Local Trajectory Planning for Autonomous Driving

Dissertation zur Erlangung des Grades
eines Doktors der Naturwissenschaften (Dr. rer. nat.)
Fachbereich Mathematik und Informatik
der Freien Universität Berlin

von

Zahra Boroujeni

Berlin, 2020

Betreuer:
Prof. Dr. Raúl Rojas
Freie Universität Berlin
Institut für Mathematik und Informatik
Dahlem Center for Machine Learning and Robotics

Erstgutachter:
Prof. Dr. Raúl Rojas
Freie Universität Berlin
Institut für Mathematik und Informatik
Dahlem Center for Machine Learning and Robotics

Zweitgutachter:
Prof. Dr. Hans-Dieter Burkhard
Humboldt-Universität zu Berlin
Institut für Informatik

Tag der Disputation: 27.08.2020

Abstract

This thesis presents novel local trajectory planning methods to provide a safe, time optimal, and comfortable passenger ride. Trajectory planning is an essential part of autonomous driving systems, which has been extensively studied for robots during the past few decades. Providing passenger comfort, especially when working within a highly dynamic environment, makes the trajectory planning problem more challenging.

Based on the autonomous car situations, three different trajectory planning methods are proposed. The first method is a reactive trajectory planning which works in structured road maps with reference paths. The trajectory speed points are limited based on the road curvature, the traffic rules, and the distances to obstacles. A new algorithm is developed to smooth the speed profile considering the jerk and acceleration constraints. The jerk constraint is defined to provide passengers with a comfortable ride. The acceleration is limited based on the vehicle model and passenger comfort. The vehicle model is determined using the system identification. This approach is suitable for driving in urban areas with dynamic environments in which the obstacle speed changes frequently and the ego car trajectory must react to obstacles while avoiding instant braking or accelerating.

In the second trajectory planning approach, a new artificial force vector in three dimensions (longitudinal and lateral position, and speed) allows the autonomous car to follow a specific path. The vector field is created based on distance from the path and the car speed. It is locally modified by presence of obstacles. By switching between two different vector fields, the vehicle can change a lane or follow another path. The vector field approach is suitable for complicated paths with low traffic such as parking lots.

The third trajectory planning approach, Flexible Unit A* (FU-A*), is a new modified tree-based search algorithm in 3-dimensional space (longitudinal and lateral position, and time) in which lane-changing decision is also considered. The energy consumption, time duration, and displacement are integrated in the cost function of the algorithm. This combining of decision-making for lane-changing and following the reference path is one of this thesis' innovations.

The feasibility and reliability of the designed methods are validated through several simulations and implementation on Freie University autonomous cars.

Zusammenfassung

Die Trajektorienplanung ist ein wesentlicher Bestandteil autonomer Fahrsysteme. Sie wurde in den letzten Dekaden für verschiedene Roboter intensiv erforscht. Die Anwesenheit menschlicher Fahrgäste in autonomen Fahrzeugen schafft jedoch zusätzliche Herausforderungen, weil in einem hochdynamischen Umfeld auch Sicherheit und Komfort der Fährgäste berücksichtigt werden müssen. In dieser Arbeit werden neue Methoden der lokalen Trajektorienplanung entwickelt, um eine sichere, zeitoptimale und komfortable Fahrt für die Passagiere zu gewährleisten.

In dieser Arbeit werden drei verschiedene Methoden zur Trajektorienplanung vorgeschlagen. Für jede von ihnen gelten folgende Anforderungen: 1) die Bahn muss durch Aktuatoren des Autos abfahrbar sein und 2) sie muss eine sichere und komfortable Fahrt für die Passagiere garantieren. Hierfür werden zunächst die Kinematik und Dynamik des Fahrzeugs für die Steuerung der Aktuatoren modelliert. Die zweite Voraussetzung ist die Systemidentifizierung, die das Sammeln der erforderlichen Daten zur Festlegung der Systemeinschränkungen umfasst.

Die erste vorgeschlagene Methode ist ein reaktives Verfahren, das auf strukturierten Straßenkarten mit Referenzpfaden arbeitet. Die Geschwindigkeiten an den Stützpunkten der Trajektorie werden basierend auf der Straßenkrümmung, den Verkehrsregeln und den Entfernungen zu Hindernissen begrenzt. Ein neuer Algorithmus wurde entwickelt, um das Geschwindigkeitsprofil unter Berücksichtigung der Ruck- und Beschleunigungsbeschränkungen zu glätten.

Im zweiten Ansatz der Trajektorienplanung ermöglicht ein neuer künstlicher Kraftvektor in drei Dimensionen (Längs- und Querposition sowie Geschwindigkeit) dem autonomen Fahrzeug, einem bestimmten Pfad zu folgen. Das Vektorfeld wird basierend auf der Entfernung vom Pfad und der Fahrzeuggeschwindigkeit erstellt. Es wird lokal durch das Vorhandensein von Hindernissen verändert.

Der dritte Ansatz für die Trajektorienplanung, Flexible Unit A^* (FU-A^*), ist ein neuer modifizierter baumbasierter Suchalgorithmus im dreidimensionalen Raum (Längs- und Querposition sowie Zeit), in welchem auch Spurwechselentscheidungen berücksichtigt werden. Der Energieverbrauch, die Zeitdauer und die Verschiebung sind in die Kostenfunktion des Algorithmus integriert. Diese Kombination aus Entscheidungsfindung für den Spurwechsel und Verfolgung des Referenzpfades ist eine der Innovationen dieser Arbeit.

Die Machbarkeit und Zuverlässigkeit der entworfenen Methoden werden durch Simulationen und die Implementierung auf den autonomen Fahrzeugen der Freien Universität validiert.

The Table of Contents

Chapter 1

Introduction and Motivation

Autonomous cars are the emerging future of the automotive industry. In capturing this new market, passenger safety and energy efficiency are two important factors to consider. Although the main role of this new technology is to provide comfort and additional spare time to the passengers, a key priority in academic and industrial research centers is avoidance of driver-less cars' negative side effects. Researchers have high hopes that these innovations will improve safety and organization, as improvements to airplanes did several decades ago [1]. As computers do not tire, do not get anxious, do not break known rules, and do not lose concentration, they will likely cause fewer fatal and non-fatal accidents than human drivers. This ambitious vision fuels the progressive growth of autonomous cars technology despite all the challenges.

1.1 Motivation of the thesis

The research field of trajectory planning for autonomous cars investigates solutions is primarily focused on avoiding collisions and thus guaranteeing the safety of the passengers, and with a secondary goal of energy efficiency enhancement. To achieve these two main objectives, trajectory planning should provide smooth solutions in order to improve convenience and driving comfort. This includes measures such as unnecessary braking and accelerating - which work to provide a smooth ride while reducing energy consumption - or finding the shortest navigational path when considering all constraints - such as traffic or road closures.
Autonomous cars encounter a range of different situations, especially in the urban environment. Therefore, The author of this thesis believes one type of trajectory planning is not sufficient to deal with the various scenarios. As a part of this thesis, three different trajectory planning approaches are developed:

- First, a **reactive trajectory planner** is designed resembling human behaviour. Succinctly, this approach is as follows: the autonomous car will overtake the adjacent car when the provided lateral distance is enough; otherwise, it follows the lead car at the same speed. This type of an uncomplicated trajectory planner provides safe solutions for the passengers of the ego car and the other surrounding vehicles, as the human driver of the vehicles can simply predict the autonomous car's behaviour and properly react. The traffic rules, such as the two-second rule, collision time, and sufficient space, are all included in the decision of whether to brake for the obstacles or overtake them.

- Then, a **vector field based trajectory planner** method, which has a partially offline process to mitigate the computational load of the autonomous cars' internal computer, is designed. This trajectory planner introduces a novel approach for autonomous vehicle navigation in environments with a structured map by creating offline force vector fields, which specify the desired heading angle of the vehicle in order to fulfill *path following* and *lane keeping* tasks. The force vector fields are augmented and modified locally in case of presence of obstacles, which result from the obstacles force vector field. In creating force fields, we take into account the vehicle velocity, along with its distance from the path, to find force vectors that are feasible to follow.

- Lastly, the **trajectory planner using Flexible Unit-A* (FU–A*)** is designed to provide more sophisticated trajectories than humans are able to do. It is an optimal trajectory planning method to save time and energy. Autonomous cars are usually equipped with different sensors, and thus have fewer blind spots than humans. This facilitates the development of trajectories more mature than that of the previous approach. This approach uses the well-known A* path planning algorithm [2] while considering time as an extra dimension of the nodes. The grid unit of the search area changes depending on the speed of the nodes. Decreasing or increasing the speed makes the grids shorter or longer, thereby making the grid units flexible. The structured road map in which the autonomous car moves is not obstacle free, *e.g.*, there are other cars on the road, which we consider as dynamic obstacles. The proposed FU-A* search algorithm predicts the position of the obstacles on the structured map to evaluate which nodes will be obstacle free in the future.

Each trajectory planning approach is implemented in ROS framework and tested in real scenarios. In order to evaluate energy efficiency and compare it with human driving efficiency, this project developed a energy consummation model of the car[1] in which the car energy

[1]The experimental test-bed was an i-MiEV car.

consumption can be monitored online[1]. The energy consumption model is fitted to a data set of five hours of human driving. It is based on velocity, acceleration, and energy consumption, and can also adapt to the battery charge.

1.2 Thesis Contribution and Publications

The contributions of this thesis with respect to the state of the art are summarized as follows.

- The reactive trajectory planning method is extended to handle the jerk and non-linear acceleration constraints, in order to guarantee safety and provide the passengers with a comfort drive in urban areas. This method can cope with any form of acceleration constraint derived from actual car model. To this aim, first, the speed is limited in each point. Then, the velocity profile under nonlinear acceleration constraint and with limited jerk is smoothed numerically.

- A new vector field based trajectory planning is designed to generate the desired heading angle of a vehicle toward a specified road lane and prevent the car from colliding with obstacles. This trajectory planning method can be used computationally effective for relatively small areas with structured maps, such as parking lots.

- The FU-A* trajectory planning method is designed to generate a time optimal trajectory and to allow automatic lane changing decisions. For this purpose, the graph is generated dynamically using a structured road map with fixed time differences and flexible distances between nodes which change based on the vehicle's velocity. This method, generating the path and velocity profile simultaneously, is suitable for driving in highways.

The last two approaches are published as conference papers by the author of this thesis:

- Flexible Unit A-star Trajectory Planning for Autonomous Vehicles on Structured Road Maps (2017, IEEE International Conference on Vehicular Electronics and Safety (ICVES); Zahra Boroujeni, Fritz Ulbrich, Daniel Neumann, Daniel Goehring, Raul Rojas) [3]

[1] All the variable values and parameters used in the model are available through the CAN bus of the vehicle via OBD-II, which has been standardized in all cars since 1979. OBD-II of electric cars provides energy data such as current, voltage, and charge percentage with a sufficiently high refresh rate (100 Hz in our case).

- Autonomous Car Navigation Using Vector Fields (2018, IEEE Intelligent Vehicles Symposium; Zahra Boroujeni, Mostafa Mohammadi, Daniel Neumann, Daniel Goehring, Raul Rojas) [4]

1.3 Thesis Outline

The rest of the thesis, as shown in Fig. 1.1, is organized as follows. The next chapter provides a review of related research. Chapter 3 details the reactive trajectory planning approach. The feasibility and reliability of the reactive trajectory planning is also validated through implementation on an autonomous car from Freie University, "Mig". Chapter 4 explains the vector field based trajectory planning. This method is also tested on the Freie University Model car. Then, chapter 5 describes the use of the FU-A* algorithm to solve the shortest path problem. In each of these three methodological chapters, numerical simulation results are provided to show the effectiveness and efficiency of the proposed approach. Finally, concluding remarks are outlined in Chapter 6.

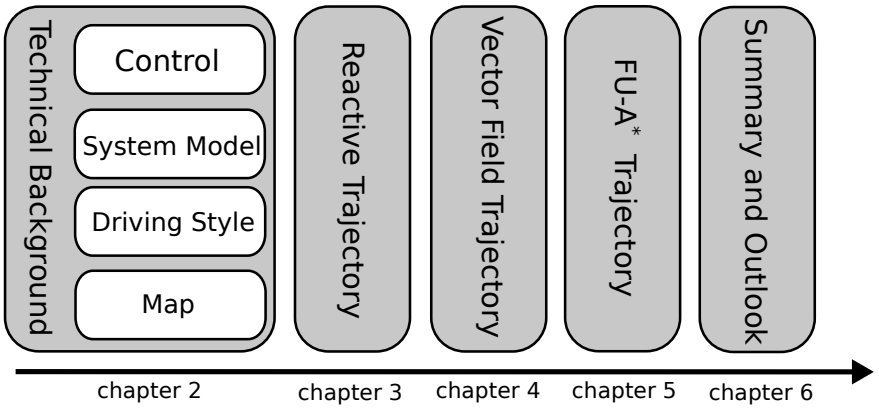


Fig. 1.1: Schematic outline of the dissertation.

Chapter 2

Related Work and Preliminaries

This chapter gives an overview of the current condition of the field in planning and its related tasks. Section 2.1 presents a review of the conventional and well-known trajectory planning approaches. As the trajectory must be tracked by a low-level control, Section 2.2 presents a review of the common control methods of self-driving cars. Section 2.3 presents Freie Universität Berlin's self-driving cars (i-MiEV and MIG). This includes an introduction to their software and hardware and the modelling of throttle and brake pressures as a function of acceleration and speed. As preferred driving style affects trajectory parameters such as maximum acceleration and jerk[1], Section 2.4 presents studies related to maintaining passenger comfort while driving. Finally, Section 2.5 explains the procedure of generating a structured road map.

2.1 Trajectory Planning

Calculating an optimal trajectory with regard to safety, passenger comfort, time, and energy constraints, all while under real-time conditions, is a challenge that has researchers have addressed in various ways. The proposed approaches in this field can be categorized in two main classes: decoupled and coupled trajectory planners. In the first category, a path is generated and then a speed profile along the path is constructed. In the latter category, the path and speed profiles are generated simultaneously. Decoupled trajectories are computationally faster than coupled trajectories, and they are utilized in several real-time applications. However, coupled trajectories provide optimal solutions, and they can deal with more complex scenarios. Some of the recent works in each class are highlighted in the rest of this section.

[1] Jerk is the time derivative of acceleration.

2.1.1 Decoupled Trajectory Planner

Generating Path

A path should be feasible despite nonholonomic constraints of car-like robots. Based on the Ackermann or bicycle kinematic model, car-like robots can follow a circle whose radius is limited based on the maximum steering angle and the wheelbase. Thus, the curvature of the path must be limited. In [5, 6], a variant of A* algorithm combined with the Reed–Shepp algorithm [7] is used for free environments (unstructured or semi-structured) while just considering static obstacles. In the Reed–Shepp algorithm, the car follows arc circles or straight lines as it goes forward and backward. The car speed is assumed to be constant, but can be modified to avoid obstacles.

A well-known variant of A*, the so-called dynamic A* or D* [8] updates edge costs incrementally rather than recalculating them when some of the edges change. D* computes the optimal plan from the goal to a starting point. For large graphs, this variant saves considerable computation time.

Randomized search algorithms, such as RRT, create paths by using random samples from the search space [9]. For unstructured environments, they provide good results which converge to an optimal solution with large number of samples. As the focus of this thesis is on a structured environment, it would be possible to shrink the search space based on the map and then to choose samples randomly. However, the grid sampling scheme has sufficient speed for real-time application to mean that this approach would not give any additional advantage.

The **Frenet** Coordinate is a popular coordinate system for path planning in the field of autonomous driving at structured road maps [10]. The Frenet frame moves along the path curve. Its axes are the tangent and normal unit vectors at each path point. The total arc length along the path is denoted by s. Fig. 2.1 shows a path in both Cartesian and Frenet

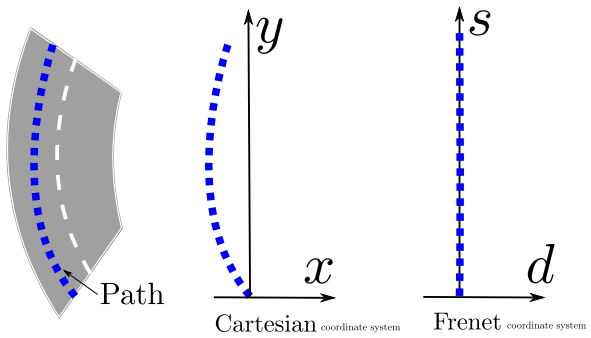


Fig. 2.1: A path is shown in the Frenet and Cartesian coordinate systems. In the Frenet coordinate system, the s-axis indicates how far the car is on the path, and the d-axis indicates the lateral distance of the car from the path.

coordinate systems. The s-axis indicates the car longitudinal displacement on the path. The d-axis indicates the lateral distance of the car from the path.
To generate an initial path, the lane center points are considered to be sample points. The sample points are connected to each other by a cubic spline [11, 12] or with a combination of cubic and quartic splines [13, 10]. In Frenet frame, the path curvature κ is a cubic/quartic polynomial of the arc length s[1]. Generating the quartic spline takes longer. Therefore, the quartic spline is only used instead of cubic spline to connect the current position of the car to other sample points as the curvature profile is smoothed. In Chapter 3, the cubic spline method is used. However, the curvature at each point is smoothed based on its neighbor points. In the lane change maneuver, a cubic spline is defined between two lanes. The length of the polynomial is proportional to the vehicle speed, thereby limiting the curvature along the path.
Using potential/vector field maps is another method to generate the path. The rationale of using vector fields for the navigation of autonomous vehicles is to define an artificial vector field which attracts the vehicle toward a desired point (goal) and prevents collision with obstacles [14]. The concept of using vector and potential fields for finding optimal paths in environments with static and dynamic obstacles for mobile robots is still a developing research topic, and researchers suggest new techniques for path planning and collision avoidance. Vector fields for robotic navigation are used in several applications ranging from mobile robots [15, 16] to aerial vehicles [17], space crafts [18], and recently for autonomous cars [19, 20]. In [16], Bacterial Potential Field (BPF) and Artificial Potential Field (APF) are compared to generate a path. In APF, a Mobile Robot (MR) is presented by a point and a heading vector. The potential function is the weighted sum of an attractive potential function (proportional to the squared error of MR position to the goal point) and a negative repulsive potential function (inversely proportional to the shortest distance to an obstacle). The total force for MR navigation is the gradient of the potential field. The weighting coefficients of the potential field are specified by trial and error in classic APF while they are optimized in BPF by a random search algorithm (referred as the bacterial mutation). An integrated motion planning and control approach for autonomous car navigation based on potential fields is presented in [21]. In this paper, the control effort is reduced while maintaining a desired tracking error tolerance. A framework for path planning and tracking is suggested in [22], which focuses on collision-free paths.
A novel path planning using vector field approaches is proposed in Chapter 4. Instead of generating a potential field, a three dimensions vector field (two dimensions in the space domain, one dimension in the speed domain) is generated toward a lane center points.

[1] In Cartesian frame, the path points coordinates x and y are cubic/quartic polynomials of the arc length s.

Constructing Speed Profile

After generating the feasible paths, the speed profile is constructed for each individual path. In [23], after generating the path, a constant maximum speed and a constant maximum acceleration are defined as car dynamic constraints. The upper and lower boundaries for the final trajectory point are introduced based on two scenarios: completely stopping with zero final acceleration and speed, and merging into an expressway with the final speed in the range of the expressway speed limit. Finally, the speed planning problem is formulated as a convex optimization problem, and it is solved with Interior Point Method (IPM) [24] using Gurobi optimizer [25]. The optimization problem objective function contains two parts (time efficiency, and smoothness): the total travelling time, and pseudo jerk, which is the first derivative of acceleration with respect to the arc-length.
In [26] the pseudo jerk replaces the jerk to create a convex smoothness objective function. The IPM algorithm is not guaranteed to converge until the step size is very small. Solving the speed planning as a convex optimization problem is not a time-efficient method in practice. Furthermore, in [27], it is shown that the feasible domain of the speed planing is non-convex. Therefore, the problem is solved in two steps: first a rough speed profile is found by searching in Station-Time (s-t) graph with equal intervals, and then a quadratic programming (QP) algorithm is used to optimize the rough speed profile by the CasADi optimizer [28]. The delay of the CasADi interface is not reported in this paper. The optimizer interfaces such as CasADi and Gurobi have delay and are not thread-safe, which make them incompatible with our framework (ROS) and with feasible practice.

In method proposed in [13, 29], the speed profile is found by searching in Velocity-Station (v-s) graph with equal intervals. The speed is limited between two constant values. The speed sample points are connected with a cubic spline of arc length s instead of a cubic spline of time t as proposed in [10]. The validation of the sample points are examined with a constant maximum acceleration. A straightforward and exhaustive search (*e.g.*, Dijikstra) is applied to find the optimal trajectory.
In [30, 31], a trapezoidal speed profile is proposed. In a trapezoidal speed profile, acceleration switches between a maximum value, a minimum value, and zero, which causes infinite jerk at switching points. Controllers cannot follow a speed profile with infinite jerk. Double S (bounded jerk) speed profile is proposed in [32, 33]. In [32], a speed profile is divided into an acceleration phase and a deceleration phase; in the case of autonomous cars, a speed profile contains several acceleration and deceleration phases. In [33], the problem is formulated as a Model Predictive Control (MPC) with linear constraints. In Chapter 3 of this thesis, the speed profile with bounded jerk is generated and generalized for the nonlinear constraints (non-convex domain). The acceleration constraint function can be any function derived from

actual car model. The local extrema of speed constraints in Velocity-Station (v-s) graph are connected by a cubic spline of time.

2.1.2 Coupled Trajectory Planner

In coupled trajectory planners, the path and speed profiles are generated simultaneously. In [34], two methods, "Partial Motion Planning (PMP)" and "Quintic Polynomial Planner," are used to determine the car's next movement based on its current state and to avoid dynamic obstacles with a conservative prediction. First, a maneuver grid [35] is generated by combining three longitudinal actions (deceleration, acceleration, or hold in the current speed) and three lateral actions (changing the lane to left, right, or staying in the current lane). From the longitudinal maneuvers at each lane, the best one based on a cost function is selected. The cost function takes the collision risk, speed, comfort, and traffic rules into account. Then, a feasible trajectory using a grid search (in PMP method) or quintic polynomial is generated for each best three lateral maneuvers. The maximum acceleration and steering rate are the search space constraints. The best trajectory is chosen based on the cost function. It is shown that the quintic polynomial approach is ten times more time efficient. In Chapter 5 of this thesis, a tree of maneuvers is generated. The feasibility of the maneuvers is guaranteed by selecting a feasible time interval. The path sample points are connected by the cubic spline of arc length s.

Optimization methods are introduced to avoid discretization with equal intervals like search algorithms. A linear MPC is developed in [36]. The feasible domain is restricted by linearzing the constraints. In [37], an adaptive time elastic band is proposed to cope with highly dynamic traffic scenarios. In [38], the time elastic band method is combined with MPC approach to work within unstructured environments; the sequential programming approach is used to optimize the problem solution. Stabilization of MPC approaches is challenging, and so they are not used in real autonomous cars.

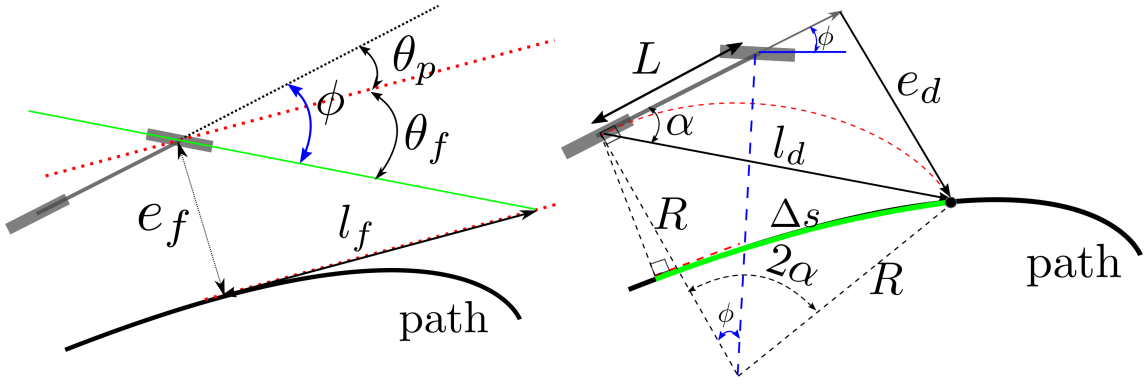
2.1.3 Obstacle Prediction

Prediction of dynamic obstacle behaviour is a challenging part of trajectory planning, which has been studied utilizing machine learning techniques [39] and probabilistic models [40, 41]. In [40], a dynamic obstacle is modeled as a box; it is assumed that the car drives on the road while following the traffic rules. In [41], an obstacle behaviour prediction is modeled as a quintic polynomial based on the deviation of the obstacle's movement from the street center and under the assumption of small road curvature. In contrast to both of these methods, the approach proposed in this thesis determines the target lane of the obstacle

based on the minimum distance of the vehicle from the lane's center. The predicted trajectory is then modeled as a cubic spline along the road (which could be a curvy road) under the assumption of a slow time varying velocity, which is the most probable prediction.

2.2 Control Approaches

The objectives of a control system for an autonomous vehicle must include stability, trajectory tracking precision, and passenger comfort. To fulfill these goals, different controllers have been designed for autonomous cars during the last few decades.



(a) Stanley [42]

(b) Pure pursuit

Fig. 2.2: Geometric lateral controllers: in the Stanley controller (a), the desired steering angle (ϕ) is given by (2.1b) based on a look-ahead distance (l_f) along the the tangent line of the nearest point on the path from the front axle. The desired steering angle is sum of θ_p (the angle between the tangent line and the vehicle heading vector) and θ_f (the angle between the the tangent line and the line between the look ahead point and the front axle). The lateral distance of the front axle from the tangent line is e_f. Pure pursuit controller (b) defines the look-ahead distance (Δs) along the path from the rear axle nearest point on the path, and the desired steering angle (ϕ) is determined by (2.2d). The length of the arc chord is l_d. The angle between the arc chord and the car heading vector is α. The radius of the arc between the look-ahead point and the current rear axle position is R. The lateral distance of the look ahead point from the vehicle heading vector is e_d.

Four different lateral controllers are compared in [42]: Stanley, Pure pursuit, LatVel, and Sliding mode controllers. Pure pursuit and Stanley controllers are geometric controllers, as shown in Fig.2.2. In Stanley controller, the desired steering angle (ϕ) is defined based on a *look-ahead distance* l_f along the derivative of the nearest point on the path from the front

axle as follows:

$$l_f = \lambda \; v \tag{2.1a}$$

$$\phi = \theta_p + \theta_f = \theta_p + tan^{-1}\left(\frac{e_f}{l_f}\right) \tag{2.1b}$$

where θ_p is the difference angle between the vehicle heading vector and the tangent line of the nearest path point to the front wheel[1], e_f is the distance between the front axle and nearest point on the path, and the l_f distance is proportional to the vehicle longitudinal speed v by coefficient $\lambda > 0$. As the l_f distance is only defined based on velocity and does not depend on the path curvature, in the modified approach–pure pursuit control–the desired steering angle (ϕ) is determined based on a look-ahead distance Δs from the rear axle nearest point along the path as:

$$\Delta s = \lambda \; v \tag{2.2a}$$

$$R = \frac{l_d}{2\sin\alpha} \tag{2.2b}$$

$$\sin\alpha = \frac{e_d}{l_d} \tag{2.2c}$$

$$\phi = tan^{-1}\left(\frac{L}{R}\right) = tan^{-1}\left(L\frac{2e_d}{l_d^2}\right) \tag{2.2d}$$

where Δs is a distance along the path from the rear axle nearest point on the path and proportional to the vehicle longitudinal speed v by coefficient $\lambda > 0$. This distance determines the look-ahead point. The length of the vector between the current rear axle position and the look-ahead point is l_d, the angle between[2] this vector and the car heading vector is α. The car heading vector is a tangent vector of the circular arc that connects the rear axle location to the look-ahead point; therefore, the arc angle is 2α. The arc radius is R. To understand the control low better in [43] a new variable e_d is defined which is the lateral distance of the look-ahead point from the vehicle heading vector. The tangent of the desired steering (ϕ) is proportional to e_d with a gain of $2L/l_d^2$, where L is the vehicle wheelbase. Pure pursuit controller is precise enough to follow a trajectory on a curved road and provides passenger comfort, while Stanley controller shows a poor performance at high speeds.

Sliding and LatVel controllers are the kinematic-based controllers, which use the kinematic bicycle model in the Frenet frame. Fig. 2.3 shows the kinematic bicycle model, in which κ is the path's curvature on the nearest point to the rear axle, d is the lateral distance to the path,

[1]The vehicle heading angle at a Frenet frame is θ_p which is the difference between the heading of the vehicle and the heading of the path at the nearest point at a Cartesian frame.

[2]The angle between two vectors can be obtained from their dot product.

and θ_p is the angle between the car heading vector and the tangent line of the nearest path point to the rear wheel. The kinematic model is defined as:

$$\dot{s} = \frac{\cos\theta_p}{1-\kappa\, d} v \tag{2.3a}$$

$$\dot{d} = \sin\theta_p\, v \tag{2.3b}$$

$$\dot{\theta}_p = \frac{v}{R} - \kappa\, \dot{s} = \left(\frac{\tan\phi}{L} - \frac{\kappa}{1-\kappa\, d}\cos\theta_p\right) v \tag{2.3c}$$

where $\dot{s}$ is the projection of the vehicle rear wheel speed v on the tangent vector of the Frenet frame, κ is the road curvature, $\dot{d}$ is the lateral speed perpendicular to the path, $\dot{\theta}_p$ is the relative rotational velocity in Frenet frame, which is the subtraction of the car rotational speed ($\frac{v}{R}$) from path rotational speed $\kappa\, \dot{s}$, in which R is the radius of the related rotational circle. The steering angle is denoted with ϕ, and wheelbase is denoted with L. By defining an auxiliary controller input W, the kinematic model can be linearized as:

$$\dot{\theta}_p = W \tag{2.4a}$$

$$\phi = \arctan\left(L\left(\frac{W}{v} + \frac{\kappa}{1-\kappa\, d}\cos\theta_p\right)\right) \tag{2.4b}$$

Sliding mode control pushes the states θ_p and d toward a desired sliding surface σ by defining a discontinuous control signal. States can slide along the sliding surface and move toward desired values (zero in our case). The controller pushes the states toward the desired values if the sign of surface σ and its derivative are opposite. Therefore, the input controller defines as:

$$\sigma = k_\theta \theta_p + k_d\, d \tag{2.5a}$$

$$\dot{\sigma} = -k_\sigma sign(\sigma) \tag{2.5b}$$

$$W = \frac{-k_\sigma sign(\sigma) - k_d \dot{d}}{k_\theta} \tag{2.5c}$$

Where k_θ, k_d, and k_σ are tuning parameters. Chattering is a common problem of Sliding mode control, which is disadvantageous in terms of motion smoothness and passenger comfort.

LatVel controller is another kinematic-based controller introduced as:

$$W = -k_\theta(v\, \sin\theta_p + k_p\, d) \tag{2.6}$$

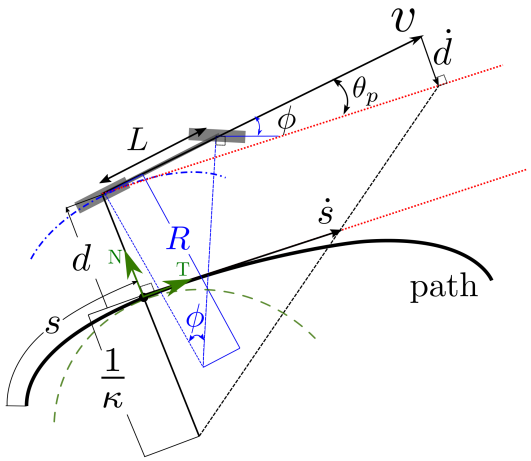


Fig. 2.3: The kinematic bicycle model in the Frenet frame (T,N): the green arc illustrates the road curvature κ at the nearest point on the path from the rear axle. The axes (T,N) are tangent and perpendicular to this arc at the nearest point. The longitudinal displacement of the rear wheel along the path is denoted with s, and the lateral distance from the path is denoted with d. The angle between the heading vector and T-axis is denoted with θ_p. The vehicle states $\dot{s}$ and $\dot{d}$ in the Frenet frame are the projection of the vehicle speed v on T-axis and N-axis. The vehicle with steering angle ϕ and wheelbase L follows the blue arc with the radius R.

The LatVel method satisfactorily provides precise tracking and smoothness by considering the vehicle kinematic and the path curvature in the controller. However, this method increases complexity and requires to tune more parameters.

Among other approaches, in [44], an adaptive PID controller is designed in which the controller input is the lateral error to the path. The PID parameters are changed based on the controller input, resulting in a smaller proportional coefficient on straight lanes and a bigger one on curves. However, the results have been presented up to speeds of 20 km/h, due to choosing only lateral position error as input.

Of the controllers described above, our system uses the pure pursuit lateral controller. This lateral controller is best suited for this project because it is easier to tune and because it is better capable of achieving the control objectives, compared to the other methods. Also. to follow the velocity profile, a PD controller is designed for which the desired velocity is selected as a T times ahead of the current time.

2.3 Experimental Setup: i-MiEV and MIG as Testbeds

The proposed methods in this dissertation are implemented and tested on an i-MiEV (an electric car from Mitsubishi modified as an autonomous car testbed) and a VW Passat (a

petrol car called Made In Germany (MIG)) by utilizing the software packages developed for autonomous cars at the Dahlem Center for Machine Learning and Robotics (Freie Unversität Berlin). In the following, our testbeds, their sensors and actuators, and the structure and interconnection of their basic software packages are described.

2.3.1 Hardware Setup

The autonomous vehicles (i-MiEV and MIG) are equipped with the following sensors and actuators:

- **POS LV Applanix GPS-INS system**: this navigation system combines GPS with an Inertial Navigation System (INS) and wheel encoder data to provide the position, velocity, and acceleration of the car at a sufficiently high data rate (100 Hz) [45].
- **Ibeo LUX**: the Ibeo laser scanners are installed around the car to detect obstacle on the path. Ibeo's typical range for vehicles is about 150 m, and about 50 m for pedestrians [46]. By fusing Ibeo and Applanix data, we can track and classify the obstacles.
- **HDL-32E/64E Velodyne**: this is a multi-array rotating laser scanner with a range of about 70/120 m for vehicles. Velodyne data is gathered from all around the car by 32/64 beams and a rotating speed of up to 600/900 rpm[10/15 Hz] [47]. This data is used to detect objects around the car and is utilized for ego-car localization[1] and obstacle avoidance in planning.
- **Stereo Camera**: i-MiEV also benefits from a TomTom stereo camera, with range about 70 m, to detect the obstacles on the path.
- **Long-Range Radar LRR**: MIG also benefits from four RADAR sensors, with range about 150 m, installed around it to detect the obstacles on the path.
- **Paravan system**: it augmented i-MiEV with steering engine, brake, and accelerator pedal engine in order to to execute the control actions [48].

Fig. 2.4 and 2.5 shows the placement of the above-mentioned hardware on the cars and the scanning area of each scanner.

[1]The Velodyne data is used to detect the poles, *e.g.*, trees, around the street. As we have the real map of the poles, we can localize the car by measuring the range and bearing of the poles (and tracking them), and associating these measurements with poles on the map combined with the odometry data. The localization method is out of the scope of this thesis.

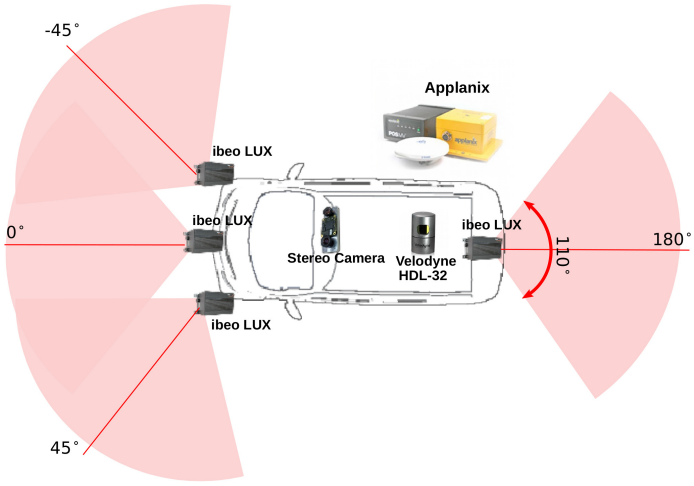


Fig. 2.4: The i-MiEV sensor configuration contains four components: 4 Ibeo LUX, a Velodyne HDL-32E, a stereo camera, and a Applanix POS LV. To minimize the blind area around the car, the scanning ranges of Ibeo sensors in front overlap. The red lines illustrate the angles of the central beam of the sensors with respect to the car heading vector.

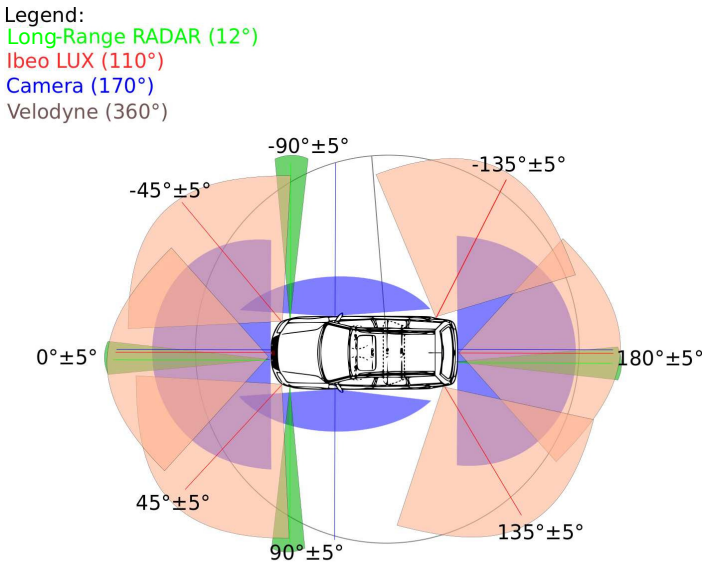


Fig. 2.5: MIG sensor configuration: 6 Ibeo LUX, Velodyne HDL-64E, Applanix POS LV, 4 Long Range Radar. To minimize the blind area around the car, the scanning ranges of sensors overlap. The red lines illustrate the angles of the central beam of the sensors with respect to the car heading vector.

Finally, in order to calculate and monitor the energy consumption, we used OBD-II to get the data for on-line measurements of the i-MiEV battery current and voltage. We recorded data from one hour of manual driving to create an energy consumption model for the car that is used to predict the total energy consumption of the trajectory.

2.3.2 Software Structure

The software structure of the testbed system consists of three subsystems: perception, planning, and execution. The overall software structure is illustrated in Fig. 2.6 and is described below.

Perception involves the processing of raw sensor data to find exactly where the car is located and how the car is situated within its surroundings. Data processing in the perception subsystem includes filtering, classification, and tracking of objects that are themselves created from raw sensory data using pattern recognition techniques. The high level extracted data is then fused, *e.g.*, the pose and twist of objects, from different sources to complement and reduce uncertainties. In particular, car localization is performed as follows: the Applanix sensor package provides the GPS (and optional Real Time Kinematic (RTK) GPS) and IMU data that we use to estimate the car position and its derivatives. The estimated pose from the Applanix data is precise with an uncertainty of less than 0.1 m. However, in GPS-denied areas, it can jump (up to 2 m) due to lack of signals. Therefore, the car positioning is augmented by a landmark-based localization technique in which the landmarks that are described in an environment map, namely poles such as trees or traffic lights, are used to localize the car. We use the Velodyne point cloud data to detect the poles[1] and to find the car's position in the global coordinate frame based on the range and bearing measurements from the detected poles and the association of these measurements with map data. Practically, the more detected poles there are, the less uncertain the localization result will be. The odometry data is used to track the detected poles and predict the car pose in the period between pole detections, as object detection in general, and pole detection in our case, that is based on laser range scanner data usually has a lower output rate than the odometry data from the GPS-INS system. Another example of the complementary data fusion performed in the perception subsystem is the detection of objects in the car's surroundings by combing the data from the four Lux Ibeo laser sensors and the Velodyne. The Velodyne data is gathered from all around the car by 32 beams with a rotating speed of up 600 rpm, *i.e.*, up to 20 Hz output data rate, and is reliable within 80 m distance. On the other hand, the high speed Ibeo sensor data

[1] The stereo camera can be used as a pole detection [49]. In our project (KLEE project [50]), however, it is mainly used to detect the traffic light position and status [51].

are static but are reliable within 150 m distance. Therefore, these data are used together to classify the obstacles in two main categories: static and dynamic.

Status monitoring of sensors and actuators is an essential requirement for the planning subsystem. For example, before starting the car and using the planning output, the autonomous actuator system (Paravan system) is checked in park mode to ensure the correct functionality of the accelerator, brake pedal, and steering actuators. Some of the sensory data are also checked by the human user to ensure all the sensors are active and functioning correctly. Fig. 2.7 shows the status monitoring user interface.

Planning subsystem consists of two modules: *behavior and trajectory planning* in the higher level and *controller* in the lower level. Behavior planning receives the data from the perception subsystem and makes decisions, such as lane keeping, lane changing, intersection behaviours (turn left/right, go straight) for a long-term length, whereas trajectory planner calculates for a short length (maximum 150 m, limited by sensor ranges). In next chapters, the main focus of this thesis is this module. The controller receives the trajectory planner output and calculates the required values of throttle/brake and steering wheel for the actuators. The controller will be described in more detail in the next sub-section.

Execution subsystem executes the controller output. The signal passes through the Safe Box. The desired steering is limited and converted to voltage in the Safe Box. The steering angle is limited based on the car wheel limitation and is converted to voltage for the Paravan steering actuator. The accelerator and brake signal are also converted to the desired voltage and sent through the serial port to the Parvan system. Finally, for safety, there are two switches near the human driver's seat: a big red push button on the dashboard and a pedal near the left foot of the driver. To change from autonomous mode to manual mode, both buttons need to be pushed.

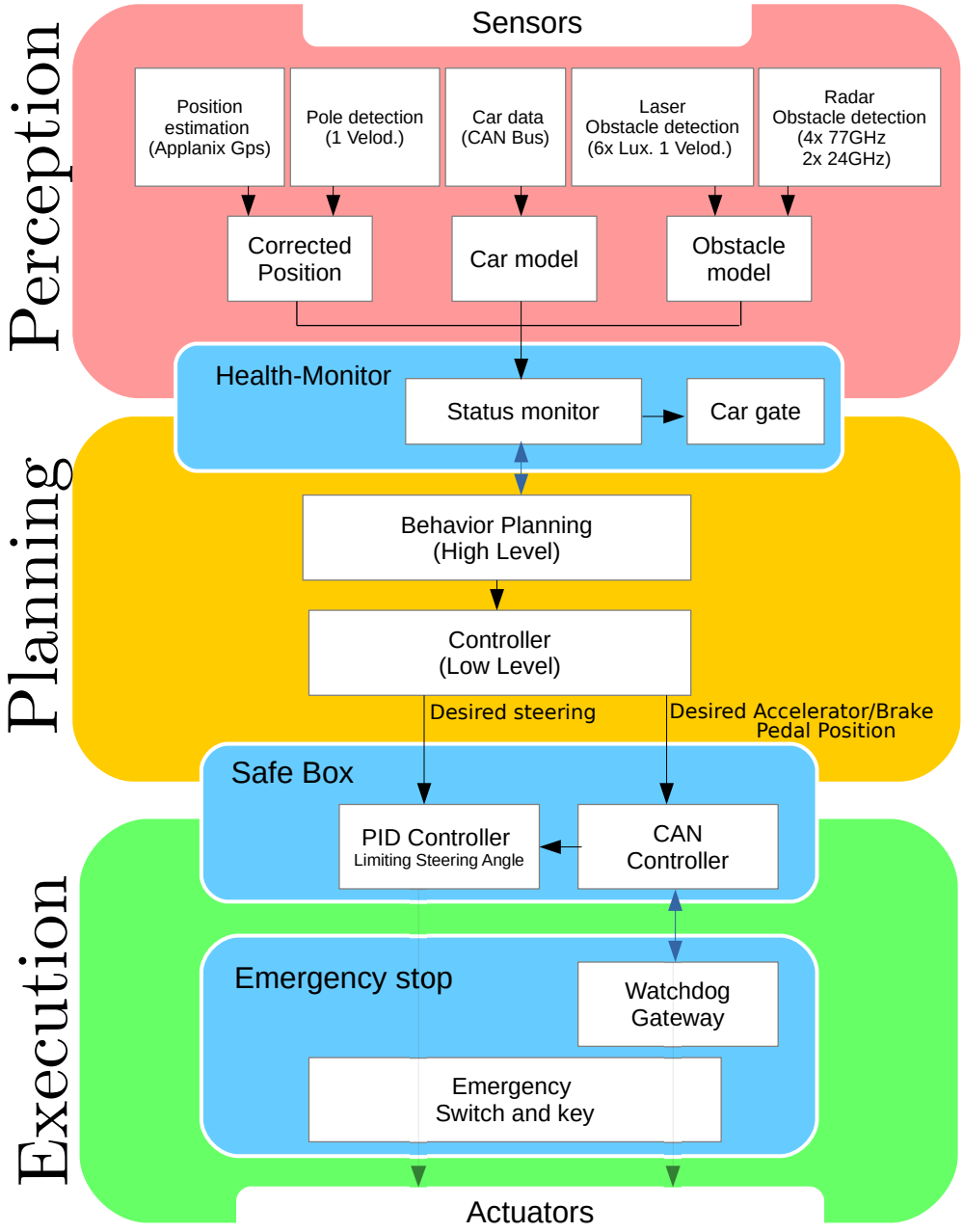


Fig. 2.6: Testbeds software structure: composed of perception, planning, and execution subsystems. The diagram is a modified version of the software structure at the KLEE project proposal [52].

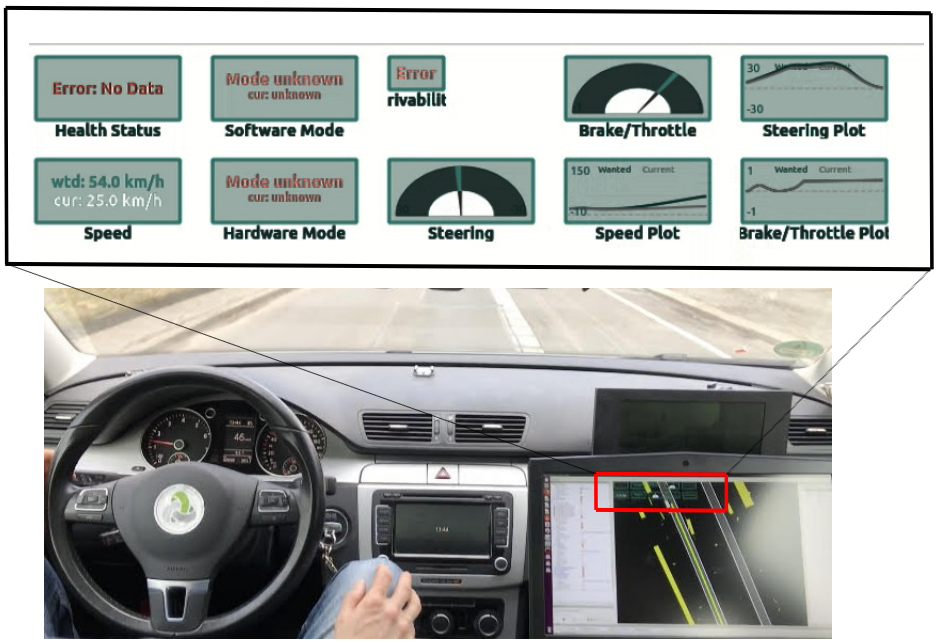


Fig. 2.7: Status Monitoring: health status, software and hardware mode, autonomous drivability, real and planned speed, steering and accelerator/brake pedal position.

2.3.3 Dynamic Model

The proposed trajectory approaches are tracked through the execution of speed and steering commands, which are regulated by low level PID controllers. In order to properly design and implement these controllers, it is necessary to understand and analyze the system's kinematics and dynamics. The car acceleration constraint will be determined by a simplified model of the car. This acceleration constraint will be used in the next chapter in trajectory planning. The models described below are used in our simulation to test the trajectories and controller before their implementation on the real car. The speed and acceleration of trajectory points should be determined so that the system (controller and actuators) can follow them. For example, in i-MiEV (or another electric motor), the car can accelerate from the beginning with maximum limited torque. As the speed increases, the torque/acceleration decreases since the motor power is constant. Therefore, we need to know the model of the system and its actuators. Passenger comfort also limits the acceleration and jerk of trajectory points, just as the traffic rules limit the velocity.

Bicycle Dynamic Model

Our testbed, i-MiEV, is a Rear-Wheel-Drive (RWD) electric car, which has a 2-Wheel-Drive (2WD) traction system. Four wheel dynamic model is used in the literature, e. g. [53]

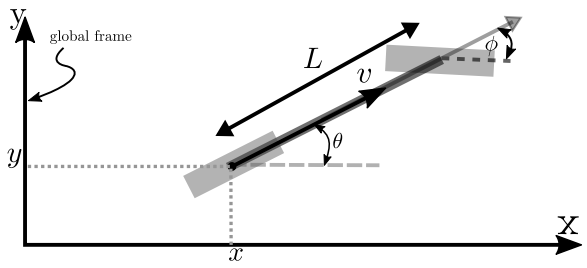


Fig. 2.8: Bicycle kinematic model in a global Cartesian frame (X,Y): in the simulator, the map frame is considered to be the global frame, where x, y, and θ specify the position of the rear wheel and the vehicle heading angle. The vehicle velocity, the steering angle and the vehicle wheelbase are denoted with v, ϕ and L, respectively.

and [54], to create feasible trajectories. In this thesis, for the sake of clarity and simplicity, the *bicycle model* of the car in the Cartesian frame is utilized, as shown in Fig 2.8. The dynamics of the bicycle model, as shown in Fig. 2.9, is given by:

$$m\, a_x = F_x - F_a - F_r - F_\alpha \tag{2.7a}$$

$$m\, a_y = m\dot{\theta}\, v = F_y \tag{2.7b}$$

where $F = [F_x\ F_y]^T$ is the longitudinal and lateral forces. The longitudinal force changes the velocity magnitude and the lateral force changes its direction. The vehicle states a_x and a_y represent the longitudinal and lateral acceleration, respectively. The angular velocity is denoted with $\dot{\theta}$. The parameter m is the vehicle mass[1]. In order to accelerate the vehicle, the drive train force F has to overcome the aerodynamic drag force F_a, the uphill force

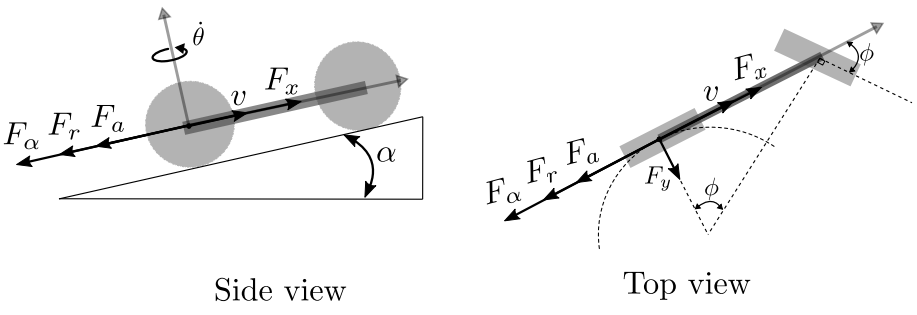


Fig. 2.9: Bicycle dynamic model: the longitudinal force F_x changes the velocity magnitude $|v|$ and the lateral force F_y changes the velocity direction. In order to accelerate the vehicle longitudinally, the driving force F_x has to overcome the aerodynamic drag force F_a, the rolling friction force F_r, and the uphill force F_α. Here α, and $\dot{\theta}$ represent the inclination of the road, and angular velocity, respectively. The rear wheel follows the black dashed arc with the related steering angle ϕ.

[1]The i-MiEV weight is 1450 kg after modifications (adding actuators and sensor setups).

Table 2.1: Constant parameters of the car dynamic equations, where g represents the gravitational acceleration, ρ is the density of the air , C_w is the air drag coefficient, A_v is the frontal area of the vehicle, and μ_r is the initial rolling friction coefficient [55].

g [m/s^2]	ρ[kg/m^3]	A_v[m^2]	m[kg]	C_w	μ_r
9.8	1.293	2.15	1450	0.35	0.007

F_α, and the rolling friction force F_r. The aerodynamic drag force $F_a = 0.5\rho C_w A_v v^2$, where ρ is the density of the air, C_w is the air drag coefficient, and A_v is the frontal area of the vehicle. The uphill force $F_\alpha = mg\sin(\alpha)$, where g represents the gravitational acceleration, α represents the inclination of the road. The rolling friction force $F_r = mg\cos(\alpha)\mu_r(v)$, where the parameter μ is the rolling friction coefficient which is linearly proportional to the speed (for low speeds). The parameters are listed in Table 2.1.

The car linear and angular velocities, v and $\dot{\theta}$, in simulator is given by:

$$v = v_0 + a_x\,\Delta t \tag{2.8a}$$

$$\dot{\theta} = \frac{v}{R} = \frac{\tan\phi}{L}v \tag{2.8b}$$

where Δt is the 0.01s (the simulator thread runs at 100 Hz). The initial velocity is v_0, the inputs are the longitudinal acceleration a_x and steering angle ϕ. The rear wheel follows an arc with the radius R, which is calculated from the steering angle ϕ and the vehicle wheelbase L.

Finally, as shown in Fig 2.10, the change of the car position in the global coordinate frame is given by the following Equations:

$$\Delta\theta = \dot{\theta}\,\Delta t \tag{2.9a}$$

$$l = \begin{cases} v\,\Delta t & \text{if } \dot{\theta} = 0 \\ 2\,R\,\sin\left(\frac{\Delta\theta}{2}\right) & \text{otherwise} \end{cases} \tag{2.9b}$$

$$\Delta x = l\,\cos\left(\theta + \frac{\Delta\theta}{2}\right) \tag{2.9c}$$

$$\Delta y = l\,\sin\left(\theta + \frac{\Delta\theta}{2}\right) \tag{2.9d}$$

where $\Delta\theta$ is the difference between the heading vector angles of two sample time Δt. The arc chord length of the rear wheel rotation is l. The change of the rear axle position on X-axis and Y-axis are denoted with Δx,and Δy. The initial heading angle is denoted with θ. The angle between the arc chord and the heading vector is $\frac{\Delta\theta}{2}$.

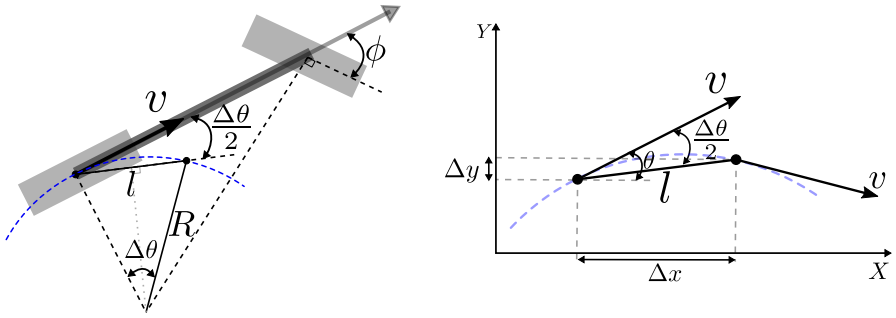


Fig. 2.10: The change of the vehicle position in the global coordinate: the rear wheel follows the blue dashed arc with the radius R. The related steering angle is ϕ. The difference angle between two sample time is $\Delta\theta$. The angle between the related arc chord l and the heading vector is $\frac{\Delta\theta}{2}$. In the right picture, the center of the rear wheel at two sample time is shown. The change of the rear axle position on X-axis and Y-axis are denoted with Δx,and Δy. In this picture, the angular velocity is negative (CW), and the sign of $\Delta\theta$ therefore is negative.

i-MiEV Throttle Model

In order to generate feasible commands for the low-level velocity and steering controller, the actuator constraints should be considered in the trajectory planner. Therefore, it is necessary to examine the vehicular response to the accelerator pedal position (**APP**) and the brake pedal position (**BPP**).

Our testbed, i-MiEV, benefits from a high-efficiency permanent magnet synchronous motor[1] (PMSM) with 180 Nm maximum torque in a rotational speed range of $0-2600\,\text{rpm}$[2] and can provide maximum mechanical power 49 kW in the range of $2600-8000\,\text{rpm}$. The vehicle reaches speeds of up to 130 km/h. Fig. 2.11 shows motor characteristics (torque versus rotational speed) [56]. Based on the recorded data, the maximum voltage and current

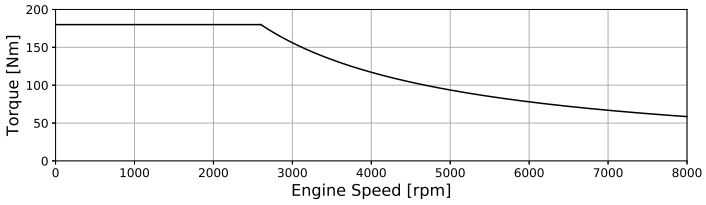


Fig. 2.11: Characteristic curve of Y4F1 (i-MiEV electric motor) taken from [56]: the characteristic curve determines the motor torque output at a specific rotational speed. The maximum torque is 180 Nm.

[1] Motor model is Y4F1, and the final gear ratio between motor and wheels is 7.

[2] Consider that 1 rpm=$\frac{\pi}{30}$ **rad**/s.

of the battery are about 300 V and 190 A, and the maximum power of the car can reach 57 kW. Therefore, the car energy conversion efficiency is $\eta \simeq \frac{49kW}{57kW} = 0.86$.

A standard system identification procedure is used to find the car model parameters. Data (such as speed, acceleration, and power) are collected in response to different accelerator pedal positions (APPs), and the data is then fitted to an appropriate curve. Ten APP inputs, $10\% - 100\%$ of the maximum accelerator pedal displacement, were given to the system in the form of step inputs, and the resulting data was collected. The car drove in a straight line, *i.e.*, with zero steering angle ($\delta_f = 0$).

The recorded data is used to create a car acceleration map based on the speed and the APP (step input) (Fig. 2.12). This map can be used to predict the car behaviour in different step APPs and speeds. However, as our test field length is limited to 2 km, we cannot test the car up to the full speed. Also, data were recorded from zero speed with different APP, so the car's acceleration with low APP and at high speeds are missed. In this situation, interpolation of the lookup table will not give us a correct answer. Missing data can be found based on a mathematical model. Therefore, the system is modeled as a mathematical relation between a step input (accelerator/brake pedals position) and the system output (acceleration).

Fig. 2.13 shows the motor power and acceleration data versus speed in the different APPs. While the APP changes with a step input, the power output changes smoothly through a curve. As we do not have access to the ECU model, an RC circuit is used as its system model, whose resistor changes by the APP, and its capacitor models the delay in the system. The

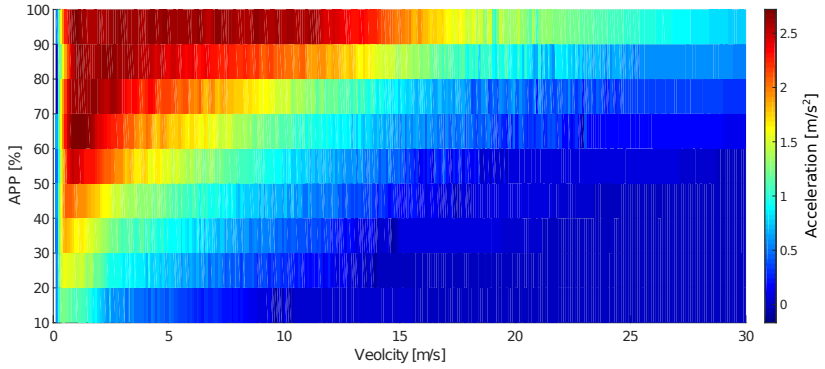


Fig. 2.12: The i-MiEV acceleration map based on the accelerator pedal position (APP) and the car longitudinal speed: the recorded data is used to create the car acceleration map. The color bar illustrates the acceleration amount, the darkest blue shows that the car can not accelerate with the related APP in the specified speed (the car can not reach the speed after that with the same APP). The darkest red shows the maximum acceleration value which is measured with the maximum throttle in the low speeds.

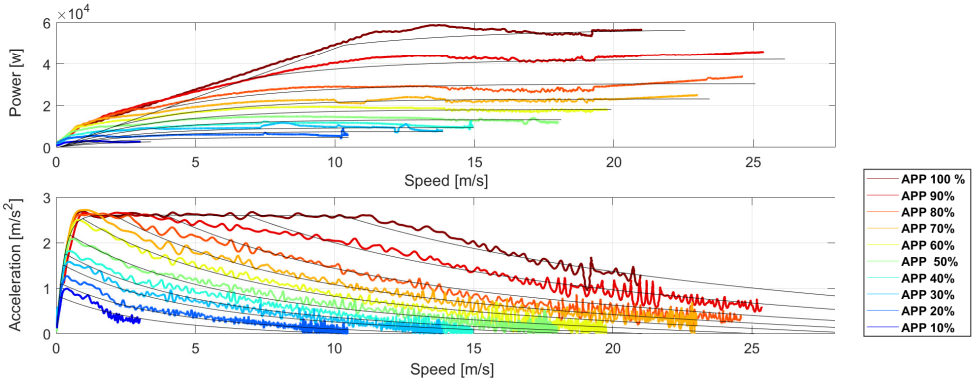


Fig. 2.13: Recorded data when i-MiEV was accelerated and their mathematical model curves (given by (2.11),(2.12)): the vehicle power and acceleration are plotted against the vehicle speed in the different accelerator pedal positions (APP) which are represented by different colors. The darkest blue indicates 10% of maximum accelerator pedal travel, and the darkest red indicates the full accelerator pedal travel. First, the acceleration is constant until a certain speed threshold; there is a constant dead time at the beginning of the time frame. After the speed threshold is met, the acceleration reduces by increasing the speed. The threshold changes by the APP percentage. The black curves are the model results.

relation between the APP and the throttle value (u_{ECU}) is given by,

$$R_1 = R_0 + \alpha_1 u \tag{2.10a}$$

$$u_{ECU} = R_1 \left(1 - e^{\frac{-t}{C_u R_1}}\right) \tag{2.10b}$$

in which u is the APP normalized to $0-1$. Here we assume that the accelerator pedal is connected to a potentiometer and changes the circuit resistor (R_1). R_1 changes linearly (with slop value $\alpha_1 \simeq 0.5$) proportional to the APP, and $R_0 \simeq 0.5$ is the initial resistor value. The exponential form with the RC time constant ($C_u R_1$) is the normalized voltage across the capacitor. As all parameters are normalized here, C_u is estimated around 5.15. For full throttle, RC time constant is around 5.15s. The time t starts to increase just after the accelerator pedal is pressed, and resets when the accelerator pedal is released.

The motor provides nominal torque when its throttle input is maximum $u_{ECU} = 1$, more specifically when its nominal voltage is supplied. The motor current is limited according to its rotational speed to bound the torque. In other words, the motor needs less power at lower velocities and the torque is constant in the low rotational speed (denoted as τ_{max} in (2.11a)). Then, when the motor reaches its maximum power, the torque is inversely proportional to the engine speed. The maximum power is limited based on the throttle level, according to

(2.11b). The relationship between power and the car speed and throttle is given by:

$$F_1 = \frac{G_r}{r}\tau_{max} \tag{2.11a}$$

$$F_2 = \frac{\eta\ P_{max}\ u_{ECU}^b}{v} \tag{2.11b}$$

$$F = min(F_1, F_2) \tag{2.11c}$$

$$P = \frac{F\ v}{\eta} \tag{2.11d}$$

where F_1 is the drive force when the motor torque curve is at its highest value τ_{max}. The gear ratio between the motor shaft and the wheel is denoted with $G_r \simeq 7$. The wheel radius is denoted with $r \simeq 0.31$ [m]. The drive force is F_2 when the motor power reaches its maximum value $\eta P_{max} u_{ECU}^b$ - a value which is limited by ECU command. The car energy conversion efficiency is $\eta \simeq 0.86$. The constant value $b \simeq 5.7$ is power of u_{ECU} to limit the maximum motor power non-linearly based on the APP. The parameter b is found when fitting the recorded data using the method of nonlinear least squares. The car speed is denoted with v. The final drive force (F) is the minimum of F_1 and F_2. The parameters of the equations are found using the method of nonlinear least squares when fitting the recorded data. The electrical power P is then calculated based on the mechanical power ($F\ v$).

Finally, the relation between velocity (v) and acceleration (a_+) at a certain APP is described as (2.12). The parameters $c_a = 0.45$[kg/m], $c_f = 0.012$[kg/s] are found using the real data plotted in Fig. 2.13. The system time delay is modeled with a first-order system, in which $T_e \simeq 0.45$[s].

$$a_+(v,u,t) = \frac{F\left(1 - e^{\frac{-t}{T_e}}\right) - c_a v^2 - c_f v}{m} \tag{2.12}$$

i-MiEV Brake Model

The brake model also is provided based on the recorded data. The i-MiEV brake model is given by:

$$a_-(u_b,t) = -a_f u_b\left(1 - e^{(-t/T_d)}\right) - c_0 v \quad \text{if } |v| > 0 \tag{2.13}$$

where u_b is the normalized brake pedal position (BPP) in range of [-1,0], and $a_f \simeq 7.2$ [m/s^2] is the maximum deceleration that the maximum brake pressure causes. If no pedals are pressed in i-MiEV, it will accelerate and reach 1 m/s speed; the initial deceleration is modeled by the second term $c_0 v$ with $c_0 \simeq 0.15$ [s^{-1}]. The time delay (T_d) in the brake system is almost 0.1 [s]. The deceleration model prevents the car from increasing the speed value. Therefore, this equation is valid when the speed value is larger than zero.

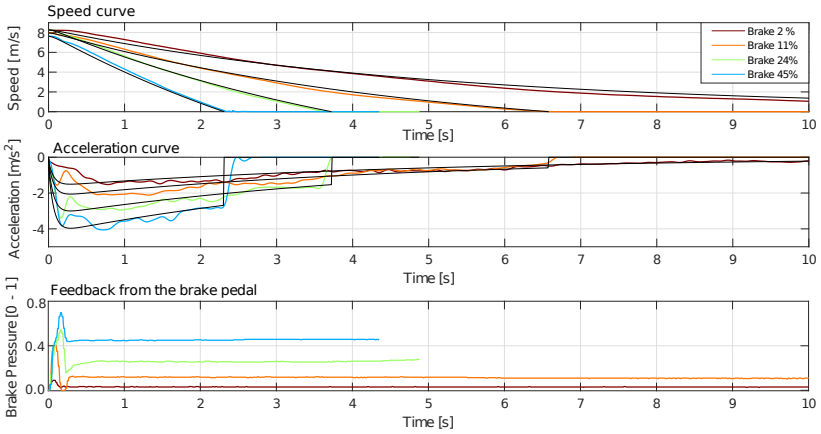


Fig. 2.14: Recorded data when i-MiEV was braked and their mathematical model curves (given by (2.13)): the braking tests were done for different brake pedal position commands while the initial speed was 8 m/s. The tests data including the vehicle speed, the acceleration, and the normalized brake pedal pressure of i-MiEV over time are plotted. The colors indicate the different percentage of maximum brake pedal travel. We did not test the full brake pressure in this speed as it causes a high force and consequently intolerable passengers discomfort. The black curve are the related mathematical model curves.

The final car acceleration model derives from adding the both acceleration models a_+ and a_-:

$$a(v,u,u_b,t) = a_+(v,u,t) + a_-(u_b,t) \tag{2.14}$$

Notice that if the brake pedal is engaged ($u_b > 0$), the accelerator pedal will be disengaged ($u = 0$).

MIG Throttle Model

Unlike the i-MiEV–an electric car, the MIG model is a petrol car therefore has more parameters to be considered. In the MIG throttle model, the nonlinear petrol engine model, the gearbox, and the time delay of closing clutch are considered.

The same system identification procedure as the previous subsection is followed by collecting data with the APP inputs, from 0% to 100% of the maximum APP. The experiments started with the open clutch. The collected data consists of the engine revolution, the car speed, the car acceleration, the current gear, the accelerator pedal voltage, and the brake pedal pressure.

First, the engine speed, the vehicle speed, and the gear numbers are recorded with different APP percentages while the gears change automatically. The recorded data is sorted based on the gear numbers, as shown in Fig. 2.15. The gear ratios between engine and vehicle

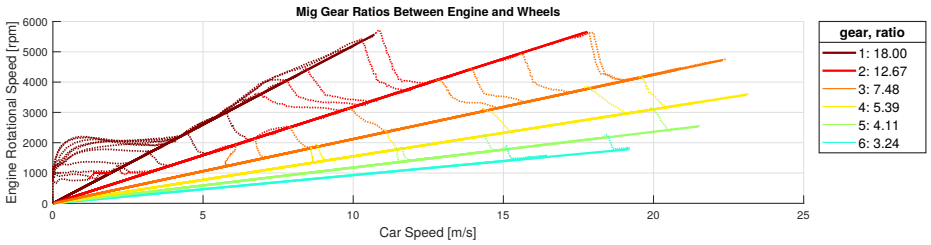


Fig. 2.15: The MIG's recorded data - the vehicle speed versus engine rotational speed in different gears: data are assorted based on the gear numbers and are represented by different colors. The gear ratios between engine shaft and the vehicle wheel are given by (2.15) and specified in the figure legend. The radius of the vehicle wheel is 33.25 cm.

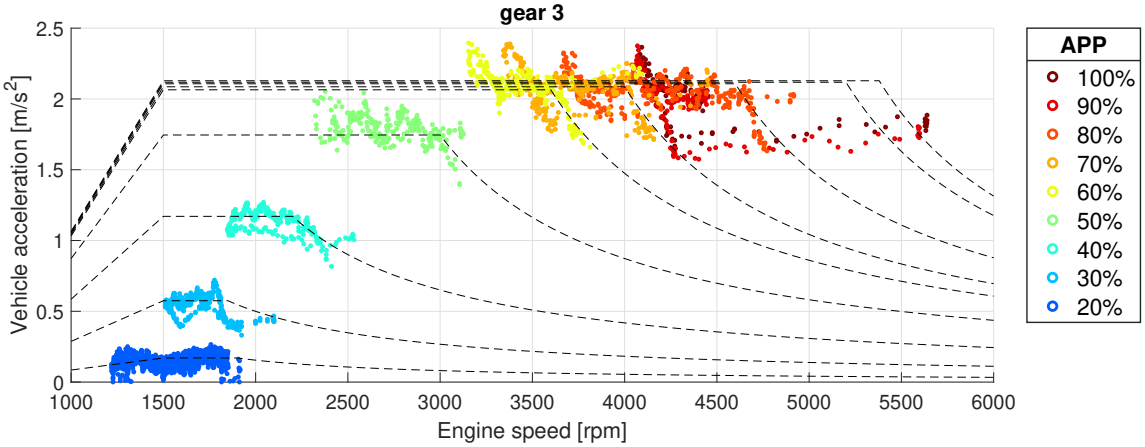


Fig. 2.16: Recorded data of engine rotational speed versus vehicle acceleration in gear three: data was recorded with different percentage of maximum accelerator pedal travel and are represented by different colors. The black dashed lines show the result of fitting data to the proposed model (2.16).

wheels is given by:

$$\omega = \frac{60 G_r}{2\pi r} v \quad \text{with} \quad G_r \in \{18, 12.66, 7.47, 5.39, 4.10, 3.24\} \tag{2.15}$$

where r is wheel radius[1], and G_r is the gear ratio. The ratios are found by fitting data to lines based on the gear numbers.

The next step involves finding the nonlinear model of the petrol engine. Fig. 2.16 shows the car's longitudinal acceleration versus engine speed from the time that the clutch is open until the gearbox stays in gear one. The data is grouped based on the APP percentage. The engine revolution is always above a certain threshold (Ω_0). When the clutch is closed, the engine model can be divided to three parts, Ω_0 to Ω_1, Ω_1 to Ω_2, and above Ω_2. In the first speed range, the torque or the final acceleration is proportional to the engine speed. In the

[1]The tire model is 215/60 R16, which means the tire diameter is 665 mm.

second part, the acceleration is constant. In the third part, the engine torque drops. The acceleration therefore is inversely proportional to the speed. The effects of changing the clutch from open mode to close mode is modeled over time. To sum, the vehicle acceleration model $a_e(\omega, u, g)$ and the engine torque model $\tau_e(\omega, u, g)$ are formulized as follow:

$$a_e(\omega, u, G_r) = \frac{G_r}{rm}\tau_e \tag{2.16a}$$

$$\tau_e(\omega, u, G_r) = \begin{cases} K_0 K_1(u)(\omega - \Omega_0) & \text{if } \Omega_0 \leq \omega \leq \Omega_1(u) \\ \tau'_{max} = \tau_{max} K_1(u) & \text{if } \Omega_1(u) \leq \omega \leq \Omega_2(u) + 1 \\ \frac{\tau'_{max}}{K_2(\omega - \Omega_2(u))} & \text{if } \Omega_2(u) + 1 \leq \omega \end{cases} \tag{2.16b}$$

in which m is the vehicle mass, r is the wheel radius, and G_r is the gear ratio, u is normalized throttle voltage, ω is the engine rotational speed. The line slope of the first part is specified by a coefficient K_0. By increasing the accelerator pedal voltage, the throttle valve - which allows air flow in the engine - opens more. Therefore, the engine torque is proportional (by coefficient $K_1(u)$) to the APP; It is not, however, linearly proportional. In the second part, the engine can provide the maximum torque τ_{max}[1] while its rotational speed is between $\Omega_1(u)$ and $\Omega_2(u)$ at the full throttle. The maximum torque τ'_{max} is proportional to the APP coefficient $K_1(u)$. By decreasing the gear ratio, the output torque of the gearbox (the car wheels' torques) decreases while the output speed increases. Therefore, by increasing the vehicle speed, the gear ratio decreases to keep the engine rotational speed in the second part. In the third part, after a certain speed revolution (in correspondence with the APP) the engine torque drops, which causes to drop the acceleration. The slop at drops point can be specified with a factor of K_2.

As shown in Fig. 2.17, the gear shifts automatically when the acceleration drops in the third part of the model. By increasing the gear, the gear ratio between engine and wheels (G_r) decreases; consequently, the engine rotational speed decreases. The time of gear shifting is 0.5 s (50 points in 100 Hz sample rate)[2].

In addition to the engine model, parameters such as rolling friction, air resistance, and the

[1] The maximum useful torque for MIG is estimated at the third gear to be 155 Nm while the car mass is 1650 kg and the wheel radius is 33 cm.

[2] 50 points at the beginning of each gear shifting are not plotted to keep the plot intelligible. These points connect the last point of the previous gear to the first point of the next gear

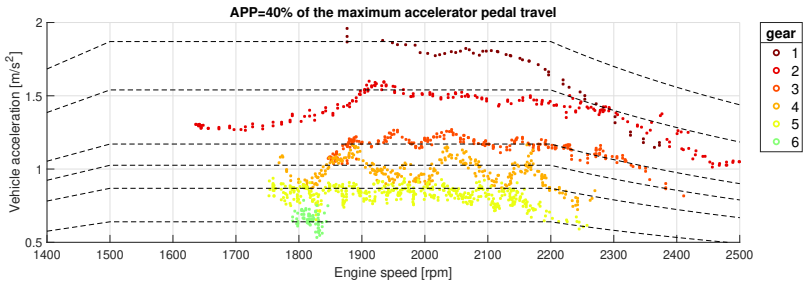


Fig. 2.17: Recorded data of vehicle acceleration versus engine rotational speed with different gears at 40% of the maximum accelerator pedal travel: the gear changes from 1 to 6 and are represented by different colors. The modeled fitted to each gear data is divided to three parts: 1) increasing acceleration linearly , 2) constant acceleration 3) decreasing acceleration non-linearly. Data, when the clutch is completely engaged, are plotted here.

effect of closing the first clutch[1] are represented in the acceleration model as:

$$a_+(\omega,v,u,t) = a_e(\omega,u,G_r)f(t) - \frac{c_a v^2 + c_f v}{m} \tag{2.17a}$$

$$f(t) = \begin{cases} 0 & \text{if } \omega \leq \Omega_0 \\ max(1, A - K(T_c - t)^2), & \text{if } \Omega_0 \leq \omega \end{cases} \tag{2.17b}$$

in which the coefficient related to the air drag and the rolling friction are c_a and c_f. The effects of closing the first clutch is modeled by $f(t)$. By closing the clutch, the friction of clutch disk is added to system which causes the acceleration to drop. The clutch starts to close at the time T_c. The clutches will open if the engine revolution reaches Ω_0 and $u = 0$ (the accelerator pedal is released). When the accelerator pedal is pressed, the clutch starts to close again; this moment is considered to be the initial time $t = 0$ in the equation. Parameters A and K are the model coefficients obtained by fitting the curve to the raw data. By multiplying the initial engine model (a_e) to $f(t)$ the final model is obtained. An example is shown in Fig. 2.18.

In the simulation, the car states (the vehicle speed $v(t)$ and the engine speed $\omega(t)$) in acceleration phase are updated by:

$$v(t) = a_+(t)\delta t + v(t-1) \tag{2.18a}$$

$$\omega(t) = \frac{60 G_r}{2\pi r} v(t) \tag{2.18b}$$

[1] MIG is equipped with dual-clutch transmission (DCT). It uses two separate clutches for odd and even gear sets. The second clutch is also used for reverse. Both clutches are disengaged when the car stops [57].

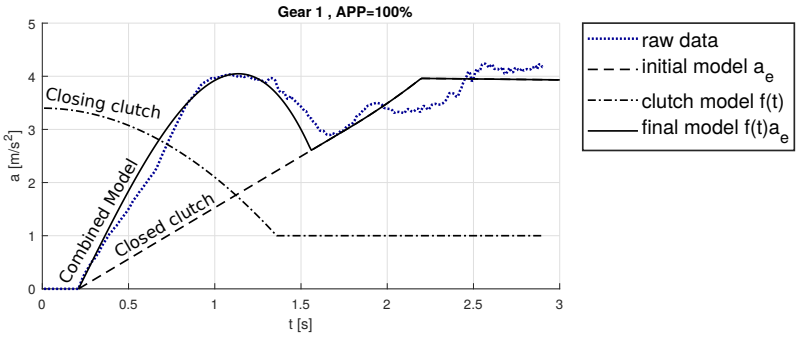


Fig. 2.18: Effect of switching from open clutch mode to close clutch mode: when the clutch is closed, the car's acceleration (a) increases linearly at low speeds. This is illustrated as the initial model with dash line. However, as the clutch is closing, the acceleration increases non-linearly, the effect of closing the clutch is modeled separately (dot-dash line) and is multiplied to the initial model which provides the final model (solid line). The blue points are the recorded data while the vehicle accelerated with maximum accelerator pedal travel at gear1.

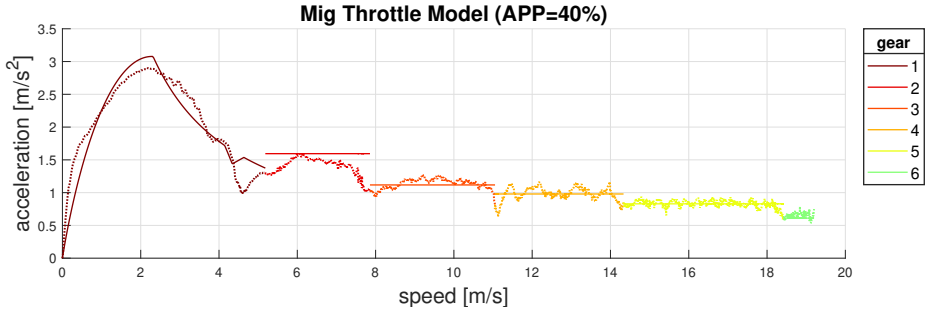


Fig. 2.19: Recorded data when MIG was accelerated with a constant accelerator pedal position (40%) and the vehicle model result given by (2.17): the points represent the recorded data and the solid lines represents the result of the model. The different colors represent the different gears.

in which $a_+(t)$ is obtained from (2.17a). We assume that the acceleration for the small time interval δt is constant. The engine rotational speed ω is calculated from the vehicle speed based on the wheel radius (r), and gear ratio(G_r). The gears change automatically when the clutch is closed and the engine revolution reaches Ω_2. Fig. 2.19 shows the fitted model to the raw data. The car behaviour is simulated using this model.

On the other hand, as shown in Fig 2.20, a time independent model is needed to predict the maximum acceleration which the car can provide at any speed. In the next chapters, the model is used to ensure that the generated trajectory can be followed by the controller. As shown in the previous functions, while the clutch is closing, the system provides a high acceleration in low revolutions which is proportional to throttle value. To provide a comfort ride for passengers, the throttle output in our controller is limited to 20% of the maximum throttle voltage at low speeds. It is also limited to 50% of the maximum throttle voltage at

the all speeds to avoid unnecessary acceleration. The maximum throttle value can change between this two values over a ramp. Therefore, for modeling just the raw data under 50% throttle is considered, and also data of the throttle above 20% with open clutch is omitted. The model is given by:

$$a_+(v) = max(0.25, -0.0109v^2 + 0.264v + 0.23) \tag{2.19}$$

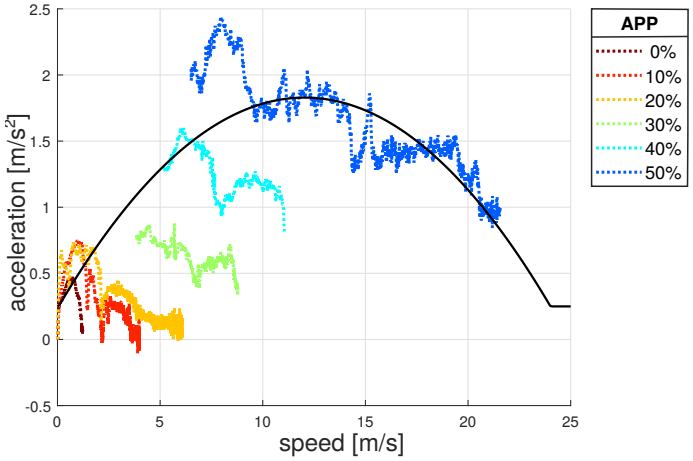


Fig. 2.20: A time independent model of the vehicle acceleration versus the vehicle speed in MIG (a petrol engine car): the maximum possible acceleration at each speed is modeled by fitting a curve to the recorded data. The model is represented by the solid line curve. The data contains only the APP under 50% of the maximum accelerator pedal travel. The APP in our controller is limited to 20% of its maximum for the low speeds; therefore, the data of the APP above 20% of the low speed is omitted. The different colors represent the different APP percentage.

MIG Brake Model

The MIG brake model is given by:

$$a_{-}(t) = \left(-a_{\omega} - a_f u\right)\left(1 - e^{(-t/T_d)}\right) \tag{2.20}$$

where a_f is the maximum deceleration at maximum braking, the parameter a_{ω} tales the clutch friction into account, the time delay (T_d) in the MIG brake system is almost 0.1 s. The clutch opens when the engine revolution reaches 1000 rpm by which the friction decreases. As shown in Fig. 2.21, the gears still engaged when both the throttle and brake pedals are released. For example, after the gear shifts from 5 to 4[1], the engine revolution increases while the vehicle speed decreases. By shifting to the gear 2, the clutch opens and the clutch friction force decreases. By pushing the brake pedal, the gear does not change, and the engine revolution decreases proportional to the car speed till it reaches 1000 rpm. Then, the clutch opens and the friction reduces. The clutch friction is modeled with a constant value.

In this subsection, the car throttle and brake models are provided.

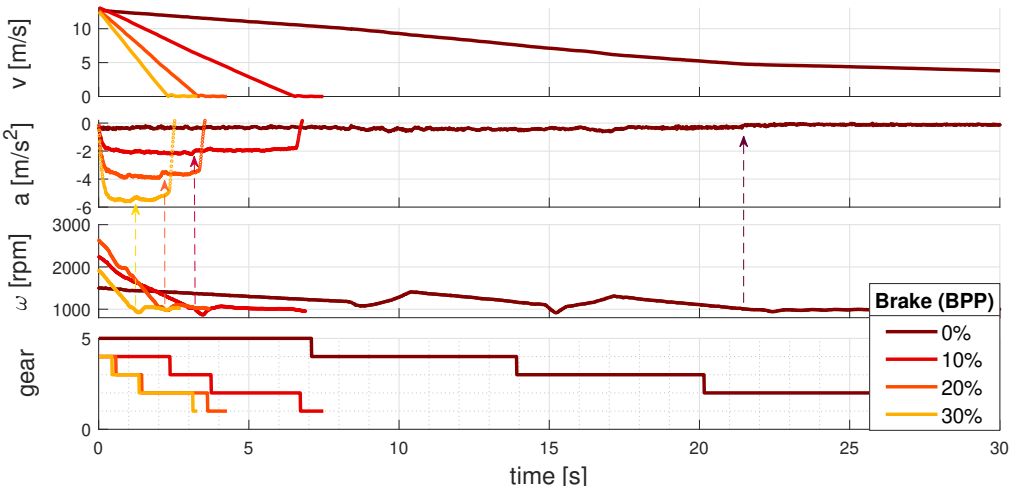


Fig. 2.21: Recorded data when MIG was braked: the braking tests were done for different brake pedal position commands while the initial speed was around 13 m/s. The recorded data including the vehicle speed (v), the vehicle acceleration (a), the engine rotational speed (ω), and the car gear over time are plotted. The colors indicate the different percentage of maximum brake pedal travel. The arrow shows the time that the engine revolution reaches 1000 rpm - at which point the clutch opens and its friction force is omitted and the absolute value of deceleration therefore is decreased.

[1] When the gear shifts from 5 to 4, the gear ratio therefore increases from 3.24 to 4.10.

2.4 Driving Style

The goal of autonomous cars is to transport people from one place to another while providing them with comfort and free time. However, people's comfort zones differ vastly. Everybody has their individual comfort levels, limits, and desires. Recently, several studies have introduced different factors which define the level of passenger comfort, such as time to collision, lateral and longitudinal acceleration, and jerk [58]. These factors can be adjusted in the trajectory based on the passenger's desires. In [59], the parameters are tuned automatically using a feature-based inverse reinforcement learning method by asking the user to drive the car manually for 25 minutes. Experimental tests on highways for speeds between 20 m/s to 30 m/s show average acceleration around 1 m/s^2 and average jerk around 0.3 m/s^3.

We gathered the data from half an hour of manual driving inside the city with two different cars (i-MiEV and MIG). The goal was to find the workspace of manual driving in terms of jerk and longitudinal acceleration. Fig. 2.22 shows the data from manual driving with the electric car i-MiEV. In terms of speed up in i-MiEV, the maximum acceleration is bounded by 2.75 m/s^2 due to the car constraint. In the deceleration phase, the lower acceleration bound is -3.4 m/s^2 which is a high value, because the car must deal with the emergency situations. The median acceleration and deceleration, and the mean absolute jerk are better criteria to determine the passenger comfort zone. The median and boundary values of acceleration and deceleration, and the mean of jerk are also extracted from data of the test with MIG. Table 2.2 summarizes the test results. The median acceleration in the case of electric cars is larger than that of petrol cars as the electric cars accelerates with higher value at low speed (by pressing the accelerator pedal slightly[1].), and consequently the maximum jerk and average jerk are higher.

In this project, the maximum jerk and acceleration of the trajectory points are chosen with the trial-and-error procedures to keep the average jerk under 0.5 m/s^3. As we will see in

Table 2.2: Result of half an hour manual driving with i-MiEV and MIG: the goal was to find the workspace of manual driving in terms of longitudinal acceleration (a) and jerk (j). The median of acceleration and deceleration, maximum jerk, and the jerk absolute mean are denoted with med(($\mathbf{a}_-, \mathbf{a}_+$), max($|\mathbf{j}|$), and $\overline{|\mathbf{j}|}$, respectively.

Car Model	$\mathbf{a}$ [m/s^2]	med($\mathbf{a}_-, \mathbf{a}_+$)[$m/s^2$]	max($\|\mathbf{j}\|$)[m/s^3]	$\overline{\|\mathbf{j}\|}$[$m/s^3$]
i-MiEV	[-3.40,2.75]	[-0.98,0.96]	11.01	0.49
MIG	[-3.42,2.57]	[-0.97,0.8]	6.77	0.36

[1]For example by pressing the accelerator pedal 30% of the maximum accelerator pedal travel, i-MiEV accelerates with 1.5 m/s^2 while MIG accelerates with 0.75 m/s^2

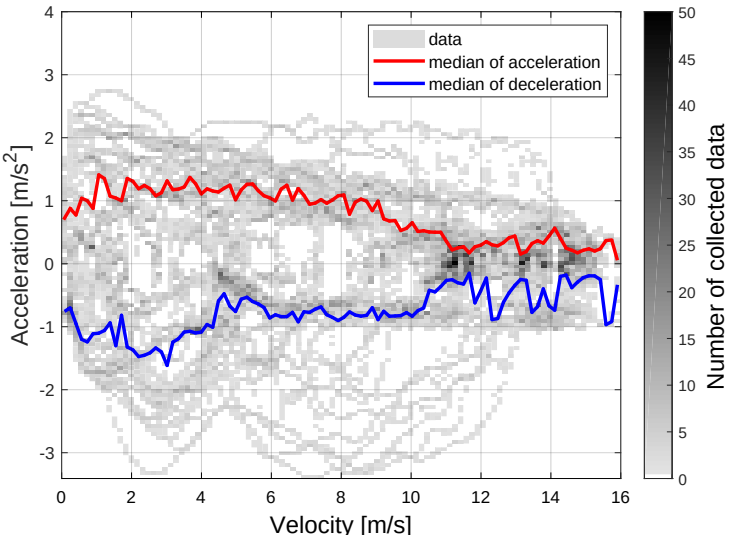


Fig. 2.22: The Applanix data was recorded to show how a human driver drove a car (i-MiEV) on the street with a 13.8 m/s speed limit. The acceleration results are over the interval $[-3.40, 2.76]$ m/s^2. The number of data at each discretized grid 0.16 m/s,0.06 m/s^2 is colored over grayscale (described in the legend). The darker color shows bigger number of data. The red line show the median value of the positive acceleration. The blue line shows the median value of the negative acceleration (deceleration).

the next chapter, the acceleration for trajectories is limited between -1 and 0.7 m/s^2 and the absolute jerk limited by 0.85 m/s^3.

2.5 Generating Structured Road Maps

An essential requirement for any trajectory planner is a map of the environment, *i.e.*, road or street on which the autonomous car drives, the specified car position, and its destination on the map. Maps of urban areas have become increasingly precise, popular, and publicly accessible[1]. The structured road maps contain nodes and arcs: nodes represent important features, such as entries and exits, while arcs indicate drive lanes and street borders between neighboring nodes. The drive lanes are defined as splines. One of the first structured road maps for autonomous car driving was a simple high-definition (HD) map Route Network Definition File (RNDF) [60] used in the DARPA challenge in 2007. While driving in a structured map has restrictions - *e.g.*, remaining within lanes, overtaking only from the side - finding the trajectory while using constraints is rather quick since the search space is limited. In this thesis, the Atlas road map [61] is used to generate the test road maps[2].

[1]There are many different companies developing HD maps, including HERE, DeepMap, Civil Maps, Carmera, TomTom, lvl5, *etc*.

[2]Atlas road map was the initial TomTom HD map structure.

To generate the structured road map of a test field, we first manually drive the car in the center lane of a street and record the car path (using *e.g.*, GPS). Then the lane border points are generated either by considering the width of the street or by filtering the LiDAR sensor data; LiDAR data corresponding to the lane street borders have higher intensity than other ground points. A structured map should have the following properties:

- it should provide us with a smooth drive spline at the center of the lanes,
- it should be searchable, such that in each trajectory generation step the closest point, the look-ahead point, or the adjacent point can be found computationally efficient.

Recorded path points are manually categorized to different segments of a street or junctions between streets. Then, a cubic Akima spline [62] is aligned to each segment. These splines are called drive splines in the rest of this dissertation. Furthermore, the lane borders splines are generated by shifting the drive spline points to the left and right.

2.6 Conclusions

In this chapter, I explained the system model and the required software structure for developing and implementing trajectories. The model of the system, throttle, and brake were derived theoretically and empirically. In the next chapters, these models are used to generate feasible trajectories for the low level controller. The limitation of acceleration and jerk -which are required for passenger comfort- are also determined. These constraints will be incorporated when determining the velocity and acceleration of each trajectory point. The lateral and longitudinal controllers are described in detail. The controllers' structures impose limitations on trajectory replanning. These limitations will be made clear in the next chapter. The rest of this thesis focuses on developing different trajectories considering structured maps, car model, actuator models, and passenger comfort.

Chapter 3

Reactive Trajectory Planning

In the near future, our streets will have a mix of autonomous and human-driven cars. It is therefore essential that the actions of autonomous vehicles be predictable for human drivers. If the behavior of autonomous cars be unpredictable, human drivers may become nervous and lose their control, thus leading to generally unsafe and unreliable streets. The passengers inside an autonomous car should also feel comfortable and secure, and the actions of the autonomous vehicles in traffic or in front of pedestrians should be reasonable and predictable for the passengers.

This chapter describes a reactive trajectory planning method developed to achieve smooth behaviors in autonomous driving; the behaviors will be predictable and comfortable for passengers and other human drivers. The planned speed is kept as high as possible to avoid impatience in other drivers–a condition which may lead human drivers to dangerously overtake. The actuator constraints are also taken into account to limit the maximum desired acceleration which makes the trajectory a feasible command for the controller.

In the process of developing using the the proposed method in this chapter, it is assumed that a global path is provided. Then, a local path from the nearest point on the global path to a point S meters ahead of the car along the global path is created by sampling with δ_s meters, providing a sequence of points. The curvature at each point and speed limit extracted from the map give the speed constraint at each point. Obstacles on the road are considered on deciding whether to swerve the path away from the obstacle or to adapt the speed constraint. Swerving provides enough space to pass the obstacle, and limiting the speed avoids the obstacle collision. Finally, the car dynamic constraints and passenger comfort, which limit the acceleration and jerk[1], respectively, are considered to provide a continuous and smooth **speed profile**[2]. The look-ahead point technique is used to keep the trajectories smooth and

[1] Jerk is the time derivative of acceleration.

[2] Speed profile refers to speed along the path, which is the points' speed against their longitudinal distance.

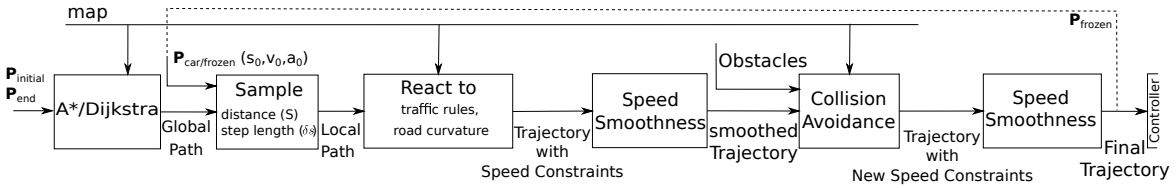


Fig. 3.1: Reactive trajectory planning: it is assumed that the global path is provided from an initial point to an end point. The global path is sampled S meters from the car position with step length δ_s to provide a local path. The traffic rules and road curvature define the initial speed (speed constraint) for each path point. The speed profile is smoothed based on the acceleration and jerk constraints. Then obstacles are considered, which shifts the path away from obstacles and/or reduces the speed of the points. By smoothing the trajectory considering the new speed constraints, the final trajectory is provided as a controller input. The first part of the final trajectory is frozen until a look-ahead point, which is used as the first point for the next trajectory.

continues [63–65] (explained in the previous chapter). The trajectory look-ahead point is an input for the controller. The first part of the trajectory is frozen until the look-ahead point, which is used as the first point for the next trajectory planning.

3.1 Generating the Initial Trajectory

To drive an autonomous car from an initial position to a goal, a global route should be generated offline. There are different methods to find the global route, *e.g.*, using A* or Dijkstra in a structured road map [66]. The global route consists of a sequence of road segments and the lanes in which the car can drive. The global route is used as the reference for the local trajectory, which is in the range of autonomous car sensors (almost 100 meters). The local trajectory must follow the global route direction and adjust the speed at each point based on the traffic rules, traffic flow, passenger comfort, and car limitations.

The distance and parameters of this chapter are defined in Frenet coordinate system as described in previous chapter in section 2.1. The local trajectory is designed in two steps. The first step is limiting the maximum speed of the initial path (map drive spline points) based on the static structured environment and road information. Then, the desired speed and position of each point of the trajectory are adjusted based on dynamic information such as obstacles and traffic lights.

3.1.1 Reaction to the Road Information

In the first step, a local path is generated by sampling the drive splines (along the desired lanes taken from the global plan) with a predefined step length (δ_s) in the space domain.

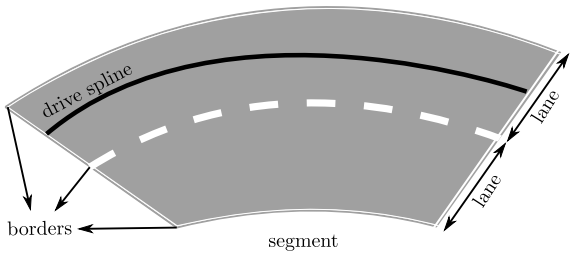


Fig. 3.2: A segment at a structured road map: each segment contains information of lanes that connects the segment to the previous and the next segments. The map also provides the splines of the lane center and borders. A drive spline is the lane center spline, which is the best path on the lane to follow.

The initial speed of each point is calculated based on 1) the maximum allowable road speed (based on traffic lights, speed limits), 2) the car's lateral distance from the plan, and 3) the road curvature. Speed is inversely proportional to the curvature to guarantee the safety and comfort of the passengers. A complete road map is necessary to generate the trajectory. However, a human driver is responsible for bringing the car on the road when the map is incomplete. Therefore, the desired initial speed is reduced based on the vehicle lateral distance to the road.

Curvature Calculation

There are several ways to calculate the path curvature at a point [67]. One way is to use two other points. To calculate the curvature at a point P on the path, two points P_b and P_f with arc length l are selected on the path behind and ahead of P. As shown in Fig. 3.3, the angle between the two vectors $\vec{P_bP}$ and $\vec{PP_f}$ is equal to the angle β of an arc of length l. A reasonable l, *i.e.,* comparable to step length δ_s, should be selected to mitigate the map inaccuracies. Because a road curvature changes continuously and smoothly [1], a reasonable l helps us to better estimate the road curvature based on the road map precision. For Atlas road maps l =2 m is selected. We can calculate the curvature (κ) and related maximum speed (v_{max}) as:

$$\beta = cos^{-1}\left(\frac{\vec{P_bP}.\vec{PP_f}}{\|\vec{P_bP}\|\|\vec{PP_f}\|}\right) \tag{3.1a}$$

$$\kappa = \frac{1}{r} = \frac{\beta}{l} \tag{3.1b}$$

$$v_{max} = \sqrt{\frac{a_c}{\kappa}} \tag{3.1c}$$

[1] There are standards and constraints in road construction.

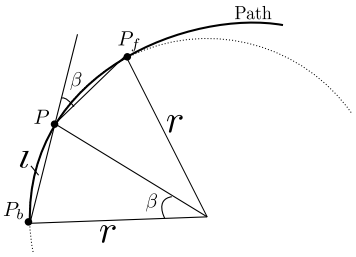


Fig. 3.3: The curvature at point P is calculated using two auxiliary points P_b and P_l with the arc length l. The angle between the two vectors $\vec{P_bP}$ and $\vec{PP_f}$ is β. The related circle radius is r.

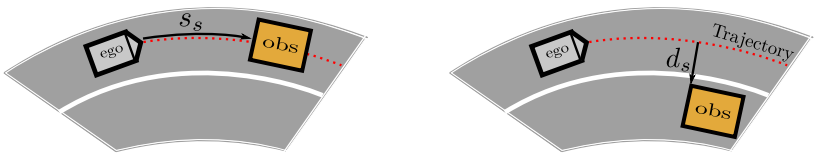
where β is the arc angle with l length, r is the related circle radius, and a_c is the maximum centrifugal acceleration, which is chosen based on passenger comfort.

3.1.2 Reaction to Obstacles/Collision Avoidance

Once the initial trajectory–based on the structured road map–is calculated, we must consider the dynamic obstacles[1]. To avoid obstacle collision, the initial trajectory speed is redefined while accounting for the longitudinal distance to the obstacle in front. At the same time, lateral distance to the obstacles redefines the optimal speeds. The initial trajectory is along the lane center. In the dynamic environment, sometimes it is necessary to drive to the left or right side of the lane center, *e.g.*, when the car wants to overtake a bus or truck. Therefore, the position of the trajectory points swerves from the lane center based on the safe lateral distance to the obstacles. As shown in Fig .3.4, the longitudinal and lateral safety distances from an obstacle, s_s and d_s, respectively, are calculated as:

$$s_s = \max(S_{min}, T_s v_o) \tag{3.2}$$

$$d_s = \min(D_{min} + T_d v_e, D_{max}) \tag{3.3}$$



a) longitudinal saftey distance b) lateral saftey distance

Fig. 3.4: Safety distances are given by (3.2) and (3.3): the red dot line is the trajectory, the longitudinal safety distance s_s is the minimum required distance along the path to the obstacle, and the lateral safety distance d_s is the minimum required perpendicular distance from an obstacle to the path.

[1]Dynamic obstacle could occasionally have zero speed.

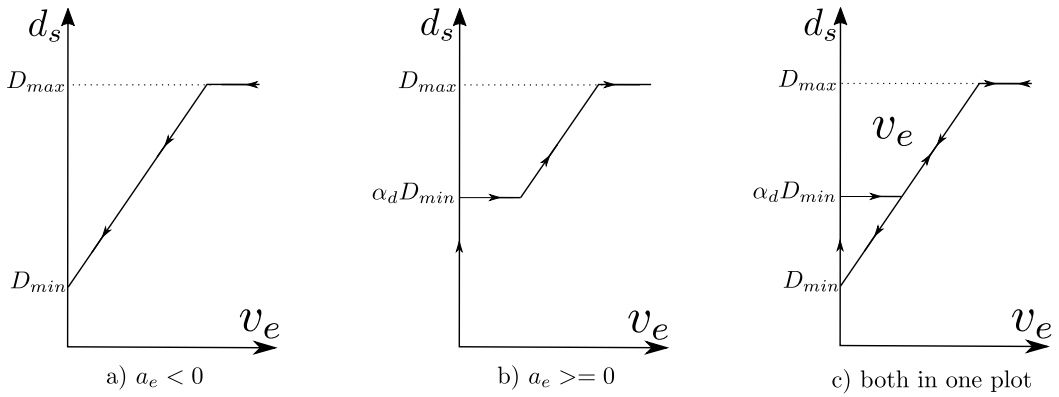


Fig. 3.5: Lateral safety distance (d_s) versus the ego speed (v_e): a lateral safety distance should be considered when passing a static obstacle. When the ego car reduces the speed (the acceleration of the car is negative $a_e < 0$), the minimum lateral distance to the obstacle (d_s) is proportional to the ego car speed and is limited between D_{min} and D_{max}. Under D_{min}, the car cannot pass the obstacle. When the lateral safety distance is bigger than D_{max}, it does not limit the ego car speed. The car can stop with D_{min} lateral distance adjacent to an obstacle. But it needs a bigger lateral distance ($\alpha_d D_{min}$,$1 < \alpha_d < \frac{D_{max}}{D_{min}}$) to speed up. The parameters can be tuned based on the sensors' quality and the car width.

where $S_{min} \in \mathbb{R}^+$ is the minimum longitudinal safety distance when the obstacle is stationary, v_o is the current absolute speed of the obstacle. The parameter $T_s \in \mathbb{R}^+$ is the time constant derived from traffic rules. According to the traffic rules, the ego car should have enough reaction and braking time in case of an emergency stop[1]. Thus, by keeping longitudinal safety distance, the ego car can follow the traffic rules. The longitudinal safety distance is proportional to the obstacle speed which limits the maximum desired speed of the ego car based on the current longitudinal distance between the ego car and the obstacle, as we will see later in this subsection.

The minimum lateral distance when the ego car stops is $D_{min} \in \mathbb{R}^+$. In order for the ego car to safely pass an obstacle a bigger lateral distance is required. The speed of the nearest trajectory point to the obstacle is denoted with v_e. The lateral distance is proportional to v_e with a constant value $T_d \in \mathbb{R}^+$. However, a constant value $D_{max} \in \mathbb{R}^+$ restricts the maximum necessary lateral distance. The constant values D_{min}, T_d, and D_{max} are defined based on the sensors' accuracy and passenger preferences. For example, if D_{min}=0.3 m, T_d=0.06 s, and D_{max}=2.0 m; then, the ego car must keep at least 0.3 m distance to the adjacent obstacle. The ego car can pass an obstacle with a high speed (higher than 28 m/s) by keeping 2 meters lateral distance.

When the car starts driving, (3.3) can cause rapid engagement and disengagement of accelerator and brake pedals. To avoid this discomfortable behavior, it is modified to a

[1]The reaction and braking time is around 2.5 s for a human driver.

piece-wise linear function, as shown in Fig. 3.5, expressed by:

$$d_s(v_e, a_e) = \begin{cases} \alpha_d D_{min}, & \text{if} \quad a_e \geq 0 \quad \text{and} \quad v_e < \frac{D_{min}}{T_d} \\ \min(D_{min} + T_d v_e, D_{max}), & \text{otherwise} \end{cases} \tag{3.4}$$

where a_e is the ego car acceleration, and $1 < \alpha_d < \frac{D_{max}}{D_{min}}$ is a tuning parameter. The discontinuity of this function will be smoothed, as we will see in the sequel, in the smoothing steps. To better clarify the functionality of (3.4) consider the following example: assume the ego car stops behind the traffic light near other cars. When the traffic light turns green, the other cars start to move. If the lateral distance to an obstacle changes around D_{min}, the planned speed will oscillate between zero and a positive value in (3.3). Such an oscillation engages and disengages the brake and accelerator pedals, which cause passenger discomfort. Modifying the formula as (3.4), the ego car will not start again until the desired trajectory has a bigger lateral distance to the obstacle. Therefore, the planned speed will change smoothly around a positive number.

Passing the obstacle by swerving maneuver

For each obstacle, I first checked the possibility of passing the obstacles only by swerving away from the obstacle. Swerving maneuvers shift the car laterally to the left or right of the path center to provide enough free space adjacent to the obstacles to overtake them. This decision procedure is presented in Alg.1 and clarified by Fig. 3.6. If the planned speed v_p at each point is bigger than obstacle speed v_o at the adjacent lanes, the car planned to pass the obstacle. In this situation, if the current lateral distance d_o is smaller than the safe lateral distance $d_s(v_e, a_e)$, the lateral distance is increased as much as possible. The car can swerve away from the obstacle until the swerving room border d_b, which is specified based on the adjacent obstacles on the other side and the lane borders. If the maximum lateral distance $d_o + d_b$ to the obstacle is still less than the safe lateral distance, then the safe speed v_c is calculated as:

$$v_c(d_o) = \begin{cases} 0, & \text{if} \quad d_o < D_{min} \\ \frac{d_o - D_{min}}{T_d}, & \text{if} \quad D_{min} \leq d_o \leq D_{max} \\ \infty, & \text{if} \quad d_o > D_{max} \end{cases} \tag{3.5}$$

where (3.5) indeed is the inverse of (3.3) while d_o is the current lateral distance of the obstacle from the desired path. If v_c is still bigger than the obstacle speed, we perform the swerving maneuver with the new lateral distance and the new desired speed. The new desired lateral distance d_s^* is the minimum of the initial required safe lateral distance and the possible

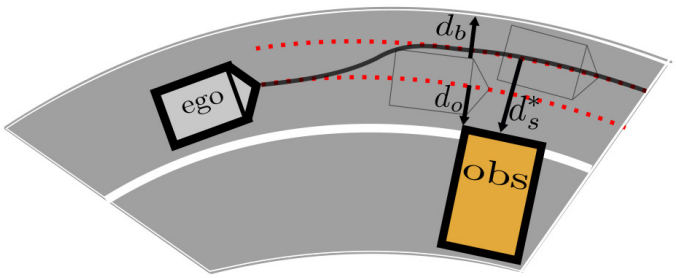


Fig. 3.6: Swerving maneuver: when the lateral distance between an obstacle and the path center (d_o) is not enough; if there is enough free space d_b to the lane border, the ego car can swerve from the path center to provide the sufficient lateral distance d_s^* to pass the obstacle.

maximum lateral distance. The new desired speed v^* is the minimum of the initial planned speed and the safe speed based on the new desired lateral distance. This desired swerve is reserved to be applied to the trajectory.

Algorithm 1: Overtaking decision by a swerving maneuver

input : the obstacle speed v_o, the planned speed and acceleration (v_p,a_p), the lateral distance between the obstacle and the path d_o, distance to the swerving room border d_b

output : Decision, the desired speed v^*, the desired lateral distance d_s^*

$d_s(v_p, a_p) \leftarrow$ (3.4)
$d_s^* \leftarrow \min(d_s, d_o + d_b)$
$v_c(d_s^*) \leftarrow$ (3.5)
$v^* \leftarrow \min(v_c, v_p)$
if $(d_o < d_s^*) \wedge (v^* \geq v_o)$ **then**
  Decision $\leftarrow$ True
else
  Decision $\leftarrow$ False

Slowing down policy

If there is an obstacle in front of the ego car, or if the ego car could not pass the obstacle around by using the previous algorithm, then it must keep the longitudinal safety distance d_x. To this aim, the speed limit v_d of each trajectory point based on its distance to the obstacle and obstacle speed is [68]:

$$v_d = \sqrt{max\,(0, v_o^2 + 2|a_{des}|(s_o - s_s))} \tag{3.6}$$

where a_{des} is desired deceleration (*e.g.* -1 $\mathrm{m/s^2}$), s_o is the longitudinal distance of each point to the obstacle, and s_s is the longitudinal safety distance (3.2). To better clarify this slowing down policy, the desired speed versus the obstacle distance for two different cases are shown in Fig. 3.7. The red curve shows the desired speed as a function of the longitudinal distance to a static obstacle, while the black curve assumes an obstacle with a constant speed of 10 $\mathrm{m/s}$ (shown by the dashed line). Following the black solid curve, the ego car keeps a safe longitudinal distance to the driving obstacle. We want to ensure that the ego car reaches the obstacle smoothly and keeps longitudinal safety distance to the obstacle.

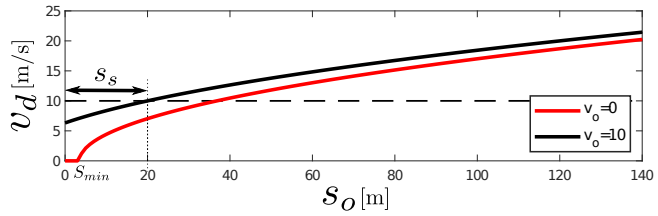


Fig. 3.7: The desired speed (v_d) with respect to the distance to the obstacle (s_o): if the obstacle stops, the desired speed follows the red curve to provide the desired minimum distance S_{min} to the obstacle while smoothly decelerating with a constant acceleration. If the obstacle moves, the red curve shifts to the left proportional to the obstacle speed. The desired minimum distance to the obstacle while both ego car and obstacle have the same speed is s_s. For example, if the obstacle has 10 $\mathrm{m/s}$ speed (black dash line), the desired speed follows the black solid curve. In this case, the ego car desired speed is less than the obstacle speed at the zero distance; it therefore avoids collision. The desired speed is less than obstacle speed until s_s distance, which increases the longitudinal distance over time. It is greater than obstacle speed after s_s distance, which decreases longitudinal distance over time.

3.2 Smoothing Speed Profile

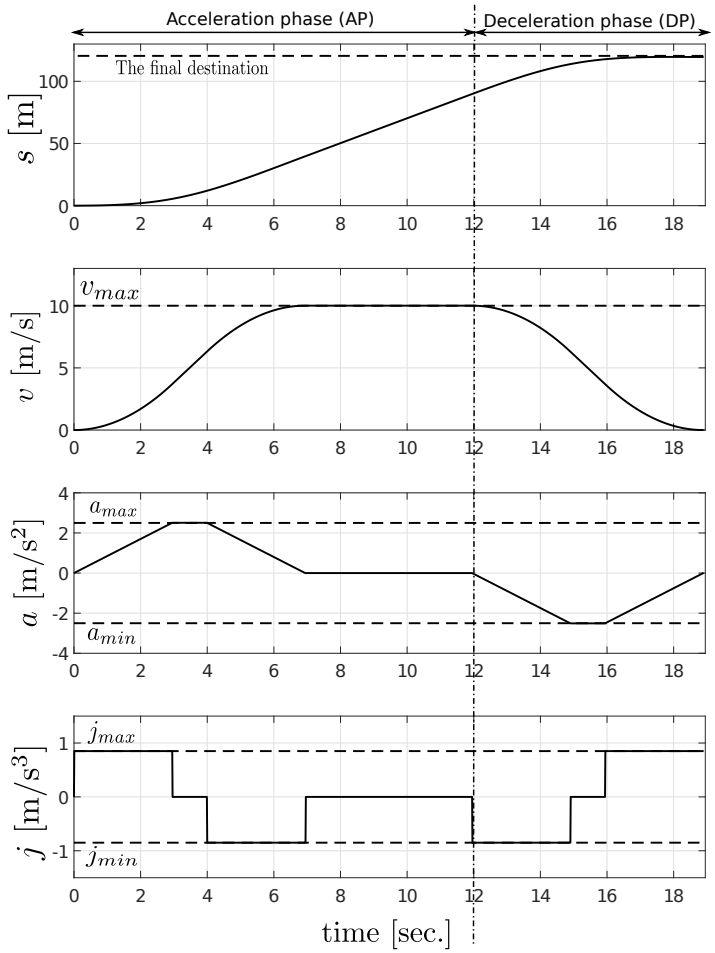
In the previous subsections, the desired path is created, and the maximum speed of each point is determined based on the road curvature, the traffic rules, and potential obstacles. In the next step, a smoothed trajectory should be provided to allow the low-level speed and steering controller to track it, and to provide a safe and comfortable drive. Although numerous methods have been proposed for smoothing the speed in the literature, most of them considered just the constant constraints for speed and acceleration without limiting jerk [69–71]. As the maximum jerk was not limited, the acceleration switches between zero and its upper and lower bounds, *i.e.*, it jumps from maximum positive to minimum negative and vice versa. The discontinuous acceleration or unlimited jerk not only causes passenger discomfort, but also it is not feasible for actuators to follow the trajectory. Therefore, in the proposed method in this section, the jerk constraint is also considered. Chapter 3 of [72] provides an approach for the computation of the **double S-trajectory**[1] to reach a specified speed with a constant boundary on jerk, acceleration, and speed.

As shown in Fig .3.8, in the double S-trajectory, the jerk is considered to be a piecewise constants of J_{max}, 0, and $-J_{max}$, meaning that the graph of acceleration over time contains two trapezoids (one above the time axis and another under the time axis). The graph of speed over time therefore contains two S-shapes. In the first S-shape, the car accelerates to reach maximum possible speed (**acceleration phase**). In the second one, it decelerates to reach the final position at the desired speed and acceleration (**deceleration phase**).

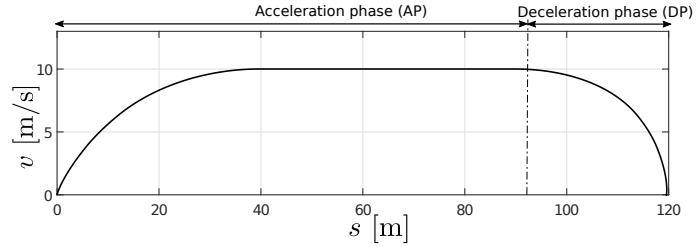
The method proposed below provides a numerical solution for computing a piecewise double S-trajectory to consider also non-constant constraints. Solving the problem numerically allows the speed limitation enforced by traffic rules or road curvature, be of any form. Furthermore, the acceleration constraints in any form, applied by the vehicle and its engine dynamics, can also be incorporated into the planning. Maximum possible acceleration and speed are imposed on the trajectory to make a time-optimal trajectory while ensuring that a proper controller can follow the trajectory. The jerk is also limited to provide passenger comfort.

The speed constraints, generated in the last section, enforce several acceleration/deceleration phases to the final speed profile. It is necessary to find the switching points where the sign of acceleration changes to separate the acceleration phases from deceleration phases and create a smoothed trajectory. Initial guesses for the switching points are the local extrema (maxima and minima) of the speed constraints (for abbreviation, we will call minimum point -min, and maximum point -max). Except for the first point, the switching points' initial accelerations

[1] More specifically it is called trajectory with double S velocity profile, and referred to also as bell trajectory or seven segments trajectory.



(a) Double S-trajectory profile in **time** domain (longitudinal displacement (s), speed(v), acceleration(a), and jerk(j) over time).



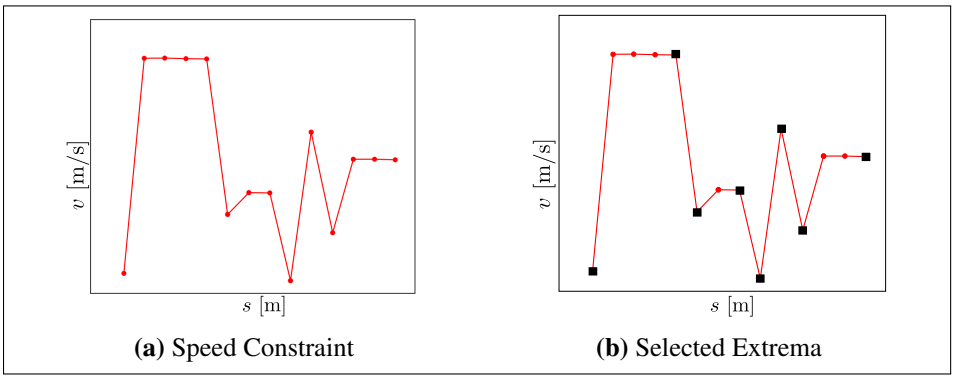
(b) Double S-trajectory speed profile in **space** domain (speed (v) over longitudinal displacement (s)).

Fig. 3.8: Double S-trajectory, a smoothed and time-optimal trajectory to reach a destination (*e.g.* 120 meters ahead) with speed, acceleration, and jerk constraints: the jerk is a piece-wise series of constant values $j_{max}, 0, -j_{max}$, and the acceleration value is limited by a_{min} and a_{max}. The graph of acceleration over time therefore contains two trapezoids (one above the time axis and another under the time axis), and the graph of speed over time contains two **S- shapes**. In the first S-shape, the car accelerates to reach maximum possible speed (**acceleration phase**). The zero acceleration is also considered as a part of acceleration phase in the following algorithms. Then in the second phase, the car decelerates to reach the final speed in the destination (**deceleration phase**).

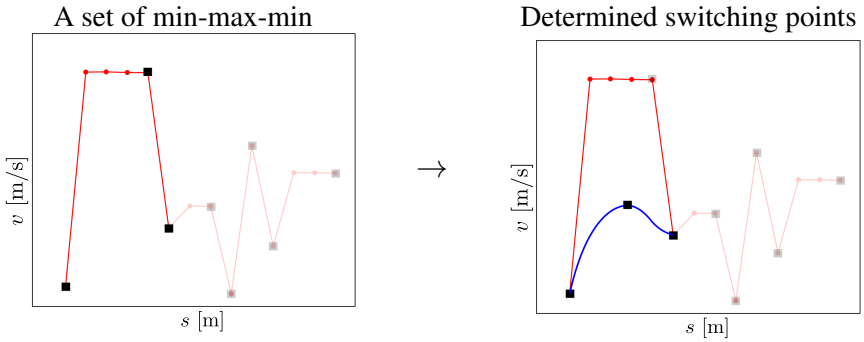
are zero. The first point acceleration is initialized from the last trajectory or the current car acceleration (in the lack of the previous valid trajectory). Then, the switching points (position, speed, and acceleration) are updated by checking three different cases. The acceleration and deceleration phases are established by determining the switching points. The following procedure is designed to specify the final switching points with their corresponding position, speed, and acceleration, as shown in Fig. 3.9, and Alg. 2).

- **Local extrema**: find maxima and minima of the speed constraints versus distance by Alg. 4) explained in 3.2.2.
- **(Min-Max-Min) set**: select the first or the next new set of (min-max-min) $[\mathbf{p_0}, \mathbf{p_1}, \mathbf{p_2}]$,
  - $\mathbf{p_1} = (s_1, v_1, a_1)$: the first/next local maximum point (the distance from the previous minimum/updated switching point to the local maximum, its speed, and acceleration),
  - $\mathbf{p_0} = (s_0, v_0, a_0)$: the minimum/updated switching point before $\mathbf{p_1}$,
  - $\mathbf{p_2} = (s_2, v_2, a_2)$: the local minimum point after $\mathbf{p_1}$.
- **First case (a double S-trajectory)**: find a valid local maximum between two local minima considering the constraints by Alg. 5 (explained in subsection 3.2.3).
- **Second case (only slow down)**: if it is impossible to find a valid local maximum in the set, and the speed of the first local minimum is bigger than the second one, $v_0 > v_2$, omit the local maximum. Omit or updated the first local minimum based on its previous points by Alg. 6 (explained in 3.2.4).
- **Third case (only speed up)**: if it is impossible to find a valid local maximum in the set, and the speed of the first local minimum is less than the second one, $v_0 < v_2$, omit the local maximum. Omit or updated the second local minimum based on its next points by Alg. 7 (explained in 3.2.5).
- Go to the next new set.

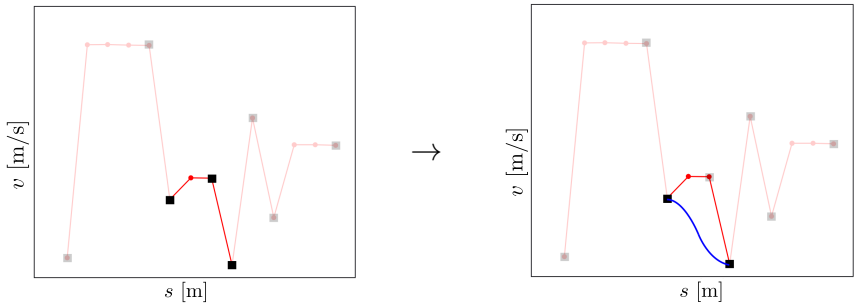
In the rest of the section, a numerical algorithm is proposed for the acceleration/deceleration phase (Alg. 3). Then the steps mentioned above are explained in detail to provide a trajectory consist of piecewise double S-trajectories.



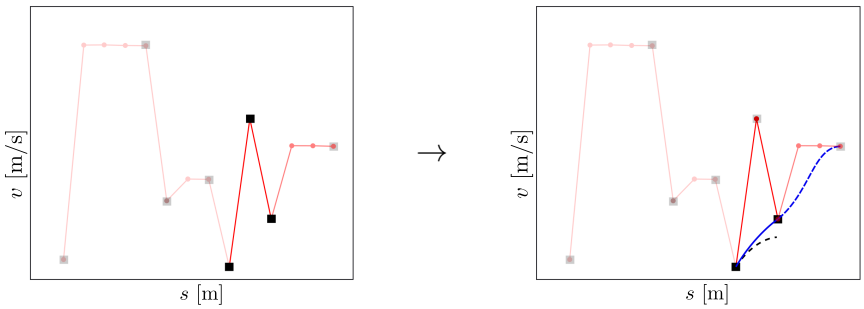
(a) Speed Constraint **(b)** Selected Extrema



(c) First case (a double S-trajectory)



(d) Second case (only slow down)



(e) Third Case (Only speed up)

Fig. 3.9: Smoothing speed profile: plotted here are the longitudinal speeds (v) over longitudinal displacement (s). The red graph illustrate the speed constraint. The black markers are the selected local extrema. When the speed of sequence points are the same, in the case of local maximum, the last point of them is selected, and in the case of local minimum, the first and last points of them are selected. After finding the extrema, a set of min-max-min points are selected and different cases (a double S-trajectory, only slow down, only speed up) are examined to find the final local switching points.

Algorithm 2: Check and Update Switching points

input : speed profile $\mathbf{v}_l$, initial switching points $\mathbf{P}^*$
output : updated switching points $\mathbf{P}^*$
constant : a_{max}, J_{max}
$i \leftarrow 1$ ▷ `index of` $\mathbf{P}^*$ `elements`
for `all local maxima in` $\mathbf{P}^*$ **do**
  $\mathbf{p_1} \leftarrow (s_1, v_1, a_1)$ next local maximum after $\mathbf{P}^*[i]$
  $\mathbf{p_0} \leftarrow (s_0, v_0, a_0)$ the local minimum before $\mathbf{p_1}$
  $\mathbf{p_2} \leftarrow (s_2, v_2, a_2)$ the local minimum after $\mathbf{p_1}$
  $i \leftarrow$ index of $\mathbf{p_1}$ in $\mathbf{P}^*$
  $\mathbf{P}^* \leftarrow$ **Alg. 5**$(\{\mathbf{p_0}, \mathbf{p_1}, \mathbf{p_2}\}, \mathbf{v}_l, \mathbf{P}^*)$ ▷`First case: a double S-trajectory`
  if *the case of double S-trajectory (Alg. 5) is invalid* **then**
    Delete $\mathbf{P}^*[i]$ ▷ `Remove` $\mathbf{p_1}$
    if $v_0 > v_2$ **then**
      $\mathbf{P}^* \leftarrow$ **Alg. 6**$(\{\mathbf{p_0}, \mathbf{p_2}\}, \mathbf{v}_l, \mathbf{P}^*)$ ▷ `Second case: only slow down`
    else
      $\mathbf{P}^* \leftarrow$ **Alg. 7**$(\{\mathbf{p_0}, \mathbf{p_2}\}, \mathbf{v}_l, \mathbf{P}^*)$ ▷ `Third case: only speed up`

3.2.1 Acceleration/Deceleration Phase

The maximum speed, acceleration, and jerk constraints must be taken into account to generate a feasible acceleration phase. As mentioned in the previous chapter, the acceleration limitation based on the speed for i-MiEV is:

$$p = p_{max} b_p \tag{3.7a}$$

$$a_l = min\left(a_{max}, \frac{p}{mv}\right) - c_a v^2 - c_f v \tag{3.7b}$$

$$j_l = (-2c_a v - c_f) a_l \tag{3.7c}$$

where b_p is the battery percentage (measured by voltage), p_{max} is the maximum power of the car with fully charged battery, p is the current car power, m is the vehicle mass, a_l is maximum acceleration considering the current power and speed, and c_a and c_f are the constant coefficients found empirically (using data fitting). Fig. 3.10 present an example of how the constraints are enforced in a speed up scenario. The solid blue curve shows the fastest acceleration path from v_0 to v_f when jerk value is limited by a constant. The black dash curve illustrates the acceleration constraints.

Using time step dt , *e.g.*, 0.01 s, the distance, speed, acceleration with constant jerk in each

step before hitting the constraints can be calculated through iterations of (3.8):

$$s_{k+1} = s_k + v_k\,dt + \frac{1}{2}a_k\,dt^2 + \frac{1}{6}j_k\,dt^3 \tag{3.8a}$$

$$v_{k+1} = v_k + a_k\,dt + \frac{1}{2}j_k\,dt^2 \tag{3.8b}$$

$$a_{k+1} = a_k + j_k\,dt \tag{3.8c}$$

where j_k, a_k, v_k are the jerk, acceleration, speed of the car, respectively, and s_k is the longitudinal displacement from the initial point, dt is the time step, and k is iterations' counter.

Furthermore, as the final acceleration is specified, if it is less than a_k, from a certain point (**P**) the acceleration must decrease smoothly with a constant negative jerk before reaching the final speed. From that certain point (**P**) the speed still increases by Δv (3.9) to reach the final acceleration a_f. In each iteration with a positive jerk, it is checked to ensure that $v_k + \Delta v$ is

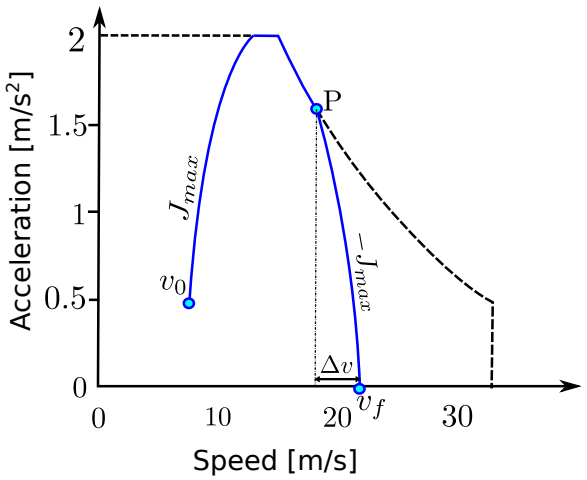


Fig. 3.10: Deal with the i-MiEV acceleration constraints: the proposed algorithm to smooth the speed profile can deal with any acceleration constraint function. The i-MiEV acceleration constraint - shown in the picture - illustrates this concept well. The acceleration constraint is a function of speed which is represented by the black dash curve. The solid blue curve is the desired speed profile from v_0 to v_f. The initial acceleration is 0.5 $\mathrm{m/s^2}$ and the final acceleration is zero. The acceleration increases with a maximum jerk J_{max} until reaches the acceleration constraint. Then it follows the constraint curve before reaching the point P. From point P, the jerk is considered to be negative $-J_{max}$. While the car speeds up smoothly by Δv, the acceleration reaches the desired final acceleration.

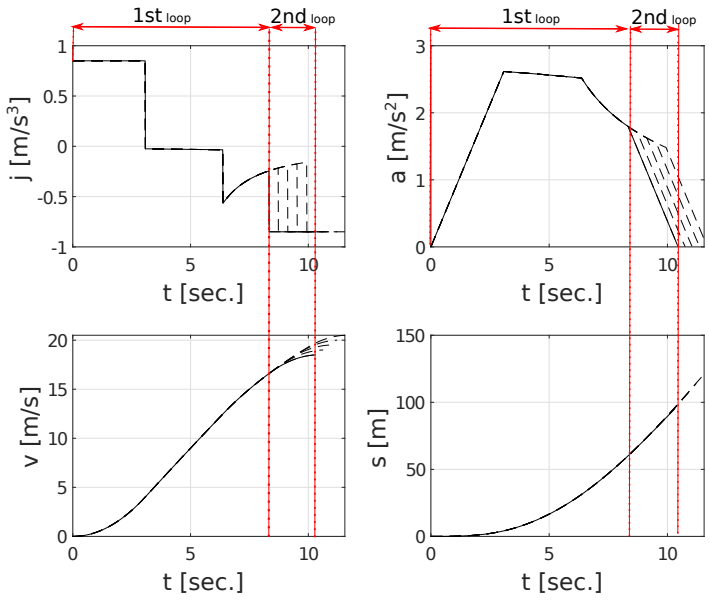


Fig. 3.11: The S-trajectory to speed up: plotted here are jerk (j), acceleration (a), speed (v), and longitudinal displacement (s) over time (t). The jerk value is limited by a constant value. The acceleration is limited by the i-MiEV car model. The required distance for reaching the maximum speed, *e.g.*, (20 m/s), is calculated. The calculated distance is bigger than desired distance, *e.g.*, (100 m), so the desired maximum speed is reduced by ε, *e.g.* (-0.5 m/s) until the required distance is less or equal to the desired distance (the black dash lines). The red dash lines specify the first and second loops of Alg. 3 called Acceleration Phase **AP** when the desired distance is also satisfied. In the second loop the jerk is -J_{max}. The final desired speed is 18 m/s.

less than v_f.

$$T = \frac{max(0, a_k - a_f)}{J} \tag{3.9a}$$

$$\Delta v = a_k T - \frac{1}{2} J_{max} T^2 \tag{3.9b}$$

The generalized algorithm for an **S**-trajectory to increase speed, enforcing numerical constraints is presented in Alg. 3 called Acceleration Phase (**AP**). The algorithm inputs are the initial and final accelerations and speeds. In this step, the final destination distance from the initial point s_f is infinity and thus does not impose any constraints. At the end of this section -after all the switching points are specified- s_f will be initialized by δ_s, and the algorithm will be used again to update the final trajectory speed points. In this scenario, the acceleration and speed of each sample point are calculated with a displacement constraint (δ_s) and they will be used as the initial acceleration and speed for the next sample point. We then calculate the required displacement to reach the final speed (s_a), where subindex

Algorithm 3: Acceleration Phase

input : initial acceleration (a_0), initial speed (v_0), final acceleration (a_f), final speed (v_f), and max distance $s_f = \infty$

output : the required distance s_a, the possible final acceleration (a_k), the possible final speed (v_k)

constant : a_{max},J_{max}, dt

▷ `Initialization`
$k \leftarrow 0, a_k \leftarrow a_0, j_k \leftarrow J_{max}$
$\Delta v \leftarrow (3.9)$
$a_l \leftarrow (3.7)$
while $v_k \leq (v_f - \Delta v) \ \wedge \ s_k < s_f$ **do**
  ▷ `Check Constraints`
  if $a_k \geq a_{max}$ **then**
    $j_k \leftarrow 0.0$
    $a_k \leftarrow a_{max}$
  if $a_k \geq a_l$ **then**
    $j_k \leftarrow a_l'$
    $a_k \leftarrow a_l$
  ▷ `Update States`
  $s_{k+1}, v_{k+1}, a_{k+1} \leftarrow (3.8)$
  ▷ `Update constraints`
  $\Delta v \leftarrow (3.9)$
  $a_l \leftarrow (3.7)$
  $k \leftarrow k+1$
$j \leftarrow -J_{max}$
▷ a_k `remains positive till end of the loop below, so,` $v_{k+1} > v_k$.
while $v_k \leq v_f \ \wedge \ s_k < s_f$ **do**
  ▷ `Update States with Negative Jerk`
  $s_{k+1}, v_{k+1}, a_{k+1} \leftarrow (3.8)$
  $k \leftarrow k+1$
$s_a \leftarrow s_k$

a stands for acceleration. We will use this algorithm as an acceleration phase in the next algorithms. The last acceleration, a_k, is therefore an output as well.
In the initialization step, the acceleration constraint a_l is initialized using the car model. In the case of using i-MiEV (3.7), the initial jerk is the maximum jerk J_{max}. Then, we calculate the speed change (Δv) required to reach the final acceleration from the current acceleration.

After the initialization step, we have two consecutive loops. In the **first loop**, the jerk is either positive or the derivative of the acceleration limitation. In other words, acceleration increases or follows the acceleration constraints. In each iteration, acceleration constraints (maximum passenger comfort acceleration a_{max}, and maximum acceleration which the car in

the current speed can provide a_l) are checked to limit the acceleration a_k and jerk j_k. Then, the next states (s_{k+1}, v_{k+1}, and a_{k+1}) are updated (3.8). This is also the point at which the speed change (Δv) in each iteration is calculated. The first loop iterates until $v_k < v_f - \Delta v$. After the point at which $v_k = v_f - \Delta v$, the jerk is considered negative and we enter the **second loop**. The second loop continues until the speed reaches the final speed while the acceleration is still positive/bigger than a_f. Finally, we obtain the required distance to reach the desired speed (s_a).

For the deceleration phase, we utilize a similar algorithm (**DecP**) in which the final speed is considered as an initial speed, and the opposites of the acceleration is used. Only passenger comfort constraint is enforced as it is less than the car brake limitation. The required distance to reach the final speed (s_d), which sub-index d stands for deceleration, is calculated.

The required distance should be less than the given distance in our initial planning. Therefore, the maximum desired speed (v_f) should be decreased if there is not enough distance to reach the given point. Fig. 3.11 shows an example to find the desired maximum speed within 100 meters. In the beginning, the maximum speed is 20.5 m/s which requires 121 m. Therefore, the maximum desired speed is decreased by 0.5 m/s until the required distance is less than 100 meters. The maximum desired speed is calculated 18.5 m/s.

3.2.2 The Local Extrema

As the first step of the smoothing algorithm, the local extrema of the speed constraints and the distance between them should be found by using Alg. 4 as an initial guess for the switching points. In Alg. 4, the inputs are the initial acceleration a_i, and the speed constraints $\mathbf{v}_l$. The distance between any two consecutive points of the array $\mathbf{v}_l$ is δ_s meters. The algorithm output consists of an array $\mathbf{P}^*$, which contains the information(s', v, a) of the first point, the local extrema, and the last point of the speed constraints. The distance of each local extrema from the previous one is denoted with s'. In the local extrema, the acceleration is zero or the sign of the acceleration is changed. The acceleration change is checked after calculating the acceleration by subtracting the array of v from its right-shifted array. The points at which the acceleration sign differs from the next point, or changes from zero to negative/positive, or changes from negative to zero ($a^+ \to a^-$, $a^- \to a^+/0$, $0 \to a^{+/-}$) are considered as the local extrema. In the next algorithm, the speed up phase includes the zero acceleration; therefore, changing the acceleration from positive to zero is not marked in the extrema list.

The black points in Fig.3.12 are the initial switching points $\mathbf{P}^*$, the red points are the speed constraints, and the blue line is the final smoothed trajectory. As we will see later, some of the switching points will be omitted or non-zero accelerations in some switching points will

be specified.

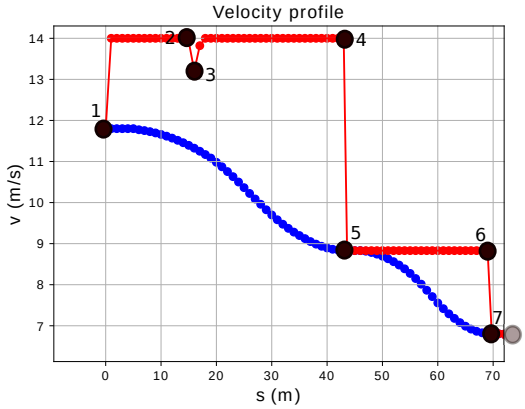


Fig. 3.12: The initial local extrema of the speed constraints in space domain: plotted here are the trajectory points speed (v) versus their longitudinal displacement (s). The red line is the speed constraints, the black points are the initial selected extrema, the blue line is the final smoothed speed profile. When the speed of sequence points are the same, in the case of local maximum, the last point of them is selected (like point number 2), and in the case of local minimum, the first and last points of them are selected (like point number 5 and 6).

Algorithm 4: Find Initial Switching Points (Local Extrema)

input : speed constraints $\mathbf{v}_l$, initial acceleration a_i, distance between two sequence points δ_s, whole path displacement S

output : first, last, and local extrema (initial switching points): $\mathbf{P}^*$

$\mathbf{a} \leftarrow \mathbf{v}_l -$shift to the right($\mathbf{v}_l$) ▷ `Differentiate` $\mathbf{v}_l$

$s' \leftarrow 0$ ▷ `distance between two consecutive local extrema`

$n \leftarrow 1$ ▷ `auxiliary variable`

Add $(s', \mathbf{v}_l[1], a_i)$ to the list $\mathbf{P}^*$ ▷ `first point of` $\mathbf{v}_l$

$i \leftarrow 1$ ▷ `loop counter`

▷ `Find the place where sign changes`

for $i \leftarrow 1$ ***until*** $length(\mathbf{a}) - 1$ **do**

  if $(sgn(\mathbf{a}[i]) \neq sgn(\mathbf{a}[i+1])) \vee (\mathbf{a}[i] = 0 \wedge \mathbf{a}[i+1] \neq 0) \vee (\mathbf{a}[i] < 0 \wedge \mathbf{a}[i+1] = 0)$ **then**

    $n \leftarrow n + s'$

    $s' \leftarrow (i+1)\delta_s - n$

    Add $(s', \mathbf{v}_l[1], a_i)$ to the list $\mathbf{P}^*$

$n \leftarrow n + s'$

Add $(S - n, \mathbf{v}_l[end], 0)$ to the list $\mathbf{P}^*$ ▷ `last point of` $\mathbf{v}_l$

3.2.3 First Case: A Double S-trajectory

Alg. 4 provides us the initial local extrema of the speed constraints. A set of three local extrema min-max-min is selected through each iteration of Alg. 2. In the first situation, a valid local maximum is found in the set as described below; consequently, a piece of a double S-trajectory can be specified between two local minima.

In Alg. 5, a min-max-min set $\{\mathbf{p_0}, \mathbf{p_1}, \mathbf{p_2}\}$ from the list of local extrema $\mathbf{P}^*$ is an input. Each $\mathbf{p_i}$ contains information of s_i, v_i, and a_i which are the distance from the previous local extremum, its speed, and its acceleration. The min-max-min set covers two phases: acceleration and deceleration phases.

Step 1 (the first grey box) of Alg. 5 focuses on only the acceleration phase. The goal is to find the maximum speed within the distance between first min and max s_1 while the initial speed v_0 and the initial acceleration a_0 are specified and the final acceleration is zero. First, the required distance s_a to reach v_1 is calculated using Alg. 3. In the loop, v_1 is reduced by ε rate until the required distance is equal or less than the desired distance s_1, or the maximum speed v_1 reaches the initial speed v_0 (therefore, it must not accelerate). Meanwhile, acceleration distance s_1 can be increased by δ_s if the desired speed v_1 is less than its next point speed $\mathbf{v}_l[r]$. In this way, the local maximum point is shifted to the right; therefore, the distance for deceleration s_2 is decreased. Notice the black dashed line in Fig. 3.13 which moves $\mathbf{p_1}$ to the

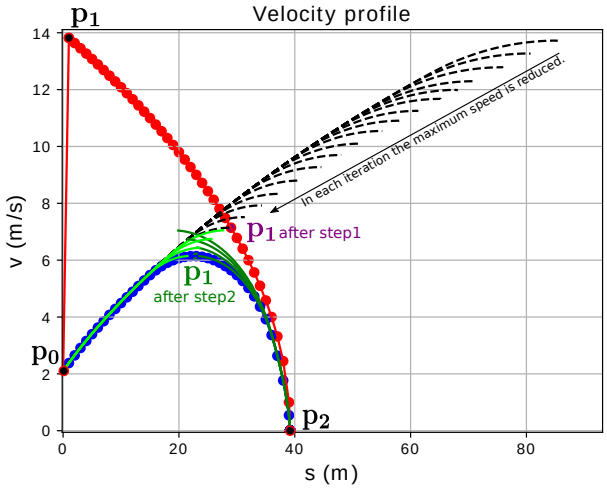


Fig. 3.13: A double S-trajectory: plotted here are the trajectory points' speeds (v) versus their longitudinal displacement (s). The red line is speed constraints, $\{\mathbf{p_0}, \mathbf{p_1}, \mathbf{p_2}\}$ on the red line are a min-max-min set, the blue line is the final smoothed speed profile. The black lines are the Alg. 5-step1 iteration results, force the valid local maximum be under the speed constraints curve (the red line). The green lines are the step2 iteration results. The position and speed of $\mathbf{p_1}$ changes through step1 and step2.

right along the speed constraint (red line) .

If v_1 is still bigger than v_2, in **step 2** of Alg. 5, the validity of the local minimum v_2 will be checked. The local maximum speed v_1 is decreased until it meets the required distance to accelerate and decelerate smoothly. The required distance s_a to accelerate from v_0 to v_1 and the required distance s_d to decelerate from v_1 to v_2 are obtained through Alg. 3 (AP and DecP).

Meanwhile, the first point of the set is replaced by its previous switching point p_{-1} if it has

---

Algorithm 5: Check and Update Switching points though a double S-trajectory

---

input : min-max-min set $\{\mathbf{p_0}(s_0, v_0, a_0), \mathbf{p_1}, \mathbf{p_2}\}$, speed profile $\mathbf{v}_l$, initial switching points $\mathbf{P}^*$, distance between two sequence points δ_s

output : updated switching points $\mathbf{P}^*$

$i \leftarrow$ index of $\mathbf{p_1}$ in $\mathbf{P}^*$ ▷`Index of the local maximum`

$s_{max} \leftarrow s_1 + s_2$ ▷`The distance between two minimums`

$n \leftarrow$ index related to one point after $\mathbf{p_1}$ in $\mathbf{v}_l$

$s_a \leftarrow \text{AP}(a_0, v_0, 0, v_1, \infty)$ ▷ `Alg.` 3: the required distance from (a_0, v_0) to (a_1, v_1)

while $s_a > s_1 \wedge v_1 > v_0$ **do**
- $v_1 \leftarrow v_1 - \varepsilon$ ▷`ε is the reduction amount in each step.`
- **if** $v_1 <= \mathbf{v}_l[n] \wedge s_2 > 0$ **then**
  - $v_1 \leftarrow \mathbf{v}_l[n]$
  - $s_1 \leftarrow s_1 + \delta_s$, $s_2 \leftarrow s_2 - \delta_s$.
  - $\mathbf{P}^*[i] \leftarrow (s_1, v_1, 0)$, $\mathbf{P}^*[i+1] \leftarrow (s_2, v_2, 0)$
  - $n \leftarrow n + 1$
- $s_a \leftarrow \text{AP}(a_0, v_0, 0, v_1, \infty)$ ▷`Alg. 3.`

$s_d \leftarrow \text{DecP}(0, v_2, 0, v_1, \infty)$ ▷ `Modified Alg.` 3: the required distance from (a_2, v_2) to (a_1, v_1)

while $s_d + s_a > s_{max} \wedge v_1 > v_0 \wedge v_1 > v_2$ **do**
- $v_1 \leftarrow max(v_2, v_1 - \varepsilon)$
- **if** $i > 1 \wedge a_0 > 0$ **then**
  - $\Delta v(a_0) \leftarrow (3.9)$
  - **if** $v_1 < v_0 + \Delta v$ **then**
    - $\mathbf{P}^*[i].s \leftarrow \mathbf{P}^*[i].s + \mathbf{P}^*[i-1].s$
    - $v_0, a_0 \leftarrow \mathbf{P}^*[i-2]$
    - $s_{max} \leftarrow s_{max} + \mathbf{P}^*[i-1].s$
    - Delete $\mathbf{P}^*[i-1]$. ▷ `Remove` $\mathbf{p_0}$
    - $i \leftarrow i - 1$
- $s_a \leftarrow \text{AP}(a_0, v_0, 0, v_1, \infty)$, $s_d \leftarrow \text{DecP}(0, v_2, 0, v_1, \infty)$

if $v_1 > v_2 \wedge s_d + s_a <= (s_1 + s_2)$ **then**
- $\mathbf{P}^*[i] \leftarrow (s_1 + s_2 - s_d, v_1, 0)$
- $\mathbf{P}^*[i+1] \leftarrow (s_d, v_2, 0)$

else
- This case is invalid.

---

less speed than the first point $v_{-1} < v_0$, and it is impossible to reach maximum speed v_1 from the first point v_0. In this situation the switching points are updated by omitting the first point and updating v_0; accordingly, increasing the desired distance of the acceleration phase.
If a proper v_1 (v_1 is bigger than v_2 and v_0) is found, a double S-trajectory can be specified for this set. In this case, the index is increased and the next min-max-min set is chosen.
As an example, Fig 3.13 illustrates the step1 and step2. The speed constraint $\mathbf{v}_l$ is illustrated by the red line. A min-max-min set $\{\mathbf{p_0}, \mathbf{p_1}, \mathbf{p_2}\}$ are selected. The first point is (2 m/s^2, 2 m/s, 0 m), the local maximum is (0 m/s^2, 14 m/s, 1 m), and the end point is (0 m/s^2, 0 m/s, 40 m). In step 1, while the local maximum speed v_1 is decreasing, its next point speed constraint $\mathbf{v}_l[r]$ is checked. If $\mathbf{v}_l[r]$ is less than v_1, then the distance for acceleration s_1 is increased. The step 1 iteration stops when the point (s_1,v_1) is under the speed constraints (the red line). In this case, the first step iteration stops when $s_1 = 29$ and $v_1 = 7.2$. As v_1 is still bigger than both previous and next local extrema speeds (v_0 and v_2), we next go through step 2. The speed v_1 decreases until it is reachable from both local minima. The light and dark green lines show the results of Alg. 3 for acceleration and deceleration phases while v_1 is decreasing, s_1 is decreasing and s_2 is increasing. v_1 reduction speed continues until both dark and light green lines curve sufficiently to avoid crossing (sum of $s_d + s_a$ be less than provided distance s_{max}, which is in this case 40 m). If a valid local maximum is not found, the local maximum will be omitted from the set. Two other cases will also be checked to either update or omit the first and second local minima in Alg. 6 and Alg. 7, respectively.

3.2.4 Second Case: Only Slow Down

If a valid local maximum is not found between a set of min-max-min, and the speed of the first minimum is bigger than the second minimum, then the first local minimum may not be valid as a local minimum anymore. After omitting the local maximum, in Alg. 6 the switching point before the set is also checked. If its speed v_{-1} is less than the first min ($v_{-1} < v_0$), then the first local min is selected as a new guess for the local max. By decreasing the index, the next loop will be executed for the new set. Otherwise, the first point is neither a minimum nor maximum point anymore. However, the smooth speed profile should be kept under the speed constraints. The required distance to reach v_0 from v_2 with the maximum acceleration at $\mathbf{p_0}$ is denoted with s'[1]. If s' is bigger than the desired distance, it means that the speed at point $\mathbf{p_0}$ will be less than v_0; the first local minimum $\mathbf{p_0}$ will therefore be omitted.

[1]While Alg. 3 is utilized for deceleration phase, we flip the points vertically. Then the initial point is the minimum point and the final point has bigger speed. Therefore, the initial acceleration is zero, and the final acceleration is the negative acceleration of the last point. This calculates the required distance from a smaller speed to a bigger one.

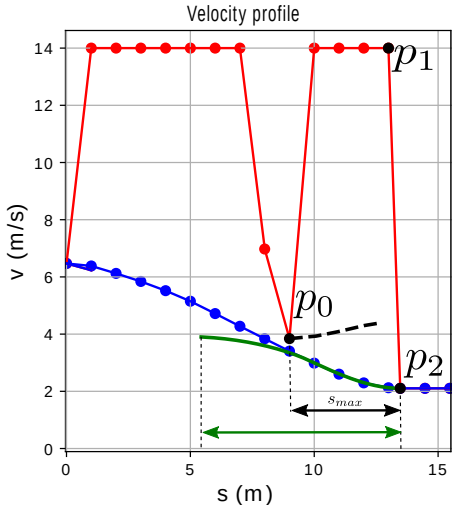


Fig. 3.14: Only slow down (Alg.6): plotted here are the trajectory points' speeds (v) versus their longitudinal displacement (s). The red line represents speed constraints, the blue line is the final smoothed speed profile, and the black points $\{\mathbf{p_0}, \mathbf{p_1}, \mathbf{p_2}\}$ are the second set of (min-max-min) of the selected local extrema. The black dash line illustrates the maximum speed that we can reach with the final zero acceleration from $\mathbf{p_0}$ in the distance between $\mathbf{p_0}$ and $\mathbf{p_1}$. The green line illustrates the required distance to decelerate from the speed of $\mathbf{p_0}$ to $\mathbf{p_2}$ with initial and final zero acceleration is bigger than their distance s_{max}. The distance for reaching v_2 from v_0 with the maximum negative acceleration value is also bigger than s_{max}. Therefore, the first local minimum and maximum $\mathbf{p_0}$, and $\mathbf{p_1}$ of the set are not reachable. They are omitted from the switching points list to have a smooth deceleration from the initial point to $\mathbf{p_2}$.

The index is decreased by two, and this loop continues until it reaches the new min-max-min set.

On the other hand, if s' is less than s_{max}, a proper acceleration a_0 should be specified by which the speed profile reaches v_2 from v_0 in the distance s_{max}. Therefore, a_0 is reduced by ε until the required distance from v_0 to v_2 with a_0 acceleration becomes sufficient. Then, the acceleration of the first minimum is updated to a negative proper value. However, before going to the next min-max-min set, we must also check the previous switching point condition if it has a bigger acceleration. For example, a positive acceleration must continuously decrease to reach zero, meanwhile the speed still increases. We need to calculate this speed change. Therefore, the required change in speed Δv to have a continuous acceleration with constant jerk is calculated utilizing (3.8). If the speed difference between v_{-1} and v_0 is less than Δv, the first local minimum $\mathbf{p_0}$ is omitted. Braking from v_{-1} to v_2 considering the limited jerk automatically provides a less speed than v_0 in the point $\mathbf{p_0}$; so, $\mathbf{p_0}$ is omitted. By reducing the index by two, this loop continues till it reaches a new set min-max-min. On the other hand, if the speed difference between v_{-1} and v_0 is sufficient, the index is increased to reach

Algorithm 6: Update Switching Points through Reducing Speed

input : { $\mathbf{p_0}(s_0, v_0, a_0)$, $\mathbf{p_2}(s_2, v_2, a_2)$}, initial switching points $\mathbf{P}^*$
output : updated switching points $\mathbf{P}^*$, the index of next local maximum i
constant : a_{max},J_{max}

$s_{max} \leftarrow s_1 + s_2$
$i \leftarrow$ the index of $\mathbf{p_0}$ in $\mathbf{P}^*$
$\mathbf{p_{-1}} \leftarrow \mathbf{P}^*[max(1, i-1)]$ ▷ `the switching point before` $\mathbf{p_0}$ `in` $\mathbf{P}^*$
if $v_{-1} \leq v_0$ **then**
  $i \leftarrow i - 1$
else
  while $i > 1 \ \wedge \ v_{-1} > v_0$ **do**
    $a' \leftarrow \min(\sqrt{2J_{max}(v_0 - v_2)}, a_{max})$
    $s' \leftarrow \mathrm{DecP}(0, v_2, -a', v_0)$ ▷ `Modified Alg. 3`
    if $s' > s_{max}$ **then**
      $\mathbf{P}^*[i] \leftarrow (s_{max}, v_2, 0)$,Delete $\mathbf{P}^*[i-1]$ ▷ `Remove` $\mathbf{p_0}$
    else
      s" $\leftarrow \mathrm{DecP}(0, v_2, -a_0, v_0)$
      while s" $> s_{max} \ \wedge \ a_0 > -a_{max}$ **do**
        $a_0 \leftarrow a_0 - \varepsilon$
        s" $\leftarrow \mathrm{DecP}(0, v_2, -a_0, v_0)$
      $\mathbf{P}^*[i-1].a \leftarrow a_0$
      $\Delta v \leftarrow (3.9)(a_0)$
      if $a_{-1} > a_0$ *and* $\Delta v > v_{-1} - v_0$ **then**
        $\mathbf{P}^*[i] \leftarrow (s_{max} + s_0, v_2, 0)$,Delete $\mathbf{P}^*[i-1]$ ▷ `Remove` $\mathbf{p_0}$
      else
        Break the loop.
    $i \leftarrow max(1, i-2)$,
    $\mathbf{p_0} \leftarrow \mathbf{P}^*[i]$, $\mathbf{p_{-1}} \leftarrow \mathbf{P}^*[max(1, i-1)]$
    $s_{max} \leftarrow s_{max} + \mathbf{P}^*[i-1].s$

the next new set min-max-min.

An example of this case is shown in Fig. 3.14. In the second set, a proper local maximum cannot be found (v_1 reaches v_0). In this situation, v_2 is less than v_0, and the local maximum $\mathbf{p_1}$ is omitted. The distance for reaching v_2 from v_0 with the maximum negative acceleration value is calculated with the DecP algorithm, and it is shown that it is bigger than s_{max}; therefore, the first local minimum is also omitted. The initial speed is reduced to v_2.

3.2.5 Third Case: Only Speed Up

In the last case, Alg. 7, the maximum speed v_1 is invalid and the first local minimum is less than second minimum ($v_0 < v_2$). Therefore, the second local minimum is not valid anymore as the acceleration is positive before and after $\mathbf{p_2}$. The speed constraints increases after v_2; so, a positive acceleration in this point is calculated to force the speed profile to be under v_2. As in the previous case, first the maximum speed v_k which is possible to reach in s_{max} from v_0 with the maximum acceleration a_2 is calculated. If v_k is less than v_2 then $\mathbf{p_2}$ is also omitted, otherwise a_2 is decreased till a proper acceleration which makes the acceleration distance s_a to be equal to s_{max}. Then the index is increased. Before going to the next new set, as was done in the previous case, Δv is calculated to find out if it is enough to reach smoothly to the next local maximum. If it isn't enough, the second local minimum $\mathbf{p_2}$ is omitted, the index is decreased, and the algorithm goes to the next new min-max-min set. This case is illustrated in Fig. 3.15 in which it can not reach v_2 with zero acceleration with in distance s_{max}, and if a_2 is a maximum value, the related speed v_k will exceed v_2. Therefore, a proper acceleration is defined for $\mathbf{p_2}$.

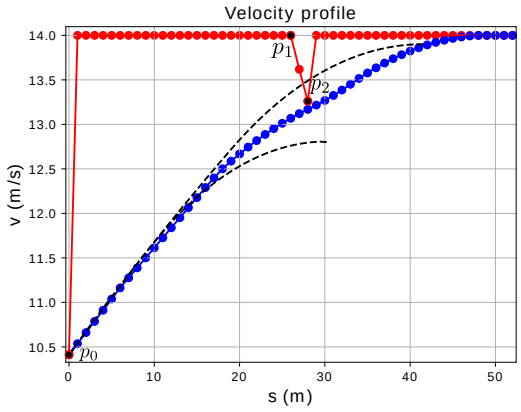


Fig. 3.15: Only Speed Up (Alg.7): plotted here are the trajectory points' speeds (v) versus their longitudinal displacement (s). The red line is speed constraints, the blue line is the final smoothed speed profile, and the black points $\{\mathbf{p_0}, \mathbf{p_1}, \mathbf{p_2}\}$ are a set of (min-max-min) of the selected local extrema. The upper black dash line illustrates that the required distance to reach the speed of $\mathbf{p_1}$ with the final zero acceleration is bigger than the provided distance. The first local maximum $\mathbf{p_1}$ therefore is not reachable and it is omitted from the switching points. The lower dash line illustrates that the maximum speed with final zero acceleration in the distance between $\mathbf{p_0}$ and $\mathbf{p_2}$ is less than the speed of $\mathbf{p_2}$. The acceleration at $\mathbf{p_2}$ therefore is increased that we can reach its speed.

Algorithm 7: Update Switching Points through Increasing Speed

input : { $\mathbf{p_0}(s_0, v_0, a_0)$, $\mathbf{p_2}(s_2, v_2, a_2)$}, speed profile $\mathbf{v}_l$, initial switching points $\mathbf{P}^*$
output : updated switching points $\mathbf{P}^*$
constant : a_{max},J_{max}

$s_{max} \leftarrow s_1 + s_2$
$i \leftarrow$ the index of $\mathbf{p_2}$ in $\mathbf{P}^*$
if $v_0 \leq v_2$ **then**
  $a_2 \leftarrow min(a_{max}, f_{i-MiEV}(v_2))$ ▷(3.7)
  $s_a \leftarrow -\infty, s' \leftarrow 0$
  $s_a, v_k, a_k \leftarrow \text{AP}(a_0, v_0, a_2, v_2, s_{max}, \infty)$. ▷ `Alg. 3`
  while $(s_a < s_{max} \vee v_k > v_2) \wedge a_2 > 0$ **do**
    $a_2 \leftarrow a_2 - \varepsilon$
    $s_a, v_k, a_k \leftarrow \text{AP}(a_0, v_0, a_2, v_2, \infty)$ ▷ `Alg. 3`
  $\mathbf{P}^*[i] \leftarrow (s_{max}, v_k, a_k)$
  ▷ `check the point after` $\mathbf{p_2}$
  if $i < length(\mathbf{P}^*) \wedge a_k > 0$ **then**
    $\Delta v \leftarrow (3.9)(a_k)$
    if $\mathbf{P}^*[i+1].v < v_k + \Delta v$ **then**
      $\mathbf{P}^*[i+1].s \leftarrow \mathbf{P}^*[i+1].s + \mathbf{P}^*[i].s$
      Delete $\mathbf{P}^*[i]$. ▷ `Remove` $\mathbf{p_2}$

3.2.6 Update Trajectory Point Speeds

Alg. 2 continues until all the initial local extrema are updated to a valid ones with respect to the jerk and acceleration constraints. Then Alg. 3 will be used to define the speed of the points between local extrema by limiting s_f with the distance between two points δ_s. Alg. 8 clarifies the updating trajectory point speed procedure.

Algorithm 8: Update Trajectory Speeds Values

input : speed profile $\mathbf{v}_l$, final switching points $\mathbf{P}^*$,distance between two sequence points δ_s
output : updated speeds $\mathbf{v}_l$

$m \leftarrow \text{length}(\mathbf{P}^*)$
for $i \leftarrow 1$ ***until*** $m-1$ **do**
  if $\mathbf{P}^*[i] > \mathbf{P}^*[i+1]$ **then**
    $a_0 \leftarrow \mathbf{P}^*[i].a$
    $n \leftarrow$ index of $\mathbf{P}^*[i]$ in $\mathbf{v}_l$
    for `all` $\mathbf{v}_l$ `between` $\mathbf{P}^*[i]$ `and` $\mathbf{P}^*[i+1]$ **do**
      $a_0, \mathbf{v}_l[n+1] \leftarrow \text{AP}(a_0, \mathbf{v}_l[n], \mathbf{P}^*[i+1].a, \mathbf{P}^*[i+1].v, \delta_s)$ ▷ `Alg. 3.`
      $n \leftarrow n+1$
  else
    $a_0 \leftarrow \mathbf{P}^*[i+1].a$
    $n \leftarrow$ index of $\mathbf{P}^*[i+1]$ in $\mathbf{v}_l$
    for `all` $\mathbf{v}_l$ `between` $\mathbf{P}^*[i]$ `and` $\mathbf{P}^*[i+1]$ **do**
      $a_0, \mathbf{v}_l[n-1] \leftarrow \text{DecP}(a_0, \mathbf{v}_l[n], \mathbf{P}^*[i].a, \mathbf{P}^*[i].v, \delta_s)$ ▷ `Modified Alg. 3.`
      $n \leftarrow n-1$

3.3 Re-planning from Look-ahead Points and Interpolation

In this section, an important practical issues is discussed. The importance of using the previous trajectory to initialize the next trajectory. As explained in the last chapter, in the pure pursuit controller, a steering look-ahead point is required to define the desired steering angle. For example, a steering look-ahead point is 3 meters ahead of the car along the trajectory. While the trajectory planner is executed at a rate of 10 Hz, the position of the steering look-ahead point should change smoothly. Otherwise, the desired set point for the controller will jump, and it will cause chattering/jittering for the steering angle. Thus, we keep/freeze the last trajectory **positions** until the steering look-ahead point by which the new trajectory is initialized. Furthermore, we keep the last trajectory **speeds** until a look-ahead point in time domain (*e.g.*, 0.6s ahead) to have a smooth desired set points for the longitudinal speed controller.

By using the steps of the previous, we obtained a trajectory which contains n points with δ_s distance, continuous acceleration, and piece-wise constant jerk. The initial part of the trajectory should be frozen, and replanning starts from the last frozen point instead of the current car position. The frozen part is in time domain while our sample points are in space domain. As the longitudinal speed controller is responsible for keeping the car on a trajectory,

it has a look-ahead point in time domain; the desired speed is the speed of the point where is T time ahead of the car. Consequently, we need to re-sample (interpolate) the trajectory to define the frozen part in the time domain and the controller look-ahead point. There are different methods to interpolate the multi-point trajectories[72] interpolation by polynomial function, orthogonal polynomials, trigonometric polynomials, and cubic splines. The interpolation by polynomial function for trajectories with large values of n (number of points) may produce non-negligible numerical errors.[1] The local trajectory usually contains around 100 points (*e.g.*., 100 meters) as the curvature, obstacle and traffic lights, jerk and acceleration constraints must be included in the calculation; therefore, the polynomial function method is not suitable for the interpolation. In the orthogonal polynomials approach, the least squares approach is utilized to calculate the polynomials coefficients; this method is relatively inefficient from the computational point of view. The trigonometric polynomials approach is usually convenient for the periodic motion with even number of points; the advantage of this method is the all derivatives are continuous. The cubic splines usually provides smaller acceleration and jerk compared to the trigonometric polynomials approach. In the smoothness step, we choose the piece-wise constant jerk to define the point's speed. Therefore, this thesis uses the cubic spline interpolation. In (3.10) $s(t)$ is the cubic spline.

$$s(t) = f_k(t), t \in [t_k, t_{k+1}], k = 0, ..., n-1 \tag{3.10a}$$

$$f_k(t) = a_{k0} + a_{k1}(t-t_k) + a_{k2}(t-t_k)^2 + a_{k3}(t-t_k)^3 \tag{3.10b}$$

$$a_{k0} = s_k \tag{3.10c}$$

$$a_{k1} = v_k \tag{3.10d}$$

$$a_{k2} = \frac{1}{T_k}\left(\frac{3(q_{k+1}-q_k)}{T_k} - 2v_k - v_{k+1}\right) \tag{3.10e}$$

$$a_{k3} = \frac{1}{(T_k)^2}\left(\frac{2(q_k-q_{k+1})}{T_k} + v_k + v_{k+1}\right) \tag{3.10f}$$

where (s_k, v_k, t_k) are position, speed and time of each sample point in the trajectory. Considering $T_k = t_{k+1} - t_k$, the coefficients a_k of the cubic polynomial ($f_k(t)$) between q_k and q_{k+1} are specified.

[1] For instance, for a trajectory with 20 points and using IEEE standard double precision, the error of polynomial coefficients can reach 25 meters [72].

3.4 Simulation Results

Several simulations are performed to validate the proposed algorithm. The simulator environment contains some Berlin streets, MIG kinematic and dynamic, car-like obstacles programmed in ROS framework and visualized in Rviz. The reactive trajectory planner ran at 10 Hz. Using the controller output, the simulated car states updates at 100 Hz. Fig. 3.16 shows a part of Thielallee street near the Freie Universtiät Berlin, on which we have often tested our autonomous car. Thielallee street has an intersection with traffic lights, two U-turns, two bus stations. Therefore, it is the right test area as several scenarios can happen in it.

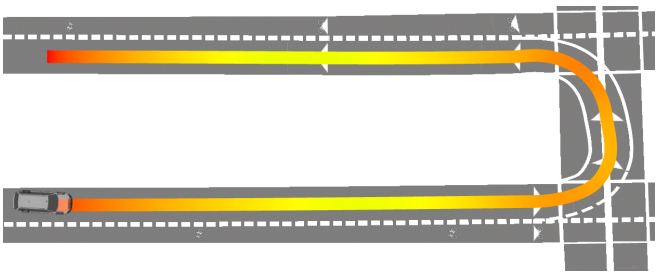


Fig. 3.16: Thielallee street: the first U-turn.

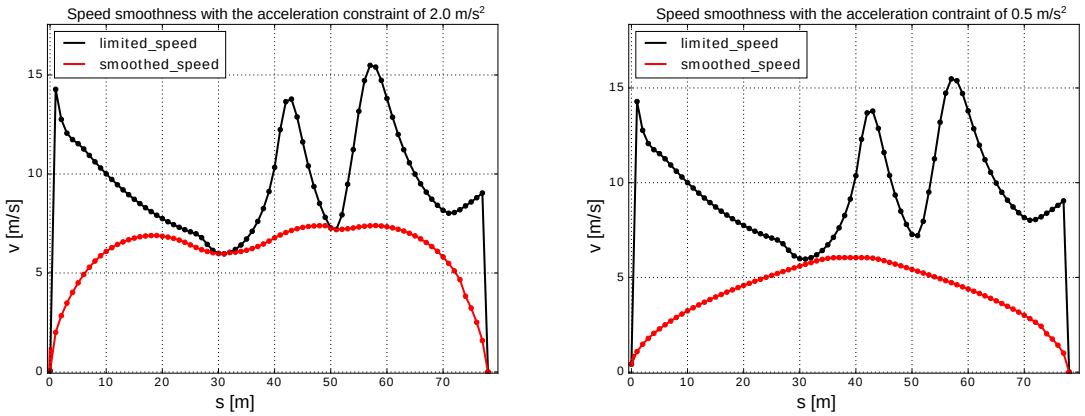


Fig. 3.17: Speed smoothness: plotted here are the trajectory points speed (v) versus their longitudinal displacement (s). The black curve is the limited speed at each point based on the road curvature. The red curves are the smoothed speeds when the maximum acceleration is $2\,\mathrm{m/s^2}$ (the left picture) and when the maximum acceleration is $0.5\,\mathrm{m/s^2}$ (the right picture).

Fig. 3.17 shows the (s-v) phase plane of a U-turn with a list of speeds with 1 m distance from each other. The black curve shows the initial speeds calculated based on road constraints, and red curves are the smoothed speeds with different passenger comfort accelerations ($2\,\mathrm{m/s^2}$ (left) and $0.5\,\mathrm{m/s^2}$ (right)). As we can see, the left trajectory contains three piecewise double S-trajectories, while the left one with lower acceleration constraint has only a double S-

trajectory. In Fig. 3.18 the two trajectories with and without bounded jerk are compared for a

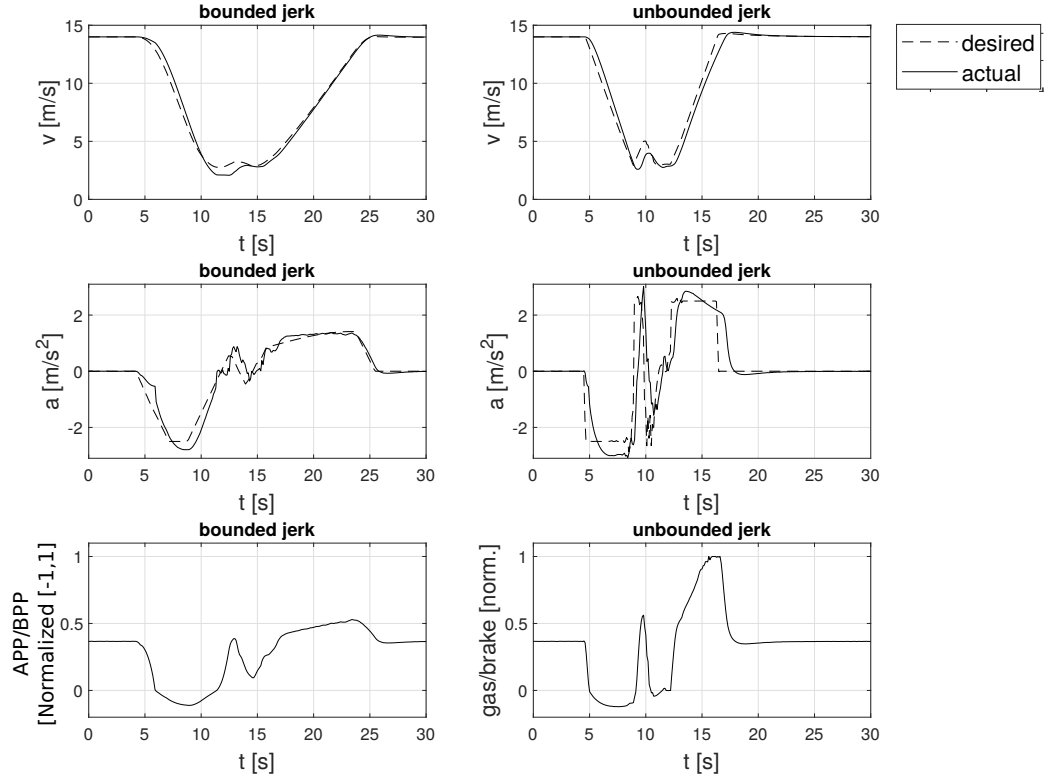


Fig. 3.18: Comparing two trajectories with and without bounded jerk: plotted here are the vehicle velocity (v), acceleration (a), accelerator pedal position (APP) normalized between [0,1] and brake pedal position (BPP) normalized between [-1,0] over time. The dashed lines are the desired control output and the solid lines are the simulation actuator output. The acceleration graph over time is continuous with the bounded jerk. As a result, the actuator output is smoother with lower values compared to the unbounded jerk graph.

U-turn. Speed, acceleration, actuators' output are plotted in different rows. As we can see in second row the acceleration of bounded jerk is continuous; therefore, the actuators output changes smoothly with lower values. In this case, the actuators can follow the provided trajectory while providing passenger comfort.

3.5 Experimental Results

The proposed method was tested on the same street (Thieallee) with 1 km length, two U-turns, an intersection in the middle. We have permission to drive autonomously with MIG in Berlin Urban area. The two trajectories are compared with each other. In the first trajectory, the acceleration is switches between constant values (zero, a positive value, and a negative

value). In the other trajectory, the acceleration is bounded by MIG model and by constant lower and upper bounds, allowing it to change with the limited jerk value. Passenger comfort is evaluated by average jerk ($\bar{\mathbf{j}}$) and maximum jerk (max($\mathbf{j}$)). In a dynamic environment, changing the acceleration (jerk) is inevitable. As jerk increases, the passenger comfort decreases. People's comfort thresholds vary extensively. While we are driving with our autonomous car, a human inside the car is responsible for emergency situations; therefore, our test drive cannot go into parameters which would cause him/her high discomfort. Our experiments show that instant jerk under 3.5 $\mathrm{m/s^3}$ still satisfies the passenger comfort and it will be lost above this threshold. The proposed trajectories approaches are evaluated in three test cases in order to assess the functionality of the method in different circumstances separately. The test scenarios are the followings:

- Speed up,
- Slow down for an U-turn,
- Interaction with obstacles.

In all cases, the maximum speed of the car is 13 $\mathrm{m/s}$. In the following, each of these cases and their results is described.

Speed Up

Fig. 3.19 shows two trajectories (with and without a jerk constraint) in the speed up case when the car accelerated from zero to maximum speed (13 $\mathrm{m/s}$) .

The tests' results are depicted in Table 3.1. In the first test, we used Alg.3 to accelerate from 0 $\mathrm{m/s}$ to 13 $\mathrm{m/s}$ in 19 s. As shown in Fig. 3.20, MIG model Eq. (2.19) and a constant upper bound (0.7 $\mathrm{m/s^2}$) limit the acceleration in the trajectory while its jerk value constraint is 0.85 $\mathrm{m/s^3}$. The trajectory following task was performed sufficiently accurate with the average error ($\bar{\mathbf{e}}$) of 0.13 $\mathrm{m/s}$ and a maximum error of 0.5 $\mathrm{m/s}$ from the desired speed profile. Passenger comfort was provided with the average jerk ($\bar{\mathbf{j}}$) of 0.39 $\mathrm{m/s^3}$ and maximum jerk of 1.95 $\mathrm{m/s^3}$.

In the second test, the trajectory acceleration upper bound(a_{max}) is 0.7 $\mathrm{m/s^2}$ without taking MIG model and the jerk constraint into account. The car reaches the desired speed 13 $\mathrm{m/s}$ in 19 s. While the average error and jerk are similar to those of the limited jerk method, the maximum speed error and consequently the maximum jerk increase slightly to 0.79 $\mathrm{m/s}$ and 3.15 $\mathrm{m/s^3}$.

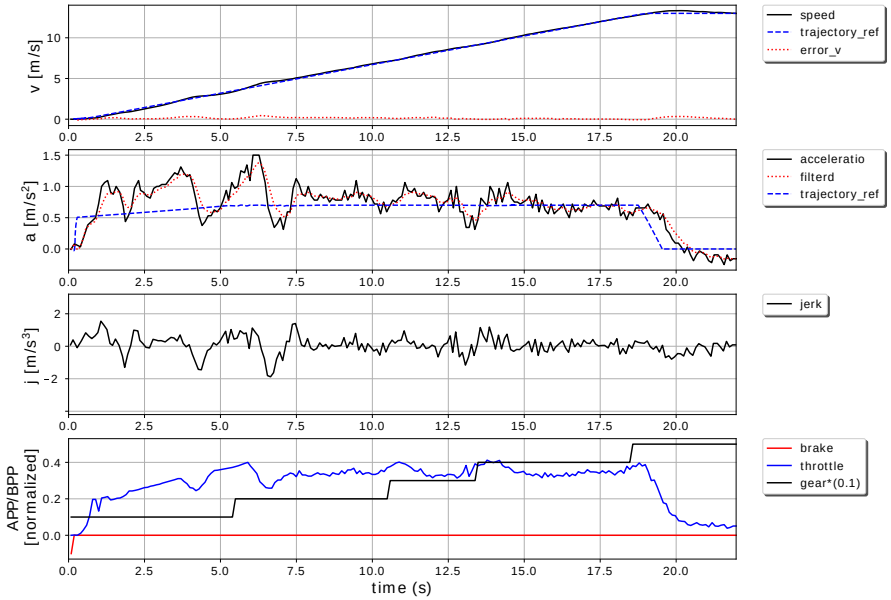
By increasing the acceleration upper bound above 0.7 $\mathrm{m/s^2}$- without the jerk constraint and MIG model- the speed errors increase and passenger comfort cannot be satisfied.

As shown in Fig 3.20, without the car model constraints which limits the acceleration at low speeds, even if a_{max} increases slightly to 0.8 m/s^2, the maximum speed errors increases to 1.1 m/s and the maximum jerk jumps to 4.79 m/s^3. On the other hand, when the MIG model limits the acceleration at low speed, a_{max} can increase to 1.5 m/s^2 and the car can reach the maximum speed in 16 s (16% time improvement). The passenger comfort was also provided with the maximum jerk of 2.25 m/s^3 and average jerk of 0.53 m/s^3.

Table 3.1: Compare results of four parameters groups for the speed up cases: each row illustrates 1) if the MIG model was used to constrain the acceleration or not, 2) the maximum acceleration constraint, 3) the maximum jerk constraint. The maximum error between the vehicle speed and the trajectory reference (max(**e**)), its mean absolute ($\bar{\mathbf{e}}$), the vehicle maximum jerk (max(**j**)), and its mean absolute ($\bar{\mathbf{j}}$) are compared.

	Trajectory Parameters			Results			
	Model	$j_{max}[m/s^3]$	$a_{max}[m/s^2]$	max(**e**)[m/s]	$\bar{\mathbf{e}}$[m/s]	max(**j**)[m/s^3]	$\bar{\mathbf{j}}$[m/s^3]
1	✓	0.85	0.7	0.5	0.13	1.95	0.39
2	×	∞	0.7	0.79	0.18	3.15	0.43
3	✓	0.6	1.5	0.58	0.19	2.25	0.53
4	×	∞	0.8	1.10	0.42	4.79	0.55

With jerk constraint $a_{max} = 0.7\ \mathrm{m/s^2}$



Without jerk constraint $a_{max} = 0.7\ \mathrm{m/s^2}$

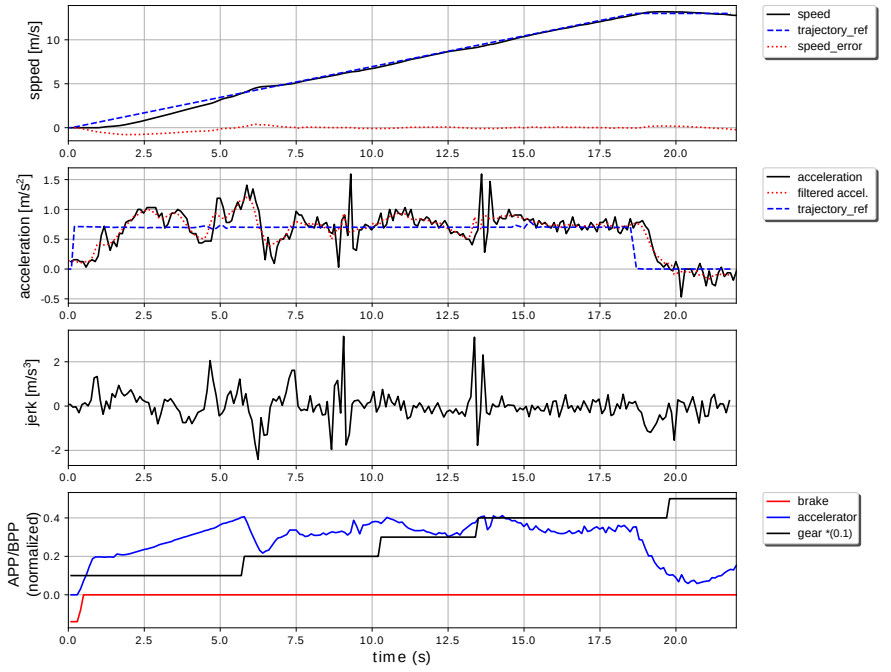
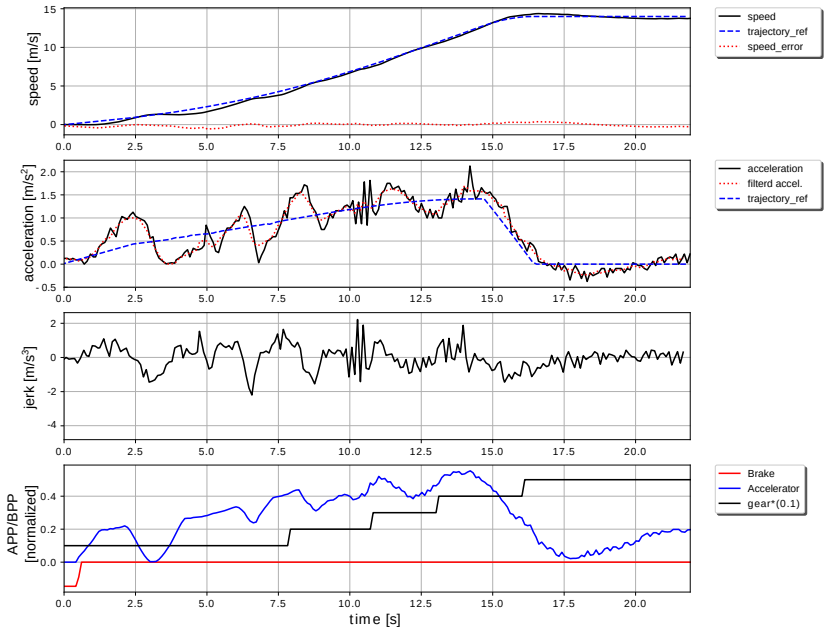


Fig. 3.19: Speed up tests for two trajectories with and without a jerk constraint while the acceleration was limited by 0.7 $\mathrm{m/s^2}$: this plots the vehicle speed, acceleration, accelerator pedal position (APP) normalized between [0,1] and brake pedal position (BPP) normalized between [-1,0] over time. By limiting the jerk with a constant value, the maximum speed error and consequently the maximum jerk are decreased from 0.79 $\mathrm{m/s}$ to 0.5 $\mathrm{m/s}$ and from 3.15 $\mathrm{m/s^3}$ to 1.93 $\mathrm{m/s^3}$. Nevertheless, the passenger comfort in the both cases are provided. The test results are compared in the first two rows of Table 3.1.

With jerk constraint $a_{max} = 1.5\ \mathrm{m/s^2}$



Without jerk constraint $a_{max} = 0.8\ \mathrm{m/s^2}$

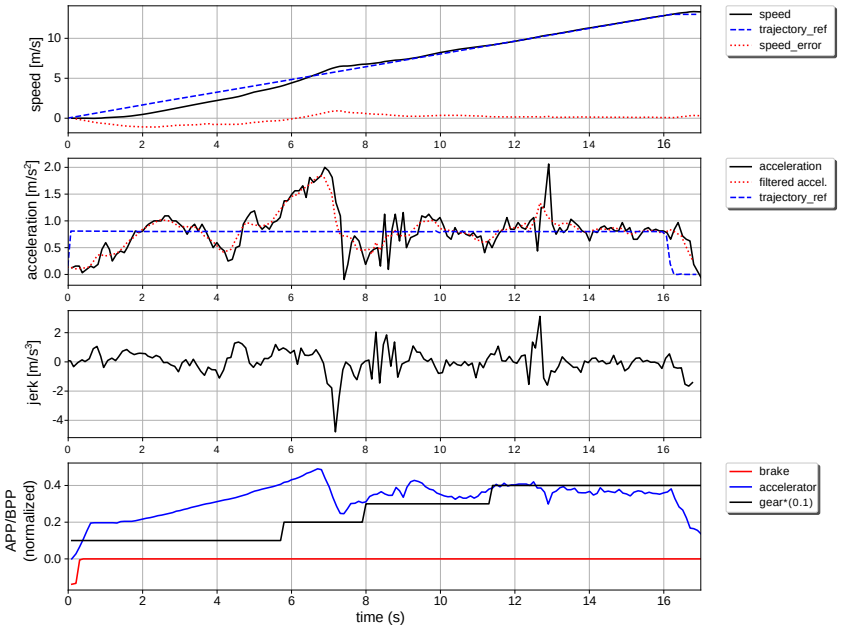
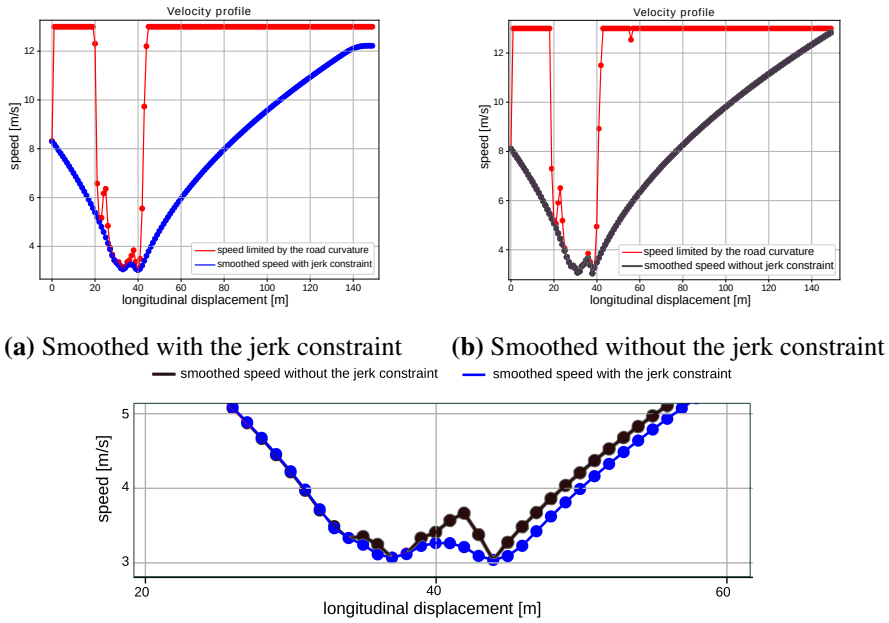


Fig. 3.20: Speed up tests for two trajectories with and without a jerk constraint: in the first trajectory, a jerk constraint and the MIG model constrain the acceleration's upper bound to increase to 1.5 $\mathrm{m/s^2}$, thereby preserving passenger comfort. In the second, even with the acceleration constraint of 0.8 $\mathrm{m/s^2}$ the passenger comfort lost. Plotted here are: the vehicle velocity, acceleration, jerk, accelerator pedal position (APP) normalized between [0,1] and brake pedal position (BPP) normalized between [-1,0] over time during the speed up test. In the first trajectory compare to the second one, maximum speed error and consequently the maximum jerk were decreased from 1.1 $\mathrm{m/s}$ to 0.5 $\mathrm{m/s}$ and from 4.79 $\mathrm{m/s^3}$ to 2.25 $\mathrm{m/s^3}$.

Slow down for an U-turn

In an U turn, the road curvature limits the speed as declared in Eq. (3.1); therefore, direction of acceleration changes several times. A speed constraint example created by an U turn road curvature is illustrated with a red line in Fig. 3.21. In the picture, the smoothed trajectories with and without jerk constraint are illustrated with blue and black lines.

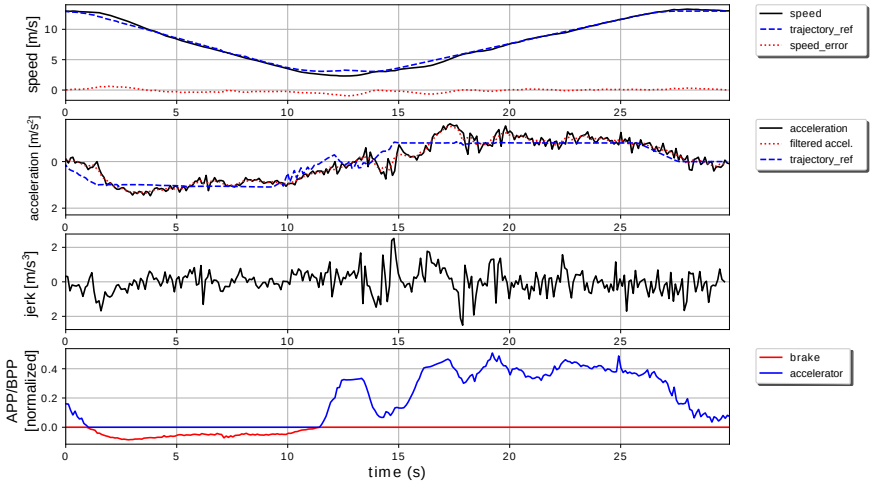


(a) Smoothed with the jerk constraint **(b)** Smoothed without the jerk constraint

Fig. 3.21: trajectories at an U turn: the graphs plot the trajectory points speed versus their longitudinal displacement. The red line is a speed constraint created by an U turn road curvature as given by (3.1). The blue line is the smoothed speed profile with a constant jerk(0.85 m/s^3) and limited acceleration in [-1.0,0.7] m/s^2. The black line is the smoothed speed profile without jerk limitation but with the same limited acceleration.

The result of the tests driving in an U-turn are depicted in Table 3.2. In the first test, we used Alg.2, which provides a double-S trajectory using Alg.5, to decelerate from 13 m/s to 3 m/s and again to accelerate to 13 m/s. As shown in Fig. 3.22.a, a constant value (-1.0 m/s^2) limited the acceleration in the trajectory while its jerk value constraint was 0.85 m/s^3. The trajectory following task was performed with sufficient accuracy with the average error ($\bar{\mathbf{e}}$) of 0.25 m/s and a maximum error of 0.97 m/s from the desired speed profile. Passenger comfort was provided with the average jerk ($\bar{\mathbf{j}}$) of 0.49 m/s^3 and maximum jerk of 2.25 m/s^3. The passenger comfort criteria show improvement compare to the method without the jerk constraint which is shown in Fig. 3.22.b.

With jerk constraint



Without jerk constraint

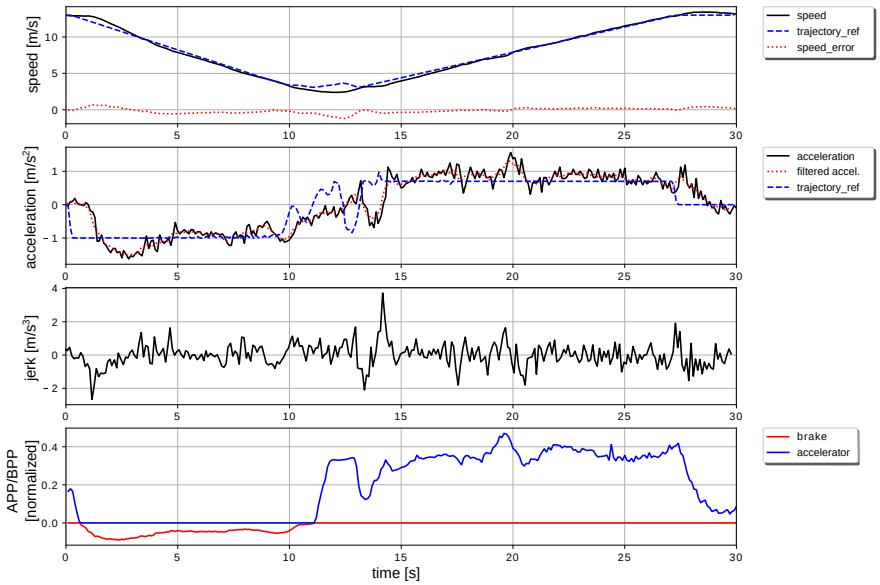


Fig. 3.22: Driving in an U turn with two trajectories with and without jerk constraint while the acceleration was limited between [-1, 0.7] m/s^2: plotted here are the vehicle speed, acceleration, jerk, accelerator pedal position (APP) normalized between [0,1], and brake pedal position (BPP) normalized between [-1,0] over time. The road curvature limits the speed as given by (3.1); therefore, the speed was decreased to 3 m/s following the road curve. The test results are compared in Table 3.2. Constraining the jerk decreased the maximum speed error and the maximum jerk from 1.2 m/s to 0.97 m/s and from 3.73 m/s^3 to 2.52 m/s^3, and provided a more comfortable ride for passengers.

Table 3.2: Compare results of three parameters groups for the U-Turn case: each row illustrates 1) if the MIG model was used to constrain the trajectory acceleration or not, 2) the trajectory jerk constraint, 3) the trajectory acceleration limits. The maximum error between the vehicle speed and the trajectory reference (max(**e**)), its mean absolute ($\bar{\mathbf{e}}$), the vehicle maximum jerk (max(**j**)), and its mean absolute ($\bar{\mathbf{j}}$) are compared.

	Trajectory Parameters			Results			
	Model	j_{max}[m/s^3]	a_{max}[m/s^2]	max(**e**)[m/s]	$\bar{\mathbf{e}}$[m/s]	max(**j**)[m/s^3]	$\bar{\mathbf{j}}$[m/s^3]
1	✓	0.85	[-1.0,0.7]	0.97	0.25	2.52	0.49
2	×	∞	[-1.0,0.7]	1.2	0.31	3.73	0.51

Interaction with traffic

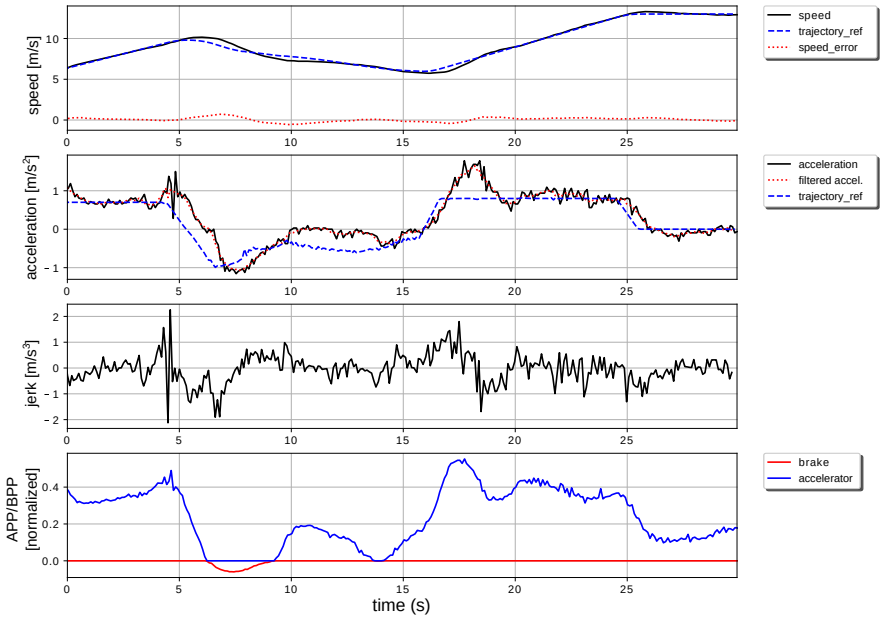
Equation (4.4) limits the trajectory speed points based on the lateral and longitudinal distance from the obstacle. The result of the tests during which the car interacts with obstacles are depicted in Table 3.3.

In the tests, the ego car interacted with an obstacle while the acceleration was limited over the interval [-0.1,0.7]. The car reduced the speed to avoid the collision with an obstacle, and then when the obstacle in front turned to another street and make free space for the ego car, it accelerated. The results of the test with the bounded jerk are shown in Fig. 3.23.a. The passenger comfort is provided by the maximum jerk of 2.25m/s^3. In the similar test (Fig. 3.23.b) without jerk constraint the maximum jerk jumps to 5.37m/s^3.

Table 3.3: Compare results of two parameters groups in interaction with obstacles: each row illustrates 1) if the MIG model was used to constrain the trajectory acceleration or not, 2) the trajectory jerk constraint, 3) the trajectory acceleration limits. The maximum error between the vehicle speed and the trajectory reference (max(**e**)), its mean absolute ($\bar{\mathbf{e}}$), the vehicle maximum jerk (max(**j**)), and its mean absolute ($\bar{\mathbf{j}}$) are compared.

	Trajectory Parameters			Results			
	Model	j_{max}[m/s^3]	a_{max}[m/s^2]	max(**e**)[m/s]	$\bar{\mathbf{e}}$[m/s]	max(**j**)[m/s^3]	$\bar{\mathbf{j}}$[m/s^3]
1	✓	0.85	[-1.0,0.7]	0.71	0.21	2.26	0.4
2	×	∞	[-1.0,0.7]	1.58	0.36	5.37	0.67

With jerk constraint



Without jerk constraint

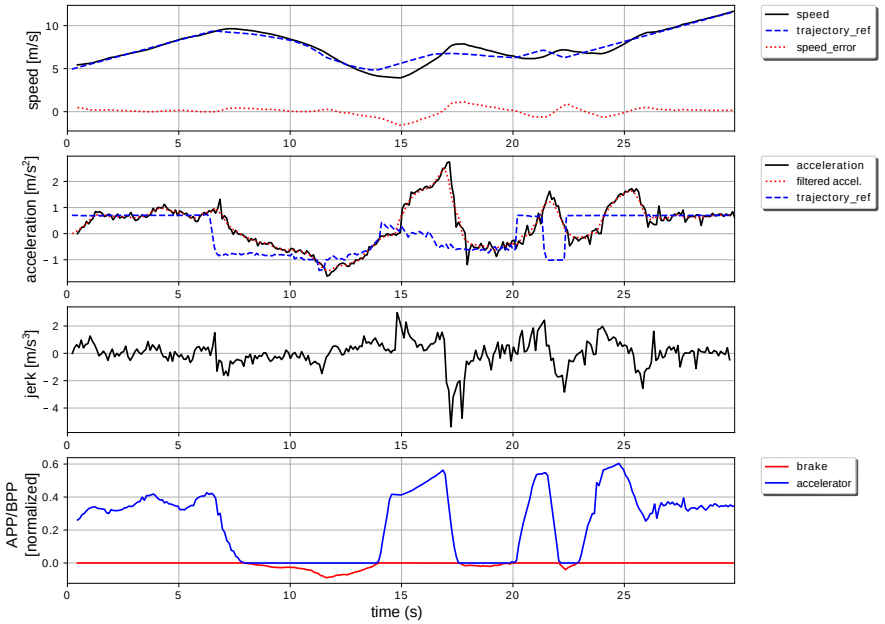


Fig. 3.23: Interaction with an obstacle for two trajectories with and without jerk constraint while the acceleration is limited over the interval [-1, 0.7] m/s^2: plotted here are the vehicle velocity, acceleration, jerk, accelerator pedal position (APP) normalized between [0,1] and brake pedal position (BPP) normalized between [-1,0] over time. The car reduced the speed to avoid the obstacle collision. When the obstacle in front turned to another street, it made free space for the ego car to accelerate. In the case of using jerk constraint, the passenger comfort was provided by the maximum jerk of $2.25 m/s^3$, while without limiting the jerk in the trajectory, the maximum jerk increased to $5.37 m/s^3$ and caused passenger discomfort. The test results are compared in Table 3.3.

3.6 Conclusions

This chapter explains the reactive trajectory planning method. The drive spline is re-sampled. The maximum speed of each trajectory point is defined based on the road curvature, traffic rules, and dynamic objects. The lateral distance to the obstacle is found to design an algorithm to swerve or reduce the speed. The car model, jerk, acceleration, speed constraints are considered in the proposed smoothing method. Practical issues such as trajectory freezing, and interpolation methods are also discussed in this chapter. Finally, the proposed methods are compared with the unbounded jerk method in simulations and experiments. The simulation shows that in addition to providing passenger comfort, this method leads to less control output and actuator force. The experiment results show that the passenger comfort can be provided with less average and maximum jerk.

Chapter 4

Trajectory Planning Based on a Developed Force Vector Field

This chapter addresses the navigation of autonomous cars using vector fields in structured road maps. The ego car should follow the desired road lane while avoiding obstacles on the road. The proposed approach combines map data and path planning algorithms and provides vector fields that navigate the ego car toward the road. The vector fields are calculated offline to obtain the force vector of each point efficiently. By making offline calculations, especially when the autonomous cars drive inside cities with restricted areas, the calculation capacity of the car's computer might be saved.

A vector field is like a magnetic field in that the desired road lane attracts the ego car. This chapter presents a novel approach for autonomous vehicle navigation in environments with a structured map by creating offline force fields, which specify the desired heading angle of the vehicle. The approach fulfills *path following* and *lane keeping* tasks. The force fields are augmented and modified locally by the presence of obstacles as a result of the obstacles force field. The velocity of the vehicle and the distance of the vehicle from the path are taken into account, to find feasible force vectors. A control law is also developed to define the velocity direction and the desired steering angle based on the angle between the car and the vector field. The contribution of this chapter with respect to state of the art is summarized as follows.

- Instead of considering a single point as a goal, a lane of goal points is considered, which specifies the path. The goal points are specified on the lane based on the vehicle speed and the vehicle lateral distance to the lane.

- A force field is generated offline to reach high rate and computationally light navigation. The desired force vector is updated online in case of detecting obstacles in the vicinity of the car.
- To optimize the memory usage, and achieve a smooth motion, the vector fields are interpolated not only in the space domain but also in the speed domain.

The rest of the chapter is organized as follows: Section 4.1 describes the vector fields. In Section 4.2 the obstacle repulsive vectors are designed. Then, in Section 4.3, numerical simulation and experimental results are provided to show the effectiveness and efficiency of the proposed approach. Finally, conclusions are outlined in Section 4.4.

4.1 Defining a Force Vector Field for Autonomous Car

In order to perform the path following task using force vector field approach, let G be the desired path to be followed, *e.g.*, a lane of the road. This path is considered to be a collection of goal points. The area around the path is discretized with an appropriate resolution. The idea is to define a force field $\mathbf{F}$, such that—for each position $P = (x, y)$—the goal pulls the autonomous car, and obstacles repel the autonomous car through their local artificial repulsive forces. Thus, the attractive force vector $\mathbf{f}_G(P) \in \mathbb{R}^2$ and repulsive force vectors $\mathbf{f}_{O_i}(P) \in \mathbb{R}^2$ constitute in each position P the force vector $\mathbf{f}(P)$ as:

$$\mathbf{f}(P) = \mathbf{f}_G(P) + \sum \mathbf{f}_{O_i}(P) \tag{4.1}$$

The following shows how to calculate $\mathbf{f}_G$ and $\mathbf{f}_{O_i}$ in each point.

In calculating $\mathbf{f}_G$, a constant magnitude for it in each point is considered, and its orientation is calculated, which specifies the appropriate vehicle heading angle in order to reach and follow the path. The direction of $\mathbf{f}_G(P)$ is defined by a vector from the current position of the car to a goal point (P_g) on the path. In the following sub-sections, the goal points are defined first, and then the process of generating an attractive force field is described.

4.1.1 Goal Points Definition

The main goal of the proposed approach is to follow the desired path. A road lane can be considered to be a desired path without loss of generality. If the objective was only to reach the road lane, for each point P, the choice of the nearest point (P_n) on the path as a goal point would be the best choice ($P_g = P_n$). However, to follow the lane, the goal point must be ahead of the nearest point which pulls the car along the path. When the autonomous

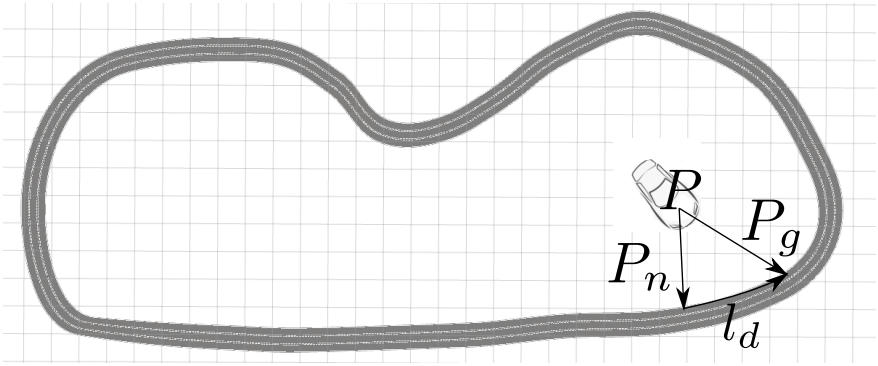
vehicle reaches the path, the look-ahead point is chosen based on the car's speed in order to obtain smooth changes in the desired heading angle of the car and to prevent oscillation of its heading angle.

$$P_g = \mathsf{G}(P_n, l_d) \tag{4.2}$$

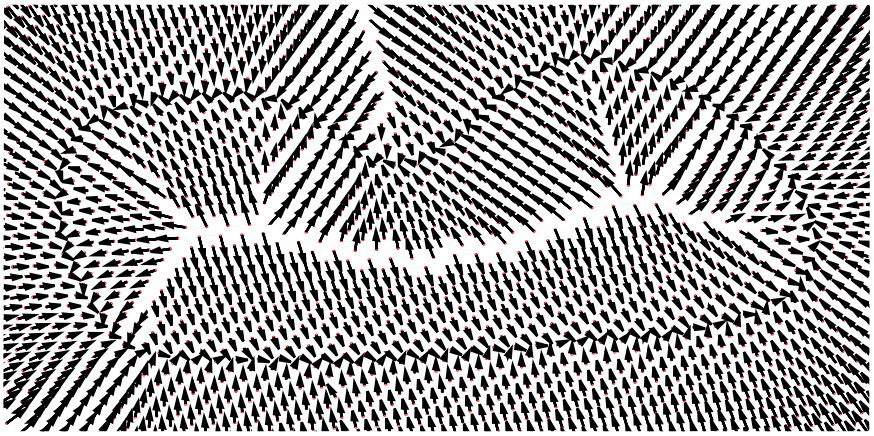
The point P_g is l_d meters along the path ahead of P_n. l_d is defined based on the speed and the car distance to the path.

$$l_d = \alpha \min\left(|v|, \frac{|v|}{||\vec{PP_n}||}\right) \tag{4.3}$$

in which $v \in \mathbb{R}$ is the ego car's speed, $||\vec{PP_n}||$ is the distance between the ego car's position and its nearest point on the road path, and $\alpha \in \mathbb{R}$ is a tuning parameter. When the ego car is far from the path, l_d is approximately zero, while once the ego car is on the road, l_d reaches its maximum value. Fig. 4.1a depicts P_g, P_n for a car outside the path.



(a) Tempelhof test area is discretized in position. Vector $\vec{PP_g}$ shows the attractive force vector direction at a particular point outside the path (P). The point P_g is l_d meters along the path ahead of the vehicle nearest point on the path (P_n).



(b) A vector field obtained for a constant velocity of 2 m/s; this guides the car along the desired path.

Fig. 4.1: A typical map, path, and the corresponding vector field for a constant speed.

For the surrounding area of the path with length of l and width of w, the vector field, with resolution r, is saved in a matrix $\mathbf{F}$. Each element of the matrix is a 2D force vector containing f_x in x-direction and f_y in y-direction.

As it is evident from (4.3) the force vector field depends on the car speed, *i.e.*, the car's ability to steer depends on its speed: at lower speeds, the autonomous car can effectively and safely rotate more than in higher speeds. Therefore, the force vector fields are calculated for minimum and maximum speeds offline as $\mathbf{F}_{min}$ and $\mathbf{F}_{max}$, respectively. Then, the appropriate force vectors will be interpolated online for a specific speed.

4.1.2 Interpolation

As we have seen, a force field is calculated for a discretized set of positions at minimum or maximum speeds. In order to use it in any position in the field and any speed in range, two interpolations are utilized:

- Interpolation in the space domain, that calculates the force vector for any position based on the surrounding force vectors within a specified vicinity of the position.
- Interpolation in the speed domain, that calculates the appropriate force vector for a specific speed based on two force vectors (for minimum and maximum speeds) at the same position.

The space interpolation is performed first, and is followed by the speed interpolation. In the space domain, the force vector in a specific position P is the weighted average of the force vectors belong to the elements of $\mathbf{F}$ which P located between them. Alg 9 presents the spatial

Algorithm 9: The spatial interpolation algorithm

input : for grid points around the car position, distance to the car position $d_i = [d_1, d_2, d_3, d_4]$

output : the interpolated force vector $\mathbf{f}(P)$

$w_i = \frac{1}{d_i + \varepsilon}$

$w_i = \frac{w_i}{\sum w_i}$ ▷ `normalization step`

$\mathbf{f}_{min/max}(P) = \sum w_i \mathbf{F}_{min/max}[i]$

interpolation. First, the grid points near the current car position (P) are selected, and the distance of each grid points from the current car position (d_i) is calculated. The inverse of d_i is the weight of each grid point[1]. Then, weights are normalized to keep the result value in the range of the input elements. Finally, the interpolated force vector ($\mathbf{f}(P)$) is calculated as

[1] a small value (ε) is added to the denominator to avoid big values in case of a very short distance.

the weighted average of the input force vectors.
Fig. 4.2(a) shows an example of the spatial interpolation for a car approaching the path. The

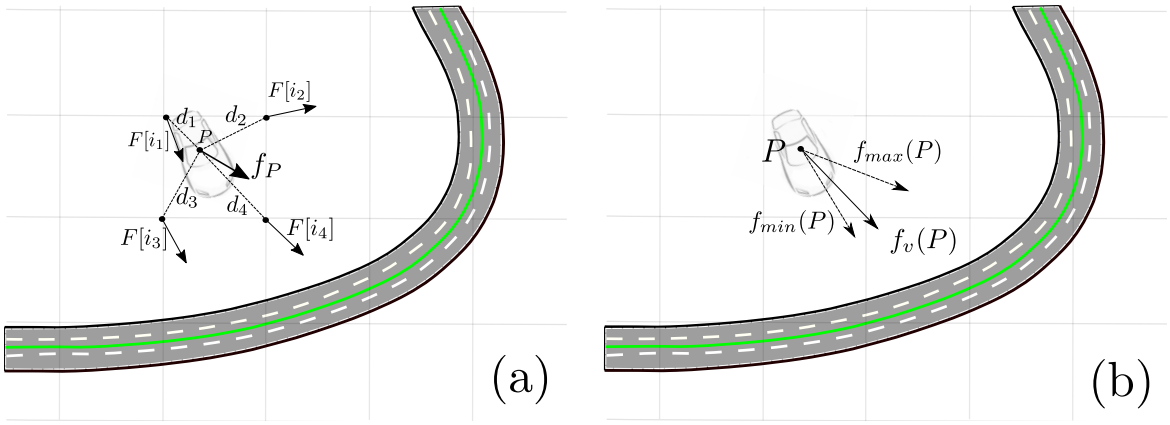


Fig. 4.2: Force vector interpolation: a) in the space domain, b) in the speed domain. The force vectors are calculated offline and the forces for grid points around the car position are selected ($\mathbf{F}[i]$). The force at the car position ($\mathbf{f}(P)$) is the weighted average of $\mathbf{F}[i]$. In space domain $\mathbf{f}(P)$ is calculated twice - once time for minimum velocity $\mathbf{f}_{min}(P)$ and once time for maximum velocity $\mathbf{f}_{max}(P)$. From this, the final force vector $\mathbf{f}_v(P)$ is calculated based on the desired car velocity v in speed domain.

next interpolation step is done in the speed domain; the force vector ($\mathbf{f}_v(P)$) in a specific speed (v) in a specific position (P) is expressed as:

$$\mathbf{f}_v(P) = \mathbf{f}_{min}(P) + \frac{\mathbf{f}_{max}(P) - \mathbf{f}_{min}(P)}{v_{max} - v_{min}} (v - v_{min}) \tag{4.4}$$

where $\mathbf{f}_{min}(P)$ and $\mathbf{f}_{max}(P)$ are the force vectors of the minimum and maximum speeds (v_{min} and v_{max}), which are calculated in the previous interpolation. Fig. 4.2(b) shows an interpolation example in the speed domain for a car approaching the path.

4.1.3 Motion Direction and Steering Angle

The motion direction and the steering angle at each instance are obtained by projecting the force vector on the car frame. The longitudinal and lateral element of the force vector, with respect to (w.r.t.) the body frame of the vehicle are denoted with f_x and f_y, respectively. If f_x is negative, the car drives backward, and if f_x is positive, the car moves forward. The steering angle (ψ) is expressed as:

$$\psi = \begin{cases} \beta \text{ atan}(f_y, f_x) & \text{if } f_x \geq 0 \\ -sign(f_y)\psi_{max} & \text{if } f_x < 0 \end{cases}$$

where $\beta \in \mathbb{R}^+$ is a positive tuning parameter, $atan(f_x, f_y)$ is representing the angle between the force vector ($\mathbf{f}(P)$) and the car heading vector, and $\psi_{max} \in \mathbb{R}^+$ is the maximum steering angle.

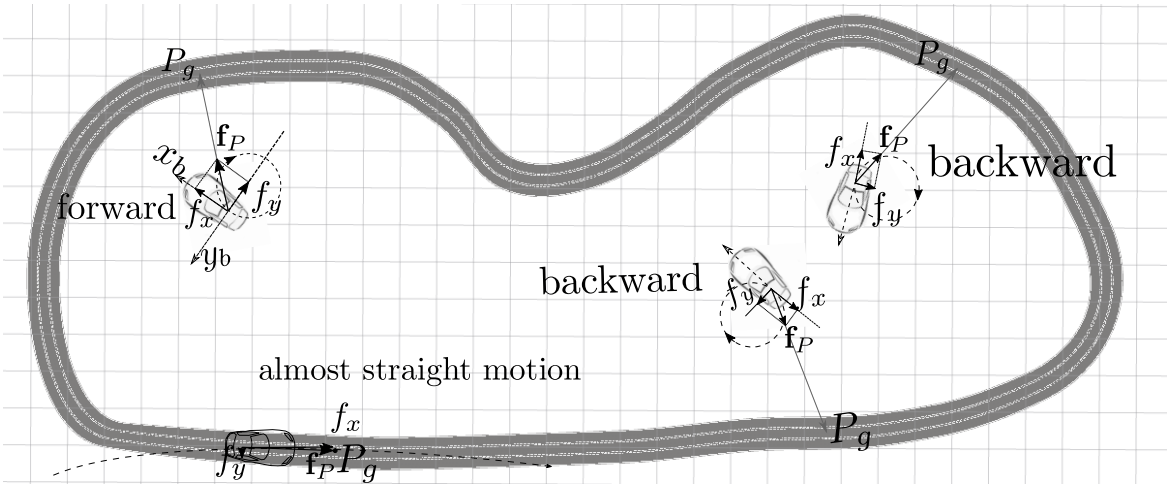


Fig. 4.3: Direction of motion and steering for different orientations of the car in various positions w.r.t. the path: the point P_g is the look-ahead point on the path. The steering angle at each instance is obtained by projecting the attractive force vector ($\mathbf{f}_P$) on the car frame (x_b,y_b). The longitudinal and lateral element of the force vector, w.r.t. the body frame of the vehicle are denoted with f_x and f_y, respectively. If f_x is negative, the car drives backward, and if f_x is positive, the car moves forward. The circles represent the car motions as a result of the steering corresponding to $\mathbf{f}_P$.

Fig. 4.3 shows the force vector ($\mathbf{f}_P$) in four different car positions w.r.t. to the path. In each instance, $\mathbf{f}_P$ is projected to the body frame of the car. If $f_x > 0$, the car moves forward, while $f_x < 0$ means a backward motion of the car. The circles in the picture represent the car motions as a result of the steering corresponding to $\mathbf{f}_P$.

4.1.4 Lane Changing

An autonomous car driving on a road quite often needs to change lanes to overtake other vehicles, to change its speed, or to find a better path with less traffic. The motion planning subsystem of the autonomous car is in charge of selecting the lane. When lane changing action is needed, the system only needs to load the vector field of the new lane[1]; then, the ego car will automatically change lanes. Fig. 4.4 shows a schematic representation of the force field for a road of three lanes, and two consecutive lane changes are necessary in order to perform an overtaking action. Once Lane 1 is the desired lane, its corresponding force vector navigates the car along the desired path. By changing the desired lane to the second

[1]The force field for each lane is separately calculated.

lane, the lane-changing action takes place automatically by loading the force field of Lane 2 and using it as the navigation source.

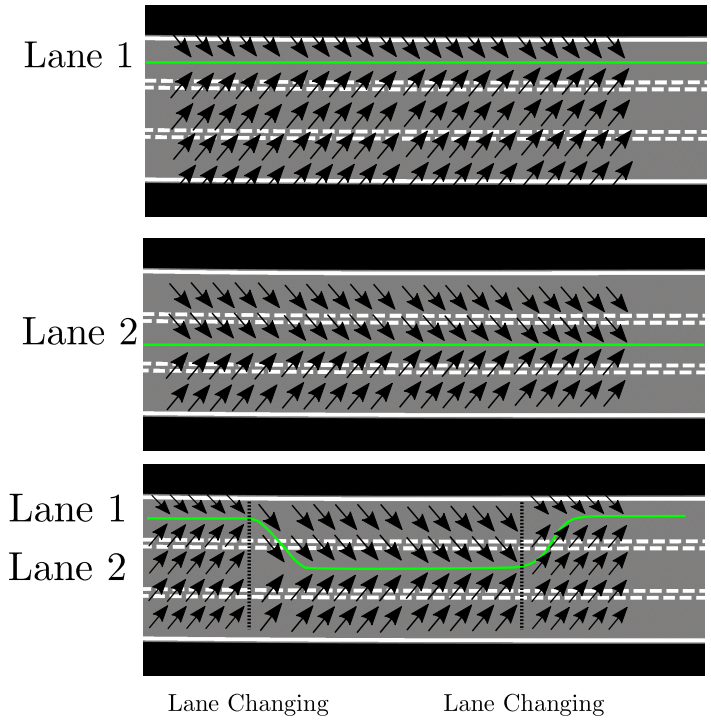


Fig. 4.4: For each lane, the vector field is separately generated and saved. When a lane-changing command received, the appropriate force field is loaded, and lane changing takes place automatically by switching between two vector fields.

4.2 Obstacle Repulsive Force Field

Collision avoidance is an essential demand for any path following approach in the navigation of autonomous cars. Besides the current framework of using force fields to navigate the car along the desired path, a separate force field for each detected obstacle should be defined. In order to avoid obstacles, a repulsive force field is defined for each obstacle whose force element in each point is added to the offline calculated attractive force field (F_G). The repulsive force vector of an obstacle is inversely proportional to the distance to it. The force vector around an obstacle, centered in $P_{O_i} = (x_i, y_i)$ in each point $P = (x, y)$ is defined as:

$$\mathbf{f}_{O_i}(P) = \begin{cases} \frac{k d_o}{(||d_o||^2 + \varepsilon)} & \text{if } \frac{(d_x C\theta - d_y S\theta)^2}{a^2} + \frac{(d_x S\theta + d_y C\theta)^2}{b^2} < 1 \\ 0 & \text{otherwise} \end{cases} \tag{4.5}$$

where $d_x = x - x_i$, $d_y = y - y_i$, $d_o = ||\vec{P_{O_i}P}||$ is the distance of the center of the obstacle from the point P, and $k \in \mathbb{R}$ is the tuning parameter to adjust the intensity of the repulsive force. The parameter $k = ||\mathbf{f}(P)||$ is chosen to keep the intensity of the repulsive force in range of attractive force. As previously stated, the magnitude of the car velocity is constant, and the force field controls the direction of velocity; therefore, the intensity of $\mathbf{f}_O$ is defined proportional to the attractive force $\mathbf{f}(P)$. The area of influence for each obstacle locate at $P_{O_i} = (x_i, y_i)$ is defined as an ellipse–that is rotated along the path with angle θ–that can be tuned by parameters $a, b \in \mathbb{R}$ depending on the car velocity. The obstacle influence threshold in the longitudinal direction of the path is longer than in lateral direction ($a > b$). Fig. 4.5 depicts the repulsive force field around an obstacle.

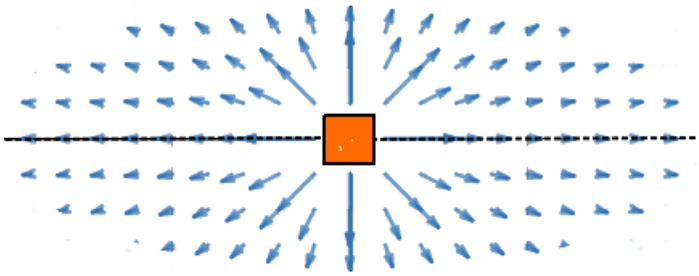


Fig. 4.5: The vector field around an obstacle resembles an ellipse that is rotated along the path. The lengths of two ellipsoid axes is tuned based on the vehicle speed.

4.3 Simulations and Experiments

In order to evaluate the feasibility and to assess the efficiency of the proposed method, simulations and experiments are performed on a model car designed and fabricated at Freie University Berlin for research and educational purposes.

4.3.1 Experimental Setup

The experimental setup was a four-wheel-drive model car, whose size was one-tenth of its actual counterpart. A BLDC motor equipped with an encoder drives it, and a DC servo-motor controls its steering. The perception system of the model car consists of different sensors: an IMU, a Lidar, an RGBD camera, and a fish-eye camera (Fig .4.6a). Data acquisition, navigation, and control are performed on-board using an Odroid XU4 computer. The model car can communicate with external systems - *e.g.,* PC, smartphone, or other model cars - via WiFi connection for monitoring, data collection, and supervisory control purposes. A Li-Po

battery powers the whole system. The software is implemented using C++ and Python in the ROS framework installed on the Linux core of the on-board computer.

The car is tested on a test field shown in Fig .4.6b. In order to facilitate localization, four colored lamps are installed on the ceiling over the test field. The upward-looking fish-eye camera observes the lamps in each moment and using a real-time range-based localization [73], localizes the vehicle on the test field. This indoor localization system is called *Visual-GPS* as it imitates the outdoor Global Positioning System in which the position information is obtained from the external sources, namely the satellites around the earth. Lidar can be used to detect and avoid static and dynamic obstacles. The RGBD camera can not only be used in conjunction with Lidar to obstacle avoidance, but also to lane-keeping.

The whole system, including the model car with all its subsystems and the test field, is simulated using the ROS framework and visualized by Rviz. The model car can be operated in either simulation or real mode. The proposed approach for navigation is implemented using python [74], and validated in simulations and experiments, that are reported in the rest of this section.

4.3.2 Simulation Results

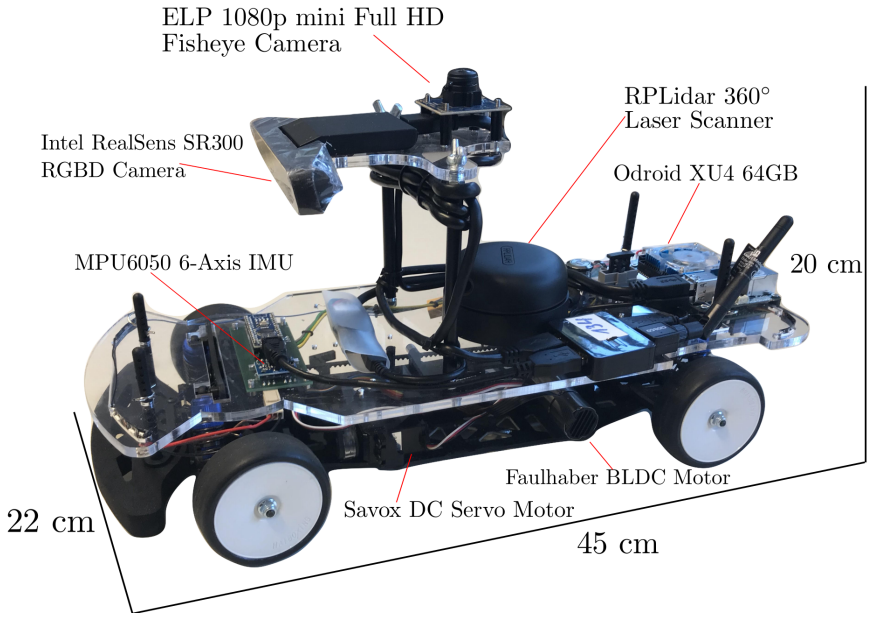
The proposed method was tested on a test field that is a circular road with two lanes, several connections and intersections between its different sections. The circumference of the road map can be inscribed in a $6 \times 4\,\mathrm{m}^2$ rectangle (Fig 4.6b). The same map is also used for the simulation study. Each one of the two lanes might be the desired path.

The area was discretized to a $10\,\mathrm{cm}^2$ grid. For each corner of the grids, the nearest point on the road lane was calculated based on KD-tree algorithm [75]; then, the vector field for each one of the lanes was calculated and saved separately. A visualization of the vector field for Lane 2 of the road circle (the inner lane) is shown in Fig. 4.7 in which the center of each grid cell is shown with a red dot and the outpointing arrow shows the appropriate heading of the car to follow the lane, which is shown by colored dots.

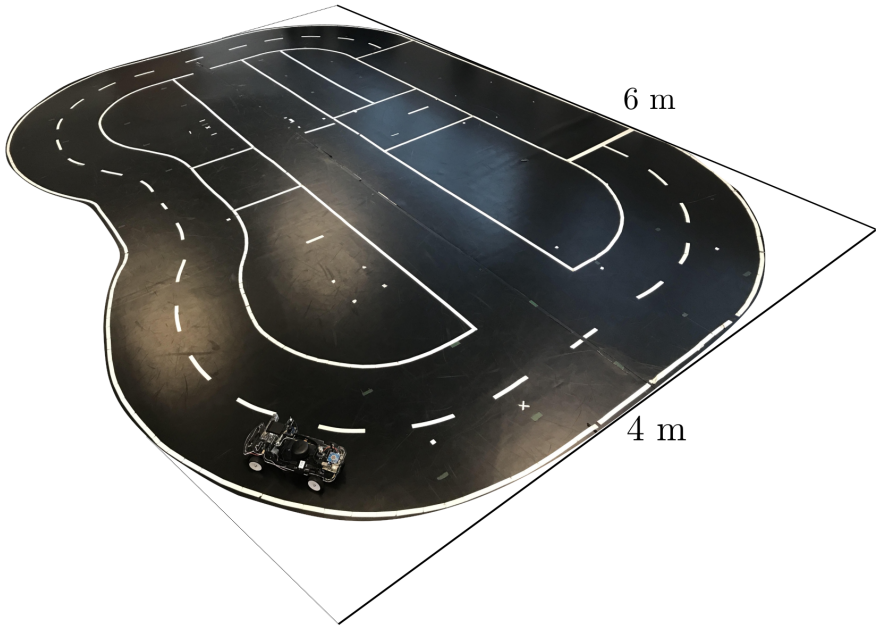
The proposed approach was evaluated in four test cases in order to assess the functionality of the method in different circumstances separately. The test scenarios were the followings:

- lane keeping with initial heading toward the lane,
- lane keeping with initial opposite heading with respect to the lane,
- lane changing, and
- overtaking and obstacle avoidance.

In all cases, the constant speed of the car was $0\,6\,\mathrm{m/s}$. In the following, each of these simulations and their results is described.



(a) The experimental test-bed (model car): a four-wheel-drive vehicle of one-tenth of a real car, driven by a BLDC motor and steered by a DC servo-motor, equipped with IMU, Lidar, RGBD camera, and fish-eye camera, and an Odroid XU4 computer.



(b) The test field: A circular road with two lanes that has connections and intersections between its different sections. It is inscribed within a rectangle of 6 m length and 4 m width.

Fig. 4.6: Model car in the test field.

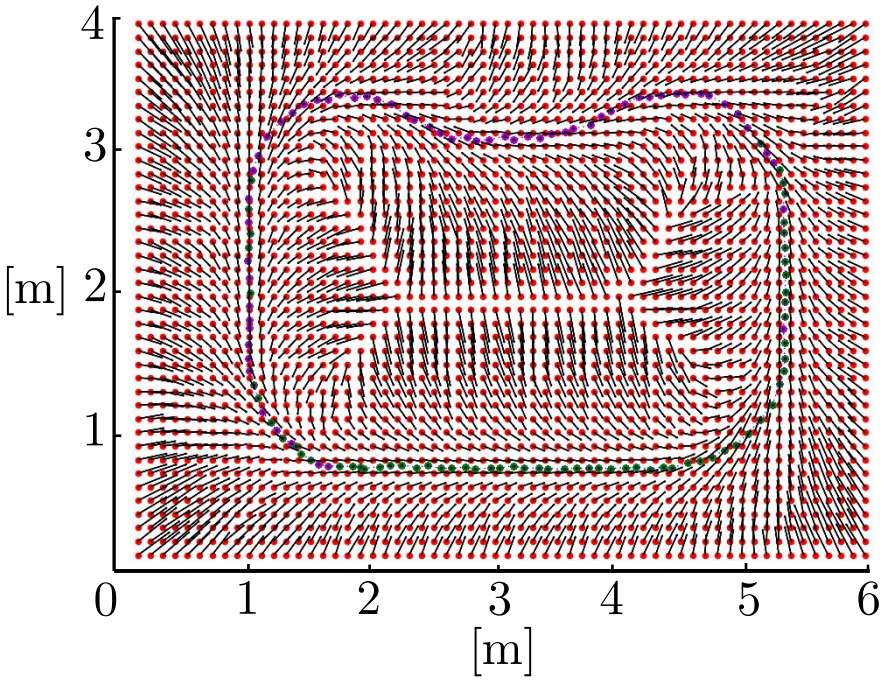


Fig. 4.7: Generated force field for the second lane of the road map used in simulations and experiments: the center of each grid cell is shown with a red dot. The outpointing arrow indicates the appropriate heading of the car to follow the lane, which is shown by colored dots.

Lane keeping with initial heading toward the lane

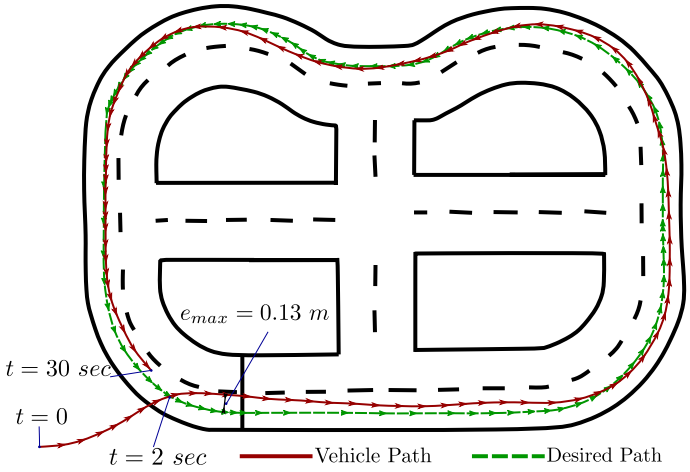


Fig. 4.8: Simulation result of lane following with initial heading toward the lane: the desired path is the outer lane shown by green and the result is shown by red. The artificial force vector field navigated the car toward and along the desired path from a location inside the map.

Fig. 4.8 shows the simplest case in which the model car was initially located outside the road and was heading towards the road. The desired path to follow was the outer road

lane. The result of the test is depicted in the picture, the path following task was performed sufficiently accurate with the average error of 0.03 m and a maximum error of 0.13 m distance from the desired path, which compared to the dimension of the model car is acceptable.

Lane keeping with initial opposite heading with respect to the lane

In this test, the vehicle was initially located on the lane, but with opposite heading with respect to the direction of the lane. As is shown in Fig. 4.9, the vehicle initially moved

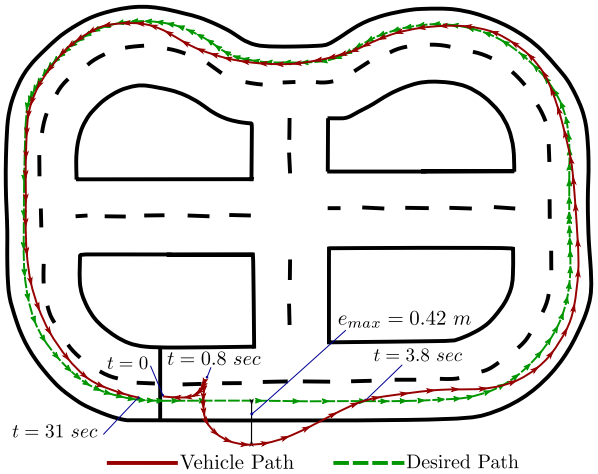


Fig. 4.9: Simulation result of lane keeping with initial opposite heading with respect to the lane: the desired path is the outer lane represented by the green and the actual path traveled is charted with the red. The artificial force vector field navigated the car toward and along the desired path from a location inside the map.

backward in the direction of the guiding force vectors until its heading became perpendicular to the path. Then it took the forward direction and the guiding force vectors attracted it to the desired path. The average and maximum errors in this test were 0.04 m and 0.42 m distance from the desired path, respectively. The comparably significant error from the moment the vehicle heading was perpendicular to the path till its heading becomes along the path, *i.e.*, between $t = 0.8$ to $t = 3.8$ s is due to the nonholonomic kinematic of the vehicle and its constant speed.

Lane changing

In the real world, lane changes are inevitable. Human drivers choose to change their driving lanes after considering the situation and objectives, *e.g.*, destination, time, traffic. As previously stated, the vector fields for both driving lanes were separately generated and saved off-line. A mission planning system, that could be an AI system based on cognitive

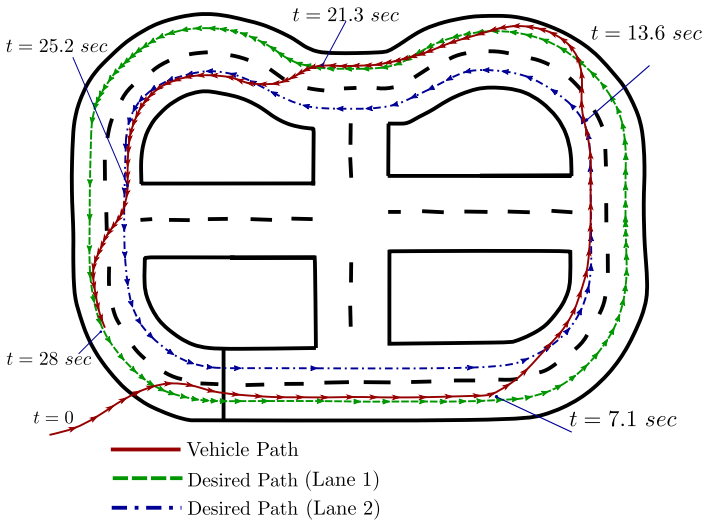


Fig. 4.10: Simulation result of lane changing during path following task: the desired path switched between the outer road lane (green path) and inner road lane (blue path). The vector fields for both driving lanes were separately generated. A human operator commanded the vehicle to change the lane four times. By switching the vector fields, the car was navigated toward and along the desired path. The car path is shown by red.

interpretation of the circumstances and objectives, could command our navigation system to change the driving lane. However, an automatic mission planning system is beyond the scope of this thesis. Therefore, to test the lane changing task, a human operator commanded the vehicle to change the lane. The vehicle was initially located outside the road heading toward the direction of the lane, and the initial desired path was the outer lane. After reaching the lane, the human operator sent the lane changing command (using keyboard) four times, and, as it can be observed in Fig. 4.10, the lane-keeping and lane changing in constant speed were performed sufficiently accurate.

Overtaking and obstacle avoidance

Overtaking is another common action while driving a vehicle. In this test, there existed two static obstacles, *e.g.*, parked vehicles, on the same lane on which the test car drove. As is shown in Fig. 4.11, our autonomous vehicle performed the overtaking maneuver and came back to the desired path keeping its constant speed by considering the obstacles' repulsive force vectors along with the guiding vector field of the desired path. The overtaking actions were sufficiently smooth, and the path following on the rest of the path was accurate.

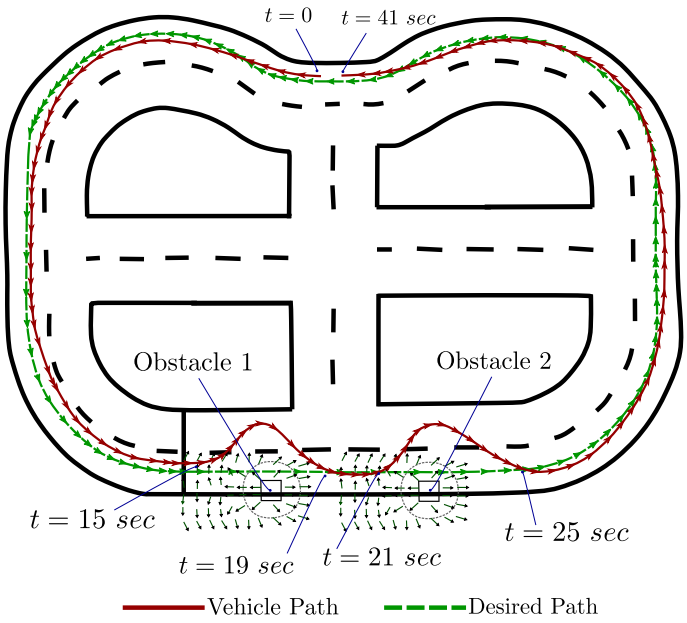


Fig. 4.11: Simulation result of overtaking and obstacle avoidance (lane keeping in presence of static obstacles): the desired path is the outer lane shown by the green line and the path travelled is shown by the red. The black boxes are the two obstacle on the the vehicle route. The black arrows show the obstacles' repulsive force vectors which caused the overtaking actions.

4.3.3 Experiment Results

In order to experimentally evaluate the proposed approach, the proposed approach was tested on the model car shown in Fig. 4.6a over the test field shown in Fig. 4.6b. Similar to the simulation test, a human operator decided to change the lanes. The model car was initially located on the road heading along the path, and the desired constant speed of the car set to be 0.6 m/s. The human operator instructed the car to change lanes fifth times during the path following task, which took 48 seconds. As a result of the experiment in Fig. 4.12 shows, the lane-changing actions took place smoothly. The lane-keeping was sufficiently accurate such that in the first turn around the field, *i.e.,* between $t = 0$ to $t = 24$ s that no lane changing command is given, the average and maximum error were 0.045 m and 0.15 m distance from the desired path.

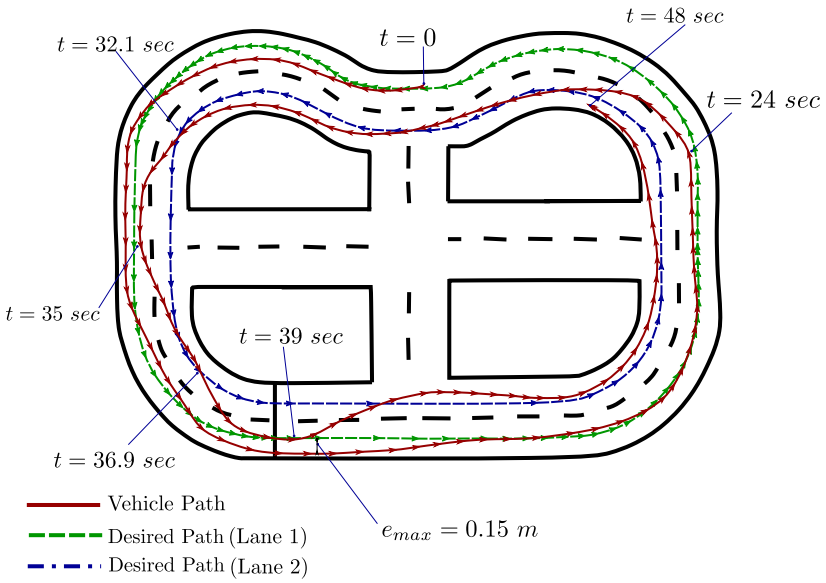


Fig. 4.12: Experimental result of lane changing during path following task: the desired path switched between the outer road lane (green path) and inner road lane (blue path). The model car followed the outer lane until second 24. Then the human operator instructed the car to change lanes five times during the path following task.

4.4 Conclusion

A vector field approach has been proposed for navigation and path following of autonomous cars. The vector fields are calculated and saved offline, which allows the system to perform path following task in real-time and with a very low computational load. The car distance from the path and the vehicle speed are taken into account to calculate the vector fields. The vector field calculations are performed for a discretized area (the area around the desired path), and the minimum/maximum car speeds. Then in each moment and speed, the appropriate force vector is calculated by interpolation on space and speed. Furthermore, a repulsive force field in the vicinity of each observed obstacle along the path is calculated online. The repulsive force from all obstacles is added to the attractive force field of the path to obtain the final force vector that specifies the desired heading of the car. This approach, as validated by simulations and experiments, is compatible with normal driving tasks such as lane-keeping, lane changing, and overtaking.

Chapter 5

Trajectory Planning using Flexible Unit-A* algorithm

This chapter proposes a trajectory planning approach for autonomous vehicles on structured road maps using the well-known A* optimal path planning algorithm. A safe optimal trajectory is generated through a three-dimensional graph which incorporates the two-dimensional position and time. (1) The graph is generated dynamically with fixed time differences and flexible distances between nodes, based on the vehicle's velocity, using a structured road map. (2) The position of dynamic obstacles is predicted over time along the road lanes. The grid unit of the search area changes, depending on the speed of the nodes. Decreasing or increasing the speed makes the grids shorter or longer, in other words makes grid units flexible. The structured road map in which the autonomous car moves, is not obstacle free. E.g. there exist other cars on the road, which are considered as dynamic obstacles. An approach to predict the position of the obstacles is also proposed, to evaluate which areas are obstacle free (in the future) during the execution of the FU-A* search algorithm. The rest of the chapter is organized as follows: Section 5.1 describes the utilization of the A* algorithm to solve the shortest path problem. In Section 5.2 the practical issues are highlighted. Then, in Section 5.3 numerical simulation results are provided to show the effectiveness and efficiency of the proposed approach. Finally, concluding remarks are outlined in Section 5.4.

5.1 The FU-A* algorithm

While in common A* algorithm the environment map is girded in fix units, in the proposed approach flexible grids are defined in 3 dimensions: x, y, and time. It means for each point (x,y), the next sequence points within a fixed time step are in a flexible distance.

Alg. 10 shows the steps of Flexible Unit A* (FU-A*) algorithm. In the first step, an open and a closed list are created and the closest point to the car position on the structured map is added to the open list as a first node. Then, the neighbor points are determined in the same lane and in the adjacent lanes, with different speed in the next T seconds. It is assumed that a car can change lanes in T seconds. A good practical approximation of T can be found for a specified speed range of a car. The neighbor points - which are free from collision with dynamic obstacles - are placed into the open list. The rest of the occupied neighbor points are placed into the closed list. The cost function for each point is then calculated, and the open list is sorted. This algorithm continues until the car reaches its goal. A goal point for

Algorithm 10: Flexible Units A* Algorithm.

```
input  : start(n), goal(n)
output : path
g: Cost of reaching node
h: Heuristic function
f: g+h
node(n): x, y, v, parent, f
open ← closestPoint(start)
closed ← 0
while open ≠ 0 do
    sort(open)
    n ← open.pop()
    if reachAroundGoal(n) = true then
        makePath(n)
    neighbors ← expandFlexibleUnits(n)
    for all the neighbors do
        if neighbor ∉ Obstacles then
            neighbor.f ← (n.g+g) + (n.p+p) + h
            if neighbor ∩ closed = 0 then
                open ← neighbor
            else
                closed ← neighbor
                closed ← n
                return 0
```

each planning would be N meters ahead of the car on the desired offline path which is given by the structured map. The following sections clarify each step of the FU-A* algorithm.

5.1.1 Neighbors

By considering time as a dimension, each grid can be defined with a specified speed and a different acceleration action. As a result, the distance between the grids are unfixed and have deterministic overlaps. In the structured environment, the number of lanes, an their positions are well defined. Therefore, there are at most nine actions are possible for each grid cell. As shown in Fig. 5.1, the actions are:

- "following the same lane",
- "go to the left lane" (if existing),
- "go to the right lane" (if existing),

while

- "decelerating",
- "continuing with the same speed",
- "accelerating".

The destination of the nine actions after the specified time (T) are called children nodes of a parent node. Although the child node may reach the same positions from different parent nodes, it usually will not have the same time stamp.

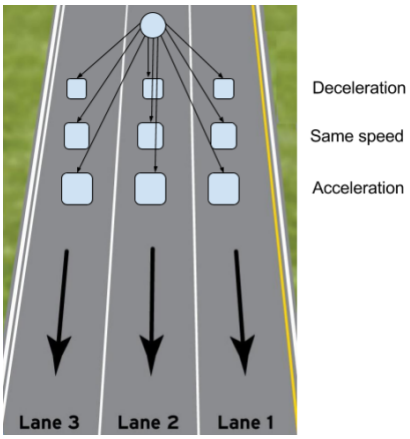


Fig. 5.1: Graphical representation of the proposed search algorithm: each parent node has nine neighbors which are defined as "decelerating", "continuing with the same speed", or "accelerating" in "the same lane" or in the "left/right lane"

5.1.2 Obstacle Position Prediction

The 3D laser scanner and stereo camera provide sensory data that, when combined, help to detect obstacles reliably[1]. The classified obstacle detection system provides obstacles' width, length, and current speed. Inferring the future behaviour of the obstacles based on their type and direction is a very challenging part of the urban driving. Many existing approaches assume dynamic obstacles as a quasi-static or assume that they linearly continue their path along their current heading and with their present velocity. In this chapter, it is assumed that the obstacle will remain in the same lane of the street, which may cause a change of the heading. For example, if the street is curvy, the car will follow the street. Therefore, we have more realistic predictions using structured maps. Signals from the vehicle ahead about a lane-changing could also be considered; however, since the replanning time (around 50 ms) is negligible compared to the lane change time (3 to 8 s), this complex prediction seems unnecessary. But, in an intersection area, all possible actions (going straight, turning to left or right) are considered.

As shown in Fig. 5.2, there are red bands (occupied area along the street) which predict the position of the obstacles. The band curves follow the street curve and their length are adjusted to consider the uncertainty of the obstacles' position within a lane. To predict the position of the obstacles over time, the travel distance at time sample i is evolved from the current velocity of the obstacle according to:

$$d_i = v_o\ (i\,T) + w_i \tag{5.1}$$

where $d_i \in \mathbb{R}^+$ is the travel distance calculated in at time sample i with the time step T, the current linear longitudinal velocity of the obstacle is denoted by v_o, and $w_i \in N(0, \sigma_i^2)$ is the process noise, which is assumed to be drawn from a zero mean Gaussian distribution with variance σ_i^2.

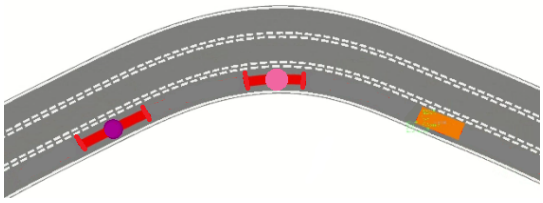


Fig. 5.2: Schematic representation of the proposed obstacle prediction: the orange box is a dynamic obstacle, the red bands show the predicted obstacle position in next T and $2T$ seconds.

[1]Recently, commercial products like Mobileye© and Ibeo© have made the classification of the obstacles easier

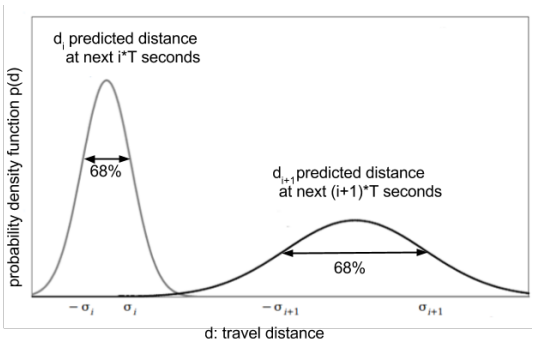


Fig. 5.3: Travel distance at time sample i vs time sample $i+1$: The horizontal axis represents travel distance of the obstacle center, and the vertical axis represents the probability density function of travel distance.

Fig. 5.3 compares the prediction of the obstacles travel distance at time sample i with time sample $i+1$. The horizontal axis represents travel distance of the obstacle center, and the vertical axis represents the probability density function of travel distance. The left Gaussian function shows the predicted position distribution of an obstacle at a certain time sample i. By increasing the uncertainty (variance) over time, the distribution of the probability density function becomes wider at the next time sample (the right Gaussian function).
One can define the band length b_l as

$$b_i = N\sigma_i + l_o \tag{5.2}$$

where N determines the confidence interval, the standard deviation is denoted by σ, and l_o is the obstacle length.

By finding the closest point to the obstacle on the lane splines (from the map) we can make an assumption in which lane the obstacle is driving. The band position will be calculated along the drive spline of the street for the given predicted travel distance as shown in Fig. 5.2. In the case of static obstacles the band length is the same as the obstacle length over the time. However, for dynamic obstacles the band becomes longer in each step as the probability density function for the obstacle's position become wider over time.

5.1.3 Obstacle Avoidance

For obstacle avoidance to be possible, the entire path from the parent node to the child node should be free of the predicted obstacle positions. As shown in Fig. 5.4, there are two possible conditions. The first condition is if child and parent nodes are at the same lane, and the second one is if the child is in the adjacent lane of the parent node. In the first condition, only the obstacles in the shared lane will be considered. The predicted obstacle band should

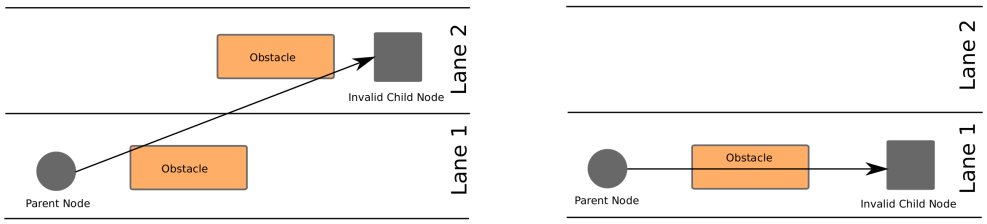


Fig. 5.4: Invalid child nodes: the circle is a parent node and the gray square is an invalid child node. The orange boxes are the obstacles/predicted position of obstacles. As shown in the left picture, a child node is in the adjacent lane, obstacles on both lanes must be checked. As shown in the right picture, both parent and child nodes are at a same lane, obstacles in the shared lane are checked. If an obstacle is between the parent and child nodes, then the child node is blocked.

Algorithm 11: Obstacle avoidance

input : child node n_c, parent node n_p, parent node velocity v_p, obstacle_band
output : validation of the child node
if *obstacles in child node lane* **then**
  $n'_p \leftarrow$ the projection of n_p on the child lane
  if $n'_p < obstacle\_band < n_c$ **then**
    return **False** ▷ `child_node is invalid.`
if *obstacles in parent node lane* **then**
  $s_{min} \leftarrow S_{min} + v_p$ ▷ S_{min} `is the minimum distance at zero velocity.`
  if $n_p < obstacle\_band < n_p + s_{min}$ **then**
    return **False** ▷ `child_node is invalid.`
return **True**

not come between the parent and the child nodes. In the second condition, the obstacles in the child node lane and the parent node lane must be checked. For the obstacles in the child node lane, the predicted obstacle band should not be between the child node and the projection of parent node (on the child node lane). A lane change cannot be performed if the obstacle ahead is very near to the parent node. Therefore, for the obstacles ahead and in the parent node lane, the obstacle band should not be in the specified distance s_{min} ahead of the parent node, where s_{min} is proportional to the parent node velocity. The obstacle avoidance algorithm is presented in Alg. 11.

In both conditions, it is obvious that if the child node of a lane with decreasing speed is blocked, the next two children of the same lane (with the same speed and accelerating speed) will be blocked as well. They must then be put in the closed list. Additionally, if the child node with the same speed of a lane is blocked, the next child of the same lane (with accelerating speed) should be blocked and they do not need to be checked again. In this method can bypass unnecessary calculations for obstacle avoidance.

5.1.4 Cost Function

At each iteration of the FU-A* algorithm, the free nodes in the open list are sorted based on minimizing a cost function which is defined as follows:

$$f(n) = g(n) + p(n) + h(n) \tag{5.3}$$

The cost function contains three terms. The first term ($g(n)$) is the travel time of reaching a node, which is defined by increasing the step from start point.
The second term $p(n)$ penalizes hazardous motions, such as going to the adjacent lane which costs k_1. Aborting a lane change maneuver and going back to the previous lane can cause other drivers to be confused while causing passenger discomfort. Therefore, if in the last trajectory the car decided to do a lane change in the first T seconds of trajectory, changing this decision is penalized by k_2, which means the planner shall not change its decision until a lane change saves more than k_2 seconds to reach the goal. Another discomfort action is unnecessary acceleration change, therefore acceleration change costs k_3, which means till acceleration change does not provide us more than k_3 seconds time saving, it will not be chosen. The acceleration change also effects the energy consumption, the energy consumption can be penalized in the cost function. The power model of iMiEV is provided as a lookup table in Appendix A. The energy consumption between parent and child nodes can be calculated using this table.
The third term ($h(n)$) is the distance to the goal point which leads to the preference of search solutions closer to the goal.

5.1.5 Reaching the Goal

The search algorithm must stop when the car reaches the goal point. The goal point is not necessarily an integer multiple of flexible units, therefore if the goal point is between parent and child nodes, the parent node will be chosen as the end node of the graph.
In the case of a blocked street, the search algorithm cannot reach the goal. Therefore, based on the obstacle distance, a "smooth brake" or "emergency brake" maneuver will be chosen as the desired trajectory.

5.2 Practical Issues

The FU-A* path gives us a sequence of set points which their distance are $d = VT$, being V the former speed of the car. Thus, the distance is proportional to the speed. The long distance between set points causes two issues:

- the car may not stay on the street lane;
- a big difference between the points of the resulting trajectory results in a large error for control input which causes uncomfortable steering or gas changes.

To deal with these issues, the gaps between the points of the solution trajectory are filled with sub-sampling points (for every meter) w.r.t. the drive lane spline or a predefined lane changing spline described in the following sub-section. If the parent and child nodes are at the same lane, then the drive lane spline is used for the sampling points, as shown in Fig. 5.5. Otherwise, the lane change spline is sampled as described below.

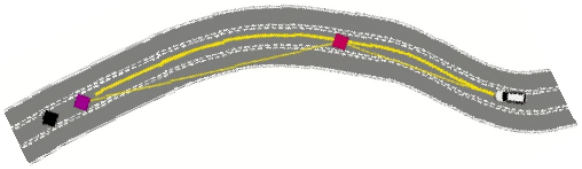


Fig. 5.5: Re-sampling along the road: the vehicle location and colored diamond markers illustrate the final nodes of FU-A*. The parent and child nodes are in the same lane, so the nodes can be connected in the trajectory with sub-sampling for every meter along the drive spline.

5.2.1 Predefined Lane Changing Spline

To have a smooth and convenient lane change, a cubic polynomial is defined between parent node and child node in the adjacent lane, as shown in Fig. 5.6. The time distance between the parent and child nodes is T seconds. This time should be practically sufficient for a lane change. There are four assumptions needed to find the parameters of a cubic polynomial:

- the first and end points of the spline are equal to the parent and child nodes' position,
- the first derivative of the start point and end point must be the same as the first derivative of the drive lane splines at the same positions.

In order to avoid set points jumping during a lane change, and to allow the car to follow the same trajectory until it finishes the lane change, it is important not to update the predefined

spline during a lane change maneuver until the corresponding child node stays at the same lane. If, however, car decides reverse its lane changing decision and go back to the previous lane, the new spline between the current position of the car and child node will be defined and sampled.

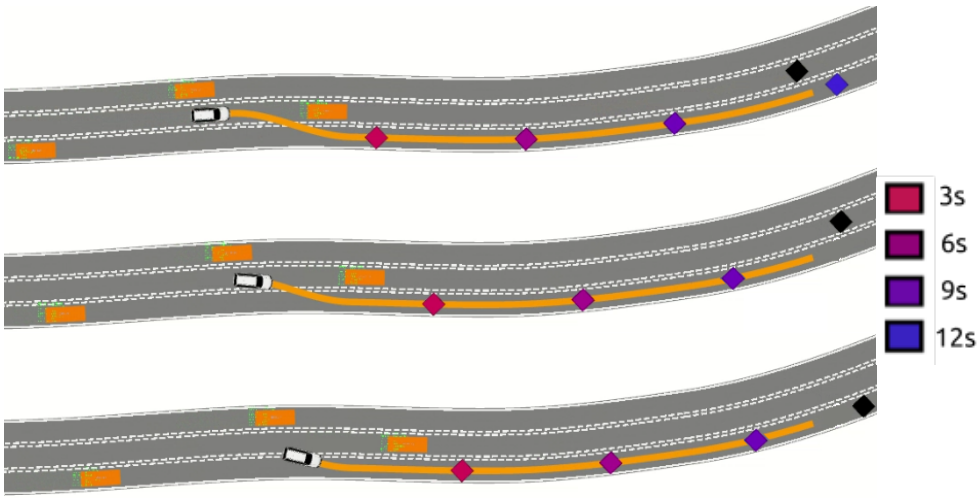


Fig. 5.6: Re-sampling along predefined cubic spline: the vehicle location and colored diamond markers illustrate the final nodes of FU-A* with 3s time interval. The black diamond shows the goal point. In the first picture, the first parent and child nodes are in the different lanes, and a cubic spline is defined between them. We connect the first two nodes in the trajectory with sub-sampling the cubic spline until the car reaches the adjacent lane. The sequence pictures illustrate the car position and behaviour overtime with 1 second time stamp.

5.3 Simulation Results

The proposed algorithm is validated using a comprehensive simulation study. Two common scenarios were simulated to show the safety and efficiency of the proposed algorithm. The algorithm and simulation were implemented using the ROS framework. The FU-A* trajectory planning for the autonomous car was ran at 20 Hz, and a path planner (following the lane road) for the other cars was run at 100 Hz. The simulated road map was the map of the former Tempelhof airport, Berlin. The FU-A* parameters used in simulation are described in table 5.1.

Fig. 5.7 illustrates a simple lane change maneuver. The sequence points of FU-A* are shown with diamond markers. The predicted obstacle distribution centers are shown with circle markers. The colors of the diamonds and circle markers for the same sample time are similar and are described in the legend. The color changes from pink to blue over the time. The black diamond shows the goal point. The color of the lane between the markers shows

Table 5.1: FU-A* parameters used in simulation: The time step between two nodes is T. Going to the adjacent lane costs k_1. Aborting a lane change maneuver costs k_2. Acceleration change costs k_3.

T (sec.)	k_1 (sec.)	k_2 (sec.)	k_3 (sec.)
3	3	10	1

speed of the action between nodes. The speed color changes from red to green when the velocity changes from 0 to 18 m/s.

In the second test the autonomous car merge to traffic speed as shown in Fig. 5.8. The autonomous car decreased speed from 10 m/s to 7 m/s to merge into traffic speed and to plan with the traffic speed. The sequence pictures shall illustrate the car position and behaviour overtime with 3 seconds timestamp.

In the third scenario, the car ahead braked instantaneously. The autonomous car therefore decreased its speed and then overtook from the left side while monitoring the car driving on the left lane Fig. 5.9. The sequence pictures illustrate the car position and behaviour over time with 1 second timestamp. Interested readers are encouraged to watch a video of the simulations at https://youtu.be/Lw_Mk37N6G0.

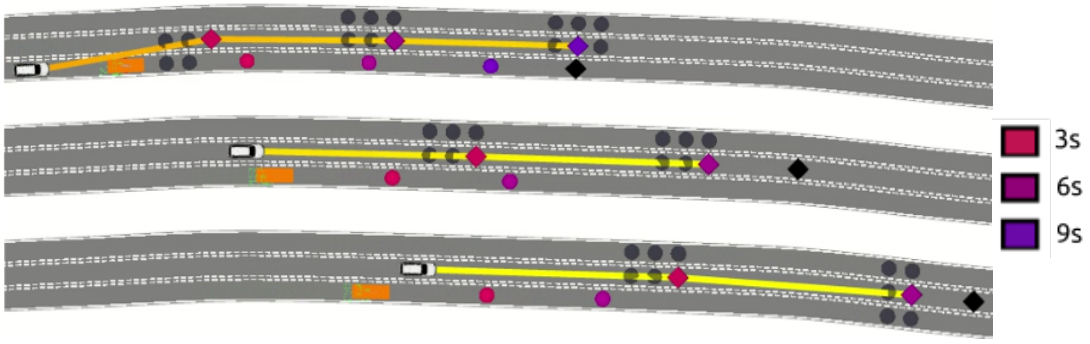


Fig. 5.7: Simple lane changing: the gray circles are the collision free nodes of the FU-A* open list. The final nodes of FU-A* are shown with diamond markers. The orange boxes are the obstacles. The predicted obstacles distribution centers are shown with colored circle markers. The colors of the diamond and circle markers for the same sample time are similar and are described in the legend. The sequence pictures illustrate the car position and behaviour overtime with a 3 seconds time stamp. The color of the lane between the diamond markers changes from orange to yellow, shows that the car increased its speed from 8 m/s to 9 m/s while it was changing lanes. The car then followed the middle lane wile increasing the speed from 9 m/s to 10 m/s.

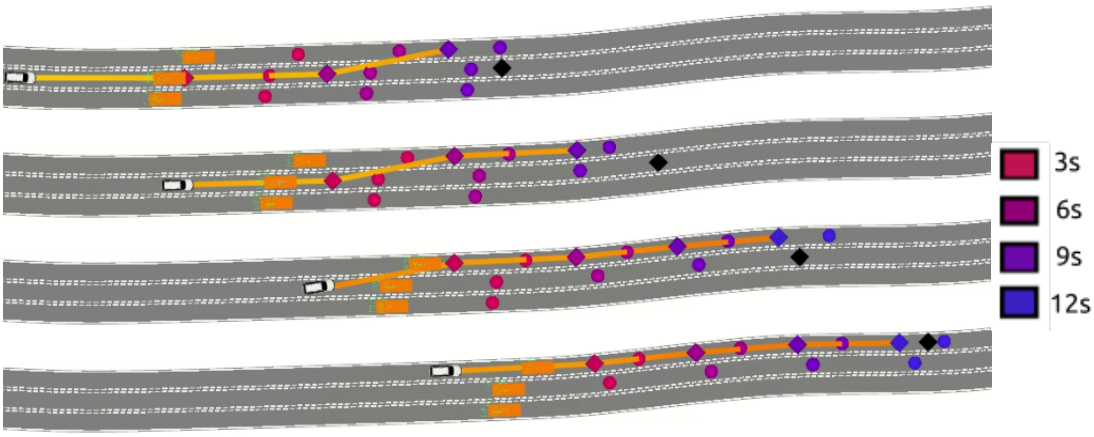


Fig. 5.8: Merge into traffic: the final nodes of FU-A* are shown with diamond markers. The orange boxes are the obstacles. The predicted obstacles distribution centers are shown with colored circle markers. The colors of the diamond and circle markers for the same sample time are similar and are described in the legend. The sequence pictures illustrate the car position and behaviour overtime with a 3 seconds time stamp. The color of the lane between the diamond markers changes from yellow to orange, shows that the autonomous car decreased its speed from 10 m/s to 7 m/s to merge into traffic.

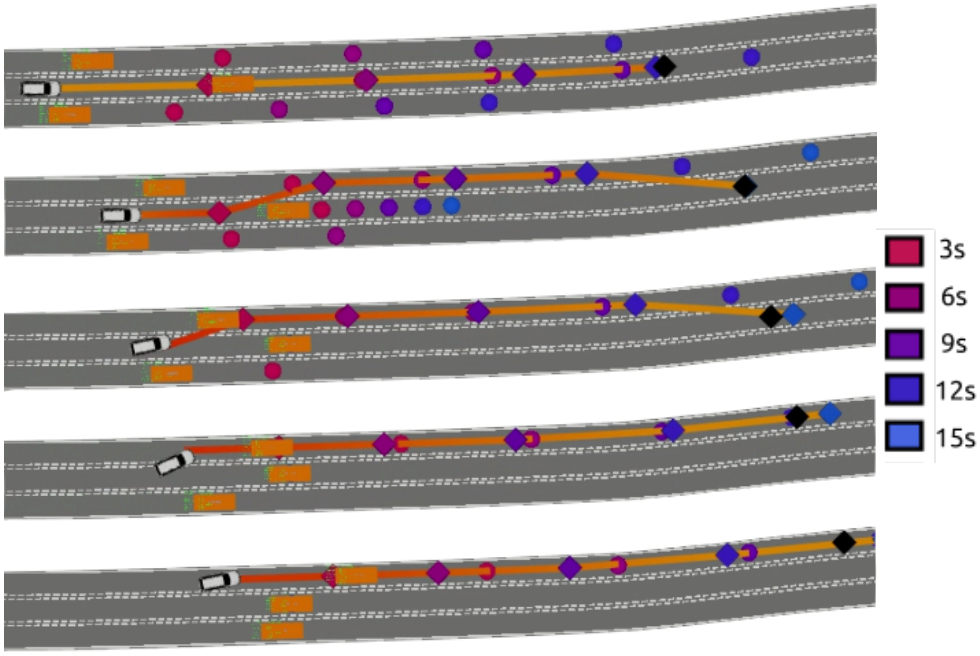


Fig. 5.9: Decrease the speed and overtake: the obstacle ahead (on the middle lane) braked instantaneously. The autonomous car therefore decreased its speed and then overtook from the left. The final nodes of FU-A* are represented by diamond markers. The orange boxes are obstacles. The predicted obstacles' distribution centers are shown with colored circle markers. The colors of the diamond and circle markers for the same sample time are similar and are described in the legend. The sequence pictures illustrate the car position and behaviour overtime with 1 second time stamp. The color of the lane between the markers changes from orange to red, shows that the autonomous car decreased its speed from 7 m/s to 6 m/s.

5.4 Conclusion

FU-A*, the proposed algorithm, is a new approach for trajectory planning in a structured urban area and considers both static and dynamic obstacles. Its output trajectory is locally optimized and feasible. Dynamic obstacles in the road maps are carefully considered by utilizing a predictive approach that takes into account the velocity of the obstacles and the spline of the road. Simulation results, in which the simulated autonomous car "MadeInGermany" is used on a road map of the former airport Tempelhof (Berlin, Germany), revealed the validity and reliability of the proposed algorithm.

Chapter 6

Summary and Outlook

Trajectory planning is one of the essential components of autonomous cars. In this thesis the car model kinematic and dynamic are investigated to provide a proper controller and trajectories to deal with the models constraints. Three local trajectories for autonomous car are proposed (reactive trajectory, vector field, and FU-A* trajectory). In the first two trajectories, the local planner is focused to follow a lane or the global planner reference lane, while the lane changing command is made in another layer of decision making. The FU-A* trajectory instead considers all possible lanes from global planner, so the lane changing happens locally based on the cost function.

In chapter two, the experimental setups (i-MiEV as an electric car and MIG as a petrol car) are explained. The hardware setup, containing the different sensors and actuators, are described in detail. The sensory data are used to provide the environment perception such as the car localization, the obstacle detection. The software structure is divided into perception, condition monitoring, planning, execution layers, from which the third layer planning is the main focus of the thesis. The kinematic of the car is modeled as the bicycle model; therefore, the pure pursuit lateral controller is implemented to cope with the nonholonomic constraints of the model. The dynamic model of the car - including the traction force and aerodynamic drag force -is also presented. The throttle and brake model of both cars are provided to map the necessary force to throttle or brake command. Finally, the acceleration constraints at each speed is specified, and are then used to plan a feasible trajectory in chapter 3. The i-MiEV throttle model depends on the battery power, the PMSM motor model, the car weight, and the air and roll frictions. Using electric car throttle model, the maximum acceleration and speed for a feasible trajectory can easily be defined. Conversely, engaging the gearbox of a petrol car makes the throttle model more complicated. The brake model also specifies the maximum deceleration respect to the brake command. Finally, by gathering the data from a

human driver, the maximum jerk is determined. Although, the maximum constraints for jerk, acceleration, speed can change based on passenger preferences.

Chapter three provides the reactive trajectory which reacts to the traffic rules, the road curvature, and the lateral distance of the car to the road to specify the speed limits in each point. Then, a new smoothing method is proposed and considered the car dynamic model (limiting the acceleration based on the speed) and constant jerk. By considering the jerk constant, the trajectory becomes a piece-wise double S-trajectory which smoothly decelerates and accelerates to provide a comfortable ride for passengers. The obstacles on the road also considered and limits the speed of each point based on the desired longitudinal distance and swerves the path from its reference to provide the desired lateral distance. The lateral distance to the obstacle also affects the maximum speed of the points if the car wants to pass the obstacle within a threshold. The proposed method is validated with several simulations.

In chapter four, the local trajectory and controller method are combined to provide the force vector field which computed offline for a specified map and speed. The vector field contains the attractive forces which pull the car along a map lane. The vector field is stored at discrete locations, thus the final attractive force is generated by the interpolation of four forces around the car position. The force vector also depends on the car speed. Therefore, two different force vectors for the minimum and maximum speed are calculated. Then, the attractive force is interpolated in the speed domain based on the desired speed. The repulsive force is calculated online to avoid collision with obstacles. The desired steering angle is calculated based on the car's position and orientation respect to the force vector. By switching the vector field to a different lane, the car will change lanes. The proposed trajectory is validated on a model car.

In chapter five, an optimal local trajectory planning is proposed. The well-known A* algorithm is modified to have the flexible grids in x, y dimensions while considering the constant time interval as a third dimension. Each parent node has up to nine neighbor nodes across in three lanes with negative, zero, and positive acceleration. The neighbor nodes are limited inside the street lane area, meaning that the number of the nodes does not increase exponentially like the traditional A*. The problem is therefore solved faster. The obstacles positions are predicted with the assumption that they will follow the road with a constant speed and with considering the normal process noise; accordingly, the equipped area with the predicted obstacle position becomes longer over time. The neighbor node which is blocked by the predicted obstacle function is omitted from the search tree, compelling the car to stay behind the obstacle or change the lane based on the cost function. The cost function has three different terms: the travel time, hazardous motion - which cause passenger discomfort and increases energy consumption-, and distance to the goal. Each term of the cost function

gets different weights which can be tuned based on the passengers' preferences. Finally, the gap between the final nodes are filled with sub-sampling the drive spline of a lane or a cubic spline generated between two nodes from two different lanes. The proposed method is validated with several simulations.

outlook

To the future work :

- **Set the constraints automatically** : the constraints like maximum acceleration and jerk should be set based on the passengers' preferences, by can be set automatically using gathering the personal drivers.
- **Set the cost function weights automatically**: based on the traffic data, and passengers' preferences the weights of the cost function terms of FU-A* can be set automatically.
- **Take the pedestrians and other drivers preferences into account**: I considered only the preferences of the passengers inside the car and assumed that following the traffic rules will fulfill other human desires. However, the program can also consider the preferences of other drivers and pedestrians (e.g., seniors or children) based on the urban area data, and traffic time tables.

References

[1] Tonoy Chowdhur Atul Garg, Rezawana Islam Linda. Evolution of aircraft flight control system and fly-by-light flight control system. *International Journal of Emerging Technology and Advanced Engineering*, 3, 2013.

[2] Nils J Nilsson. *Principles of artificial intelligence*. Morgan Kaufmann, 2014.

[3] Z. Boroujeni, D. Goehring, F. Ulbrich, D. Neumann, and R. Rojas. Flexible unit a-star trajectory planning for autonomous vehicles on structured road maps. In *2017 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, pages 7–12, June 2017.

[4] Z. Boroujeni, M. Mohammadi, D. Neumann, D. Goehring, and R. Rojas. Autonomous car navigation using vector fields. *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 7–12, April 2018.

[5] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Path planning for autonomous vehicles in unknown semi-structured environments. *I. J. Robotics Res.*, 29(5):485–501, 2010.

[6] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Practical search techniques in path planning for autonomous driving. In *Proceedings of the First International Symposium on Search Techniques in Artificial Intelligence and Robotics (STAIR-08)*, Chicago, USA, June 2008. AAAI.

[7] Steven M. LaValle. *Planning Algorithms (Section 15.3.2)*. Cambridge University Press, USA, 2006.

[8] S. Koenig and M. Likhachev. Improved fast replanning for robot navigation in unknown terrain. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, volume 1, pages 968–975 vol.1, 2002.

[9] S. Karaman and E. Frazzoli. Sampling-based optimal motion planning for non-holonomic dynamical systems. In *2013 IEEE International Conference on Robotics and Automation*, pages 5041–5047, May 2013.

[10] Moritz Werling, Sören Kammel, Julius Ziegler, and Lutz Groell. Optimal trajectories for time-critical street scenarios using discretized terminal manifolds. *International Journal of Robotic Research - IJRR*, 31:346–359, 03 2012.

[11] M. McNaughton, C. Urmson, J. M. Dolan, and J. Lee. Motion planning for autonomous driving with a conformal spatiotemporal lattice. In *2011 IEEE International Conference on Robotics and Automation*, pages 4889–4895, May 2011.

[12] M. Rufli and R. Siegwart. On the design of deformable input- / state-lattice graphs. In *2010 IEEE International Conference on Robotics and Automation*, pages 3071–3077, May 2010.

[13] Wenda Xu, Junqing Wei, J. M. Dolan, Huijing Zhao, and Hongbin Zha. A real-time motion planner with trajectory optimization for autonomous vehicles. In *2012 IEEE International Conference on Robotics and Automation*, pages 2061–2067, May 2012.

[14] Ronald C. Arkin. Integrating behavioral, perceptual, and world knowledge in reactive navigation. *Robotics and Autonomous Systems*, 6(1):105 – 122, 1990. Designing Autonomous Agents.

[15] F. Bounini, D. Gingras, H. Pollart, and D. Gruyer. Modified artificial potential field method for online path planning applications. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 180–185, June 2017.

[16] Oscar Montiel, Ulises Orozco-Rosas, and Roberto Sepúlveda. Path planning for mobile robots using bacterial potential field for avoiding static and dynamic obstacles. *Expert Systems with Applications*, 42(12):5177 – 5191, 2015.

[17] Derek R Nelson, D Blake Barber, Timothy W McLain, and Randal W Beard. Vector field path following for miniature air vehicles. *IEEE Transactions on Robotics*, 23(3):519–529, 2007.

[18] Nicoletta Bloise, Elisa Capello, Matteo Dentis, and Elisabetta Punta. Obstacle avoidance with potential field applied to a rendezvous maneuver. *Applied Sciences*, 7:1042, 10 2017.

[19] Y. Rasekhipour, A. Khajepour, S. K. Chen, and B. Litkouhi. A potential field-based model predictive path-planning controller for autonomous road vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 18(5):1255–1267, May 2017.

[20] Danilo Alves De Lima and Guilherme Augusto Silva Pereira. Navigation of an autonomous car using vector fields and the dynamic window approach. *Journal of Control, Automation and Electrical Systems*, 24(1-2):106–116, 2013.

[21] E. Galceran, R. M. Eustice, and E. Olson. Toward integrated motion planning and control using potential fields and torque-based steering actuation for autonomous driving. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 304–309, June 2015.

[22] J. Ji, A. Khajepour, W. W. Melek, and Y. Huang. Path planning and tracking for vehicle collision avoidance based on model predictive control with multiconstraints. *IEEE Transactions on Vehicular Technology*, 66(2):952–964, Feb 2017.

[23] Y. Zhang, H. Chen, S. L. Waslander, T. Yang, S. Zhang, G. Xiong, and K. Liu. Speed planning for autonomous driving via convex optimization. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 1089–1094, Nov 2018.

[24] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization (Chapter 11)*. Cambridge University Press, USA, 2004.

[25] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2019.

[26] Thomas Lipp and Stephen Boyd. Minimum-time speed optimisation over a fixed path. *International Journal of Control*, 87(6):1297–1311, 2014.

[27] Y. Meng, Y. Wu, Q. Gu, and L. Liu. A decoupled trajectory planning framework based on the integration of lattice searching and convex optimization. *IEEE Access*, 7:130530–130551, 2019.

[28] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019.

[29] S. Heinrich, A. Zoufahl, and R. Rojas. Real-time trajectory optimization under motion uncertainty using a gpu. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3572–3577, Sep. 2015.

[30] John Horst and Anthony Barbera. Trajectory generation for an on-road autonomous vehicle - art. no. 62302j. *Proc SPIE*, pages 82–, 06 2006.

[31] T. Kroger, A. Tomiczek, and F. M. Wahl. Towards on-line trajectory computation. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 736–741, Oct 2006.

[32] R. Haschke, E. Weitnauer, and H. Ritter. On-line planning of time-optimal, jerk-limited trajectories. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3248–3253, Sep. 2008.

[33] J. Lin, N. Somani, B. Hu, M. Rickert, and A. Knoll. An efficient and time-optimal trajectory generation approach for waypoints under kinematic constraints and error bounds. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5869–5876, Oct 2018.

[34] P. Resende and F. Nashashibi. Real-time dynamic trajectory planning for highly automated driving in highways. In *13th International IEEE Conference on Intelligent Transportation Systems*, pages 653–658, Sept 2010.

[35] S. Glaser, B. Vanholme, S. Mammar, D. Gruyer, and L. Nouveliere. Maneuver-based trajectory planning for highly autonomous vehicles on real road with traffic and driver interaction. *IEEE Transactions on Intelligent Transportation Systems*, 11(3):589–606, Sept 2010.

[36] M. Jalalmaab, B. Fidan, S. Jeon, and P. Falcone. Model predictive path planning with time-varying safety constraints for highway autonomous driving. In *Advanced Robotics (ICAR), 2015 International Conference on*, pages 213–217, July 2015.

[37] F. Ulbrich, D. Goehring, T. Langner, Z. Boroujeni, and R. Rojas. Stable timed elastic bands with loose ends. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 186–192, June 2017.

[38] C. Rösmann, F. Hoffmann, and T. Bertram. Timed-elastic-bands for time-optimal point-to-point nonlinear model predictive control. In *Control Conference (ECC), 2015 European*, pages 3352–3357, July 2015.

[39] Frank Havlak and Mark E. Campbell. Discrete and continuous, probabilistic anticipation for autonomous robots in urban environments. *CoRR*, abs/1309.0766, 2013.

[40] D. Ferguson, M. Darms, C. Urmson, and S. Kolski. Detection, prediction, and avoidance of dynamic obstacles in urban environments. In *2008 IEEE Intelligent Vehicles Symposium*, pages 1149–1154, June 2008.

[41] C. Guo, C. Sentouh, B. Soualmi, J. B. Haué, and J. C. Popieul. Adaptive vehicle longitudinal trajectory prediction for automated highway driving. In *2016 IEEE Intelligent Vehicles Symposium (IV)*, pages 1279–1284, June 2016.

[42] S. Dominguez, A. Ali, G. Garcia, and P. Martinet. Comparison of lateral controllers for autonomous vehicle: Experimental results. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1418–1423, Nov 2016.

[43] Jarrod Snider. Automatic steering methods for autonomous automobile path tracking. Master thesis, Robotics Institute Carnegie Mellon University, 2011.

[44] Pan Zhao, Jiajia Chen, Yan Song, Xiang Tao, Tiejuan Xu, and Tao Mei. Design of a control system for an autonomous vehicle based on adaptive-pid. *International Journal of Advanced Robotic Systems*, 9(2):44, 2012.

[45] Pos lv,position and orientation system, utilizing integrated inertial technology t for land-based vehicle applications. https://www.applanix.com/products/poslv.htm.

[46] Ibeo, lidar sensors. https://www.ibeo-as.com/ibeoreference/reference-sensor-system/.

[47] Velodyne, lidar sensor. https://velodynelidar.com/.

[48] Paravan. https://www.paravan.de/startseite/.

[49] R. Spangenberg, D. Goehring, and R. Rojas. Pole-based localization for autonomous vehicles in urban scenarios. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2161–2166, Oct 2016.

[50] Klee project. https://www.elektronikforschung.de/projekte/klee.

[51] T. Langner, D. Seifert, B. Fischer, D. Goehring, T. Ganjineh, and R. Rojas. Traffic awareness driver assistance based on stereovision, eye-tracking, and head-up display. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3167–3173, May 2016.

[52] Klee project proposal. https://www.mi.fu-berlin.de/inf/groups/ag-ki/Projects/Abgeschlossene_Drittmittelprojekte/KLEE/index.html.

[53] Florent Altché, Philip Polack, and Arnaud de La Fortelle. A simple dynamic model for aggressive, near-limits trajectory planning. In *Intelligent Vehicles Symposium (IV), 2017 IEEE*, pages 141–147. IEEE, 2017.

[54] Philip Polack, Florent Altché, Brigitte d'Andréa Novel, and Arnaud de La Fortelle. The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles? In *Intelligent Vehicles Symposium (IV), 2017 IEEE*, pages 812–818. IEEE, 2017.

[55] The mitsubishi imiev, an electric mini-car. https://www.nrel.gov/docs/fy12osti/48528.pdf.

[56] A. Rassõlkin, T. Vaimann, A. Kallaste, and R. Sell. Propulsion motor drive topology selection for further development of iseauto self-driving car. In *2018 IEEE 59th International Scientific Conference on Power and Electrical Engineering of Riga Technical University (RTUCON)*, pages 1–5, Nov 2018.

[57] Volkswagen-Media-Services. Dual-clutch gearbox (dsg). https://www.volkswagen-newsroom.com/en/dual-clutch-gearbox-dsg-3651, 2019.

[58] M. Elbanhawi, M. Simic, and R. Jazar. In the passenger seat: Investigating ride comfort measures in autonomous cars. *IEEE Intelligent Transportation Systems Magazine*, 7(3):4–17, Fall 2015.

[59] M. Kuderer, S. Gulati, and W. Burgard. Learning driving styles for autonomous vehicles from demonstration. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2641–2646, May 2015.

[60] Defense Advanced Research Projects Agency. *route Network definition file (rndf) and mission data file (mdf) formats*, 2007. Urban Challenge.

[61] Paul Czerwionka. A three dimensional map format for autonomous vehicles. Master dissertation, Freie University of Berlin, 2014.

[62] Hiroshi Akima. A new method of interpolation and smooth curve fitting based on local procedures. *J. ACM*, 17(4):589–602, October 1970.

[63] Sujeet Chand and Keith L Doty. On-line polynomial trajectories for robot manipulators. *The International Journal of Robotics Research*, 4(2):38–48, 1985.

[64] Monte Andre Dickson, Bingcheng Ni, Shufeng Han, and John F Reid. Trajectory path planner for a vision guidance system, May 7 2002. US Patent 6,385,515.

[65] Mohammad Mahdi Emami and Behrooz Arezoo. A look-ahead command generator with control over trajectory and chord error for nurbs curve with unknown arc length. *Computer-Aided Design*, 42(7):625–632, 2010.

[66] Bing Liu. Route finding by using knowledge about the road network. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 27(4):436–448, July 1997.

[67] Ron Goldman. Curvature formulas for implicit curves and surfaces. *Computer Aided Geometric Design*, 22(7):632–658, 2005.

[68] Daniel Göhring. Controller architecture for the autonomous cars: Madeingermany and e-instein. Technical report, AutoNOMOS-Labs, Freie Universität Berlin, Germany, 2012.

[69] Abhijeet Ravankar, Ankit A. Ravankar, Yukinori Kobayashi, Yohei Hoshino, and Chao-Chung Peng. Path smoothing techniques in robot navigation: State-of-the-art, current and future challenges. In *Sensors*, 2018.

[70] N. Roy, P. Newman, and S. Srinivasa. *Time-Optimal Trajectory Generation for Path Following with Bounded Acceleration and Velocity*, pages 209–216. MITP, 2013.

[71] X. Li, Z. Sun, D. Cao, Z. He, and Q. Zhu. Real-time trajectory planning for autonomous urban driving: Framework, algorithms, and verifications. *IEEE/ASME Transactions on Mechatronics*, 21(2):740–753, April 2016.

[72] Luigi Biagiotti and Claudio Melchiorri. *Trajectory Planning for Automatic Machines and Robots*. Springer Publishing Company, Incorporated, 1st edition, 2008.

[73] Waltenegus Dargie and Christian Poellabauer. *Fundamentals of Wireless Sensor Networks: Theory and Practice*. Wiley Publishing, 2010.

[74] Zahra Boroujeni. Navigation package. https://github.com/AutoModelCar/catkin_ws_user/tree/master/src/fub_navigation, Feb 2018.

[75] Songrit Maneewongvatana and David M. Mount. On the efficiency of nearest neighbor searching with data clustered in lower dimensions. In Vassil N. Alexandrov, Jack J. Dongarra, Benjoe A. Juliano, René S. Renner, and C. J. Kenneth Tan, editors, *Computational Science — ICCS 2001*, pages 842–851, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

Appendix A

Trajectory Energy Consumption

Another parameter that can be considered in the optimal trajectory planning is energy consumption. This parameter can be added in the cost function with a tuning parameter to control the priority of energy consumption regarding the other parameters like safety, time, and comfort. For example, it is more efficient to release both the gas and brake pedals before reaching a traffic light. This allows the car to more gradually decelerate, thereby saving energy rather than accelerating (push the gas pedal) and then quickly using the brake pedal stop behind the traffic light. Although the car reaches the traffic light sooner, it uses more energy and must spend some time waiting at the traffic light regardless. Thus, the first scenario allows the car to save energy while starts to move at the same time (when the traffic light turns to green). Decreasing the acceleration at higher speed is also another term that causes to save energy. As shown below, the power consumption for an acceleration value is almost four times bigger than the power saving with the same deceleration value (with a negative sign). Avoiding the acceleration change not only helps passengers feel more comfortable (less jerk) but also saves energy.A charge and discharge model of an electric car (i-MiEV) is investigated to quantify this measure. The model is presented as a lookup table contains the power (W) (the battery voltage (V) multiply the current (A), the velocity (m/s), and acceleration (m/s^2) at each point. Average velocities and accelerations between two sequential trajectory points are calculated when finding the related power (W) - recorded in the lookup table. Energy consumption (J) is the power (W) times the difference time (s) between two sequential trajectory points. Energy consumption graphs over time can help human drivers also to train themselves to drive more efficiently. Therefore, an echo coach is also designed to show how the human driver drives compare to the optimal planned trajectory. By comparing the current consumed energy with the proposed energy in time, the human driver can adjust its pressure on gas/brake pedals. For example, when the consumed energy

is more than the proposed one, the driver should decrease the acceleration, or when it is less than the suggested energy should increase the gas pedal pressure.

A.1 Electric Car Power Model

The electric motor can work as an engine or a generator and discharge or charge the battery of the electric car. When the car speed is reducing, the wheels turns the motor, so the motor works like a generator and produces electricity and charges the battery. The ratio of the charging depends on the battery charge system.The purpose of finding the power model is to find how much energy the car consumes to accelerate at a certain velocity or keep a certain velocity; on the other hand, if the car decelerates at a certain velocity, how much the battery will charge. For each car, the motor, the car weight, the battery, the systems of charging are the main parameters that can affect its dis/charge model. There are also outside parameters like the street slop, the velocity wind which affect the car energy consumption.

By reading the battery current and voltage decoded in OBD-II data, the current power (W)

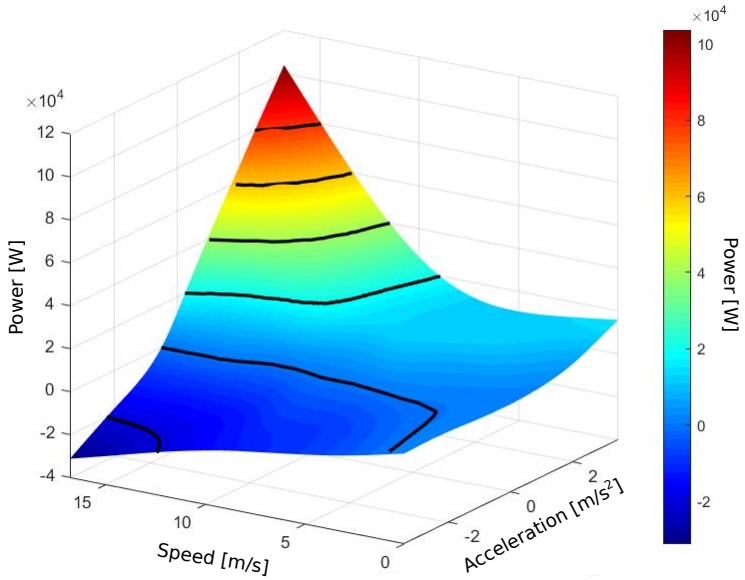


Fig. A.1: i-MiEV power model: The black line separates the power regions from -20kW to 80Kw into 20kW intervals. The colors represent the power, as shown in the right side color bar. The darkest blue represents the lowest negative power (charging mode), and the darkest red represents the biggest positive power (discharge mode).

of the car is measured. The maximum voltage of the battery is around 350 Volt. When the car uses the power, the current is positive, and while the battery charges, the current is negative.

The current of iMiEV while the car standing is roughly 2 Ampere. Using the maximum cooling system in Autumn (outside temperature 10 °) increases the current to 5 Ampere. Also, the maximum heating system in the same outside temperature pushes the current to 20 Ampere. For this model, the i-MiEV data is gathered in flat test fields and without any accessories (cooling or heating systems, ...). The Applanix data is used to measure the car velocity and acceleration. Although the OBD-II provides the velocity data, the velocity coded in the car CAN-BUS is (around 6%) bigger than real velocity. Accordingly, humans can see 6% increased velocity on the dashboard velocity indicator for their safety. Fig. A.1 shows the recorded data and the surface which is fitted to the data. A lookup table is made based on the recorded data to calculate the trajectory energy consumption.

A.2 Eco Coach

We designed an Eco-coach to guide the human driver in more efficient. The proposed energy consumption for T seconds intervals are calculated using the provided power look-up table and a proposed trajectory. Reading the current power from OBD-II over time gives the real energy consumption. These two values are compared to each other in Eco-coach.
The past and current (expected and real) energy consumption, as well as the proposed future

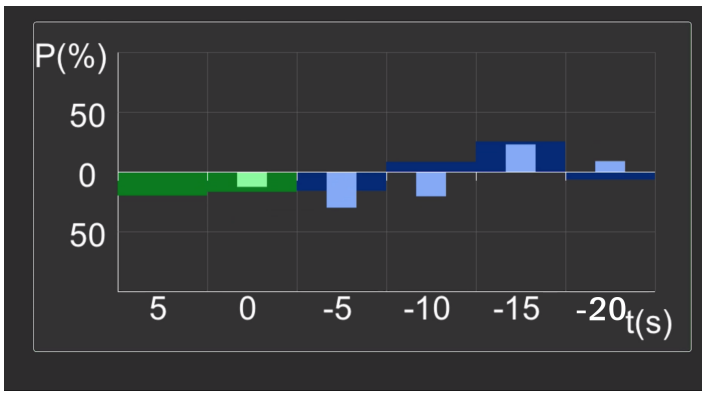


Fig. A.2: Eco-coach: each bar of the graph shows T (5 s) time intervals. The expected energy is shown as a thin bar inside the wider bar of real consumed energy. The four rightest bars belong to the past and show the last 4T (20 s). Then, the fifth bar from right shows the current energy consumption., and the sixth bar is the future bar between 5 and 10 s. By comparing the last two green bars, the driver can predict how to act in the future (release/push the pedals).

energy consumption, are shown in the graph of the Eco-coach in Fig A.2. Each bar of the graph shows the T seconds (e.g., 5 s) interval. The expected energy is shown as a thin bar inside the wider bar of real consumed energy. The four bars furthest to the right belong to the past and show the last 4T seconds. The fifth bar from right shows the current energy

consumption. In this bar, the last moment (dt) energies (expected and consumed) multiplied a coefficient T/dt is shown to keep the bar time interval constant. Finally, the proposed energy consumption for T time ahead is calculated from the last trajectory.
For example, in the last 5 seconds, the car expected to save more energy than the human driver saved. In the current moment, the expected and real energies are the same; therefore, the driver acts correctly. If the thinner bar was lower than the wider one, the driver should push the gas pedal less or push the brake pedal. On the other hand, if the thinner bar of the current moment is above the wider one, the driver should release the brake more or push the gas pedal more. By comparing the last two green bars, the driver can predict how to act in the future (release/push the pedals). After reaching the goal, the total expected energy is compared with the consumed energy, and the driving style is categorized into efficient and inefficient driving.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Zahra Boroujeni
2020

Acknowledgements

I wish to thank all the member of the Autonomous Cars group in the Freie Universität Berlin for their great support. Prof. Dr. Raúl Rojas was always supportive and patient, and he has always illuminated my way. I was very lucky to have such a wise supervisor. Apart from my supervisor, my sincere thanks go to Prof. Dr. Daniel Göhring, Fritz Ulbrich and Daniel Neumann for their insightful suggestions and hours on-road tests. I would also like to express my gratitude to Mostafa Mohammadi, Tobias Langner, Stephan Sundermann, Ricardo Carrillo, and Khaled Alomari for helping me in proofreading and the valuable comments. Finally, I would like to thank my family for their persistent encouragement and support. I am grateful to my spouse for all his support; he has always been there for me whenever I needed.