta_3) \\ \dot{\vartheta}_3 &= -\frac{v}{L_3} \sin(\vartheta_{23}) \\ \dot{\vartheta}_{12} &= \frac{v}{L_2} \sin(\vartheta_{12}) - \frac{v}{L_1} \tan(\delta) \\ \dot{\vartheta}_{23} &= \frac{v}{L_3} \sin(\vartheta_{23}) - \frac{v}{L_2} \sin(\vartheta_{12}) \,. \end{aligned} \tag{3.19}$$

3.3.3 Linearization

To design a linear controller (such as the LQR controller), it is necessary to compute the linearization of a nonlinear system. To obtain a linear system of the form $\dot{x} = Ax + Bu$ from a nonlinear system of the form $\dot{x} = f(x) + g(u)$, the *Jacobian linearization* can be used around a certain setpoint (x^*, u^*).

The linearized system will be used for the design of linear controllers and theoretical analysis only. For simulation, design of nonlinear controllers and training of neuro-controllers, the nonlinear model will be used.

Jacobian linearization

The equation for the linearization of a multivariable function $f(z, y)$ at a point (a, b) is:

$$f(z,y) \approx f(a,b) + \frac{\partial f(z,y)}{\partial z}\Big|_{(a,b)} (z-a) + \frac{\partial f(z,y)}{\partial y}\Big|_{(a,b)} (y-b)$$

A nonlinear system of the form $\dot{z} = \tilde{f}(z,u) = f(z) + g(u)$ can be linearized at a setpoint (z^*, u^*) to obtain a linear approximation, valid around the setpoint. For the case $z^* = 0$ and $u^* = 0$, the linear system has the form $\dot{z} = Az + Bu$ with

$$A := \frac{\partial \tilde{f}(z,u)}{\partial z}\Big|_{\substack{z=z^* \\ u=u^*}} = \frac{\partial f(z)}{\partial z}\Big|_{z=z^*} \qquad B := \frac{\partial \tilde{f}(z,u)}{\partial u}\Big|_{\substack{z=z^* \\ u=u^*}} = \frac{\partial g(u)}{\partial u}\Big|_{u=u^*}$$

In the following, the linearization will be conducted for the one trailer model for the general case with $x^* = [y^* \quad \vartheta_2^* \quad \vartheta_{12}^*]$, $u^* = \tan(\delta^*)$. Afterwards, the obtained linearization will be used to compute the explicit matrices A and B at one setpoint.

$$\begin{aligned} A = \frac{\partial f(z)}{\partial z}\Big|_{z=z^*} &= \begin{bmatrix} \frac{\partial f_1(z)}{\partial z_1} & \frac{\partial f_1(z)}{\partial z_2} & \frac{\partial f_1(z)}{\partial z_3} \\ \frac{\partial f_2(z)}{\partial z_1} & \frac{\partial f_2(z)}{\partial z_2} & \frac{\partial f_2(z)}{\partial z_3} \\ \frac{\partial f_3(z)}{\partial z_1} & \frac{\partial f_3(z)}{\partial z_2} & \frac{\partial f_3(z)}{\partial z_3} \end{bmatrix}\Big|_{z=z^*} \\ &= \begin{bmatrix} 0 & v\cos(\vartheta_2^*)\cos(\vartheta_{12}^*) & -v\sin(\vartheta_2^*)\sin(\vartheta_{12}^*) \\ 0 & 0 & -\frac{v}{L_2}\cos(\vartheta_{12}^*) \\ 0 & 0 & \frac{L_2}{v}\cos(\vartheta_{12}^*) \end{bmatrix} \end{aligned} \tag{3.20}$$

$$B = \frac{\partial g(u)}{\partial u}\Big|_{u=u*} = \begin{bmatrix} \frac{\partial g_1(z)}{\partial u} \\ \frac{\partial g_2(z)}{\partial u} \\ \frac{\partial g_3(z)}{\partial u} \end{bmatrix}_{u=u*} = \begin{bmatrix} 0 \\ 0 \\ -\frac{v}{L_1} \end{bmatrix}_{u=u*} = \begin{bmatrix} 0 \\ 0 \\ -\frac{v}{L_1} \end{bmatrix}. \tag{3.21}$$

Now consider the important setpoint $x^* = [y \quad 0 \quad 0]^T$ and $u^* = 0$. The linear state-space equation $\dot{x} = Ax + Bu$ is given by:

$$\begin{bmatrix} \dot{y} \\ \dot{\vartheta}_2 \\ \dot{\vartheta}_{12} \end{bmatrix} = \begin{bmatrix} 0 & v & 0 \\ 0 & 0 & -\frac{v}{L_2} \\ 0 & 0 & \frac{v}{L_2} \end{bmatrix} \begin{bmatrix} y \\ \vartheta_2 \\ \vartheta_{12} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -\frac{v}{L_1} \end{bmatrix} u. \tag{3.22}$$

The *controllability* analysis of the linearized system can be found in Appendix A.1.1.

The values of y^* and u^* do not influence the linearized matrices A and B as linearization is independent of y and u.

3.4 Model Behavior

The derived models from Section 3.3 are *nonlinear*, as there are nonlinear functions, such as $\sin(\cdot)$ and $\cos(\cdot)$ in it. As there is only one control variable δ and more than one state to be controlled (even for most of the reduced state cases), the systems are *underactuated*. In addition, those robots have *nonholonomic* behavior, as they cannot move in any direction instantly and therefore requiring turns to move into certain directions.

Considering the linearized system, the roots in the open loop can be obtained:

$$det(sI - A) = det\begin{bmatrix} s & -v & 0 \\ 0 & s & \frac{v}{L_2} \\ 0 & 0 & s - \frac{v}{L_2} \end{bmatrix} = s^3 - s^2 \frac{v}{L_2} = s^2 \left(s - \frac{v}{L_2}\right). \tag{3.23}$$

Therefore, the behavior in backward motion $(v > 0)$ is open loop unstable, as $s_1 = \frac{v}{L_2} > 0$.

This can be seen in Figure 3.4, as the angle ϑ_{12} converges to 0 in forward movement and grows even over $90^\circ = \frac{\pi}{2}$ in the backward movement.

If an angle between two bodies ϑ_{ij} reaches 90° while driving backwards, the velocity in both directions vanishes as $cos(\vartheta_{12}) = cos(90^\circ) = 0$:

$$\begin{aligned} \dot{z} &= v\cos(\vartheta_{12})\cos(\vartheta_2) = 0 \\ \dot{y} &= v\cos(\vartheta_{12})\sin(\vartheta_2) = 0. \end{aligned} \tag{3.24}$$

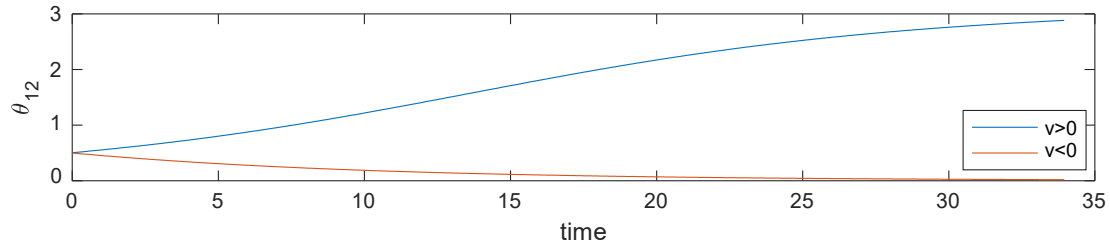


Figure 3.4: Open loop behavior for offset in ϑ_{12}

This situation is called *jack-knife* state, which makes the system hard to control. This is why it is aspired to avoid the jack-knife state. This becomes increasingly complex if there are several trailers, as then several angles ϑ_{ij} can cause the jack-knife state. Figure 3.5 illustrates several angles ϑ_{12} between truck and trailer. Note that (c) is the jack-knife state and (d) is impossible.

(a) $\vartheta_{12} = 0^\circ$ (b) $\vartheta_{12} = 45^\circ$ (c) $\vartheta_{12} = 90^\circ$ (d) $\vartheta_{12} = 135^\circ$

Figure 3.5: Visualizations of different ϑ_{12}

Analyzing equation (3.14), it can be observed, that for ϑ_{ij} with $v > 0$ to increase or decrease, it is sufficient to decrease or increase δ, respectively[1]. This is, because the term $\frac{v}{L_2}\sin(\vartheta_{12})$ only changes with ϑ_{12} (which should be changed), as v and L_2 are constant. The second term $-\frac{v}{L_1}\tan(\delta)$ only changes with δ, which can be actively changed. As $\tan(\cdot)$ grows monotonously, δ has to be positive for ϑ_{12} to shrink and vice versa. Figure 3.6 shows ϑ_{12} for $v > 0$ with $\delta = \delta_{max}$ and $\delta = \delta_{min}$, respectively.

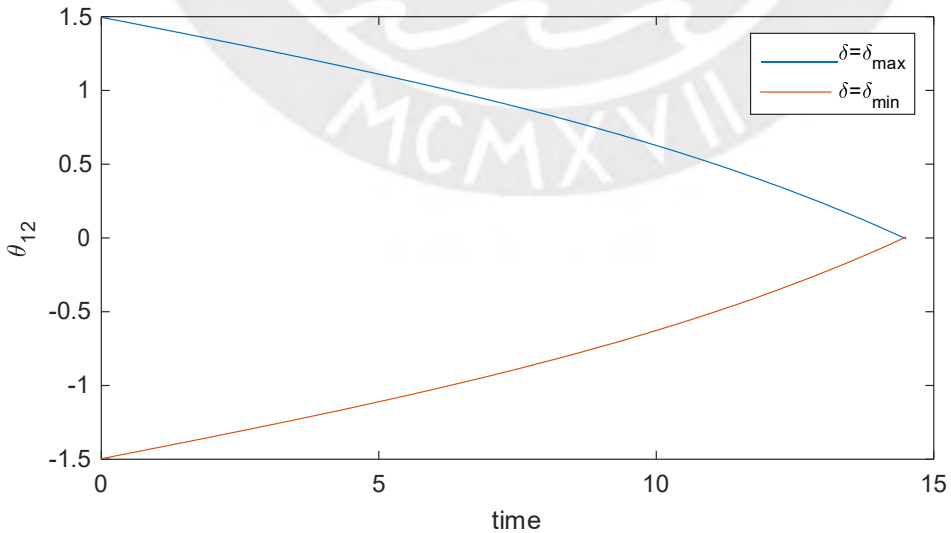


Figure 3.6: Reducing ϑ_{12} with δ

1 As the sign changes for $v < 0$, the required sign for δ changes, too.

3.5 Computation

To simulate the system $\dot{x} = \tilde{f}(x, u)$, given an input u, the systems differential equations can be evaluated. Using a small timestep Δt and integrating over the derivatives, the new states can be obtained. Let $t_{k+1} = t_k + \Delta t$ and note that $\tilde{f}(x, u)$ is independent of time t

$$\begin{aligned} x_{k+1} &= x_k + \int_{t_k}^{t_{k+1}} \dot{x}(\tau)\, d\tau = x_k + \int_{t_k}^{t_{k+1}} \tilde{f}(x(\tau), u(\tau), \tau)\, d\tau \\ &= x_k + \int_0^{\Delta t} \tilde{f}(x_k, u_k)\, d\tau \approx x_k + \tilde{f}(x_k, u_k)\Delta t\,. \end{aligned} \tag{3.25}$$

To compute the new states for a timestep $t_{total} >> \Delta t$, the model can be evaluated several times, see Figure 3.7.

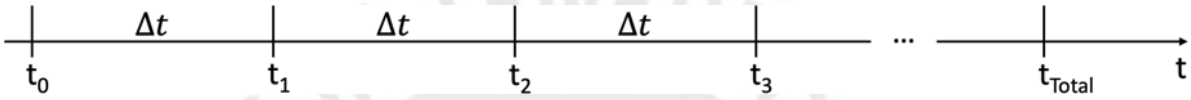


Figure 3.7: Sampling with Δt

Many methods are known to solve the system with initial conditions $x(t_0) = x_0$ more precise than the given approximation and therefore, no further investigation is conducted on this topic. For completeness, Appendix A.1.2 presents another method.

To simplify computation of the next states for the simulation, a compact function can be used[1]. The algorithm that implements this model including the reduced states and noise, is shown in Algorithm 1. Note that the function f is the ones derived in Section 3.3.

input : State $x = [z\ y\ \vartheta_N\ \vartheta_{12}\ \vartheta_{23}...]$, control input $u = \tan(\delta)$, parameters $p = [L, v] = [L_1, L_2, ..., v]$ and indicators *reduced* and *noise* with standard deviation σ
output: Function value *func*

```
Model:
// compute f with given parameters
func = f(x, u, p)
if reduced then
    // ignoring first entry
    func = func(2 : end)
if noise then
    // adding noise
    func = func + σ ∗ randn(size(func))
return func
```

Algorithm 1: Function to compute model related variables

1 This way, changes to the model can be done easy and fast, as only one function has to be changed.

3.6 Environment

The robot is located in an environment. Let E_t denote the environment at time t. The environment is a tuple consisting of the truck with its trailers O, information about the target state or trajectory h_{target}, information about objects or buildings B close to the robot and the time t

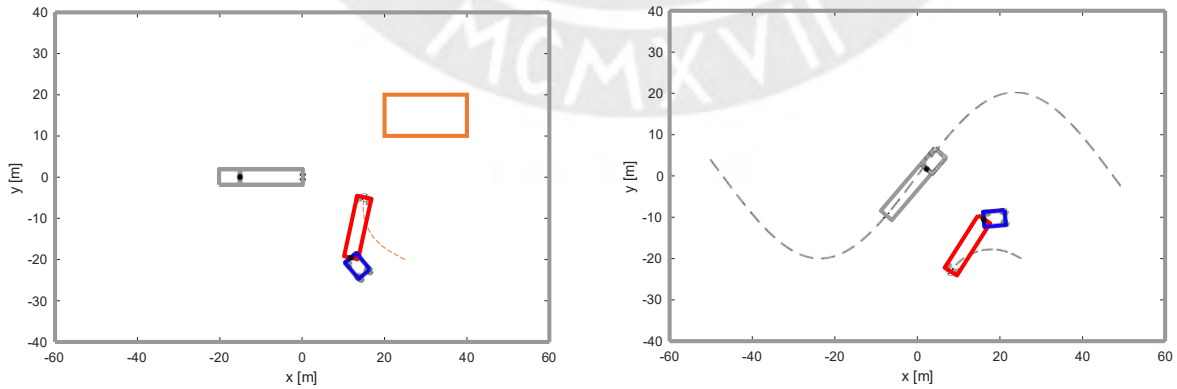
$$E_t := \{O_t, h_{target}, B, t\} \qquad \text{with} \qquad O_t := \{N, >_t, M, P, t\} \tag{3.26}$$

and the robot itself being a tuple of the number of bodies N, the state $>_t$ with full state information, the model M, its parameters[1] P and time t.

The environment can be interpreted in interacting layers, each holding a certain part of the information. For visualization purposes, the operation area is surrounded by a grey frame and buildings are displayed in orange (Layer 1). Targets are shown with a grey truck or a grey dashed line, respectively (Layer 2). The robot has a simplified visualization consisting of the truck in blue and the trailer in red. The path that the robot already moved is shown in an dashed orange line (Layer 3). The layers are shown in Figure 3.8.

Figure 3.8: Visualizations of different layers of the environment

Two simple environments, as generated by the framework, are shown in Figure 3.9. Figure 3.9a has one building and a fixed target state. Figure 3.9b represents an environment with target trajectory and the desired state at time t is shown.



(a) Sample environment with target state

(b) Sample environment with target trajectory

Figure 3.9: Visualizations of different environments with truck-trailer

1 Namely the velocity v, the steering angle δ, the maximal steering angle δ_{max} and the lengths $L = [L_1, L_2, \ldots]$.

4 Control Strategy

*For the **control** of autonomous multibody vehicles, a strategy needs to be developed. This includes pathplanning, direction switching, control of the steering angle and jack-knife avoidance.*

The overall control strategy is shown in Figure 4.1. The given reference signal r is used to compute a desired state x^* for the controller using a *pathplanning* algorithm. The desired state might change over time. The *switching* module determines which controller will be used to compute the control variable, e.g. if the truck should move forwards or backwards. Next, the chosen *controller* computes the control signal u based on the desired state and the measurement of the current state x. Lastly, the *jack-knife avoidance* module transforms the control signal to $\tilde{u}$ such that it cannot lead to a jack-knife position.

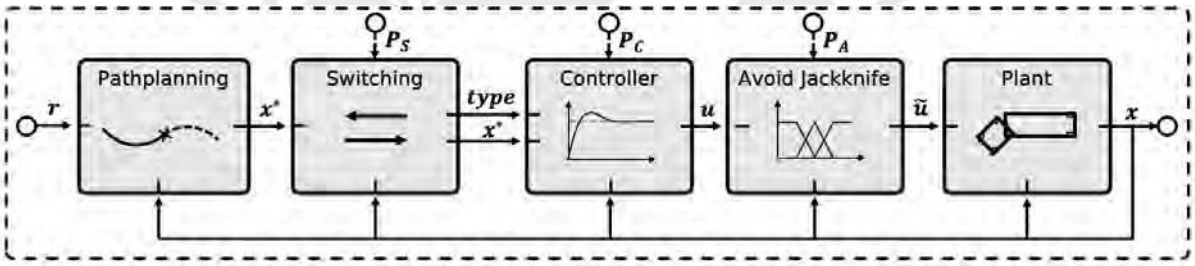


Figure 4.1: Control strategy

The switching module, the controller and avoid jack-knife module need their respective parameters. For example, the controller module has precalculated parameters for all possible types of controllers.

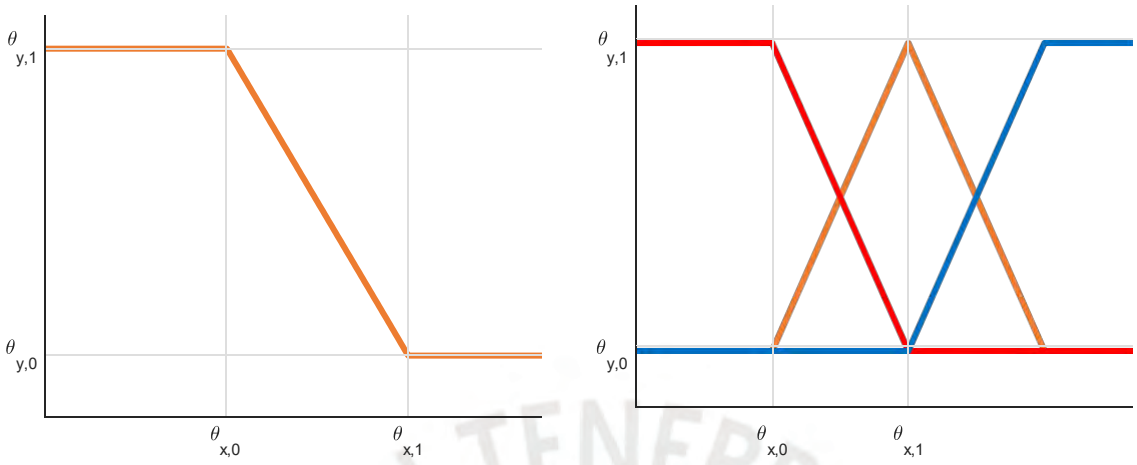
4.1 Jack-Knife Avoidance

If a truck-trailer reaches the jack-knife state, it is hard to control, especially as the linearized model is only valid around the linearization point. To avoid the jack-knife position, the concept proposed in [24] is used. In terms of jack-knife avoidance, the angle between the bodies, i.e. ϑ_{12} is important.

As mentioned in Section 3.4, the maximum or minimum steering angle is effective to reduce a low or high ϑ_{12}, respectively. To smooth the transition between the steering angle (determined by the controller for ϑ_{12} close[1] to 0) and the maximum or minimum

1 How close is determined by the fuzzy membership function.

steering angle, a fuzzy membership function is introduced, see Figure 4.2[1].



(a) Type A (b) Type B

Figure 4.2: Membership functions

Mathematically, they have the form

$$f_A(\vartheta_{12}, \vartheta_{z,0}, \vartheta_{z,1}, \vartheta_{y,0}, \vartheta_{y,1}) := \begin{cases} \vartheta_{y,0} & , \text{if } \vartheta_{12} < \vartheta_{z,0} \\ (\vartheta_{12} - \vartheta_{z,0}) \cdot \frac{\vartheta_{y,1} - \vartheta_{y,0}}{\vartheta_{z,1} - \vartheta_{z,0}} + \vartheta_{y,0} & , \text{if } \vartheta_{z,0} < \vartheta_{12} < \vartheta_{z,1} \\ \vartheta_{y,1} & , \text{if } \vartheta_{12} > \vartheta_{z,1} \end{cases} \tag{4.1}$$

$$f_B(\vartheta_{12}, \vartheta_{z,0}, \vartheta_{z,1}, \vartheta_{y,0}, \vartheta_{y,1}) := \begin{cases} f_A(\vartheta_{12}, \vartheta_{z,0}, \vartheta_{z,1}, \vartheta_{y,0}, \vartheta_{y,1}) & , \text{if } \vartheta_{12} < \vartheta_{z,1} \\ f_A(\vartheta_{12}, \vartheta_{z,1}, 2\vartheta_{z,1} - \vartheta_{z,0}, \vartheta_{y,1}, \vartheta_{y,0}) & , \text{if } \vartheta_{12} > \vartheta_{z,1} \end{cases}$$

$$f_A(\vartheta_{12}, \vartheta_{z,1}, 2\vartheta_{z,1} - \vartheta_{z,0}, \vartheta_{y,0}, \vartheta_{y,1}) \tag{4.2}$$

with $f_A : \mathbf{R} \to \mathbf{R}$ and $f_B : \mathbf{R} \to \mathbf{R}^3$. Note that $\vartheta_{z,0}$, $\vartheta_{z,1}$, $\vartheta_{y,0}$ and $\vartheta_{y,1}$ are parameters of the function and a function of type B consists of several separate fuzzy functions of type A.

The membership functions determine y-value(s) for any point on the x-axis. Note that functions of type B have more than one, i.e. three y-values associated with any x-value.
Example: In Figure 4.2a, the value for $\vartheta_{12} = \vartheta_{z,0}$ is $f_A(\vartheta_{z,0}, \vartheta_{z,0}, \vartheta_{z,1}, \vartheta_{y,1}, \vartheta_{y,0}) = \vartheta_{y,1}$.
In Figure 4.2b, the values for $\vartheta_{12} = \vartheta_{z,0}$ are $f_B(\vartheta_{z,0}, \vartheta_{z,0}, \vartheta_{z,1}, \vartheta_{y,1}, \vartheta_{y,0}) = [\vartheta_{y,1} \quad \vartheta_{y,0} \quad \vartheta_{y,0}]^T$.
Let the membership function for the jack-knife avoidance be $f_m(\vartheta_{12}) := f_B(\vartheta_{12}, -\frac{\pi}{3}, 0, 0, 1)$.
Then the control signal u will be transformed by

$$\tilde{u} = f_m(\vartheta_{12})^T \cdot [\delta_{min} \quad u \quad \delta_{max}]^T = f_{m,1}(\vartheta_{12})\delta_{min} + f_{m,2}(\vartheta_{12})u + f_{m,3}(\vartheta_{12})\delta_{max}. \tag{4.3}$$

1 Fuzzy functions can be nonlinear, e.g. S-shaped as well.

At least one component of f_m is zero. In the case of $\vartheta_{12} = 0$, Equation (4.3) is reduced to $\tilde{u} = u$ as $f_m(\vartheta_{12})^T = f_m(0)^T = [0 \quad 1 \quad 0]$.

4.2 Controller

Consider the linear system[1]

$$\dot{x} = Ax + Bu \tag{4.4}$$

with given matrices $A \in \mathbf{R}^{n \times n}$ and $B \in \mathbf{R}^n$. The state is $x \in \mathbf{R}^n$ and the control input $u \in \mathbf{R}$. Let the target state be $x^* = \mathbf{0}_n$ and $u^* = 0$.

Finding a controller can be reduced to the search for a control law to determine the control input u. A simple controller for state-space systems has the form

$$u = -Kx = -\sum_{i=1}^{n} k_i x_i \tag{4.5}$$

with control gain $K \in \mathbf{R}^{1 \times n}$. In the closed loop system, the dynamics of the state are given by

$$\dot{x} = Ax + Bu = Ax + B(-Kx) = (A - BK)x \,. \tag{4.6}$$

The system is stable, if and only if

$$eig(A - BK) \subseteq \mathbf{C}^- \,. \tag{4.7}$$

Therefore, the control gain K can be determined such that the eigenvalues have a specified position in the complex plane [34].

A *fuzzy* control approach is presented in Appendix A.1.3.

Remarks

- For a target state $\tilde{x}^* = 0$ and $\tilde{u}^* = 0$, a simple transformation $\tilde{x} := x - \tilde{x}^*$ and $\tilde{u} := u - \tilde{u}^*$ leads to a system with the required properties: $\dot{\tilde{x}} = A\tilde{x} + B\tilde{u}$.

  Proof: Let $\tilde{u} = -K\tilde{x}$ and it follows $\dot{\tilde{x}} = (A - BK)\tilde{x}$ with the same K as before.

- The control variable can then be computed as

$$u = \tilde{u} + \tilde{u}^* = -K\tilde{x} + \tilde{u}^* = -K(x - \tilde{x}^*) + \tilde{u}^* \,.$$

- To turn the continuous system into a discrete system, the approach from Section 3.5 with a small mt can be used.

1 Note that x and its derivatives are functions of time, but for improved readability, the arguments are left out unless to underline the time dependence.

4.2.1 Linear Quadratic Optimal Control (LQR)

Optimal Control Theory

As described in [65], an optimal controller can be designed with the quadratic cost function

$$J(z_0, u) := \int_0^\infty z(t)^T Q z(t) + u(t)^T u(t) dt \tag{4.8}$$

with $Q \in \mathbf{R}^{n*n}$ as $z \in \mathbf{R}^n$ and $u \in \mathbf{R}$. To minimize the cost function J, the smallest real symmetric positive semidefinite solution of the *Algebraic Riccati Equation (ARE)*, called P^-, has to be found. Smallest meaning that $z^T P z \geq z^T P^- z, \forall z$ and for every real symmetric $P \geq 0$ satisfying the ARE. The ARE has the form $A^T P + PA - PBB^T P + Q = 0$.

Then for the minimum $J_{min}(z_0) := \inf\{J(z_0, u) \mid u \in \mathbf{R}\} = z_0^T P^- z_0$ holds. The respective optimal control law using P^- is $u = -B^T P^- z = -K z$.

Applied to the truck-trailer system with weight matrix $Q_{LQR} = \begin{bmatrix} 128 & 0 & 0 \\ 0 & 100 & 0 \\ 0 & 0 & 3000 \end{bmatrix}$, the control gain K_{LQR} can be computed. The resulting control law for $v < 0$ is given by

$$u = -K_{LQR,v<0} > = -[11.3 \quad 137.7 \quad -55.9] > = -11.3z_1 - 137.7z_2 + 55.9z_3, \tag{4.9}$$

while for $v > 0$, the result is $K_{LQR,v>0} = [-11.3 \quad 137.7 \quad 55.3]$. Figure 4.3 shows the step response for the target point $[1\ 0\ 0]^T$, starting from $[0\ 0\ 0]^T$ with $v < 0$.

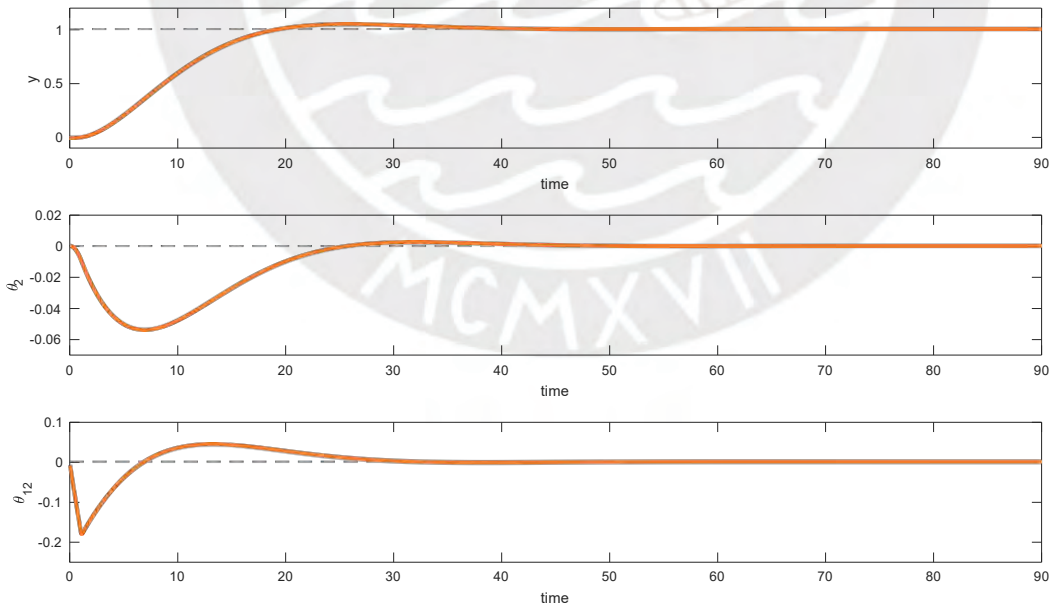


Figure 4.3: Step response of LQR controller for target point $[1\ 0\ 0]^T$

The dynamical system (A,B), and Q determine P^- and therefore K_{LQR} differs for different parameter values and for different weight matrices Q.

4.2.2 Model Predictive Control (MPC)

The general idea of Model Predictive Control (MPC) is to transform the control into an optimization problem to compute the optimal control input within a control horizon t_c . The optimization problem can take current states, limitations and constraints into account. Therefore, the future states are predicted for the prediction horizon t_p using a mathematical model of the dynamics [41]. The control horizon can be equal the prediction horizon, as shown in the visualization of the overall concept in Figure 4.4. If $t_c < t_p$, the last control input can be used for all following timesteps, such that without loss of generality

$$t_c = t_p = N \cdot \mathrm{m}t \tag{4.10}$$

with mt being the sampling time and N the resulting number of intervals for the optimization.

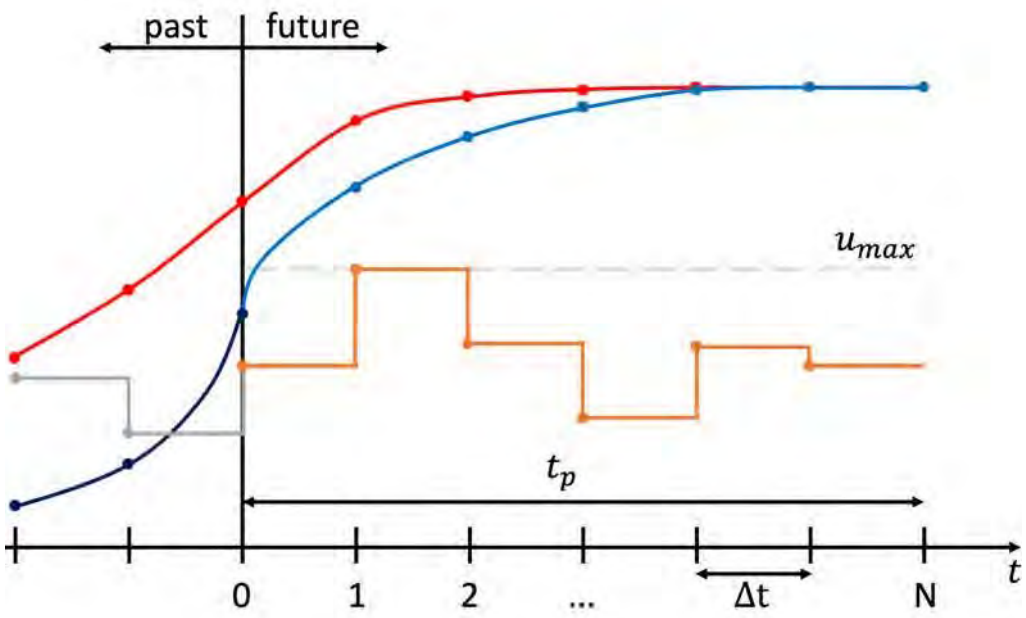


Figure 4.4: Concept of the MPC control
Reference Trajectory (red), Predicted Output (blue), Measured Output (dark blue), Computed Control Input (orange) and Past Control Input (grey)

The optimization problem can be formulated as follows

$$\min_{u}\{J = >_N^T P >_N + \sum_{k=0}^{N-1} (>_k^T Q >_k + u_k^T R\, u_k)\} \tag{4.11}$$

with equality constraints $G(>, u) = 0$ and inequality constraints $H(>, u) \leq 0$ with limits for inputs and states

$$u_{min} \leq u_k \leq u_{maz} \tag{4.12}$$

$$>_{min} \leq >_k \leq >_{maz} . \tag{4.13}$$

The step responses for one MPC controller using a linear and another using a nonlinear model are shown in Figure 4.5. The linear MPC (lMPC) has $t_i = 6s$, while the nonlinear MPC (nMPC) has $t_c = 1s$. Both controllers use $Q_{MPC} = \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{bmatrix}$ and $t_p = 15s$ with $mt = 0.5s$.

It can be seen that the nMPC performs better overall, while the lMPC is slightly faster for the cost of a lot of overshoot. In terms of computing time for the 90 seconds simulations, both MPCs are slow, while the nMPC controller takes the most time (lMPC: 28.8s; nMPC: 176.3s).

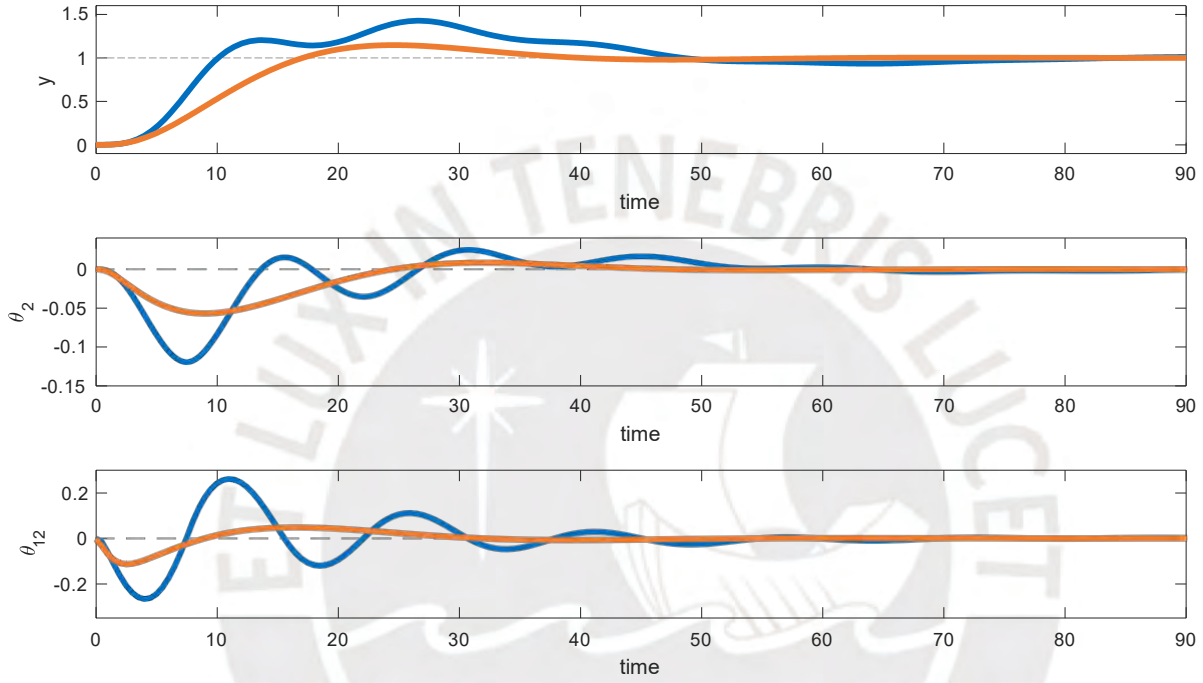


Figure 4.5: Step response of MPC controller for target point $[1\ 0\ 0]^T$
linear MPC (blue) and nonlinear MPC (orange)

Remarks

- The dynamics of the system are part of the equality constraints.
- As the discrete values $>_k$ and u_k are required, a discrete model has to be used.
- The optimization problem can be solved by a suitable method, as available e.g. in MATLAB or presented in Appendix A.1.4, to find the best control values u.
- The problem will be solved in every sampling step, but only the first control variable u_0 will be used as input to the system.

4.3 Switching between Controllers

In this work, switching between controllers refers to the changing of the driving direction and the respective change of control parameters. Changing the controller based on the velocity was done in [38]. One can think about changing the controller itself, e.g. change from a MPC controller to a neuro-controller, as this might improve performance for certain situations. For example, a certain controller might be better for high accuracy parking maneuver, while another is better in avoiding obstacles or for energy optimal navigation.

4.3.1 Collision Detection

If the signal of the current controller results in a collision, the direction should be switched. This is why collisions have to be detected. For the simulations it is sufficient to detect only real collisions[1]. As described in Chapter 3, the model consists of one rectangle per body, i.e. truck and the trailers. The environment is assumed to consist only of convex polygonal shapes[2]. For an efficient implementation of collision detection, the *Separating Axis Theorem (SAT)* can be used.

Separating Axis Theorem

Convex polygons do not intersect with each other if and only if they can be separated by a straight line, i.e. all vertices of the polygons are on different sides of the line. Furthermore, if the polygons do not overlap, one of the edges will be such a separating axis, see Figure 1. The separating axis is shown in green. None, one or several separating axis may exist.

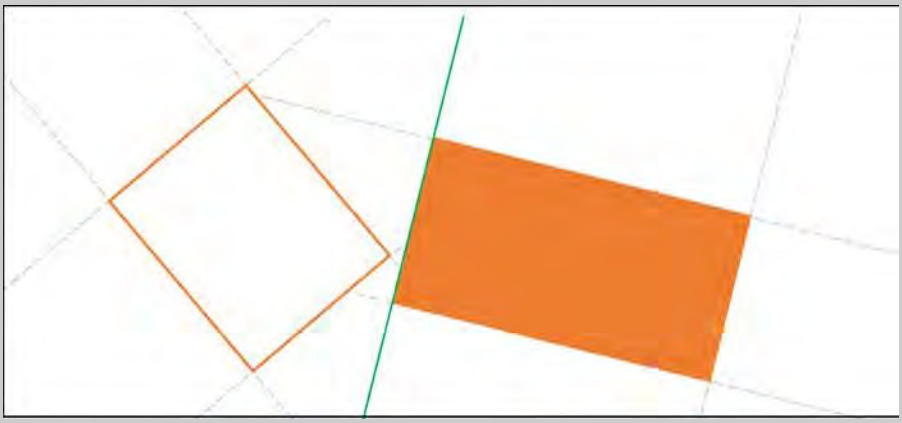


Illustration 1: Visualization of the SAT: Separation of two rectangles

Therefore, it can be checked if any of the edges separates the two polygons. The sign of the dot product can be used to check on which side of an axis a vertice is. If not all vertices of one polygon are on the same side, then the edge does not separate the shapes.

An implementation of the SAT can be found in Appendix A.1.5.

1 In reality, as sensor measurements are noisy, there should be a safety margin to avoid collisions that result from wrong measurements.

2 Non polygonal shapes can be placed inside infinitesimally larger polygonal objects [13].

Switching to avoid Collisions (switching type 1)

First, if a collision in the next step is detected, then a change of the direction will avoid that collision. Therefore, the driving direction will be changed if a collision is detected.

Second, if a cost function that describes the progress of the control has certain properties, the direction should be changed to improve performance. This is, when the cost function indicates that the robot is driving away from the target[1].

4.3.2 Cost Function

A cost function J is a function that evaluates a state or a situation[2]. To do this, the current state of the robot $>_t$ and the target state $>_{target}$ can be used to compute the error

$$e_t := >_t - >_{target} . \tag{4.14}$$

Then either the error itself can be used as a cost function, e.g. $J_t := e_t^T e_t$. Another approach is to use a positive definite symmetric[3] weight matrix $R = R^T > 0$, which can be used to transform the error, such that

$$J_t := e_t^T \, R \, e_t. \tag{4.15}$$

The value of the cost function is then used to determine the best point in time to switch the controllers.

Aside from collision avoidance, a cost function is used to determine switches. The overall logic is visualized in Figure 4.6 and explained afterwards.

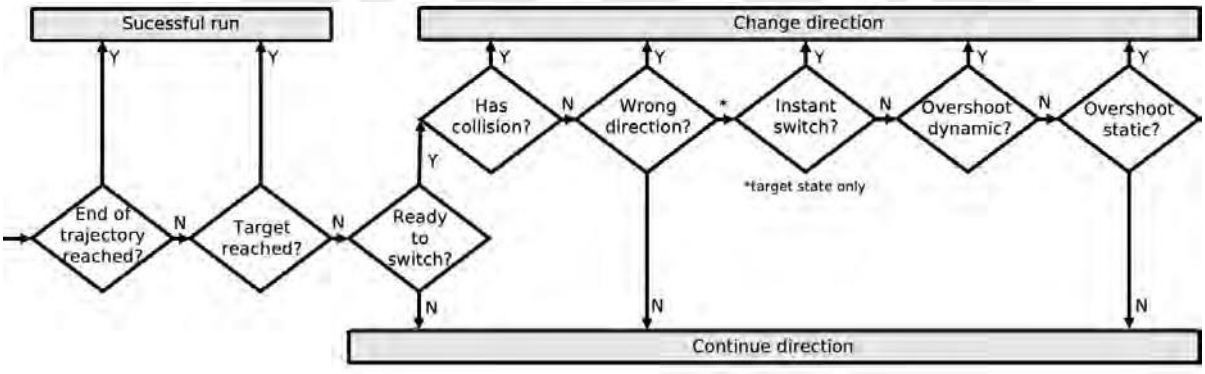


Figure 4.6: Logic used to determine switches

1 This is especially relevant for trajectory following, as usually there is a correct direction to follow the trajectory and going in the other direction does not make sense.

Reaching the Target (stopping)

The first, straightforward case in terms of the cost function is, when the system reached its target. That is the case, if the value of the cost function is at or below a threshold $S \geq 0$, $S \in \mathbf{R}$:

$$J_t \leq S \quad \Rightarrow \quad v = 0\,. \tag{4.16}$$

The condition $S = 0$ is very restrictive, as it might be impossible to reach the target state exactly, e.g. because of noise.
If the velocity is set to zero, the resulting controller is the one that always returns $u = 0$, as the system should not move anymore.

Trajectory Direction (switching type 2)

The driving direction is encoded in the given trajectory by order of the points, i.e. the second point should be targeted after the first point. This is the truck is moving in the wrong direction if previous points become closer to the truck during driving. Then, the direction is changed.

Instant Switching (switching type 3)

If the cost function increases in the first few steps of the simulation, this suggests that the truck is moving away from the target. In principle, this might be the right thing to do, but in other cases it is better to move in the other direction. Since it cannot be known if the cost function will decrease in the other direction, but there is a high chance of it doing so, the direction will be changed in such cases.

Dynamic Overshooting (switching type 4)

Next, the driving direction should be changed if the current configuration does not improve the cost over a certain period of time. Therefore, the minimum costs $J_{min}(t, \tilde{t})$ since time $\tilde{t}$ until time t with $\tilde{t} \leq t$ are introduced:

$$J_{min}(t, \tilde{t}) := \min_{\tilde{t} \leq \tau \leq t} J_\tau\,. \tag{4.17}$$

Let in the following $\tilde{t}$ denote the time since the last switch. If now the costs rise much higher than the minimum costs since the last switch, then the system moves away from the target – potentially in the wrong direction – and the controller should switch[1]

$$J_{min}(t, \tilde{t}) \leq J_t - \rho_1 \quad \Rightarrow \quad v = -v \tag{4.18}$$

with $\rho_1 \in \mathbf{R}$ being a hysteresis coefficient, e.g. allowing the system to turn. This is, because of the nonholonomic system properties, as the truck is not able to change its angle instantly.

1 The driving direction changes, if and only if $v = -v$.

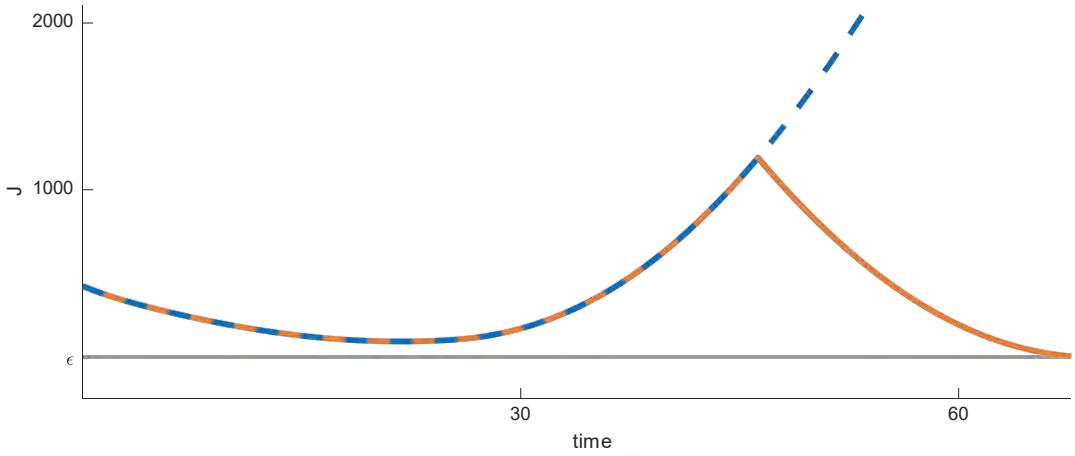


Figure 4.7: Cost function example: with switch (orange) and without switch (blue)

The dashed blue curve in Figure 4.7 shows how the cost function from equation (4.15) with

$$R = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 25 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.19}$$

starting from $[\,20,\ 0,\ -\ 1,\ 0.1]^T$ and targeting the origin behaves. In this case, no switching is done. Note that the third state (ϑ_2) has a larger factor, while the fourth state (ϑ_{12}) is ignored entirely. This is, because there should not be a direction change because of the angle between truck and trailer, as it might be required for a turn. The third state needs a larger multiplier to compensate that its values in *rad* are smaller than the position of the truck in meters.

The orange curve in Figure 4.7 illustrates the influence on the cost function with a switch starting according to Equation (4.18) with $\rho_1 = 3000$. Note that by the switching, the target could be reached.

Static Overshooting (switching type 5)

In certain situations, it can be useful to change the direction dependent on the overall minimum costs as well. This is why as second condition for the situation that the system moves away from the target is introduced:

$$J_{min}(t, \tilde{t}) \leq J_t - J_{min}(t, 0) - \rho_2 \qquad \Rightarrow \qquad v = -v \tag{4.20}$$

with $J_{min}(t,\ 0)$ begin the overall minimum costs and another constant $\rho_2 \in \mathbf{R}$, usually smaller than ρ_1.

4.4 Pathplanning

Let r be the reference or the target state. In case of a non-moving target state, r is one state of the robot that should be reached. In case of a target trajectory, the reference is a function of points to be followed. This function is called *path*.

To follow paths, such as lines and nonlinear functions, a method is needed to compute an instantaneous desired state (IDS) x^* that is used to compute the control input u. Let in the following $x^* = [z^* \quad y^* \quad \vartheta_2^* \quad \vartheta_{12}^*]$, the desired state for a truck with one trailer. The value of x^* changes over time unless it is identical with the reference or target state.

There are many ways to compute x^* for all kinds of function types, e.g. linear functions, circles and trigonometric functions. In the following, a generic algorithm is derived, that can find or approximate the desired position for more general types of trajectories.

Target State

In case of a parking position or a simple target state with $\vartheta_2 = 0$, this state can be used as desired position in any step. As the state not only consists of the position, but also of the orientation, it may be necessary to plan a straight line to the desired position. The line then has the inclination of the desired angle ϑ_2^* and leads to the target state r, see Figure 4.8.

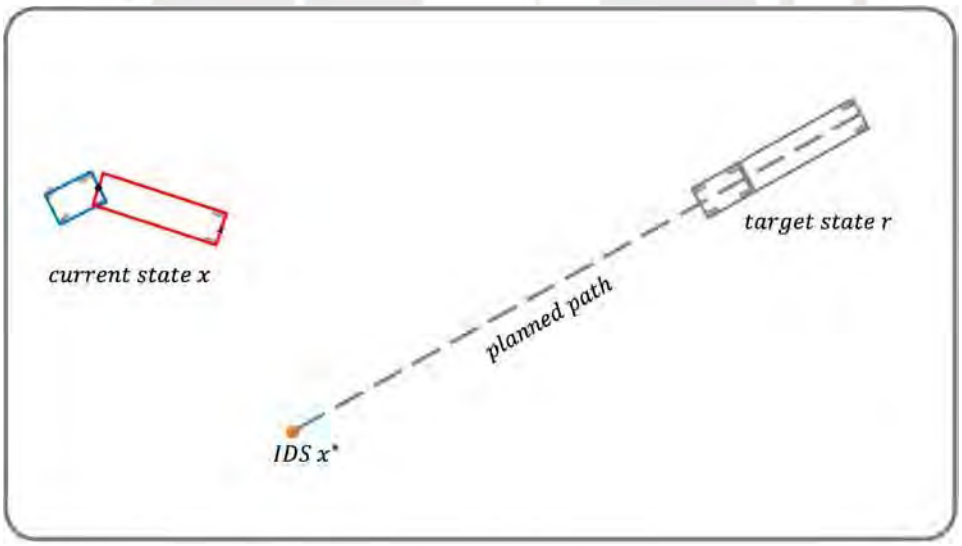


Figure 4.8: Pathplanning for target states

A general-shape trajectory f is given by the equation:

$$y = f(z) \,. \tag{4.21}$$

If the reference is given as a trajectory, a method to determine the desired state in every step is required. In the following such methods will be derived.

Line of sight method

For trajectories that satisfy the given conditions, the *line of sight method* (LOS), as presented in [24], can be used. Figure 4.9 illustrates the constellation, with $>$ being the robots current state and $>^*$ representing the robots *instantaneous desired state* (IDS) given by its coordinates (z^*, y^*), the orientation ϑ_2^* and the angle ϑ_{12}^* .

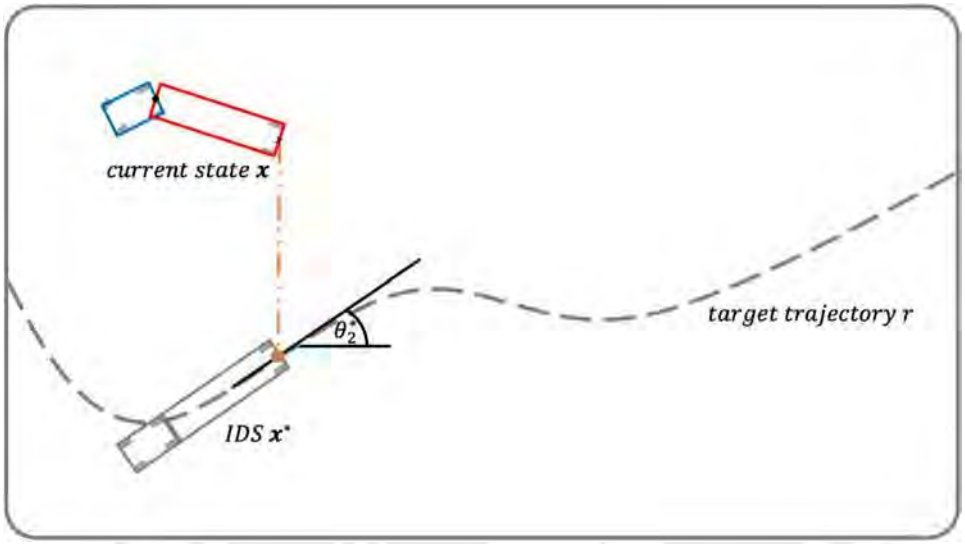


Figure 4.9: Trajectory following using the line of sight method

The conditions imposed on the trajectory $f(z)$ are:
1. It is well-defined for every x-coordinate that the mobile robot moves by.
2. It is differentiable up to the third derivative: $f'(z)$, $f''(z)$ and $f'''(z)$.

Algorithm:

Step 1: Set $z^* := z$

Step 2: Compute $y^* = f(z^*)$

Step 3: Compute the desired orientation as the inclination angle of f at (z^*, y^*):

$$\vartheta_2^* = \arctan(f'(z^*)) .$$

Step 4: Using the models equation from Section 3.3 and differentiating ϑ_2^*, one obtains:

$$\vartheta_{12}^* = \arctan(-L_2 f''(z) \cos^3(\vartheta_2)) .$$

Step 5: Differentiating again, the desired steering angle δ^* can be computed with respect to the models equations:

$$\delta^* = \arctan(L_1 L_2 f'''(z^*) \cos^4(\vartheta_2^*) \cos^3(\vartheta_{12}^*) + \frac{L_1}{L_2} \sin(\vartheta_{12}^*)(1 - 1.5 \sin^2(\vartheta_2^*) \cos(2\vartheta_{12}^*))) .$$

This is only an approximation, as the projection onto the x axis (as done by step 1) is not necessarily the closest point on the trajectory.

Extended Perpendicular Desired Position Method (EPDP)

The main drawbacks of the line of sight method are the approximation of the x value in step 1 and the resulting conditions on the target trajectory, especially that f has to be well-defined for every x. This is also relevant for trajectories that are jumping in the y coordinate. Furthermore, both conditions do not allow trajectories as shown in Figure 4.10. The approximation error is visualized in Figure 4.11.

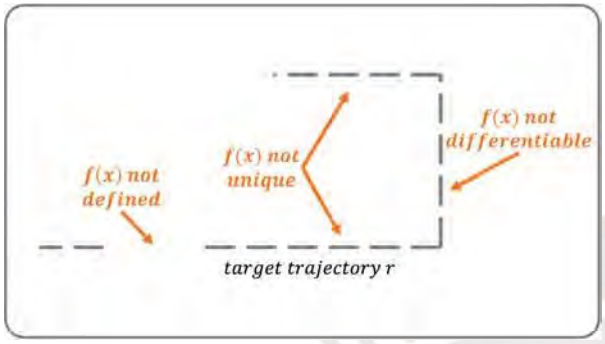


Figure 4.10: Complex trajectory

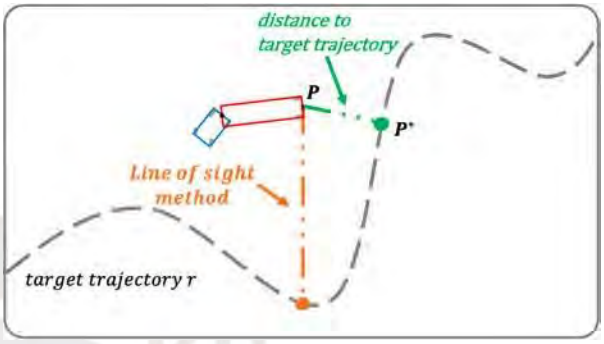


Figure 4.11: Approximation error

This is why a more generic method is aspired. As proposed in [24], the *perpendicular desired position method* can be used. This method is presented for linear functions and circles only, which results in restrictions for the target trajectory f [1]. Furthermore, as discussed by the author, the method is only applicable when the intersection point of the perpendicular line between point and trajectory is unique and its computation is reasonable.

To improve the computation of the desired state, the *extended perpendicular desired position method* (EPDP) is derived. First, the desired position $P^* := (z^*, y^*)$ is computed by a minimum distance optimization between the current position $P := (z, y)$ and the target trajectory f. This resolves not only the condition of a well-defined trajectory, but also enables the handling of general-shaped functions, instead of just certain types of functions. Furthermore, the steps to compute the desired angles, are provided in a more general manner.

Algorithm:

Step 1: The distance d between an arbitrary point (z, y) and a point on the trajectory $(z_t, y_t) = (z_t, f(z_t))$ can be computed by

$$d(z, y, z_t, y_t) := \sqrt{(z - z_t)^2 + (y - y_t)^2} = \sqrt{(z - z_t)^2 + (y - f(z_t)^2}\,.$$

To find the desired point P^* on f with minimum distance to P, a minimization over the trajectory can be conducted to obtain the minimum distance $d_{min} := \min_{z_t} d(z, y, z_t, f(z_t))$ and thereby the desired point

$$z^* := \arg\min_{z_t} \sqrt{(z - z_t)^2 + (y - f(z_t))^2}\,.$$

1 There are different notations for the target trajectory. f is related to the mathematical function and r to the target object.

Step 2: Compute $y^* := f(z^*)$.

Step 3: The last trailer, here the only trailer, should follow the angle of the tangent of the trajectory at (z^*, y^*). The tangential angle can be computed using the derivative of f:

$$\vartheta_2^* = \arctan(f'(z^*)) .$$

Step 4: Using the equation of the truck-trailer model, one can obtain ϑ_{12}^* by

$$\dot{\vartheta}_2^* = -\frac{v}{L_2}\sin(\vartheta_{12}^*) \Leftrightarrow \vartheta_{12}^* = \arcsin(-\frac{L_2}{v}\dot{\vartheta}_2^*) .$$

Step 5: Again with the models equation, the desired steering angle δ^* can be computed by:

$$\dot{\vartheta}_{12}^* = \frac{v}{L_2}\sin(\vartheta_{12}^*) - \frac{v}{L_1}\tan(\delta^*) \Leftrightarrow \delta^* = \arctan\left(\frac{(\frac{v}{L_2}\sin(\vartheta_{12}^*) - \dot{\vartheta}_{12}^*)L_2}{v}\right) .$$

Remarks

- As the steering angle δ is limited, one has to make sure that δ^* is limited, too.
- The values of $\dot{\vartheta}_2^*$ and $\dot{\vartheta}_{12}^*$ can be computed by step 3 and step 4, respectively.
- As $\sin(\cdot) : \mathbf{R} \to [-1, 1]$, appropriate means have to ensure that $(-\frac{L_2}{v}\dot{\vartheta}_2^*) \in [-1, 1]$.
- For computation, the trajectory might be given in a sampled form or as set of two-dimensional points describing the desired path:

$$f_s := \{(z_1, y_1), (z_2, y_2), \ldots, (z_N, y_N)\} .$$

- For efficient computation, step 1 can be simplified by
$$z^* := \min_{z_i \in f_s} d(z, y, z_i, f(z_i)).$$
- The derivatives can be computed by numerical approximation (forward and backward differentiation) with a small $h \in \mathbf{R}$:

$$f'(z) \approx \frac{f(z+h) - f(z)}{h} \approx \frac{f(z) - f(z-h)}{h} .$$

- The method can be generalized for N-bodies as well. This can be done in a straightforward way by inserting further steps to compute $\vartheta_{23}, \ldots$. It should be noted that after the calculation of ϑ_N, the next value to be computed is the angle between trailer N-1 and N-2: $\vartheta_{(N-2)(N-1)}$. Afterwards, all the preceding angles can be computed, as the model is in a chained form. However, the algorithm will not be presented here, but the 2-trailer case can be found in the implementation files.

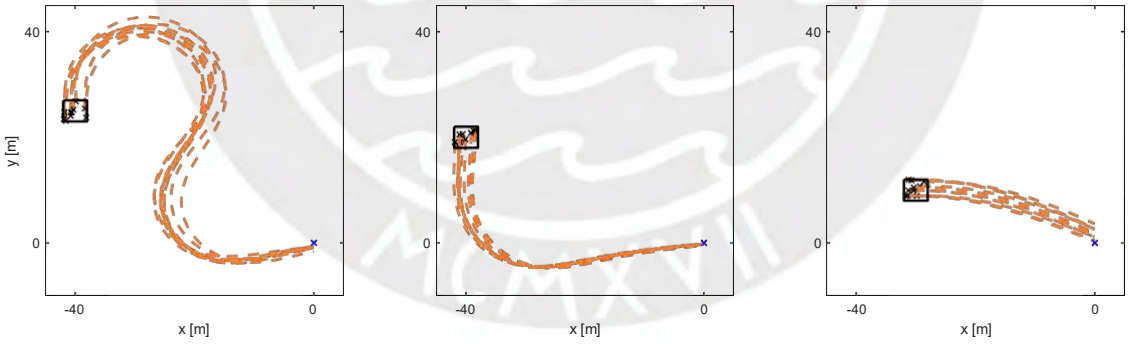
4.5 Stability Analysis

It has been shown in previous works that there are stable controllers for the truck-trailer system [31]. For the practical application, stability[1] alone is not sufficient. It is rather important for which environment around the target the controller converges. In addition, the convergence should be as fast as possible, while demonstrating a reasonable behavior[2]. To analyze the stability in a practical way, a structured approach will be used.

First, the behavior for a single target state will be analyzed. Therefore, the convergence of the system with initial states close to the target state will be investigated. Next, to generalize the findings from this investigations, a study with many initial conditions will be conducted, showing an empirical approximation of the area of stability. Third, it will be shown how the area of stability can be extended, such that it is sufficient to design controllers for a small area around the origin. Finally, research on the behavior for pathfollowing will be done. It will be shown how the system behaves for different types of paths, such as linear or nonlinear functions.

Convergence

Figure 4.12 shows the convergence to the origin from various initial conditions for the LQR controller, the RL agent[3] and the nonlinear MPC. The initial conditions were chosen randomly from a quadratic box with side length 4 around the start position. The convergence behavior shows that the controllers are able to reach the origin. Note that the y-axis with $z = 0$ is always reached, but the exact target position is missed from time to time with certain initial conditions. This is because no switching was used in this case.



(a) LQR from $(-40, 25, \frac{\pi}{2}, 0)$ (b) RL from $(-40, 20, -\frac{\pi}{2}, 0)$ (c) nMPC from $(-30, 10, 0, 0)$

Figure 4.12: Visualization of the convergence behavior

1 Stability in the sense, that the robot eventually reaches the target.
2 This is especially important for autonomous driving applications, as there will be other vehicles and humans, that have to trust the autonomous controller.
3 Refer to Chapter 5.3.1.

Area of Stability

To visualize the area of stability, an experiment with 1000 runs was conducted with the LQR controller. To always reach the desired target position, switching conditions, as introduced in Section 4.3, are used. The performance of the entire system in different scenarios is analyzed in Chapter 6. The results can be seen in Figure 4.13. The red points represent an initial position that did not lead to a successful run, while green points indicate a successful run.

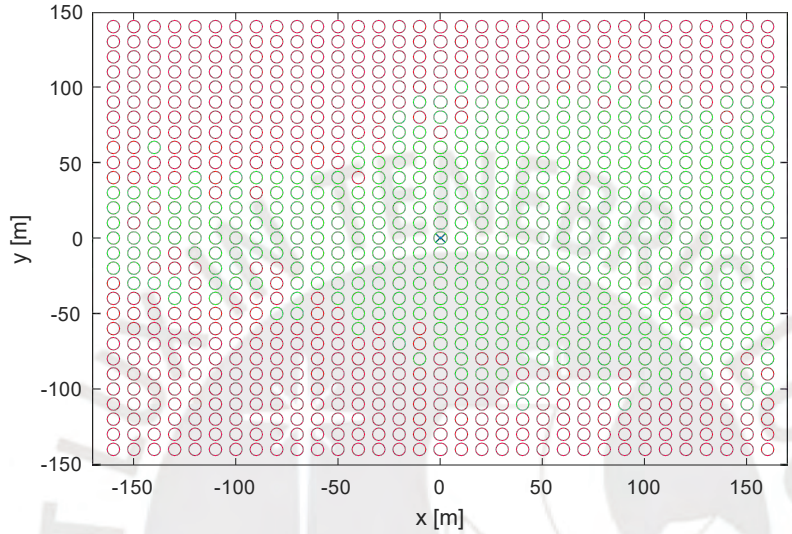


Figure 4.13: Area of stability by sampling initial points with $\vartheta_{12} \in \left(-\frac{\pi}{4}, \frac{\pi}{4}\right)$ and $\vartheta_2 \in (-\pi, \pi)$

In this setting, the stability of the controller depends primarily on the distance to the target. In detail, if the initial point has a y-coordinate larger than some value or from another point of view the initial position has a large y-distance to the target, the system becomes unstable. Performance for positive initial x-values is better than for negative ones.

How to extend the area of stability

The area of stability can be extended for any controller that is stable in an area with $|y| \leq d_{stab}$, $d_{stab} \in \mathbf{R}^+$ around the origin. By transforming the y-position for the controller by

$$y = \begin{cases} y & \text{, if } y < d_{stab} \\ d_{stab} & \text{, if } y > d_{stab} \end{cases}$$

with e.g. $d_{stab} = 40$. This way, initial points farther away from the origin are stable as well.

This is the reason why the stability need not be investigated further, and it is sufficient to design a controller that is stable in an area with $|y| < d_{stab}$ around the origin. Note that this is only valid for object-free environments without noise.

Pathfollowing Behavior

Given a controller is stable around the origin, the same controller can be used to follow a given trajectory by applying the approach described in Section 4.4. Figure 4.14 shows the behavior for several trajectory-types, initial conditions and controllers. The initial positions are sampled from a square box around a given start position. It can be observed that both, the LQR and RL controller, are able to follow the linear, nonlinear and non-differentiable trajectory in the proposed system.

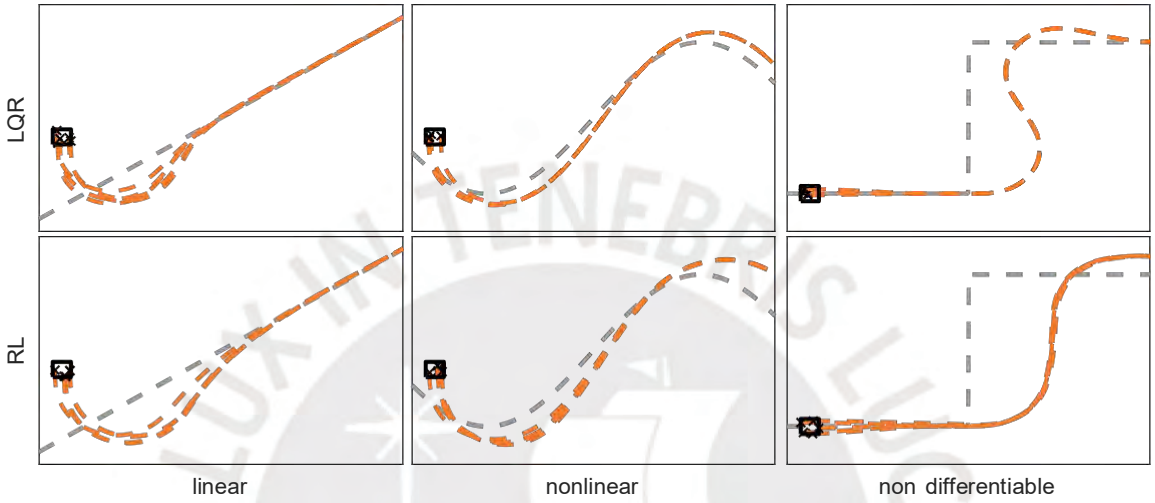


Figure 4.14: Convergence behavior for pathfollowing

A sufficient pathfollowing behavior leads to another way extending the area of stability. This is to plan a trajectory, e.g. a line, from the robots current position to the target position or close to it to ensure convergence by first following the line and afterwards converge to the target, when the robot is close to it.

ET LUX IN TENEBRIS LUCET
MCMXVII

5 Control with Artificial Intelligence

*For the control of autonomous multibody vehicles using **artificial intelligence**, neural networks can be used to compute the control signal. Several network types and training approaches are known to handle this task.*

This chapter deals with the application of artificial intelligence, especially machine learning, for the control of dynamical systems. More precisely, the controller from Chapter 4 that computes the control variable will be implemented using neural networks.

Therefore, the overall structure changes, as there will be a *training phase* and a *production phase*, using parameters from memory determined in training. During training, the structure will look as shown in Figure 5.1.

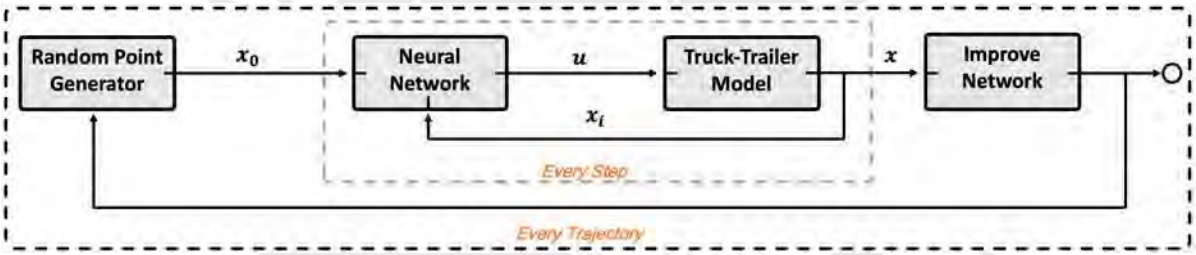


Figure 5.1: Structure in the training phase of the neuro-controller

5.1 Neural Networks

Artificial Neural Networks (ANN) consist of structured connections between artificial neurons. Those are technical simplifications of biological neurons from the human brain. The networks can build structures, such as feedforward or recurrent networks. They are *trained* to improve their performance. Therefore, the weights between neurons in different layers are adapted. The network should generalize after the training, so the mode can work even for unseen situations. This can be done using labeled training data or by interacting with the environment and providing a reward, just to name two examples.

In general, a neural network produces an input-output mapping. The complexity of the mapping depends on the activation functions, the number of layers and neurons in each layer as well as the extent and quality of training. A neural network is called a *deep neural network* if it has more than one hidden layer. The network is denoted as

$$u = \Omega(>, \xi) \tag{5.1}$$

with u being the output of Ω, ξ representing the parameters, such as weights and structure, and $>$ is the input to the network. Let in the following $> \in \mathbf{R}^3$ and $u \in \mathbf{R}$.

5.1.1 Structure

So-called *Multilayer Perceptrons* (MLP) have an input layer, hidden layers and an output layer. The number of neurons in the input and output layer depends on the dimensionality of the input data $x^{(i)}$ and output data $u^{(i)}$, respectively. The required number of hidden layers and the number of neurons in each layer depends on the complexity of the underlaying function that should be approximated. Those are parameters that can be tuned to improve performance of the network. The connection from neuron i in layer l to neuron j in layer $l+1$ has weight $w_{ij}^{[l]} \in \mathbf{R}$.

Any function can be approximated as precisely as desired with one hidden layer of sufficient size [66]. This requires a sufficient amount of training data. In certain situations it can be more efficient to use more than one hidden layer, as the number of neurons per layer can be reduced and the training is more effective in terms of the number of training examples. Let $H = [H_1, H_2, \ldots]$, $H_l \in \mathbf{N}^+$ be the number of neurons in the respective hidden layer l. The overall structure of a neural network is shown in Figure 5.2.

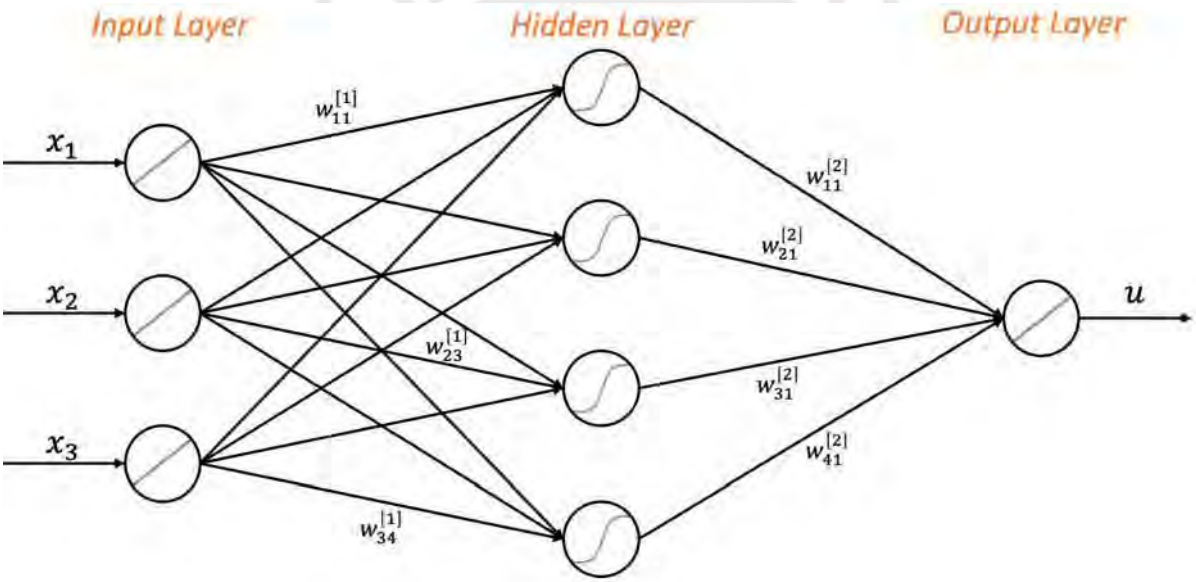


Figure 5.2: Visualization of a MLP with one hidden layer

To improve the understanding of the structure of neural networks, and to simply display larger networks, only one neuron per layer will be shown. Therefore, the structure shown in Figure 5.3 is equivalent to Figure 5.2. Note that the number above the neuron indicates the number of neurons in the respective layer. In addition, the weights between layer l and $l+1$ will be noted as $w^{[l]} \in \mathbf{R}^{H_l \cdot H_{l+1}}$, $l \in \mathbf{N}^+$ to simplify handling of more than one hidden layer.

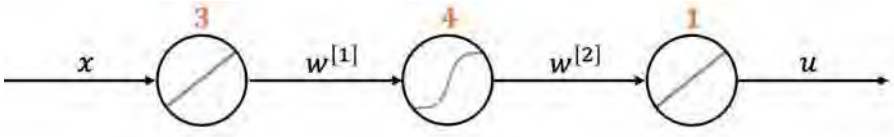
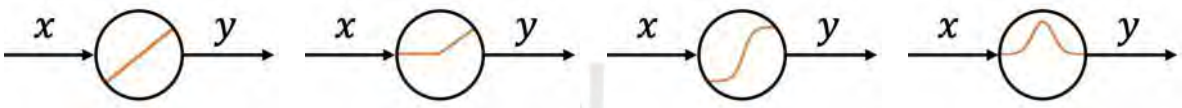


Figure 5.3: Simplified visualization of a MLP with one hidden layer

A neuron computes the dot product of the incoming values. For neuron i in layer l, the activation $m_i^{[l]}$ is given by

$$m_i^{[l]} := \sum_{j=1}^{\#} w_{ij}^{[l-1]} \cdot m_j^{[l-1]} . \tag{5.2}$$

An activation function f is applied afterwards: $n_i^{[l]} = f(m_i^{[l]})$. Common activation functions are the linear function, the rectified linear unit (ReLU), the gaussian function and the tanh function (sigmoid). Simplified neurons are visualized in Figure 5.4. Most of the time, the activation function of hidden layers is chosen to be relu or tanh functions while the output layer has a linear or tanh function.



(a) Linear activation: $f(z) = z$

(b) ReLU activation: $f(z) = max(0, z)$

(c) Gaussian activation: $f(z) = e^{-z^2}$

(d) Tanh activation: $f(z) = \frac{2}{1+e^{-2z}} - 1$

Figure 5.4: Visualizations neurons with different activation functions

5.1.2 Training

After fixing a structure of a neural network, the parameters are initialized randomly. At this point, the network is able to make predictions with its parameters, but as they are initialized randomly, the result is expected to be random as well. Using an optimization approach, such as gradient descent, the network can adapt its parameters to improve its performance. The optimization is done by presenting data to the network.[1] By carefully choosing the data and validating the trained network, it can then generalize to new data. After the training, outputs can be computed for unknown data points [55].

Data

Most classical approaches use so-called *supervised learning*. This is, using labeled data to train the network to minimize the error on a given dataset. For supervised learning not only the input data with $N \in \mathbb{N}$ samples $x^{(input)} = [x^{(0)}, x^{(1)}, ..., x^{(N)}]$, but also the respective output data $u^{(i)}$, $i = 0, 1, ..., N$ are known for training.

This is why in general a sufficiently large dataset is required. The data can be generated artificially, e.g. by a mathematical model. However, this has the disadvantage of patterns not matching with reality. Real data more accurately represents the reality, including noise, e.g. by a sensor. The quality of the datasets has a huge influence on the performance.

In case of *reinforcement learning*, data is obtained by the interaction with the environment by observing rewards, as well as trial and error, see Section 5.3.

1 The data should be scaled (e.g. to $[0, 1]$) and shuffled.

Algorithm

The training can be done by the *Stochastic Gradient Descent* (SGD) algorithm. The algorithm consists out of three steps to adapt the weights. Without loss of generality, let w_{ij} represent a general weight in any layer. Let u_ρ^* be the teacher, this is the label for the input x_ρ, $\rho \in \{1, 2, \ldots, N\}$.

Step 1: Select a random sample ρ and compute the error between the output of the network $u_\rho = \Omega(x_\rho, \xi)$ and the respective label u_ρ^*

$$e_\rho := u_\rho - u_\rho^* . \tag{5.3}$$

Step 2: Compute the loss J_ρ and its derivative $\frac{\partial J_\rho}{\partial w_{ij}}$ using error e_ρ.

Step 3: The update for the weight w_{ij} with learnrate $\eta \in \mathbf{R}$ is then given by

$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial J_\rho}{\partial w_{ij}} . \tag{5.4}$$

To complete one epoch, steps 1–3 are repeated for all data points. To obtain a well-trained network, usually several epochs are required.

Evaluation

As stated before, some measure for the performance of the network, such as a cost function J or a reward function R is used for training[1]. When learning from data, the cost function is based on the labels (*teachers*) or the error between labels and output of the neural network.

A widely used cost function is the *Mean Squared Error* (MSE). It computes the sum of the square of the distance of the network output of all relevant points $u_k = \Omega(x_k, \xi)$ to its teachers u_k^*, for $k = 1, \ldots, N$:[2]

The derivative $\frac{\partial J_{MSE}}{\partial w_{ij}}$ for the gradient descent algorithm can be obtained straightforward by chainrule [55]

$$\frac{\partial J_{MSE}}{\partial w_{ij}} = \sum_{k=1}^{N} (u_k - u_k^*)^T \frac{\partial (u_k - u_k^*)}{\partial w_{ij}} = \sum_{k=1}^{N} (u_k - u_k^*)^T \frac{\partial u_k}{\partial w_{ij}} , \tag{5.5}$$

while $\frac{\partial u_k}{\partial w_{ij}}$ depends on the position of w_{ij} in the network.

1 Will be covered later.

2 Note that $N = 1$ for stochastic gradient descent, while $N > 1$ for (mini) batch gradient descent.

5.2 Neuro-Controller

The design of a neuro-controller starts with the design of a neural network $u_k = \Omega(>_k, \xi)$ to determine the action u_k based on the state $>_k$ at step k. Goal of the controller is to track a desired setpoint $>^*$. Figure 5.5 shows the block diagram for the closed loop system with a neuro-controller.

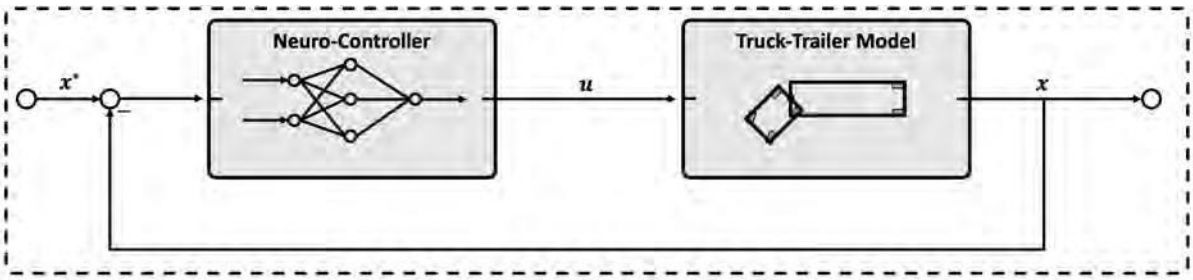


Figure 5.5: Structure of the neuro-controller

The inputs of the neural network can be the state $>$ of the truck-trailer system with training points

$$>^{(input)} = \begin{bmatrix} y^{(0)} & y^{(1)} & \dots & y^{(N)} \\ \vartheta^{(0)} & \vartheta^{(1)} & \dots & \vartheta^{(N)} \\ \vartheta_2^{(0)} & \vartheta_2^{(1)} & \dots & \vartheta_2^{(N)} \end{bmatrix} . \tag{5.6}$$

If supervised learning is applied, the labels are the respective control variables, e.g. determined by another control algorithm

$$u^{(input)} = \{u^{(0)}, u^{(1)}, \dots, u^{(N)}\} . \tag{5.7}$$

The steps from network design to the deployment of the model are visualized in Figure 5.6. Note that the network architecture can change during the training and validation process.

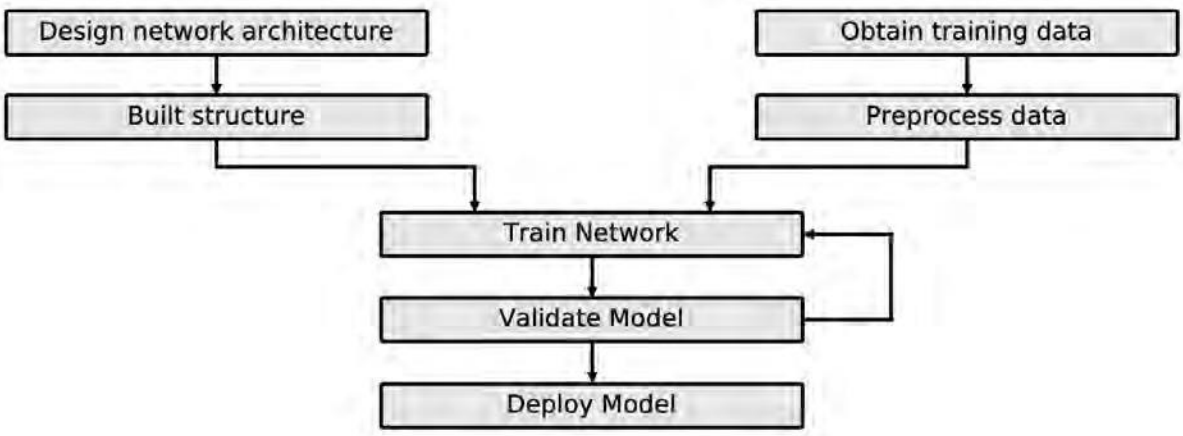


Figure 5.6: Visualization of the development process for neural networks

Training Approach

As the setup is fixed, it has to be determined how to best train the network and which network architecture is the most suitable. As part of the training process, the tuning of hyperparameters, such as the learning rate η and the number of neurons, is crucial. In the following, four ways to train a deep neural network to control a dynamical system, such as the truck-trailer, are presented:

1.) A neural network can imitate a controller, such as designed in Chapter 4. The advantages are the generalization and possibly the faster computation.

Therefore, a dataset consisting of 400 trajectories with 250 points each, so 100000 points, was generated with the respective control inputs, computed by a linear quadratic optimal controller. A network with one hidden layer and tanh activation function as well as linear output neurons was then trained with the Levenberg-Marquardt algorithm and validated. By trial and error, the number of neurons $H_1 = 50$ was found. The result is not improved by a larger number of neurons, so the additional effort for training and the unnecessary degrees of freedom are not needed.

After 265 iterations, the training was completed with a MSE of 0.000280. Figure 5.7 shows the training progress.

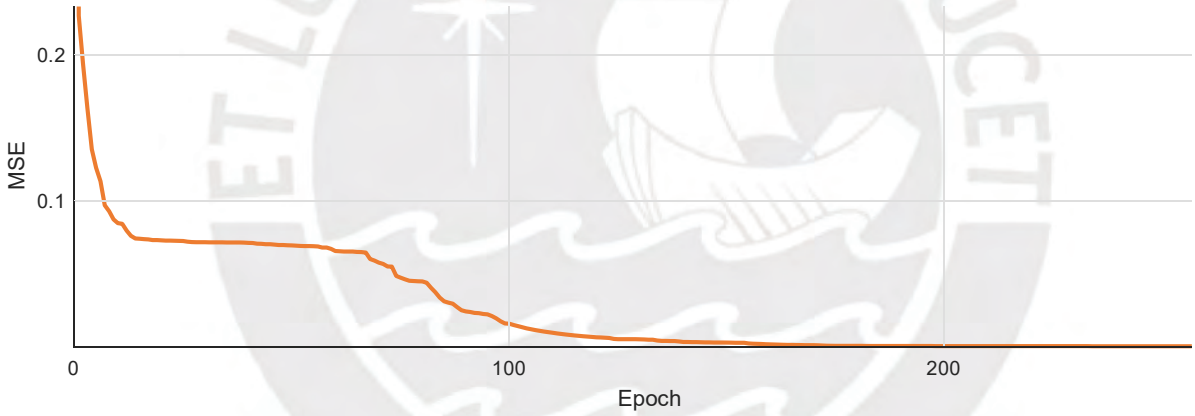


Figure 5.7: Learning curve (MSE per epoch) of the neuro-controller

However, this neuro-controller requires the design of another controller to generate the training data. As there are only few advantages, another approach is aspired. Note that, the performance of this neuro-controller can be improved by generating a more diverse set of data, e.g. with different controllers for different areas, such as a fuzzy controller for points far away and MPC for points close to the target.

A neuro-controller can be used to find a smaller representation of a complex controller, such as a MPC. In this case, the neural network simplifies the computation by approximating the real controller (there are also cases with the neuro-controller being equivalent to another controller) and is faster.

2.) Starting with a stable neuro-controller, such as derived in the previous section, the training can be continued with the *Dynamic Back Propagation* (DBP) algorithm. This method uses the model of the system to further improve the dynamic capabilities. This improves the performance even further.

In contrast to the first approach, the cost function used for the DBP algorithm does not use known-to-be-good control signals, but rather utilizes the mathematical model of the system and the desired output of the system. This way, the cost function

$$J_{DBP} := \frac{1}{2} \sum_{k=1}^{N} (>_k - >_k^*)^2 \tag{5.8}$$

makes direct use of the desired state $>_k^*$.

Attention has to be paid to maintain stability of the closed-loop system, as the algorithm is likely to produce jack-knife states otherwise. Therefore, a very small learning rate, such as $\eta \leq 0.0000001$, should be used. This leads to a convergence inside the current minimum of the network.

3.) Using two deep neural networks, forming an agent that can interact with the environment, a neuro-controller can be trained using *Reinforcement Learning* (RL), more specific the DDPG algorithm, see Section 5.3.

There is another approach to use two neural networks as a controller: One network represents the controller while another one learns the behavior of the plant and therefore replaces the model. The model network is then trained by the behavior of the real plant based on inputs of the controller network. The controller network is then updated using the derivative of the model network.

4.) Finally, a genetic algorithm can be applied to evolve the network parameters. This will not be investigated further in this work, but more information can be found in [60].

Other ideas to approach this type of problem are (a) using a neural network with two or more outputs, so it can also determine the velocity or the direction or (b) using a modified cost function, e.g. with a term for the deviation to another controller, see Chapter 7.

5.3 Reinforcement Learning

In the following, a so-called *agent* is trained. An agent is a neural network that is capable of determining an action u. While training, the agent starts with random actions and learns by trial and error. The training can be done while interacting with the environment, so without explicit model knowledge.[1] Note that for simulation, model knowledge is necessary.

The goal in reinforcement learning is to maximize the cumulative reward function [67]

$$R_k := r_k + \gamma\, r_{k+1} + \gamma^2\, r_{t+2} + \ldots = \sum_{i=0}^{\infty} \gamma^i\, r_{k+i} \tag{5.9}$$

with $\gamma \in [0, 1]$ a discount factor for expected future rewards. The control actions u_k are chosen such that

$$u_k^* = \arg\max_{u_k} Q(>_k, u_k) \tag{5.10}$$

with $>_k$ and u_k being the state of the system and the control action at step k, respectively. To obtain the optimal control actions, they are taken from the control policy μ:

$$u_k := \mu(>_k)\,. \tag{5.11}$$

A control policy computes an action u_k based on the current state $>_k$ and therefore acts like a controller. The action-value function $Q(>_k, u_k)$ describes the expected reward after taking an action u_k in state $>_k$ and thereafter following policy μ. It is defined as:

$$Q(>_k, u_k) := \mathbf{E}[R_k | >_k, u_k] \tag{5.12}$$

with all u_i, $i > k$ sampled from μ. To maximize the cumulative reward from equation (5.9), the optimal control policy μ^* has to be found.

The optimal action-value can be obtained by finding the policy that maximizes the expected value of the future reward:

$$Q^*(>_k, u_k) = maz_\mu \mathbf{E}[R_k | >_k, u_k]\,. \tag{5.13}$$

This can be solved with dynamic programming and the *Bellman* equation:

$$Q^*(>_k, u_k) = \mathbf{E}[r(>_k, u_k) + \gamma\, Q^*(>_{k+1}, u_{k+1})]\,. \tag{5.14}$$

For discrete states and actions, a Q-table can be used to store the action-values. For continuous states and actions, a neural network can be used to approximate the action-value functions, providing Q-values for any combination of state and action, see Chapter 5.3.1.

1 The training can be improved by artificial noise or closed-loop training with the real sensors.

5.3.1 Deep Deterministic Policy Gradient (DDPG)

In the following, the *Deep Deterministic Policy Gradient (DDPG)* algorithm will be used for the training of the neuro-controller, as proposed in [62]. The DDPG approach implements an actor and a critic neural network and therefore works for continuous states and continuous actions [61]. The overall architecture is illustrated in Figure 5.8.

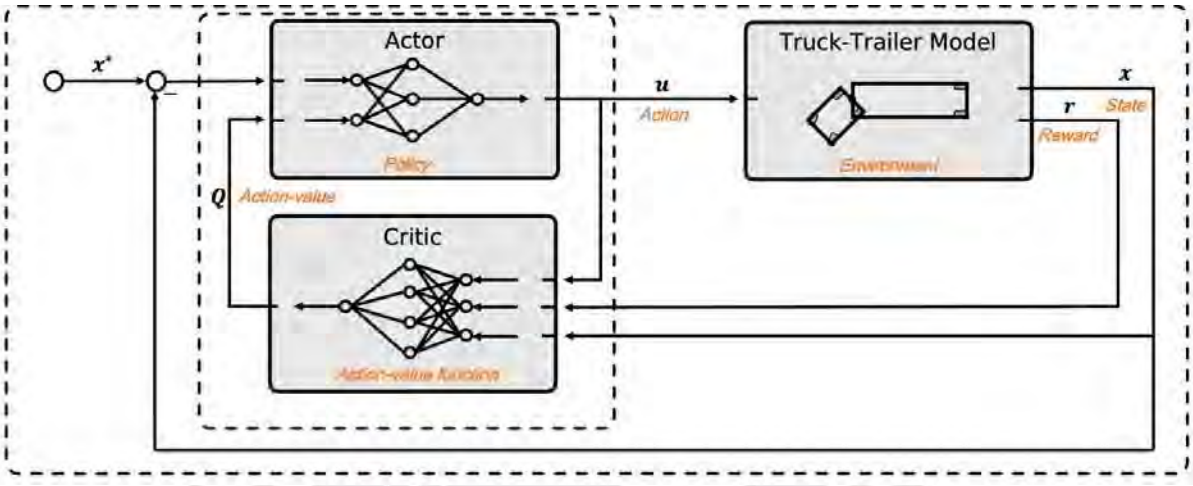


Figure 5.8: Architecture of the DDPG concept

Note that the action-value Q and the reward r are not direct inputs to the actor and critic, network respectively, but are used to adapt the parameter.

Within this framework, one deep neural network is used to compute the optimal control actions u_k^* and another network approximates the action-value function $Q^*(x_k, u_k)$. These networks are called actor and critic networks, respectively. The actor network[1] is represented by $\mu(x_k, \xi^\mu)$ and the critic network by $Q(x_k, u_k, \xi^Q)$ where ξ^μ and ξ^Q are the parameters of the actor and critic networks, respectively. The critic network is trained in a supervised learning manner with gradient descent to predict the optimal action-value function given by equation (5.14). Therefore, the cost function that this network minimizes is:

$$J(\xi^Q) := \mathbf{E}[(Q^*(x_k, u_k) - Q(x_k, u_k, \xi^Q))^2] \tag{5.15}$$

where $Q^*(x_k, u_k)$ is computed with Equation (5.14). The gradient of this cost function is given by:

$$\frac{\partial J(\xi^Q)}{\partial \xi^Q} = \mathbf{E}\left[(Q^*(x_k, u_k) - Q(x_k, u_k, \xi^Q))\frac{\partial Q}{\partial \xi^Q}\right] . \tag{5.16}$$

At the same time, the optimal control actions u_k^*, given by Equation (5.10), are computed with gradient ascent, where the gradient of the actor network is given by:

$$\frac{\partial Q(x_k, u_k, \xi^\mu)}{\partial \xi^\mu} = \mathbf{E}\left[\frac{\partial Q(x_k, u_k, \xi^\mu)}{\partial u_k}\frac{\partial u_k}{\partial \xi^\mu}\right] . \tag{5.17}$$

1 Note that the notation for policy and neural network is both μ, as the policy is approximated by a neural network.

Equation (5.16) shows that $\frac{\partial \mu(\xi^Q)}{\partial \xi^Q}$ requires the computation of $Q^*(>_k, u_k)$, which depends on $Q^*(>_{k+1}, u_{k+1})$. As proposed in [61], a target critic network Q' is used to compute $Q^*(>_{k+1}, u_k)$ and u_{k+1} is computed with a target actor network μ'. The target actor and critic networks are two additional neural networks with parameters $\xi^{\mu'}$ and $\xi^{Q'}$, respectively. The parameters are updated according to:

$$\begin{aligned} \xi^{\mu'} &\leftarrow \tau\xi^{\mu} + (1 - \tau)\xi^{\mu'} \\ \xi^{Q'} &\leftarrow \tau\xi^{Q} + (1 - \tau)\xi^{Q'} \end{aligned} \tag{5.18}$$

with $0 < \tau << 1$. Note that target networks have the same structure as non-target networks.

The algorithm includes a replay memory to estimate the expectations of Equations (5.16) and (5.17). In addition, the algorithm also uses noise to explore new states. As proposed in [61], an Ornstein-Uhlenbeck process is used. Figure 5.9 shows the executed action during one episode of training. The influence of the noise can be seen as the random spikes.

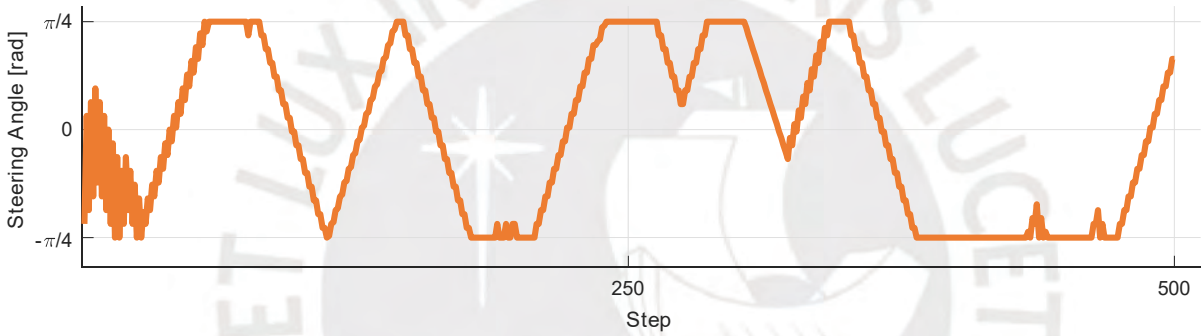


Figure 5.9: Noisy steering angle in the first episode

Reward

To successfully apply DDPG to the given problem, a suitable reward function has to be designed. A high reward given for a certain situation should represent the desired behavior. This is, that the agent approaches the target position as fast as possible, but at the same time pay attention to the control effort. Let the control effort in every step c_k be proportional to the difference between current and preceding control input: $c_k := \beta\,|\delta_k - \delta_{k-1}|$.

The input to this algorithm are the states $>$ and the respective reward for each state:

$$r_k(>_k, u_k) = \begin{cases} \kappa & \text{, if } ||>_k - >^*||_2^2 \leq \varepsilon \\ -||>_k - >^*||_2^2 - c_k & \text{, otherwise} \end{cases} \tag{5.19}$$

with β, κ and $\varepsilon \in \mathbf{R}$ design parameters. Note that κ has to be large enough to dominate the total cumulative reward, if the target position is reached.

5.3.2 Training

In every step, the agent gets its action from the actor. During training, noise is added to the action. The action is executed in the training environment and the respective reward is observed. Using the replay memory and the formulas from Section 5.3.1, the actor and target networks are updated.

An episode terminates if the maximum number of steps k_{maz} is reached or the agent reaches the target:

$$isDone = k \geq k_{maz} \text{ OR } ||>_k - >^*||_2^2 < \varepsilon\,. \tag{5.20}$$

To begin a new episode, the agent will be reset to a random initial position. The start position of episode i can be obtained by

$$>_i^0 = \begin{bmatrix} \tilde{z}_1 \\ \tilde{z}_2 \\ \tilde{z}_3 \\ \tilde{z}_4 \end{bmatrix} \sim \begin{bmatrix} U(z_{min}, z_{maz}) \\ U(y_{min}, y_{maz}) \\ U(\vartheta_{2,min}, \vartheta_{2,maz}) \\ U(\vartheta_{12,min}, \vartheta_{12,maz}) \end{bmatrix} = \begin{bmatrix} U(-25, 25) \\ U(-25, 25) \\ U(-\pi, \pi) \\ U(-1, 1) \end{bmatrix} \tag{5.21}$$

with $z \sim U(a, b)$ being a uniformly distributed variable between a and b. Note that initial positions that satisfy the condition in Equation (5.20) will be discarded. To ensure that the jack-knife state is avoided, the approach described in Section 4.1 is used before applying the action.

The training is done using the **nonlinear model** of the system to obtain the best possible result, even for states that do not match the linearized version.

Actor

The actor is a network that computes an action u_k from a state $>_k$. During training, a mini batch of size 256 is used to approximate the gradient for the network update with the learnrate $\eta^\mu = 0.0001$. Figure 5.10 shows the architecture with $H_i^\mu = 100$ neurons in the hidden layers.

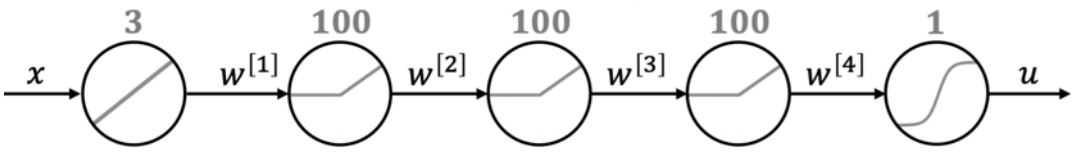


Figure 5.10: Architecture of the actor network

This network, after being trained, is the controller that will determine the input for the steering.

Critic

The critic is a network that computes the action-value $Q(>_k, u_k)$ from a state $>_k$ and a respective action u_k. During training the learnrate $\eta^Q = 0.001$ is used. Figure 5.11 shows the architecture with $H^Q = 100$ neurons in the hidden layers.

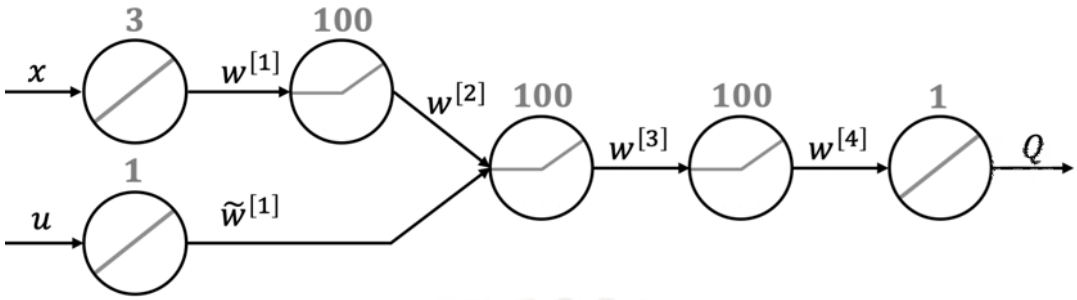
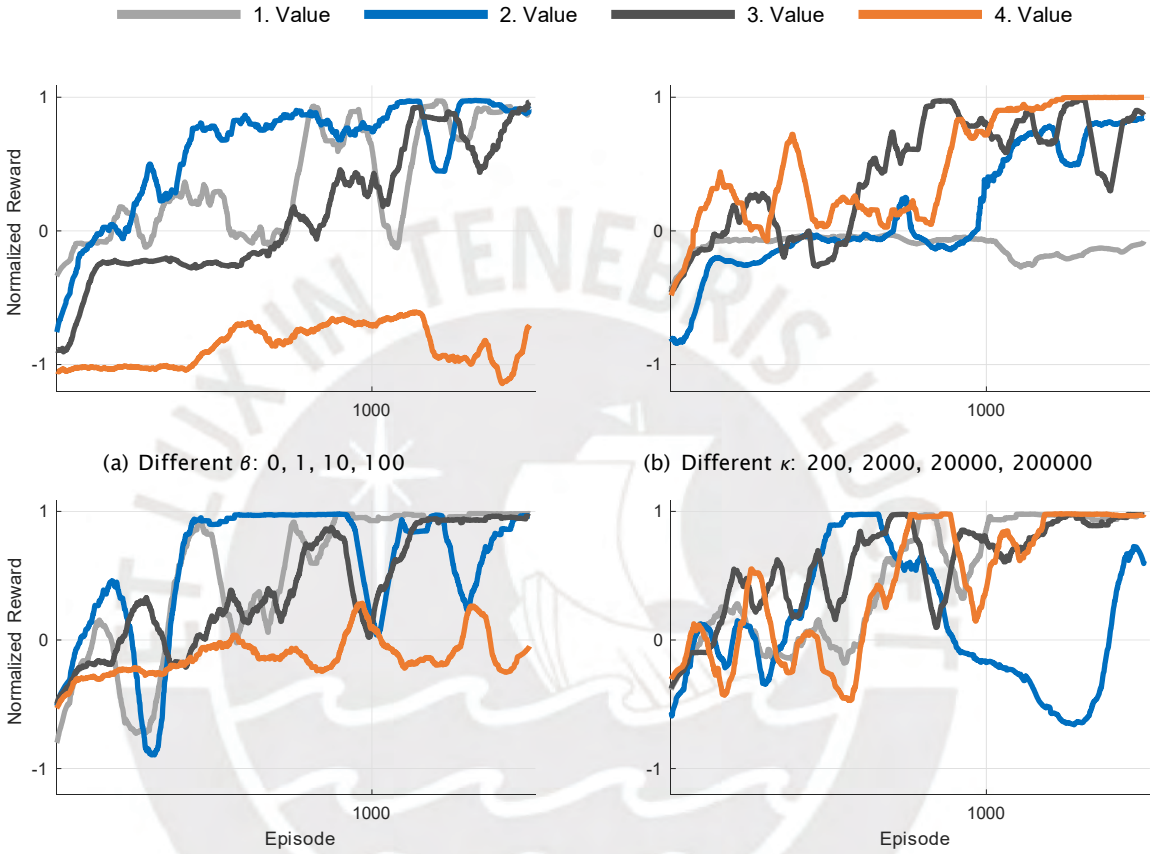


Figure 5.11: Architecture of the critic network

The critic network and the reward function are only used for training and have no further use. Afterwards, only the trained actor network is used. The critic is only trained to obtain an approximation for the Q values which are used to train the actor. The critic is trained based on the rewards from the reward function.

5.4 Results

In the following, results will be shown for the training of several DDPG agents. Therefore, the influence of the parameters β, κ and σ, as well as the structure of the actor will be investigated. To find the best parameters, one parameter at a time will be changed and the resulting performance will be evaluated by training the DDPG agent for 1500 steps.



(a) Different β: 0, 1, 10, 100

(b) Different κ: 200, 2000, 20000, 200000

(c) Different σ: 0.1, 0.3, 0.5, 0.7

(d) Different structures

Figure 5.12: Performance curves (cumulative reward after episode termination, either because of success or after 500 steps) for a selection of the parameters β, κ, σ and the structure of the actor network using different parameter values
First value (grey), second value (blue), third value (dark grey) and fourth value (orange).

Note that the performance of the parameters β and κ can not be evaluated using the cumulative reward directly, as the parameters have an influence on the reward. This is why the performance values are normalized, such that the theoretical maximum is 1. Note that the theoretical minimum does not exist, therefore -20000 was chosen for appropriate scaling. A positive value indicates a successful training. The curves are smoothed over 80 points. Only one example training process is shown. Training took~10 hours per configuration.

Punishment of Control Effort

Table 5.1 shows the influence of the parameter β (factor for the punishment related to the control effort). For $\beta = 0$, the reward is determined based on the state only. Learning steps are the number of episodes to reach the highest reward for the first time.

No.	β	Avg. reward	Target reached	# Learning steps
1	0	7340	✓	**1208**
2	1	**12111**	✓	1329
3	10	2487	✓	1496
4	100	-17350	C	1139

Table 5.1: Performance for different β with $\kappa = 20000$, $\sigma = 0.3$ and baseline structure

The respective performance curves are shown in Figure 5.12a. It can be seen that for $\beta = 100$ the training is successful. As the performance of the remaining configurations is in the same order of magnitude, $\beta = 0$ will be used for simplicity.

Reward for Success

Table 5.2 shows the influence of the parameter κ (reward for reaching the target). For $\kappa = 200$, the total reward will likely turn out to be negative, while $\kappa = 200000$ will most likely lead to a positive total reward if the target is reached.

No.	κ	Avg. reward	Target reached	# Learning steps
1	200	-2162	C	**699**
2	2000	-1750	✓	1496
3	20000	8369	✓	1290
4	200000	**97356**	✓	1499

Table 5.2: Performance for different κ with $\beta = 0$, $\sigma = 0.3$ and baseline structure

The respective performance curves are shown in Figure 5.12b. It can be seen that $\kappa = 200$ does not lead to successful training, as the reward is too small to induce desired behavior, but even for $\kappa = 2000$ the average reward is negative. To obtain a fast but stable convergence, $\kappa = 20000$ is chosen. This also fits the order of magnitude of the negative reward obtained by an untrained agent.

Influence of Noise

Table 5.3 shows the influence of the parameter σ (variance of the noise process). For $\sigma = 0$, the process would not explore at all, while a large σ (close to 1) leads to a lot of exploration while not exploiting much.[1]

No.	σ	Avg. reward	Target reached	# Learning steps
1	0.1	9998	✓	1345
2	0.3	**11479**	✓	**896**
3	0.5	7910	✓	1489
4	0.7	-2558	C	970

Table 5.3: Performance for different variances σ, $\kappa = 20000$, $\beta = 0$ and baseline structure

The respective performance curves are shown in Figure 5.12c. It can be seen that $\sigma = 0.7$ does not lead to enough exploration while $\sigma = 0.5$ converges slower compared to the smaller values. To boost exploration (as the system should get to know the setting) $\sigma = 0.3$ will be chosen.

Structural Changes

Table 5.4 shows the influence of the structure of the actor network. For the first run, the structure in unchanged to obtain a baseline.

No.	Change of actor	Avg. reward	Target reached	# Learning steps
1	Baseline (no changes)	9558	✓	1114
2	Use tanh activation	2423	C	**658**
3	Use $H = 50$	**11672**	✓	736
4	Use 2 hidden layers	8448	✓	1262

Table 5.4: Performance for different structural changes of the actor network with $\kappa = 20000$, $\beta = 0$ and $\sigma = 0.3$

The respective performance curves are shown in Figure 5.12d. It can be seen that the tanh activation function does not perform well, while the other modifications do not improve the network significantly. This is why the structure will remain unchanged.

1 This is called the *Exploration-Exploitation Dilemma*.

Final results

The final training process for 10000 episodes is shown in Figure 5.3.1.[1] Note that in the first 500 episodes the agent slowly learns and thereby improves its performance. To make sure the agent performs well from a large variety of initial positions, the agent is trained for many more episodes. All used parameters can be found in Table 5.5.

Parameter	Value	Description
σ	0.3	Variance of noise
γ	0.99	Discount factor
Θ	10^7	Length of experience buffer
mt	0.5	Sampling time
k_{maz}	500	Maximum number of steps
ε	0.2	Threshold to reach target
κ	20000	Reward for reaching the target
β	5	Factor for control effort reward
H^{μ}	100	Number of actor hidden neurons
H^{Q}	100	Number of critic hidden neurons
τ	0.001	Target network smooth factor

Table 5.5: Parameters for training

Those problems (or reinforcement learning problems in general) are sensitive to most of the parameters and the system will not converge to acceptable performance for most of the parameter values. In general, learning rates for the actor and critic should be selected rather small, with the learning rate of the critic being larger than the learning rate of the actor. In addition, the simulation of the environment has to work flawlessly to ensure convergence. In this setting, the correct reward is key, i.e. only desired behavior should be rewarded and episodes should terminate afterwards.

1 The processing was done on an Intel i7 2600K CPU @3.40 GHz with 8 GB RAM and took ~28 hours to complete. The system is trained once for forward driving and once for backward driving and both actor networks are saved as controllers.

Figure 5.13 shows the training process of the first DDPG agent that is only used for backward driving. It can be seen that the training is successful. The cumulative reward in later episodes almost reaches 20000. This shows that the agent learned to reach the target, as positive values are only possible if the target was reached. It cannot be expected to further improve, as $\kappa = 20000$ is the upper limit. This is, because κ is only rewarded once and even with perfect behavior, it takes the truck some steps to reach the target.

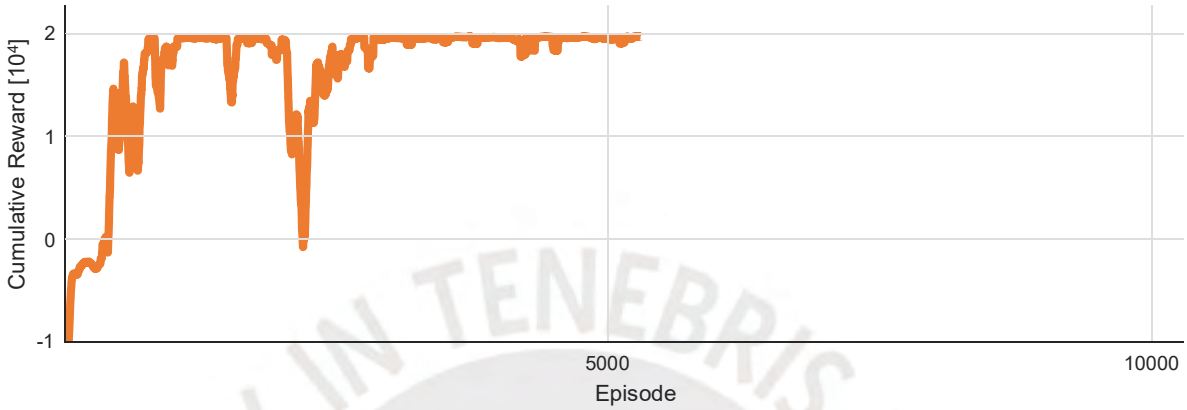


Figure 5.13: Performance curve of the backward driving DDPG agent

Figure 5.14 shows the training process of the forward driving DDPG agent. The training is successful and compared to the first agent, faster and more stable. This is because the forward driving problem is much easier that the backward driving, so the training is faster and in later episodes the reward is almost always at the same high level.

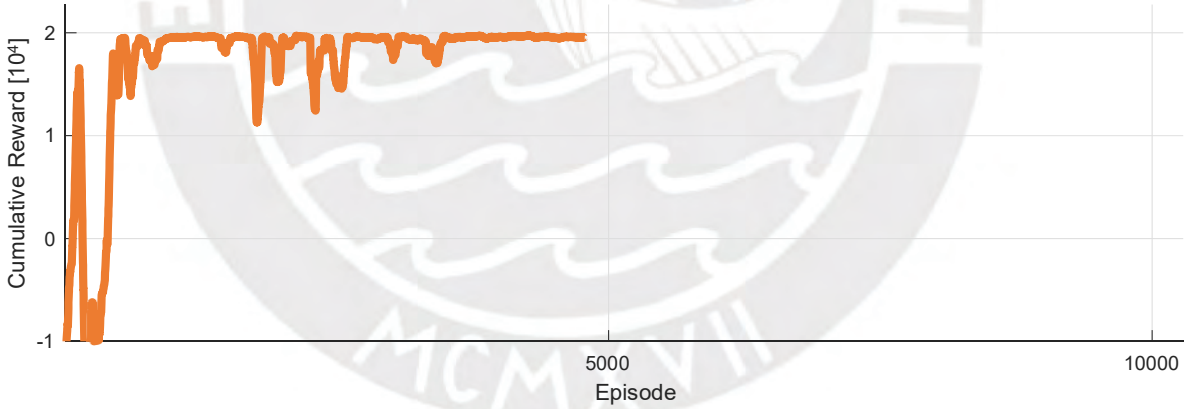


Figure 5.14: Performance curve of the forward driving DDPG agent

If the original setup does not converge for any known parameter set, the following approach can be used: Pretrain a neural network with the architecture of the actor with known controller data (e.g. from a LQR controller). The learning rate of the pretrained actor is set to 0 and the critic network is trained. After the convergence of the critic network, one has to start retraining the actor with the pretrained critic. In this way, the solution space is a lot smaller, hence the convergence should be more robust.

ET LUX IN TENEBRIS LUCET
MCMXVII

6 Simulations and Evaluation

*For the control of **autonomous** multibody vehicles using artificial intelligence, simulations have to be done, to asses the capabilities of the system regarding autonomy. This means, the system being able to deal with unknown situations.*

6.1 Implementation

The overall system was implemented in MATLAB. The main functions are:

- Build complex environments consisting of objects, targets and trucks
- Update the state of the truck with trailers (Chapter 3)
- Determine steering angle based on different controllers (Chapter 4) and artificial intelligence approaches (Chapter 5)
- Determine the desired state based on a given target trajectory (Section 4.4) and switching the driving direction based on collisions and a cost function (Section 4.3)
- Simulate the full system including interaction with the environment (Chapter 6)

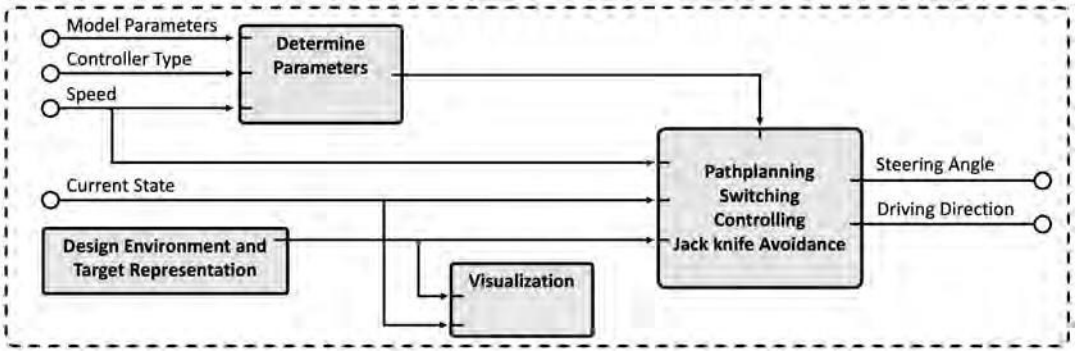


Figure 6.1: Configuration of the modules

Figure 6.1 illustrates the interaction of the respective modules.[1] In a first step, the controller parameters, e.g. control gain or parameters of a neural network, are determined based on the model parameters. Independently, the environment including obstacles and a representation of the target, i.e. a target state or trajectory is designed. In every cycle, the main control module computes a steering angle and a driving direction based on the current state, the parameters and the environment. For debugging and presentation purposes, the state is visualized in the respective environment.

1 This visualization is only conceptual, as the actual implementation is different for the sake of modularity and generality, see Appendix A.2.

6.2 Simulation Results

To investigate the proposed controllers and the overall system, a structured approach will be taken. First, several test scenarios, described in Chapter 6.2.1, will be designed (first page of every scenario). Second, the performance of the system in every single scenario will be analyzed in terms of statistical measures and overall performance and computational efficiency.

For the simulations, the test will be conducted several times with random initial position to obtain statistically relevant insights. Therefore, not only the information about success[1] or failure is relevant, but also the length of the path travelled, as well as the needed time and number of switches. Generally, a short path in a short time and with a low number of switches, all while reaching the target, is the best outcome.

The simulations were partly computed on a Workstation with Intel Core i7 2600K CPU (4 Cores @3.40 GHz) and 8GB of memory and on a 2013 MacBook Air with Intel Core i3 CPU (2 Cores @1.30 GHz) and 4GB of memory. An average simulation with a simple controller takes approximately 30 seconds to compute up to 500 seconds of simulated time including visualization.

Without visualization, the computation is around $10z$ faster. Compared to the visualization, the simulation of the environment and the collision checking, the computations for the controllers are economical in terms of computational resources. However, the required time depends on the selected controller, see Tables 6.3 – 6.10 of the simulated scenarios.

Noise

To account for measurement errors and model inaccuracies, noise will be added to the states and the control input, see Illustration 2. The noise will be sampled from a normal distribution with mean 0 and a standard deviation that depends on the value range of the respective variable, such that

$$z_{meas} := z + e_{meas}$$
$$e_{meas} \sim N(0, \sigma_z^2)$$

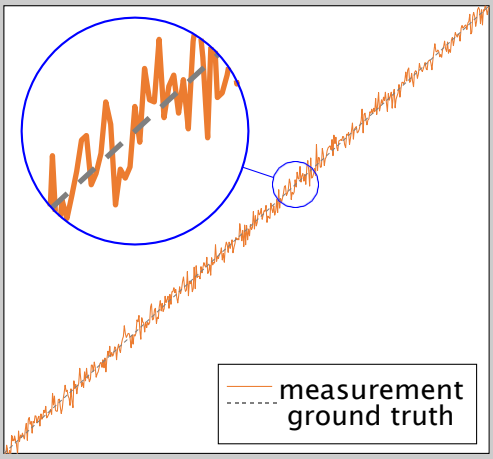


Illustration 2: Measurement noise

1 A simulation run is successful if the truck reaches the target state and stops based on the s value or follows the general course of the trajectory.

Parameters

The parameters used for the following simulations are shown in Table 6.1. The manipulatedVariablesRate (mVR) is a Matlab parameter that is given here for the sake of completeness.

Module	Parameter	Value	Parameter	Value
General:	Controller	LQR or RL or nMPC	mt	0.05 or 0.5 (nMPC)
	Pathplanning	EPDP Method[1]	t_{maz}	500s
Model:	L_1	5	L_2	15
	$v_{Forward}$	-1.5 m/s	$v_{Backward}$	1.5 m/s
	δ_{min}	$-\frac{\pi}{6}$	δ_{maz}	$\frac{\pi}{6}$
	$\vartheta_{12,maz}$	$\frac{\pi}{2}$	S	0.03
	σ_{zy}	0.3	σ_ϑ	0.03
Jack-Knife:	z_0	$-\frac{\pi}{3}$	z_1	0
	y_0	0	y_1	1
Switching:	ρ_1	1000	ρ_2	750
	$diag(R)$	[1 1 25 25]		
LQR:	$K_{LQR,v<0}$	[11.3, 137.7, -55.9]	$diag(Q_{LQR})$	[124, 100, 3000]
	$K_{LQR,v>0}$	[-11.3, 137.7, 55.9]		
nMPC:	t_c	10s	t_p	2s
	$diag(Q_{MPC})$	[0.1 0.1 0.1]	mVR	0.3
RL Agent:	Activation	relu	κ	20000
	H_i^μ, H_i^Q	100	β	0
	η^μ	0.0001	τ	0.001
	η^Q	0.001	γ	0.99
	(z_{min}, z_{maz})	$(-25, 25)$	σ	0.3
	(y_{min}, y_{maz})	$(-25, 25)$	k_{maz}	500
	$(\vartheta_{2,min}, \vartheta_{2,maz})$	$(-\pi, \pi)$	ε	0.2
	$(\vartheta_{12,min}, \vartheta_{12,maz})$	$(-1, 1)$	Θ	10^7

Table 6.1: Parameter values for the simulation

Controller and network parameters are saved, so they do not have to be computed for every execution. If relevant model parameters (e.g. the speed) change and new parameters are required, new controller and network parameters are computed by optimization of parameters or training of the networks.

1 With runtime optimizations as mentioned in the respective remarks.

6.2.1 Behavior in Test Scenarios

Table 6.2 gives an overview of the tasks that will be investigated in the following sections. Aside of the general overview, a short description introduces the basic task of every scenario.

No.	Name	Description
1	Basic Parking	Starting from a random position inside the operation area, the system has to reach a final state with a random angle without any further constraints.
2	Change Direction	Starting from an initial position close to the target with the exact opposite orientation, the system has to reach a final state in a smaller operation area.
3	Simple Trajectory	Starting from a random position inside the start area, the system has to follow a simple but nonlinear sinusoidal trajectory without any further constraints.
4	Trajectory	Starting from a random position inside the operation area, the system has to follow a nonlinear trajectory without any further constraints.
5	Slalom	Starting from a random position inside the start area, the system has to follow several partial trajectories to avoid non-solid objects.
6	Bottleneck	Starting from a random position inside the start area, the system has to pass a bottleneck with a trajectory segment to reach a final state at a loading dock.
7	Perpendicular Parking	Starting from a random position close to the target but with a perpendicular orientation, the system has to reach a final state constrained by nearby objects.
8	Parallel Parking	Starting from a random position parallel to the target with the same orientation, the system has to reach a final state constrained by nearby objects.

Table 6.2: List of all test scenarios

Scenario 1-2 test the ability to reach a target position. Scenario 3-5 are about trajectory following in different levels of difficulty. Appendix A.1.6 shows the behavior for longer trajectories. Finally, Scenarios 6-8 showcase more complex challenges including objects. Additional visualizations can be found in Appendix A.1.7.

As only the nonlinear MPC (nMPC) is analyzed in this chapter, it will be referenced as MPC from now on.

Legend

Every scenario will be studied in detail on the following pages. To illustrate the given situation, the legend shown in Figure 6.2 will be used.

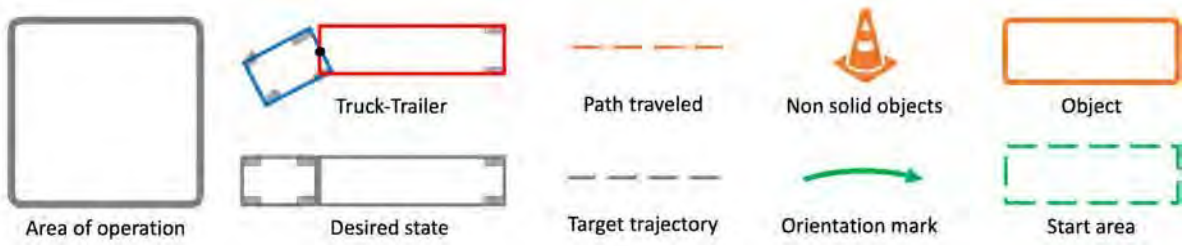


Figure 6.2: Elements in the simulation scenarios

The *area of operation* is the entire area that the system operates in. The boundaries can be seen as objects. In the real-world, this area can be a parking lot or a factory site. The *Truck-Trailer* (potentially with more than one trailer) is the system. It moves around in the area of operation and is controlled by the proposed approaches. The *desired state* is the state that the system tries to reach. It can be the only target, then it is the final or target state, or it can be accompanied by a target trajectory. The *target trajectory* is a function that describes a path inside the area of operation. When given, the system should follow the path as close as possible. The *path traveled* describes the past positions of the system. In case of trajectory following, the path traveled should be similar to the target. A *non-solid object* describes a thing that will not directly lead to a collision such as a traffic cone, but should be avoided anyway. An *orientation mark* illustrates a variable or fixed orientation of certain elements. An *object* is a solid thing inside the area of operation, such as a building or other trucks. The *start area* describes an area from which the truck can start in a given scenario. The exact start points will be uniformly sampled from that area. Note that the size of the start area refers to the position of the robot, so the rest of the robot can actually start outside of it.

Desired Behavior

The main goal of the truck-trailer system is to reach a target state or to follow a given target trajectory. When only the target state is given, the system should find a reasonable way to drive in the wanted direction, i.e. no unnecessary moves and no collisions. As there is no explicit global pathplanning in this approach, a top level planner has to make sure that the target is easily reachable (e.g. when no objects are around) or basic information about the path to the target are given. When following a target trajectory, first the system should find its way onto the trajectory and second the system should track the trajectory as close as possible.

As the system should operate in areas with humans around, the desired behavior should enable others to trust the system. This includes understandable turns and points to switch the direction, as well as safety margins in terms of obstacle avoidance. Finally, the behavior should be somewhat predictable, so that people can get used to it.

6.2.2 Basic Parking

The first scenario is about basic parking situations for a truck with one trailer. The target state is fixed and there are no objects. The robot starts from a random position inside the start area, see Figure 6.3. The only constraint is the border of the operation area.[1]

Operation area	Start area	Initial position	Target
$120m \cdot 80m$	$80m \cdot 40m$	Sampled randomly	Target state: $[0\ 0\ 0\ 0]^T$

This is the easiest test to show the basic functionality of the navigation. In real life, this situation comes into play on large and empty parking lots or during training for truck drivers.

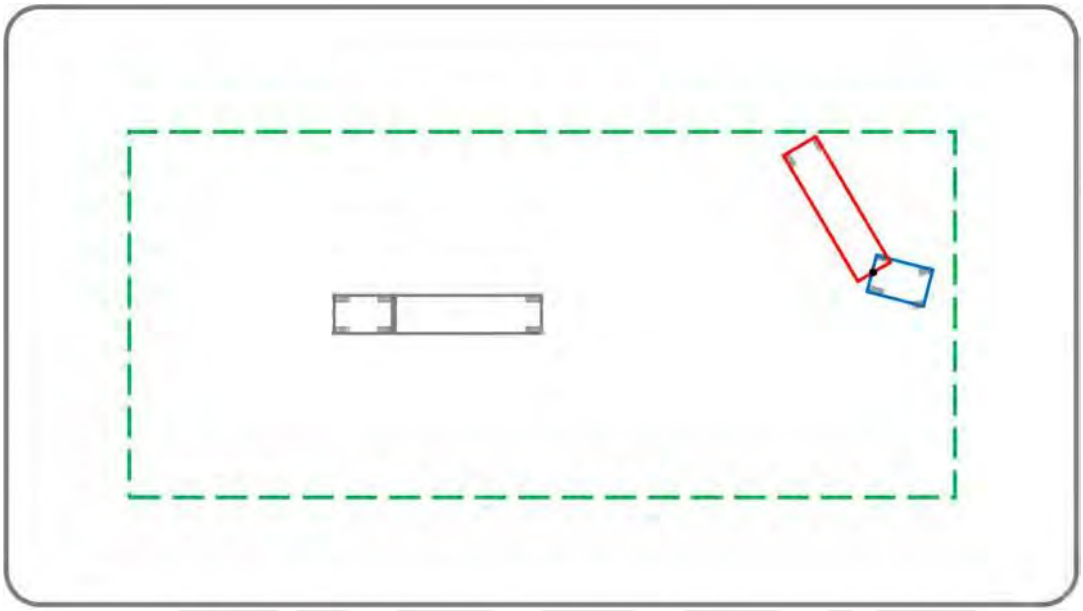


Figure 6.3: First scenario: Basic Parking

The desired behavior in this situation is simply a fast and smooth convergence to the target state. However, some switches might be necessary to avoid collision with the border or as the system might overshoot the target.

Here and in the following, $\vartheta_{12} = 0$ is assumed for target states. The implementation of the feature to enable $\vartheta_{12} = 0$ is not straightforward, but possible. To do so, the concept of an angled line to a target position with $\vartheta_2 = 0$ has to be extended. Therefore, a circle segment has to be planned adjacent to the target state with a ϑ_{12}^* such that the circle segment is followed.

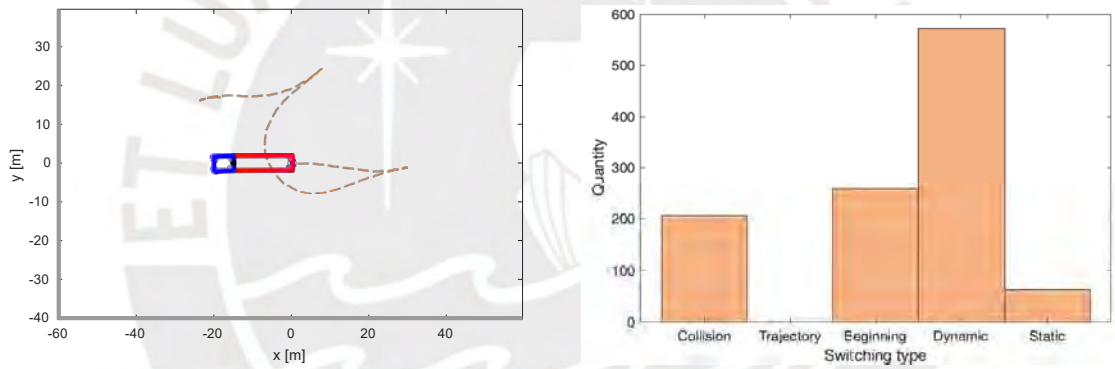
1 This scenario also refers to all parking situations without objects, as a target state with another position or angle describes the same scenario just in a different coordinate system (shifted and rotated).

Results

In 99.7 % of the cases with the RL controller, the simulation was successful. All controllers are able to complete the task, while the MPC controller has a lower success rate.[1] On the one hand, based on the simulations, the best controller is the RL controller with the shortest time and path. On the other hand, it has the cost of a long training process pre-startup and a longer computation time.

✓	RL	190m	127s	4
Success	Controller	Path length	Time passed	# switches

Figure 6.4 shows visualizations, which illustrate the behavior in space and across all runs. Figure 6.4b shows that there are 4 used switching types with dynamical switching being the one most used. The other ones are mostly to avoid the border and to switch right at the beginning in the direction of the target. In special cases, there are more switches e.g., if the start point is close to an edge of the area of operation.



(a) Exemplary path of the truck (b) Histogram of switching types

Figure 6.4: Visualization of the first scenario

Table 6.3 shows the average results for 300 runs. It can be seen that the LQR and the RL controller have a similar performance. The MPC controller requires more time for the computation with a lower success rate.

Controller	Success rate	Path length	Time passed	# switches	Compute Time
LQR	99.0 %	212.3 m	141.5 s	4.1	**1.3 s**
RL Agent	**99.7 %**	**190.8 m**	**127.2 s**	**3.6**	2.9 s
MPC	62.3 %	566.5 m	377.7 s	9.8	63.6 s

Table 6.3: Averages for the first scenario

1 This could potentially be improved by better fine tuning of the parameters.

Change Direction

In the second scenario, another parking situation for a truck with one trailer is studied. The target state is parallel to the initial position of system and there are no objects. However, the system starts with the exact opposite orientation and has to change its direction by 180° to reach the target, see Figure 6.5. The situation is constrained by a smaller area of operation.

Operation area	Start area	Initial position	Target
$60m \cdot 80m$	$20m \cdot 60m$	Initial orientation: π	Target state: $[10\ 0\ 0\ 0]^T$

This test shows more advanced navigation capabilities in tight spaces, compared to real life. The area of operation might be limited, because of buildings or other vehicles.

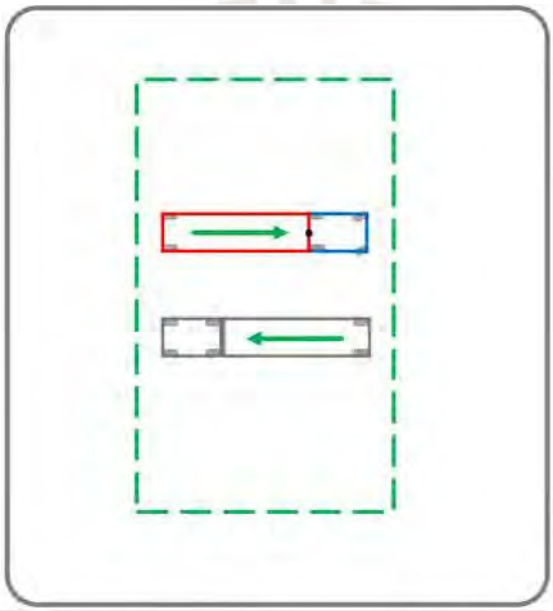


Figure 6.5: Second scenario: Change Direction

The desired behavior in this situation is a full turn with as few switches (at the border) as possible. However, the number of switches is influenced by the effectiveness and space requirements of the controller.

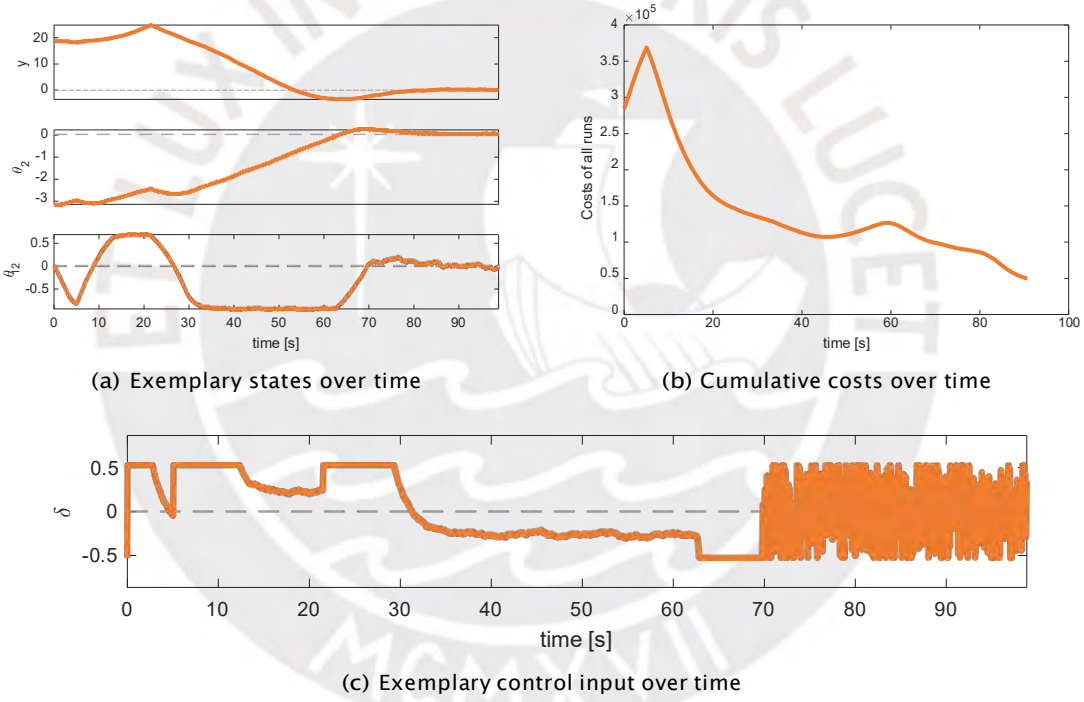
In this case, as no trajectory is given, the system plans a path indirectly by the controller. This might lead to suboptimal behavior, as the controller has no way to gain knowledge about the future as only the current state is directly fed into the control pipeline.

Results

In 94.3 % of the cases with the LQR controller, the simulation was successful. The MPC controller is not able to complete the task, while the RL controller has a lower success rate. Based on the simulations, the best controller is the LQR with the shortest time and path.

✓	LQR	275m	183s	11
Success	Controller	Path length	Time passed	# switches

Figure 6.6 illustrates the inputs and outputs over time as well as the cumulative costs of all simulated trucks over time. Figure 6.6a shows the convergence of the individual states. It can be seen that all of them converge, even with noise. Figure 6.6c shows the respective control input. The oscillations in the end result from the noise while driving forward. Figure 6.6b visualizes how the costs decrease over time for the minimum time of 90.8 s.



(a) Exemplary states over time

(b) Cumulative costs over time

(c) Exemplary control input over time

Figure 6.6: Visualization of the second scenario

Table 6.4 shows the statistics for 300 runs. It can be seen that this task is more difficult than the first scenario. In some situations, the controllers are not able to make the full turn, as the area is to small. Therefore, most of the switches are to avoid collisions.

Controller	Success rate	Path length	Time passed	# switches	Compute Time
LQR	**94.3 %**	**274.5 m**	**183.0 s**	**10.8**	**1.6 s**
RL Agent	86.3 %	306.0 m	204.0 s	16.8	4.6 s
MPC	0.7 %	749.7 m	499.8 s	21.5	156.1 s

Table 6.4: Averages for the second scenario

6.2.3 Simple Trajectory

The third scenario studies the pathfollowing behavior of the system. Starting inside the start area, a truck with one trailer has to converge to the trajectory and then stick to it. The simple but nonlinear sinusoidal trajectory is given by

$$f(z) = 20 \sin\left(\frac{z}{15}\right) \tag{6.1}$$

for $z \in [-30, 55]$, see Figure 6.7. There are no further constraints.

Operation area	Start area	Initial position	Target
$120m \cdot 80m$	$30m \cdot 40m$	Sampled randomly	Sampled with step length $h = 0.1$

This test analyzes the capabilities to follow a simple trajectory. In this scenario, the internal trajectory planning is important. A real-life truck might have a high-level trajectory planning system that determines the target trajectory.

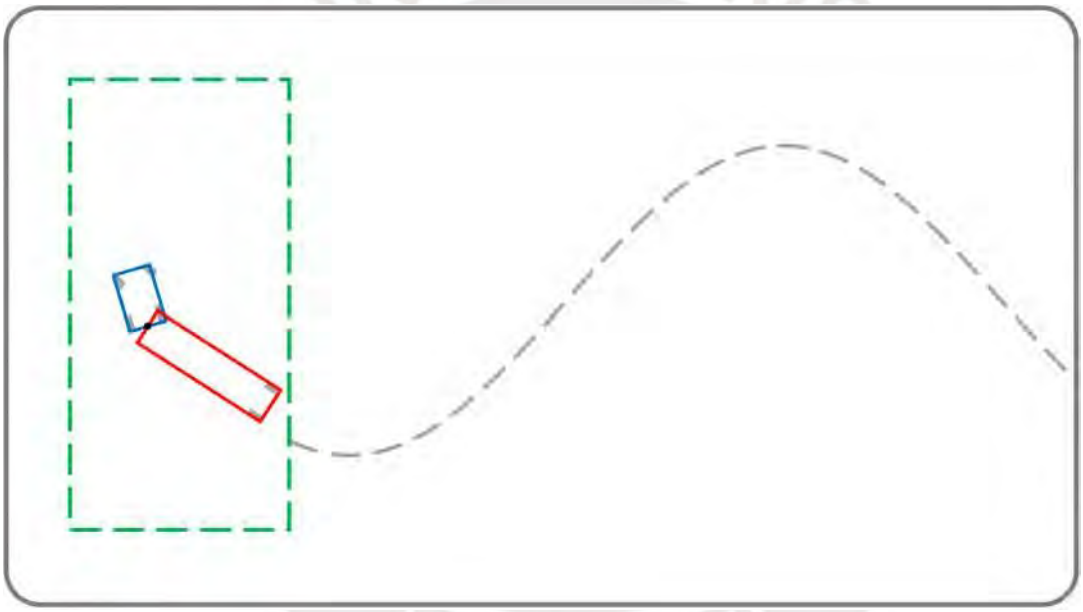


Figure 6.7: Third scenario: Simple Trajectory

The desired behavior is to converge to the trajectory as fast as possible and then following it as close as possible. As the trajectory is constantly changing, a lot of steering is required.

The desired direction is encoded in trajectory as well, so the system has to change the driving direction if it is moving in the wrong direction.

Results

In 89.7 % of the cases with the LQR controller, the simulation was successful. The RL controller is able to complete the task as well, while the MPC controller fails most of the time. Based on the simulations, the best controller is LQR with the shortest time and the highest success rate.

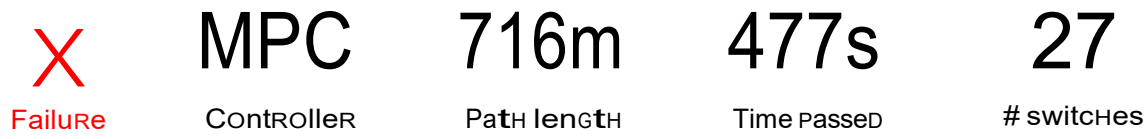
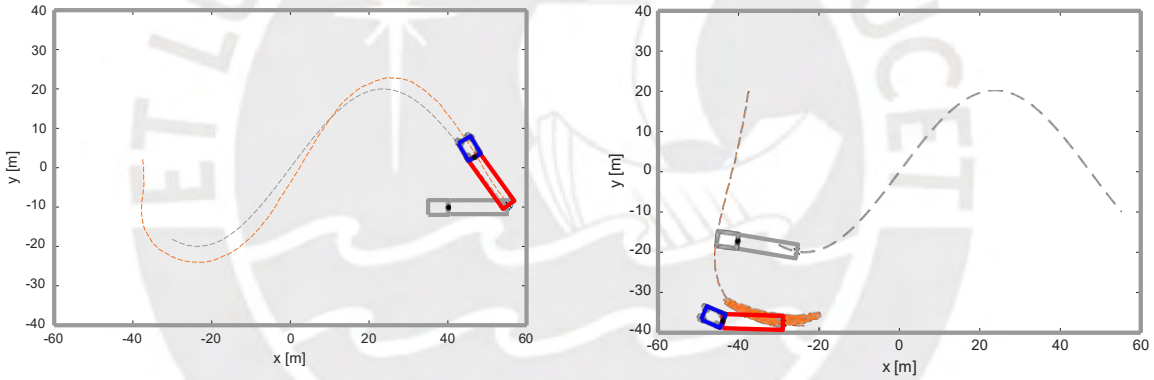


Figure 6.8 shows two exemplary paths of the system with the LQR and the MPC controller, respectively. It can be seen that the LQR controller in Figure 6.8a converges to the trajectory quickly and follows it until the end. Figure 6.8b in contrast shows the MPC controller being trapped close to a border with forward and backward driving resulting in a collision. Note that this behavior can be improved by a better collision avoidance module. This is, e.g., switching the direction some distance before reaching the border or move from the border away first after switching the direction.



(a) Exemplary path of a successful truck (b) Exemplary path of a failed truck

Figure 6.8: Visualization of the third scenario

Table 6.5 shows the statistics for 300 runs. It can be seen that LQR and RL controller can follow the trajectory, while the MPC controller has a low success rate. The shortest path in a successful run was 120.4 m (LQR), 130.6 m (RL) and 113.0 m (MPC). Note that especially the initial conditions with a wrong initial orientation lead to a failed run and longer paths.

Controller	Success rate	Path length	Time passed	# switches	Compute Time
LQR	**89.7 %**	**295.8 m**	**197.2 s**	**10.6**	**8.6 s**
RL Agent	86.3 %	312.0 m	208.0 s	11.0	11.9 s
MPC	10.0 %	715.6 m	477.0 s	27.4	160.3 s

Table 6.5: Averages for the third scenario

Complex Trajectory

In the fourth scenario, the trucks target is a complex nonlinear trajectory. Starting from a random position inside the operation area, the system has to reach and follow a nonlinear trajectory without any further constraints. The trajectory is given by

$$f(z) = 10 \sin\left(\frac{z}{30}\right) + 15\frac{z}{1 + |z|} \tag{6.2}$$

for $z \in [-30, 55]$, see Figure 6.9. There are no further constraints.

Operation area	Start area	Initial position	Target
$120m \cdot 80m$	$30m \cdot 40m$	Sampled randomly	Sampled with step length $h = 0.1$

This scenario investigates the capabilities to follow an arbitrary trajectory. Trajectories often do not follow simple shapes, so the resulting functions can be complex and challenging to describe mathematically.

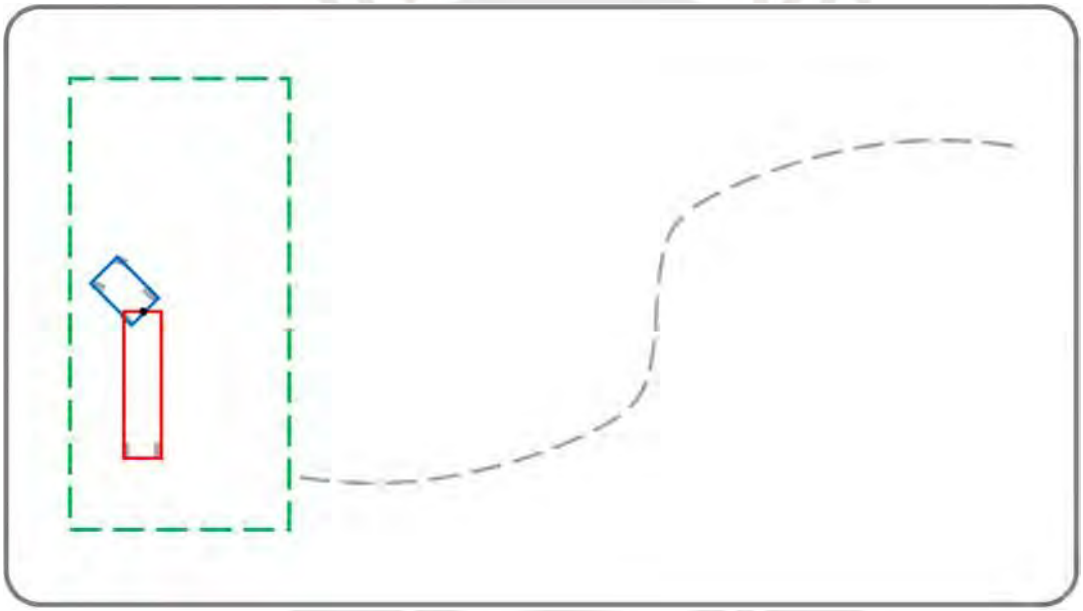


Figure 6.9: Fourth scenario: Complex Trajectory

The desired behavior is to reach the trajectory, follow it and get back to the target, even if the truck cannot follow certain parts of the trajectory (e.g. tight turns[1]) closely. The steering needs to adapt constantly to the trajectories shape.

In contrast to Scenario 3, the description of this trajectory is more complex. This is why a generalization of the planning for the desired position is crucial, as the theoretical solution cannot be known for any trajectory, so the numerical approach derived in Section 4.4 has to be used.

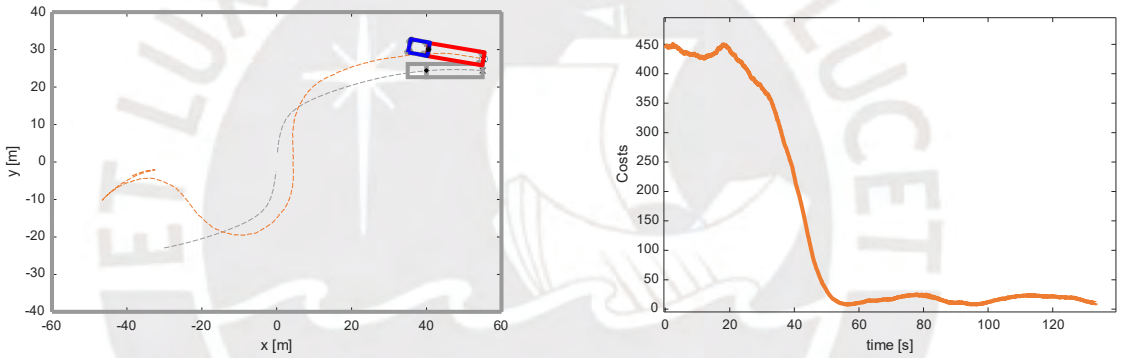
1 This can lead to edge cases with a trajectory so steep or discontinuous that it cannot be followed closely anymore.

Results

In 98.3 % of the cases with either LQR or LR controller, the simulation was successful. In general, all controllers are able to complete the task, while the MPC controller has a lower success rate. Regarding performance of the MPC controller only in *successful* cases, the average path of 353 m and the average number of switches of 4.3 is in the same order of magnitude as the other controllers.

✓	RL	190	127s	3
Success	Controller	Path length	Time passed	# switches

Figure 6.10 shows an exemplary path of the system with the respective costs over time. The truck converges to the trajectory and follows it until the end, see Figure 6.10a. The same situation can be seen in Figure 6.10b as the costs are directly related to the distance from truck to trajectory.



(a) Exemplary path of a successful truck (b) Exemplary costs over time

Figure 6.10: Visualization of the fourth scenario

Table 6.6 shows the statistics for 300 runs. It can be seen that the LQR and RL controller perform quite similar with the RL controller being marginally better. Compared to Scenario 3, it can be seen that even though Scenario 4 has a mathematically more complex trajectory, the performance is better. This might be because of the overall simpler shape with fewer and not so tight turns.

Controller	Success rate	Path length	Time passed	# switches	Compute Time
LQR	**98.3 %**	194.2 m	129.5 s	2.9	**5.6 s**
RL Agent	**98.3 %**	**189.9 m**	**126.6 s**	**2.7**	7.2 s
MPC	56.7 %	525.0 m	350.0 s	17.3	150.0 s

Table 6.6: Averages for the fourth scenario

Slalom

Scenario five represents a slalom situation. From inside the start area, the system has to follow several partial trajectories. Some non-solid objects are shown to visualize the task at hand. The target trajectory is given by

$$f(z) = \begin{cases} 20 & \text{, if } 10 \leq z \leq 25 \\ -10 & \text{, otherwise} \end{cases} \tag{6.3}$$

for $z \in [-20, -5] \cup [10, 25] \cup [40, 55]$, see Figure 6.11.

Operation area	Start area	Initial position	Target
$120m \cdot 80m$	$30m \cdot 40m$	Sampled randomly	Sampled with step length $h = 0.1$

This type of setting enables many new applications in terms of navigation. For example, the trajectory for certain parts of the way might be known (e.g. because of experience) but other parts of the way can change or are not important at all. In those cases, partial target trajectories can fix certain segments of the trajectory but leave the rest to be determined during operation.

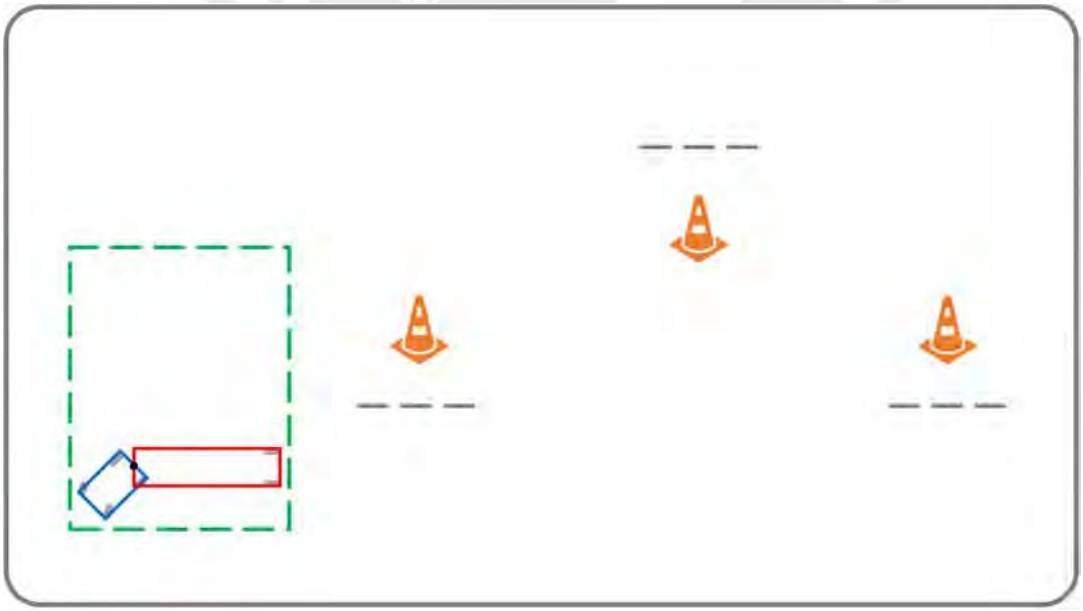


Figure 6.11: Fifth scenario: Slalom

First, the given trajectory segments should be followed as close as possible. Second, the system should reach the next segment in a reasonable manner.

This scenario introduces a new type of targets. Namely, this combines path-following with target tracking, as between segments the first point of the next trajectory segment is the target position until the segment is reached. A new segment is defined by a distance between two points on the trajectory larger than 1m.

Results

In 98.3 % of all the cases with the RL agent, the simulation was successful. The LQR controller has a similar performance. Looking at the best cases, the LQR controller has 0 switches with a path less than 200 meters in 130 seconds and the RL controller has 0 switches with a path of less than 150 meters in 96 seconds. On the other hand, by visual inspection it becomes clear that the LQR controller shows the better driving behavior and completes the slalom course correctly more often, thereby needing more time and therefore driving a longer path. Thus, the LQR controller has a better performance overall.

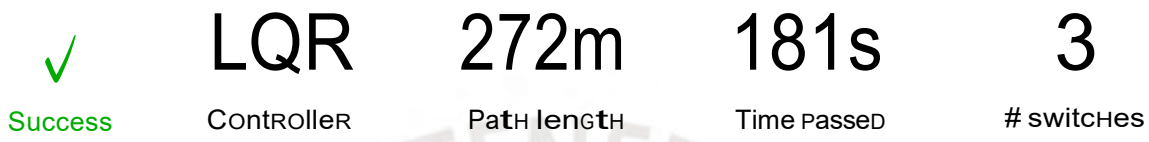
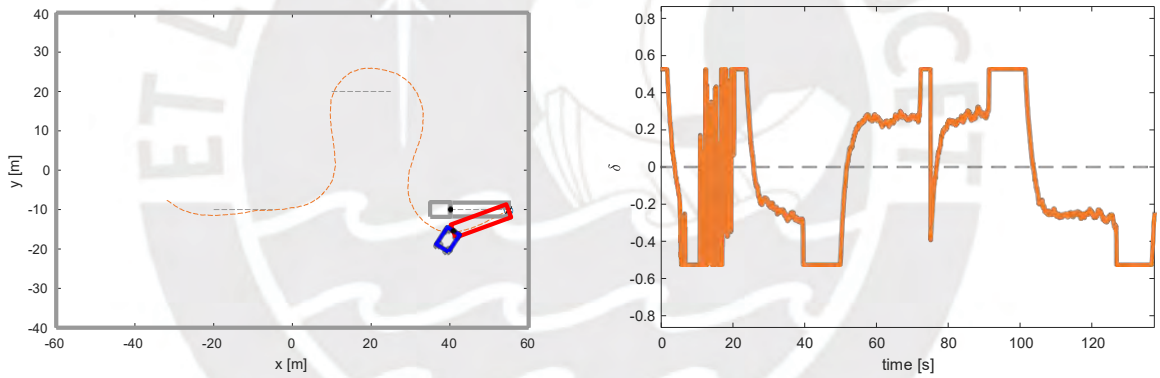


Figure 6.12 shows an exemplary path of the system with the respective control input. In Figure 6.12a, it can be seen that the slalom course is completed successfully and Figure 6.12b shows that the control input is highly dynamical with small spikes because of the noise. No switch was needed in this run.



(a) Exemplary path of a successful truck (b) Exemplary control input over time

Figure 6.12: Visualization of the fifth scenario

Table 6.7 shows the statistics for 300 runs. It can be seen that this task is not a problem for any of the controllers, but by visual inspection it becomes clear, that the MPC controller, despite a relatively high success rate, shows non satisfactory driving behavior in many cases. In addition, it needs 10 times the computation time.

Controller	Success rate	Path length	Time passed	# switches	Compute Time
LQR	**98.3 %**	272.2 m	181.4 s	3.1	**4.4 s**
RL Agent	**98.3 %**	**213.7 m**	**142.5 s**	**3.1**	5.4 s
MPC	85.7 %	356.7 m	237.8 s	6.7	56.1 s

Table 6.7: Averages for the fifth scenario

6.2.4 Bottleneck

In the sixth scenario, a complex situation with buildings, partial trajectories and a final position is tested. Starting inside the start area, the system has to pass a bottleneck with a trajectory segment to reach a final state. The bottleneck has a total width of 15 meters with the truck having a width of 5 meters. The trajectory is defined by $f(z) = 0$ for $z \in [-15, 15]$ sampled with a step length of $h = 0.1$, see Figure 6.13.

Operation area	Start area	Initial position	Target
$120m \cdot 80m$	$15m \cdot 40m$	Sampled randomly	Target state: $[53\ 25\ 0\ 0]^T$

The situation models a typical loading dock scenario for trucks. Commonly, loading docks are not located next to a street, but rather somewhere on a plant. In addition trucks often need to navigate some sort of bottleneck to reach their target, e.g. between two buildings.

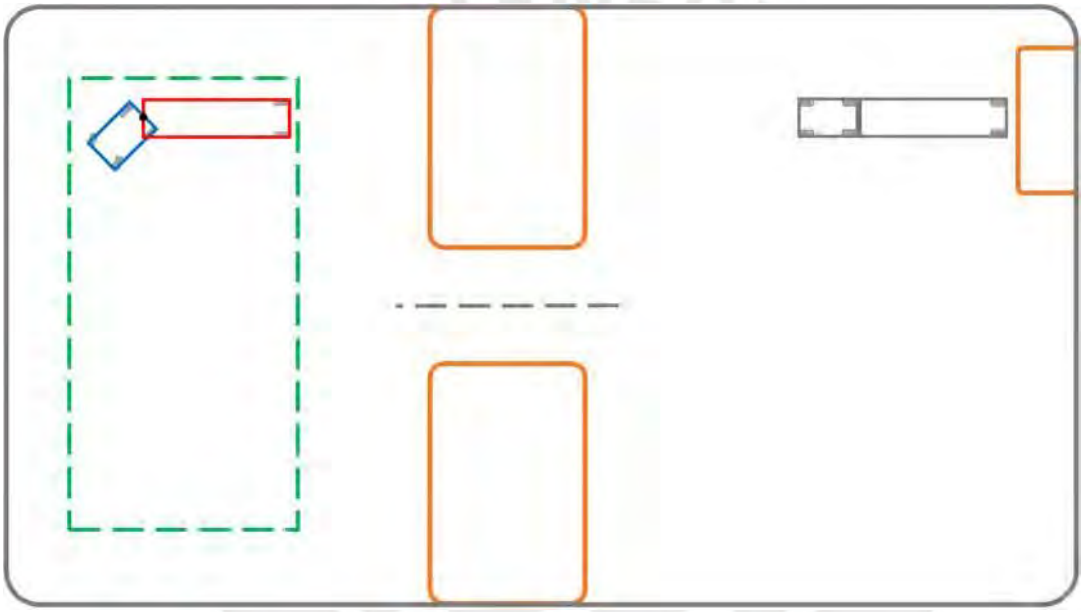


Figure 6.13: Sixth scenario: Bottleneck

In this case, the desired behavior is primarily to reach the target position without any collisions. In a second step, the distance to buildings should be safe while the number of switches as well as the time needed should be minimized.

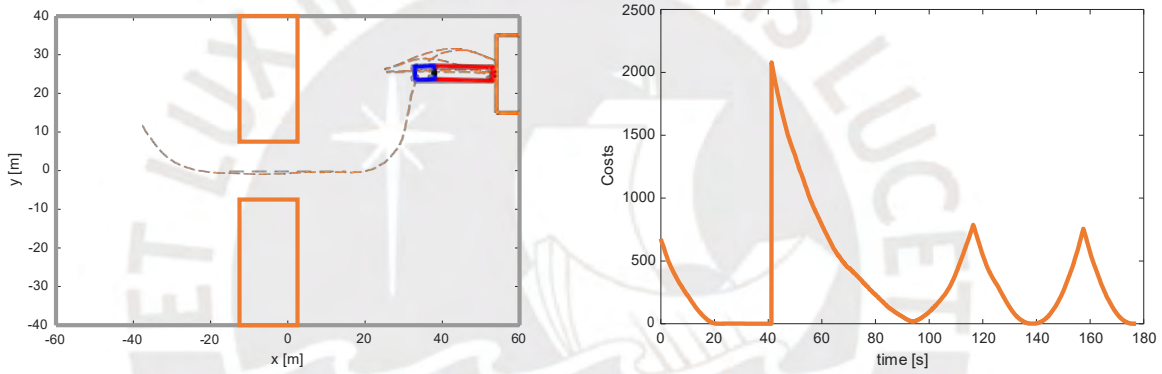
This scenario illustrates a common application of the developed system. Furthermore, the partial trajectory will not change over time as it is the only way to pass the two buildings. Without a given trajectory, it is not possible for the low-level planner to reach the target directly, as it cannot know where to drive through the buildings.

Results

In 97 % of the cases, the simulation with LQR or RL controller was successful, whereas the nMPC controller seems not to be able to complete the task.

X	MPC	749m	499s	25
Failure	Controller	Path length	Time passed	# switches

The exemplary path in Figure 6.14a follows the trajectory and tracks the final target with a few switches for the parking maneuver. Figure 6.14b shows the respective costs, which consist of several parts: from 0–20 seconds the truck approaches the trajectory, from 20–40 seconds the trajectory is followed. Then the target changes whereby the costs "jump" and from 40–90 seconds the truck approaches the target position and from 90–180 seconds a parking maneuver including four switches is performed.



(a) Exemplary path of a successful truck (b) Exemplary costs over time

Figure 6.14: Visualization of the sixth scenario

Table 6.8 shows the statistics for 300 runs. Note that the number of switches in case of a failed run is usually much higher (e.g. 85 switches for the LQR controller). This indicates that failures happen around borders or objects, when the robot gets trapped.

In contrast, MPC has not so many switches even though the success rate is almost 0. Additionally, many of the simulation runs have pretty low costs, such as 0.58 for example. In fact 66 % of the failed runs have a minimal cost below 10. This does not lead to a successful run but indicates that the MPC controller is actually able to reach the target, but does not have enough time to do so. This is, the MPC controller will be successful, eventually.

Controller	Success rate	Path length	Time passed	# switches	Compute Time
LQR	**97.3 %**	384.1 m	256.0 s	10.1	**4.0 s**
RL Agent	97.0 %	**334.6 m**	**223.0 s**	**9.6**	6.6 s
MPC	0.7 %	749.0 m	499.3 s	25.1	87.7 s

Table 6.8: Averages for the sixth scenario

Perpendicular Parking

In this scenario, a complex parking situation is analyzed. Starting from a random position close to the target but with a perpendicular orientation, the system has to reach a final state constrained by nearby objects. The parking space is $30m \cdot 7m$ while the overall truck dimensions are $20m \cdot 5m$.

Operation area	Start area	Initial position	Target
$120m \cdot 80m$	$30m \cdot 40m$	Initial Orientation: $\pm\frac{\pi}{2}$	Target state: $[55\ 0\ 0\ 0]^T$

The situation models a perpendicular parking maneuver with tight constraints in terms of the parking position. This is the case on parking lots, i.e. the nearby objects are other vehicles, or when entering a parking garage.[1]

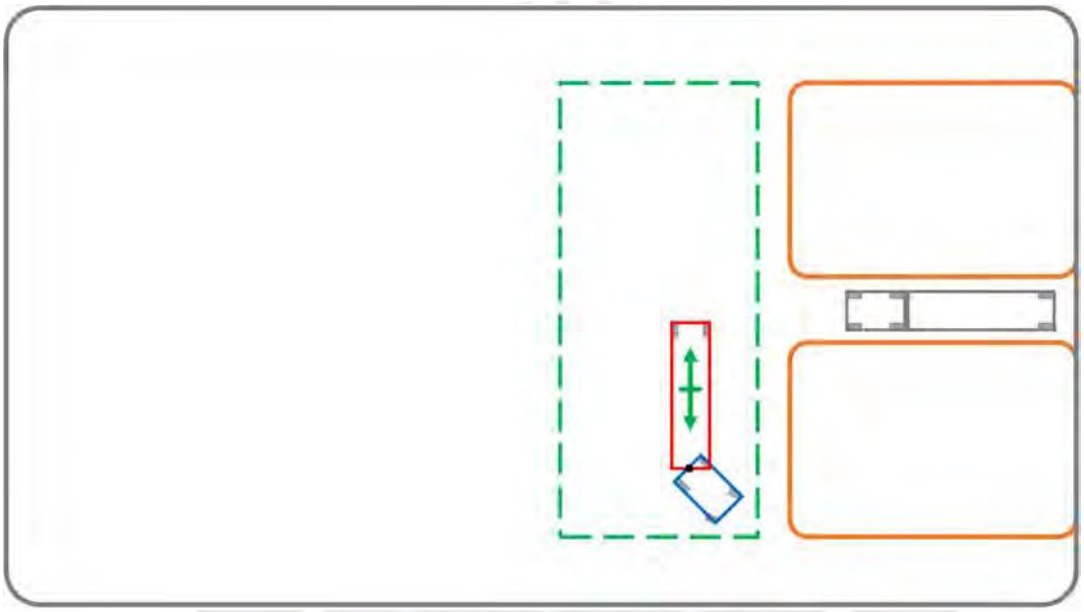


Figure 6.15: Seventh scenario: Perpendicular Parking

The desired behavior is to reach the parking position without any collision. Furthermore, few or even no switches while not using much space describes the perfect behavior.

To accurately model a real-world parking situation, it is not assumed that enough space is available. This can lead to forced changes in the driving direction, such as in a real-world parking situation. Note that the forward parking case is not shown here as it is easier and more uncommon.

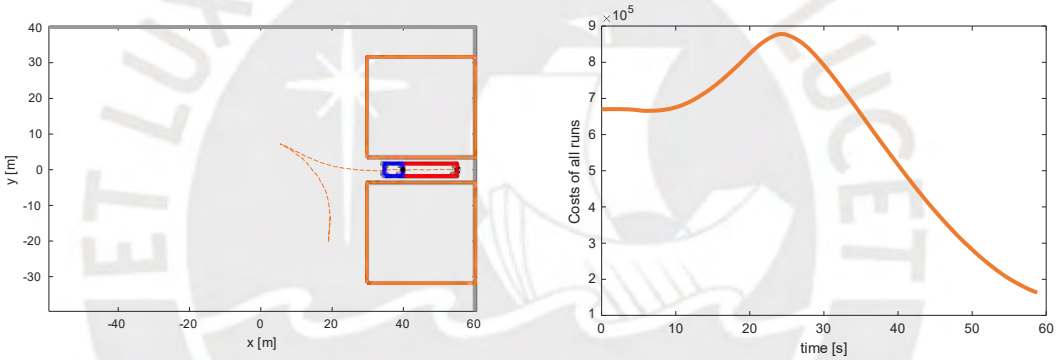
1 Watch https://www.youtube.com/watch?v=jhhqkHsGrsA for a real life video of a human driver performing a similar task.

Results

In over 99 % of all the cases with LQR or LR agent, the simulation was successful. The MPC controller has a low success rate, because it is not able to enter the small space between the two objects most of the runs.

✓	RL	179m	119s	6
Success	Controller	Path length	Time passed	# switches

Figure 6.16a shows an exemplary path of the system, which illustrates the parking behavior. Often there are switches to enter the space between the buildings and sometimes there are additional switches to improve the position. Figure 6.16b displays the cumulated costs over time. One can observe that in the first 30 seconds the costs are increasing, because the truck needs to move away from the target to enter the parking area. After that, the costs decrease monotonously.



(a) Exemplary path of a successful truck (b) Cumulated costs over time

Figure 6.16: Visualization of the seventh scenario

Table 6.9 shows the statistics for 300 runs. Additionally, the control effort can be investigated. It is 103 rad/run with a standard deviation of 63.7 for the RL agent. This indicates that the control effort varies a lot from run to run. In addition, the effort depends heavily on the update frequency of the controller.

Controller	Success rate	Path length	Time passed	# switches	Compute Time
LQR	**99.7 %**	207.0 m	138.0 s	6.2	**1.5 s**
RL Agent	99.3 %	**178.9 m**	**119.3 s**	**5.8**	3.0 s
MPC	8.3 %	730.0 m	486.7 s	42.4	62.3 s

Table 6.9: Averages for the seventh scenario

Parallel Parking

In this scenario, two parking situations (case A and B) are analyzed. Starting from position parallel to the target with the same orientation, the system has to reach a final state constrained by nearby objects. In the first case, the parking spot is only constrained on the left side of the target position. In the second case, the target position is constrained on three sides, leaving only the right side open. The resulting parking space is $50m \cdot 6m$ while the overall truck dimensions are $25m \cdot 5m$.

Operation area	Start area	Initial position	Target
$120m \cdot 40m$	$90m \cdot 12.5m$	Initial Orientation: 0	Target state: $\mathbf{[}10\ -8\ 0\ 0\mathbf{]}^T$

The situation models a parallel parking maneuver with constraints in terms of the parking position. This is the case on parking lots, i.e. the nearby objects are other vehicles, or while parking directly next to the street.

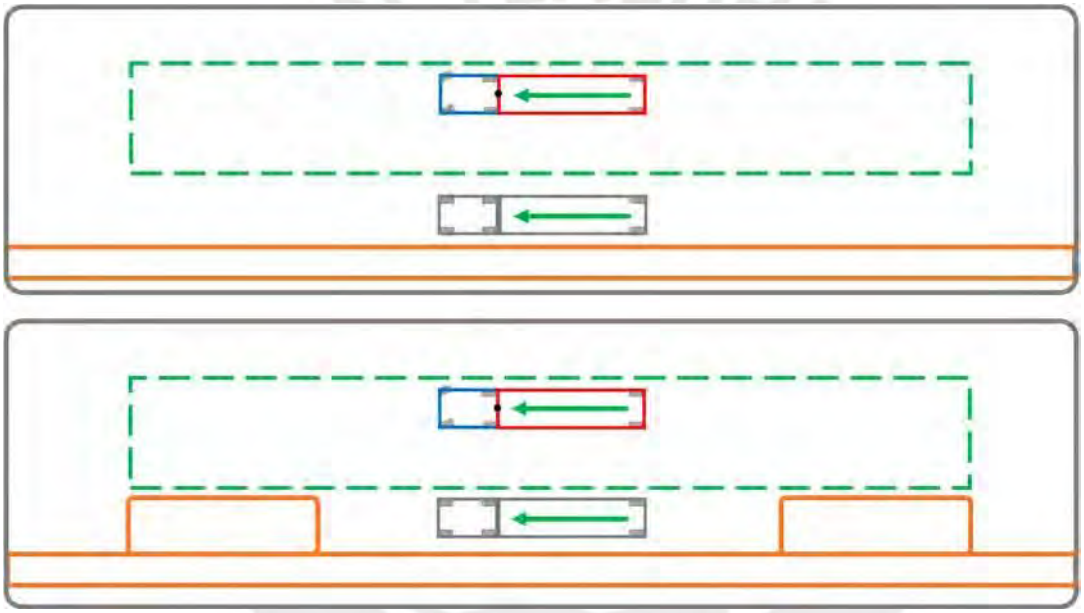


Figure 6.17: Eighth scenario: Parallel Parking

The desired behavior is to reach the parking position without any collision. Furthermore, few or even no switches while not using much space describe the perfect behavior.

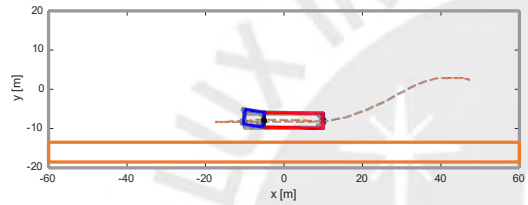
To accurately model a real-world parking situation, it is not assumed that enough space is available. This can lead to forced changes in the driving direction, such as in a real-world parking situation. Note that this model might represent a street, so other vehicles can constrain time and space even more.

Results

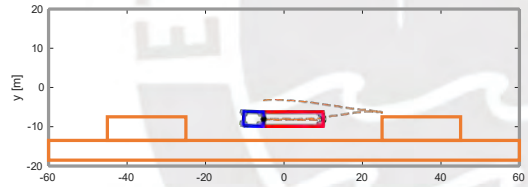
In 100 percent of the runs with case A, the simulation for was successful using the LQR or RL agent. Case B has a slightly lower success rate. All controllers are able to complete the task, while the MPC controller has a long computation time.

✓	LQR	182m	121s	4
Success	Controller	Path length	Time passed	# switches

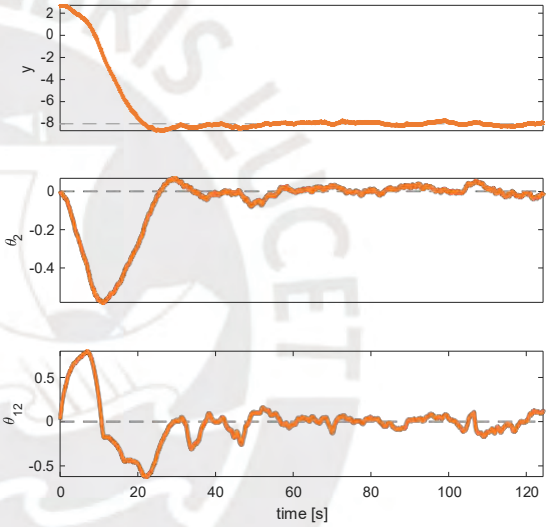
Figures 6.18a and b show exemplary paths of the system for cases A and B, which illustrate the convergence behavior. Figure 6.18c displays the respective states over time. It can be observed, that the first and second component converge quickly, while the third component has small spikes to account for the noise.



(a) Exemplary path of a successful truck (case A)



(b) Exemplary path of a successful truck (case B)



(c) Exemplary states over time

Figure 6.18: Visualization of the eight scenario

Table 6.10 shows the statistics for 300 runs per case. It can be seen that case B is more challenging than case A, but most of the measures are in the same order of magnitude. The shortest path for this scenario is shorter than 50 meters (e.g. 47 for LQR and 38 for the RL agent). Note that all of this happens *without* a high-level planner. Only a low-level planner without predetermined information about the path and the objects is used.

Contr.	Success rate	Path length	Time passed	# switches	Comp. Time
LQR	**100** | **93.0 %**	182.3 | 238.3 m	121.5 | 158.9 s	3.7 | 11.5	**1.2** | **1.8 s**
RL A.	**100** | 92.7 %	**144.5** | **168.5 m**	**96.3** | **112.3 s**	**2.7** | **10.0**	2.3 | 2.9 s
MPC	70.3 | 75.7 %	506.3 | 525.1 m	337.6 | 350.0 s	10.2 | 18.5	44.3 | 47.8 s

Table 6.10: Averages for the eight scenario

6.3 Summary

Overall, the LQR controller and the RL agent performed well with a success rate of 96.62 % and 95.32 %, respectively over all 2700 runs of the 9 scenarios. In the simulated runs, the controller has a worst-case scenario performance of 89.7 % (scenario 3), while the agent has at least 86.3 % success (scenario 2 & 3).

In general, the LQR controller and the RL agent are comparable in terms of performance, while they differ in the design process, as the LQR controller requires model knowledge, while the RL agent can be trained using only observations from the actual system. On the other side, the training and fine-tuning of the parameters takes a lot of computational resources for the DDPG approach.

The MPC controller has an overall success rate of 41.16 % with some scenarios being sufficient (e.g. Scenario 5 with 85.7 %) while others fail entirely (e.g. Scenario 6 with 0.7 %). The MPC controller has the additional disadvantage that the time for computation is $10z$ the time of the other controllers. That said, the performance, as well as the computational efficiency can be improved with better parameters and design, as this was not the primary goal of this work.

From time to time, even the best performing controllers get "trapped" close to edges. In this situation, forward and backward movement (according to the current control signal) leads to a collision. This way, the direction is changed forever, and the truck is stuck. This is, because no high-level obstacle avoidance is implemented that can switch the direction earlier or change the control signal to move away from an object first.

In addition, sometimes the movement of the controller does not track the given trajectories sufficient. This might be the case because of complex trajectories or the form factor of the truck. This is, because the long trailer makes it hard to follow tight turns while driving backwards.

7 Conclusions and Outlook

In this work, several models for multibody robots were derived and analyzed. Especially the standard truck-trailer model was then used for further investigation. It has been shown that a linearized model can be used for the control of the nonlinear system. Furthermore, a plurality of controllers were introduced for setpoint tracking and trajectory following. In addition, the control strategy was extended by the switching of the driving direction to avoid obstacles and to eventually reach the target position even in complex environments.

Next, artificial intelligence techniques were presented to improve the control performance, especially in difficult cases. Therefore, deep reinforcement learning was used to learn the best actions in a plurality of scenarios. Simulations showed that such approaches have a good performance in several test cases, including obstacles, nonlinear trajectories and switching between forward and backward movement.

There are many advantages to the system presented: First, it can successfully navigate, simple and complex situations, including objects, nonlinear and partial trajectories as well as target state tracking. Second, the system automatically switches the driving direction if needed, e.g. to avoid collisions, to reach the target faster or to follow trajectories. In addition, the system can handle noise and use different controllers, such as classical controllers, neuro-controllers or model-predictive controllers.

On the other hand, the system has no high-level object avoidance so far and is based on some assumptions, e.g. the availability of truck dimensions. For example, it was assumed that the length of the truck and all trailers are known. In the real world, this might not be the case, especially if new trailers or more than one trailer are used. Therefore, a state estimator or an adaptive controller could be necessary to estimate the length of the trailers. Additionally, the current system can only operate with constant velocity.

Advantages	Disadvantages
+ Navigate complex environments	– Gets "trapped" around edges
+ Switch direction automatically	– Some conditions apply
+ Use of different controllers	– Only constant velocity possible
+ Handle noise	

Future Work

In the future, how to deal with edge cases, such as being stuck between objects and how to determine the optimal point in time to switch the direction should be investigated. Potential performance improvements can be achieved by preview control for certain trajectory-following situations or using the *guard for pilot* principle.

Guard for pilot principle

The guard for pilot principle uses two parts to control a given system. The so-called *pilot*, usually a complex artificial intelligence controller, is used to compute driving instructions. As it is difficult to obtain a rigorous proof of stability for such systems, another simple controller, the *guard*, such as a LQR controller, is used to determine a set of possible actions. Furthermore, steering angles larger than the maximum, actions leading to objects and other dangerous actions can be excluded from the set of possible actions of the AI controller. This way, the performance can be improved by also guaranteeing stability of the resulting system.

To improve the switching module, further switching conditions can be introduced. Switching based on the angle of the truck relative to the target position and angle can lead to faster convergence and a smoothed driving behavior.

To extend the current system, that only consists of a low-level planner, a path can be planned by using a combination of planning with a noise-free model and incorporating different controllers to do so. This involves planning different paths and even combining path segments from two or more different controllers, e.g. the first segment with the LQR controller and after a change in direction switching to the RL controller.

Another way to improve the reinforcement learning agent is to use images that represent the environment as an input. This means using a pixel image (e.g. 320 x 320 pixels) instead of only a 3-element vector for training. This way, the controller is able to avoid obstacles by itself (as they are part of the environment) and can directly *plan* ahead.

As mentioned before, the problem with more than one trailer is also relevant. The modelling is considered in this work, while the explicit control was not discussed in detail. To control a truck with two trailers, only a few changes have to be made: either retrain the reinforcement learning agent with a two-trailer model and compute new controller gains for the simplified model to control the model directly. Alternatively, the current setup can be used for forward driving, as the system is stable in those cases and the second trailer will just follow the first one. This also leads to another way to approach targets. As backwards driving with two or more trailers is challenging and the stabilization task, i.e. driving straight backwards and just having noise, is more workable, the truck could move such that it drives just forward to reach a line directly to the target and then move straight backwards to finally reach it.

A Appendix

A.1 Theoretical Appendix

A.1.1 Controllability

A linear system $\dot{z} = Az + Bu$ is called *controllable* if the Kalman matrix K has full rank

$$rank(K) = rank([B\ AB\ A^2B\ \dots\ A^{n-1}B]) = n\ . \tag{A.1}$$

In this case with n = 3, assuming $v = 0$,

$$rank(K) = rank([B\ AB\ A^2B]) = rank\begin{bmatrix} 0 & 0 & \frac{v^3}{L_1L_2} \\ 0 & \frac{v^2}{L_1L_2} & \frac{v^3}{L_1L_2^2} \\ -\frac{v}{L_1} & -\frac{v^2}{L_1L_2} & -\frac{v^3}{L_1L_2^2} \end{bmatrix} = 3 \tag{A.2}$$

holds and therefore the system is controllable[1]. This is, there exists a feedback such that the system can reach any desired state.

1 Note that in the case of $v = 0$ the system does not move and is therefore not controllable.

A.1.2 The classical Runge-Kutta-Method

In the following section, a representative of the class of numerical solution methods is presented. The classical Runge-Kutta method (RK4) is an explicit 4-step method for solving initial value problems of ordinary differential equations. The RK4 offers a good compromise between accuracy (all discretization errors up to the third derivative are compensated) and speed or computational effort, since only four function evaluations are necessary.

Given an ordinary first-order differential equation of the form

$$\dot{y}(t) = f(t, y(t)) \tag{A.3}$$

with $t \in \mathbf{R}$, $y : \mathbf{R} \to \mathbf{R}^r$, $r \in \mathbf{N}$ and $f : \mathbf{R}^{r+1} \to \mathbf{R}^r$. With the known initial condition $y(t_0) = y_0$ (first order problem). If f is four times continuously differentiable, the method has consistency order 4.

Fixing a step length $h \in \mathbf{R}^+$, an approximation for $u_{i+1} \approx y(t_{i+1})$ can be obtained. The recursive equation

$$u_{i+1} = u_i + h \cdot \Phi(t_i, u_i, f, h) \tag{A.4}$$

$$\Phi(t_i, u_i, f, h) = \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4 \tag{A.5}$$

with

$$\begin{aligned} k_1 &= f(t_i, u_i) \\ k_2 &= f\left(t_i + \frac{h}{2}, u_i + \frac{h}{2}k_1\right) \\ k_3 &= f\left(t_i + \frac{h}{2}, u_i + \frac{h}{2}k_2\right) \\ k_4 &= f(t_i + h, u_i + hk_3) \end{aligned}$$

holds.

A.1.3 Fuzzy Control

A fuzzy controller as described in [50] can be used to control a truck-trailer vehicle. In this case, no mathematical model is required. Still, there are a few design parameters to be selected, e.g. by an expert. Then, only the state x and the target state x^* are needed.

In the case of one trailer, a virtual controller for the trailer is used. Therefore, the desired input angle for the trailer and the distance to the desired y-position are determined:

$$\gamma^* := -sign(v) \cdot (\vartheta_2 - \vartheta_2^*) \qquad \text{and} \qquad dist := y - y^* . \tag{A.6}$$

Next, those values are fuzzified using a fuzzy function of type B, as introduced in Section 4.1, with the angle AE (e.g. $\frac{\pi}{4}$) for γ and the distance DE (e.g. 80m) for $dist$:

$$\begin{aligned} \mu_\gamma &:= [\mu_{\gamma,1} \ \mu_{\gamma,2} \ \mu_{\gamma,3}]^T = f_B(\gamma^*, -AE, AE, 0, 1) \\ \mu_{dist} &:= [\mu_{dist,1} \ \mu_{dist,2} \ \mu_{dist,3}]^T = f_B(dist, -DE, DE, 0, 1) . \end{aligned} \tag{A.7}$$

The *correlation-minimum inference formula* $\mu_{u,i} = min\{\mu_{\gamma,j}, \mu_{dist,k}\}$ with $i := 3(k-1) + j$ is used for fuzzy inference. To *defuzzify* the values before applying them to the system, the *centroid defuzzification formula* yields the virtual control by

$$u_{virtual} = \frac{\sum_{i=1}^{n} u_i \mu_{u,i}}{\sum_{i=1}^{n} \mu_{u,i}} \tag{A.8}$$

where n is the number of rules, and u_i is 0 for $i \in \{1, 5, 9\}$, $-B_u$ for $i \in \{2, 3, 6\}$ and B_u for $i \in \{4, 7, 8\}$. This is the corresponding set of rules and B_u is another design parameter, which limits the angle between truck and trailer, e.g. $B_u = \frac{\pi}{6}$. Finally, the steering angle δ can be obtained by its inverse relationship to the required difference of the angle between truck and trailer, see Chapter 3.4. Let $m\vartheta_{12} := \vartheta_{12} - u_{virtual}$ and with a fuzzy function of type A, δ can be obtained: $\delta = -sign(v) \cdot f_A(m\vartheta_{12}, -IN, IN, \delta_{max}, \delta_{min})$, e.g. with $IN = \frac{\pi}{2}$. The step response for the y-component is shown in Figure A.1.

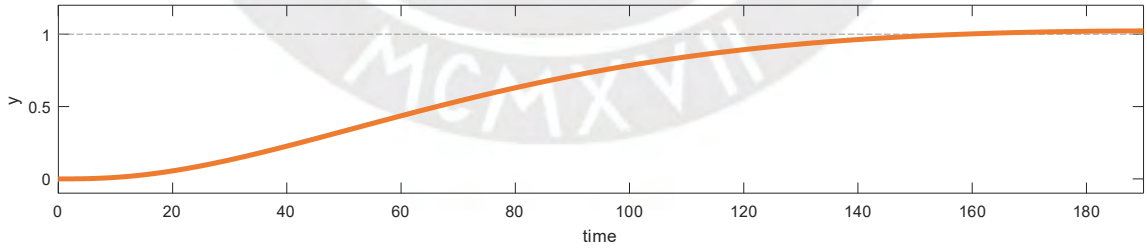


Figure A.1: Step response of fuzzy controller for target point $[1\ 0\ 0]^T$

This controller is 5z slower than the LQR approach, but the performance can possibly be increased by better expert knowledge.

A.1.4 Active-Set-Methods

Given constraints and a function f, which should be minimized.[1] The problem

$$\min_{\mathbf{x}} f(\mathbf{x}) \tag{A.9}$$

s.t.

$$g(\mathbf{x}) = \mathbf{0}, \qquad h(\mathbf{x}) \geq \mathbf{0}, \qquad \mathbf{x}_{min} \leq \mathbf{x}_j \leq \mathbf{x}_{maz} \tag{A.10}$$

should be solved. For a nonlinear f , a problem of this form can be solved with active set methods. For this a valid start value $\mathbf{x}^0$ is required. This is iteratively improved. For this purpose, f is approximated by a square function and g or h by linear functions [68].

A.1.5 Implementation of the Separating Axis Theorem

Algorithm 2 describes the implementation of the SAT for two rectangles.

input : Two representations *rectangles* = [*rect*1, *rect*2] with 5 points each
output: Boolean value *doOverlap* indicating if collision was detected

```
1  checkCollision:
2  for i = 1:2 do
3      // loop through all edges, except first (as is equal to last)
4      for j = 2:5 do
5          rect = rectangles(i);
6          z_ref = rect(j − 1, 1);
7          y_ref = rect(j − 1, 2);
8          // compute the perpendcular to the edge vector
9          z_rot = −rect(j, 2) + rect(j − 1, 2);
10         y_rot = rect(j, 1) − rect(j − 1, 1);
11         for k = 1:4 do
12             // compute projection of vertex onto perpendicular edge
13             s1(k) = sign(z_rot ∗ (rect1(k, 1) − z_ref) + y_rot ∗ (rect1(k, 2) − y_ref));
14             s2(k) = sign(z_rot ∗ (rect2(k, 1) − z_ref) + y_rot ∗ (rect2(k, 2) − y_ref));
15         end
16         // check if vertices are on different sides of the edge
17         if (min(s1) > −1 and maz(s2) < 1) || (maz(s1) < 1 and min(s2) > −1) then
18             // case polygons are intersected, so return 0
19             return 0;
20     end
21 end
22 return 1;
```

Algorithm 2: Function to detect collisions between rectangles

1 The terms used in the appendix differ from the terms used in the rest of the thesis. In particular, $\mathbf{x}$ here is any vector and f is any function.

A.1.6 Visualization of longer Trajectories (LQR)

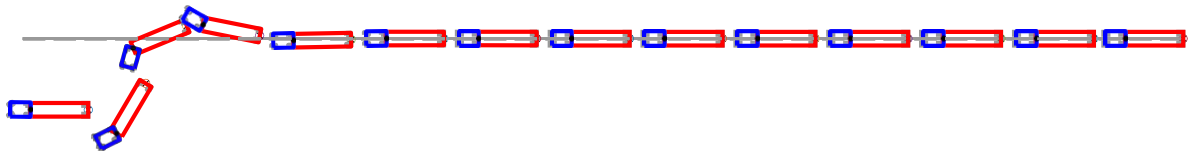


Figure A.2: Behavior of the system with a simple line as target

Figure A.3: Behavior of the system with an angled line as target

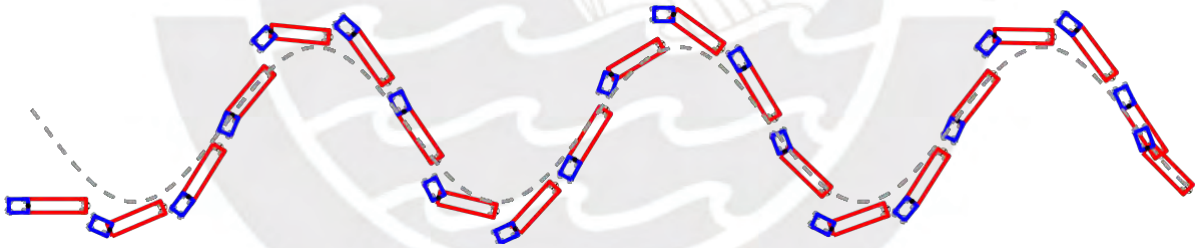


Figure A.4: Behavior of the system with a sinus trajectory as target

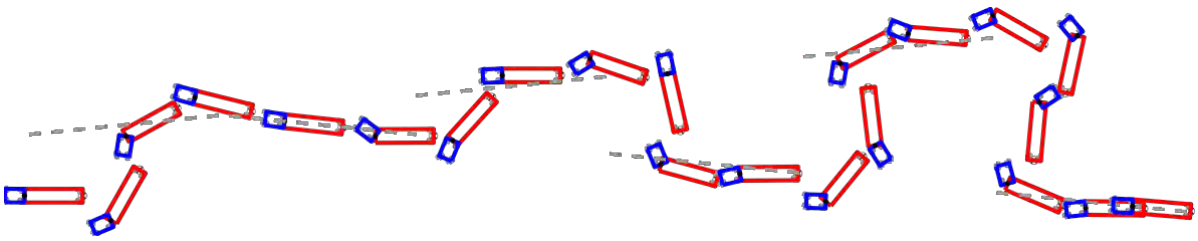


Figure A.5: Behavior of the system with a non-continuous trajectory as target

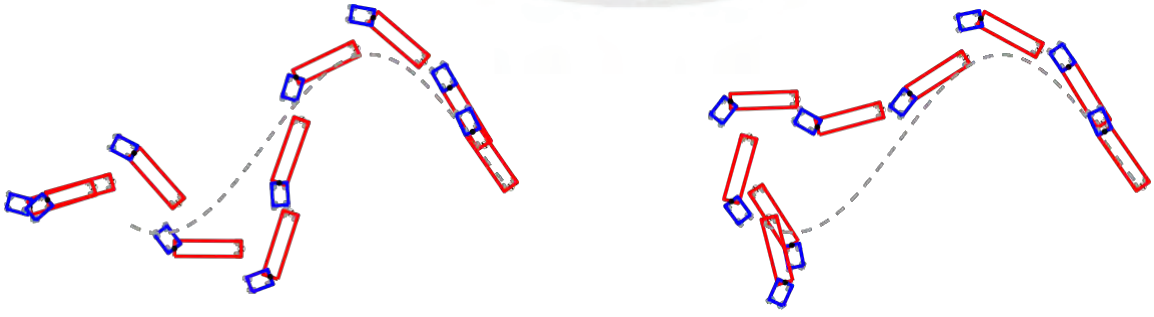
A.1.7 Additional Visualizations for the Scenarios (LQR)

(a) Exemplary path (b) Alternative path

Figure A.6: Typical behavior for Scenario 1

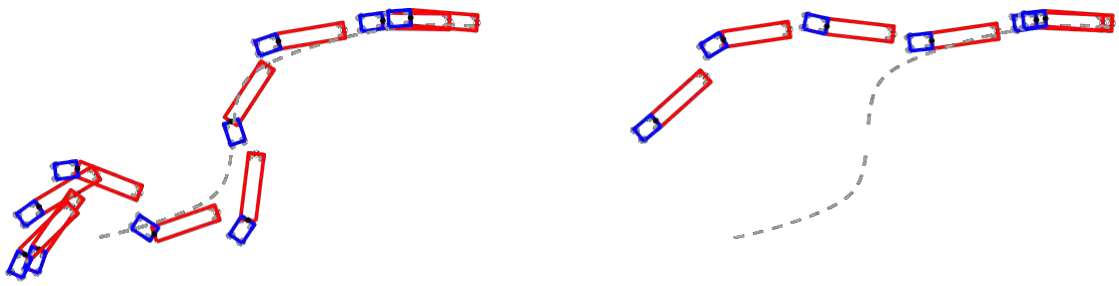
(a) Exemplary path (b) Alternative path

Figure A.7: Typical behavior for Scenario 2



(a) Exemplary path (b) Alternative path

Figure A.8: Typical behavior for Scenario 3

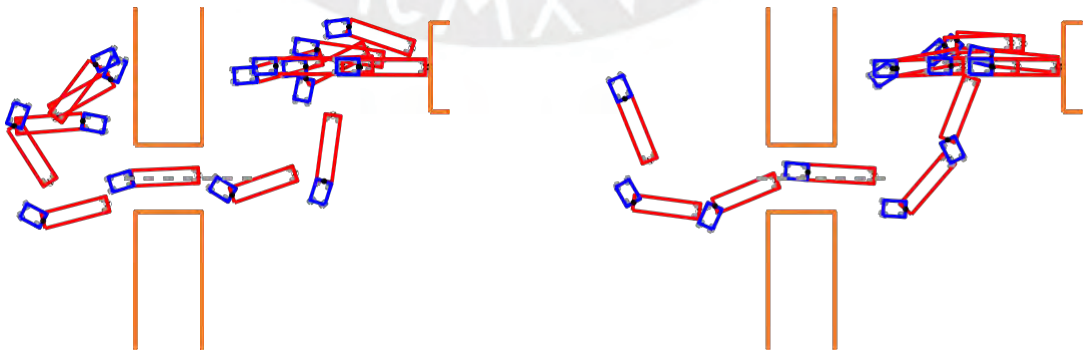


(a) Exemplary path (b) Alternative path

Figure A.9: Typical behavior for Scenario 4

(a) Exemplary path (b) Alternative path

Figure A.10: Typical behavior for Scenario 5

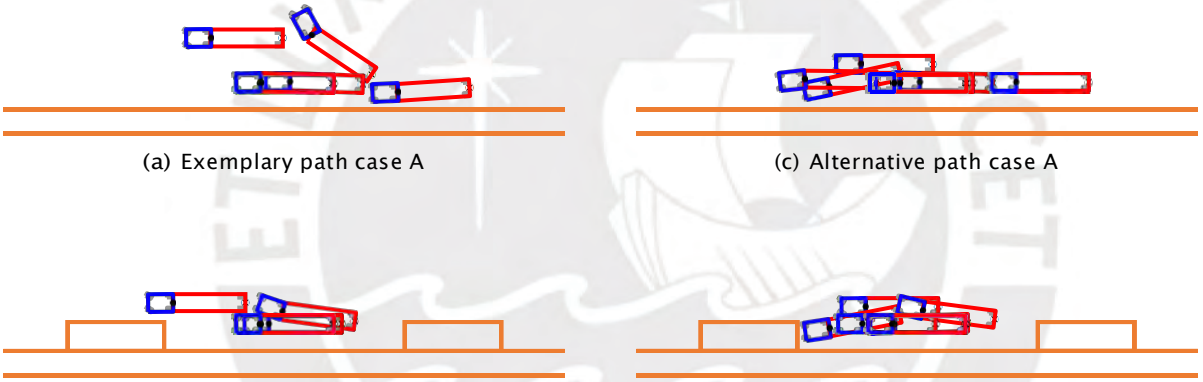


(a) Exemplary path (b) Alternative path

Figure A.11: Typical behavior for Scenario 6

(a) Exemplary path (b) Alternative path

Figure A.12: Typical behavior for Scenario 7



(a) Exemplary path case A (c) Alternative path case A

(b) Exemplary path case B (d) Alternative path case B

Figure A.13: Typical behavior for Scenario 8

All plots show the trucks state, sampled every 16 seconds.

A.2 Software Architecture

To better understand the software architecture and the interplay between the different classes and methods, the most relevant classes will be presented below. In this section, classes and structures will be written like `Class` and methods are referenced by *method()* (even if the method has parameters). The overall software architecture is illustrated in Figure A.14.

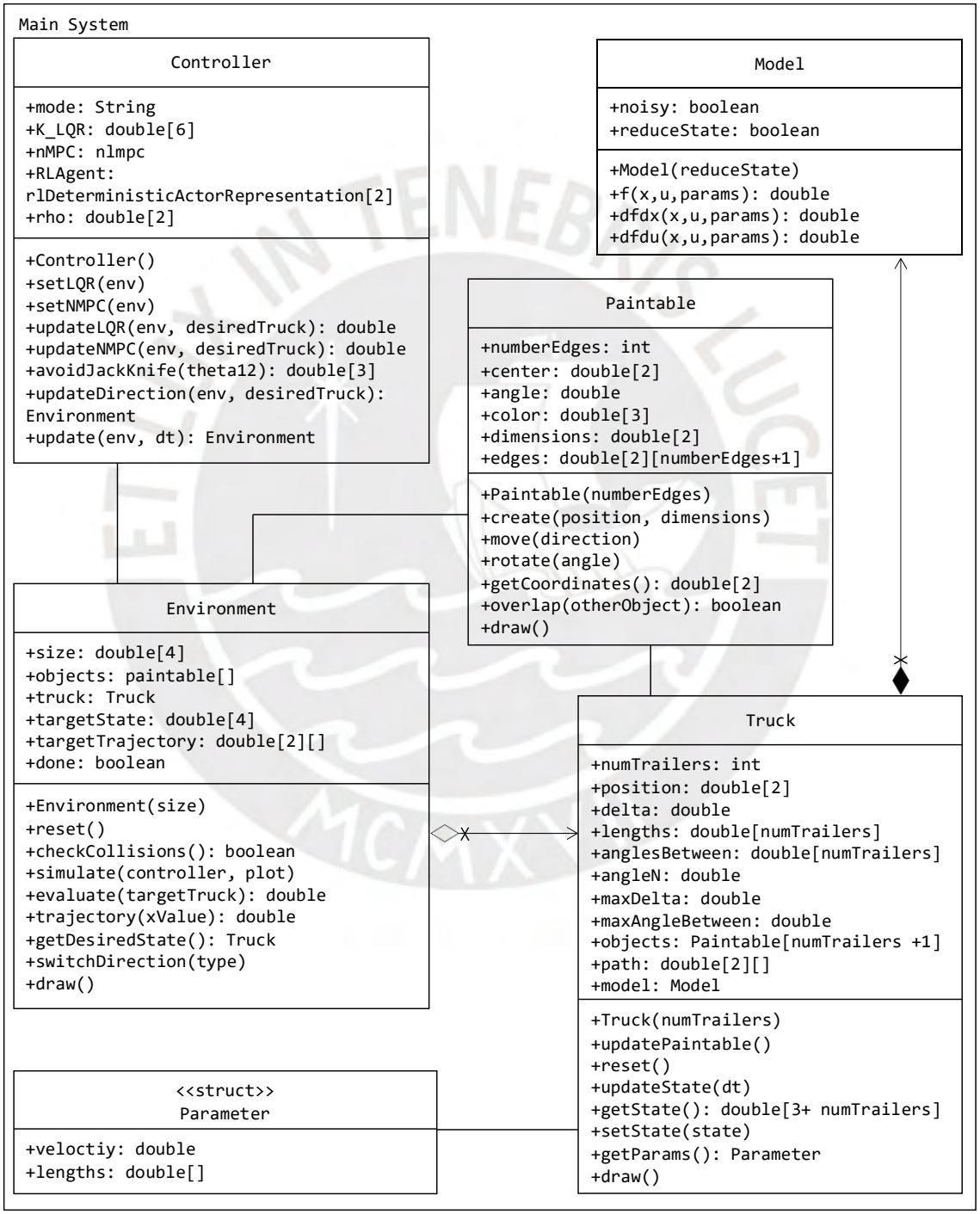


Figure A.14: Software architecture of the main system

The class `Truck` represents the entire truck that is controlled in this work. It consists of the tractor and an arbitrary number of trailers. To describe the truck completely, its dimensions, state and its mathematical model are needed. For the visualization, the truck has `Paintable` objects, that represent the rectangular shapes. The truck can use *updateState()* to obtain the resulting truck after the time *dt* with the control input *delta*. The *draw()* method shows the truck by using the *draw()* method of the `Paintable` objects.

Objects of the class `Paintable` are geometrical shapes that can be painted. The shapes have an arbitrary number of edges and an array to save the respective points with the first point being identical to the last to close the circumference. In addition, the center, dimensions, rotation angle and color describe every shape possible. Those attributes can be changed by the *create()*, *move()* and *rotate()* method. The *overlap()* method determines if there is an overlap between another object and the object itself. This is implemented using the SAT, see Section 4.3.

To compute new states based on Algorithm 1 proposed in Chapter 3, `Model` is used. It consists of three methods *f()*, *dfdx()* and *dfdu()* to obtain the respective mathematical values by providing the current state >, the control input *u* and the model parameters of the type `Parameter`. It also implements the features of a noisy output and it can reduce the state of the output by simply leaving out the first component. It is used inside the `Truck` class to compute the new state and during network training to provide the resulting behavior.

The structure `Parameter` is used to hold the velocity of the truck and all the lengths of the tractor and trailers. This is useful, because this set of parameters is used often in the model, truck and other methods, so it is easier to handle.

The `Controller` class implements the functionalities introduced in Chapter 4. It has attributes for the different controller and switching parameters. Aside of initializing the controllers, it updates the control variable with the chosen controller using the *update()* methods. Additionally, it uses the *updateDirection()* method to determine if a change of the direction is necessary. The actual direction change is executed by the `Environment` object. Finally, *avoidJackKnife()* applies the approach presented in Section 4.1.

An object of the class `Environment` represents a whole scenario, including objects, a target and a truck. The theory is presented in Section 3.6, but it also works as the overall structure to keep all relevant data in one place. This is why it features the *simulate()* method that is called from outside to start the simulation and why it is used to visualize the whole system with the *draw()* method. It has attributes to describe the chosen scenario by specifying its *size* and the *objects*. Plus, the *target state*, *target trajectory* or both are saved there. As stated before, it *simulates* the whole system, but also assigns costs with *evaluate()* to every situation, uses *checkCollisions()* and actually changes the driving direction with *switchDirection()*.

Other methods are used to create the scenarios and controllers, handle the simulation and analyze the results afterwards. The file *param* contains all parameters in a single place location.

The handling of the learning with the DBP algorithm is done with two classes `Network` and `Layer`. The `Network` class implements in neural network with all the *layers* and the training process using the *train()* method. It can do a forward pass with the *pass()* method and can be used in the production setting with the *useNet()* method. The layers itself are implemented as `Layer`. This class represents one single layer with all its *neurons*, the *weights* and the *activationFunction*. At this level, the forward pass is computed using the *activation()* function and the error backpropagation gets executed. Figure A.15 illustrates the architecture.

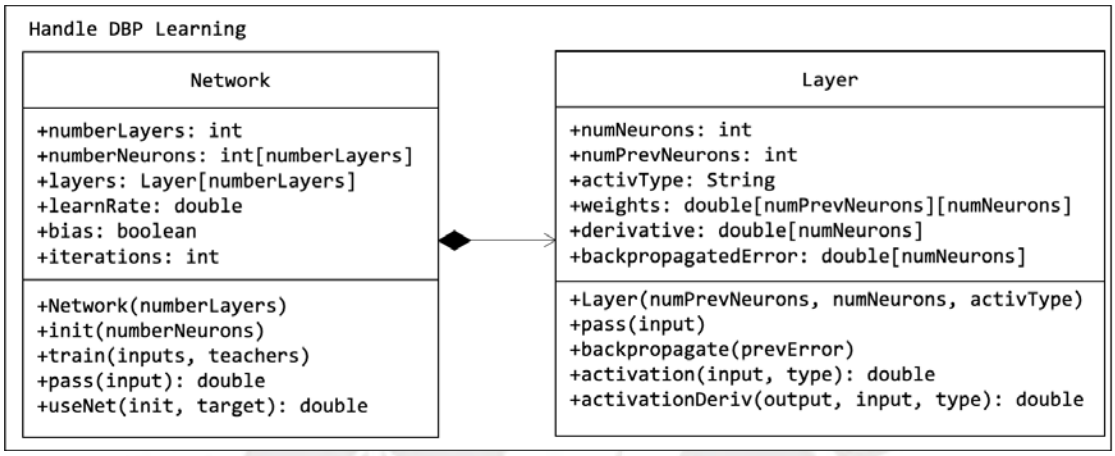


Figure A.15: Software architecture of the DBP learning

The training of the reinforcement learning agent is done using a MATLAB function. The `rlDDPGAgent` requires a custom environment with certain methods to do the training. This is why the `RLEnvironment` class was implemented. It is a combination of a simplified environment and support for the training process. For example, it holds the parameters for the reward and the simulation, like the parameter *kappa* or the *samplingTime*. When it is executed, the *reset()* method starts a new episode. Afterwards, the *step()* method is called continuously to simulate the behavior. Afterwards the actor and critic networks are updated using the `rlDDPGAgent`. For debugging purposes, the `RLEnvironment` can be visualized with a *plot()* method as well.

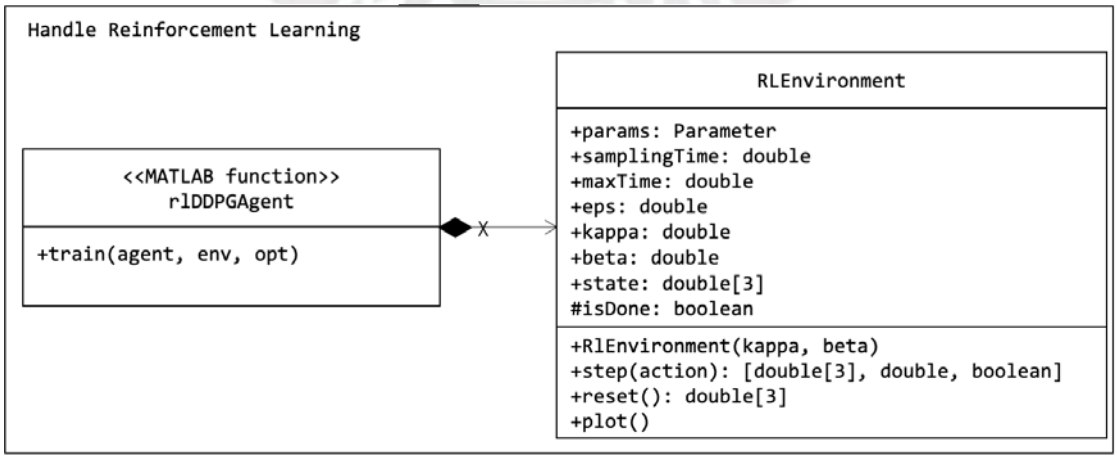


Figure A.16: Software architecture of the reinforcement learning

ET LUX IN TENEBRIS LUCET
MCMXVII

Bibliography

[1] SAE. *J3016*. [Online; accessed June 01, 2020]. 2018. URL: https://saemobilus.sae.org/content/j3016_201806.

[2] Clarissa Hawes. *Hefty Freight Demand Pushes Trailer Orders, Used Truck Prices Higher*. [Online; accessed July 07, 2020]. 2018. URL: https://www.trucks.com/2018/03/28/freight-demand-trailer-orders-used-truck-prices/.

[3] BMW. *BMW Group está haciendo que los robots de logística sean más rápidos e inteligentes*. [Online; accessed July 07, 2020]. 2020. URL: https://www.press.bmwgroup.com/latin-america-caribbean/article/detail/T0308406ES.

[4] Adalidda. *Robot-based transport eases and optimises SEAT factory workers' jobs and reduces production time by 25 percent*. [Online; accessed July 07, 2020]. 2018. URL: https://adalidda.com/posts/yxu53Fy4QvB4Cjgnc/robot- based- transport-eases-and-optimises-seat-factory.

[5] Nathan van de Wouw et al. "Active trailer steering for robotic tractor-trailer combinations". In: *IEEE 54th Annual Conference on Decision and Control (CDC)* (2015).

[6] DaCoTA Manual. *Combination Type - Truck*. [Online; accessed June 01, 2020]. 2020. URL: https://dacota-investigation-manual.eu/English/548.

[7] Maciej Michalek. "Tracking control strategy for the standard N-trailer mobile robot - geometrically motivated approach". In: *8th Workshop on Robot Motion and Control (RoMoCo)* (2011).

[8] Yevgen Sklyarenko, Frank Schreiber, and Walter Schumacher. "Maneuvering assistant for truck and trailer combinations with arbitrary trailer hitching". In: *IEEE International Conference on Mechatronics (ICM)* (2013).

[9] E. P. Ferreira and V. M. Miranda. "Development of Static Neural Networks as Full Predictors or Controllers for Multi-Articulated Mobile Robots in Backward Movements - New Models and Tools". In: *9th IEEE International Conference on Control and Automation (ICCA)* (2011).

[10] Daniel Reichegger. "Aufbau, Inbetriebnahme und Regelung eines rückwärtsfahrenden Gelenkzuges". In: *Masterthesis* (2015).

[11] E. P. Ferreira and V. M. Miranda. "Full Neural Predictors, with Fixed Time Horizon, for a Truck-Trailer-Trailer Prototype of a Multi-Articulated Robot, in Backward Movements - Singular Conditions and Critical Angles". In: *9th IEEE International Conference on Control and Automation (ICCA)* (2011).

[12] Jean-Paul Laumond. "Controllability of a multibody mobile robot". In: *IEEE Transactions on Robotics and Automation* (1994).

[13] Tomas Lozano-Perez. “Spatial Planning: A Configuration Space Approach”. In: *IEEE Transactions on Computers* (1983).

[14] Roland Stahn, Tobias Stark, and Andreas Stopp. “Laser Scanner-Based Navigation and Motion Planning for Truck-Trailer Combinations”. In: *IEEE/ASME international conference on advanced intelligent mechatronics* (2007).

[15] Robert S. Woodley and Levent Acar. “Neural Network Based Control for a Backward Maneuvering Trailer Truck”. In: *Proceedings of the 37th IEEE Conference on Decision and Control* (1998).

[16] Augie Widyotriatmo, Parsaulian Ishaya Siregar, and Yul Yunazwin Nazaruddin. “Line Following Control of an Autonomous Truck-Trailer”. In: *International Conference on Robotics, Biomimetics and Intelligent Computational Systems (Robionetics)* (2017).

[17] H. Choset et al. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005.

[18] Ruofan Kong. “Accurate Parking Planning of Tractor-Trailer-Trailer Mobile Robot”. In: *IEEE International Conference on Mechatronics and Automation* (2012).

[19] Ronald Uriol and Antonio Moran. “Mobile Robot Path Planning in Complex Environments Using Ant Colony Optimization Algorithm”. In: *3rd International Conference on Control, Automation and Robotics* (2017).

[20] Dieter Zöbel. “Trajectory Segmentation for the Autonomous Control of Backward Motion for Truck and Trailer”. In: *IEEE Transactions on Intelligent Transportation Systems* (2003).

[21] Xinxin Du and Kok Kiong Tan. “Autonomous Reverse Parking System Based on Robust Path Generation and Improved Sliding Mode Control”. In: *IEEE Transactions on Intelligent Transportation Systems* (2015).

[22] Luis Velasco Mellado, Antonio Morán Cárdenas, and Francisco Cuellar Córdova. “Optimal Control of a Mobile Robot Describing Minimum-Length Paths with Forward and Backward Motion”. In: *IEEE 4th International Conference on Advanced Robotics and Mechatronics (ICARM)* (2019).

[23] Guanrong Chen and Delin Zhang. “Back-Driving a Truck with Suboptimal Distance Trajectories: A Fuzzy Logic Control Approach”. In: *IEEE Transactions on Fuzzy Systems* (1997).

[24] Antonio Moran. “Autonomous path following of truck-trailer vehicles using linear-fuzzy control”. In: *3rd International Conference on Control, Automation and Robotics, ICCAR* (2017).

[25] Adam W. Divelbiss and John T. Wen. “A Path Space Approach to Nonholonomic Motion Planning in the Presence of Obstacles”. In: *IEEE Transactions on Robotics and Automation* (1997).

[26] Sepanta Sekhavat and Jean-Paul Laumond. “Topological Property for Collision-Free Nonholonomic Motion Planning: The Case of Sinusoidal Inputs for Chained Form Systems”. In: *IEEE Transactions on Robotics and Automation* (1998).

[27] M. Sharafi and A. zare. “Intelligent Parking Method for Trucks in Presence of Fixed and Moving Obstacles”. In: *International Conference on Information, Networking and Automation (ICINA)* (2010).

[28] T.R. Ren et al. “Controller Design of a Truck and Multiple Trailer System”. In: *Proceedings of the 2010 IEEE International Conference on Robotics and Biomimetics* (2010).

[29] Patrik Zips, Martin Böck, and Andreas Kugi. “An Optimisation-Based Path Planner for Truck-Trailer Systems with Driving Direction Changes”. In: *IEEE International Conference on Robotics and Automation (ICRA)* (2015).

[30] Robert Woodley and Levent Acar. “Autonomous Control of a Scale Model of a Trailer-Truck Using an Obstacle-Avoidance Path-Planning Hierarchy”. In: *Proceeding of the 2004 American Control Conference* (2004).

[31] Rizqi Ardhi, Augie Widyotriatmo, and Yul Y. Nazaruddin. “Backward Motion Path Following Control of Autonomous Truck-Trailer: Lyapunov Stability Approach”. In: *IEEE Conference on Control Technology and Applications (CCTA)* (2019).

[32] Claudio Altafini, Alberto Speranzon, and Bo Wahlberg. “A Feedback Control Scheme for Reversing a Truck and Trailer Vehicle”. In: *IEEE Transactions on Robotics and Automation* (2001).

[33] Jesús Morales et al. “Driver Assistance System for Backward Maneuvers in Passive Multi-trailer Vehicles”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems* (2012).

[34] Carlos Muñoz, María José Castilla, and Mario Fernandez-Fernández. “Pole Placement applied to drive a Truck and Trailer in backward. A didactic view”. In: *IEEE CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON)* (2019).

[35] John C. Doyle and Gunter Stein. “Multivariable Feedback Design: Concepts for a Classical Modern Synthesis”. In: *IEEE Transactions on Automatic Control* (1981).

[36] Andri Riid, Alar Leibak, and Ennu Rustern. “Fuzzy Backing Control of Truck and Two Trailers”. In: *IEEE International Conference on Computational Cybernetics* (2006).

[37] Jürgen Guldner, Vadim I. Utkin, and Jürgen Ackermann. “A Sliding Mode Control Approach to Automatic Car Steering”. In: *American Control Conference* (1994).

[38] Claudio Altafini and Alberto Speranzon. “Backward line tracking control of a radio-controlled truck and trailer”. In: *Proceedings of the 2001 IEEE International Conference on Robotics and Automation* (2001).

[39] G. Rigatos et al. “Nonlinear optimal control for autonomous navigation of a truck and trailer system”. In: *18th International Conference on Advanced Robotics (ICAR)* (2017).

[40] David Di Ruscio. “MODEL PREDICTIVE CONTROL AND IDENTIFICATION: A Linear State Space Model Approach”. In: *36th Conference on Decision and Control* (1997).

[41] Jay H. Lee, Manfred Morari, and Carlos E. Garcia. “State-Space Interpretation of Model Predictive Control”. In: *Automatica* (1994).

[42] Jean Paul Barreto Guerra. “Design of a Mobile Robot’s Control System for Obstacle Identification and Avoidance using Sensor Fusion and Model Predictive Control”. In: *Masterthesis* (2017).

[43] Yasumasa Fujisaki and Takeshi Narazaki. “Optimal Preview Control Based on Quadratic Performance Index”. In: *Proceedings of the 36th Conference on Decision and Control* (1997).

[44] Eduardo Bejar and Antonio Morán. “Reverse Parking a Car-Like Mobile Robot with Deep Reinforcement Learning and Preview Control”. In: *IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)* (2019).

[45] Eduardo Bejar and Antonio Morán. “A Preview Neuro-Fuzzy Controller Based on Deep Reinforcement Learning for Backing Up a Truck-Trailer Vehicle”. In: *IEEE Canadian Conference of Electrical and Computer Engineering (CCECE)* (2019).

[46] Rafika El Harabi, Saloua Belhaj Ali Naoui, and Mohamed Naceur Abdelkrim. “Fuzzy Control of a Mobile Robot with two Trailers”. In: *First International Conference on Renewable Energies and Vehicular Technology* (2012).

[47] Xiyang Yang, Jie Yuan, and Fusheng Yu. “Backing Up a Truck and Trailer Using Variable Universe Based Fuzzy Controller”. In: *Proceedings of the IEEE International Conference on Mechatronics and Automation* (2006).

[48] Antonio Moran Cardenas et al. “Autonomous Motion of Mobile Robot Using Fuzzy-Neural Networks”. In: *12th Mexican International Conference on Artificial Intelligence* (2013).

[49] Jianqiang Yi, N. Yubazaki, and K. Hirota. “Backing up control of truck-trailer system”. In: *10th IEEE International Conference on Fuzzy Systems* (2001).

[50] Guanrong Chen and Delin Zhang. “Backing Up A Truck-trailer With Suboptimal Distance Trajectories”. In: *Proceedings of IEEE 5th International Fuzzy Systems* (1996).

[51] Kazuo Tanaka and Manabu Sano. “A Robust Stabilization Problem of Fuzzy Control Systems and Its Application to Backing up Control of a Truck-Trailer”. In: *IEEE Transactions on Fuzzy Systems* (1994).

[52] Dieter Zöbel. “Trajectory segmentation for the Autonomous Control of Backward Motion for Truck and Trailer”. In: *The IEEE 5th International Conference on Intelligent Transportation Systems* (2002).

[53] Hung-Ching Lu and Ted Tao. “The CMAC based FLC and its application to rear-loading truck problems”. In: *The 12th IEEE International Conference on Fuzzy Systems* (2003).

[54] Ramón Parra-Loera and David .J. Corelis. “Expert system controller for backing-up a truck-trailer system in a constrained space”. In: *Proceedings of 1994 37th Midwest Symposium on Circuits and Systems* (1994).

[55] Stuart Russell and Peter Norvig. *Artificial Intelligence: A modern approach*. Pearson, 2010.

[56] Derrick Nguyen and Bernard Widrow. “The Truck Backer-Upper: An Example of Self-Learning in Neural Networks”. In: *Proceedings of International Joint Conference on Neural Networks IJCNN* (1989).

[57] Abdulla Ismail and Emadeddin A. G. Abu-Khousa. “A Comparative Study of Fuzzy Logic and Neural Network Control of the Truck Backer-Upper System”. In: *Proceedings of the 1996 IEEE International Symposium on Intelligent Control* (1996).

[58] K. Tanaka and K. Yoshioka. “Fuzzy trajectory control and GA-based obstacle avoidance of a truck with five trailers”. In: *IEEE International Conference on Systems, Man and Cybernetics. Intelligent Systems for the 21st Century* (1995).

[59] Hiroshi Kinjo et al. “Adaptive Genetic Algorithm Observer and its Application to a Trailer Truck Control System”. In: *SICE-ICASE International Joint Conference* (2006).

[60] H. Kinjo, Bingchen Wang, and T. Yamamoto. “Backward movement control of a trailer truck system using neuro-controllers evolved by genetic algorithm”. In: *26th Annual Conference of the IEEE Industrial Electronics Society, IECON* (2000).

[61] Timothy P. Lillicrap et al. “Continuous Control with Deep Reinforcement Learning”. In: *International Conference on Learning Representations (ICLR)* (2016).

[62] Eduardo Bejar and Antonio Morán. “Backing Up Control of a Self-Driving Truck-Trailer Vehicle with Deep Reinforcement Learning and Fuzzy Logic”. In: *IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)* (2018).

[63] H.-H. Nagel et al. “T3wT: Tracking Turning Trucks with Trailers”. In: *IEEE Workshop on Visual Surveillance* (1998).

[64] Kody Law, Abhishek Shukla, and Andrew Stuart. “Analysis of the 3DVAR Filter for the partially observed Lorenz 63 Model”. In: *Discrete and Continuous Dynamical Systems* (2014).

[65] Harry L. Trentelman, Anton A. Stoorvogel, and Malo Hautus. *Control theory for linear systems*. Springer-Verlag London, 2001, pages 211-236.

[66] J. Park and I. W. Sandberg. “Universal approximation using radial-basisfunction networks”. In: *Neural Computation* (1991), pages 246-257.

[67] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. MIT Press, 1998.

[68] Jorge Nocedal and S. Wright. *Numerical Optimization*. Springer-Verlag New York, 2006.

ET LUX IN TENEBRIS LUCET
MCMXVII

Statement

I hereby confirm that I have written the submitted thesis by myself, without using any sources other than those indicated. Appropriate credit has been given where reference has been made to the work of others. This thesis is part of the integrated international double degree program with the Technical University of Ilmenau and the Pontificia Universidad Católica del Perú and will therefore be submitted to both universities.

Iserlohn, January 9th, 2020 Benedikt Roder