Cone Slalom With Automated Sports Car – Trajectory Planning Algorithm

Hanzálek, Zdeněk and Záhora, Jiří and Sojka, Michal

DOI: https://doi.org/10.1109/TVT.2023.3309554
Cite as: Z. Hanzálek, J. Záhora, and M. Sojka. Cone slalom with automated sports car–trajectory planning algorithm. *IEEE Transactions on Vehicular Technology*, 73(1):362–374, 2024

Cone Slalom with Automated Sports Car – Trajectory Planning Algorithm

Zdeněk Hanzálek , Jiří Záhora , Michal Sojka

***Abstract*—In this paper, we present the system architecture and algorithms of an automated vehicle to perform a slalom. We demonstrate a novel trajectory planning algorithm based on optimization techniques using logic-based Benders decomposition, where an external loop optimizes the position of the nearest waypoint and an internal loop generates the optimal trajectory. The positions of the cones in this use case are unknown, but a mono camera and LiDAR detect them. They can be in a line or dispersed, have equal or unequal spacing, and the U-turns can be symmetric or asymmetric. A bicycle model is used to formulate a non-linear quadratic optimization problem aimed at optimal trajectory generation considering vehicle kinematics. Finally, the trajectory tracking control keeps the vehicle on the planned slalom trajectory while driving. The control system is interfaced with the vehicle via CAN and FlexRay buses. Much of the work was devoted to experiments with a real vehicle and fine-tuning the system parameters. During the validation of the system, interesting observations were made regarding the components' precision, frequency, and sensitivity.**

Keywords: **automated car, slalom, cone detection, robotic operating system, trajectory planning, temporal properties**

I. Introduction

[1] The automotive industry is increasingly relying on the automated functions of cars. Fully self-driving vehicles will serve us in the future, but new automated functions are already helping drivers in their daily lives. While many people appreciate driver assistance on the road, there are cases where automation is also essential, but for a smaller audience. One of these cases is repetitive car testing, which is a motivation behind the paper in hand. Testing new vehicle prototypes [1] requires many test drives to be conducted by professional drivers, especially for premium vehicles known for their excellent performance under extreme driving conditions. Automating this kind of testing can lead to the easier development of advanced control algorithms thanks to the test drives' reproducibility. Compared to highway tests, a cone slalom is an appropriate use case for university R&D teams since it does not require a large experimental space, keeps developers reasonably safe, and stresses system components in a variety of ways.

This paper describes an entire automated vehicle platform for software prototyping and demonstration. The platform is based on a series-produced car, add-on sensors, and a control system. We designed an innovative trajectory planning algorithm based on optimization techniques and validated it by maneuvering around a slalom course built of traffic cones and performing U-turns at its ends. The cone positions are unknown; they may be in-line or dispersed, equally or unequally spaced, and the U-turns may be symmetric or asymmetric. The aim is to close the perception/planning/control loop and demonstrate its behavior.

The base of the system is the Porsche Panamera car (see Fig. 1). The car is instrumented so that our control system communicates with the car via CAN and FlexRay buses [2]. We send steering and speed commands to the actuation-related Electronic Control Units (ECUs) and read signals from the internal car sensors, such as the accelerometer. In addition to the internal sensors, we equipped the car with an external monocular camera, LiDAR, and a differential GPS (DGPS). The computations are carried out on an NVIDIA TX2 embedded computer. For more details, see our illustrative videos [2].

In summary, in this paper, we describe a universal, and lightweight control system. The particular contributions of the paper are:

- We propose an efficient trajectory planner for the slalom drive, including the U-turns. Based on an optimization algorithm, the planner minimizes the drive time, respects the vehicle kinematic constraints, ensures smooth actuation, and can re-plan the trajectory from a vehicle state. We evaluated the planner on different configurations of cones in both simulation and practical experiments.
- We describe a holistic control system integrated within the car infrastructure and fine-tuned in outdoor experiments on a complex case study. The control system is based on a commercially available HW, well-established ROS middleware, standard libraries, and software components described in this paper.

The paper is structured as follows: After reviewing the related works, we describe the overall goal, concept, and architecture in Section II. Section III introduces a kinematic model used for vehicle motion planning. The paper's main contributions are the slalom trajectory planning algorithm, described in Section IV, and the optimization

Z. Hanzálek is with the Industrial Informatics Research Center, Czech Institute of Informatics, Robotics and Cybernetics, Czech Technical University in Prague, Prague 16000, Czech Republic (e-mail: Zdenek.Hanzalek@cvut.cz)

J. Záhora is with the Industrial Informatics Research Center, Czech Institute of Informatics, Robotics and Cybernetics, Czech Technical University in Prague, Prague 16000, Czech Republic (e-mail: Jiri.Zahora@cvut.cz)

M. Sojka is with the Industrial Informatics Research Center, Czech Institute of Informatics, Robotics and Cybernetics, Czech Technical University in Prague, Prague 16000, Czech Republic (e-mail: Michal.Sojka@cvut.cz)

[2]https://youtu.be/692lVNWT-MA, https://youtu.be/3V1lD2Hof4Q

Fig. 1: Automated car slalom

algorithm, described in Section V. The algorithms are evaluated by simulation in Section VI and the results of the experimental drives in the outdoor environment are analyzed in Section VII. Finally, the outcomes of this paper are discussed in Section VIII.

A. Related work

In this section, we start with generic automated driving papers and follow with works related to the main contribution of this paper – trajectory planning.

1) Automated driving: The automotive community distinguishes six levels of automation [3] from no automation (level 0) to a fully self-driving vehicle (level 5). Nowadays, most of the available cars offer assistance technologies in level 2, such as adaptive cruise control or a lane-keeping assistant system. These are deeply investigated in many papers [4], [5], [6], [7], [8]. Zhang et al. [9] designed a personalized motion planning and tracking control framework to prevent collisions between autonomous vehicles and the obstacles ahead. They devised a multi-constrained numerical optimization method used to avoid obstacles according to the specific needs of the passengers. Li et al. [10] proposed an interesting semantic-level maneuver sampling and trajectory planning algorithm to solve this high-dimensional motion planning problem.

There are commercial vehicles with advanced automated functions up to fully self-driving cars, but only for different use cases. Some examples are Tesla or Audi with the recently presented automatic parking or traffic jam autopilot [11]. Our cone slalom use case falls between levels 2 and 3 and can also be considered “well-defined”, but it is not present in production cars.

The control systems of all automated cars are distributed systems with a similar architecture [12], [13]; our automated car is not an exception.

As experimenting with real vehicles is quite time-consuming and resource-demanding, many automated driving applications are tested in virtual simulations [14] or with the use of available datasets [15], but the ability to experiment with a real car in a real environment opens a much wider space of experiences that one can gain during the development. This paper focuses on physical experiments, and simulations are used only for testing and developing individual components.

A related technology is vehicular platooning, in which vehicles automatically follow each other with small inter-vehicle spacing. Guo and Wang [16] dealt with a two-layered control architecture, based on a speed planner (calculates the expected platoon speed profile in a fuel–time-optimal manner) at the top layer and a speed tracking controller (follows the planned speed with guaranteed vehicle stability and platoon string stability) at the lower layer. Herman et al. [17] investigated how the H-infinity norm and the steady-state gain of bidirectional platoons scale with the number of vehicles. Distributed trajectory optimization for the vehicular platoon is considered in [18], where a quadratic spacing policy is used to improve the flexibility of the speed planning and control. Furthermore, in [19], Guo et. al, solve the speed optimization problem using distributed convex optimization based on spacing error minimization. Compared to platooning, in our paper dealing with the slalom use case, we concentrate more on lateral movements rather than longitudinal ones.

Autonomous/automated driving is a topic not only for traditional passenger cars, but also for racing events. For example, Roborace[3] is an attractive competition for professional teams. In the academic world, F1/10 competition [20], [21], [22] is gaining in popularity, especially when overtaking maneuvers are considered [23]. Our students regularly participate in these competitions while winning medals and learning the concepts applicable to real cars.

The vehicles used in other competitions are also an inspiration to us. For example, the one developed at ETH Zurich for racing in Formula Student competitions [24]. Another source of inspiration was Autoware.AUTO [25], the open-source software for autonomous driving based on ROS, tools for the control system design [26], LiDAR simulation [27], localization systems [28] as well as autonomous driving courses taught at various universities, e.g., [29]. Li et al. [30] observed that an important issue is the misunderstanding between self-driving systems and human drivers and proposed a human-like driving system to give autonomous vehicles the ability to make decisions like humans.

Turning any car into an automated car does not always require starting from scratch and controlling the car by injecting messages into the onboard buses, as was in this paper. Commercial electromechanical driving robots can be mounted on the steering wheel and pedals to control the car. For example, AB Dynamics [31] provides a path-following system that uses a driving robot and real-time feedback from GPS-corrected motion information to guide a vehicle along a specified path.

All of the above projects consume a large amount of

[3] https://roborace.com/

computational resources. On the contrary, we are trying to combine all the basic functions needed for automated driving into one inexpensive ECU. This allows us to easily benchmark the system performance, sensitivity, and reliability of the platform without needing a whole trunk full of powerful servers.

2) Trajectory planning: Approaches to path and trajectory planning for automated cars are surveyed in [32]. According to their classification, our algorithm falls into the "Numerical optimization approaches". Another good overview of the state-of-the-art motion and trajectory planners is [33]. Many planners for complex environments are based on the RRT algorithm [34]. In the following, we narrow our review to the approaches used for race and sports cars.

Funke et al. [35] present an approach to control their autonomous Audi TTS on a race track. With respect to trajectory planning, they simply follow an offline pre-computed and optimized path. Such an approach is not applicable to our task, because the position of the cones, and thus the path, is not known in advance. Drage [36] uses a similar approach in their autonomous Formula SAE race car. They select the waypoints for the trajectory generator by manual selection based on the path from a human driver. The trajectory is generated from the waypoints by interpolating several future waypoints with a cubic spline which allows them to achieve a mean lateral error of 70 cm. We experimented with spline-based trajectories and found them unsuitable for several reasons: they do not minimize the steering energy (steering wheel movements), they do not limit the curvature rate of the resulting trajectory, they are very sensitive to imprecise position measurements, and it is difficult to join separate trajectory segments. This turned out to be a problem, especially for dispersed cones, i.e., when the cones are not in a line.

A large body of literature is devoted to the calculation of an optimal racing line – a trajectory around the track that minimizes the lap time [37]. In the mentioned article, Vesel uses a genetic algorithm to find the racing line. Such an approach is not suitable for online trajectory planning. Another popular approach is the application of model predictive control (MPC). For example, Liniger et al. [38] use a non-linear MPC to solve a combination of path planning and trajectory tracking problems. While their approach is suitable for real-time applications, it relies on the offline fitting of the reference trajectory (center line) with a third-order polynomial. We can neither use this approach nor a similar one [39] because our reference path is not known in advance.

Some authors look only at specific parts of the racing trajectories. For example, You and Tsiotras [40] study "cornering", i.e., how to drive through sharp turns. Their solution involves splitting the trajectory into multiple phases and using different approaches in different phases. Our solution is different, but we also detect specific parts of the trajectory (called scenarios below) in order to perform the U-turn, for example.

Determining the waypoints which lie on the optimal racing line is a difficult problem. In the analysis of paths driven by professional race drivers, Kegelman et al. [41] concluded that while different drivers exhibit different driving styles (and paths) to achieve similar drive times, certain track features, such as those with high curvature, lead all the drivers to pass through the same waypoint. Similar claims were made by professional race drivers, when we talked to them. Therefore, our planner tries to find those "important" waypoints and construct the trajectory through them.

While the literature on racing line construction is easily available, works on slalom trajectory construction are much less common. Slalom trajectory planners and related platforms are often mentioned in the context of drones [42]. In [43], the authors define the slalom trajectory through a sequence of waypoints and use quadratic programming to find control actions. This is similar to our internal planning loop (subsequently explained in Section V), but we extend it with an external loop that optimizes the waypoint positions. The drone slalom is also a use case in [44], where a non-linear MPC is used to find the trajectory. However, due to the computational requirements, the MPC solver runs on a powerful off-board PC rather than on the on-board computer. Moreover, the kinematics and dynamics of areal vehicles differ significantly from car-like vehicles, so all these methods are not directly applicable to our use case.

II. Problem statement and architecture overview

A. Problem statement

The goal is to drive a car through a traffic cone slalom. The environment is a flat, obstacle-free area with an arbitrary number of cones arranged approximately in a line, but not necessarily aligned. The positions of the cones are not known; they may be aligned or dispersed (displaced up to 3 meters from the line), they may be equidistant or unequally spaced (from 9 to 20 meters), and the U-turns may be symmetrical or asymmetrical. The drive starts with the car at the beginning of the cone line in the direction of the line. After passing through the cone slalom (later called the *slalom part*), the car turns around the last cone at the end of the line (makes a *U-turn*) and drives back through the slalom part. Then, it makes another U-turn, and the whole process repeats.

B. Hardware and instrumentation

The main computation unit in our solution is an NVIDIA TX2 embedded computer. A single monocular *Basler ace* camera is used to capture the images for the cone detection. We use the camera with a wide-angle lens to detect the cones that are close to the car. A multi-layer Velodyne LP-16 LiDAR is mounted near the camera. We use its data to localize the cones, since calculating the distance from a monocular camera is highly inaccurate.

To localize the car, we employ a differential GPS (DGPS) consisting of a base station and a rover. The rover is mounted on the vehicle and measures its position, receiving corrections from the base station. The rover can be equipped with a second antenna to measure the car's heading, but we do not use it because the heading was often inaccurate due to the car being too short. To improve

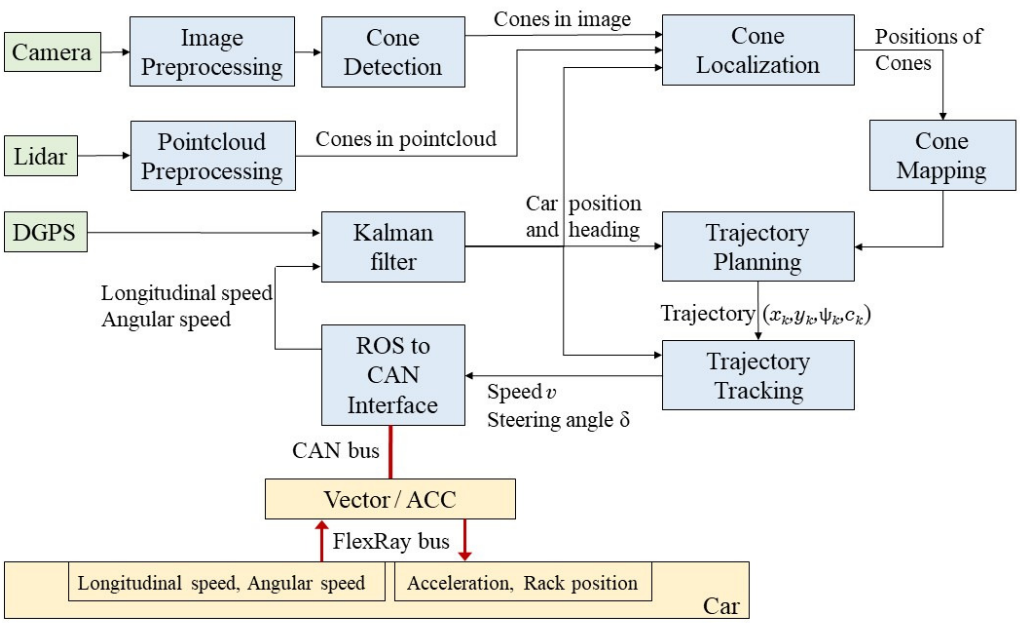


Fig. 2: Functional block diagram – the software components are in blue, the sensors are in green and the car interface is in yellow. The thin lines represent the data flow within the software, the thick red lines are the communication buses.

the results of the vehicle localization, we use the data measured by the vehicle itself, i.e., the longitudinal and angular velocities and the steering angle.

All the sensors are read at a rate of 10 Hz because the TX2 computer cannot perform all the calculations faster. The vehicle is also equipped with a Vector VN8900 computer which interfaces the FlexRay bus in the vehicle to the TX2 computer (TX2 supports the CAN bus only).

C. Architecture

The general functional architecture of our solution is visualized in Fig. 2. Each of the depicted blocks solves a different sub-problem. First, the cones need to be detected in the camera images. This is called *Cone Detection*. Then, the computation of the cone positions on the Earth's surface, called *Cone Localization*, is carried out using information from the LiDAR and the position and heading of the vehicle. The localized cones are stored in the map from which their most probable position is estimated in a process called *Cone Mapping*. The *Kalman Filter* is used to fuse the DGPS position and the internal car signals to obtain a robust estimate of the car's position and heading. The sub-problems of generating a trajectory for the slalom and U-turn and driving the vehicle along that trajectory are referred to as the *Trajectory Planning* and *Trajectory Tracking*. The problem of *interfacing* the vehicle sensors and actuators must also have been solved. This paper concentrates on *Trajectory Planning*. The other blocks are described in [45].

We use the ROS[4] framework as the software platform. Our algorithms are implemented in blocks, called *ROS nodes*, depicted in blue. ROS significantly simplifies the development due to its implementation of sensor drivers and recording/visualization capabilities.

[4]https://www.ros.org/

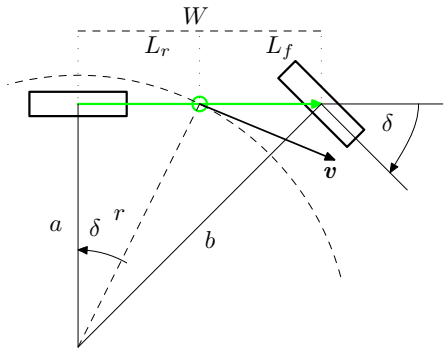


Fig. 3: Kinematic bicycle model

III. Vehicle Modeling

To model the car, we use a kinematic bicycle model [46] with a few additional simplifications. Due to the relatively slow motion of the car, we do not need to work with a dynamic car model that considers the tires' behavior since we do not consider wheel slips. Figure 3 shows a graphical representation of the kinematic model and Fig. 4 illustrates its evolution at three different points in time.

The parameter and variable meanings are as follows:

L_r, L_f	– distance from the center of the car to the rear/front axle
W	– car wheelbase ($L_r + L_f$)
x^g, y^g, ψ^g	– position of the car center and heading in the global-coordinate system
δ	– steering angle
$\boldsymbol{v^g}$	– velocity vector of the car center in the global-coordinate system
$v = \lvert\boldsymbol{v^g}\rvert$	– velocity magnitude
β	– slip angle between the velocity vector and the car heading
r	– car turning radius

The following non-linear differential equations describe this model:

$$\begin{aligned} \beta &= \tan^{-1}\left(\frac{L_r}{W}\cdot\tan(\delta)\right) \\ \dot{x}^g &= v\cdot\cos(\psi^g+\beta) \\ \dot{y}^g &= v\cdot\sin(\psi^g+\beta) \\ \dot{\psi}^g &= v\cdot\frac{\cos(\beta)}{W}\cdot\tan(\delta). \end{aligned} \tag{1}$$

The steering angle δ is not so well suited for the description of the trajectory, since the resulting motion also depends on the vehicle geometry, so instead we use the signed *steering curvature* c, defined as:

$$c = \begin{cases} 1/r & \text{steering clockwise} \\ -1/r, & \text{steering counterclockwise.} \end{cases}$$

While driving through the slalom part, we assume a relatively big turning radius r and a small steering angle δ. Therefore, for the trajectory planner, we use the following simplification of the model:

$$\begin{aligned} \tan(\delta) \approx \delta &\approx c\cdot W \\ \beta &\approx c\cdot L_r \\ \cos(\beta) &\approx 1. \end{aligned} \tag{2}$$

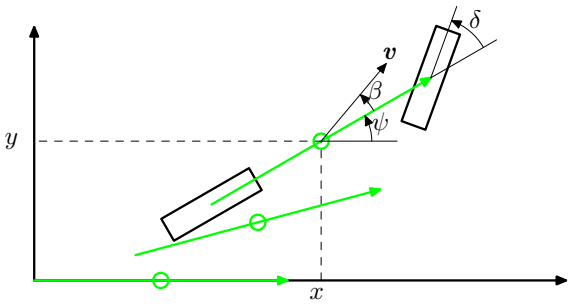


Fig. 4: Evolution of the car's position and heading (the green arrows represent the car at different points in time) in the global-coordinate system

A. Model limitations

The kinematic bicycle model is only a rough approximation of the actual vehicle. This subsection summarizes the conditions under which we can use this model.

Polack et al. [47] show that the kinematic bicycle model is suitable for motion planning purposes if we limit the vehicle's lateral acceleration to a value lower than $0.5\,\mu\,g$, where μ is the tire-road friction coefficient, and g is the gravitational acceleration. Under this condition, the tire slip can be neglected. Typically, for a standard tire on a dry asphalt road, μ is in the range of 0.8 to 1.0 [48]. Therefore, we can safely set the maximal lateral acceleration $a_c = 3\ \mathrm{m}s^{-2}$.

Even under the above-mentioned condition, the model is not completely precise. Its error is caused by (i) the simplification of the real vehicle geometry and (ii) the linearization in Equation (2). In our case, the steering angle error is about 10% for a 6 m turning radius with a 3 m wheelbase. This error decreases to zero for greater radii but increases rapidly for lower radii.

The bicycle model is an underactuated non-linear system with non-holonomic constraints. The control of such a system is not straightforward. We assume only forward motion, which allows us to apply a simple gradient search for an optimal control strategy, as discussed later in Section V.

B. Simple state-space model for the slalom trajectory planner

The state vector $\boldsymbol{x^g}$ is given by the variables of the bicycle model (1). The model inputs $\boldsymbol{u^g}$ are longitudinal acceleration a and curvature rate ε. Formally

$$\boldsymbol{x^g} = \begin{bmatrix} x^g \\ y^g \\ \psi^g \\ c \\ v \end{bmatrix}, \quad \boldsymbol{u^g} = \begin{bmatrix} a \\ \varepsilon \end{bmatrix}. \tag{3}$$

The state-space model in the form $\dot{\boldsymbol{x}}^{\boldsymbol{g}} = \boldsymbol{f}(\boldsymbol{x^g}, \boldsymbol{u^g})$ based on Eqs. (1) to (3) is the following:

$$\begin{aligned} \dot{x}^g &= v\cdot\cos(\psi^g + c\cdot L_r) \\ \dot{y}^g &= v\cdot\sin(\psi^g + c\cdot L_r) \\ \dot{\psi}^g &= v\cdot c \\ \dot{c} &= \varepsilon \\ \dot{v} &= a. \end{aligned} \tag{4}$$

Due to the specifics of the slalom use case, we can simplify the problem by assuming that the vehicle is heading toward the row of cones (except for the U-turn). Namely, we transform the global coordinate system x^g, y^g, ψ^g model (4) to the model in the "slalom" coordinate system x, y, ψ, such that the x axis intersects the cone row (or minimizes the least-square distance from the cones if they are dispersed). Before starting the planning algorithm, we assume that the car is on the x axis, so that the center position and heading of the car is $(x, y, \psi) = (0, 0, 0)$. Also, we simplify the longitudinal equation and assume that it is $\dot{x}v$ since the heading $\psi \approx 0$ and the steering angle δ are small, as suggested in Eq. (2). Moreover, the longitudinal speed remains constant (except for the U-turns), and thus $v = \text{const}$. The curvature c remains the same in both coordinate systems.

Differential equations (5) summarize the model, the state vector $\boldsymbol{x}$, and single input $\boldsymbol{u}$. Note that v and L_r are constants.

$$\begin{aligned} \dot{y} &= v \cdot \sin(\psi + c \cdot L_r) \\ \dot{\psi} &= v \cdot c \\ \dot{c} &= \varepsilon \end{aligned} \qquad \boldsymbol{x} = \begin{bmatrix} y \\ \psi \\ c \end{bmatrix} \qquad \boldsymbol{u} = \begin{bmatrix} \varepsilon \end{bmatrix} \tag{5}$$

For the system optimization, we transform (5) into a discrete-time model with the sample time T given by the difference equations (6).

$$\begin{aligned} y_{k+1} &= v \cdot \sin(\psi_k + c_k \cdot L_r) \cdot T + y_k \\ \psi_{k+1} &= c_k \cdot v \cdot T + \psi_k \\ c_{k+1} &= \varepsilon_k \cdot T + c_k \end{aligned} \tag{6}$$

The longitudinal position $x_{k+1} = v \cdot T + x_k$ is not a part of the dynamic system optimization. Hence, it is not included in the model.

IV. Slalom Trajectory Planning

In this section, we describe the novel algorithm for planning slalom trajectories. The algorithm finds a near-optimal slalom trajectory that minimizes the drive time, steering "energy" and passenger discomfort, while respecting the vehicle kinematics and dynamics.

The inputs to the algorithm are the positions of the cones on the map, as well as the initial position, speed, heading, and steering angle of the vehicle. The online algorithm calculates the trajectory for both the slalom part and the U-turns at the ends. We implement a rolling horizon approach that considers at most three cones in front of the car. In this way, the algorithm works for any number of cones in the slalom, even if not all of them are initially detected by the on-board camera and LiDAR images. Moreover, as the car approaches a cone, its position becomes more accurate, allowing it to find a better trajectory.

The algorithm uses the notion of waypoints that must be driven through by the car. The waypoint is a vector $(x, y, \psi, c)^T$ where x, y are the Cartesian coordinates on the map, ψ is the car heading, and c is the steering curvature. The car drives through the waypoint when the center of the rear axle is at its coordinates, with the given heading and steering angle matching the curvature. We use two types of waypoints. The *fixed waypoint* is defined as $\boldsymbol{x} = (x, y, \psi, c)^T$, and we use it for the final point on the planned trajectory. The other type is called a *flexible waypoint*, which defines only the position (x, y), and is used for an intermediate point on the trajectory. The other state variables, namely ψ and c, are determined later in the algorithm run.

The online trajectory planning process works as shown in Algorithm 1. The actual position of the car at the replanning point is denoted by $\boldsymbol{x}_{\text{rpln}}$. First, the $ScenarioDetect\&WaypointsGeneration$ function determines the actual scenario (see the description of the three scenarios in Subsection IV-A), the fixed waypoint, and the initial value of the flexible waypoint. The waypoints and scenario, along with the current car position and cone map, are passed to the $OptimizationAlgorithm$,

Algorithm 1: Slalom trajectory planning

Input : $\boldsymbol{x}_{\text{init}}$, map, $replan\_distance$
Output: trajectory $\mathbf{X}_{\text{track}}$ of whole track
$\mathbf{X}_{\text{track}} \leftarrow \emptyset$
$\boldsymbol{x}_{\text{rpln}} \leftarrow \boldsymbol{x}_{\text{init}}$
while *driving* **do**
  [scenario, $wpt_{\text{fixed}}, wpt_{\text{initflex}}$] = ScenarioDetect&WaypointsGeneration($\boldsymbol{x}_{\text{rpln}}, map$)
  $\mathbf{X}$ = OptimizationAlgorithm($\boldsymbol{x}_{\text{rpln}}, wpt_{\text{fixed}}, wpt_{\text{initflex}}, scenario, map$)
  $\mathbf{X}_{\text{track}} = Append(\mathbf{X}_{\text{track}}(1, \ldots, \arg(\boldsymbol{x}_{\text{rpln}})), \mathbf{X})$
  $\boldsymbol{x}_{\text{rpln}} = \mathbf{X}(replan\_distance)$
  wait for reaching $\boldsymbol{x}_{\text{rpln}}$ while updating map

which finds a flexible waypoint and calculates trajectory $\mathbf{X}$, which is then appended to the previously calculated one. A new replanning point is a point on the trajectory determined from the $replan\_distance$ parameter that must be smaller than the length of the currently planned trajectory $\mathbf{X}$. Then the car follows the planned trajectory until it reaches the replanning point $\boldsymbol{x}_{\text{rpln}}$. A new iteration is started and the trajectory is replanned. This continues until the algorithm stops.

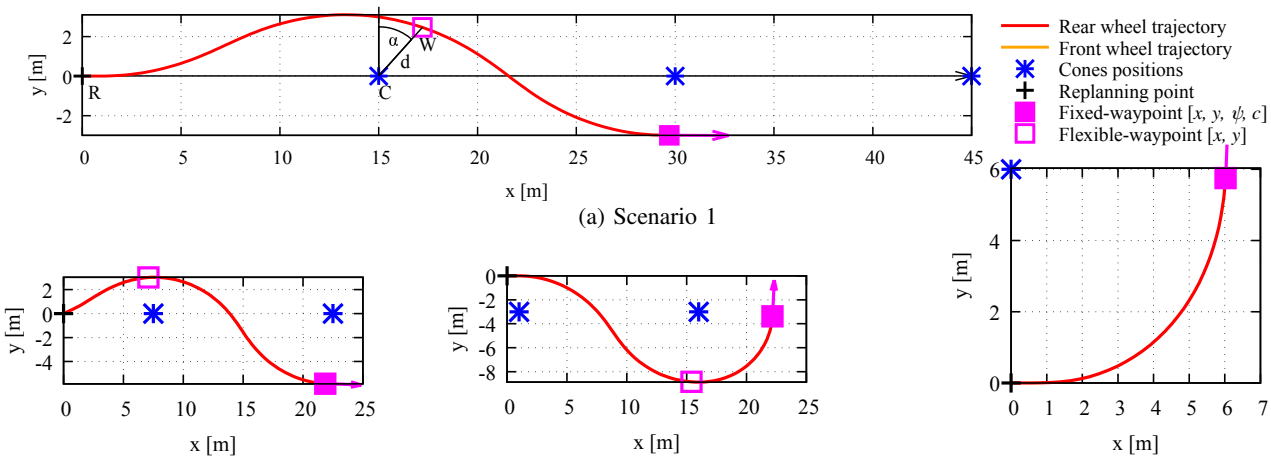
To find the best-performing algorithm, we conducted many simulations and real-world experiments with a variable number of waypoints and different criteria. We experimented with different lengths of the rolling horizon (i.e., from the actual position to the most distant waypoint) and several algorithms combining optimization and exhaustive search in the space of waypoint parameters.

Our solution uses one flexible waypoint positioned next to the nearest cone and one fixed waypoint further away. The only exception is a simple scenario related to the U-turn, where a fixed waypoint is used without any flexible waypoint. The exact position, direction, and curvature of the fixed waypoint are not critical because the position of the distant cone becomes more accurate as it gets closer, and the new iteration of Algorithm 1 generates new waypoints and a new trajectory. The flexible waypoint only specifies the position the car must pass, and the optimization algorithm computes the trajectory tangent and curvature at that position. This solution gives surprisingly good results even if the distances between the cones are unequal and the cones are not on a straight line. This ensures a smooth overlap of the trajectories provided by the subsequent iterations of Algorithm 1. Experiments in Section VII show that such an approach is robust enough to handle inaccuracies in the cone detection and perturbations in the DGPS.

A. Scenario detection

The position of the waypoints depends on what we call a scenario. We distinguish three different scenarios, which are shown in Fig. 5 and described below:

a) Scenario 1: When at least three cones are detected in front of the car, no U-turn will be performed in the near future. We set the initial position of the flexible waypoint next to the first cone and the fixed waypoint next to the second cone, as shown in Fig. 5(a). The trajectory (starting



(a) Scenario 1

(b) Scenario 2 with two detected cones

(c) Scenario 2 with one detected cone

(d) Scenario 3

Fig. 5: Setting of the fixed and flexible waypoints in three different scenarios

at the current position of the vehicle and ending at the fixed waypoint) and the final parameters of the flexible waypoint are determined by the Optimization Algorithm described in Section V.

b) Scenario 2: When the scenario detected in the previous iteration of Algorithm 1 was Scenario 1, and now only one or two cones are detected in front of the car, we need to perform the U-turn around the last cone of the slalom. We set the initial position of the flexible waypoint next to the cone in front of the car. If two cones are detected, we set the fixed waypoint next to the last cone, as shown in Fig. 5(b). If only one cone is detected (see Fig. 5(c)), the fixed waypoint is placed behind the cone. The trajectory and the final parameters of the flexible waypoint are determined by the Optimization Algorithm in the same way as in Scenario 1.

c) Scenario 3: When there is no cone in front of the car, then we already perform the U-turn around the cone. We set one fixed waypoint on a circle around the cone 90 degrees from our current position, as shown in Fig. 5(d). Since there is no flexible waypoint, the trajectory is found only by the internal loop of the Optimization Algorithm (as described in Subsection V-B). The parameters of the U-turn circle are found while considering a constant velocity. We want to minimize the drive time t_c on the circular path. This time is constrained by the maximum lateral acceleration a_c (related to the tire friction limit) and by the minimum turning radius r of the vehicle, so that $t_c = 2\pi\sqrt{\frac{r}{a_c}}$. Therefore, we set the U-turn radius as low as possible, while still keeping some safety margin for maneuvering. Specifically, we set $r = 6\,\mathrm{m}$ for our vehicle. The center of the circle is placed either (based on the configuration parameter) in the center of the cone (see symmetric U-turn in Figure 9) or such that the car leaves the U-turn circle very close to the cone (see asymmetric U-turn in Figure 10). In the next iteration of Algorithm 1, Scenario 1 will be used as there will be at least three cones in front of the car.

B. Waypoints in polar coordinate system

The position x, y of the flexible waypoint is based on a Cartesian coordinate system, which is unsuitable for the optimization algorithm because the feasibility space of the possible car positions around the cone is clearly non-convex (infeasible car positions are surrounded by feasible ones). Therefore, in the Optimization Algorithm (see, e.g., Fig. 7(d)-(f)) we represent the waypoints in the polar coordinate system determined by distance d from the corresponding cone center C and angle α from the direction perpendicular to the cone axis, oriented to the side where the car passes the cone, as seen in Fig. 5(a).

V. Optimization Algorithm

In this section, we describe the Optimization Algorithm inspired by the Logic-Based Benders Decomposition, which is well known to solve various combinatorial optimization problems [49]. The input of the algorithm is the current vehicle state $\boldsymbol{x}_{\text{rpln}}$, the initial position of the flexible waypoint, the fixed waypoint, and the cone map. The Optimization Algorithm consists of two nested loops (see Fig. 6):

- an external loop whose iteration starts with the positioning of the flexible waypoint (determining its (x, y); the initial position was set in accordance with the detected scenario) and ends with the evaluation of the solution feasibility and criterion value,
- an internal loop which plans trajectory $\mathbf{X}$ given by vector $\boldsymbol{x}$ of each discrete-time step of the trajectory including heading ψ and curvature c of the flexible waypoint.

After the internal loop is executed and the velocity profile is computed, the feasibility of the found trajectory is evaluated (i.e., it is verified that the car avoids colliding with the cone) and the criterion function described in the next section is evaluated. The algorithm then performs another iteration of the external loop to find another position (x, y) of the flexible waypoint, and repeats the process as shown in Fig. 6.

The criterion function U of the Optimization Algorithm is given as a minimization of the weighted sum of the time to reach the fixed-waypoint, the sum of the squares of the curvature c (assuming constant speed, which should correspond to the energy spent on steering), and the sum

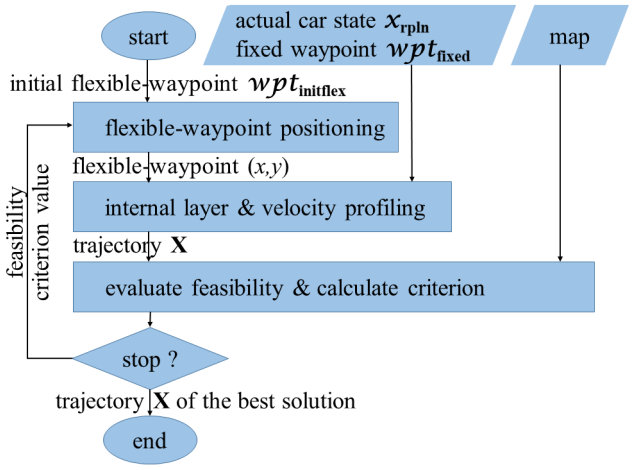


Fig. 6: Flow chart of the Optimization Algorithm (in the case of Scenario 3, the algorithm simply executes the internal loop only).

of the squares of the curvature rate ε (corresponding to the lateral jerk; a large lateral jerk makes discomfort even for very short periods of time [50]):

$$U = w_t \cdot (k_{\text{fw}} - k_{\text{rp}}) \; + \; w_c \cdot \sum_{k=k_{\text{rp}}}^{k_{\text{fp}}} c_k^2 \; + \; w_\varepsilon \cdot \sum_{k=k_{\text{rp}}}^{k_{\text{fw}}} \varepsilon_k^2$$

The variables k_{rp} and k_{fw} are the indices of the discrete-time steps when the car is at its replanning point and fixed waypoint, respectively.

Note: One can also think of a part of the criterion function related to the energy consumed by the longitudinal motion (as in platooning [51]), namely given by the variable longitudinal velocity. The energy consumption (see accurate models for electric cars, e.g., in [52], [53]) is equal to the integral of the power over time. The power is equal to the velocity of the vehicle times a force acting on the vehicle, which is composed of the rolling resistance force between the tires and the road, the force resulting from the differences in elevation, the resistance of the car against the air, and the acceleration force. These forces depend on static parameters (such as the total mass of the vehicle, rolling resistance, elevation changes along the way, air density, aerodynamic drag coefficient expressing the aerodynamics of the vehicle shape, and vehicle frontal area) and variables such as the speed and acceleration. Therefore, our optimization algorithm that derives the velocity profile and then computes the criterion value might, in principle, be able to optimize the energy consumption as well, but we would need to investigate whether the criterion function has a convex shape to make it fast, i.e., to minimize the number of iterations of the external loop.

A. External loop – flexible-waypoint positioning and criterion calculation

The position of the flexible waypoint next to the first cone is a crucial variable of the solution. To observe its influence on the criterion value, we performed simulations of several cases shown in Fig. 7(a), (b), (c) (different positions of the car and the cones). For each of these cases, we computed the criterion map shown in Fig. 7(d), (e), (f) in polar coordinates. The color of a small rectangle for each discrete value of angle α and distance d represents the value of the criterion function U of the trajectory $\mathbf{X}$ found by running the internal loop separately. Given an unknown trajectory, it would be quite difficult to analytically deduce whether the position of the waypoint yields an infeasible solution, since the car is not a single point. However, using the internal loop, we obtain the trajectory, and we can evaluate its feasibility (whether the cone was hit or not). The white rectangles represent infeasible waypoint positions that lead the car too close to the cone. The colored rectangles represent a feasibility set of the given case. Therefore, the objective of our Optimization Algorithm is to find a solution represented by one of the dark blue rectangles, but we want to derive it online without having the criterion map found by the exhaustive search for the given case, as in Fig. 7.

To find an optimal feasible solution, we use the Border Discovery procedure for Scenario 1 and the Local Search procedure for Scenario 2.

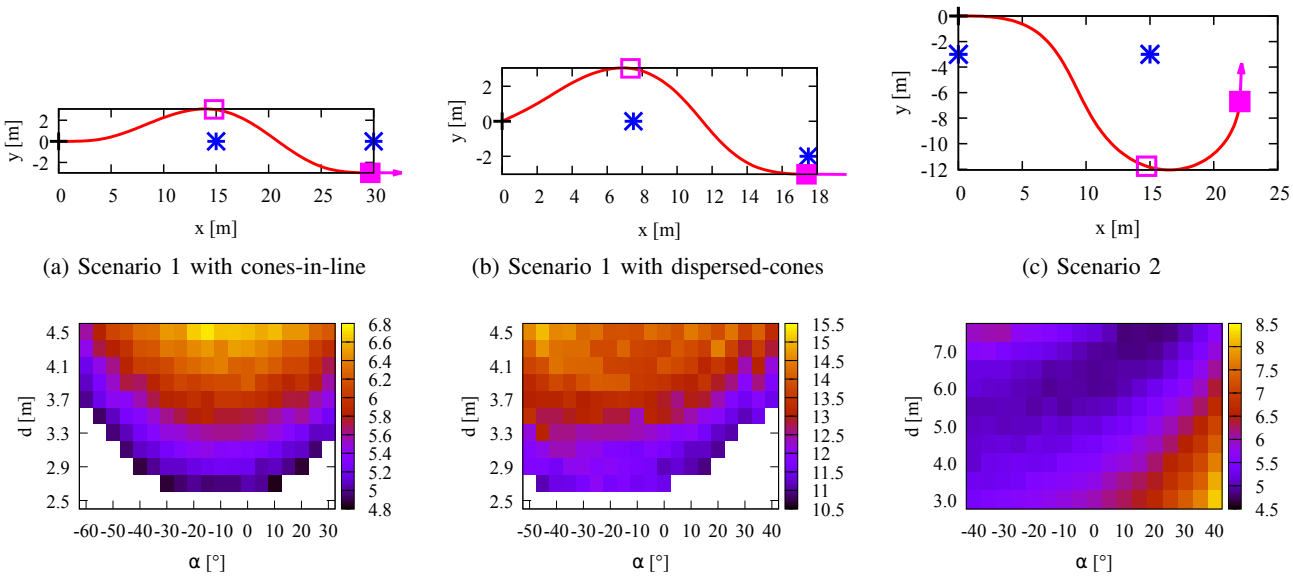
1) Flexible-waypoint positioning by the Border Discovery procedure: Initially, we represented the criterion map of the position of the flexible waypoint in Cartesian coordinates (x, y), where the set of feasible solutions is a nonconvex space (the car goes around the cone, so there are infeasible positions surrounded by the feasible ones), making it difficult for the Optimization Algorithm to escape from the local minimum. Fortunately, we have changed the representation to polar coordinates $[\alpha, d]$, resulting in convex feasibility space and a (mostly) concave criterion function, as shown experimentally in Fig. 7(d), (e) (due to the complexity of the internal loop and feasibility evaluation, we cannot provide a theoretical proof here). Minimizing the concave criterion on the convex feasibility space ensures that the minimum lies on the border of the feasibility set, thus simplifying the optimization problem. Taking advantage of this property, we have developed the Border Discovery procedure using the interval bisection method in the direction of d and in the direction of α.

Border Discovery procedure:

Init Let $[\alpha^f, d^f]$ be an initial position of the flexible waypoint that is undoubtedly feasible (e.g., $[0, 4]$), and d^i be a distance that is undoubtedly infeasible (for example 0.5 meter) for any angle α. Let d^{diff} and α^{diff} be two constants influencing the precision of the result.

(i) Distance discovery. Using the interval bisection method (i.e., in each iteration, the interval is divided between feasible and infeasible endpoints) with initial endpoints $[\alpha^f, d^f]$ and $[\alpha^f, d^i]$, find a feasible solution with the smallest d, using iterative calls to the internal loop for the given position. Terminate the vertical discovery when the change in d in two consecutive iterations is less than d^{diff} and save the obtained distance as d^v. If the termination condition is satisfied, then EXIT, otherwise proceed to (ii).

(ii) Angle discovery. Using one sample position, find whether the feasibility set is on the left or on the right of $[\alpha^f, d^v]$. Find the infeasible position $[\alpha^i, d^v]$ behind the feasibility set in the horizontal direction. Using



(a) Scenario 1 with cones-in-line (b) Scenario 1 with dispersed-cones (c) Scenario 2

(d) Criterion map of (a) (e) Criterion map of (b) (f) Criterion map of (c)

Fig. 7: Criterion function for three different cases

the interval bisection method with initial endpoints $[\alpha^f, d^v]$ and $[\alpha^i, d^v]$, find a feasible solution closest to $[\alpha^i, d^v]$ using iterative calls to the internal loop for the given position. Stop the horizontal discovery when the change in α in two consecutive iterations is less than α^{diff} and save the obtained angle as α^h. Set $[\alpha^f, d^f]$ in the middle of the interval $[\alpha^f, d^v]$ and $[\alpha^h, d^v]$. If the stopping condition is satisfied, then EXIT, otherwise continue with (i).

Stop The stopping condition is related to the elapsed time or distance between $[\alpha^f, d^v]$ and $[\alpha^h, d^v]$. The algorithm returns the feasible position with the best value of the criterion function.

2) Flexible-waypoint positioning by the Local Search procedure: The criterion function of the case in Fig. 7(f) has its minimum within the feasibility set and thus it must be found by a simple Local Search procedure. Within the feasibility set, the Local Search applies local moves in eight different directions and with constant distance and chooses the best of them as the starting point for the next iteration. The next iteration uses a shorter distance. The best solution is selected among the evaluated ones. The termination condition corresponds to the elapsed time. The Local Search procedure can also be used to solve the cases Fig. 7(d), (e), but the DB procedure was approximately three times faster in our experiments due to a smaller number of iterations.

B. Internal loop – trajectory planning via two waypoints

We formulate the internal loop trajectory planning as a discrete-time dynamic system optimization problem. The solution is based on the Newton-type optimization algorithm [54].

We are given the initial state $\mathbf{x}_0$ (i.e., the replanning point), the final state $\mathbf{x}_f$ (i.e., the fixed waypoint), and, optionally, a flexible waypoint, $\mathbf{x}_{\text{flex}}$. Recall that the state is $[y, \psi, c]^T$, since we assume a constant speed, see Eq. (6), and the longitudinal coordinate is computed using the velocity profile described below in Section V-C. We want to generate such a control input sequence that leads the car optimally from the state $\mathbf{x}_0$ to the state $\mathbf{x}_f$ in N steps, optionally via the position of the flexible waypoint $\mathbf{x}_{\text{flex}}$. In other words, we want to find control inputs that minimize the cost function L. In mathematical formalism, the statement of a constrained optimization problem is as follows:

$$\begin{aligned} \underset{\mathbf{x}_1 \ldots \mathbf{x}_N, \mathbf{u}_0 \ldots \mathbf{u}_{N-1}}{\text{minimize}} \quad & L(\mathbf{X}) \\ \text{subject to:} \quad & \mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) \quad k = 0 \ldots, N-1 \\ & \mathbf{g}(\mathbf{x}_m, \mathbf{x}_{\text{flex}}) = 0 \quad m = 1, \ldots, N-1 \\ & \mathbf{x}_N = \mathbf{x}_f \end{aligned} \tag{7}$$

Where $\mathbf{X}$ is a vector of all the states and inputs in the optimization and $L(\mathbf{X})$ is the general cost function restricted to be a weighted quadratic function.

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_N \\ \mathbf{u}_0 \\ \vdots \\ \mathbf{u}_{N-1} \end{bmatrix}, \quad L(\mathbf{X}) = \sum_i w_i \mathrm{X}_i^2, \quad w_i \geq 0 \tag{8}$$

Equation $\boldsymbol{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$ is a discretized version of the simplified state-space model (6). Constraint $\mathbf{x}_N - \mathbf{x}_f = 0$ defines the final state, optional constraints $\mathbf{g}(\mathbf{x}_m, \mathbf{x}_{\text{flex}}) = \mathbf{x}_m - \mathbf{x}_i = 0$ define that the flexible waypoint must be on the trajectory (in general, this method allows multiple $\mathbf{g}()$ constraints, but we only use one for one flexible waypoint).

Following [54], this problem can be solved with a Newton-type iterative algorithm. We determine the number of samples N in two steps as follows. First, N is estimated based on the Euclidean distance to the final state and the sample distance. Then, the calculated curvature rate control input is applied to model (4) along with a constant

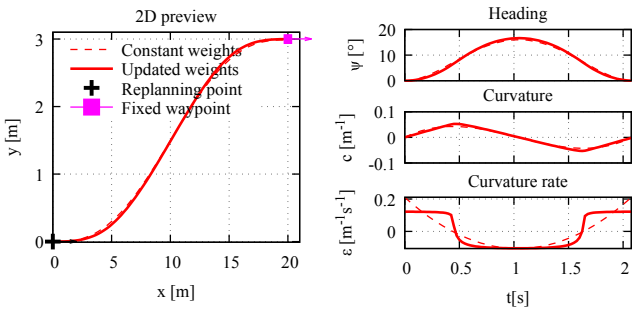


Fig. 8: Computed trajectory (red) between two fixed waypoints (magenta). The effect of different weights w_i is visible, namely on the curvature rate.

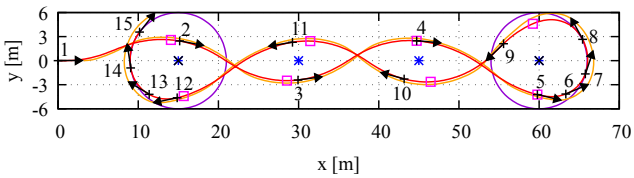


Fig. 9: Slalom with cones-in-line, equal distances, and symmetric U-turns (the nomenclature is the same as in Fig. 5).

velocity, and the real final state $\mathbf{x}'_f$ is obtained, which may differ slightly from $\mathbf{x}_f$. In the second step, we correct N so that the trajectory ends in the desired final state $\mathbf{x}_f$. The difference between the real and desired final states can also indicate that we choose the final state outside the reachable range, as discussed in Section III-A. The result of the algorithm (found after a single iteration) with constant weights w_i is shown in Fig. 8 with the dashed line (the weights are constant in the sense that they do not change for different $k = 0, \ldots, N-1$). The index m of the flexible waypoint $\mathbf{x}_{\text{flex}}$ is calculated in relation to N.

By choosing different weights w_i for different states, the planner can implement several different driving strategies. For example, to minimize the curvature rate for the driver's comfort or to minimize the curvature to drive faster with maximum lateral acceleration. However, an output trajectory is not always well suited for driving, as it can create unacceptable spikes in the curvature and curvature rate. This can be eliminated by shaping the weights. If we have a spike at time step i, we set a higher weight w_i for that particular state. The implemented solution sets the weights proportional to the absolute values of the associated states of the result with constant weights and runs the Optimization Algorithm again. Then we obtain the trajectory with suppressed peaks. We add a small constant value to the weights to prevent the weights w_i from being zero, which would lead to the infeasibility of the algorithm. The example result with updated weights is shown in Fig. 8 with a solid line. The driving strategy shown in the figure is to minimize the curvature rate. The result is close to the clothoid curve.

In summary, in two iterations of the algorithm, we obtain a trajectory that respects the kinematic constraints of the vehicle.

C. Velocity Profiling

To minimize the drive time, after calculating the trajectory with constant velocity, we recalculate the velocity to satisfy three constraints. The first constraint is the maximum value of the longitudinal acceleration and deceleration. The second constraint is the maximum value of the lateral acceleration during steering, which limits the maximum speed. The third constraint is the maximum speed that the vehicle can travel.

We adapted the two-pass algorithm presented in [55]. The algorithm consists of the following steps. The maximum velocity is determined at each point along the trajectory. The velocity change is given by the equation $v[k+1] = v[k] + a[k]T$. Then the velocity profile is calculated by applying the maximum allowable acceleration and deceleration. We calculate the velocity profile for the acceleration by starting at the first point of the trajectory and applying the acceleration until we reach the velocity limit. Then the same procedure is performed, starting at the end of the trajectory, going back to the beginning and applying the maximum allowable deceleration.

VI. Evaluation of Trajectory Planner

To evaluate the proposed trajectory planner, we perform simulations on three different configurations depicted in Figs. 9 to 11.

Figure 9 shows the cones-in-line configuration with equidistantly arranged cones. The U-turns are symmetric, that is, their center lies at the position of the outer cones. The trajectory starts at replanning point 1, Scenario 1 is detected, and the vehicle passes over the flexible waypoint on the left side of the first cone. Later, Scenario 2 is detected at replanning points 3 and 4 as the vehicle approaches the U-turn around the last cone. Then the U-turn is performed and Scenario 3 is detected at replanning points 5, 6, and 7, where no flexible waypoint is generated. The car returns when Scenario 1 is detected at replacing point 8, and the sequence of scenarios repeats.

Figure 10 shows the cones-in-line configuration, but it is different from Fig. 9 in two aspects:

- The U-turns are asymmetric, e.g., the cone is at position $[15, 0]$, but the center of the U-turn is at $[15, -3]$. The U-turn is placed so that the car leaves the U-turn in a similar state as if there were no U-turn. Let us look at the left U-turn – the end of the U-turn almost overlaps with the trajectory generated between replanning points 1 and 2. Specifically, at replanning point 15, we get the flexible waypoint at $[15, 2]$, which almost overlaps with the flexible waypoint of replanning point 1. This way of making asymmetric U-turns, which was advised to us by a professional driver, has nearly similar drive times as the symmetric version. However, the trajectory with symmetric U-turns has much higher sums of squares of curvature and curvature rate. Thus, the proposed asymmetric U-turn planning results in lower lateral acceleration and a more comfortable ride with lower lateral jerk.
- The distance between the third and fourth cones is shorter, and therefore the car must pass through the space between them almost vertically. Therefore,

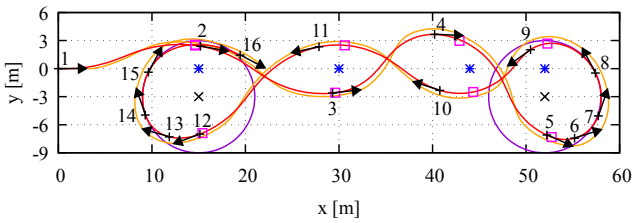


Fig. 10: Slalom with cones-in-line, unequal distances, and asymmetric U-turns (the nomenclature is the same as in Fig. 5).

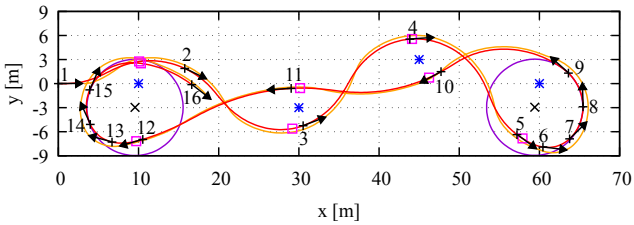


Fig. 11: Slalom with dispersed cones (the nomenclature is the same as in Fig. 5).

when it is at replanning point 3, the flexible waypoint at $[44, 2.5]$ has a heading that points straight between the cones (i.e., not parallel to the cone line, as is the case with replanning point 10 and the corresponding flexible waypoint at $[30, 2.5]$).

The last configuration with dispersed cones, unequal cone spacing, and asymmetric U-turns, shown in Figure 11, is the most interesting. Figure 12 shows the remaining states of this trajectory with the replanning points matching both figures. We see a serpentine slalom drive from left to right, then an almost straight drive from right to left. Our simple speed profiling algorithm results in spikes in the slalom speed profile. The real driver would instead drive at a constant speed, with the car swinging on its suspensions. However, we do not model the suspensions and roll rotation in this paper. Therefore, the car accelerates and brakes in relatively straight segments between the cones. The curvature rate has peaks in Scenarios 1 and 2 (replanning points 1 to 5 and 9 to 12), which is understandable since it must switch between left and right turns. However, these peaks should not occur during the U-turns (replanning points 6, 7, 8 and 13, 14, 15). We see two reasons for this behavior. First, we operate at the limit of the internal loop planner (sharp steering). Second, we do not consider flexible waypoints in Scenario 3, so the external loop does not optimize them, resulting in a spike at the corresponding rescheduling points. These spikes could interfere with the slalom drive, but are smoothed out by the feedback control.

From the presented simulation results, we conclude that the proposed method is suitable for a wide range of slalom configurations. The results of the experiments with the real vehicle are presented in Section VII.

VII. Experiments

We performed many test drives with the entire setup described in the previous sections. In this section, we present data from some of these real-world experiments.

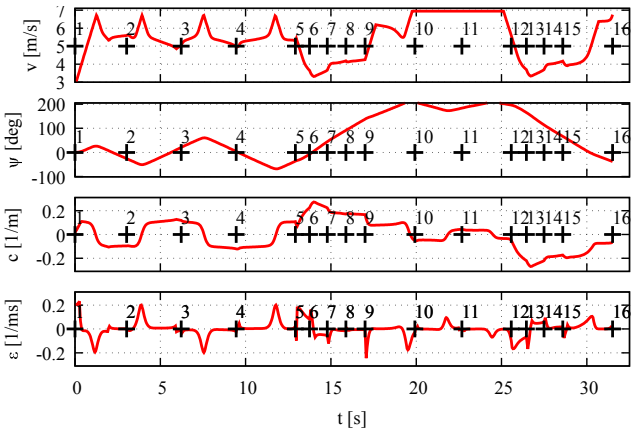


Fig. 12: Slalom with dispersed cones – states

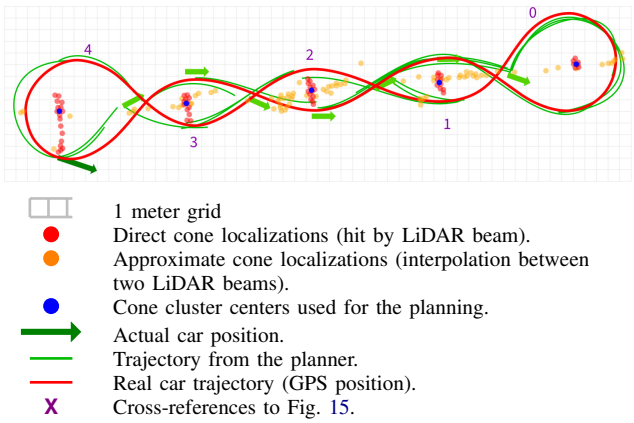


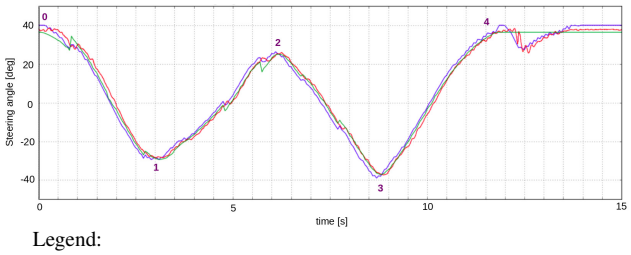
Fig. 13: Automated slalom map

An overview visualization of the data of the slalom drive is shown in Fig. 13. As mentioned above, the approximate cone localization (orange dots) is used only during the first slalom part (detailed in Fig. 14). After the car is close enough for direct localization, more precise cone positions (red dots) are used. The approximate positions are only used to calculate the initial trajectory.

The trajectory tracker exhibits higher tracking errors in the U-turns, as shown in Fig. 13 on the left. This is due to the fact that the tracker controller is tuned for the slalom part.

The graph of the steering angle during a slalom part is shown in Fig. 15. The matching points in Figs. 13 and 15 are marked with numbers. In the output of the trajectory tracking controller (blue line), overshoot can be seen at the beginning of the U-turn after point 4. This is again due to the trajectory tracker controller being tuned to the slalom part of the drive. The step changes in the output of the local planner (green line) caused by the re-planning of the trajectory are partially filtered by the trajectory tracker and by the power steering ECU itself, so that the resulting

Fig. 14: Creation of the cone map during the first slalom part drive. For the legend, see Fig. 13.



Legend:

- Steering angle reported by the car.
- Steering angle output of the local planner.
- Trajectory tracking controller output (incorporates communication delay, deviation of the car from the planned position and interpolation of the old and new plans).
- X Cross-references to Fig. 13.

Fig. 15: Steering angle during the slalom drive

Legend:

- LiDAR pointcloud projected to the camera image. The point colors represent the distance.
- Cone centers detected in the camera image.
- Computed cone positions with the map (absolute) coordinates projected to the image.

Fig. 16: Camera view from an experimental drive

steering wheel movement is close to the steering of a human driver.

Figure 16 shows what the camera and LiDAR see when the car is in front of the cones.

VIII. Conclusion

This paper describes the architecture and algorithms of a prototype automated vehicle performing slalom drives. The goal is to automate repetitive vehicle tests to make them more reproducible, thus facilitating the development of advanced control algorithms. Our trajectory planner plans the whole trajectory, including U-turns, that minimizes the driving time, steering energy, and passenger discomfort.

The planner constructs the path from the waypoints that must be driven through by the car. It places the waypoints based on the position of the cones detected by the LiDAR and camera. The initial position of the waypoints is then optimized by an online optimization algorithm based on a logic-based Benders decomposition and Newton-type optimization. Fast computation times are achieved by formulating the optimization in polar coordinates, which leads to convex feasibility space. Both the simulations and practical experiments show that the method is robust enough to handle disturbances caused by the inaccuracy of the cone detection and DGPS.

Our prototype is based on a proven modular architecture that allows for future improvements to the individual components. A simple initial design, implementation, and early outdoor experimentation proved to be the right way to uncover bottlenecks and weaknesses in the development versions of the system and to enable the improvement of the individual components in an agile manner. In particular, it quickly became apparent, in practice, that the accuracy of one component (heading estimation) is crucial for the correct behavior of the other components.

Given the limited resources of the project, we have learned that outdoor experiments are more efficient than simulating the entire system and environment, which cannot model every detail of the real world. It proved beneficial to simulate only certain parts of the system and environment to reproduce problems encountered in outdoor experiments.

Acknowledgments

We thank the anonymous reviewers for improving the readability of the paper. We thank Martin Vajnar, David Kopecký, Riccardo Mancini, and Mario Terlizzi for their work on several important parts of this project. We also thank Porsche Engineering Services for providing the experimental Porsche Panamera and professional drivers. This work was supported by the Grant Agency of the Czech Republic under Project GACR 22-31670S.

References

[1] X. Zhang, J. Tao, K. Tan, M. Törngren, J. M. G. Sánchez, M. R. Ramli, X. Tao, M. Gyllenhammar, F. Wotawa, N. Mohan, M. Nica, and H. Felbinger, "Finding critical scenarios for automated driving systems: A systematic mapping study," *IEEE Transactions on Software Engineering*, vol. 49, no. 3, pp. 991–1026, 2023.

[2] J. Dvorak and Z. Hanzalek, "Using two independent channels with gateway for FlexRay static segment scheduling," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 5, pp. 1887–1895, Oct 2016.

[3] SAE International, "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles," 2018, https://www.sae.org/standards/content/j3016_201609/.

[4] P. A. Ioannou and C. C. Chien, "Autonomous intelligent cruise control," *IEEE Transactions on Vehicular Technology*, vol. 42, no. 4, pp. 657–672, 1993.

[5] U. Z. A. Hamid, F. R. A. Zakuan, K. A. Zulkepli, M. Z. Azmi, H. Zamzuri, M. A. A. Rahman, and M. A. Zakaria, "Autonomous emergency braking system with potential field risk assessment for frontal collision mitigation," in *2017 IEEE Conference on Systems, Process and Control (ICSPC)*, 2017, pp. 71–76.

[6] C. Jang, C. Kim, S. Lee, S. Kim, S. Lee, and M. Sunwoo, "Re-plannable automated parking system with a standalone around view monitor for narrow parking lots," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 2, pp. 777–790, 2020.

[7] V. Gadepally, A. Krishnamurthy, and U. Ozguner, "A framework for estimating driver decisions near intersections," *IEEE Trans. on Intelligent Transportation Systems*, vol. 15, no. 2, pp. 637–646, 2014.

[8] Z. Wang, X. Zhao, Z. Xu, X. Li, and X. Qu, "Modeling and field experiments on autonomous vehicle lane changing with surrounding human-driven vehicles," *Computer-Aided Civil and Infrastructure Engineering*, 2020.

[9] X. Zhang, W. Zhang, Y. Zhao, H. Wang, F. Lin, and Y. Cai, "Personalized motion planning and tracking control for autonomous vehicles obstacle avoidance," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 5, pp. 4733–4747, 2022.

[10] H. Li, G. Yu, B. Zhou, P. Chen, Y. Liao, and D. Li, "Semantic-level maneuver sampling and trajectory planning for on-road autonomous driving in dynamic scenarios," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 2, pp. 1122–1134, 2021.

[11] Audi, "2019 Audi A8 level 3 self-driving real world test," 2019, https://www.youtube.com/watch?v=WsiUwq_M8lE.

[12] S. Behere, "Reference architecture for highly automated driving," Ph.D. dissertation, KTH, Stockholm, 2016.

[13] K. Jo, J. Kim, D. Kim, C. Jang, and M. Sunwoo, "Development of autonomous car—part ii: A case study on the implementation of an autonomous driving system based on distributed architecture," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 8, pp. 5119–5132, Aug 2015.

[14] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*, M. Hutter and R. Siegwart, Eds. Cham: Springer International Publishing, 2018, pp. 621–635.

[15] Y. Kang, H. Yin, and C. Berger, "Test your self-driving algorithm: An overview of publicly available driving datasets and virtual testing environments," *IEEE Transactions on Intelligent Vehicles*, vol. 4, no. 2, pp. 171–185, June 2019.

[16] G. Guo and D. Li, "Pmp-based set-point optimization and sliding-mode control of vehicular platoons," *IEEE Transactions on Computational Social Systems*, vol. 5, no. 2, pp. 553–562, 2018.

[17] I. Herman, D. Martinec, Z. Hurák, and M. Sebek, "Scaling in bidirectional platoons with dynamic controllers and proportional asymmetry," *IEEE Transactions on Automatic Control*, vol. 62, no. 4, pp. 2034–2040, 2017.

[18] G. Guo, D. Yang, and R. Zhang, "Distributed trajectory optimization and platooning of vehicles to guarantee smooth traffic flow," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 1, pp. 684–695, 2023.

[19] G. Guo, Z. Zhao, and R. Zhang, "Distributed trajectory optimization and fixed-time tracking control of a group of connected vehicles," *IEEE Transactions on Vehicular Technology*, vol. 72, no. 2, pp. 1478–1487, 2023.

[20] M. O'Kelly, V. Sukhil, H. Abbas, J. Harkins, C. Kao, Y. V. Pant, R. Mangharam, D. Agarwal, M. Behl, P. Burgio, and M. Bertogna, "F1/10: An open-source autonomous cyber-physical platform," arXiv:1901.08567, 2019.

[21] J. Klapálek, A. Novák, M. Sojka, and Z. Hanzálek, "Car racing line optimization with genetic algorithm using homeomorphic transformation," in *IROS,*, 2021.

[22] V. Cataffo, G. Silano, L. Iannelli, V. Puig, and L. Glielmo, "A nonlinear model predictive control strategy for autonomous racing of scale vehicles," in *2022 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2022, pp. 100–105.

[23] J. Bhargav, J. Betz, H. Zehng, and R. Mangharam, "Deriving spatial policies for overtaking maneuvers with autonomous vehicles," in *2022 14th International Conference on COMmunication Systems and NETworkS (COMSNETS)*, 2022, pp. 859–864.

[24] J. Kabzan, M. Valls, V. Reijgwart, H. Hendrikx, C. Ehmke, M. Prajapat, A. Bühler, N. Gosala, M. Gupta, R. Sivanesan, A. Dhall, E. Chisari, N. Karnchanachari, S. Brits, M. Dangel, I. Sa, R. Dube, A. Gawel, M. Pfeiffer, and R. Siegwart, "AMZ driverless: The full autonomous racing system," arXiv:1905.05150, 2019.

[25] The Autoware Foundation, "Autoware.AUTO: Open-source software for autonomous driving technology based on ROS," 2020, https://www.autoware.ai/.

[26] M. Torngren, D. Henriksson, K.-E. Arzen, A. Cervin, and Z. Hanzalek, "Tool supporting the co-design of control systems and their real-time implementation: current status and future directions," in *IEEE Conference on Computer-Aided Control System Design*, 2006, pp. 1173–1180.

[27] P. Vacek, O. Jašek, K. Zimmermann, and T. Svoboda, "Learning to predict lidar intensities," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 4, pp. 3556–3564, 2022.

[28] T. Krajník, M. Nitsche, J. Faigl, P. Vaněk, M. Saska, L. Přeučil, T. Duckett, and M. Mejail, "A practical multirobot localization system," *Journal of Intelligent & Robotic Systems*, vol. 76, no. 3, pp. 539–562, 2014.

[29] MIT, "MIT 6.S094: Deep learning for self-driving cars," 2019, https://selfdrivingcars.mit.edu/.

[30] L. Li, K. Ota, and M. Dong, "Humanlike driving: Empirical decision-making system for autonomous vehicles," *IEEE Trans. on Vehicular Technology*, vol. 67, no. 8, pp. 6814–6823, 2018.

[31] AB Dynamics, "Path following, an upgrade for ABD steering robots," 2019, https://www.abdynamics.com/en/products/track-testing/driving-robots/steering-robots/path-following.

[32] D. González, J. Pérez, V. Milanés, and F. Nashashibi, "A review of motion planning techniques for automated vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 4, pp. 1135–1145, 2016.

[33] B. Paden, M. Cap, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," 2016, arXiv.

[34] J. Vlasak., M. Sojka., and Z. Hanzálek., "Parallel parking: Optimal entry and minimum slot dimensions," in *Proceedings of the 8th International Conference on Vehicle Technology and Intelligent Transport Systems*, 2022, pp. 300–307.

[35] J. Funke, P. Theodosis, R. Hindiyeh, G. Stanek, K. Kritatakirana, C. Gerdes, D. Langer, M. Hernandez, B. Müller-Bessler, and B. Huhnke, "Up to the limits: Autonomous Audi TTS," in *2012 IEEE Intelligent Vehicles Symposium*, Jun. 2012, pp. 541–547.

[36] T. Drage, J. Kalinowski, and T. Braunl, "Integration of drive-by-wire with navigation control for a driverless electric race car," *IEEE Intelligent Transportation Systems Magazine*, vol. 6, no. 4, pp. 23–33, 2014.

[37] R. Vesel, "Racing line optimization @ race optimal," *ACM SIGEVOlution*, vol. 7, no. 2-3, pp. 12–20, Aug. 2015.

[38] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1:43 scale RC cars," *Optimal Control Applications and Methods*, vol. 36, no. 5, pp. 628–647, 2015.

[39] R. Verschueren, M. Zanon, R. Quirynen, and M. Diehl, "Time-optimal race car driving using an online exact hessian based nonlinear MPC algorithm," in *2016 European Control Conference (ECC)*, Jun. 2016, pp. 141–147.

[40] C. You and P. Tsiotras, "High-Speed Cornering for Autonomous Off-Road Rally Racing," *IEEE Transactions on Control Systems Technology*, pp. 1–17, 2019.

[41] J. C. Kegelman, L. K. Harbott, and J. C. Gerdes, "Insights into vehicle trajectories at the handling limits: analysing open data from race car drivers," *Vehicle System Dynamics*, vol. 55, no. 2, pp. 191–207, 2017.

[42] D. Hert, T. Baca, P. Petracek, and et al., "MRS drone: A modular platform for real-world deployment of aerial multi-robot systems," *ournal of Intelligent and Robotic Systems*, vol. 108, no. 4, 2023.

[43] G. Loianno, C. Brunner, G. McGrath, and V. Kumar, "Estimation, Control, and Planning for Aggressive Flight With a Small Quadrotor With a Single Camera and IMU," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 404–411, Apr. 2017.

[44] N. D. Potdar, G. de Croon, and J. Alonso-Mora, "Online trajectory planning and control of a MAV payload system in dynamic environments," *Autonomous Robots*, vol. 44, no. 6, pp. 1065–1089, Jul. 2020.

[45] J. Záhora, M. Sojka, and Z. Hanzálek, "Perception, planning and control system for automated slalom with Porsche Panamera," in *38th FISITA 2021 World Congress*, 2021.

[46] R. Rajamani, *Vehicle Dynamics and Control*. Mechanical Engineering Series, Springer, 2012.

[47] P. Polack, F. Altché, B. d'Andréa-Novel, and A. de La Fortelle, "The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?" in *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 812–818.

[48] J. C. Kegelman, "Learning from professional race car drivers to make automated vehicles safer," Ph.D. dissertation, Stanford, 2018.

[49] J. Hooker and G. Ottosson, "Logic-based Benders decomposition," *Mathematical Programming*, vol. 96, no. 1, pp. 33–60, Apr 2003.

[50] I. Bae, J. Moon, J. Jhung, H. Suk, T. Kim, H. Park, J. Cha, J. Kim, D. Kim, and S. Kim, "Self-driving like a human driver instead of a Robocar: Personalized comfortable driving experience for autonomous vehicles," in *Machine Learning for Autonomous Driving Workshop at NeurIPS*, 2019.

[51] G. Guo and Q. Wang, "Fuel-efficient en route speed planning and tracking control of truck platoons," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 8, pp. 3091–3103, 2019.

[52] J. Wang, I. Besselink, and H. Nijmeijer, "Electric vehicle energy consumption modelling and prediction based on road information," *World Electric Vehicle Journal*, vol. 7, no. 3, pp. 447–458, 2015.

[53] M. Vybíralík and A. Novák, "Route planning for electric vehicles with charging stops and points of interest," 2022, https://dspace.cvut.cz/handle/10467/101447.

[54] M. Diehl and S. Gros, *Numerical Optimal Control*. IMTEK, 2017.

[55] J. Subosits and J. C. Gerdes, "Autonomous vehicle control for emergency maneuvers: The effect of topography," in *American Control Conference*, 2015, pp. 1405–1410.

Biographies

Zdeněk Hanzálek received his Ph.D. degree in Industrial Informatics from the Universite Paul Sabatier Toulouse, France, and his Ph.D. degree in Control Engineering from the Czech Technical University in Prague. He was with LAAS Toulouse, and with INPG Grenoble. Besides this, Zdenek founded and led the SW development team of Porsche Engineering Services in Prague and he founded Merica Company. Currently, he leads a group of 20 talented researchers and Ph.D. students. His research interests include automated cars and combinatorial optimization. He is the author or co-author of 61 journal papers and 100 conference papers.

Jiří Záhora received his master's degree in control engineering and robotics from the Czech Technical University in Prague in 2018. He then worked as an engineer in the Industrial Informatics Department on automated cars. Jiri is interested in control systems, robotics, electronic design, and automated cars.

Michal Sojka received his Ph.D. in control engineering and robotics from the Czech Technical University in Prague in 2011. He then worked as a postdoc on microkernel-based hypervisors at the Chair of Operating Systems at TU Dresden, Germany. After his return to Prague, he works as an assistant professor specializing in robotics, real-time, embedded systems, and software with a focus on autonomous driving and safety.