Theory of Hybrid Systems Informatik 2
RWTH AACHEN UNIVERSITY

The present work was submitted to the LuFG Theory of Hybrid Systems

BACHELOR OF SCIENCE THESIS

APPLICATION OF FORMAL METHODS IN AUTONOMOUS VEHICLE CONTROL

Niklas Kotowski

Examiners:
Prof. Dr. Erika Ábrahám
apl. Prof. Dr. rer. nat. Thomas Noll

Additional Advisor:
Stefan Schupp

Aachen, July 15, 2019

Abstract

The current development and the increased usage of autonomously driving vehicles in real traffic applications and the corresponding obligation of the passenger's safety has resulted in an increased interest in the verification of the underlying controller systems.

In this work we present an overview of the current research status of formal verification of autonomous vehicle controllers to guarantee the safety of autonomous vehicles. We limit our focus on verification methods from hybrid systems safety verification, namely flowpipe-construction-based reachability analysis and its application in the field of autonomous vehicles. Initially, we present a comparison between suitable theoretical vehicle models, which differ in complexity and expressiveness. Based on this, we discuss various controller models ranging from different verified controller structures to an model predictive control (MPC) based trajectory planner for autonomously driving vehicles. We discuss how to model the presented controller structures as hybrid system to allow for their verification using flowpipe-construction-based reachability analysis.

In an experimental evaluation, various benchmarks and numerical experiments are used to analyze a prototypical implementation of elaborated and verified controller structures.

Erklärung

Hiermit versichere ich, dass ich die vorgelegte Arbeit selbstständig verfasst und noch nicht anderweitig zu Prüfungszwecken vorgelegt habe. Alle benutzten Quellen und Hilfsmittel sind angegeben, wörtliche und sinngemäße Zitate wurden als solche gekennzeichnet.

Niklas Kotowski
Aachen, den 15. Juli 2019

Acknowledgements

I would like to thank my advisor Stefan Schupp who took the time to discuss with me about varying questions and ideas. I would also like to thank my family and Jana for keeping me motivated and provided me with the support I needed in the progress of writing this thesis. Finally, I want to express my gratitude to my grandpa.

Contents

Chapter 1

Introduction

The current development and technological progress in autonomous vehicle control and its increased application have resulted in an increased demand on safety verification of such systems [HAS14]. Numerous vehicles are tested in simulations and real traffic scenarios with sensors recognizing the current environment of the operator's vehicle. These situations range from competitions in robotics to autonomous vehicles participating in actual traffic interactions [BIS09]. Naturally, as human life is at stake, autonomously driving vehicles have to fulfill the highest safety specifications in all traffic scenarios. Controllers of autonomous vehicles have to generate optimal inputs regarding driver's comfort while ensuring safety. Apart from offline path planning, autonomous vehicles have to react on changing traffic situations based on their sensor readings.

In this thesis we focus on using formal methods to analyze different controller types and the improvement of controller verification. Accordingly, we aim at presenting a structured overview into safety verification of autonomous vehicle controllers.

The continuous nature of a moving vehicle in combination with a digital controller which creates movement inputs for said vehicle result in a mixed discrete-continuous system. In the past decades, formal models for such hybrid systems combining discrete and continuous behavior have been in the focus of research. Therefore we are going to use hybrid systems as a formal specification to display controller models and the corresponding vehicle model. Hybrid systems allow us to model dynamic and discrete systems in one formal specification.

Intuitively, one can test the resulting computations to be safe in the current context by analyzing if the current trajectory enters a possible unsafe region. This procedure is however static and cannot cover all possible outcomes of the underlying system and trajectory generation. Therefore in the past decades the research shifted to formal methods which focus on verifying sets of trajectories.

In order to apply certain formal methods the system of interest has to be transformed into a formal model to allow it's verification. In the context of this thesis we decided to use hybrid automata as a formal specification to model the corresponding control system.

Further to decide which verification method fits our requirements, we compared certain approaches in Section 2.1.2. The analysis resulted in the decision to use a flowpipe-construction-based reachability analysis throughout this thesis, especially in the mentioned verification module (see Chapter 6).

In this paper we present and compare six different approaches towards safety verification of autonomous vehicle controllers.

The first presented approach is based on so called maneuver automata, in which finite sets of maneuvers are concatenated in verified trajectory planning. Secondly, an approach to formally verify the structure of a path planner with the help of linear temporal logic (LTL) is displayed. Moreover, a maneuver template approach is presented, which is used as a tool in a motion planner to improve the speed of the path planner. In a short section the idea of a verification method relying on a proof assistant is shown. Additionally in another approach, an electric field concept is used to model the vehicle as an electron and all other traffic interactions as potentials repelling the vehicle [RS94]. The electric field model is a theoretical concept used to plan an optimal trajectory through changing environment. This is realized by a global planner which constructs an initial trajectory which is later optimized by a local planner using the electric field approach. Finally, the procedure of a model predictive control (MPC) in trajectory planning for autonomous vehicles is presented. The MPC-based controller solves the task to generate optimal inputs regarding efficiency and comfort while ensuring safety. In this paper we implement a light weight MPC-based controller for trajectory planning. We transform said controller into a hybrid automaton to identify challenges and open problems which need to be tackled by the research community in order to be able to verify MPC-based trajectory planning. Additionally, a verification module is presented to cope with the challenge of verifying inputs of a MPC-based controller. The verification module evaluates the inputs generated by a certain controller with the help of flowpipe-construction-based reachability analysis. Moreover, a hybrid automaton is implemented to generate verified input values for a trajectory described by a set of concatenated maneuvers. We evaluate our methods on various numerical experiments.

This thesis is structured as follows:

To work on the safety aspect and verification of controllers, we begin with a fundamental overview of the current research in autonomous vehicle control. First of all, in Chapter 2 preliminaries including definitions about geometric sets and the in this thesis used algorithmic methods are depicted. Chapter 3 introduces certain research papers we used in our thesis. Afterwards, in Chapter 4 we compare common theoretical vehicle models in complexity and expressiveness [Alt17]. Then in Chapter 5 we present different verification methods, together with common path planners and the earlier mentioned MPC-based trajectory planner. Afterwards in Section 5.7, we depict a hybrid automaton, which constructs inputs for verified trajectories. Finally, the verification issues regarding the MPC-based controller are summarized in Chapter 6, together with a prototypical implementation of a verification module. Finally in Chapter 7 we evaluate our prototypical implementation of some of the presented controllers.

Chapter 2

Preliminaries

In this chapter we present the theoretical background and concepts required for the further understanding of this thesis.

2.1 Hybrid Systems

The variables of dynamic systems change continuously, according to differential equations. Exemplary systems are temperature, physical quantities of moving objects or other systems with continuous variable derivations.

On the contrary, discrete systems are specifications of systems with discrete changes. These changes are initiated by specified rules of the system or sensors or sensors examining certain environment conditions [ALU95].

Hybrid systems are a combination of both systems, merging the continuous variable evolution with discrete changes. Examples for hybrid systems are systems digitally controlling the in- and outflow of water tanks, thermostats and general systems with dynamic derivations and determined discrete changes triggered by certain events. Further descriptions about the mentioned systems are found in [ALU95, Hen00].

In the context of autonomous vehicle control, the combination of discrete changes initiated by the controller and the continuous evolution of the vehicle's state can be represented by a hybrid system. Therefore we decided to use hybrid systems as a formal specification for the closed loop between controller and vehicle.

2.1.1 Hybrid Automata

In this thesis we investigate the application of hybrid systems safety verification in the field of autonomous vehicle control. In order to be able to apply formal methods for safety verification, the system under interest needs to be modeled. A commonly used model for hybrid systems are hybrid automata [Hen00]. A hybrid automaton combines discrete state changes with the dynamic behavior of a system in one closed automaton model. A hybrid automaton is an extension of the well-known model of labeled transition systems (LTS). Similar to a LTS it is built from a finite set of locations (Loc) connected by guarded discrete transitions. A finite set of variables (Var) is used to describe the system state.

In contrast to LTS, hybrid automata allow to model the passage of time. The system variables are updated according to location-specific flow (Flow) as long as control stays in a location. The control can stay as long in the location as the invariant (Inv) is satisfied by the system's variable valuation. The discrete transitions connect the locations of the automaton and are called edges (Edge), they are further specified by a guard set and a reset function. The set of initial states is given as a tuple of location and variable valuation (Init) [Hen00].

The following formal definition states a simplified version, omitting factors which are only important for a parallel composition of two hybrid systems [SAC+15].

Definition 2.1: Hybrid Automaton [SAC+15]

$$\mathcal{H} = (Loc, Var, Flow, Inv, Edge, Init)$$

Loc : A finite set of locations $Loc = \{l_1, \ldots, l_m\}$.

Var : A finite ordered set X of variables *Var* $= \{x_1, \ldots, x_n\}$, in which a valuation of a variable x_i is denoted by v_i. Additionally, doted variables $\{\dot{x_1}, \ldots, \dot{x_n}\}$ specify the derivatives of the corresponding variable, while primed variables $\{x'_1, \ldots, x'_n\}$ specify variables after discrete changes through reset functions.

$Flow$: Each location has a defined flow, $Flow : Loc \to Pred_{Var \cup Var'}$.

Inv : Each location has a defined invariant, $Inv : Loc \to Pred_{Var}$.

$Edge$: The *Edge* set is a subset of $Loc \times Pred_{Var} \times Pred_{Var \cup Var'} \times Loc$ defining a finite set of transitions or jumps.
A $jump$ is a tuple $(l_1, g, r, l_2) \in Edge$, where l_1 is the source location, l_2 the target location, g specifies the guard set and r defines the reset function.

$Init$: A set of tuples defining the initial locations and their variable valuation $(l_0, v_0) \in Init$.

The set $Pred_X$ describes all predicates with free variables in X.

In addition to the formal definition, a basic automaton describing a thermostat is shown in Figure 2.1 to display an exemplary graphical representation of a hybrid automaton. The thermostat has two locations, one to describe the active state and one to describe the case the thermostat is off. The automaton consists only of one variable x, depicting the current temperature. The guard sets constrain the temperature to dynamically switch the system on or off. In case the temperate exceeds 21, the thermostat is switched off and in case the temperate is below 19, the thermostat is switched on. Additionally, Figure 2.1 illustrates the flow of the temperature variable in each location is. The initial state is the off mode with an initial temperature of $20°$.

After defining the syntax of a hybrid system, we have to declare the discrete and continuous semantics in detail. The discrete semantics are defined in Rule (2.1), where a discrete transition step is denoted by $\xrightarrow{e}$ [SAC+15].

$$\frac{e = (l, g, r, l') \in Edge \;\; v, v' \in \mathbb{R}^d, \;\; v \models g, \;\; v, v' \models r, \;\; v' \models Inv(l')}{(l, v) \xrightarrow{e} (l', v')} \tag{2.1}$$

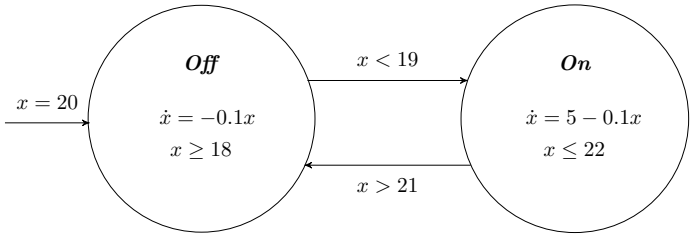


Figure 2.1: Thermostat automaton [Hen00].

The discrete Rule (2.1) states the discrete change of the variable valuation by taking a jump, while ensuring that the according to the reset function modified variables satisfy the invariant of the target location.

Furthermore, the continuous time semantics are specified by Rule (2.2), in which $\xrightarrow{\delta}$ defines a time step relation [SAC+15].

$$\frac{\begin{array}{c} l \in Loc \quad v, v' \in \mathbb{R}^d \\ f : [0,\tau] \to \mathbb{R}^d \;\; df/dt = \dot{f} : (0,\tau) \to \mathbb{R}^d \;\; f(0) = v \;\; f(\tau) = v' \\ \forall \epsilon \in (0,\tau).\; f(\epsilon), \dot{f}(\epsilon) \models Flow(l) \;\; \forall \epsilon \in [0,\tau].\; f(\epsilon) \models Inv(l) \end{array}}{(l,v) \xrightarrow{\delta} (l,v')} \tag{2.2}$$

Rule (2.2) specifies the semantics of the variable evolution induced by the flow in the current location of the hybrid automaton. The time Rule (2.2) ensures that the evolution of variable valuations in a defined time duration does not violate the invariant (Inv) of the current location.

The main purpose of transforming a system description into a hybrid automaton is to have a formal model of the system. Further this model can be verified to prove the safety of the underlying system. The verification is required to ensure safety, before a system can be instated in real applications. Therefore it has to be checked, if a system can reach a state containing a risk. This can be analyzed by a flowpipe-construction-based reachability analysis which is presented in the following section.

2.1.2 Reachability Analysis

Reachability analysis (RA) for hybrid systems is used to analyze how a system evolves in a defined time duration and which states it can possibly reach. We verify the safety of a system by means of RA. RA is used to determine, which states are reachable in a system starting from a set of initial states. Afterwards a verification step examines if at a certain time the set of reachable states intersects with set of determined bad states. Bad states represent malicious behavior which is usually given as a safety specification. Owing to the fact that the reachability problem for hybrid systems is in general undecidable, we compute an over-approximation of the set of reachable states.

If the intersection between the set of reachable states of the system and the bad state's set is empty the system can be declared as safe. If, however, the intersection

between the set of reachable states of the system and the set of bad states is not empty, the system is potentially unsafe. Due to the over-approximation of the set of reachable states we can only declare the system to be potentially unsafe [Gir05].

We decided to use the method of a flowpipe-construction based reachability analysis in this thesis. In addition to the flowpipe-construction-based reachability analysis, there are various other methods to verify hybrid systems.

Additionally, hybrid systems can be verified by theorem proving. Certain proof assistants as Isabelle/HOL, PVS or Coq make use of first-order-logic or arithmetic formulas to state properties of an underlying system. Further to realize a verification of hybrid systems, the specified formulas can be used to derive proofs for safety or other properties of the system. The described proofs are realized automatically or semi-automatically by user interaction directing the proofs.

Furthermore, a hybrid system can be verified by bounded model checking, which often relies on the usage of satisfiability checking. The reachability problem has to be transformed into mixed integer-real-arithmetic formulas defining the set of reachable states for the next discrete time step [SAC+15], which are solved by a satisfiability-modulo-theories (SMT) solver. A tool concerning this algorithmic verification method is found in [KGCC15].

The Algorithm 1 depicting the general flowpipe-construction-based reachability algorithm has to be further explained. The algorithm gets as input a hybrid system model H and has the purpose to compute the set of reachable states R for the given model H (we assume the model to be a hybrid automaton).

The set R_{new}, contains all the states which have to be processed by the algorithm and is initially set to the initial state of H (Init). An additional set R depicts all the states currently reachable by H. In a loop executed until all states of R_{new} have been processed, the flowpipe for a selected state set taken from R_{new} is computed with the function computeFlowPipe, considering the location-specific flow. In a second step the function computeJumpSuccessor computes all possible jumps that can be taken, realized by examining which guard sets are satisfied at the current location and with the current variable valuation. The obtained states are added to the set R and R_{new}. As mentioned before Algorithm 1 terminates if the set R_{new} is empty and all state sets R_{new} have been processed. An additional $termination_{cond}$ is introduced to realize the jump and time-bounds of the RA. In detail, the algorithm computes an over-approximated set of reachable states for a bounded time duration and a bounded quantity of discrete jumps, which is realized by the $termination_{cond}$.

The following paragraph explains the flowpipe-construction-based reachability analysis in detail (Algorithm 1 displays the general procedure).

We illustrate the method to compute a flowpipe-construction-based reachability analysis for linear hybrid systems, in case of non-linear hybrid systems we require taylor models and certain approach based on them, further described in [CAS12].

To begin with the further explanation of the flowpipe-construction-based reachability analysis, we define the flow (Flow) in the current location of the automaton by a system of linear ODEs (Equation (2.3)):

$$\dot{x} = A \cdot x(t) \tag{2.3}$$

In case that the underlying system is not autonomous, the flow of the system is defined by:

$$\dot{x} = A \cdot x(t) + B \cdot u(t) \tag{2.4}$$

Data: Hybrid system H
Result: Set of reachable states R
(1) $R \coloneqq Init$;
(2) $R_{new} \coloneqq R$;
(3) **while** $R_{new} \neq \emptyset \wedge \neg termination_{cond}$ **do**
(4) | $let\ stateset \in R_{new}$;
(5) | $R_{new} \coloneqq R_{new} \backslash \{stateset\}$;
(6) | $R' \coloneqq computeFlowPipe(stateset)$;
(7) | $computeJumpSuccessors(R')$;
(8) **end**
(9) **return** R;

Algorithm 1: General algorithm for flowpipe-construction-based reachability analysis [Gir05].

The function $u(t)$, defines certain inputs given to the system.

The set of reachable states at a certain time t is defined by the matrix exponential multiplied with the initial state set: $x(t) = e^{tA}x_0$. The used matrix A is specified by the activity (Flow) in the current location.

Further to divide the flowpipe-computation in discrete time steps we introduce a time step size δ. The equation $x(t) = e^{tA}x_0$ allows it to compute the set of reachable states after a specified time δ. To further consider the activities during the time interval $[0, \delta]$, we calculate the error coefficient between the actual trajectory and the convex hull of the initial set and the at time δ calculated set. The computed factor called bloating coefficient is used to over-approximate the set at time δ. Afterwards, the convex hull between initial and described set is computed to define the initial flowpipe segment Ω_0 [Gir04, Gir05].

After computing the over-approximated initial flowpipe segment, consecutive time steps can be computed by a linear transformation on the set Ω_0. Reasoned by the fact that e^{tA} is the same matrix for a constant t, we can formulate this as a recurrence equation as described in Equation (2.5) [Gir05].

$$\Omega_{i+1} = e^{\delta A}\Omega_i \tag{2.5}$$

The equation allows us to compute the set of reachable states for the next discrete time steps $i \cdot \delta, i \geq 0$ with a certain linear transformation on Equation (2.5). Consequently, we can compute the flowpipe for a given time interval $[i \cdot \delta, (i+1) \cdot \delta]$ which will be relevant in later verification procedures. In case of a non autonomous system each flowpipe segment has to be over-approximated by a certain bloating coefficient.

Finally, in order to verify the underlying system with the described method, the computed set of reachable states and certain as unsafe defined sets are checked for an intersection. In case that the intersection between both sets is empty, the system is verified to be safe. However, if the intersection is non-empty we are unable to state anything about the system's status reasoned by the over-approximation used while computing the set of reachable states.

Figure 2.2 depicts a plot of the over-approximated flowpipe of the set of reachable states of the automaton in Figure A.1 in the Appendix. The figure displays the initial set in the left bottom corner and the two flowpipes connected by a jump transition computed over a time duration of 5s.

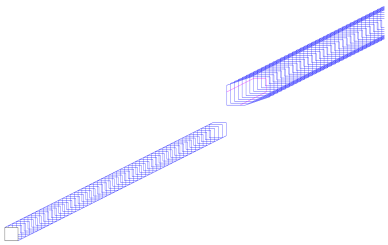


Figure 2.2: Flowpipe with jump.

The initial state set is centered around an initial variable state $[x_0, \ldots, x_{n-1}]$ and expanded at both sides by ϵ, formally depicted below:

$$x_0, \ldots, x_{n-1} \in [x_0 - \epsilon, x_0 + \epsilon], \ldots, [x_{n-1} - \epsilon, \ldots, x_{n-1} + \epsilon].$$

Figure 2.2 is computed with $\epsilon = 0.07$. It should be noted that the flowpipe is displayed by a set of zonotopes (see Definition 2.5).

Owing to the fact that the flowpipe construction-based reachability analysis highly depends on the used state set representation, we define the geometric representations (see Section 2.2) which we use in later implementation methods to represent the flowpipe's state sets (see Sections 5.7 and 6.1). In later implementations we use zonotopes as the state set representation of computed flowpipes. In addition to zonotopes, there are several other state set representations, which can be used in a flowpipe construction. We work with zonotopes because they are implemented in the toolbox Cora used in implementations of this thesis [AK18].

The choice of a state set representation is crucial for the analysis outcome and usually a trade-off between precision and running time [SK03].

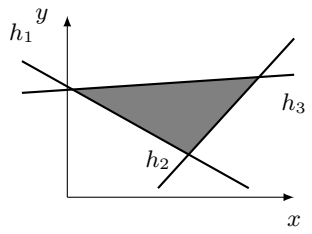
2.2 Geometric Set Representations

This section explains the geometric state set representations we used in implementations of this work. Reasoned by the fact that we decided to use hybrid systems as a formal specification of the closed loop between controller and vehicle model, a special focus lies on depicting the geometric representations used to define hybrid automata and the flowpipes constructed by the corresponding reachability analysis.

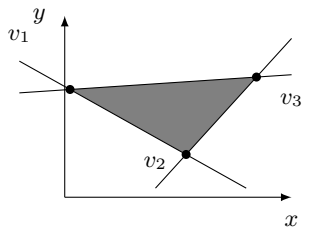
2.2.1 Half-space

Intuitively, a half-space describes the partition of the d-dimensional real vector space along a hyperplane. A half-space is defined by an inequality constraint grouping all valuations that fulfill the inequality condition into one set.

Throughout this thesis half-spaces are used in Sections 5.7 and 6.1 to constrain the guard sets of the implemented hybrid automata.



(a) H-representation of a polytope.



(b) V-representation of a polytope.

Figure 2.3: Convex polytope representations [Zie12].

Definition 2.2: Half-space [Zie12]

$$h = \left\{x \in \mathbb{R}^d \mid cx \leq d\right\} \text{ where } c \in \mathbb{R}^d \text{ and } d \in \mathbb{R}.$$

2.2.2 Convex Polytope

A convex polytope is a geometric representation of a set of points, depicting a bounded area of values. An example polytope is depicted by two separate representations in Figures 2.3a and 2.3b. We present two possibilities to represent a convex polytope. Either it can be displayed by a set of half-spaces or by a set of vertices describing the convex hull of the region defined by the polytope. We begin with the H-representation.

Definition 2.3: H-representation [Zie12]

A polytope is represented by a set of half-spaces:

$$P_H = \bigcap_{i=1}^{n} h_i, \text{ where } h_i = \left\{x \in \mathbb{R}^d \mid c_i x \leq d_i\right\} \text{ with } c_i \in \mathbb{R}^d, d_i \in \mathbb{R}.$$

Another way to define a convex polytope is with the V-representation. The V-representation describes a compact convex set by the convex hull of a finite set of vertices.

Definition 2.4: V-representation [Zie12]

A polytope is displayed by the convex hull of a finite set of points V:

$$P_V = conv(V), \quad V = \{v_0, \ldots, v_{n-1}\}, v_i \in \mathbb{R}^d.$$

The different options to represent a convex polytope have an impact on the time complexity of the operations used on the sets. In the hybrid automaton in Section 5.7 and Section 6.1, we use all displayed set representations. In the implementation of the hybrid automaton (see Section 5.7), we use the H-representation of a polytope to define the invariants of the location's and the guard sets of the automaton. However, in various methods of the toolbox Cora, executed in the reachability analysis, the

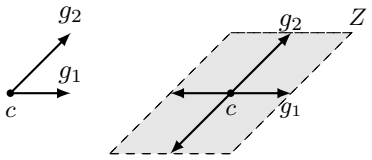


Figure 2.4: Example of a zonotope with two generators [Gir05].

polytope representations are varied [AK18]. This transformation process is done because depending on the current operation used on the sets a specific representation is advantageous.

2.2.3 Zonotope

In this thesis we use zonotopes (see Figure 2.4) as state set representations of computed flowpipes (see Figure 2.2). A zonotope consists of a center vector c and a set of generator vectors $g_0, \ldots, g_{k-1}$. A zonotope is depicted by the Minkowski sum of line segments, defined by the set of generator vectors and the coefficients $\delta \in [-1, 1]$.

Definition 2.5: Zonotope [Gir05]

For $c, g_0, \ldots, g_{k-1} \in \mathbb{R}^n$ with $k \in \mathbb{N}$

$$Z = \left\{ x \in \mathbb{R}^n \;\middle|\; x = c + \sum_{i=0}^{k-1} \delta_i \cdot g_i, -1 \leq \delta_i \leq 1 \right\}.$$

Finally, we conclude about the presented geometric set representations. The first presented sets are convex polytopes and half-spaces. They are used in the definition and implementation of hybrid automata, including the guard sets and invariants of the automatons' locations.

Secondly, to cover the state set representations of the flowpipe computation, we have to decide on one option to model the throughout this thesis computed flowpipes. In addition to the convex polytopes, which are as well suited to represent state sets of flowpipes, zonotopes are defined. As mentioned before, throughout this thesis we use zonotopes to represent the state sets of a flowpipe. Further information about the state sets and the computation time of the four basic operations (union, intersection, Minkowski sum and linear transformation) on the state sets can be looked up in [LG09].

2.3 Optimization Problem

In a later chapter about the MPC-based vehicle controller, inputs for a vehicle model are computed with the help of an optimization problem. In the following, we give a

short introduction on optimization problems.

In general, optimization problems are used to minimize or maximize a given function while considering various constraints. In linear optimization the cost function which is to be optimized with respect to a finite set of constraints on the variables are given as linear term respectively as linear constraints. Linear optimization aims at finding a maximizing variable assignment for the provided cost function, which satisfies constraints specifying the linear problem.

For $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$, the definition below defines a general linear programming problem:

$$\begin{aligned} \text{maximize} \quad & c^\intercal x \\ \text{s.t.} \quad & Ax \leq b, \\ & x \geq 0, \end{aligned}$$

where x is an n-dimensional vector to be modified to maximize the objective function $c^\intercal x$ [BT97].

To minimize the objective function, instead of maximizing it, it has to be negated. Moreover, instead of computing a linear problem a non-linear optimization problem is defined by the condition that the objective function as well as the constraints can be potentially non-linear. A variation of a non-linear optimization problem named quadratic programming (QP), allowing only a quadratic objective function with a linear constraint set is specified in Equation (2.6) [AFV95].

$$\begin{aligned} \min_x \quad & Q(x) = c^\intercal x + \frac{1}{2} x^\intercal D x \\ \text{s.t.} \quad & Ax \leq b, \\ & x \geq 0, \\ & \text{where } x^n, c^n, b^m, A^{m \times n} \text{ and } D^{n \times n}. \end{aligned} \tag{2.6}$$

Besides the presented structure, there are many variations of optimization problems. In the later explained MPC controller, we use a quadratic programming problem, which is solved in Matlab with the QP-Solver from the toolbox Yalmip [Löf04].

Chapter 3

Related Work

An early publication of Y.K Hwang and N. Ahjua laid the foundation of a theoretical approach to controlling vehicles in a dynamic environment. The so called electric field model uses an electron in a potential field to represent a vehicle in a traffic situation [HA92]. Later, the concept of potential fields has been developed and implemented to optimize the results of path planners [VTM00]. A similar procedure is used in current MPC-based controller models. In the same manner as in the approach presented in [VTM00], a model predictive control based algorithm splits the generation of control inputs in a global part building a reference trajectory and a local part to optimize the constructed path. As MPC reference the approach [LT18] is regarded and modified to fit the intended research aspects of this thesis.

Moreover, to cope with various situations and vehicle types, certain vehicle models have been researched and published. The vehicle models described in [Alt17] are used throughout this thesis in closed controller vehicle environments. The vehicle models are in detail presented in Chapter 4. In combination with a developed controller approach and a decisive vehicle model a realistic trajectory can be computed and evaluated in traffic situations.

Finally, the focus of this thesis shifts to the verification of controller models to ensure safety in autonomous vehicle control. Coupled with the idea to build a dynamic controller structure, different approaches have been published regarding the verification of a controller model. Related to this thesis are especially the research papers from Matthias Althoff [HAS14, MLA17, RISA18], since they cover distinct approaches verifying controller models in autonomous vehicle control.

Chapter 4

Vehicle Models

Before presenting the current research overview of controller models used in autonomous vehicle control, we present current vehicle modeling approaches increasing in complexity to depict the different options available to represent a trajectory with a theoretical model. Further, to verify controllers in autonomous vehicle control, we have to define certain vehicle models to represent a realistic vehicle behavior.

According to the considered scenario, the vehicle representation can be adjusted to fit the contexts' needs. Simplified scenarios with basic controlling approaches can be simulated and tested in combination with reduced vehicle models which solely include basic parameters, whereas benchmarks together with advanced controller models and complex traffic scenarios require a detailed model to depict a realistic traffic behavior. Further, in critical scenarios and in real traffic applications of autonomous vehicles, specific high-order models as the bicycle or multi-body model are advised to be implemented to allow the representation of a realistic trajectory and thus a safe verification [Alt17]. In the context of this thesis and considering the purpose of verifying an autonomous vehicle controller, the kinematic-single-track (KST) model has shown to suit our requirements. Therefore the KST is used as vehicle model in implementations in this thesis.

4.1 Point-Mass Model

The point-mass model (PM), as the name implies, consists of a single point in the coordinate space and a velocity in x- and y-direction. Further, the direction of the vehicle is implicitly given by the combination of the velocity values. Moreover, an acceleration of the vehicle and direction changes are realized by an increased derivative value a_v and respectively a variation of the relation between the two velocity values. The point-mass model is the most simplified representation of a moving object presented in this thesis [Alt17].

$$p\vec{m} = \begin{pmatrix} x \\ y \\ v_x \\ v_y \end{pmatrix} \qquad p\dot{\vec{m}} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{v_x} \\ \dot{v_y} \end{pmatrix} = \begin{pmatrix} v_x \\ v_y \\ a_{v_x} \\ a_{v_y} \end{pmatrix}$$

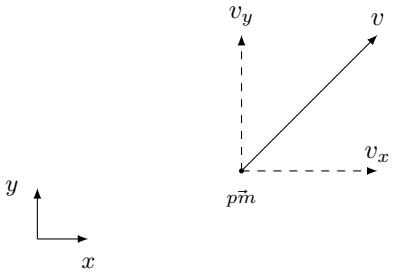


Figure 4.1: Point-mass model.

A graphical representation of the PM is displayed in Figure 4.1.

The variables x and y change according to the linear equations $\dot{x} = v_x$ and $\dot{y} = v_y$, while the velocity dynamics are defined by the constant acceleration values a_{v_x} and a_{v_y} in x and y direction [Alt17].
The point-mass model is of limited usability because it only permits to track the position of a vehicle, but ignores its heading angle. Furthermore, it is difficult to integrate more complex driving dynamics into this vehicle model.

In this thesis, the PM is presented to complete the overview of theoretical vehicle models. Moreover, the differences in the resulting state sets of a flowpipe-construction-based reachability analysis are shown in a comparison (Figure 4.5) with the kinematic single-track model [Alt17]. The made comparison illustrated problems regarding the expressiveness of the PM.

4.2 Kinematic Single-Track Model

In this section, a more detailed vehicle representation called kinematic single-track model is presented. In order to increase the precision of the vehicle representation, the KST models the driving direction by an additional variable depicting a heading angle ϕ (see Figure 4.2). Hence, the dynamics of the vehicle are non-linear enabling a representation of a curved trajectory. Moreover, the consideration of the heading angle smoothes the vehicles trajectory and enables a more realistic computation of the set of reachable states and its safety verification. Owing to the fact that the KST models a heading angle the requirements of the representation are increased, while the resulting trajectory becomes more decisive. We use the kinematic single-track model in Chapter 5 and in the verification module in Chapter 6.

Important to note here is that the in this thesis presented KST is simplified in comparison to the original derivation (see [Raj11]). This is done due to the reason that the KST is used in certain implementation parts, in which we rather focus on verifying our controller model than depicting the most realistic vehicle model [Raj11].

$$k\vec{s}t = \begin{pmatrix} x & y & v & \phi \end{pmatrix}^{\mathsf{T}}$$

The dynamics of this model are depicted in Equation (4.1). In contrast to the PM, the dynamics of the KST require trigonometric functions to display curved trajectories realized by the heading angle ϕ. Furthermore, the velocity and the heading angle change according to the acceleration a and angular velocity ϕ_v.

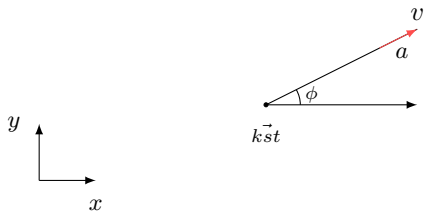


Figure 4.2: Kinematic single-track-model.

The two variables c_a and c_{ϕ_v} defining the velocity and heading angle dynamics are generated in the controller environment and proceeded as control inputs to the vehicle model specifying the derivations of v and ϕ [Raj11].

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{v} \\ \dot{\phi} \end{pmatrix} = \begin{pmatrix} v \cdot \cos(\phi) \\ v \cdot \sin(\phi) \\ c_a \\ c_{\phi_v} \end{pmatrix} \tag{4.1}$$

4.3 Bicycle Model

The bicycle model (BM) is the most complex vehicle representation discussed in detail, in this thesis. Dependent on the specific implementation, different relevant aspects regarding a physical vehicle are taken into account. In detail, the BM considers the vehicle length, width and mass together with the moment of inertia, the cornering stiffness and a tire specific friction coefficient. Consequently, the physical vehicle body is modeled and can be varied according to the current vehicle type. Hence, the verification of the trajectory can be done in a precise and realistic manner. Figure 4.3 displays a graphical representation of the bicycle model [Alt17].

In previous vehicle models the physical body is a single point which is over-approximated in collision checks to ensure safety. Since the width and length of the vehicle type are considered, the bicycle model enables a more decisive comparison of occupancy sets. As the name presumes, the vehicle has only two tires. The model combines the tires at each axis into one tire, resulting in the bicycle-like shape. Additionally, the BM adds certain state variables to represent a realistic behavior of the vehicle. Instead of a single heading angle as in the KST, the BM divides the driving direction and lane follow property in three different variables. First of all, a steering angle at the front axis is modeled by δ, secondly the yaw angle ψ and slip angle β at the vehicles center are considered to improve a realistic behavior of the vehicle in curved trajectories. The previously listed vehicle parameters, including the properties of the physical vehicle body, the moment of inertia for the entire mass and the friction coefficient, influence the derivation of the yaw rate $\dot{\psi}$ and slip angle β to get a realistic and smooth trajectory output [Raj11].

$$\vec{bm} = \begin{pmatrix} x & y & v & \delta & \psi & \dot{\psi} & \beta \end{pmatrix}^\intercal$$

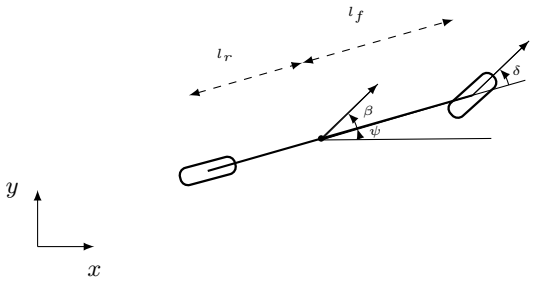


Figure 4.3: Bicycle model [Alt17].

4.3.1 Dynamics of the Bicycle Model

The dynamics for position, velocity and steering angle of the vehicle behave according to defined control inputs and the deterministic behavior of the vehicle dynamics. Conversely, the derivatives of yaw rate and slip angle are computed with a complex formulas considering individual vehicle characteristics. All parameters and vehicle coefficients used in later simulations, are stated in Table 4.1.

Further the position of the vehicle, modeled by x and y, evolves as in the kinematic single-track model with the difference that the angle modeling the driving direction is the sum of yaw angle and slip angle. Additionally, the angular velocity is defined by the control input u_1 (c_{ϕ_v}) taking upper and lower limit of the steering angle into account. In a similar way u_2 (c_a) determines the acceleration, while verifying that no vehicle specification is violated. The yaw rate and slip angle change according to a combination of system dependent coefficients mentioned in a later vehicle specific data sheet [Raj11].

The domain of the variables u_1 and u_2 which represent the acceleration and angular velocity of the car is bounded by the terms h_1 and h_2, where $\underline{\delta}, \underline{v}$ defines the minimum steering angle and velocity and respectively $\overline{\delta}, \overline{v}$ specifies the maximum steering angle and velocity of the specified vehicle type.

$$h_1 = (\delta \leq \underline{\delta} \wedge u_1 \leq 0) \vee \left(\delta \geq \overline{\delta} \wedge u_1 \geq 0\right)$$
$$h_2 = (v \leq \underline{v} \wedge u_2 \leq 0) \vee (v \geq \overline{v} \wedge u_2 \geq 0)$$

The equations h_1 and h_2 describe the vehicle steering and velocity constraints used in the case definition of their respective derivatives (Equation (4.2), Equation (4.3)). The dynamics of the bicycle model are formally defined below:

$$\dot{x} = v \cdot \cos(\psi + \beta),$$

$$\dot{y} = v \cdot \sin(\psi + \beta),$$

$$\dot{\delta} = \begin{cases} 0 & \text{for} \quad h_1, \\ \underline{v_\delta} & \text{for} \quad \neg h_1 \wedge u_1 \leq \underline{v_\delta}, \\ \overline{v_\delta} & \text{for} \quad \neg h_1 \wedge u_1 \geq \overline{v_\delta}, \\ u_1 & \text{otherwise}, \end{cases} \tag{4.2}$$

$$\dot{v} = \begin{cases} 0 & \text{for} \quad h_2, \\ \underline{a} & \text{for} \quad \neg h_2 \wedge u_2 \leq \underline{a}, \\ \overline{a} & \text{for} \quad \neg h_2 \wedge u_2 \geq \overline{a}, \\ u_2 & \text{otherwise}, \end{cases} \tag{4.3}$$

$$\dot{\psi} = \ddot{\psi}.$$

The derivations of $\dot{\psi}$ and β are combined formulas composed of the forces acting on the front and rear axis of the car, the friction coefficient μ and the cornering stiffness $C_{S,i}, i \in \{f,r\}$ of the tires (f=front, r=rear). The mentioned parameters are relevant for the angle dynamics because they are the important and only parts acting on the axes, while considering the tire properties. We omit the exact definition of β and $\ddot{\psi}$, for a further explanation, we refer to the paper [Alt17]. The presented dynamics of the bicycle-model are non-linear similar to the dynamics of the kinematic single-track model. However, they cover more aspects of a realistic vehicle representation and can be configured to display different car models. Due to the additional variables for yaw angle and slip angle, the trajectory following is more detailed and realistic, especially in curved trajectories. Consequently, the set of reachable states resulting from the flowpipe-construction-based reachability analysis represents a more defined and narrow trajectory and can verify trajectories in real traffic scenarios. On the contrary, the increased complexity of the bicycle model increases the computation time needed, when the model is used in an application. Moreover, as a result from the definition of δ and v in cases, side constraints regarding design limitations of the car are outsourced to the vehicle model. The effect this design specification has on the computation time requires a further comparison between the consideration of car specific bounds in the constraint set of an optimization problem, as in the MPC-based trajectory planner (see Section 5.6), and the modeling of design limitations in the vehicle model as shown here [Alt17].

Table 4.1: Datasheet of two different vehicle specifications in the bicycle-model [ASK+92].

vehicle parameter			vehicle id	
name	symbol	unit	1	2
vehicle length	l	[m]	4.30	4.51
vehicle width	w	[m]	1.67	1.61
total vehicle mass	m	10^3[kg]	1.23	1.09
moment of inertia for entire mass about z-axis	I_z	10^3[kg m^2]	1.54	1.79
distance from center to front axle	l_f	[m]	0.88	1.16
distance from center to rear axle	l_r	[m]	1.51	1.42
center of gravity height of total mass	$h_{c,g}$	[m]	0.56	0.57
cornering stiffness coefficient	$C_{s,f}$	$[\frac{1}{\text{rad}}]$	20.89	20.89
friction coefficient	μ	$[-]$	1.05	1.05

The friction coefficient μ is originally derived in the Adams tire documentation in [Sof11] and all other relevant vehicle parameters are taken from [ASK+92].

Besides the bicycle model there are certain vehicle models representing a car in a more realistic form. The so called multi-body model first presented in [ASK+92] increases the consideration of relevant parameters in vehicle models. It consists out of 29 variables defining the vehicle's characteristics, while considering forces acting on the tires and the physical vehicle body. A complete tire model and nearly all vehicle parameters compared to a real vehicle are considered, including the body, axes, wheels and auxiliaries. Owing to the fact that this thesis rather focuses on the verification of a vehicle controller than on the representation of a high order vehicle model, the multi-body model could be dealt with in future research, but does not fit the intention of this work.

Moreover, M. Althoff and J. M. Dolan proved that the bicycle model can be used as decisive approximation model for the multi-body model in various numerical examples [AD12]. In fact, it has been shown that introducing an over-approximation in the bicycle model suffices to approximate the behavior of more complex vehicle models. This approximation procedure shows that the approximated set of reachable states in the bicycle model safely over-approximates the set of reachable states from the multi-body model. Therefore the bicycle model is proved to work as a reliable substitute model. The verification procedure is realized by a rapidly-exploring random tree (RRT), validating that the multi-body model cannot reach a state outside of the computed set of reachable states. According to the previously mentioned paper [AD12], the bicycle model can be used as a substitution model for the multi-body model with a certain over-approximation. As a result, the processing time to verify a trajectory with a high-order vehicle model can be substantially reduced by substituting it with the bicycle model [AD12]. This is reasoned by the fact that the use of the multi-body model in computations has a higher time complexity than the usage of the bicycle model.

4.4 Vehicle Model Comparison

To illustrate the differences of the presented vehicle models we have implemented and analyzed a small selection of driving maneuvers using the different models. The point-mass model and kinematic-single track model are compared in a simulation and reachability analysis, depicting a left curve. Moreover, two vehicle specifications of the bicycle model are analyzed with a reachability analysis.

Figure 4.4, displays a simulation of the PM and the KST in a scenario with constant velocity and a left curve. It is clearly shown that the difference between the two shown trajectories constructed by a simulation lies solely in numerical differences. On the contrary the flowpipe-construction-based reachability analysis (see Figure 4.5) shows the flowpipe of the point-mass-model to evolve and widen, while the flowpipe of the kinematic-single-track models expands constantly without diverging. The in Figure 4.5 shown diverging flowpipe is reasoned by the fact that the dynamics of the PM do not model a heading angle, therefore the coordinates x and y evolve in both directions defined by the velocity variables v_x and v_y. On the other hand, the KST models an additional heading angle allowing the set of reachable states to evolve in a narrow and realistic manner.

Due to this, the kinematic single-track model is suited to verify trajectories in a realistic manner, while the diverging flowpipe of the point-mass model does not represent a realistic set of reachable states. Moreover, the point-mass model can only

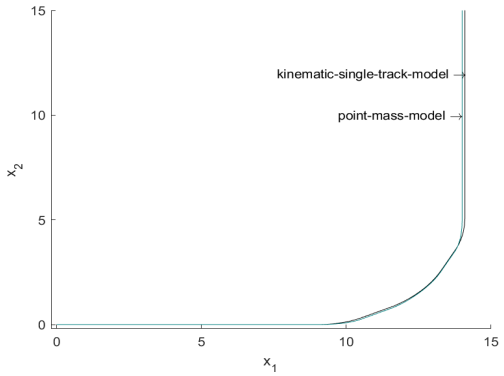


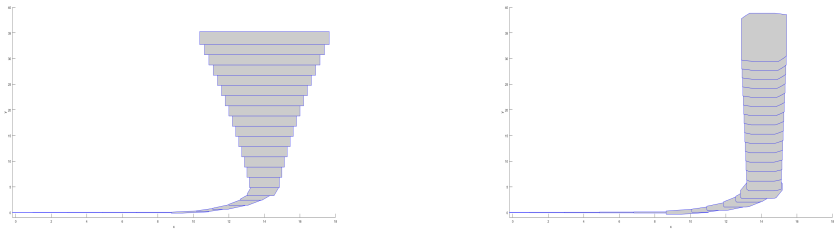
Figure 4.4: Point-mass model vs. kinematic-single track model.

work with inputs defining the acceleration values in the x and y direction, whereas realistic vehicle controllers generate inputs consisting of an angular velocity and an acceleration value.

To compare the efficiency of the two vehicle models, we further have to compare the time complexity of the PM and the KST. The computation of the flowpipe using the kinematic single-track model had a running time of 0.17s over 26 time steps with a time step size of one second, while the point-mass model needed 0.09s in the same context. The numerical results correspond to our assumption that the simulation time of the models do not differ by a considerable factor.

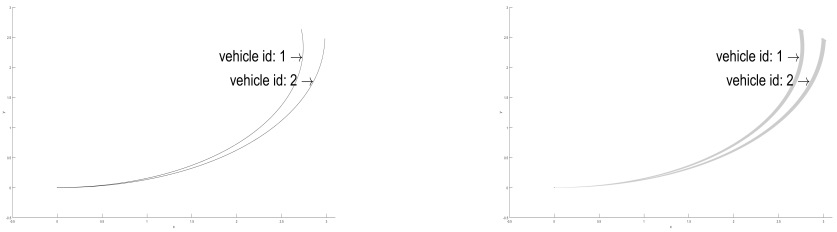
Further the two vehicle types of the BM specified in Table 4.1 are compared in a selected maneuver. The representation of different vehicle types by adjusting the vehicle body parameters of the BM is one of the advantages of the bicycle model. Hence, the bicycle models allows us to represent numerous different vehicle designs by adjusting corresponding parameters variables. This enables the option for us to compare different models of physical vehicles in numerical experiments. In Figure 4.6, the vehicles one and two are depicted and compared in a simulation and reachability analysis of a maneuver with constant acceleration and a small angular velocity. The parameters for vehicle one are taken from a Ford Escort, which is a small car compared to other vehicles [ASK$^+$92]. Additionally, to analyze and compare the constructed trajectory to the trajectory of a different vehicle type, the parameters of vehicle two are taken from a BMW 320i [ASK$^+$92]. The BMW 320i is a car with a medium sized vehicle body.

The first numerical experiment compares the two vehicle specifications over a time duration of two seconds, vehicle one had a computation time of on average 3.94s for the simulation, while vehicle two needed 3.93s. The results for the flowpipe-construction-based reachability analysis performed with both vehicle types was 3.87s for id one, respectively 3.80s for id two. The second experiment shown in Figure 4.6 divided in a simulation and flowpipe-construction-based reachability analysis was computed with a time duration of ten seconds and a time step size of 0.10s. The processing time for vehicle id one in the simulation shown left was on average 6.28s, while vehicle two needed 6.13s. Moreover, the processing time of the flowpipe construction for the two vehicle specifications was 17.19s for id one and 17.76s for vehicle



(a) Point-mass model. (b) Kinematic single-track model.

Figure 4.5: Comparison of point-mass and kinematic single-track model.



(a) Simulation. (b) Set of reachable states.

Figure 4.6: Comparison of the bicycle model with different vehicle specifications.

two.

As a result, the processing time to compute the set of reachable states is as assumed much higher than the time required to perform a simulation. Moreover, the time complexity of the two vehicle types does not vary in a notable measure.

The only difference between the two vehicle specifications is shown in the depicted plots. The computed trajectories shown in Figure 4.6 show that vehicle one had a smaller turning circle than vehicle two, which is reasoned by the fact that the Ford Escort has a much smaller physical body than the BMW 320i [ASK+92].

All computations have been performed on a system with a six core i5 processor with 2.80GHz. Further details about the collected time complexity results are declared in Appendix A.1. Additionally, zonotopes were used as state set representations of the computed flowpipes and the function *Reach* implemented in the toolbox Cora has been used to compute the set of reachable states [AK18].

Chapter 5

Controller Models

This chapter displays various approaches in controller verification and algorithms regarding path planning and input generation for autonomous vehicle control. Before going into detail, we have to explain a certain term required in order to realize the verification of a trajectory planner for autonomously driving vehicles.

Certain verification approaches in autonomous vehicle control rely on the representation of vehicle maneuvers as so called motion primitives. Maneuvers are used to define the movement of a particular vehicle in a traffic scenarios. For instance, a right or left turn, an acceleration, an overtake or a lane change are typical maneuvers executed in everyday traffic. These maneuvers can be realized by infinitely many different trajectory variations. Therefore to discretize this set into a finite set of trajectories, we have to specify a formal definition grouping certain trajectories which inherit the basic structure of the same underlying maneuver. This formal specification is called *Motion Primitives* and groups finite sets of trajectories into equivalence classes, which describe a specified maneuver [FDF05]. Intuitively, a *Motion Primitive* describes an equivalence class of a trajectory set inheriting the basic property of a specified maneuver, while differing only in a time translation and certain restricted trajectory changes.

5.1 Maneuver Automata

In theory, there are many approaches and ideas how to define a maneuver automaton. One approach is to define a maneuver automaton as a hybrid system by modeling it as an hybrid automaton, in which each location represents one maneuver. To execute the desired maneuver, the automaton has to take a transition to the specific location. This is only possible, if the corresponding guard is satisfied, ensuring maneuvers to be only executable if they are physically feasible to be executed successively. Furthermore, the named design specification is ensured by the *enclosure condition* (see Equation (5.1)). The *enclosure condition* states that the final set of reachable states of the first maneuver is fully enclosed by the initial set of reachable states of the second maneuver. Additionally, the *enclosure condition* has to be satisfied by all successive elements in the transition set Δ in Definition 5.1.

Moreover, verifying the automaton and guaranteeing a safe trajectory is possible by analyzing the set of reachable states of the system as described in Section 2.1.2.

The computation of all possible reachable states of the system requires high computational effort. In the optimal case, we want to have an online controller, which directly adapts the trajectory while ensuring safety. Therefore all motion primitives together with their set of reachable states are pre-computed offline. Hence, a run of the maneuver automaton connects the pre-computed sets of reachable states of the selected motion primitives, while checking if corresponding sets are not intersecting with unsafe states to generate a safe and verified trajectory from initial to goal position [HAS14].

We use motion primitives to reduce the infinite set of varying trajectories describing a certain maneuver to a finite set of maneuver trajectories which can be verified in finite time.

Definition 5.1: Maneuver Automaton [HAS14]

$$\mathcal{MA} = \{\mathcal{M}, \Delta, \mathcal{M}_0, G\}$$

The *maneuver automaton* consists of:

- $\mathcal{M}$ describes the finite set of maneuvers m_i (motion primitives) as locations.
- Δ is the set of discrete transitions between two maneuvers $\Delta \subseteq \mathcal{M} \times \mathcal{M}$, where a transition between two maneuver is specified by (m_a, m_b) (connects two maneuvers if they fulfill the enclosure condition defined in Equation (5.1)).
- $\mathcal{M}_0 \subseteq \mathcal{M}$ describes the initial maneuver set, consisting of all maneuvers which begin in the initial region.
- $G \subseteq \mathcal{M}$ describes the goal maneuver set, consisting of all maneuvers which are fully enclosed in the final region.

The transition set Δ describes possible connections of two locations. A transition between two locations is added to Δ, only if the *enclosure condition* is satisfied by the corresponding set of reachable states. The *enclosure condition* is formally described by Equation (5.1) and visually depicted in Figure 5.1 [HAS14].

$$\forall j \quad \mathcal{R}^j(t_j^f) \subset \mathcal{R}^{j+1}(t_{j+1}^0), \tag{5.1}$$

where $\mathcal{R}$ is the set of reachable states of a certain motion primitive at a certain time step, while j depicts the j-th motion primitive.

An execution of the hybrid automaton results in a list of motion primitives intended to be executed successively, beginning with an initial maneuver and ending with a maneuver, which is part of the goal set. The list of maneuvers depicting a successful run from the initial to the goal position is encoded in the locations visited during running time. The described procedure is structured in the following steps. First of all, the set of reachable states of the motion primitives is is pre-computed offline Secondly, the maneuver automaton is constructed, following Definition 5.1. The maneuver automaton is constructed by the following in this thesis simplified steps. First, to construct the automaton a set of maneuvers is given, defining the locations of the hybrid system. To maximize the number of transitions in the automaton, the initial set of reachable states for each maneuver is guessed. Afterwards, in each step two

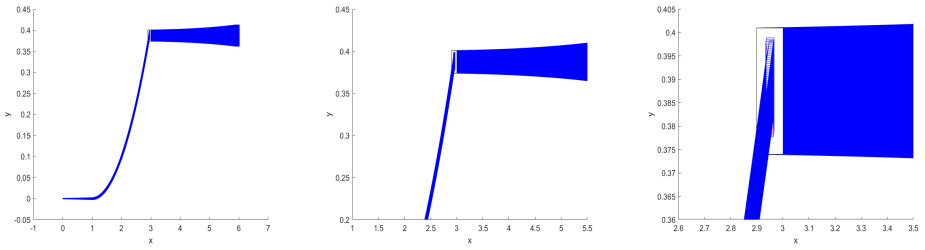


Figure 5.1: Two set of reachable states, fulfilling the *enclosure condition* (Δ).

maneuvers are checked to fulfill the *enclosure condition* that has to be satisfied to be part of the transition set. If the condition is not satisfied the initial set of reachable states of the second maneuver can be expanded to enclose the final set of the first maneuver and thus fulfill the requirement of Δ. This procedure is repeated until the quantity of elements in the transition set Δ is maximized while it is ensured that all elements satisfy the *enclosure condition*. Further explanations on the formal construction are described in [HAS14].

Afterwards, the maneuver automaton is executed connecting different sets of reachable states of motion primitives. In parallel, the occupancy set of other traffic participants is computed, describing the possible set of possible reachable states of the obstacles at a discrete time step. Eventually, the set of reachable states of the automaton and the occupancy set of the obstacle vehicles are checked for an intersection. A set is classified as safe, if there is no intersection with the set of reachable states of obstacle vehicles. If, more than one safe set of connected motion primitives exist, the driver can control the vehicle, whereas if only one safe trajectory is available, the maneuver is executed by the controller [HAS14]. The described procedure could be transformed into a complete autonomous vehicle controller by allowing the controller to execute a safe maneuver in each time step.

However, the instance that the automaton was not able to construct a safe set of motion primitives because all set of reachable states of available locations are intersection with bad states is not defined in the original approach. A possible option to be performed in the defined case would be to minimize the occurring damage of an inevitable crash by slowing down the vehicle to a minimal velocity. Otherwise one could enable the hazard warning lights of the controlled car to affect the driving behavior of other traffic participants possibly causing free space, enabling an evasive emergency maneuver to be executable.

As a result of the described transition-set condition visualized in Figure 5.1 and the repeated intersection check to verify that no bad state can be reached, the resulting maneuvers are verified to build a connected and safe trajectory. Attributable to the fact that the automaton only connects pre-computed motion primitives, fulfilling the *enclosure condition*, the approach guarantees safety while executing the maneuvers. However, they do not have to be optimal in terms of efficiency and comfort.

5.2 Formally Verified Motion Planner

Besides the flowpipe-construction-based reachability analysis, a reachability analysis can be realized by a proof assistant and satisfiability checking. In the following the proof assistant Isabelle is used to construct a reachability analysis which is verified by satisfiability checking [RISA18]. In addition to the general verification of the generated trajectory, the approach has the capability to consider uncertainty in the computation process.

Implementation details concerning the reachability analysis with the help of a theorem prover, along with the solution of technical problems to build an interface between both platforms (Matlab, Isabelle) can be found in [RISA18].

Instead of analyzing and explaining the implementation details, we rather focus on depicting the method to verify a hybrid system in autonomous vehicle control with satisfiability checking [RISA18].

The verification procedure of the motion planner starts with the definition of a maneuver automaton as in Section 5.1. The maneuver automaton varies slightly in comparison to the Definition 5.1, consisting of a maneuver set M, a set of *jumps* connecting two maneuvers satisfying the *enclosure condition* and a set of ODEs describing the underlying trajectory of a maneuver $m_i \in M$ (Definition 5.2).

Definition 5.2: Maneuver Automaton [RISA18]

$$MA = (M, jump, ode)$$

M The set of all maneuvers is denoted by M.

$jump$ The transition set connecting two maneuvers satisfying the *enclosure condition* is specified by $jump :: (M \times M)$, where two maneuvers m_a, m_b connected in a $jump$ are denoted by (m_a, m_b).

ode The set of ODEs describes the trajectory of each maneuver m_i by an ODE: $ode(m_i) :: \mathbb{R} \times \mathbb{R}^n \Rightarrow \mathbb{R}^n, \quad \forall m_i \in M$.

The explanation on how to construct the corresponding ODE, describing the trajectory from initial to final position of a certain maneuver, is omitted due to the fact that we solely focus on verifying the trajectory. To further verify the system and corresponding motion planner, we have to provide a formal specification which has to hold during the execution of the MA. In the following we denote with $reach(m_i)$, the set of reachable states of a maneuver m_i.

A *safe path* is denoted by the following formal definition.

Definition 5.3: Safe Path [RISA18]

A *safe path* of the MA is a series of maneuvers $m_0, \ldots, m_{n-1} \in M$, fulfilling the following three conditions (M is the set of all maneuvers):

(i) Sequential listed maneuvers are connected by a jump transition.
$(m_i, m_{i+1}) \Rightarrow (l_{m_i}, g, r, l_{m_{i+1}}) \in Edge \quad \forall i, i+1 \in [0, \ldots, n-1]$

(ii) No set of reachable states intersects with an unsafe region $\mathcal{D}$.
$reach(m_i) \cap \mathcal{D} = \emptyset \quad \forall m_0, \ldots, m_{n-1} \in M.$

(iii) The final part of the set of reachable states of m_i is fully enclosed in the initial set of reachable states of m_{i+1}.
$reach(m_i) \in init(m_{i+1}) \quad \forall i, i+1 \in [0, \ldots, n-1]$

The next step of the algorithm is to transform the maneuver automaton into a motion planner. This is realized by interpreting LTL over the automaton. Atomic propositions are used to represent relevant factors regarding the vehicle's environment. In particular, lane borders, other traffic participants or other influencing traffic operators can be displayed by an atomic proposition AP.

Moreover, the atomic propositions can be connected by the operations below:

$$\phi ::= true \mid \pi \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \chi\phi \mid \phi_1 \cup \phi_2,$$

where $\pi :: atom$.

In addition to the standard LTL operations, a new data type is introduced adding the possibility to add a sign to each AP, which is required to state more complex formulas to enable a more decisive verification.

$$\textbf{datatype } aprop = AP^- \mid AP^+$$

The semantics of the LTL are defined as in the standard LTL except for the semantics of the introduced data type.

We only depict the part of the semantics deviating from standard LTL:
The deviating LTL rules for the used definition of a maneuver automaton are defined over a finite sequence of sets $\delta = \mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_{n-1}$, where $A_i :: \mathbb{R}^n$ for $0 \leq i < n$, as described below, where the set $\mathcal{A}_i$ describes the set of reachable states of the i-th maneuver m_i:

$$\begin{aligned} &\delta \models \pi^+ \Leftrightarrow \mathcal{A}_0 \subseteq [[\pi]] \\ &\delta \models \pi^- \Leftrightarrow \mathcal{A}_0 \cap [[\pi]] = \emptyset, \\ &\text{where } [[\_]] :: AP \to \mathbb{R}^n \end{aligned} \tag{5.2}$$

The interpretation function $[[\_]]$ maps an atomic proposition AP to a set of real numbers to enable a comparison between a region defined by an AP and a set of reachable states of a certain maneuver m_i as defined by the π^+, π^- notation. The semantics described in Equation (5.2) are used to check for an intersection or an intersection freedom between the positions defined by the maneuver sequence and the sets defined by the atomic propositions [RISA18].

Intuitively, the purpose of describing the motion planner by LTL is to define a formula specifying regions that have to be intersected and regions that have to be avoided by the computed trajectory. Further, to formally verify a sequence of motion primitives $(m_0,...m_{i-1})$, we define a formula depicting the goal state and safety restrictions which have to be satisfied. The intended goal position of the trajectory and certain safety restrictions which have to be fulfilled are specified by an LTL formula, called reach-avoid set. In detail, a reach-avoid set describes an LTL formula divided in parts which want to be reached (goal state, current lane) and parts which have to be avoided (lane bounds, obstacles, ...) [RISA18].

The newly introduced semantics expand the verification to a sets of trajectories instead of single trajectories as it would have been possible with standard LTL. This

expansion of the verification aspect together with a detailed reasoning to introduce the new data type can be looked up in [RISA18]. Owing to the fact that a sign can be added to each atomic proposition, the reach-avoid set becomes more complex by formally defining the underlying geometric regions depicted by the LTL formulas.

Further with the defined syntax and semantics of LTL, we can state a procedure to check, if a given LTL formula can be satisfied by a sequence of motion primitives.

Definition 5.4: Satisfiability of a Motion Sequence [RISA18]

An LTL formula ρ is satisfiable, if there is a series of maneuvers, such that:

$$reach(m_1), reach(m_2), \ldots, reach(m_n) \models \rho,$$

where m_i is an element of the maneuver set M (see Definition 5.2).

Precisely, an exemplary reach-avoid set contains conditions that no traffic regulation is violated and the trajectory of the driver's vehicle intersects at the final position with the goal area. These statements can be realized with the additional semantics introduced in Equation (5.2) to prove intersection freedom with unwanted bad states and an intersection with the defined goal position. The SAT solver tries to compute a set of motion primitives to fulfill the formula defining the reach-avoidset. As a result of this, if the reach-avoid set is solvable, a list of motion primitives is obtained which can be connected and executed by the vehicle model to describe a safe trajectory from the initial to the goal position. An exemplary reach-avoid set is depicted in Equation (5.3) [RISA18].

$$\rho = (driving\_lane^{+} \wedge obstacle\_set^{-} \wedge lane\_borders^{-}) \cup goal\_location^{+} \tag{5.3}$$

The reach-avoid set can be divided in four parts to further explain the procedure. The first part $driving\_lane^{+}$, describes the condition that the trajectory described by the maneuvers satisfying the formula has to be part of the driving lane. Intuitively, the obstacle set and the lane borders have to be avoided by all motion primitives, depicted by the minus. Finally, the set of motion primitives has to intersect with the $goal\_location^{+}$. Besides the stated verification method, it has to be noted that the resulting list of motion primitives does not have to represent the optimal trajectory in terms of efficiency and comfort. The approach focuses on the verification of the path planner, rather on the optimization of generated inputs as in Section 5.6.

5.3 Safety Verification with Theorem Proving

Another method to perform a reachability analysis of a hybrid system is based on theorem proving. A special logic is used to specify the runs of hybrid systems and with an additional definition of safety terms, safety requirements can be defined and verified to hold in all executions of the hybrid system [PQ08, MGP13]. The selected logic is used to state certain terms defining safety rules for the runs of a hybrid system.

In the context of autonomous vehicle control, the safety formulas depict the distance to obstacles regarding the velocity of all traffic participants to ensure that safe stops are possible at every time step. Additionally, to prevent blocking other vehicles

that are driving in the close environment, a term restricts the operators' vehicle to stop at an adverse spot, causing a collision with an obstacle vehicle.

Finally, the logical specifications are used to advance the used algorithm for path planning to ensure additionally that the generated inputs defining the movement of the operators vehicle do not violate the specified safety terms. This is realized by a theorem prover which verifies that the specified safety terms are satisfied for all possible runs of the hybrid system [MGP13].

However, the stated approach to verify the runs of a hybrid system with a theorem prover is not suited to be used in autonomous vehicle control. This is science-based by the fact that the theorem prover does not work completely autonomous. The approach depicting the stated procedure is only to 85% automatic, the remaining 15% are covered by user interaction. This aspect contradicts the purpose of this thesis to verify an *autonomous* vehicle controller. Therefore we did not choose to use reachability analysis based on a a theorem prover to verify the later implemented controller models.

5.4 Maneuver Templates

Instead of generating and verifying a trajectory out of a set of motion primitives as in Sections 5.1 and 5.2, another approach focuses on using a similar structure called maneuver templates to group certain trajectories to improve the running time of a motion planner.

The maneuver templates coordinate a group of traffic participants, instead of controlling solely one vehicle as in the maneuver automaton. The main purpose of the approach is to improve the running time of cooperative motion planners. For further details about the time complexity reduction of the approach, we refer to [MLA17]. A minor insight into the reduction of the running time follows later.

It should be noted that the maneuver templates are not a fully functional controller model, they are an additional specification to structure trajectories in certain traffic scenarios. Nevertheless, the approach is presented in the following section to show a similar usage of a formal structure grouping trajectories in maneuver-based path planning algorithms.

First of all, all traffic participants are divided in controllable cooperative vehicles and obstacle vehicles which cannot be controlled, but have to be considered to avoid collisions. In each step the motion planner can check, if there is any template available and safe executable. In case, there is a feasible template available for the current valuation, it can be executed. This can improve the time efficiency of the algorithm. In particular, the use of a template can reduces the search space of a motion planner based on an A* algorithm.

Owing to the fact that we cannot provide a template for every possible scenario, the approach cannot be applied in every situation [MLA17]. However, it is proven that a small amount of traffic patterns is sufficient to display nearly all traffic situations [ZGU13]. Traffic patterns are formal descriptions, describing for instance the procedure to manage traffic at an intersection. The difference to maneuver templates is the different formal structure describing the allowed vehicle trajectories. Therefore it is assumed that the same is accountable for maneuver templates and traffic scenarios.

On the contrary, if no template is executable, the motion planner constructs the required inputs for the next time step without further assistance. The stated procedure is repeated in each discrete time step until the motion planner finishes computing.
First of all, we have to define the general structure of a maneuver template.

Definition 5.5: Maneuver Template [MLA17]

A maneuver template is a two tuple, consisting of $\mathcal{M}$, defining the dynamics of the traffic participants and a constraint set $\mathcal{C}$, specifying the maneuver by a set of restrictions.

$$\mathcal{T} = (\mathcal{M}, \mathcal{C})$$

$\mathcal{M} = (f, \mathcal{Z}_0, \mathcal{U})$

- f specifies the continuous dynamics of certain traffic participants, where $\dot{z} = f(z(t), u(t))$ defines the dynamics of the traffic participants with $z \in \mathcal{Z}$ being the current status vector, describing the vehicle state and $u \in \mathcal{U}$ is a vector containing the control inputs for each time step. The initial vectors defining inputs and the current state of the cooperative vehicles are restricted by $z_0 \in \mathcal{Z}_0 \subset \mathbb{R}^n, \forall t : u(t) \in \mathcal{U} \subset \mathbb{R}^m$.
- $\mathcal{C} = \{C_1(z, u), \ldots, C_p(z, u)\}$ describes the constraint set of the maneuver template.
  Certain single constraints $C_i(z, u) : \mathbb{R}^n \times \mathbb{R}^m \to \{true, false\}$ assign to a certain input set, consisting of the initial state z and the time discrete input u, the Boolean values *true* or *false*.

The maneuver templates consist of dynamics for all traffic operators and a constraint set ensuring the compliance of traffic rules and safety constraints. Intuitively, the templates define a set of trajectories satisfying and operating in a specified manner.
The constraint set can be defined as follows:

Definition 5.6: Constraint Set $\mathcal{C}$ [MLA17]

$\mathcal{C} = \{C_1(z, u), \ldots, C_p(z, u)\}$ is split into the subsets $\mathcal{C} = \mathcal{B} \cup \mathcal{H} \cup \mathcal{A}$.

$\mathcal{B} = \{b_1(z_0, t_0), \ldots, b_i(z_0, t_0)\}, i \in \mathbb{N}$ (*initial conditions*)

- $\mathcal{B}$ defines the initial terms that have to be satisfied at the initial time t_0 to apply the template. Exemplary constrains are restrictions on initial velocities of the cooperative vehicles and safety distances between the cars.

$\mathcal{H} = \{h_1(z(t), t), \ldots, h_k(z(t), t)\}, k \in \mathbb{N}$ (*general constraints*)

- $\mathcal{H}$ defines conditions to be fulfilled while the template is executed $(t_1, \ldots, t_{f-1})$. In particular, the set $\mathcal{H}$ includes constraints determining the driving direction and maximal allowed velocity, while ensuring lane keeping and collision avoidance.

$\mathcal{A} = \{a_1(z(t_f), t_f), \ldots, a_j(z(t_f), t_f)\}, j \in \mathbb{N}$ (*goal conditions*)

- $\mathcal{A}$ defines the status each vehicle has to be in, after a successful execution of the template at the final time t_f. For instance, this

> can include the final position of the vehicle, the safety distances between them and the covered track of the cooperative vehicles.

The properties stated in the constraint set definition do not cover all possible implementable constraints. They are used as examples to get an understanding of a possible template structure. For a further example applied in a numerical experiment with a motion planner, we refer to [MLA17].

In order to ensure that the algorithm is working correctly, it is examined, if for a discrete time step a feasible template is available.

Therefore the solution set of a template at a given time t is defined by:

$$\begin{aligned}\mathcal{Z}(t) = \{\chi\left(t; z_o, u(\cdot)\right) \mid z_0 \in \mathcal{Z}_0, \forall t : u(t) \in \mathcal{U}, \\ \forall t \forall i : C_i\left(\chi(t; z_0, u(\cdot)), u(t) = true\right)\}\end{aligned} \tag{5.4}$$

Equation (5.4) specifies the set of possible solutions for a template at a certain time t, an initial state z_0 and an input vector describing the dynamics of the corresponding vehicles $u(\cdot)$. Intuitively, the set $\mathcal{Z}$ contains all vehicle dynamics, defined by $u(\cdot)$ and the next discrete time steps t which satisfy the condition that for an initial state z_0 and for all following time steps the set of constraints is satisfied by the dynamics of the cooperative vehicles. For a simplified explanation of the computation process of the solution set we refer to Section 5.4.1 and for a detailed explanation we refer to [MLA17].

The complete template method is structured in smaller steps. First it is tested, if the initial conditions of a template are satisfied ($\mathcal{B}$), removing unfeasible templates. For instance a template designed for five cooperative vehicles cannot be applied in a situation with only four controllable cars. In the next step, templates with empty solution sets are discarded, because they cannot be executed. This is computed with the help of Equation (5.4). Templates can have empty solution sets, if the initial conditions are satisfied, while there are no corresponding trajectories satisfying the conditions of the subsets $\mathcal{H}$ and $\mathcal{A}$. An exemplary description would be the initial restriction set that specifies the velocity of vehicles and distances between them which is satisfied by the states of the current vehicles. However, the goal constraints define that an overtake maneuver has to be realized which is not possible, because the obstacle vehicles are blocking the required lanes.

All resulting templates can be performed safely and with the help of an optimization framework the optimal template is selected.

Afterwards, the motion planner considers the selected template in the construction of the inputs describing the next trajectory part. This has shown to improve the time complexity of the used motion planner. In detail, the evaluated path planner worked with a tree data structure and an A* search algorithm selecting the optimal trajectories based on a heuristic function. The elements of the corresponding tree are maneuvers, while a node in the tree represents a set of connected motion primitives. In case a template is selected, the search space of the A* algorithm is reduced and the heuristic function is adjusted according to the template restrictions. This restricts the solution set and search space of the algorithm, while it has been shown to improve the time efficiency.

In the other case, if no template is selected, the motion planner, in particular the A* algorithm searches for the optimal maneuver without adjusting the tree structure. The maneuver templates are solely an optional tool to reduce the computation time

of the corresponding motion planner, they cannot take over tasks of a real controller for autonomously driving vehicles [MLA17].

5.4.1 Maneuver-Template Generation

One main disadvantage of the maneuver template approach is the fact that all templates have to be generated manually. This comprises the draft of the whole scenario, including the dynamics of the cooperating vehicles and the constraint set. Conditions for the initial, general, and goal states have to be designed and implemented by mathematical restriction terms. Until now, no algorithm is available to take over this task to design constraint sets for maneuvers dynamically. Though a future plan to implement an algorithmic construction of templates was mentioned in [MLA17].

Finally, to compute the solution set for a specific template an optimization steering problem is solved, while satisfying the restrictions stated by the template. First, the constraint set $\mathcal{C}$ is divided into the defined subsets $\mathcal{B}, \mathcal{H}, \mathcal{A}$. Then, a minimization problem is introduced, computing an optimal trajectory considering the bounds defined by the template. Further details on the procedure are described in [MLA17].

5.5 Electric Field Model

The electric field model is a theoretical concept for vehicle path planning, in which the vehicle is modeled as an electron moving through an electric field containing a set of potentials repelling the electron [RS94].

The environment of the car is modeled as a risk map with different potentials representing various traffic interactions (see Figure 5.2). The potentials represent risk factors including other traffic participants, lane regulations and in general obstacles. Those potentials are weighted individually to fit the intended use, the highest priority is to avoid collisions in all cases, neglecting the driver's comfort, if needed. Therefore, to ensure safety, all objects on the map potentially causing a collision are assigned the highest weight. Additionally, to realize a certain maneuver, new potentials are placed in the close environment of the vehicle to force a trajectory modification by pushing the vehicle in the desired direction. In particular, a potential is placed before or behind the vehicle to slow down or accelerate it and left or right to push it to the desired neighboring lane [RS94]. Lane borders and obstacles are modeled by defining potentials at the specific regions of the risk map to create repulsing forces.

A dynamic motion planner is constructed, if, additionally to the potentials a goal region is specified and added to the risk map. The goal region differs in comparison to the other potentials on the map by attracting the electron instead of repelling it. Hence, by combining the forces induced by the potentials and the attracting force caused by the goal region a force field emerges, which represents the forces acting on the electron at each position of the map.

The resulting force field can be used to construct the desired path. The purpose of the theoretical concept is to construct an approach used inside a trajectory planner to build optimal paths with respect to the close environment of the vehicle [RS94].

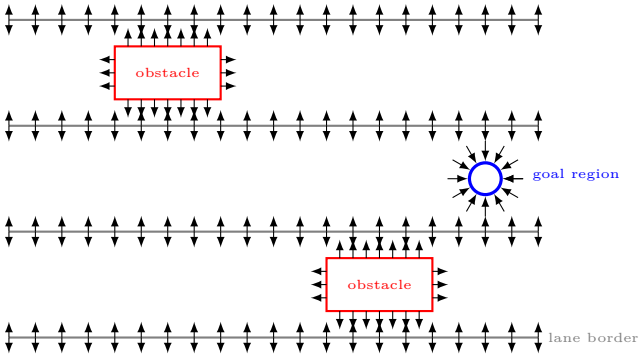


Figure 5.2: Risk map, consisting of various potentials and a goal region.

5.5.1 Trajectory planner using the E.F Model

To explain the electric field model in a more practical application, an approach is presented using the electric field model to optimize the trajectory constructed in a path planner.

The algorithm is structured in two basic steps. First, a general trajectory planner computes an initial reference trajectory with guaranteed safety under the assumption that the environment will not change. Afterwards, a local planner modifies the reference path. In the first step, the local planner detects and avoids obstacles by adjusting the reference trajectory to ensure collision avoidance. In the second step, the property of the electric field model to depict the force acting on a specific point is used to influence the control inputs by modifying the trajectory to follow the points with the minimal repelling force.

To formalize this condition, the risk map depicted by the corresponding force field is modeled by a topological structure called potential valley. Accordingly, a minimal potential valley (MPV) represents points of the surface at which the force acting on the electron is minimal. Owing to the fact that the MPV describe the whole environment map, there are infinitely many points in the MPV. To discretize the representation, the MPV is transformed into a graph. This procedure is realized by transforming local minima in the MPV to nodes. In addition to complete the graph structure, the edge set is assigned direct lines connecting these nodes. The described transformation is further described in [HA92].

Consequently, the global planner uses an improved search algorithm with a corresponding heuristic function to compute the reference trajectory. The heuristics used in the search algorithm are defined by considering the path length and distance to near obstacles. Afterwards, the local planner moves along the generated reference path and modifies the path by minimizing the repelling factors affecting the vehicle on its current position. The minimization of the sum of forces acting on the vehicle intuitively ensures the avoidance of obstacles. For a further explanation about how to modify the reference path in a complex vehicle constellation to still avoid a collision while optimizing the generated trajectory, we refer to the original paper [HA92].

We included this concept in this thesis to cover a theoretical approach in path planning, which is a precursor to the comparable MPC in the following Section 5.6.

5.6 Model Predictive Control

Model predictive control (MPC) is a modeling approach which focuses on predicting the future evolution of the underlying system to optimize the generated results based on various assumptions. To realize the aspect of predicting the future behavior, a *prediction horizon* is used, defining the next time steps that are considered in the related computation.

In the context of automotive vehicle control, MPC is used to generate optimal inputs for a vehicle, regarding driving efficiency and comfort. The corresponding system works with a closed loop between the MPC-based controller and the vehicle model. At each time step the controller generates new control inputs which are given to the vehicle model. Afterwards, the vehicle model executes a determined time step and the new state of the car is given back to the controller. Then, the controller updates the current environment data and computes the next optimal input regarding design limitations and driving efficiency. The inputs are given to the vehicle model and the described procedure is repeated until the vehicle reaches the goal region.

In order to compute optimal results, a certain behavior of all traffic participants has to be assumed. With this in mind, a MPC-based algorithm computes the inputs for a vehicle over the next time steps assuming a certain movement of the other traffic participants. Moreover, in the context of this thesis the prediction of prospective occupations of the obstacle vehicles are determined by given benchmark information. However, in real traffic situations sensors have to track the current movement of the obstacles to enable the model to predict future positions.

Diverse approaches can be followed to predict the future position of a vehicle, from intuitive ideas assuming a constant evolution of the vehicle dynamics to complex computations as in [KA17].

To focus on the general verification process the aspect to model the future position of traffic participants is simplified throughout this thesis. In real situations the future positions of cars in close range can be approximated by the current velocity and driving direction the vehicle is expected to follow. In order to consider the future behavior in mathematical computations, a *prediction horizon* is used, specifying the amount N of discrete time steps the MPC-based trajectory planner considers while computing the inputs [LT18].

The structure of the approach is divided in four parts, beginning with the data extraction, in which an offline available map is formatted to be used in later steps. In real traffic situations the map extraction has to be updated immediately, together with sensor readings updating the position of obstacles to represent the current environment. In the second step a reference trajectory is computed. A reference trajectory is an initially computed path which connects the current position of the vehicle with the desired goal location without considering smoothness of the trajectory or the driver's comfort. The reference trajectory is a basic trajectory, connecting the initial and goal position based on the given environment map. Afterwards, the computed reference trajectory is fed into the MPC-based controller. The MPC modeling approach is realized by an optimization problem refining the given trajectory over the next N time steps to have smooth curves and be optimal regarding driver's comfort and efficiency. In the last step, the inputs generated by the controller are given to the vehicle model. The vehicle model executes one time step δ and returns the new position and vehicle variable valuation to the controller. The complete procedure is repeated until the vehicle has reached the goal position. The whole structure together with the closed

loop between controller and vehicle model is depicted in Figure 5.3.

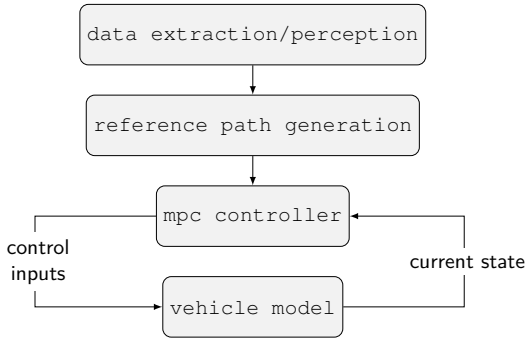


Figure 5.3: MPC-based controller model.

Further, the next section explains the four steps of the described approach in detail.

In the original approach in [LT18], a trajectory planner constructs a reference trajectory as a spline. By considering the purpose of this thesis to verify a controller rather than depicting an optimal path planner, the trajectory generation is simplified.

First, a data extraction script in Matlab transforms lanelet data [BZS14], describing the environment of the car, into objects consisting of lane center points and properties, including lane successors and lane neighbors. Afterwards, the previously created lane objects are used to construct a graph structure, containing vertices for each lane center point and edges for any possible connection between two points. To avoid unnecessary and redundant edges, we assume that the vehicle drives only in the by the road intended direction.

Before initiating the reference path computation, at first initial and desired positions of the trajectory have to be declared.

The approach stated in this thesis works differently in comparison to various current research sources [TCT+13, PKY+09]. It does not ensure collision avoidance by modeling various safety constraints in the optimization problem of the MPC-based controller. Instead, the safety requirements are implemented in the reference trajectory generation. Therefore the constructed reference trajectories are checked for an collision with an obstacle and afterwards if necessary updated to be collision free in the current environment. Formally depicting that the constructed reference trajectories are proven to be safe in the environment at the time they are constructed. Beginning with the current state, all feasible maneuver trajectories are computed. The maneuvers consist of lane following in each case and, if possible, a lane change to the left adjacent lane or right adjacent lane. This case distinction results in at most three generated maneuvers [LT18].

Finally, to compute the required initial position for the lane switch trajectories, we select the current lane point in the graph as initial coordinate. Moreover, the goal positions are defined by searching for the furthest reachable position in the neighboring lane by considering the current velocity, acceleration and the *prediction horizon* N. Coordinates describing the initial and goal position of the lane follow maneuver are intuitively computed by declaring the current position as initial point and the goal position by selecting an upcoming point on the current lane as stated before.

Thereafter, we obtain the reference path for each maneuver by searching for the shortest path between the initial point and the goal position in the graph structure. The resulting trajectories are verified to be safe under the made assumptions about the behavior of other traffic participants. This is a realized by a collision check which examines, if the constructed path intersects at any point with the current obstacle set. In case that there was a collision, the reference trajectory is updated to end with a safety offset before the obstacle. If there was no collision the trajectory is classified as safe and the algorithm proceeds. After constructing the reference trajectories, they are fed into the MPC-based optimization problem (see Equation (5.5)).

It has to be noted that the following procedure varies in comparison to the procedure in the original approach. The implementation is simplified, as the focus of the thesis does not lie on an optimal path planning algorithm based on MPC, but rather on the verification of an autonomous vehicle controller.

The implementation is required to check feasibility of an online verification method attached to a controller. Additionally, we can derive the requirements for a verification of a MPC-based vehicle controller.

In order to work as intended, the MPC based optimization problem (Equation (5.5)) specifies the vehicle's state in each time step. Therefore we define the dynamics of the KST by the two matrices A and B. We refer to the vehicle's state with the vector z_i, describing the relevant parameters of the vehicle model that are adjusted to build an optimal trajectory. Additionally, the vector u_i is used to model the derivatives of the heading angle and the velocity by realizing the dynamics of the KST as displayed in Equation (4.1). Finally, after computing the optimization problem, the vector u_i contains the resulting control inputs intended to be given to the vehicle model.

$$A = \begin{bmatrix} 1 & 0 & 0 & \cos(z_{k,3}) \\ 0 & 1 & 0 & \sin(z_{k,3}) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} B = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

In the following optimization problem (Equation (5.5)) the sequence of N states $z_{0,j}, z_{1,j}, \ldots, z_{n,j}, j = 1, \ldots, 4$ describes the current position, velocity and heading angle of the vehicle at each time step i. We have to model the position of the driver's vehicle in each time step of the optimization problem to define the evolution of the vehicles coordinates and physical variables by implementing the predicting aspect regarding the future state of the environment [LT18]. Moreover, the inputs given to the KST are defined by $u_{i,l}, l = 1, 2$, containing the acceleration a_i ($u_{i,1}$) and angular velocity ϕ_{v_i} ($u_{i,2}$). The index variable $i, i = 1, \ldots, N$ describes the current time step.

$$z_i = A \cdot z_i + B \cdot u_i$$

$$z_i = \begin{pmatrix} x_i \\ y_i \\ \phi_i \\ v_i \end{pmatrix} \quad u_i = \begin{pmatrix} a_i \\ \phi_{v_i} \end{pmatrix}$$

The MPC-based controller structure is divided in two different optimization problems, first of all, the reference trajectories are optimized according to the allowed error to the given trajectories, minimizing input terms and various design constraints by Equation (5.5) and afterwards the inputs are evaluated regarding certain penalty and reward functions in Equation (5.6). The first optimization problem realizes the MPC

approach by optimizing inputs for the given reference trajectory over the defined *prediction horizon* N considering an assumed future evolution of the environment to generate optimal results.

The *prediction horizon* N is used to optimize over the next N time steps, although only the first time step is given to the vehicle model as a control input. In each step a new optimization problem is solved to ensure a smooth and safe trajectory. The purpose of the optimization problem is to reduce lateral and longitudinal errors to the previously generated reference trajectory, while satisfying constraints for car specific design limitations and minimizing input changes.

In detail, the objective function of the optimization problem minimizes the lateral and longitudinal error between the given reference trajectory and the actual position of the vehicle by the first two terms multiplied by w_1 and w_2. Additionally, the next two terms are used to minimize the acceleration and angular velocity inputs to improve the driving efficiency. Moreover, the two fractions multiplied by the weighting coefficients w_5 and w_6 minimize the input changes between two consecutive time steps. This improves the driver's comfort by reducing high acceleration changes or steering changes. Finally, the two slack variables ϵ and ξ are added to the objective function and the constraint set to define the allowed error between the reference trajectory and the actual position. Owing to the fact that the variables are as well part of the objective function, they are used as slack variables to relax the error constraints, if needed, to increase numerical stability of the problem (see Equation (5.5)).

$$
\begin{aligned}
\min_{u \in U} \quad & \sum_{i=1}^{N} w_1 \cdot \underbrace{\left(\frac{z_{i,1} - r_{x,i}}{x_{max}} \right)^2}_{\text{longitudinal error}} + w_2 \cdot \underbrace{\left(\frac{z_{i,2} - r_{y,i}}{y_{max}} \right)^2}_{\text{lateral error}} + w_3 \cdot \underbrace{\left(\frac{u_{i,1}}{a_{max}} \right)^2}_{\text{min. acc.}} \\
& + w_4 \cdot \underbrace{\left(\frac{u_{i,2}}{\phi_{max}} \right)^2}_{\text{min. ang}} + w_5 \cdot \underbrace{\left(\frac{u_{i,1} - u_{i-1,1}}{a_{max}} \right)^2}_{\text{min. input changes}} + w_6 \cdot \underbrace{\left(\frac{u_{i,2} - u_{i-1,2}}{\phi_{max}} \right)^2}_{\text{min. input changes}} \\
& + w_7 \cdot \underbrace{(\epsilon_i)^2}_{\text{slack var.}} + w_8 \cdot \underbrace{(\xi_i)^2}_{\text{slack var.}}
\end{aligned}
\tag{5.5}
$$

$$
\begin{aligned}
\text{s.t.} \quad & z_{i+1} = A \cdot z_i + B \cdot u_i \\
& a_{min} \leq u_{i,1} \leq a_{max} \\
& \phi_{v_{min}} \leq u_{i,2} \leq \phi_{v_{max}} \\
& |(r_{x,i} - z_{i,1})| \leq \epsilon_i \\
& |(r_{y,i} - z_{i,2})| \leq \xi_i \\
& 0 \leq z_{i,4} \leq v_{max}, \text{where } N \text{ is the prediction horizon,} \\
& w_1, \ldots, w_8 \text{ define the weighting coefficients and} \\
& r_{x,i}, r_{y,i} \text{ define the reference trajectory at a certain time step.}
\end{aligned}
$$

After computing the control inputs for the generated trajectories, the inputs are fed into a second optimization problem (see Equation (5.6)). This evaluates the given inputs and selects the optimal trajectory by penalizing lane changes (P^l), proximity to obstacles (P^o) and, overall, the sum of control inputs (P^u), whereas it rewards the covered track defined by the control inputs (P^c).

The result of the second optimization problem (Equation (5.6)), is an index, specifying the optimal trajectory at the current time step defined by a set of control inputs.

Afterwards, the first input of the corresponding optimal trajectory is proceeded to the vehicle model.

$$\underset{j \in J}{\arg\min} \quad w_a \cdot \underbrace{P^o}_{\text{prox. obst.}} + w_b \cdot \underbrace{P^u}_{\text{input sum}} + w_c \cdot \underbrace{P^l}_{\text{lane sw.}} - w_d \cdot \underbrace{P^c}_{\text{cov. track}}, \tag{5.6}$$

where J is the set of all computed trajectories and
w_a, w_b, w_c and w_d describe the weighting coefficients.

The weighting coefficients can be configured to suit the specific context and purpose of the application. The described procedure is repeated in each time step, until the vehicle reaches the goal position. In each time step at most three trajectories are constructed and corresponding optimal inputs are generated in the optimization problem to be evaluated and given to the vehicle model. The vehicle model, then executes one time step and proceeds an updated system state to the controller [LT18]. The MPC approach in automotive vehicle control has shown to generate optimal results regarding efficiency and driving comfort, therefore a later chapter concludes about verification possibilities and current issues in verifying a MPC-based autonomous vehicle controller (see Chapter 6).

5.7 Hybrid Maneuver Automaton

In this section we present our own prototypical implementation of a maneuver automaton for autonomous vehicles. In Section 5.1 a maneuver automaton is already presented which verifies the concatenation of motion primitives by connecting offline computed sets of reachable states. However, in many situations a more dynamic controller is required to cope with potentially dangerous situations.

In order to work on this issue, a modified maneuver automaton is presented. The automaton works as a closed loop between a controller and a vehicle model.

We modeled the automaton with two locations to represent the controller and the vehicle each in a specific location in the hybrid automaton, additionally the vehicle's variable set is integrated in the hybrid system's variable set. As a consequence the controller can directly change the input variables of the vehicle to adjust the variable derivatives. By computing the set of reachable states of the system, this modeling aspect allows us to analyze and verify the safety of the evolution of the vehicle's position induced by the controller. The variables of the system depend on the utilized vehicle model (Chapter 4). Additionally, a clock and a maneuver mode variable are included in the variable set of the system. The clock variable is required to specify the time the control stays in each location and the maneuver variable is used to specify the current active maneuver.

Currently, the controller works according to the following procedure. A motion planner pre-computes a vector which defines a certain maneuver id for a specific time step. The complete maneuver vector describes a complex trajectory consisting of single maneuvers executed successively. This specific implementation design allows the definition of a pre-computed input set to define the maneuvers the vehicle model is supposed to execute at a discrete time.

Precisely, a motion planner creates an ordered sequence of different motion primitives and transforms them to the described maneuver vector which is given to the

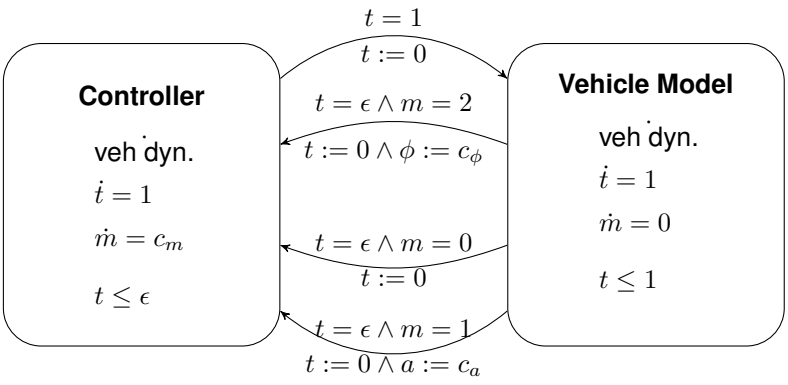
maneuver automaton. The generation of maneuver specific valuation changes are realized in the automaton by reset functions of maneuver specific transitions, hence the motion planner works with a simplified concept only defining the list of maneuver id's instead of the complete control input set, defining the trajectory. An exemplary input vector consists of a lane follow for the first ten time steps, followed by a lane switch to the left initiated at the 11^{th} time step. After completing the lane switch, the automaton returns to a follow mode and finally at time step 25 a deceleration is initiated. The trajectory ends after 30 time steps. The corresponding vector is defined in Equation (5.7).

$$\vec{inp} = [\underbrace{0,\ldots,0}_{\text{i=1,}\ldots,\text{10}},2,\underbrace{0,\ldots,0}_{\text{i=12,}\ldots,\text{24}},4,\underbrace{0,\ldots,0}_{\text{i=25,}\ldots,\text{30}}] \tag{5.7}$$

Initially, the automaton receives the current vehicle state as an input, defining the position and physical variable valuation of the vehicle. Thereafter, the maneuvers defined by the input array are executed by taking maneuver mode restricted transitions to the vehicle location. Afterwards, the automaton stays for one time step (1s) in the vehicle location, while evolving the position and variable set of the vehicle. Consequently, the system has two locations: one for the vehicle model and one for the controller.

The edges between the two locations are restricted by specific guard sets combined with reset functions.

Specifically, the guard sets are implemented by constrained hyperplanes determining variable regions in which a certain edge can be taken. We decided to use constrained hyperplanes because they showed to increase the numerical stability of our system. The guard sets are restricted by the maneuver mode variable, the clock and context specific variable valuations. In addition to the guard set, each edge consists of a reset function which modifies the systems' variables according to the current maneuver mode. After executing a maneuver successfully the maneuver mode is reset to a lane following mode $(m = 0)$ until the input vector adjusts the valuation.
The formal description of the hybrid automaton is graphically depicted below:



Provided with the described modeling design, the automaton can perform an acceleration or deceleration up to a desired velocity, a lane change to a neighboring lane or a change of the current heading angle to a certain value c_{v_ϕ}.

The advantage of the approach is to have a closed loop of the controller and vehicle model in one formal specification as a hybrid automaton. This allows us to

verify the complete controller structure with a flowpipe-construction-based reachability analysis. In each step, the set of reachable states of the corresponding system is computed and verified, assuming the real vehicle behaves as simulated. Afterwards, the computed inputs can be given to the physical vehicle. Resulting from the design of the hybrid system the maneuver mode variable m controls which maneuver is executed. In detail, the change of the maneuver mode m initiates the execution of a specified procedure of control inputs. Complex trajectories are realized by executing single maneuvers successively. This is implemented by a discrete time step specific array, assigning each time a certain maneuver mode value. The complete automaton is modeled in Figure A.2 in the Appendix A.6.

In the following, the basic procedure of the implemented maneuvers is explained:

- The first case m equals null depicts a lane follow, in which the current acceleration and angular velocity do not change and the vehicle follows the current dynamics.
- Secondly, setting m to one is the acceleration/deceleration mode, smoothly changing the acceleration value a to a certain value c_a. In particular, c_a can be set to a defined value by an external input.
- Moreover, performing a lane change to the left lane is initiated by m equals two. However, a lane change maneuver does not consist of only one variable modification, a smooth transition between two lanes is realized by a dynamic variation of the heading angle. Beginning with an increase of the heading angle up to the state in which half of the desired y-value is reached, followed by a negative angular velocity until the vehicle arrives in the goal lane. In theory, a help mode m equals 20 is introduced, executed if the heading angle attains a certain value, and eventually setting the angular velocity ϕ_v to the negative counterpart. In a last transition taken, if the heading angle is close to pointing in the x-direction, the angular velocity is reset and the maneuver is changed to the default follow mode.
- In the same manner, the lane change to the right lane is defined ($m = 3$), first by decreasing the angle up to a certain value and increasing it afterwards back to the initial valuation.
- In addition m equals four, initiates a velocity change up to a defined value. This is realized by an adjusted jerk value increasing or decreasing the acceleration a until the vehicle reaches the desired velocity.
- A needed option to cope with curves and required direction changes is to dynamically adjust the angle of the vehicle to a certain value. Finally, mode five introduces this option to initiate a smooth change of the heading angle. The maneuver intuitively increases or decreases the angular velocity until the desired heading angle is reached. The difference to the maneuver for a left and right lane change is the option to dynamically specify an intended heading angle, whereas the lane switch maneuver focuses on using the heading angle change to move the vehicle to a neighboring lane.

Further, Chapter 7 displays various numerical experiments which analyze the performance of the presented maneuver automaton. The experiments consist of single executions of all maneuvers and an additional complex trajectory realized by a sequence of maneuvers to handle a specified traffic scenario.

Chapter 6

Verification

The increased application of autonomous vehicle control in real traffic situations has resulted in an increased demand for the verification of autonomous vehicle controllers. As a matter of fact, the collision rate in traffic situations has to be minimized. This can be achieved by the development of prospective controller models with regard to the generation of safe inputs. Therefore the verification of controller models and path computations has to be further enhanced and improved to be even safe in unexpected situations and in the case of sensor faults. Different approaches regarding faulty sensor inputs have already been stated and successfully tested in numerical experiments [ZZMM13].

However, unexpected behavior of other traffic participants is in general unpredictable, therefore the only option to increase safety in case of unexpected movements of vehicles is to increase the safety distance between vehicles. Exceptions are motion planners for groups of cooperative vehicles which can control more parts of the environment which decreases the probability that we are confronted with unexpected behavior (as in [MLA17]).

Until now, one common option to handle unexpected traffic behavior is to consider uncertainty in the current vehicles position which leads to high safety offsets between moving traffic participants. As a result of this, the operator's vehicle and the corresponding controller have a higher probability to react and plan an evasive maneuver in time. On the other hand, the high off-sets result in unnecessarily imprecise trajectories. Consequently, the generated maneuvers have an increased probability to be misclassified as unsafe because the over-approximated flowpipe intersects with unsafe states in situations where a precise representation would be declared as safe. Hence, we have to select an appropriate initial set size to cover uncertainties in a realistic and safe manner [AD12].

Despite of this, the mentioned negative effect that maneuvers are misclassified as unsafe can be reduced by taking the probability to reach a certain state into account, further by including not only uncertain inputs and noisy sensor inputs, but as well probabilities of a certain behavior of traffic participants. Consequently, the probability of a collision between two objects in the current scenario is considered while classifying the safety status of the inputs [ASB09]. As far as we know, there are no other effective concepts to cope with unexpected behavior of other traffic participants. With this in mind, we focus on the controllable parts of the safety verification of autonomous vehicle controllers which are the inputs generated by the controller

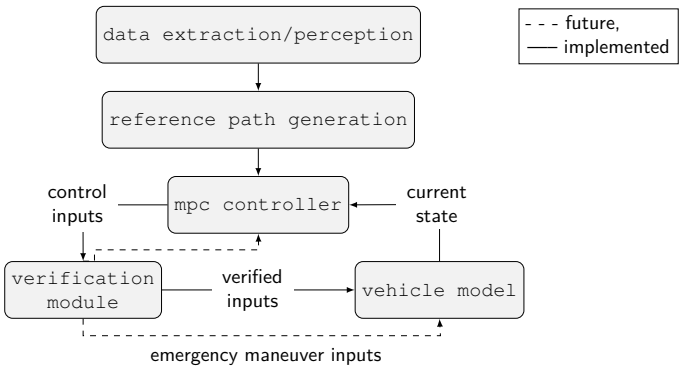


Figure 6.1: Integration of the verification module into the MPC structure.

and their effect on the future vehicle position.

After presenting various research approaches regarding controller models and especially verification options, this chapter concludes about the previous parts of the thesis and presents the implementation of a prototypical verification module.

The verification module is placed after the controller to enable the verification of inputs generated by the corresponding controller model (see Figure 6.1). Further, the verification module adapts the current vehicle dynamics and computes the set of reachable states regarding the given inputs. The resulting state sets are compared to the present obstacle sets by checking if the two sets intersect at some point. If there is no intersection, the inputs are classified as safe and passed to the vehicle model. However, if there is a non-empty intersection, the inputs are marked as malicious. In this case the module returns the inputs back to the controller with a feedback information about the intersecting region.

In effect of the feedback information, the controller could increase restrictions regarding the safety of the constructed trajectory and recompute the control inputs. In case of a repeated failure to generate save inputs, the verification module could initiate an evasive emergency maneuver.

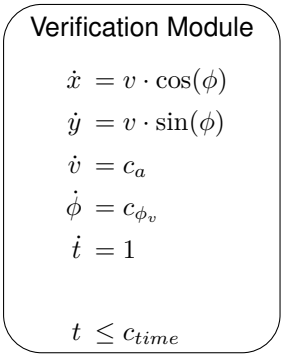
Under the context of this thesis the module is used in later benchmarks to verify inputs generated by the previous described MPC-based controller [LT18].

The previously mentioned additional features about the functionality of the module to execute an evasive maneuver are not part of the thesis because we focus on the computation and verification of the computed flowpipe. For further information about the construction of an evasive maneuver and the last possible time to execute it, we refer to [SOB06].

The verification module is implemented in Matlab using the toolbox Cora [AK18]. The invariant in the automaton's location is defined by a convex polytope, whereas single guards are defined by half-spaces and guards with more than on restriction are defined by constrained hyperplanes. Similarly to earlier computations of sets of reachable states, we used zonotopes as state set representation in the flowpipe computation.

6.1 Verification Module

The verification module is modeled by a hybrid automaton, which consists of one location and variables covering the vehicle model specific dynamics. A graphical representation of the verification module is displayed below, the formal specification is listed in the Appendix A.5.



The variable c_{time} is the time duration over which the inputs are intended to be verified. The variables v, ϕ change according to the control inputs c_a and c_{ϕ_v}.

Besides the stated system structure, the verification module has different parameters to adjust certain options of the reachability analysis. The time step size can be adjusted to fit the intended precision, the same goes for the state set representations. Moreover, the time duration in which the given inputs are evaluated can be adjusted to fit the intended purpose and the initial set size can be expanded by a factor ϵ.

The initial set is centered around the initial state of the vehicle given to the verification module $[x_0, \ldots, x_{n-1}]$ and expanded at both sides by ϵ, formally depicted below:

$$x_0, \ldots, x_{n-1} \in [x_0 - \epsilon, x_0 + \epsilon], \ldots, [x_{n-1} - \epsilon, \ldots, x_{n-1} + \epsilon].$$

In real applications the initial set size has to be set to the physical size of the underlying vehicle to allow a realistic flowpipe construction and its verification. In further experiments depicted in this thesis, we varied the initial set size to illustrate the capabilities of the verification module.

Together with the current vehicle state the controller proceeds the computed control inputs to the verification module. Afterwards, the verification module computes the corresponding set of reachable states for a determined time duration with flowpipe-construction-based reachability analysis. In order to verify the safety of the inputs, an intersection between the resulting set of reachable states and the occupancy sets of obstacles in the close environment at the current discrete time are computed. As previously mentioned in previous verification methods, if there is no resulting intersection the controller's evolution is concluded to be safe. Consequently, the inputs are verified to be safe and passed to the vehicle model. In contrast, if parts of the computed sets of reachable states intersect with unsafe states, the system is possibly unsafe. Due to the over-approximative character of the flowpipe construction, we can not distinguish between an upcoming collision or a safe path falsely

classified as risky because of an imprecise over-approximation. Accordingly, the inputs can be nevertheless safe. To minimize this case, the state set representation and time step size could be varied to increase precision of the resulting flowpipe, as demonstrated in [SA18]. In case that the intersection is still not empty, the inputs are stated to be unsafe and the verification module returns them to the controller with feedback information about the risks' origin (see Figure 6.1). Consequently, the controller knows where to restrict constraints even further or when to initiate an evasive emergency maneuver.

In the following we describe a certain procedure which does not necessarily increase the safety level of the controller, instead it solely provides a possible reduce of damage in case of a communication error between controller and verification module.

The computed time duration in which the given inputs are evaluated is depended on the desired verification of possible redundant inputs. In the following we refer with time period to the time duration the inputs constructed in the trajectory planner are supposed to be executed. If the time duration verified by the verification module is chosen higher than the time period, the module analyses inputs which are in most cases redundant.

Before describing this further, we have to make a controller specific case distinction. In case that the controller works similar to the MPC-based controller presented in this thesis, the inputs consist of many input values describing a trajectory for the next time periods, whereas only the first time period and corresponding input values are intended to be executed. If the number of considered time periods is increased, the verification module can verify inputs which are not primarily intended to be performed. In particular, the next $\delta \in \mathbb{N}$ time periods verify the next δ inputs, which are only executed if the controller cannot generate new inputs due to a system fault. In case, that the controller cannot communicate with the system anymore and previously computed control inputs are already verified and stored in the verification module, they can be executed safely.

On the other hand, if the controller does generate only inputs for the next consecutive time period, the increase of considered and verified time periods results in stronger safety guarantees. Under those circumstances, an increase of the considered time periods leads to an expanded analyze of the trajectory, a decrease of considered time periods leads respectively to a reduced trajectory analysis. If the time duration defined by the verification module is chosen larger than the time period considered by the path planner, we verify the motion the vehicle probably will not reach before getting new input values. The effectiveness of this has to be evaluated for each context. However, the increase of computed time periods in the reachability analysis improves the safety of the controller in the case that the communication in the closed loop environment breaks down. The increased time duration verified by the system allows the verification module to recognize at which discrete time the controller should initiate an evasive maneuver leading to the safe stop of the, car instead of entering potentially malicious regions.

Furthermore, to improve safety in system critical states, C.Schmidt, F.Oechsle and W.Branz provided detailed information of emergency maneuvers to be executed in situations where no safe maneuver is feasible [SOB06]. Technically, the process of planning an evasive maneuver is part of the motion planner. An approach presenting an advanced path planner generating inputs for the next time step while computing inputs for an emergency maneuver in each time step is analyzed and tested in [MA16].

The described *fail safe* property of the system should be implemented in all controller models used in real traffic situations to increase the level of safety. Therefore, the described property could be implemented in a simplified version into the verification module.

In comparison to the maneuver automaton in Section 5.7, the presented module differs in verifying only the inputs generated by an external controller, instead of taking over parts of the motion planner by generating verified inputs for certain maneuvers.

Additionally, another difference between the implemented approaches is that the trajectory constructed by the automaton does not have to be optimal, whereas the inputs generated by the MPC-based controller are in theory always optimal regarding driving efficiency and comfort.

Overall, the implemented maneuver automaton has the advantage that the complete closed controller vehicle environment is included in the verification, however the verification module solely verifies the given inputs in a certain period. At the same time the maneuver automaton cannot adapt all situations in the intended manner because it works with motion primitives, which restrict the set of possible trajectories to a discrete set of available actions. Conversely, the MPC-based controller can react to a wide range of situations with a dynamic path update considering the future evolution of the environment to make prospective decisions regarding the intended trajectory.

In conclusion, both implementations have certain advantages and disadvantages while the verification module adapted to the MPC-based trajectory planner provides more conclusive and decisive results. Additionally the following section evaluates the verification module in certain numerical experiments.

6.1.1 Numerical Experiments

Finally, to test the performance of the presented approach, several numerical experiments of the verification module adapted to the MPC-based controller are shown [LT18]. In particular, the performed experiments present the flowpipes generated by the verification module for 30 consecutive input verification's, depicting a simple lane follow maneuver constructed by the underlying MPC-based controller.

The first three images show the intern calculated sets of reachable states, resulting out of the first 30 inputs generated by the MPC-based trajectory planner (Figure 6.2). The chosen time duration the inputs are evaluated over is 0.10s, similar to the time duration the MPC-based controller intended the inputs to be executed. Moreover, the factor ϵ which expands the initial set size is set to 0.1. The processing time for the first 30 time steps was 26.91s.

Next, different ϵ valuations are displayed in Figure 6.3, varying the initial state set size of the set of reachable states. Therefore the flowpipe constructed in the left figure is much wider ($\epsilon = 0.5$) than the one in the right figure ($\epsilon = 0.05$). The verification module allows a simple adjusting of the intended initial set size by varying the ϵ value in the option parameters. Consequently, this modification allows to analyze different inaccuracies in the initial position of a trajectory. In addition to this, an incremented ϵ value expands the size of the generated flowpipe to increase the desired safety distance between obstacles.

The results of the main part of the verification module to detect potential dangerous inputs is shown in Figure 6.4a, which displays the flowpipe of the driver's

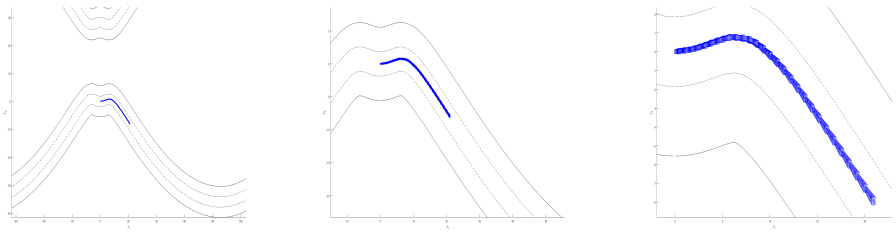
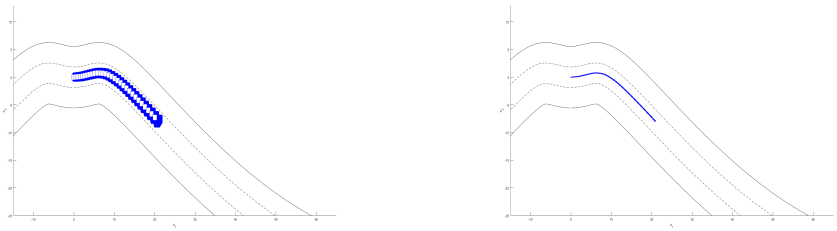


Figure 6.2: Zoom-in of a trajectory verification ($\epsilon = 0.1$).



(a) $\epsilon = 0.5$. (b) $\epsilon = 0.05$.

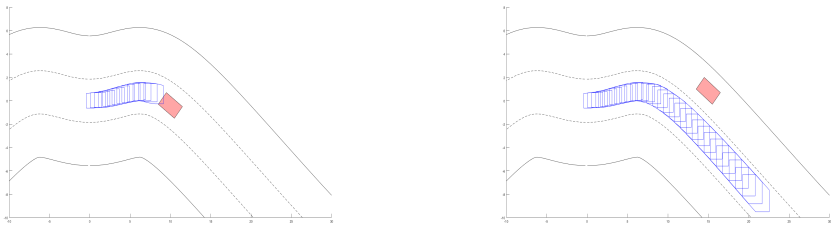
Figure 6.3: Different initial set sizes.

vehicle together with an obstacle blocking the road. At the 13^{th} time step the algorithm detects an intersection between the occupancy set of the obstacle and the set of reachable states computed from the initial set with respect to the provided inputs from the controller. As a consequence, the verification module informs the controller about the potential danger provoked by the inputs and stops the computation. Besides, the algorithm had a processing time of 11.76s to compute the first 13 time steps while verifying the possible trajectories constructed by the given inputs at each time step. Moreover, it took the algorithm 2.45s to classify the inputs as unsafe in the 13^{th} time step.

Vehicle-specific hardware is expected to improve this running time. However, it should be noted that the current processing time is not suited for an online applicable usage. Moreover, Figure 6.4b displays a map with a static obstacle next to a collision-free flowpipe. The verification module classified the situation correctly by verifying the inputs generated by the controller to be safe in each step, as illustrated in the complete trajectory shown in Figure 6.4b.

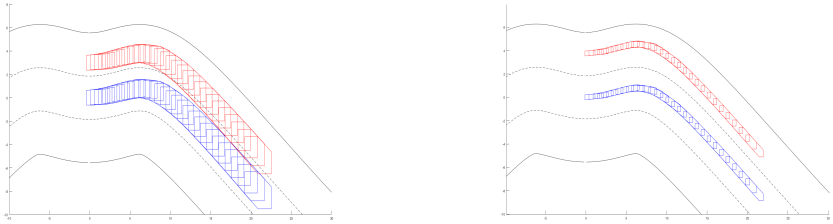
Additionally, Figure 6.5a shows a potential collision of two moving traffic participants. The situation is described by two vehicles driving next to each other on a three lane road. At a certain time, the vehicle on the left lane starts to move closer to the middle lane resulting in a possible collision in the 27^{th} time step. The algorithm detects the possible unsafe inputs after a running time of 40.58s. In the specific period, the algorithm had a computation time of 2.29s to classify the inputs as unsafe.

Secondly, Figure 6.5b shows the traffic situation of two vehicle driving next to each other on the same three lane road. The plot depicts the flowpipe of the drivers vehicle next to the occupancy set of a moving obstacle. Owing to the fact that the set of reachable states does not intersect at any time step with the occupancy set



(a) Collision with a static obstacle. (b) Collision free trajectory.

Figure 6.4: Intersection vs. no intersection ($\epsilon = 0.3$, static obstacle).



(a) Collision with a dynamic obstacle ($\epsilon = 0.3$). (b) Collision free trajectory ($\epsilon = 0.1$).

Figure 6.5: Intersection vs. no intersection.

of the dynamic obstacle, the verification module classifies the inputs in each of the 30 steps as save and forwards them to the vehicle model. The computation time it took the algorithm to check for an intersection after each new input generation was 47.61s.

In addition to the previous illustrated parameter configurations, a modification of the time duration is shown in Figure 6.6. The time duration defines the span over which the given inputs are evolved in the systems' dynamics. Hence, if the inputs are processed longer than intended, the resulting flowpipes show future positions reached, if no new input is received. This is solely relevant, if the communication between controller and vehicle is disrupted. As a result, the verification module has already verified further time steps of the current inputs and can anticipate when a risky situation approaches. Consequently, an emergency maneuver can be initiated, if the specific situation is classified as dangerous. In an error-free environment, it has shown to be the best choice to adapt the time duration to the by the controller supposed execution time with a small offset value. The small offset value is chosen to expand the flowpipe to cover certain states reached if the given inputs are executed longer than intended induced by numerical issues.

Further information on the computed running time can be found in Appendix A.2. All computations in this section were performed on a mid class computer with a 2.80GHz six-core i5 processor.

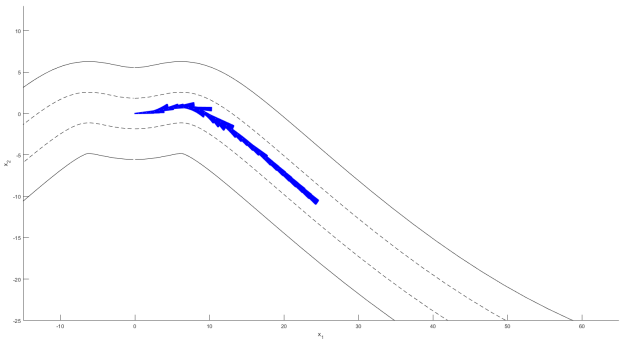


Figure 6.6: Increased time step size ($0.5s$).

6.2 Safety verification of an MPC controller

During our work with MPC-based trajectory planning and the verification thereof, we came across several open problems and challenges which need to be addressed by the research community in order to pave the way towards safety verification of MPC-based controllers for autonomous vehicles.

We choose an approach to verify a controller model by transforming it into a hybrid automaton to further allows its verification over a defined time duration. Therefore the complete decisive behavior of the controller together with the exact dynamics of the vehicle model has to be transformed into a hybrid system. After transforming the controller and the corresponding vehicle model into a hybrid system, flowpipe-construction-based reachability analysis can derive every state reachable from the initial system state within bounded time to verify if the set of reachable states is not intersecting with as malicious classified states.

The transformation of the presented vehicle models into a hybrid automaton is an intuitive process and solved by implementing the vehicle specific dynamics into the location specific activities. This is possible for all presented vehicle models and theoretical vehicle models which are not part of this thesis. In case of a non-linear vehicle model, such as the kinematic single track model, the bicycle model or any other high order model, the computation of the flowpipe requires certain methods to cover the non-linearities of the underlying system.

The transformation of an MPC-based controller together with a corresponding vehicle model into a hybrid system could verify the complete closed-loop between controller and vehicle model while computing successive sets of verified trajectories.

However, the transformation of the controller into a hybrid automaton is hindered by a few challenges. First of all, flowpipe-construction-based reachability analysis operates on sets of trajectories while the MPC-based controller works solely with single trajectories. This behavior is reflected in an optimization problem solved in each iteration of the MPC-based controller.

In the current implementation, the dynamics of the system compute an equal flow for each point in the initial set, taking the center point as reference point to be fed into the optimization problem of the MPC. This can lead to unsafe states because safe inputs generated from one input point do not have to be necessarily safe for all

points in the set.

Therefore the MPC approach has to be enhanced to work on sets of inputs, instead of single points. However, we do not know, if there is an algorithm to realize this procedure to advance an MPC-based controller to work on sets.

The currently implemented procedure to compute an equal flow for all given points is our own substitute approach, which is solely used as a workaround.

Moreover, the in Matlab used toolbox Cora [AK18], which is used throughout this thesis, has shown technical issues in parts of the implementation based on missing numerical stability of the algorithm which is required to be reduced to have a robust method to verify the underlying system. This concern does not only affect the named toolbox, in fact numerical stability has to be ensured in all situations and in all implementation environments. To work on this, one option to prevent numerical issues is implementing adapting bounds and relaxing constraints as showed in Section 5.6. However, to increase numerical soundness and minimize faulty data induced by numerical issues, advanced and robust analysis methods have to be implemented.

Finally, we focus on a practical aspect. The verification of the MPC has to work in an online applicable time, to enable the controller a decisive time amount in case an evasive maneuver has to be initiated. As the verification of inputs generated by an automotive vehicle controller is highly connected to the safety of the passengers, the inputs have to be verified quickly to enable time for alternative computations in case of potential danger. Currently, the time complexity of the verification module is not suitable for an application in real scenarios which require direct results. Thus, improvements of the time complexity of the module have to be focused on in future developments. These advancements can be realized by varying the used state set representations to fit the current purpose of a situation, which means using simple representations in low-risk situations and precise state set representations in potentially dangerous environment parts. Moreover, it has to be addressed that the numerical experiments performed in this thesis ran on a mid-range computer, further running time improvements are expected on a specific hardware component.

In conclusion, the mentioned points have to be further advanced and corresponding problems have to be solved to enable the transformation of a MPC-based trajectory planner into a hybrid system and allow its verification.

Chapter 7

Numerical Experiments

After we implemented the presented controller models, they have to be analyzed in various numerical experiments. In Section 4.4, vehicle models have already been compared with varying parameters to compare the different representation possibilities and the resulting flowpipes.

We evaluate our implementations with several numerical experiments. We begin with the evaluation of the hybrid maneuver automaton (Section 5.7) by computing several single maneuvers and a complex collision avoidance maneuver. Afterwards, the verification module in combination with the simplified MPC-based controller (Section 5.6) is evaluated in a benchmark scenario.

All computations in this section were performed on a mid-end computer with a $2.80\mathrm{GHz}$ six-core i5 processor. Moreover, the reachability analysis in each scenario has been computed with a factor $\epsilon = 0.00005$ increasing the initial set size and a time step size of $0.02\mathrm{s}$.

7.1 Hybrid Automaton - Experiments

Before evaluating the automaton in a traffic scenario requiring the concatenation of several maneuvers, all maneuvers are presented in a single execution.

The chosen initial valuation of the vehicles state for each numerical experiments is selected to solely focus on the evolution of the by the current maneuver adjusted variable. Therefore maneuvers depicting a change of the driving direction are presented with a constant velocity and maneuvers focusing on modifying the velocity of the vehicle are performed with a constant driving orientation. Due to the modeling design of the automaton, the plots depict a period behavior based on the clock resets after each time step, rather than showing an overall evolution of the specific values while realizing a maneuver.

First of all, the lane follow maneuver is presented over 20 time steps with a constant velocity of $v = 1$. The lane follow describes a maneuver, in which the vehicle model does not receive any changes through inputs. Therefore the dynamics, including the velocity and heading angle stay constant during the execution of the maneuver. Figure 7.1a depicts the set of reachable states of the described lane follow. Further the plotted set of reachable states describes a trajectory from x null to the final position x equals 20. Omitting to the fact that the initial driving direction

points in x-direction the y value stays at zero. Moreover, Figure 7.1b visualizes the constant velocity of the vehicle over the executed time duration. The clock variable t stays in the range between null and one because after staying one second in the vehicle model location of the automaton, t is reset to null. This procedure which resets the clock in each period to null is repeated in each cycle between controller and vehicle location due to the modeling design of the underlying hybrid automaton. The execution time to perform the reachability analysis for the defined 20 time steps including the verification step was 11.45s.

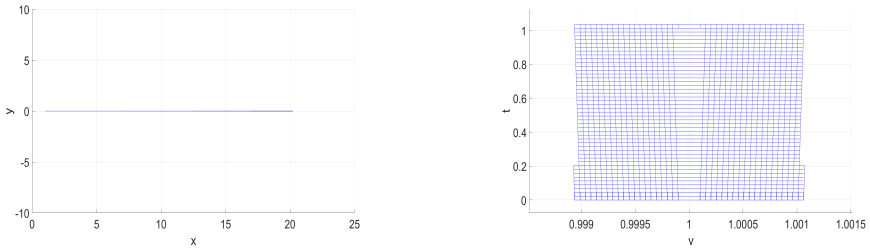
Secondly, a smooth increase of the vehicle's acceleration to a constant acceleration value of one is shown. The acceleration maneuver is executed over a time duration of 20s to have an insight into the velocity evolution of the vehicle. After reaching the desired acceleration value one, the automaton switches to an execution of the lane follow maneuver for the remaining seconds. Simultaneously to realizing the accelerating of the vehicle, the heading angle stays constant. This explains that the set of reachable states only evolves in x-direction as in the previously evaluated follow maneuver (see Figure 7.2a). In comparison to Figure 7.1a the flowpipe expands slightly reasoned by the fact that the high velocity change enlarges the set of reachable positions. Moreover, it should be noted that the deviation in y direction is negligible as it is comparatively small.

The second plot, Figure 7.2b depicts the evolution of the velocity over time. The increased velocity shown in the plot of Figure 7.2b is explained by the acceleration inputs given to the vehicle model during the run of the hybrid system. In each step of the previously described cycle, the derivative of the velocity is increased. This is reasoned by the fact that the automaton increases the acceleration input value in each period run by a constant value of 0.10. The increase of the velocity derivative is graphically depicted in Figure 7.2b by the lines describing the velocity change in each period which slightly tend to the x-axis. Due to the fact that after finishing the acceleration task, the acceleration value is set to null, the following successive line elements in the plot are parallel. The hybrid system had a running time of 17.03s to construct the flowpipe of the trajectory describing the desired acceleration change.

Another approach to perform a velocity change is realized by another acceleration maneuver. This maneuver focuses on reaching an intended velocity value c_v, instead of a certain acceleration value. Figure 7.3a displays the trajectory of the vehicle following the lane according to the initial heading angle for 20s, while accelerating until the velocity v equals three. Additionally, Figure 7.3b displays the velocity value over time. Due to the defined goal velocity, the speed value is increased in each cycle until it stays constant at the final valuation equaling three, neglecting a small numerical error. The computation time to construct the set of reachable states displaying the acceleration of the vehicle from $v = 1$ to $v = 3$ was 69.59s (see Figure 7.3a).

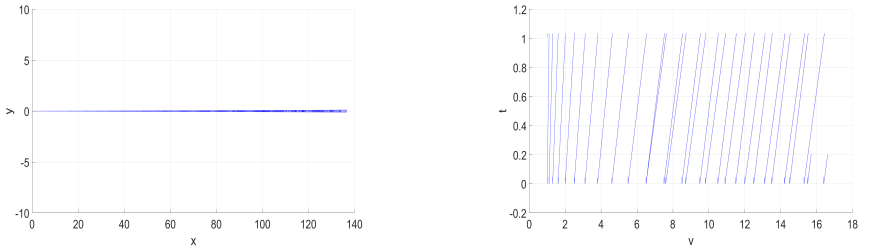
Moreover, the two lane switch maneuvers are evaluated and presented in a single execution. They consist of an initial increase of the heading angle until a certain valuation is reached. Afterwards, the heading angle is decreased to realize a smooth curved trajectory. The maneuver begins at the initial coordinates x and y equaling null and ends at the final position of x equaling 15 and y equaling three. The increase of the heading angle is realized by a small input value adjusting the angular velocity of the vehicle. After a certain angle is reached the trajectory bends (see Figure 7.4a) and respectively the angle is decreased by the negated angular velocity until the vehicle is orientated in x-direction again (see Figure 7.4b).

Figure 7.4a shows the set of reachable states of the lane switch trajectory for the



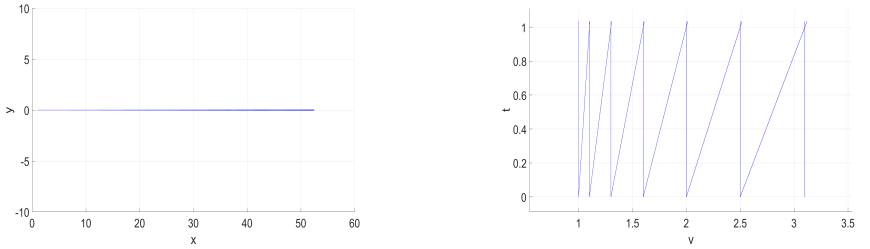
(a) Coordinates (x,y). (b) Velocity v.

Figure 7.1: Follow mode $(m = 0)$.



(a) Coordinates (x,y). (b) Velocity v.

Figure 7.2: Acceleration mode $(m = 1)$.



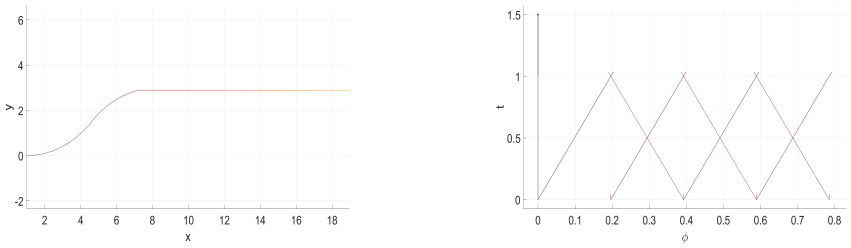
(a) Coordinates (x,y). (b) Velocity v.

Figure 7.3: Velocity change mode $(m = 4)$.

described initial and final valuation. The final y value targeted by the maneuver can be modified to fit the current environment and lane properties. Figure 7.4b displays the evolution of the heading angle over time. Due to the fact that the plots are color coded the increase of the heading angle as well as the following decrease is depicted in both figures.

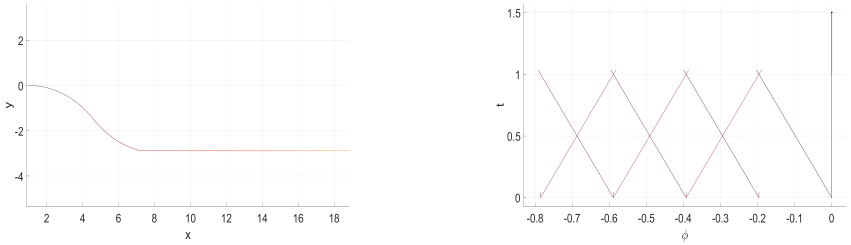
The computation time of the automaton to construct the flowpipe, displaying the corresponding lane change maneuver was 11.72s for a time duration of 20s.

The lane change to the right neighboring lane works analogously, the angular velocity is set to a negative value decreasing the angle up to a certain limit and afterwards increasing it to realize a smooth curve (see Figure 7.5). The running time to perform the reachability analysis resembles the one of the left lane change



(a) Coordinates (x,y). (b) Heading angle ϕ.

Figure 7.4: Left lane change ($m = 2$).



(a) Coordinates (x,y). (b) Heading angle ϕ.

Figure 7.5: Right lane change ($m = 3$).

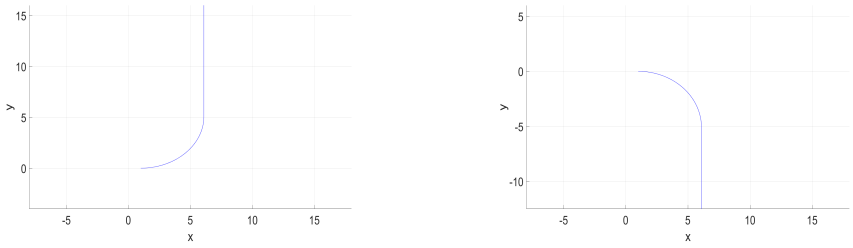
maneuver.

Further, to define maneuvers at road intersections, the change of the heading angle when taking a turn has to be displayed correctly. Therefore we implemented a set of edges in the maneuver automaton to realize a right or left turn trajectory. The reset functions of the corresponding transitions increase the input values defining the angular velocity of the vehicle until the intended driving direction is attained. At the point the vehicle is oriented in the intended direction the angular velocity is reset to null, and the controller executes the default follow mode. The automaton's formal specification allows it to define the final driving direction by an input parameter c_ϕ. Figure 7.6a, displays the set of reachable states of the left turn trajectory, respectively Figure 7.6b presents the flowpipe describing the right turn of the vehicle. The hybrid automaton had a computation time of 11.84s to construct the set of reachable states representing the intended angle variations computed over a time duration of 20s.

The previously described evaluations of the automatons performance, showed the important maneuvers required to build complex and decisive trajectories.

The analysis of the computation time shows that the automaton has a high time complexity to generate verified inputs for autonomous vehicle control. Therefore the current approach has to be further evaluated under different conditions to analyze the performance with specified hardware and various time improving implementation changes.

Further, to perform more complex maneuvers and generate trajectories in realistic traffic situations, different motion primitives are executed in succession. A complex trajectory avoiding certain static obstacles has been performed (compare Figure 7.7). Initially, the driver's car is on the middle lane in a three lane road. In front of the opera-



(a) Left curve. (b) Right curve.

Figure 7.6: Curved trajectories.

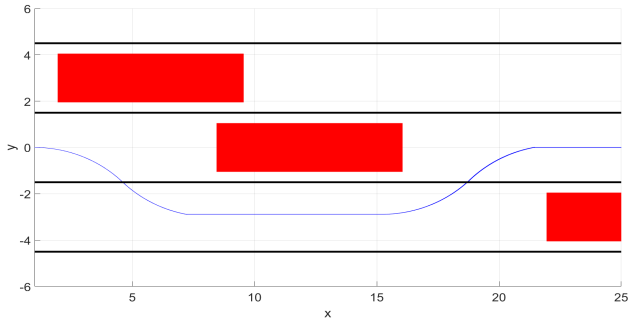
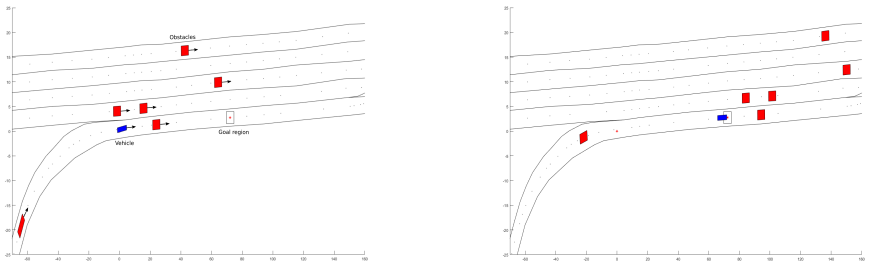


Figure 7.7: Obstacle avoidance maneuver.

tor's vehicle there is a static obstacle that has to be avoided. Additionally, in the right lane another static object is placed that has to be bypassed by the driver's vehicle. In a practical application, a dynamic motion planner has to construct a trajectory by defining a sequence of maneuvers which build a safe path passing both obstacles without colliding. However, in the context of this thesis, we statically built an input vector defining the maneuvers that have to be computed successively to drive safely through the obstacle environment. Therefore, we choose an initial lane follow for the first seconds followed by a lane change to the left lane. After that, we follow the left lane for 10 seconds and perform a consecutive lane change to the middle lane. Figure 7.7 shows the resulting flowpipe the reachability analysis has computed. The computation of the complete maneuver had a running time of 15.80s to generate the inputs in each time step and verify the resulting trajectories. Further experimental data of the running time can be found in Appendix A.3.

7.2 Verification Module - Benchmark



(a) Initial position. (b) Goal position.

Figure 7.8: Benchmark initial and final position (benchmark taken from [AKM17]).

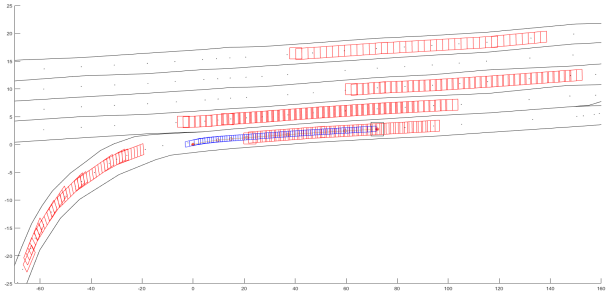


Figure 7.9: Reachable and occupancy sets (MPC algorithm combined with the verification module).

In addition to performing various numerical experiments on the implemented maneuver automaton from Section 5.7, the verification module is evaluated in combination with a simplified implementation of the MPC-based trajectory planner, originally derived in [LT18].

The environment of the benchmark scenario is taken from the Commonroad project [AKM17]. We chose a scenario with six obstacles and one operator vehicle with defined initial and goal coordinates (see Figure 7.8a). The controlled vehicle drives on a driveway to a highway road with four lanes. Additionally, one obstacle vehicle is driving on the same lane with a certain distance behind the operator's car. Moreover, two vehicles block the lane left to the driver's lane and one is driving ahead of the controlled vehicle. The other two obstacles are on the second and fourth lane and can be neglected because they do not influence the close environment of the vehicle.

This scenario was fed into our implementation of the simplified version of a MPC-based controller (Section 5.6), which constructed a trajectory and corresponding control inputs for each time step. In each time step the verification module (see Chapter 6) computed the set of reachable states for the computed control inputs and

classified them as safe. Afterwards, the inputs have been given to the vehicle model which executed one time step of 1s. In a closed loop control these steps have been repeated until the goal location was reached. Figure 7.9 shows the set of reachable states computed during the execution of the verification module, where the blue sets display the set of reachable states of the driver's vehicle and the red flowpipes depict the obstacles' time specific occupancy sets. Important to note here is that it is no safety violation that the blue flowpipe overlaps with the red one because they overlap at different time steps.

Figure 7.8a depicts the initial position of the benchmark with arrows indicating the expected movement of the cars, whereas Figure 7.8b displays the final position of the traffic participants after performing the trajectory constructed by the controller in a closed loop with the verification module. The running time of the controller with the adapted verification module was 240.69s, with a time step size of 0.00175s and an initial size factor of $\epsilon = 0.3$.

Chapter 8

Conclusion

8.1 Summary

In this thesis we worked on applying and enhancing various formal methods to ensure the safety verification of autonomous vehicle controllers.

First we have depicted and analyzed different vehicle models in their ability to represent a realistic trajectory while having an acceptable complexity which fits our intentions. The results showed that the bicycle model is the optimal choice to represent a realistic trajectory due to the consideration of various physical and vehicle specific factors, while having a good trade off between complexity and expressiveness. However, the KST has proven to fit our requirements as a simplified representation applicable in the intended verification of closed controller vehicle environments.

Further, the analyze of various controller approaches in current verification and trajectory planners regarding autonomous vehicle control has revealed several findings. The work and methods based on the usage of motion primitives have shown promising results in constructing verified trajectories, however the resulting trajectories tend to be static and not optimal in regard of passenger comfort and driving efficiency.

Owing to the fact that we used hybrid systems as a formal specification of the controller structures, we depicted different verification methods to verify said systems. In context of autonomous vehicle control we decided to use flowpipe-construction-based reachability analysis.

As an improvement to the verified trajectories based on motion primitives, we decided to enhance the work on verifying dynamic path planner that construct trajectories which are optimal regarding certain factors. The implemented MPC-based controller model has demonstrated to be the most encouraging approach in trajectory planning for autonomously driving vehicles. Therefore we focused on the transformation of said controller into a hybrid system to enable its verification.

Due to certain issues regarding the transformation process, which have to be tackled by future works (see Section 8.2), we analyzed and depicted an overview of the missing requirements needed to perform the complete verification.

Finally, we elaborated a prototypical implementation of a maneuver automaton to illustrate a different concept in constructing verified trajectories with the help of a hybrid automaton. Additionally, a verification module is displayed and used to verify the inputs generated by the described MPC-based controller.

Both implementations have been evaluated in several numerical experiments, in which they succeeded to fulfill their purpose, while we analyzed future developments required to increase the safety and efficiency of the prototypical models, further illustrated in the following.

8.2 Future Work

During our research for this thesis, we have found several directions for future work. Certain topics and methods have to be further developed to be integrated in current automotive vehicle control and to improve the theoretical aspect of controller verification.

8.2.1 Verification Module Controller Interaction

In implementations and the application of the module in numerical experiments we analyzed certain parts which can be further enhanced. Mainly, the interaction and communication between the controller and the verification module can be improved by detailed feedback information given to the controller in case input values are classified as unsafe. This further development enables a more decisive generation of a new trajectory which avoids the unsafe states.

Additionally, the verification module could focus and expand it's capability, to construct emergency inputs in each computed time step in order to ensure a *fail-safe* property of the complete system.

8.2.2 Enhancements in the MPC Verification

The main missing requirement to completely transform the MPC-based trajectory planner into a hybrid system is the development of the MPC approach to work on sets of trajectories, instead of solely single trajectories. Therefore future research has to focus on this aspect to enable the verification of an autonomous vehicle controller based on model predictive control.

Moreover, to allow an online applicable usage of our implementations, the corresponding running time of either the verified MPC-based controller or the prototypical implemented verification module has to be improved in future work.

Finally, the in this thesis occurred technical issues in the implementation of certain verification approaches have to be minimized. Especially, the numerical stability of the flowpipe-construction-based reachability analysis has to be improved to have a robust verification method.

In future work the depicted points are expected to be eliminated by advanced methods.

Bibliography

[AD12] Matthias Althoff and John M. Dolan. Reachability computation of low-order models for the safety verification of high-order road vehicle models. In *2012 American Control Conference (ACC)*, pages 3559–3566, 2012.

[AFV95] Christodoulos A Floudas and Viswanathan Visweswaran. Quadratic optimization. *Handbook of Global Optimization*, 1995.

[AK18] Matthias Althoff and Niklas Kochdumper. Cora 2018 manual. *Technische Universität München, 85748 Garching, Germany*, 2018.

[AKM17] Matthias Althoff, Markus Koschi, and Stefanie Manzinger. Commonroad: Composable benchmarks for motion planning on roads. In *Proc. of the IEEE Intelligent Vehicles Symposium*, pages 719 – 726, 2017.

[Alt17] Matthias Althoff. Commonroad: Vehicle models. *Technische Universität München, Garching*, pages 1–25, 2017.

[ALU95] The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3 – 34, 1995.

[ASB09] Matthias Althoff, Olaf Stursberg, and Martin Buss. Model-based probabilistic collision detection in autonomous driving. *IEEE Transactions on Intelligent Transportation Systems*, 10(2):299–310, 2009.

[ASK+92] Richard Wade Allen, Henry T Szostak, David Klyde, Theodore Rosenthal, and Keith J. Owen. Vehicle dynamic stability and rollover. *NASA STI/Recon Technical Report N*, 93, 1992.

[BIS09] Martin Buehler, Karl Iagnemma, and Sanjiv Singh. *The DARPA urban challenge: autonomous vehicles in city traffic*, volume 56. Springer, 2009.

[BT97] Dimitris Bertsimas and John N Tsitsiklis. *Introduction to linear optimization*, volume 6. Athena Scientific Belmont, MA, 1997.

[BZS14] Philipp Bender, Julius Ziegler, and Christoph Stiller. Lanelets: Efficient map representation for autonomous driving. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pages 420–425, 2014.

[CAS12] Xin Chen, Erika Abraham, and Sriram Sankaranarayanan. Taylor model flowpipe construction for non-linear hybrid systems. In *2012 IEEE 33rd Real-Time Systems Symposium*, pages 183–192. IEEE, 2012.

[FDF05] Emilio Frazzoli, Munther A. Dahleh, and Eric Feron. Maneuver-based motion planning for nonlinear systems with symmetries. *IEEE Transactions on Robotics*, 21(6):1077–1091, 2005.

[Gir04] Antoine Girard. *Algorithmic Analysis of Hybrid Systems*. Theses, Institut National Polytechnique de Grenoble - INPG, 2004.

[Gir05] Antoine Girard. Reachability of uncertain linear systems using zonotopes. In *Hybrid Systems: Computation and Control*, pages 291–305. Springer Berlin Heidelberg, 2005.

[HA92] Yong K. Hwang and Narendra Ahuja. A potential field approach to path planning. *IEEE Transactions on Robotics and Automation*, 8(1):23–32, 1992.

[HAS14] Daniel Heß, Matthias Althoff, and Thomas Sattel. Formal verification of maneuver automata for parameterized motion primitives. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1474–1481, 2014.

[Hen00] Thomas A Henzinger. The theory of hybrid automata. In *Verification of Digital and Hybrid Systems*, pages 265–292. Springer, 2000.

[KA17] Markus Koschi and Matthias Althoff. Spot: A tool for set-based prediction of traffic participants. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 1686–1693, 2017.

[KGCC15] Soonho Kong, Sicun Gao, Wei Chen, and Edmund Clarke. dreach: δ-reachability analysis for hybrid systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 200–205. Springer Berlin Heidelberg, 2015.

[LG09] Colas Le Guernic. *Reachability analysis of hybrid systems with linear continuous dynamics*. PhD thesis, 2009.

[Löf04] Johan Löfberg. Yalmip: A toolbox for modeling and optimization in matlab. In *Proceedings of the CACSD Conference*, volume 3. Taipei, Taiwan, 2004.

[LT18] Seungho Lee and Hongtei E. Tseng. Trajectory planning with shadow trolleys for an autonomous vehicle on bending roads and switchbacks. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 484–489, 2018.

[MA16] Silvia Magdici and Matthias Althoff. Fail-safe motion planning of autonomous vehicles. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 452–458. IEEE, 2016.

[MGP13] Stefan Mitsch, Khalil Ghorbal, and André Platzer. On provably safe obstacle avoidance for autonomous robotic ground vehicles. In *Robotics: Science and Systems*, 2013.

[MLA17] Stefanie Manzinger, Marion Leibold, and Matthias Althoff. Kooperative bewegungsplanung autonomer fahrzeuge unter verwendung von manöver-templates. In *AAET-Automatisiertes und vernetztes Fahren*, 2017.

[PKY+09] Jaemann Park, D-W Kim, Yoongsoon Yong, H J Kim, and Kyongsu Yi. Obstacle avoidance of autonomous vehicles based on model predictive control. *Proceedings of The Institution of Mechanical Engineers Part D-journal of Automobile Engineering - PROC INST MECH ENG D-J AUTO*, 223:1499–1516, 2009.

[PQ08] André Platzer and Jan-David Quesel. Keymaera: A hybrid theorem prover for hybrid systems. In *International Joint Conference on Automated Reasoning*, pages 171–178. Springer, 2008.

[Raj11] Rajesh Rajamani. *Vehicle dynamics and control*. Springer Science & Business Media, 2011.

[RISA18] Albert Rizaldi, Fabian Immler, Bastian Schürmann, and Matthias Althoff. *A Formally Verified Motion Planner for Autonomous Vehicles: 16th International Symposium, ATVA 2018, Proc.*, pages 75–90. 2018.

[RS94] Dirk Reichardt and Jeong Shick. Collision avoidance in dynamic environments applied to autonomous vehicle guidance on the motorway. In *Proc. of the Intelligent Vehicles '94 Symposium*, pages 74–78, 1994.

[SA18] Stefan Schupp and Erika Ábrahám. Efficient dynamic error reduction for hybrid systems reachability analysis. In *(TACAS'18)*, LNCS. Springer, 2018.

[SAC+15] Stefan Schupp, Erika Ábrahám, Xin Chen, Ibtissem Ben Makhlouf, Goran Frehse, Sriram Sankaranarayanan, and Stefan Kowalewski. Current challenges in the verification of hybrid systems. In *(CyPhy'15)*, volume 9361 of *Information Systems and Applications, incl. Internet/Web, and HCI*, pages 8–24. Springer, 2015.

[SK03] Olaf Stursberg and Bruce H Krogh. Efficient representation and computation of reachable sets for hybrid systems. In *International Workshop on Hybrid Systems: Computation and Control*, pages 482–497. Springer, 2003.

[SOB06] Christian Schmidt, Fred Oechsle, and Wolfgang Branz. Research on trajectory planning in emergency situations with multiple objects. In *2006 IEEE Intelligent Transportation Systems Conference*, pages 988–992, 2006.

[Sof11] MSC Software. Adams/tire help. Technical report, Adams (MD 2011) - Adams Docs, 2011.

[TCT+13] Valerio Turri, Ashwin Carvalho, Hongtei E. Tseng, Karl H. Johansson, and Francesco Borrelli. Linear model predictive control for lane keeping and obstacle avoidance on low curvature roads. In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pages 378–383, 2013.

[VTM00] Prahlad Vadakkepat, Kay Chen Tan, and Wang Ming-Liang. Evolutionary artificial potential fields and their application in real time robot path planning. In *Proc. of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512)*, volume 1, pages 256–263 vol.1, 2000.

[ZGU13] Hongyi Zhang, Andreas Geiger, and Raquel Urtasun. Understanding high-level semantics by modeling traffic patterns. In *The IEEE International Conference on Computer Vision (ICCV)*, 2013.

[Zie12] Günter M Ziegler. *Lectures on polytopes*, volume 152. Springer Science & Business Media, 2012.

[ZZMM13] Muhammad Aizzat Zakaria, Hairi Zamzuri, Rosbi Mamat, and Saiful Amri Mazlan. A path tracking algorithm using future prediction control with spike detection for an autonomous vehicle robot. *International Journal of Advanced Robotic Systems*, 10(8):309, 2013.

Appendix A

Appendix

A.1 Results - Bicycle-Model

Table A.1: Vehicle one, time duration 2s.

run	simulation	reach. analysis
1	3.98s	4.07s
2	3.95s	3.79s
3	3.95s	3.85s
4	3.83s	3.86s
5	3.98s	3.81s

Table A.2: Vehicle two, time duration 2s.

run	simulation	reach. analysis
1	4.04s	3.90s
2	3.84s	3.75s
3	3.93s	3.74s
4	3.87s	3.77s
5	3.95s	3.84s

Table A.3: Vehicle one, time duration 10s.

run	simulation	reach. analysis
1	6.40s	17.10s
2	6.28s	17.18s
3	6.40s	17.18s
4	6.12s	17.25s
5	6.25s	17.23s

Table A.4: Vehicle two, time duration 10s.

run	simulation	reach. analysis
1	6.34s	17.18s
2	6.18s	17.37s
3	6.06s	17.30s
4	6.05s	18.19s
5	6.02s	18.78s

A.2 Results - Verification Module

Table A.5: Figure 6.5 a).

run	flowpipe constr.
1	40.42s
2	40.59s
3	40.71s
4	40.55s
5	40.62s

Table A.6: Figure 6.5 b).

run	flowpipe constr.
1	47.28s
2	47.96s
3	47.19s
4	47.35s
5	48.26s

A.3 Results - Hybrid Maneuver Automaton

Table A.7: Figure 7.1 - $m = 0$.

run	flowpipe constr.
1	11.31s
2	11.33s
3	11.32s
4	11.34s
5	11.96s

Table A.8: Figure 7.2 - $m = 1$.

run	flowpipe constr.
1	16.60s
2	16.43s
3	17.33s
4	17.41s
5	17.38s

Table A.9: Figure 7.4 - $m = 2$.

run	flowpipe constr.
1	11.33s
2	11.33s
3	11.98s
4	11.94s
5	11.97s

Table A.10: Figure 7.5 - $m = 3$.

run	flowpipe constr.
1	11.98s
2	11.35s
3	11.92s
4	11.99s
5	12.00s

Table A.11: Figure 7.6 - $m = 5$.

run	flowpipe constr.
1	11.43s
2	11.41s
3	11.91s
4	12.11s
5	11.38s

Table A.12: Figure 7.7 - benchmark.

run	flowpipe constr.
1	16.35s
2	16.31s
3	15.46s
4	15.42s
5	15.45s

A.4 Example Automaton

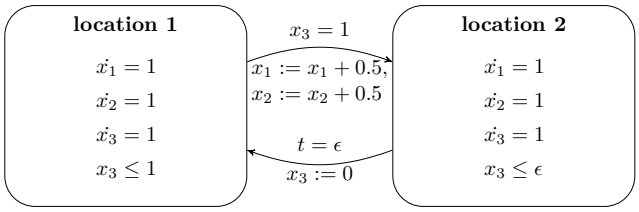


Figure A.1: Example automaton.

A.5 Verification Module

$$\mathcal{VA} = \{loc,var,act,inv,init\}$$

loc $\{vm\}$

var $\{x,y,v,a,\phi,\phi_v,t\}$

act $Act(vm) = \big\{ f : \mathbb{R}_{\geq 0} \to V \mid \exists c_x,c_y,c_v,c_a,c_\phi,c_{\phi_v},c_t \in \mathbb{R}. \forall t \in \mathbb{R}_{\geq 0}.$
$f(t)(x) = c_x + c_v \cdot \cos(c_\phi) \wedge$
$f(t)(y) = c_y + c_v \cdot \sin(c_\phi) \wedge$
$f(t)(v) = c_v + c_a \wedge$
$f(t)(a) = c_a \wedge$
$f(t)(\phi) = c_\phi + c_{\phi_v} \wedge$
$f(t)(\phi_v) = c_{\phi_v} \wedge$
$f(t)(t) = t + c_t \big\}$[1]

inv $Inv(vm) = \{t \in V \mid t(x) \leq 1\}$

$init$ $Init = \{(vm,v) \in V \mid v(v) = 0 \quad \forall v \in V\}$

[1]The activities represent the kinematic single-track model.

A.6 Hybrid Maneuver Automaton

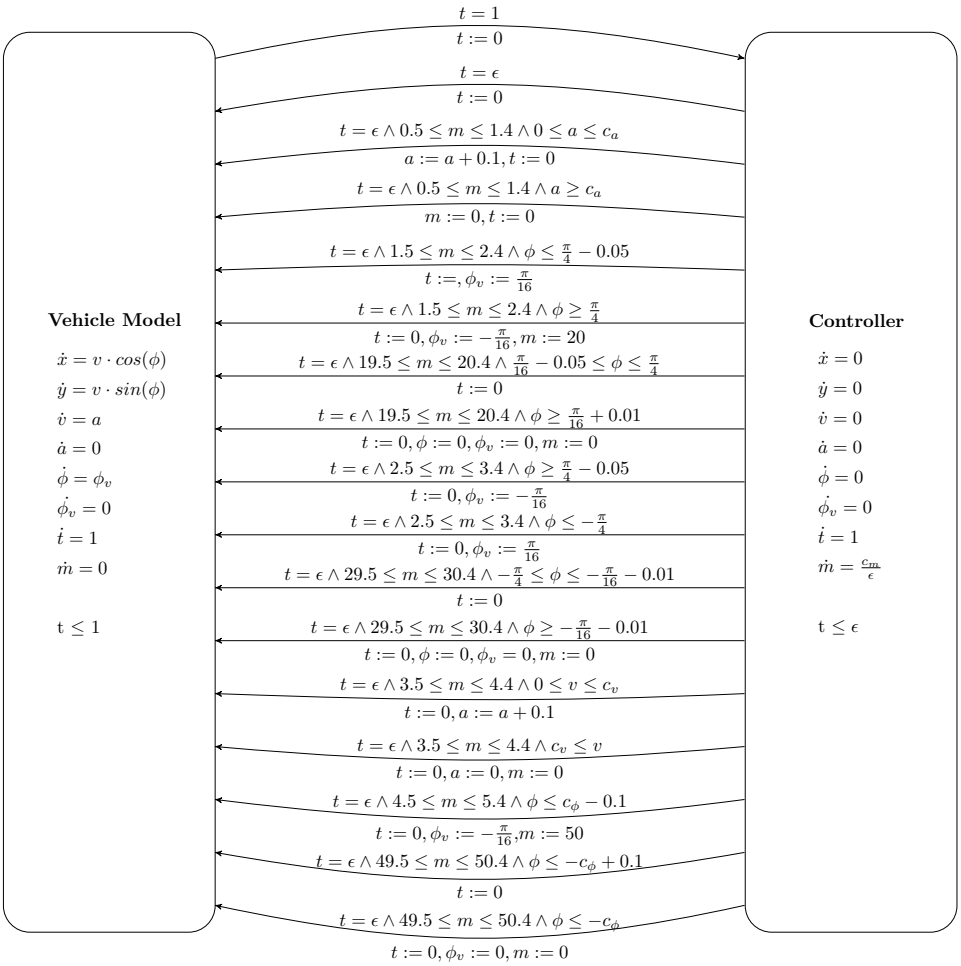


Figure A.2: Maneuver automaton.

Figure A.2 displays the maneuver automaton presented in Section 5.7, combining parts of a path planner and a vehicle model in one system to have a closed loop environment which constructs trajectories verified with a reachability analysis. In addition to this external inputs depicted in the formal specification modeled above are defined by c_a, c_ϕ and c_m. The three variables define the control inputs and the time specific maneuver mode input.