Optimization-Based Autonomous Racing of 1:43 Scale RC Cars

Alexander Liniger, Alexander Domahidi and Manfred Morari

Abstract

This paper describes autonomous racing of RC race cars based on mathematical optimization. Using a dynamical model of the vehicle, control inputs are computed by receding horizon based controllers, where the objective is to maximize progress on the track subject to the requirement of staying on the track and avoiding opponents. Two different control formulations are presented. The first controller employs a two-level structure, consisting of a path planner and a nonlinear model predictive controller (NMPC) for tracking. The second controller combines both tasks in one nonlinear optimization problem (NLP) following the ideas of contouring control. Linear time varying models obtained by linearization are used to build local approximations of the control NLPs in the form of convex quadratic programs (QPs) at each sampling time. The resulting QPs have a typical MPC structure and can be solved in the range of milliseconds by recent structure exploiting solvers, which is key to the real-time feasibility of the overall control scheme. Obstacle avoidance is incorporated by means of a high-level corridor planner based on dynamic programming, which generates convex constraints for the controllers according to the current position of opponents and the track layout. The control performance is investigated experimentally using 1:43 scale RC race cars, driven at speeds of more than 3 m/s and in operating regions with saturated rear tire forces (drifting). The algorithms run at 50 Hz sampling rate on embedded computing platforms, demonstrating the real-time feasibility and high performance of optimization-based approaches for autonomous racing.

I. Introduction

Autonomous car racing is a challenging task for automatic control systems due to the need for handling the vehicle close to its stability limits and in highly nonlinear operating regimes. In addition, dynamically changing racing situations require advanced path planning mechanisms with obstacle avoidance executed in real-time. Fast dynamics constrain the sampling time to be in the range of a few tens of milliseconds at most, which severely limits the admissible computational complexity of the algorithms. This situation is even more challenging if the autonomous algorithms shall be executed on simple, low-power embedded computing platforms.

In this paper, we investigate optimization-based control strategies for the task of racing an autonomous vehicle around a given track. We focus on methods that can be implemented to run in real-time on embedded control platforms, and present experimental results using 1:43 scale Kyosho *dnano* RC race cars that achieve top speeds of more than 3 m/s, which corresponds to an upscaled speed of about 465 km/h. For high performance, the proposed controllers operate the car at its friction limits, far beyond the linear region of what is typically used in other autonomous driving systems. This challenging task is generally mastered only by expert drivers with lots of training. In contrast, our approach requires merely a map of the track and a dynamical model of the car; in particular, we use a bicycle model with nonlinear tire forces, neglecting load transfer and coupling of lateral and longitudinal slip.

Two model-based control schemes are presented in this paper. First, we describe a hierarchical two-level control scheme consisting of a model-based path planner generating feasible trajectories for an underlying nonlinear model predictive control (NMPC) trajectory tracking controller. Our second approach combines both path planning and path following into one formulation, resulting in one NMPC controller that is based on a particular formulation known from contouring control [1], [2]. The objective of both approaches is to maximize the progress on the track, measured by a projection of the vehicle's position onto the center line of the track. Linear time varying (LTV) models obtained by linearization are employed to construct a

The authors are with the Automatic Control Laboratory, ETH Zurich, 8092 Zürich, Switzerland;
Emails: liniger|domahidi|morari@control.ee.ethz.ch

tractable convex optimization problem to be solved at each sampling time. Efficient interior point solvers for embedded systems, generated by FORCES [3], [4], are employed to solve the resulting optimization problems, which makes the approaches presented in this paper amenable for use on embedded systems.

We furthermore demonstrate that both schemes can easily be extended to incorporate obstacle avoidance by adjusting the constraints of the resulting optimization problems according to the racing situation. With this mechanism in place, the avoidance trajectory is optimized to yield maximal overall progress, which translates into highly effective overtaking maneuvers that are automatically planned and executed.

A. Related work

Safe autonomous driving at moderate speeds has been demonstrated for example in the context of Autonomous Highway Systems (AHS) in 1997 [5] within the Californian PATH programme, or in contests such as the DARPA Grand Challenge in 2005 [6] and the Urban Challenge in 2007 [7] by numerous research groups. In these projects, the main goal was do develop autonomous driving systems for public traffic situations. Autonomous racing has received less attention, but impressive advances have been made recently also in this discipline. For instance, a fully autonomous Audi TTS has been reported to race at high speeds in [8] using a trajectory tracking controller, which follows a pre-computed trajectory. However, to the best knowledge of the authors, fully automatic racing including obstacle avoidance in dynamic racing situations has not been demonstrated yet in practice.

From a control perspective, the fundamental building blocks for automatic racing can be grouped into three categories: drift control, reference tracking, and path planning. Research associated with the first group focuses on gaining a better understanding of the behavior of the car near the friction limit for designing safety control systems that try to prevent loss-of-control accidents. By analyzing the nonlinear car model, it can be shown that steady state motions corresponding to drifting can be generated, see e.g. [9] and [10]. As these approaches are not designed for trajectory tracking, the controlled car drifts in a circle, which represents a steady state drift equilibrium.

Reference tracking controllers usually are designed to operate the vehicle within the linear tire force region for maximum safety and robustness. Approaches based on NMPC, which allows one to incorporate the latter constraint in a systematic manner, deal with the nonlinearities of the model either directly by a nonlinear programming (NLP) solver [11], or use LTV [12] or piece-wise affine (PWA) approximations [13], resulting in a convex quadratic program (QP) or a mixed-integer QP, respectively. Approaches without optimization in real-time include nonlinear robust control [14] and flatness-based control [15], which however lack the ability to incorporate constraints. One approach that is explicitly designed for operating with saturated tire forces is [8], where the center of percussion (CoP) is used to design a state feedback steering controller for reference tracking. At the CoP, the rear tire forces do not influence the dynamics by definition, which allows for a simple linear steering controller.

In order to avoid obstacles, reference tracking schemes rely on a higher-level path planner. A simple point mass model in the high-level path planner is used in [16] to limit the computational complexity. This can be problematic if the generated trajectories are infeasible with respect to the car's physical dynamics, which can lead to preventable collisions [17]. The latter reference suggests to use a library of steady-state movements (trims) with parametrizable length, which are generated from a more complex dynamical model, and to connect them by maneuvers which allow the transition from one trim to another, similar to the idea of [18]. This path planning has the advantage that it generates feasible reference trajectories for the low level tracking controller, and that drifting trims can be included to increase the agility of the car. However, the resulting optimization problem is a mixed-integer program that is too complex to be solved within the typical sampling times on embedded platforms. Consequently, the approach is not well suited for real-time autonomous racing.

In order to avoid relying on the feasibility of the path planner, one-level approaches have been investigated e.g. in [19], where optimal trajectories and the associated open-loop control inputs for rally racing maneuvers are calculated numerically in simulation. In [16], obstacle avoidance is incorporated into

the tracking controller by using an additional cost term that penalizes coming too close to the obstacle. However, the controller is reported to perform worse than its two-level counterpart, especially if the car has to diverge from the reference trajectory. A one-level approach similar to [16] is studied in [20] in simulation, where obstacle avoidance is incorporated into the problem formulation by using a spatial reformulation and imposing constraints on the transformed variables. While in [19], [16] the solution to the NLP is obtained by a standard nonlinear solver with prohibitively long solve times, [20] uses real-time iterations [21] to achieve the low computation times needed for a real-time implementation. However, it is assumed that there is only one side where the obstacle can be overtaken, which may not be the case in practical racing situations.

An interesting alternative to optimization-based methods are sampling-based methods. For example, rapidly exploring random trees (RRTs) have been investigated in [22] and [23] to generate time optimal trajectories through a 180° curve. The advantage is that such algorithms tend to quickly find feasible trajectories. In [23], the differential flatness of the system is exploited to speed up the convergence of the algorithm, generating obstacle-free trajectories even in complex situations. However, the reported computation times are not yet low enough to allow for a real-time implementation.

B. Contribution

In this paper, we describe two novel autonomous, optimization-based racing controllers that incorporate obstacle avoidance, track constraints, actuator limitation and the ability for controlled drift in a systematic and straightforward way. Real-time feasibility at 50 Hz sampling rate is demonstrated in experimental results with fast RC cars. To the best knowledge of the authors, this is the first implementation of autonomous racing controllers with obstacle avoidance on an experimental testbed.

The fundamental idea of both approaches is to use a receding horizon controller, which maximizes progress on the track within the horizon as a performance measure. This is closely related to a time optimality criterion and allows for a systematic incorporation of obstacles and other constraints. It is particularly effective for overtaking opponents, as our controllers seek for a progress-optimal solution around the obstacles. In order to deal with track constraints and obstacle avoidance situations, we represent the feasible set for the position of the car at any time instance by a slab defined by two parallel linear inequalities. This ensures tractability of the subproblems by convex programming on one hand, and incorporation of track and obstacle constraints in a systematic manner on the other hand. To deal with non-convex situations such as overtaking an obstacle on the left or right, a high-level corridor planning algorithm supplies a set of appropriate convex constraints to the controllers. This constraint set is the result of a shortest path problem solved by dynamic programming.

The first, hierarchical control approach presented in this paper is in principle similar to [17], but its complexity is low enough to be fully implemented in real-time. Our path planner also uses trims, but only *one* is selected for the whole prediction horizon from a set of trajectories, which represent steady-state cornering conditions of the nonlinear model, gridded for velocity and steering angle. The path planner selects the trajectory with the largest progress that does not leave the track or hits any obstacle. Moving obstacles can inherently be dealt with, as new trims are generated every sampling time. A low level nonlinear reference tracking MPC controller tracks the selected trim, as in [17], but we additionally use obstacle avoidance constraints as described above, which turns out to be very effective in practice.

The second controller is based on a model predictive contouring control (MPCC) formulation [1], [2], combining path generation and path tracking into one problem. The MPCC essentially plans a progress-optimal path by taking into account the (nonlinear) projection of the vehicle's position onto the center line. The resulting controller is able to plan and to follow a path which is similar to the time-optimal path in [19] when the horizon is chosen long enough.

Both NMPC problems are approximated locally by linearizing the continuous nonlinear system dynamics around a state trajectory to obtain an LTV model. In the hierarchical approach, we linearize around the trajectory obtained from the path planner, while in the one-level approach the linearization points are given

by the shifted trajectory from the previous sampling time. We then discretize by the matrix exponential, and solve the resulting quadratic program (QP) by a tailored interior point solver, generated by FORCES [3], [4]. The computation time of the solvers scale linearly with the horizon length, hence we make use of long horizons of up to 40 time steps in the MPCC, for example. After the QP has been solved, the first control input is applied to the system, and the process is repeated at the next sampling time. This approach for solving the NMPC problems corresponds to the basic version of the real-time iterations from [21].

C. Outline

The paper is organized as follows. In Section II, the dynamical model of the car and some of its basic properties are described. In Section III-A, we present the hierarchical control approach with separate path planner and path tracking, while the combined approach is presented in Section III-B. In Section III-C, we briefly discuss our method for selecting convex constraints for the two controllers, enabling obstacle avoidance in dynamic racing situations. Section IV presents the testbed setup and experimental results for both controllers.

NOTATION

The following notation is used throughout the paper. The set of reals is denoted by $\mathbb{R}$, and the set of real column vectors of length n is denoted by $\mathbb{R}^n$. The set of nonnegative and strictly positive reals is denoted by $\mathbb{R}_+$ and $\mathbb{R}_{++}$, respectively. The set of positive definite matrices of size n is denoted by $\mathbb{S}^n_{++}$ and the set of positive semi-definite matrices of size n by $\mathbb{S}^n_+$. We define $\|x\|_P^2 \triangleq x^T P x$ for a vector $x \in \mathbb{R}^n$ and some positive definite matrix $P \in \mathbb{S}^n_{++}$. If $a \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$ are column vectors or scalars, we denote their concatenation by $(a, b) \triangleq [a^T, b^T]^T \in \mathbb{R}^{n+m}$.

II. CAR MODEL

A. Bicycle Model

The RC cars are modeled using a bicycle model as done in [9], [10], where the car is modeled as one rigid body with a mass m and an inertia I_z, and the symmetry of the car is used to reduce it to a bicycle. Only the in-plane motions are considered, i.e. the pitch and roll dynamics as well as load changes are neglected. As the used cars are rear wheel driven and do not have active brakes, the longitudinal force on the front wheel is neglected. The resulting model is shown in Figure 1 together with the resulting differential equations (1) defining the model:

$$\dot{X} = v_x \cos(\varphi) - v_y \sin(\varphi)\,, \tag{1a}$$
$$\dot{Y} = v_x \sin(\varphi) + v_y \cos(\varphi)\,, \tag{1b}$$
$$\dot{\varphi} = \omega\,, \tag{1c}$$
$$\dot{v}_x = \frac{1}{m}(F_{r,x} - F_{f,y} \sin\delta + m v_y \omega)\,, \tag{1d}$$
$$\dot{v}_y = \frac{1}{m}(F_{r,y} + F_{f,y} \cos\delta - m v_x \omega)\,, \tag{1e}$$
$$\dot{\omega} = \frac{1}{I_z}(F_{f,y} l_f \cos\delta - F_{r,y} l_r)\,. \tag{1f}$$

The equation of motion is derived around the center of gravity (CoG), where the states are the position X, Y of the CoG in the inertial frame and the angle of the car relative to the inertial frame, φ. This is the kinematic part of the model. The kinetic part of the model is derived around a body-fixed frame centered at the CoG, where the states are the longitudinal and lateral velocity of the car, v_x and v_y, and finally the yaw rate ω. The control inputs are the PWM duty cycle d of the electric drive train motor and

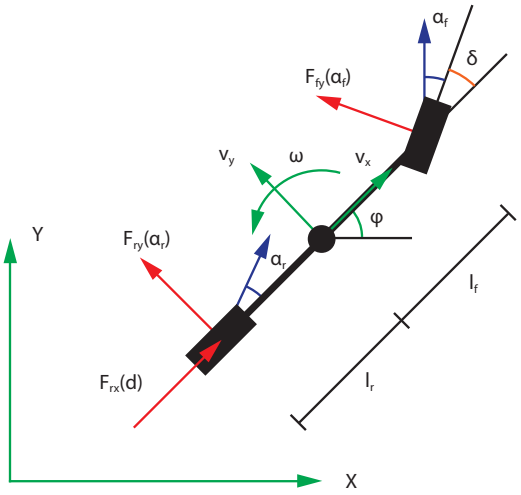

Fig. 1: Schematic drawing of the car model

the steering angle δ. The subscripts x and y indicate longitudinal and lateral forces or velocities, while r and f refer to the front and rear tires, respectively. Finally, l_f and l_r are the distance from the CoG to the front and the rear wheel.

The tire forces F model the interaction between the car and the road and are the most important part of the dynamics. As the goal is for the cars to race, the model of the tire forces has to be realistic enough to represent the car at high speeds and its handling limits. The lateral forces $F_{f,y}$ and $F_{r,y}$ are modeled using a simplified Pacejka Tire Model [24], see (2a) and (2b), which is a good approximation to measured friction curves in practice:

$$F_{f,y} = D_f \sin(C_f \arctan(B_f \alpha_f)) \quad \text{where} \quad \alpha_f = -\arctan\left(\frac{\omega l_f + v_y}{v_x}\right) + \delta\,, \tag{2a}$$

$$F_{r,y} = D_r \sin(C_r \arctan(B_r \alpha_r)) \quad \text{where} \quad \alpha_r = \arctan\left(\frac{\omega l_r - v_y}{v_x}\right)\,, \tag{2b}$$

$$F_{r,x} = (C_{m1} - C_{m2} v_x) d - C_r - C_d v_x^2\,. \tag{2c}$$

The parameters B, C and D define the exact shape of the semi-empirical curve. The longitudinal force of the rear wheel $F_{r,x}$ is modeled using a motor model for the DC electric motor as well as a friction model for the rolling resistance and the drag, cf. (2c). Due to the size of the cars, it is currently not feasible to measure the wheel speed, which makes it impossible to use combined slip models such as [9], [19].

The model is identified using a combination of stationary and dynamic identification experiments, as reported in [10]. This allows for identifying the rear wheel combined slip effects into the lateral tire friction model (2b). Thus the identified model is suitable to represent the car also at its handling limits, when the tire forces are saturated or close to saturation.

B. Stationary Velocity Analysis

For a better understanding of the model, the stationary velocities of the model are investigated. A similar, more elaborate analysis is presented in [9], [10].

The objective is to find points in the model where all accelerations are zero. This is done for different constant forward velocities $\bar{v}_x$ and constant steering angles $\bar{\delta}$, thus the problem becomes a nonlinear algebraic system of equations, with two equations ($\dot{v}_y = 0$, $\dot{\omega} = 0$) and two unknowns ($\bar{v}_y$, $\bar{\omega}$). By finding a solution to the algebraic equations for different steering angles $\bar{\delta}$ and different initial conditions, the stationary velocities for one forward velocity ($\bar{v}_y$, $\bar{\omega}$) can be calculated.

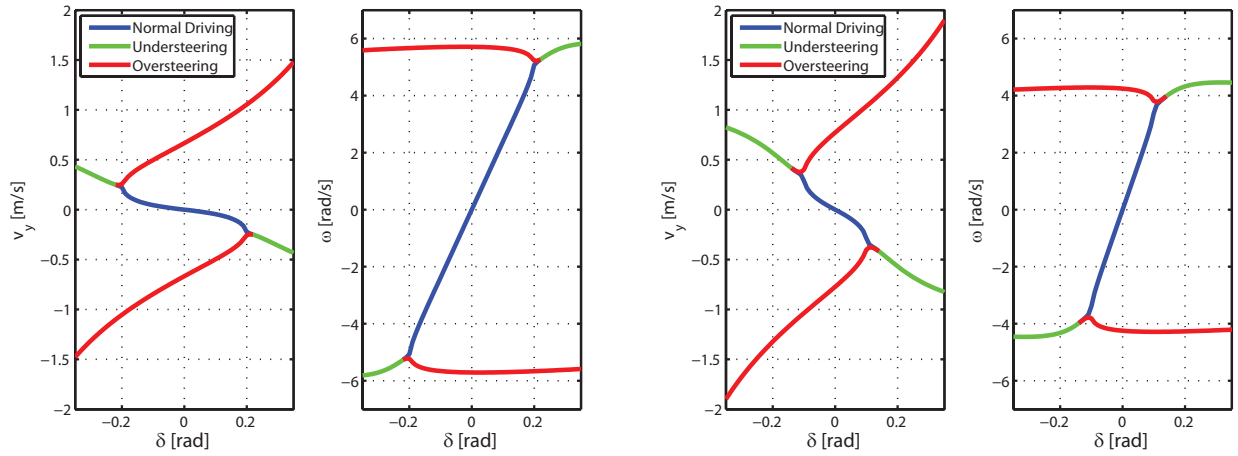

Fig. 2: Stationary velocities for $\bar{v}_x = 1.5\,\text{m/s}$ (plots on the left) and $\bar{v}_x = 2\,\text{m/s}$ (plots on the right), parameterized by the steering angle $\bar{\delta}$.

The resulting stationary velocities are shown for $\bar{v}_x = 1.5\,\text{m/s}$ and $\bar{v}_x = 2\,\text{m/s}$ in Figure 2. The resulting stationary velocities can be categorized into three different regions. The blue line is the normal driving region, which is characterized by the linear dependency of the yaw rate and the steering angle, as well as small lateral velocities. The other two regions in red and green correspond to over- and understeering points in the model. In the understeering region (green curves), the saturated front tire force law prevents the car from achieving larger yaw rates, and thus the car can not take a sharper turn. In the oversteering region (red curves), the rear tire force law is fully saturated and the car drives with a high lateral velocity, which is usually called drift.

The dynamics of the model in the different regions are different, for example in the oversteering region where the car is drifting, the steering angle can be of opposite sign than the curvature the car is driving, known as counter steering, which does not occur in the case of the normal and the understeering mode. A stability analysis of the lateral velocity model around the different stationary velocity points also shows the difference in the dynamics: While the normal driving region and the understeering region are stable, the drifting region is unstable [10].

From Figure 2 it is visible that the normal steering region is getting smaller when the car is driving faster. Thus the maximal and minimal steering angle for which the tire forces are not saturated is reduced with increasing velocities, and at the same time the maximal yaw rate decreases.

Because all velocities are constant, the movement of the car is a uniform circular movement, for which the relationship between the yaw rate, the radius R and the curvature κ is known,

$$|\omega| = \frac{v}{R} = v\kappa \quad \text{where} \quad v = \sqrt{v_x^2 + v_y^2}\,. \tag{3}$$

Consequently, all stationary velocity points correspond to the car driving in a circle with a constant radius, i.e. the model can predict drifting along a circle.

III. Autonomous Racing Control

In this main part of the paper, we present two optimization-based formulations for the task of autonomous racing for RC cars which are based on the model described in Section II. First, the hierarchical two-level approach is presented in Section III-A, followed by the one-level scheme in Section III-B. We then briefly discuss a high-level method for obstacle avoidance that is the same for both controllers in Section III-C. A summary of the algorithms is given in Section III-D.

TABLE I: Part of the Path Planning Library

Description	v_x [m/s]	v_y [m/s]	ω [rad/s]	δ [rad]
Left	1.75	0.1185	-2.7895	-0.1000
Straight	1.75	0	0	0
Slight Right	1.75	-0.0486	1.3849	0.05
Hard Right	1.75	-0.3163	4.4350	0.16
Hard Left Drift	1.75	1.1860	-4.8709	0.2

A. Hierarchical Receding Horizon Controller

In our hierarchical control approach, a high-level path planner finds a progress-optimal, feasible trajectory within a finite number of possibilities and for a horizon of N sampling times. This trajectory is then tracked by an MPC controller employing soft-constraints to ensure feasibility of the low-level optimization at all times. This process is repeated in a receding-horizon fashion, hence we call the controller Hierarchical Receding Horizon Controller (HRHC) in what follows. Details on the individual components are given in the following.

1) Path Planning.: The purpose of the path planner is to generate a trajectory with maximal progress which is feasible for the car's dynamics. It is based on gridding the stationary velocities of the nonlinear system described in Section II-B. By solving (1) for the stationary lateral velocity points given a set of different longitudinal velocities $\bar{v}_x$ and steering angles $\bar{\delta}$, a library of zero acceleration points is generated. This includes stationary velocities from the normal as well as from the drifting region. Table I shows an excerpt of such a library. The library used in our experiments consists of $N_{\bar{v}} = 95$ stationary points, out of which 26 correspond to drifting equilibria. The stationary points are selected by uniformly gridding the longitudinal velocity v_x between 0.5 and 3.5 m/s, with steps of 0.25 m/s. For each $\bar{v}_x$, about five to nine points are selected in the normal driving region, and up to four drifting points are selected for forward velocities between 1.5 and 2.25 m/s. Note that the library does not change during run-time, hence it can be generated offline.

To generate trajectories for the whole horizon, the stationary points are integrated. This is computationally cheap under the assumption that the stationary velocity can be reached within one time step, since in this case all accelerations are zero. This procedure allows us to generate reference trajectories also for unstable drifting points. The integration of the reasonable stationary velocity points leads to a countable set of possible trajectories with a constant turning radius over the horizon. A visualization of such a set of candidate trajectories is shown in Figure 3.

We now proceed with finding the best trajectory among the candidates obtained by gridding and integration, which is equivalent to solving the following optimization problem:

$$\theta^\star \triangleq \max_{j \in 1,\dots,N_{\bar{v}}} \ \mathcal{P}\left(X_N^j, Y_N^j\right), \tag{4a}$$

$$\text{s.t. } x_0^j = x, \tag{4b}$$

$$x_{k+1}^j = f_{km}(x_k^j, \bar{v}^j), \qquad k = 1, \dots, N \tag{4c}$$

$$x_k^j \in \mathcal{X}_{track}, \qquad k = 1, \dots, N \tag{4d}$$

$$\bar{v}^j \in \mathcal{V}(v), \tag{4e}$$

where $x_k \triangleq (X_k, Y_k, \varphi_k)$ is the position and orientation at time step k generated by integrating the kinematic part of the bicycle model using the stationary velocity $\bar{v}^j \triangleq (\bar{v}_x^j, \bar{v}_y^j, \bar{\omega}^j)$ from the library of stationary points. The integration is denoted by the discrete time version of the model, f_{km} in (4c). The objective (4a) is to find the trajectory with the largest progress on the track, which is measured by the projection operator $\mathcal{P} : \mathbb{R}^2 \to [0,\ L]$ that calculates the scalar projection of the position X_k^j, Y_k^j onto the piecewise linear center line parameterized by the arc length $\theta \in [0, L]$, where L is the length of the center

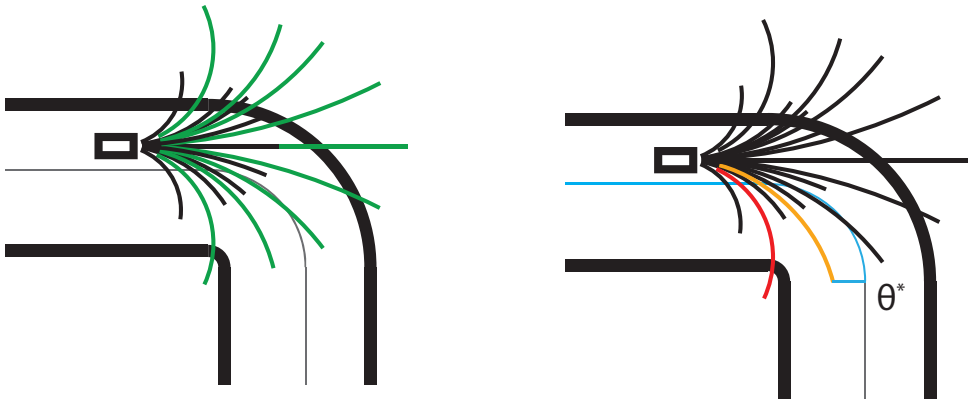

Fig. 3: Left: Generation of reference trajectories for two different longitudinal velocities (black: slow, green: fast). Right: Solution of the path planner. The red trajectory has the largest progress but leaves the track, so that the orange one is the optimal trajectory maximizing θ_N.

line. All positions of the trajectory x_k^j, $k = 0, \ldots, N$, need to lie in the set $\mathcal{X}_{track} \subset \mathbb{R}^2$ (4d) which is the set of all admissible positions inside the track boundaries. Finally, we demand that the stationary velocity $\bar{v}^j$ lies in the set $\mathcal{V} \subset \mathbb{R}^3$ (4e), which depends on the current velocity measurement $v \triangleq (v_x, v_y, \omega)$ and enforces that the planned velocity is reasonably close to the measured velocity, which helps to deal with the assumption that the stationary velocity can be reached within one time step:

$$\mathcal{V}(v) \triangleq \begin{cases} \{(\bar{v}_x, \bar{v}_y, \bar{\omega}) \in \mathbb{R}^3 \mid \rho \geq |v_x - \bar{v}_x|\} & \text{if } |\bar{v}_y| \leq \nu \\ \{(\bar{v}_x, \bar{v}_y, \bar{\omega}) \in \mathbb{R}^3 \mid \sigma_x \geq |v_x - \bar{v}_x|\,,\ \sigma_y \geq |v_y - \bar{v}_y|\,,\ \sigma_\omega \geq |\omega - \bar{\omega}|\} & \text{otherwise} \end{cases}, \tag{5}$$

where the two cases correspond to selecting non-drift or drift trajectories, respectively, based on the lateral steady state velocity $\bar{v}_y$. The tuning parameters $\nu, \rho, \sigma_x, \sigma_y, \sigma_\omega$ are used to determine the behavior of the path planner in terms of selecting candidate trajectories. Choosing parameters that lead to too loose constraints in (5) result typically in physically impossible trajectories, which can result in crashes. Conversely, constraints that are too tight limit the performance of the controller, as the planned trajectories are too conservative.

Problem (4) is an integer program with the decision variable j and solved by enumeration in practice by checking the discrete positions against local convex inner approximations of the track set, which is reasonable due to the relatively small number of trajectories (about one hundred in our case). The result of the optimization problem is visualized in Figure 3. The path planning allows to efficiently handle drift, as a drifting trajectory is selected if it yields the largest progress.

The path planner has some limitations. Firstly, on one hand, the horizon length has to be quite short, otherwise the planned trajectories become circles and always yield infeasibilities with respect to the track constraints, even if the selected velocity would be well suited for the current state. A horizon which is too short, on the other hand, leads to poor performance as the path planner recognizes a forthcoming curve too late. The second problem is that the whole horizon has the same velocity, which can lead to problems in complicated curve combinations such as chicanes.

2) Model Predictive Reference Tracking.: The reference trajectory from the path planner is generated under simplifying assumptions, thus it is not possible to directly apply the controls which correspond to the stationary velocity point. In order to efficiently track the reference the following MPC formulation is

used, penalizing the deviation from the reference trajectory using a quadratic function:

$$\min_{x,u,s} \|x_N - x_N^{\text{ref}}\|_P^2 + p\|s_N\|_\infty + \sum_{k=0}^{N-1} \|x_k - x_k^{\text{ref}}\|_Q^2 + q\|s_k\|_\infty + \|u_k - u_k^{\text{ref}}\|_R^2 \tag{6a}$$

$$\text{s.t. } x_0 = x\,, \tag{6b}$$

$$x_{k+1} = A_k x_k + B_k u_k + g_k\,, \qquad k = 0, ..., N-1 \tag{6c}$$

$$F_k x_k \leq f_k + s_k\,, \qquad k = 1, ..., N \tag{6d}$$

$$s_k \geq 0\,, \qquad k = 1, ..., N \tag{6e}$$

$$\underline{x} \leq x_k \leq \bar{x}\,, \qquad k = 1, ..., N \tag{6f}$$

$$\underline{u} \leq u_k \leq \bar{u}\,, \qquad k = 0, ..., N-1 \tag{6g}$$

where $Q \in \mathbb{S}_{++}^6$, $R \in \mathbb{S}_{++}^2$, $P \in \mathbb{S}_{++}^6$ are tuning matrices and $p, q \in \mathbb{R}_{++}$ are penalties for the soft constraints (6d), which are discussed below. To capture as much of the nonlinear model as possible while maintaining computational tractability, the model is linearized around the reference trajectory, cf. (6c). It is necessary to linearize around a trajectory rather than a single operating point, since both position and orientation can vary significantly over the horizon. The reference trajectory is by construction feasible with respect to the track. In order to also guarantee that the trajectory of the optimal MPC solution satisfies track constraints, the X and Y states are limited to stay inside the track if possible. This is achieved by limiting each point in the horizon to lie within two parallel half spaces (6d), which leads to two affine inequality constraints per time step (one for the right and one for the left border), resulting in a convex feasible set. Figure 4 depicts these border constraints (6d), which we formulate as soft constraints in order to avoid infeasibility problems in practice. By using an exact penalty function (an infinity norm in our case), we ensure that the resulting optimization problem is always feasible, and that the original solution of the hard constrained problem is recovered in case it would admit a solution [25], [26]. Finally, the control inputs are limited within their physical constraints, and all other states are limited within reasonable bounds to avoid convergence problems of the solver due to free variables (6f), (6g).

Problem (6) can be reformulated as a convex QP and thus efficiently solved. In this work FORCES [3], [4], is employed to generate tailored C-code that solves instances of (6) where parameters are the initial state x, the matrices defining the dynamics along the reference trajectory (A_k, B_k, g_k) as well as the changing halfspace constraints F_k, f_k.

B. Model Predictive Contouring Control

Contouring control is used in various industrial applications such as machine tools for milling and turning [27] or laser profiling [28]. In these applications, the challenge is to compute inputs that control the movement of the tool along a reference path. The latter is given only in spatial coordinates; the associated velocities, angles, etc. are calculated and imposed by the control algorithm. This is different from tracking controllers in that the controller has more freedom to determine the state trajectories to follow the given path, for example to schedule the velocity, which in tracking is defined by the reference trajectory.

The contour following problem can be formulated in a predictive control framework to incorporate constraints, see e.g. [1]. In this work, we adapt the particular formulation of the model predictive contouring control (MPCC) framework from [2] to obtain a high-performance controller for autonomous racing, which maximizes the travelled distance on the reference path within the prediction horizon. In our experiments, we use the center line as a reference path, but employ it merely as a measure of progress by selecting low weights on the tracking (contouring) error. As a result, the driven trajectory is very similar to those driven by expert drivers. The advantage of this approach is that path planning and path tracking can be combined into one nonlinear optimization problem, which can be solved in real-time by approximating the NLP using local convex QP approximations at each sampling time [21]. The resulting QPs can be

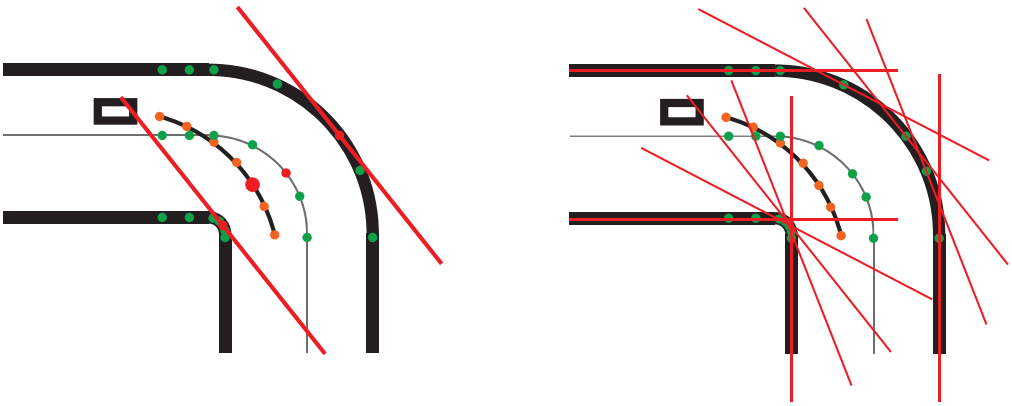

Fig. 4: Left: Halfspace constraints (red lines) of one point in the horizon (large red dot). Right: Resulting halfspaces (red lines) of the whole horizon, approximating the track. The green points on the center line correspond to the closest point on the discrete center line, for each point in the horizon. The points on the borders are the projection of the center line points on the borders.

formulated with multistage structure, which is exploited by FORCES [3], [4] to obtain solution times in the range of a few milliseconds.

Before posing the MPCC problem, a few preliminaries are introduced such as the parameterization of the reference path and the definition of useful error measures.

1) Parameterization of Reference Trajectory.: The reference path is parameterized by its arc length $\theta \in [0,\, L]$ using third order spline polynomials, where L is the total length. The splines are obtained by an offline fitting of the center line. Using this parameterization, we can obtain any point $X^{\text{ref}}(\theta)$, $Y^{\text{ref}}(\theta)$ on the center line by evaluating a third order polynomial for its argument θ. The angle of the tangent to the path at the reference point with respect to the X-axis,

$$\Phi(\theta) \triangleq \arctan\left\{\frac{\partial Y^{\text{ref}}(\theta)}{\partial X^{\text{ref}}(\theta)}\right\}, \tag{7}$$

is also readily available. This parameterization leads to an accurate interpolation within the known points of the reference path, and it is more accurate than the piece-wise linear parameterization employed for the HRHC in Section III-A, which only uses a linear interpolation between the points.

2) Error measures.: In order to formulate the MPCC problem, error measures are needed that define the deviation of the car's current position X, Y from the desired reference point $X^{\text{ref}}(\theta)$, $Y^{\text{ref}}(\theta)$. We use the same definitions as in [2], but give another derivation here. Let $\mathcal{P} : \mathbb{R}^2 \to [0,\, L]$ be a projection operator on the reference trajectory defined by

$$\mathcal{P}(X, Y) \triangleq \arg\min_{\theta}(X - X^{\text{ref}}(\theta))^2 + (Y - Y^{\text{ref}}(\theta))^2\,. \tag{8}$$

For brevity, we define $\theta_{\mathcal{P}} \triangleq \mathcal{P}(X, Y)$. The orthogonal distance of the car from the reference path is then given by the *contouring error*

$$e^c(X, Y, \theta_{\mathcal{P}}) \triangleq \sin\left(\Phi(\theta_{\mathcal{P}})\right)\left(X - X^{\text{ref}}(\theta_{\mathcal{P}})\right) - \cos\left(\Phi(\theta_{\mathcal{P}})\right)\left(Y - Y^{\text{ref}}(\theta_{\mathcal{P}})\right), \tag{9}$$

where $\Phi(\cdot)$ is defined in (7). The contouring error is depicted in the left picture in Figure 5.

The projection operator (8) is not well suited for use within online optimization algorithms, as it resembles an optimization problem itself. Thus an approximation $\theta_{\mathcal{A}}$ of $\theta_{\mathcal{P}}$ is introduced, which is an independent variable determined by the controller. For this approximation to be useful, it is necessary to link $\theta_{\mathcal{A}}$ to $\theta_{\mathcal{P}}$ via the *lag error*

$$e^l(X, Y, \theta_{\mathcal{A}}) \triangleq |\theta_{\mathcal{A}} - \theta_{\mathcal{P}}|\,, \tag{10}$$

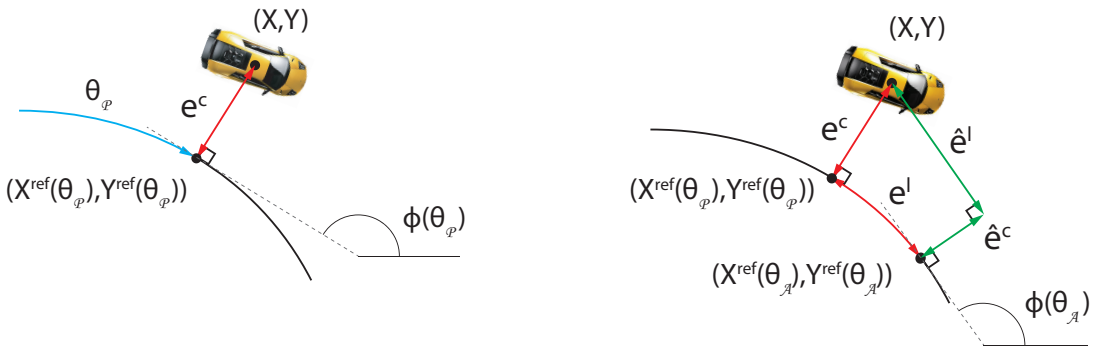

Fig. 5: Contouring error e^c (left) and lag error e^l (right) with linear approximations $\hat{e}^c$ and $\hat{e}^l$.

which measures the quality of the approximation. The lag error is depicted in the right picture in Figure 5 (red segment on reference path).

In order to be independent of the projection operator (8), the contouring (9) and the lag error (10) can be approximated as a function of the position X, Y and the approximate projection $\theta_{\mathcal{A}}$, which are variables that can be controlled by the MPCC controller:

$$e^c \approx \hat{e}^c(X, Y, \theta_{\mathcal{A}}) \triangleq \sin\left(\Phi(\theta_{\mathcal{A}})\right)\left(X - X^{\text{ref}}(\theta_{\mathcal{A}})\right) - \cos\left(\Phi(\theta_{\mathcal{A}})\right)\left(Y - Y^{\text{ref}}(\theta_{\mathcal{A}})\right), \tag{11a}$$

$$e^l \approx \hat{e}^l(X, Y, \theta_{\mathcal{A}}) \triangleq -\cos\left(\Phi(\theta_{\mathcal{A}})\right)\left(X - X^{\text{ref}}(\theta_{\mathcal{A}})\right) - \sin\left(\Phi(\theta_{\mathcal{A}})\right)\left(Y - Y^{\text{ref}}(\theta_{\mathcal{A}})\right). \tag{11b}$$

The approximate contouring error $\hat{e}^c$ (11a) and the approximate lag error $\hat{e}^l$ (11b) are defined as the orthogonal and tangential component of the error between $X^{\text{ref}}(\theta_{\mathcal{A}})$, $Y^{\text{ref}}(\theta_{\mathcal{A}})$ and the position X, Y, see the right picture in Figure 5.

3) MPCC problem.: With these error measures in place, we now formulate the model predictive contouring control problem for autonomous racing. The objective is a trade-off between the quality of path-following (low contouring error) and the amount of progress achieved over a finite horizon of N sampling times, subject to model dynamics, track- and input constraints:

$$\min \sum_{k=1}^{N} \left\{ \|e_k^c(X_k, Y_k, \theta_{\mathcal{P}})\|_{q_c}^2 \right\} - \gamma \theta_{\mathcal{P},N} \tag{12a}$$

$$\text{s.t. } x_0 = x\,, \tag{12b}$$

$$x_{k+1} = f(x_k, u_k)\,, \qquad k = 0, ..., N-1 \tag{12c}$$

$$F_k x_k \leq f_k\,, \qquad k = 1, ..., N \tag{12d}$$

$$\underline{x} \leq x_k \leq \bar{x}\,, \qquad k = 1, ..., N \tag{12e}$$

$$\underline{u} \leq u_k \leq \bar{u}\,, \qquad k = 0, ..., N-1 \tag{12f}$$

where $X_k\,, Y_k$ is the position of the car at time step k determined by the nonlinear model f in (12c), which is the discrete-time version of (1) with piece-wise constant control inputs. The contouring error $e_k^c(X_k, Y_k, \theta_{\mathcal{P}})$ in the objective (12a) is defined in (9), and $\theta_{\mathcal{P},k}$ is the associated path parameter such that $X^{\text{ref}}(\theta_{\mathcal{P},k})$, $Y^{\text{ref}}(\theta_{\mathcal{P},k})$ is the orthogonal projection of X_k, Y_k onto the reference path. The two objectives (maximum progress and tight path following) are traded off by the weights $\gamma \in \mathbb{R}_{++}$ and $q_c \in \mathbb{R}_{++}$, respectively. Constraints (12d) are the parallel half space constraints for containing the position in the allowed corridor as described in Section III-A2, while (12e), (12f) corresponds to (6f), (6g), limiting states and inputs to physically admissible values.

Due to the implicit dependency of the objective (12a) on the projection operator (8), optimization problem (12) is a bi-level NLP, which is too complex to solve in real-time. Thus the idea is to use the approximate projection $\theta_{\mathcal{A},k}$ instead of $\theta_{\mathcal{P},k}$ as introduced in Section III-B2, and to control the

approximation quality by adding a cost on the lag error (10) to the objective. In order to allow for forming the lag error at each time step in the prediction horizon, it is necessary to introduce an integrator state with dynamics $\theta_{\mathcal{A},k+1} = \theta_{\mathcal{A},k} + v_k/T_s$, where v_k can be interpreted as the *projected velocity*, and $\theta_{\mathcal{A},k}$ as the *state of progress* at time k, respectively. This approximation reduces (12) to an optimal control problem in form of an NLP that is amenable for a real-time implementation:

$$\min \sum_{k=1}^{N} \|\hat{e}_k^c(X_k, Y_k, \theta_{\mathcal{A},k})\|_{q_c}^2 + \|\hat{e}_k^l(X_k, Y_k, \theta_{\mathcal{A},k})\|_{q_l}^2 - \gamma v_k T_s + \|\Delta u_k\|_{R_u}^2 + \|\Delta v_k\|_{R_v}^2 - \gamma v_0 T_s \tag{13a}$$

$$\text{s.t. (12b)}\,, \quad \theta_0 = \theta\,, \tag{13b}$$

$$\text{(12c)}\,, \quad \theta_{\mathcal{A},k+1} = \theta_{\mathcal{A},k} + \frac{v_k}{T_s}\,, \qquad k = 0, ..., N-1 \tag{13c}$$

$$\text{(12d)}\,, \quad \text{(12e)}\,, \quad 0 \leq \theta_k \leq L\,, \qquad k = 1, ..., N \tag{13d}$$

$$\text{(12f)}\,, \quad 0 \leq v_k \leq \bar{v}\,, \qquad k = 0, ..., N \tag{13e}$$

where $\Delta u_k \triangleq u_k - u_{k-1}$ and $\Delta v_k \triangleq v_k - v_{k-1}$. Note that the approximate contouring error $\hat{e}_k^c(X_k, Y_k, \theta_{\mathcal{A},k})$ (11a) is used in the objective (13a). Furthermore, we have replaced the maximization of the final progress measure, $\theta_{\mathcal{P},N}$, by $\sum_{k=0}^{N-1} v_k T_s$, which is equivalent if the approximation is accurate. Sensible lower and upper bounds on θ_k and v_k are imposed to avoid spurious solutions of the NLP, with $\bar{v}$ denoting the largest possible progress per sampling time. The cost on the lag error $\hat{e}_k^l(X_k, Y_k, \theta_{\mathcal{A},k})$ in (13a) links the state of progress to the dynamics of the car. To ensure an accurate progress approximation and thus a strong coupling between the cost function and the car model, the weight on the lag error $q_l \in \mathbb{R}_{++}$ is chosen high as suggested in [2]. Furthermore, a cost term on the rate of change of the inputs is added to the objective (13a) in order to penalize fast changing controls, which helps to obtain smooth control inputs in order to prevent amplifying unmodeled dynamics.

4) Solving the MPCC problem.: In order to solve the nonlinear optimal control problem (13) in real-time, local convex approximations of (13) in form of the following QPs are built at each sampling time by linearization of nonlinear terms:

$$\min_{x,u,\theta,v,s} \sum_{k=1}^{N} \begin{bmatrix} x_k \\ \theta_{\mathcal{A},k} \end{bmatrix}^T \Gamma_k \begin{bmatrix} x_k \\ \theta_{\mathcal{A},k} \end{bmatrix} + c_k^T \begin{bmatrix} x_k \\ \theta_{\mathcal{A},k} \end{bmatrix} - \gamma v_k T_s + \begin{bmatrix} \Delta u_k \\ \Delta v_k \end{bmatrix}^T R \begin{bmatrix} \Delta u_k \\ \Delta v_k \end{bmatrix} + q\|s_k\|_\infty - \gamma v_0 T_s \tag{14a}$$

$$\text{s.t. } x_0 = x\,, \quad \theta_{\mathcal{A},0} = \theta\,, \tag{14b}$$

$$x_{k+1} = A_k x_k + B_k u_k + g_k\,, \qquad k = 0, ..., N-1 \tag{14c}$$

$$\theta_{\mathcal{A},k+1} = \theta_{\mathcal{A},k} + \frac{v_k}{T_s}\,, \qquad k = 0, ..., N-1 \tag{14d}$$

$$F_k x_k \leq f_k + s_k\,, \qquad k = 1, ..., N \tag{14e}$$

$$s_k \geq 0\,, \qquad k = 1, ..., N \tag{14f}$$

$$\underline{x} \leq x_k \leq \bar{x}\,, \quad 0 \leq \theta_{\mathcal{A},k} \leq L\,, \qquad k = 1, ..., N \tag{14g}$$

$$\underline{u} \leq u_k \leq \bar{u}\,, \quad 0 \leq v_k \leq \bar{v}\,, \qquad k = 0, ..., N \tag{14h}$$

where $\Gamma_k \in \mathbb{S}_+^7$ is formed by the quadratic part of the linearized contouring and lag error cost function from (13a) and $c_k \in \mathbb{R}^7$ stems from the linear part, respectively. In order to keep the linearization error small, we use an LTV approximation of the dynamics (1) as well as of the contouring and lag errors (11). Each nonlinear function is linearized around the output of the last QP iteration shifted by one stage. The measurement $x = x_0$ is used as the first linearization point, and the last input of the previous iteration is kept constant to generate a new last input. The linearization point for the terminal state x_N is calculated by simulating the nonlinear model for one time step. The track or corridor constraints (14e) are formulated by two half space constraints tangential to the track per time step, as described in Section III-A. These

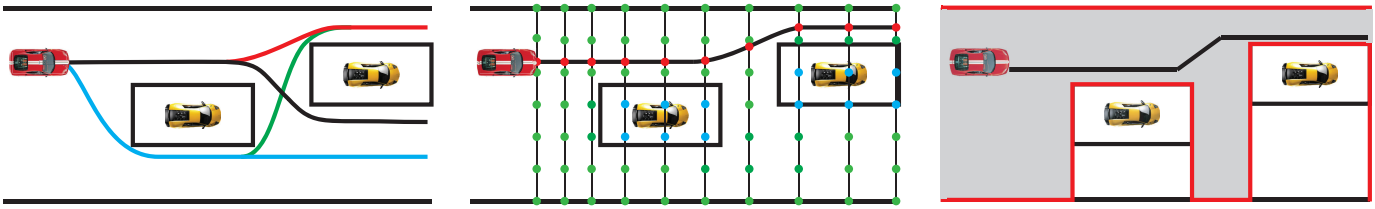

Fig. 6: Left: Combinatorial nature of the overtaking problem, as it is possible to overtake each opponent on the left or the right side. Center: Spatial-temporal grid for the shortest path problem solved by dynamic programming, where it is not allowed to visit the blue grid points. Right: Optimal path of the DP (in black) and the resulting corridor, which is issued to the HRHC (Section III-A) or MPCC (Section III-B).

constraints are formulated as soft constraints, with slack variables $s_k \in \mathbb{R}^2$ and a corresponding infinity-norm penalty in the objective (14a) weighted by $q \in \mathbb{R}_{++}$, which is chosen quite high to recover the behavior of the hard constrained problem whenever possible [25], [26].

Note that the described re-linearization scheme is essentially the basic real-time iteration from [21], where a nonlinear continuous-time optimal control problem is solved using a multiple shooting technique, with an exponential map as integrator, and in every time step only one step of a sequential QP (SQP) scheme is performed. Hence the convergence properties that are known for the real-time iteration [29] hold also for the MPCC problem (13).

The local QP (14) is solved using FORCES [3], [4]. Due to the rate costs, the problem has to be lifted to obtain the multistage structure, for which FORCES has been designed, by introducing a copy of the previous control input at each stage.

C. Obstacle Avoidance

The two controllers presented in this section plan a path within the track, or a corridor, by taking into account constraints (4d), (6f) or (14e) on the position of the car. Thus it is possible to include obstacle avoidance by adapting this corridor online depending on the current constellation of opponents. Generating the optimal corridor based on the position of the obstacles is in general hard, because of the fundamental problem that the decision on which side to overtake is non-convex and of combinatorial nature if more than one opponent is involved. This situation is depicted in Figure 6 in the left picture.

In this work, we employ a high level path planner based on dynamic programming (DP) to decide on which side to overtake the obstacles. Based on the result of the DP, the adapted corridor is given to the controllers of Section III-A or Section III-B. The high level path planner solves a shortest path problem [30] on a spatial-temporal grid, cf. Figure 6. To incorporate the planned path of the lower level controller into the DP, the grid is built up based on the last output of the path planner in the case of the HRHC or the last QP solution in the case of the MPCC. The DP minimizes the travelled distance, and additionally the deviation from the last plan of the lower level controller, such that the DP path is close to the state prediction of the lower level controller. This ensures that the linearizations stay approximately valid most of the time. Based on the optimal trajectory from the DP, the corridor constraints can easily be identified, see the right picture in Figure 6.

Due to space restrictions, we do not go further into detail, but many references exist that solve the corridor problem with different approaches. For example, a similar problem had to be solved during the DARPA Urban challenge where the goal was autonomous driving in city traffic. In [7] several approaches to this problem are presented, for example based on model predictive trajectory generation algorithms [31], modified $A^\star$-algorithms [32] or RRTs [33].

D. Summary

In this main part of the paper, we have presented two approaches for automatic racing. Both approaches are model-based and maximize progress within a given time horizon. They directly incorporate track

and obstacle constraints by using two parallel affine inequalities (slabs) representing the feasible set of positions at each time. The major difference between the approaches is that the first (cf. Section III-A and Algorithm Ia) is a two-level approach with a separate path planning mechanism combined with a reference tracking MPC controller, while the second approach (cf. Section III-B and Algorithm Ib) formulates both tasks (path planning and path tracking) into one nonlinear optimal control problem based on model predictive contouring control.

Algorithm 1 HRHC and MPCC algorithm

(Ia) HRHC

1: get current position and velocities
2: compute delay compensation
3: **function** BORDERADJUSTMENT
4: shift old planned path by one stage
5: get position of opposing cars
6: run DP and get new borders (Sec. III-C)
7: **end function**
8: run path planner (4)
9: linearize and discretize the model, i.e. build problem (6)
10: solve QP using FORCES (6)
11: send first control at the end of time slot
12: **go to** step 1

(Ib) MPCC

1: get current position and velocities
2: compute delay compensation
3: augment old QP output (Section III-B4)
4: **function** BORDERADJUSTMENT
5: get position of opposing cars
6: run DP and get new borders (Sec. III-C)
7: **end function**
8: linearize and discretize the model and the cost function, i.e. build problem (14)
9: solve QP using FORCES (14)
10: send first control at the end of time slot
11: **go to** step 1

Both controllers send the new control input at the end of the sampling period. In order to compensate for this fixed delay, the nonlinear model is simulated forward for this time interval at the beginning of the algorithm, see line 2 of Algorithm Ia and Ib. The nonlinear model is integrated using a second-order Runge-Kutta method with the current state estimate from the Kalman filter as the initial condition.

IV. RESULTS

In this section the performance of both controllers is evaluated on an experimental testbed using 1:43 scale RC race cars. Details on the experimental setup, the particular implementation and closed-loop performance are given in the following.

A. Experimental Setup

For our experiments, we use the Kyosho *dnano* cars, which are quite sophisticated with a front and rear suspension and a rear axle differential. The cars are able to reach speeds of over 3 m/s on straights, which corresponds to an upscaled speed of about 465 km/h. The wide speed range, the high maximal forward velocity and the small scale introduce further complexity which is not present in full size testbeds.

The testbed further consists of an infrared camera tracking system, a control board and a custom built race track of length $L = 18.43\,\text{m}$ (measured at the center line), see Figure 7. A PointGrey Flea3 camera captures 100 frames per second with an accuracy of below 4 mm. A wide angle lens with an infrared filter is used to capture the 4 by 4 meter track. Reflecting markers are arranged on the cars in different unique patterns and used to identify the car and to measure the position and angle of the cars. An Extended Kalman Filter (EKF) is used for state estimation, filtering the vision data and estimating the longitudinal and lateral velocity as well as the yaw rate. The vision data of all cars is passed over an Ethernet connection to a control PC, which runs one of the two control algorithms described in Section III together with the corridor planner solving the DP problem. The calculated inputs are then sent via Bluetooth to the car. As the original electronics of the *dnano* cars does not feature an open digital communication link, we have replaced it by a custom made PCB, featuring a Bluetooth chip, current and voltage sensors, H-bridges

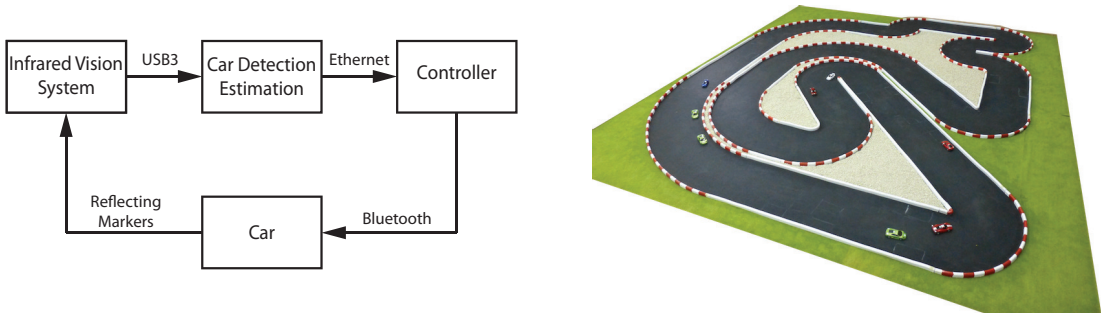

Fig. 7: Visualization of the control loop and a picture of the used track

for DC motor actuation, and an ARM Cortex M4 microcontroller for the low level control loops such as steering servo and traction motor control.

B. Implementation

The HRHC is implemented in ANSI C on an embedded platform using an ARM A9 chip (Exynos 4412) at 1.7 GHz. The chip is identical to the one used in Samsung Galaxy S3 smartphones. The runtime environment was Ubuntu Linux version 12.11, and the code was compiled with the GCC compiler 4.6.3 with option `-O2`. The sampling time of the controller is $T_s = 20\,\text{ms}$, while the discretization time of the path planner and the tracking MPC controller is 25 ms. The horizon length is $N = 14$ time steps, which gives an ahead prediction of $0.35\,\text{s}$. The tracking MPC controller results in a multistage QP with 9 variables per stage, i.e. 132 variables in total. We use a library of 95 candidate trajectories, out of which 26 are drifting stationary velocity points.

The MPCC does not have the limitation of the HRHC path planner, or in other words, the acceleration is not zero over the whole horizon, thus it is possible to use longer horizons which should increase the performance. Hence the controller runs with a horizon length of $N = 40$ with a sampling time of $T_s = 20\,\text{ms}$, which corresponds to a $0.8\,\text{s}$ ahead prediction. With the re-linearization method presented in Section III-B4, the discretization and sampling time have to be the same, as the re-linearization is based on the last QP solution. The resulting QP has 14 variables per stage and 570 in total.

The MPCC is implemented on a newer gereration embedded platform using an Exynos 5410 chip based on the ARM Cortex A15 architecture and running at 1.6 GHz. The chip is identical to the one used in the Korean version of the Samsung Galaxy S4 smartphone. The embedded board runs Ubuntu Linux version 12.04. The ANSI C code of the MPCC controller has been compiled with GCC 4.6.3 with option `-O3`. To improve the convergence of the QP, the problem data is scaled to lie between -1 and $+1$, and the objective (Hessian and linear term) is scaled by a factor of 0.1 to reduce large entries occurring due to the lag error term. Note that these scalings do not change the minimizer. The optimality criteria of the FORCES solver are adjusted such as to terminate once a sufficiently good solution has been found, with the residuals of the equality and inequality constraints set to 10^{-5}, and with the duality gap set to 10^{-4}.

C. Single Car Racing

The driven trajectories for a single car are depicted in Figure 8, with the velocity profile encoded in colors. Depicted are three laps, for which the HRHC achieves a lap time between 9.5 to 9.8 s while the MPCC yields lap times between 8.9 and 9.2 s.

To better understand the driven trajectory of the HRHC, the path planner of the HRHC has to be analyzed. The path planner has a comparably short prediction horizon, and all velocities over the whole horizon are constant, which leads to turns of constant radius. Such a path planner works fine for 90° and even up to 180° curves. However, the car tends to drive to the outer border after the curve, as this allows driving a wider radius at a faster longitudinal velocity and thus maximizes the progress. However, this is

a limitation if the car should drive a combination of curves, where the position of the car at the end of the first curve is essential for a low overall time. This problem can be seen in Figure 8, for example in the chicane in the lower left corner. Furthermore, due to the short lookahead, the HRHC path planner is not able to prevent such an ill positioning relative to a curve ahead. Due to the suboptimal position relative to the curve, the controller has to brake more compared to a car on the ideal line and it is even possible that the car touches the borders, see the narrow 180° curve in the center of the track in Figure 9.

These limitations are not present in the MPCC approach, where a comparably long horizon is employed. Unlike the HRHC, the velocity is not fixed over the horizon but computed for each time step as the result of the NLP. As a consequence of these features, the MPCC is able to plan a path even through a complicated combination of curves. This results in a trajectory which is closer to an ideal line in a least curvature sense, especially through complicated curves like the chicane in the lower left corner of Figure 8. In this curve combination, the MPCC achieves velocities not smaller than $1\,\mathrm{m/s}$, while the HRHC has to reduce the velocity to nearly $0.5\,\mathrm{m/s}$ to drive through the chicane, and additionally causes the car to drive an overall longer distance.

However, the MPCC has some disadvantages compared to the HRHC. In our current implementation, the MPCC still tracks the center line, which can be seen on the first straight, after the top left corner where a movement to the center line is clearly visible in Figure 8. This is clearly due to the objective function of the MPCC, and cannot be completely avoided, even if the contouring cost is small. The HRHC on the other hand does not have any cost related to the center line and only uses it as a measure of progress, thus the car drives straight in the aforementioned segment of the track. The second disadvantage can be seen in the S-curve at lower end of the track, where the MPCC has a small S shape in the driven trajectory. The HRHC on the other hand shows that the section can be driven completely straight, which gives a speed benefit. This can be most probably also attributed to the contouring error penalty, for which it would be beneficial to drive an S-shape in order to follow the center line.

In summary, our results indicate that the HRHC is limited by the path planner, and the fact that the whole horizon has the same velocity. Allowing multiple velocities within one prediction horizon would most probably increase the performance of the controller. However, the complexity in the path planner grows exponentially with the number of different velocity triples in the horizon. The MPCC, which does not have this limitation, is able to plan better trajectories which also improve the closed loop performance. However, the MPCC comes at a price, first in terms of computational cost (it is about a factor of five more expensive than the HRHC when comparing computation times on the same platform, see Section IV-E for more details on computation times) and, second, in terms of robustness. During our experiments, the re-linearization scheme turned out to be sensitive to measurement errors and model drift. In particular, sudden unexpected skidding of the car can cause problems in the re-linearization scheme. Such drifts are also responsible for the high variations between different trajectories in certain areas of the track. Furthermore, since the MPCC relies on the results from the previous iteration to compute the linearizations, sudden jumps in the high-level corridor path planner might make the previous trajectory infeasible. In such a situation, several time steps might be needed to recover feasibility of the planned trajectory, although by using the soft constrained formulation sensible inputs are still provided to the car. The HRHC on the other hand does not need any information from the last iteration as only the current position and velocity is needed. This helps to deal efficiently with unexpected behavior and fast changing measurements or corridors from the high-level obstacle avoidance mechanism.

D. Racing with multiple cars

Since both controllers plan a path around opponents while maximizing the progress, fast and safe overtaking maneuvers are enabled. In the current work, we focus on static obstacles, which seems sufficient to outperform non-expert human drivers. To robustly overtake dynamically moving obstacles (other automatic controllers or expert drivers), it would be necessary to have a prediction of their behavior, which is currently not available. These predictions could however be systematically incorporated into the

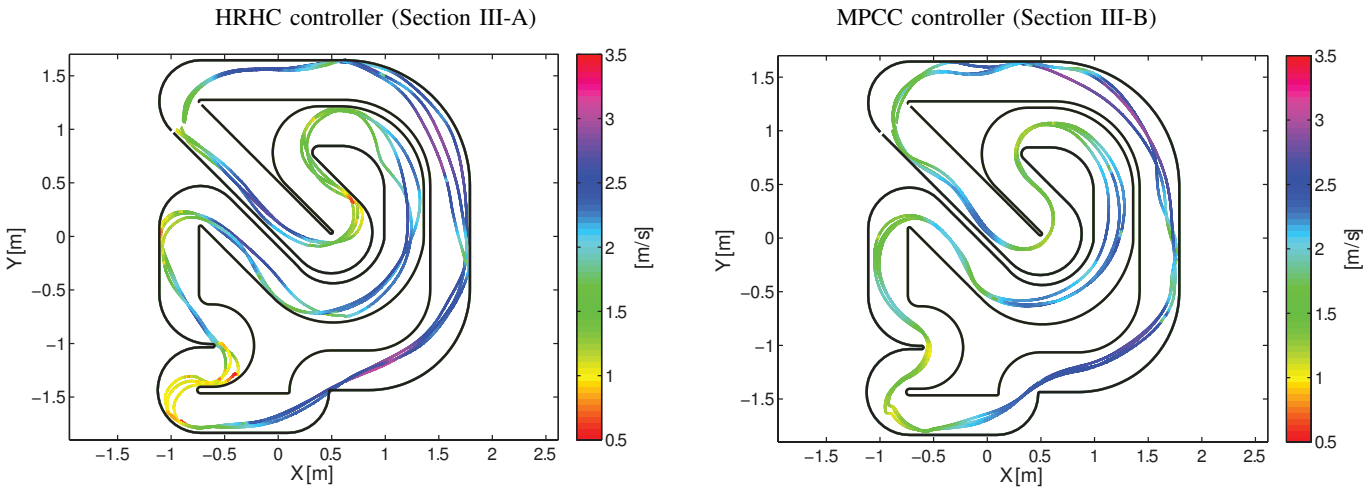

Fig. 8: The driven trajectory with velocity profile for 3 different laps. A video of the HRHC controller is available at https://www.youtube.com/watch?v=ioKTyc9bG4c.

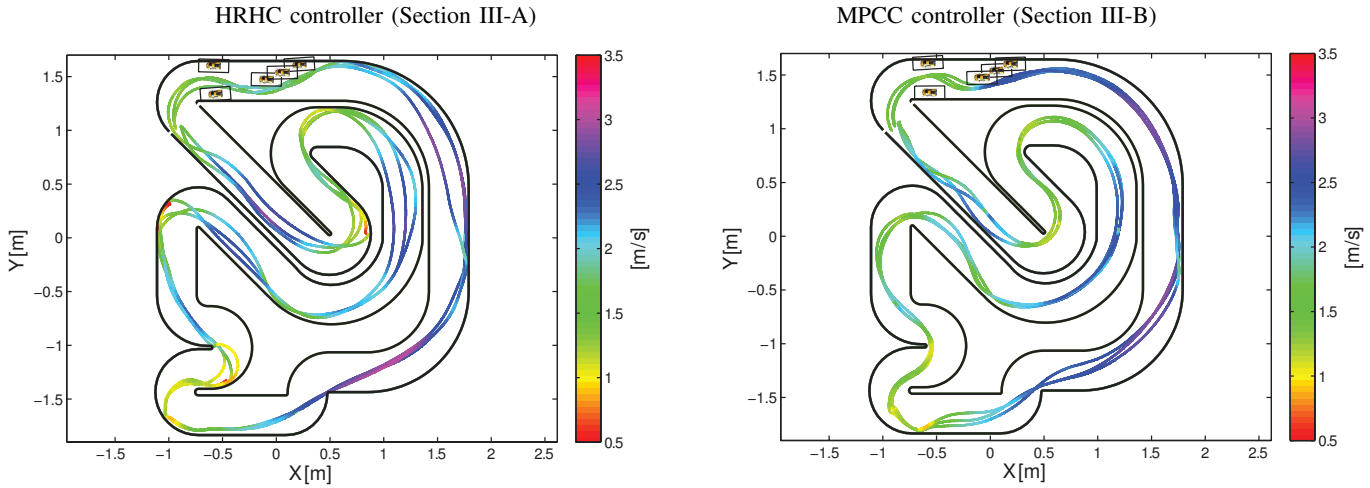

Fig. 9: Driven trajectories with static obstacles for 3 different laps. A video of the MPCC with obstacle avoidance is available at https://www.youtube.com/watch?v=mXaElWYQKC4.

high-level corridor planner outlined in Section III-C, and would merely change the corridor issued to the controllers presented in Section III-A and Section III-B.

An obstacle avoidance situation is shown in Figure 9, with a detailed zoom taken at three different time points in Figure 10. These zooms show the prediction horizon and the corresponding planned trajectories. Due to the limited path planner, the HRHC is not able to directly plan a path around all obstacles from the beginning. Thus it avoids the first three cars, without predicting how to overtake the next two cars. Only after the car has made enough progress and overtaken the first group of obstacles, can the controller find a way between of the other two cars. The MPCC on the other hand plans the path around all obstacles before it even reaches the first car due to its long horizon. This is an advantage, but if the model mismatch is significant it can lead to problems, as the planned path is too optimistic.

E. Computation Times

The computation times of the most time-consuming components of the two controllers are given in Tables II and III for the laps depicted in Figure 8 (single car racing) and Figure 9 (avoidance of opponents). Note that the computation times of the individual blocks of the two controllers cannot be compared directly.

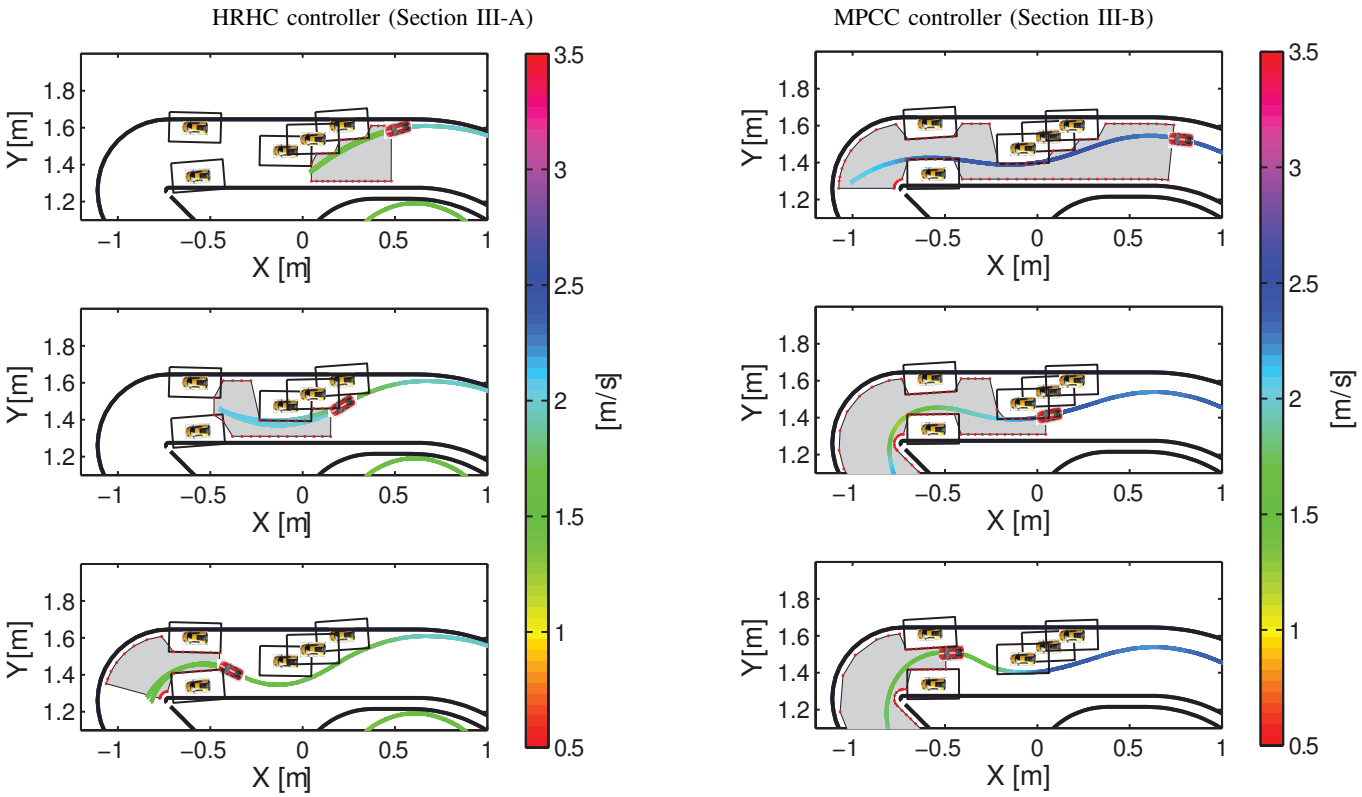

Fig. 10: Obstacle avoidance at 3 different time steps. Grey indicates the corridor chosen by the obstacle avoidance algorithm.

To directly compare the computation times of the two controllers, the horizon length should be identical. However, due to the constant velocity limitation in the HRHC path planner, a horizon length identical to the MPCC is impossible without using multiple segments of constant velocity, which would make the path planning combinatorial in complexity and thus currently prohibitive for a real-time implementation. Vice versa, the MPCC cannot work robustly with the short horizon length used by the HRHC. The multiple car case does not have a significant impact on the execution times of the lower level controller; only the path planning in the HRHC gets slightly more complicated. However, the computation time of the high-level corridor planner (the DP approach from Section III-C) varies significantly, depending on the complexity of the racing situation. For the MPCC, the limiting factor with respect to the sampling time is the computation time for solving the QP. The main bottleneck in the HRHC is the path planner, which has a very high maximal computation time caused by back up rules in the case no feasible trajectory can be found. The QP on the other hand is the most expensive step in the average, but does not have a large variation in computation time, which makes it less critical.

The $20\,\text{ms}$ sampling time was missed by the HRHC controller in only $0.07\,\%$ (one sampling instant in three laps) and by the MPCC controller in $4.4\,\%$ (60 sampling instants in three laps) of the time. Overall, our experiments demonstrate that both schemes can be implemented in (soft) real-time at sampling rates of $50\,\text{Hz}$.

ACKNOWLEDGMENT

We gratefully acknowledge the work of our students in the course of the race car project: Kenneth Kuchera, Samuel Zhao and Florian Perrodin for their fruitful help in implementing the MPCC approach; Michael Janser for his contribution to the obstacle avoidance (DP) algorithm; Sandro Merkli, Nils Wenzler and Marcin Dymczyk for the development of the embedded control software and the testbed setup; Celestine Dünner and Sandro Merkli for building the track and low-level filtering software; Michael

TABLE II: HRHC computation times in milliseconds

Without Obstacles	Mean	Stdev	Max
Border Adjustment	0.02	0.01	0.04
Path Planning	2.93	1.96	13.47
QP Generation	0.52	0.09	1.32
QP with FORCES	5.10	0.73	7.56

With Obstacles	Mean	Stdev	Max
Border Adjustment	1.33	0.37	2.45
Path Planning	3.18	2.04	9.76
QP Generation	0.53	0.06	0.76
QP with FORCES	5.11	0.92	9.25

TABLE III: MPCC computation times in milliseconds

Without Obstacles	Mean	Stdev	Max
Border Adjustment	0.34	0.11	0.64
QP Generation	1.95	0.44	2.92
QP with FORCES	15.03	1.61	21.16

With Obstacles	Mean	Stdev	Max
Border Adjustment	0.70	0.24	1.05
QP Generation	2.34	0.37	2.79
QP with FORCES	14.43	1.40	21.46

Dahinden and Christian Stocker for the initial design of the embedded PCB board; Benjamin Keiser for the current version of the infrared vision system. We furthermore thank Sean Summers, Nikolaos Kariotoglou, Christian Conte and John Lygeros from the Automatic Control Laboratory for supervising various projects and providing important comments to the methods presented in this paper. Special thanks go to Colin Jones, who originated the idea of the race car testbed.

The research leading to these results has received funding from the EU in FP7 via EMBOCON (ICT-248940).

REFERENCES

[1] T. Faulwasser, B. Kern, and R. Findeisen, "Model predictive path-following for constrained nonlinear systems," in *Proceedings of the 48th IEEE Conference on Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference*, pp. 8642–8647, 2009.

[2] D. Lam, C. Manzie, and M. Good, "Model predictive contouring control," in *Conference on Decision and Control (CDC)*, pp. 6137–6142, 2010.

[3] A. Domahidi, A. Zgraggen, M. Zeilinger, M. Morari, and C. Jones, "Efficient interior point methods for multistage problems arising in receding horizon control," in *Conference on Decision and Control (CDC)*, (Maui, HI, USA), pp. 668–674, Dec. 2012.

[4] A. Domahidi, "FORCES: Fast optimization for real-time control on embedded systems." http://forces.ethz.ch, Oct. 2012.

[5] R. Horowitz and P. Varaiya, "Control design of an automated highway system," *Proceedings of the IEEE*, vol. 88, no. 7, pp. 913–925, 2000.

[6] M. Buehler, K. Iagnemma, and S. Singh, *The 2005 DARPA grand challenge: the great robot race*, vol. 36. Springer, 2007.

[7] M. Buehler, K. Iagnemma, and S. Singh, *The DARPA urban challenge: Autonomous vehicles in city traffic*, vol. 56. Springer, 2009.

[8] K. Kritayakirana and C. Gerdes, "Using the centre of percussion to design a steering controller for an autonomous race car," *Vehicle System Dynamics*, vol. 15, pp. 33–51, 2012.

[9] E. Velenis, E. Frazzoli, and P. Tsiotras, "On steady-state cornering equilibria for wheeled vehicles with drift," in *Proceedings of the 48th IEEE Conference on Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference*, pp. 3545–3550, IEEE, 2009.

[10] C. Voser, R. Y. Hindiyeh, and J. C. Gerdes, "Analysis and control of high sideslip manoeuvres," *Vehicle System Dynamics*, vol. 48, no. S1, pp. 317–336, 2010.

[11] F. Borrelli, P. Falcone, T. Keviczky, and J. Asgari, "MPC-based approach to active steering for autonomous vehicle systems," *International Journal of Vehicle Autonomous Systems*, vol. 3, no. 2, pp. 265–291, 2005.

[12] P. Falcone, F. Borrelli, J. Asgari, H. E. Tseng, and D. Hrovat, "Predictive active steering control for autonomous vehicle systems," *IEEE Transactions on Control Systems Technology*, vol. 15, no. 3, pp. 566–580, 2007.

[13] T. Besselmann and M. Morari, "Hybrid parameter-varying model predictive control for autonomous vehicle steering," *European Journal of Control*, vol. 14, no. 5, pp. 418–431, 2008.

[14] J. Ackermann, J. Guldner, W. Sienel, R. Steinhauser, and V. I. Utkin, "Linear and nonlinear controller design for robust automatic steering," *IEEE Transactions on Control Systems Technology*, vol. 3, no. 1, pp. 132–143, 1995.

[15] S. Fuchshumer, K. Schlacher, and T. Rittenschober, "Nonlinear vehicle dynamics control - a flatness based approach," in *Conference on Decision and Control (CDC)*, pp. 6492–6497, 2005.

[16] Y. Gao, T. Lin, F. Borrelli, E. Tseng, and D. Hrovat, "Predictive control of autonomous ground vehicles with obstacle avoidance on slippery roads," *Proceedings of DSCC*, 2010.

[17] A. Gray, Y. Gao, T. Lin, J. K. Hedrick, H. E. Tseng, and F. Borrelli, "Predictive control for agile semi-autonomous ground vehicles using motion primitives," in *American Control Conference (ACC)*, pp. 4239–4244, IEEE, 2012.
[18] E. Frazzoli, M. Dahleh, and E. Feron, "Maneuver-based motion planning for nonlinear systems with symmetries," *IEEE Transactions on Robotics*, vol. 21, no. 6, pp. 1077–1091, 2005.
[19] E. Velenis, P. Tsiotras, and J. Lu, "Modeling aggressive maneuvers on loose surfaces: The cases of trail-braking and pendulum-turn," in *European Control Conference*, pp. 1233–1240, 2007.
[20] J. Frasch, A. Gray, M. Zanon, H. Ferreau, S. Sager, F. Borrelli, and M. Diehl, "An auto-generated nonlinear MPC algorithm for real-time obstacle avoidance of ground vehicles," in *Proceedings of the European Control Conference*, no. accepted, 2013.
[21] M. Diehl, H. Bock, J. Schlöder, R. Findeisen, Z. Nagy, and F. Allgöwer, "Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations," *Journal of Process Control*, vol. 12, no. 4, pp. 577–585, 2002.
[22] J. Jeon, S. Karaman, and E. Frazzoli, "Anytime computation of time-optimal off-road vehicle maneuvers using the RRT*," in *Conference on Decision and Control (CDC)*, pp. 3276–3282, 2011.
[23] J. Jeon, R. Cowlagi, S. Peters, S. Karaman, E. Frazzoli, P. Tsiotras, and K. Iagnemma, "Optimal motion planning with the half-car dynamical model for autonomous high-speed driving," in *American Control Conference (ACC)*, pp. 188–193, 2013.
[24] E. Bakker, L. Nyborg, and H. Pacejka, "Tyre modelling for use in vehicle dynamics studies," *SAE*, 1987.
[25] D. G. Luenberger, *Linear and nonlinear programming*. Springer, 2003.
[26] E. C. Kerrigan and J. M. Maciejowski, "Soft constraints and exact penalty functions in model predictive control," in *Proc. UKACC International Conference (Control 2000)*, (Cambridge, UK), September 2000.
[27] Y. Koren, "Control of machine tools," *Transactions of the ASME Journal of Manufacturing Science and Engineering*, vol. 119, 1997.
[28] R. C. Ko, M. C. Good, and S. K. Holgamuge, "Adaptive calibration of feedforward controllers for laser profiling machines," in *Proceedings of Information, Decision and Control*, 1999.
[29] M. Diehl, H. G. Bock, and J. P. Schlöder, "A real-time iteration scheme for nonlinear optimization in optimal feedback control," *SIAM Journal on Control and Optimization*, vol. 43, no. 5, pp. 1714–1736, 2005.
[30] D. P. Bertsekas, *Dynamic programming and optimal control*, vol. 1. Athena Scientific Belmont, 1995.
[31] D. Ferguson, T. M. Howard, and M. Likhachev, "Motion planning in urban environments," *Journal of Field Robotics*, vol. 25, pp. 939–960, 2008.
[32] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, and et al., "Junior: The stanford entry in the urban challenge," *Journal of Field Robotics*, vol. 25, pp. 569–597, 2008.
[33] Y. Kuwata, G. A. Fiore, J. Teo, E. Frazzoli, and J. P. How, "Motion planning for urban driving using RRT," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1681–1686, 2008.