

An Inverse Optimal Control Approach for Trajectory Prediction of Autonomous Race Cars

Rudolf Reiter¹, Florian Messerer¹, Markus Schratter², Daniel Watzenig^{2,3} and Moritz Diehl^{1,4}

Abstract—This paper proposes an optimization-based approach to predict trajectories of autonomous race cars. We assume that the observed trajectory is the result of an optimization problem that trades off path progress against acceleration and jerk smoothness, and which is restricted by constraints. The algorithm predicts a trajectory by solving a parameterized nonlinear program (NLP) which contains path progress and smoothness in cost terms. By observing the actual motion of a vehicle, the parameters of prediction are updated by means of solving an inverse optimal control problem that contains the parameters of the predicting NLP as optimization variables. The algorithm therefore learns to predict the observed vehicle trajectory in a least-squares relation to measurement data and to the presumed structure of the predicting NLP. This work contributes with an algorithm that allows for accurate and interpretable predictions with sparse data. The algorithm is implemented on embedded hardware in an autonomous real-world race car that is competing in the challenge *Roborace* and analyzed with respect to recorded data.

I. INTRODUCTION

In real-world autonomous driving scenarios, a core challenge is the prediction of other agents in the environment. The prediction algorithms differ related to the scenario and to the availability of data. For instance, in urban driving, a large amount of data might be available due to massive data collection of the vehicle industry. For autonomous racing tasks, there is a lack of extensive data sets, thus supervised learning of data-driven predictions is not feasible. In our research, we focus on a racing setting related to a competition called *Roborace*. As part of this racing series, the participating teams develop software for the fully autonomous operation of electric race cars and are confronted with increasingly demanding challenges from one event to the other. Whereas the ego vehicle moves on a real racetrack, the state observations of the (currently) purely virtual opponent race cars are provided by the mixed-reality simulator to the car's software about 200 meters in advance. The up to six virtually present opponent cars are set up by the organizers with different racing algorithms that are supposed to race with different performance and driving style. The opponent cars are currently not considered as

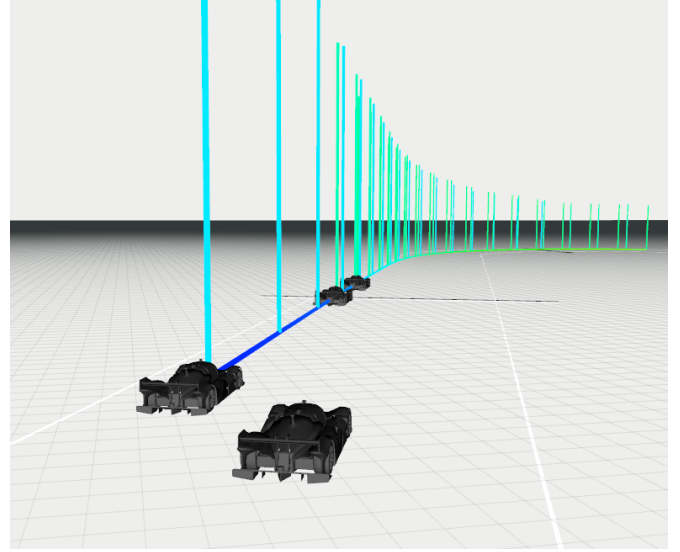


Fig. 1. Presented trajectory prediction in a simulation. The height and color of the bars correspond to the predicted speed.

strategic decision makers, i.e., they are not performing game theoretic actions such as blocking. Thus, the race cars can be seen as non-interactive agents. Generally, there is no a priori knowledge available about the opponents, except of their racing intention. Therefore, it is impossible to use an a priori fully parameterized vehicle model as a basis for prediction. Furthermore, an extensive system identification is impossible due to the short time the vehicle can be observed before it needs to be overtaken. The goal of this paper is to present a method that predicts the behavior of other race cars even with sparse data. A typical scenario is shown in Fig. 1, where the ego race car and three other opponent cars are on a racetrack and trajectories of our presented predictor are shown with bars, corresponding to the predicted velocity.

Our work starts with framing the basic and limited knowledge about the opponents as a sparsely parameterized predictor whose parameters can be estimated by a limited amount of data. Since it is known that the intention of the other opponents is time-optimal driving, the predictor is stated as a parameterized optimization problem for progress maximization, referred to as low-level nonlinear program (LLNLP) which is assumed to be solved by the other agent. The estimation of the parameters is performed by solving an inverse optimal control (IOC) problem, which enforces the optimality conditions for the LLNLP as constraints and performs least-squares optimization on the deviation of the

¹Department of Microsystems Engineering (IMTEK), University Freiburg, 79110 Freiburg, Germany
{rudolf.reiter, florian.messerer, moritz.diehl}@imtek.uni-freiburg.de

²Virtual Vehicle Research Center, Inffeldgasse 21a, 8010 Graz, Austria
{markus.schratter, daniel.watzenig}@v2c2.at

³Institute of Automation and Control, Graz University of Technology, Inffeldgasse 21b, Graz, Austria.

⁴Department of Mathematics, University Freiburg, 79110 Freiburg, Germany

resulting LLNLP-trajectory to the collected observed data of the particular opponent vehicle. This results in a set of parameters for the LLNLP which are locally optimal with respect to the chosen structure of the LLNLP and the observed data. In fact, the chosen formulation only finds a stationary point as opposed to an optimal point and is dependent on the initialization due to its non-convex structure, but in practice both were observed to not have a significant influence on the performance. The LLNLP is solved in real-time for each opponent, starting with an initial set of parameters, which are updated as soon as enough data is available.

The LLNLP is used to predict the velocity along a curve, which is obtained by blending the current motion into a previously computed minimum curvature path. The parameters related to LLNLP are the constraint limits and the square penalties on the input (jerk) and the acceleration states. The estimation of the acceleration constraints is separated from the bi-level optimization problem into a separate constraint estimation QP (CQP) whose constraint estimates update both the bi-level program for the weight parameter estimation and the final LLNLP for predicting the opponent trajectories in real time.

The performance of the described algorithm is shown with recorded data from differently driving opponent race cars in a *Hardware-In-The-Loop* setting.

A. Related work

Trajectory prediction in the domain of autonomous vehicles is dominated by data-driven approaches which are based on regression and pattern matching. This is applicable if the availability of sufficient data related to human driven vehicles on public streets is given. If interaction and sequential decision making is considered, often IOC or inverse reinforcement learning (IRL) are used. Often deep neural networks (DNNs) are used as function approximators [1] and the time dependency suggests the use of recurrent neural architectures as seen in [2], [3], [4]. Also, various other DNN architectures are used, such as convolutional neural networks [5]. If statistically qualitative data is available, these approaches work well, and even their application to real time systems as trained networks is favorable due to the high evaluation speed of DNNs. Using an optimization problem as a function approximator or even within a neural network is a field with many related research areas, ranging from reinforcement learning with an embedded MPC structure [6] to generic optimization layers [7]. Using bi-level optimization to estimate the parameters of a low-level problem is used more rarely. Related to vehicle predictions, it was used in a similar approach, which focuses on urban driving scenarios and the game theoretic interaction between agents [8], [9]. Furthermore, for robotic predictions [10] or even human motion predictions [11], a bi-level problem was used. For unconstrained linear systems, [12] the authors show that the IOC can even be stated as a convex semidefinite program.

A detailed survey of vehicle prediction approaches is given in [1], although IOC appears only in the context of IRL. A general survey on bi-level optimization is given in [13], which mentions the presented approach of solving the lower-level program by restricting it to a stationary point, especially for convex problems.

B. Contribution

In the domain of autonomous racing, to the best knowledge of the authors, this work is the first that uses bi-level optimization together with the LLNLP for real-time trajectory prediction. Since bi-level problems are hard to solve, this work also addresses novel techniques that can be used in challenging real-world conditions such as racing. This paper follows previous work for solving motion planning problems for autonomous racing [14], [15].

II. PREDICTION ARCHITECTURE

In Fig. 2, the architecture of the proposed algorithm is shown. The algorithm consists of an offline and an online part. The precomputations in the offline part account for the optimal racing path along the known racetrack. The online part is constructed for each opponent vehicle that is observed and is split into a slower (0.5 Hz) estimation part and a faster (10 Hz) prediction part. In the path prediction (PP) a curve is blended from the current opponent vehicle position to the precomputed racing line. The main prediction component is the LLNLP which computes the trajectory with respect to the parameterized constraints and the parameterized weights, starting at the observed current opponent value. The constraint estimator (CQP) passes its estimated constraint parameters to the high-level NLP (HLNLP) and both the CQP and the HLNLP estimate the parameters of the LLNLP. The online part is executed for each of the M observed vehicles.

III. PREDICTION ALGORITHM

In the following, the prediction algorithm is described by each component. In Sections III-A to III-D, the main blocks of Fig. 2 are described and in the final part the pseudocode (1) is stated.

A. Path prediction (PP)

Given the racetrack layout, a time-optimal path $p_{\text{topt}}(s)$ is computed by curvature minimization related to [14]. The path variable s is related to the position on a reference center track line. Given the current opponent vehicle state, a linear extended constant motion path $p_c(s)$ is blended into the precomputed path for $s < s_f$ with

$$p_p(s) = \frac{s}{s_f} p_{\text{topt}}(s) + \frac{s - s_f}{s_f} p_c(s). \quad (1)$$

For $s \geq s_f$ the prediction path is set equal to the racing path.

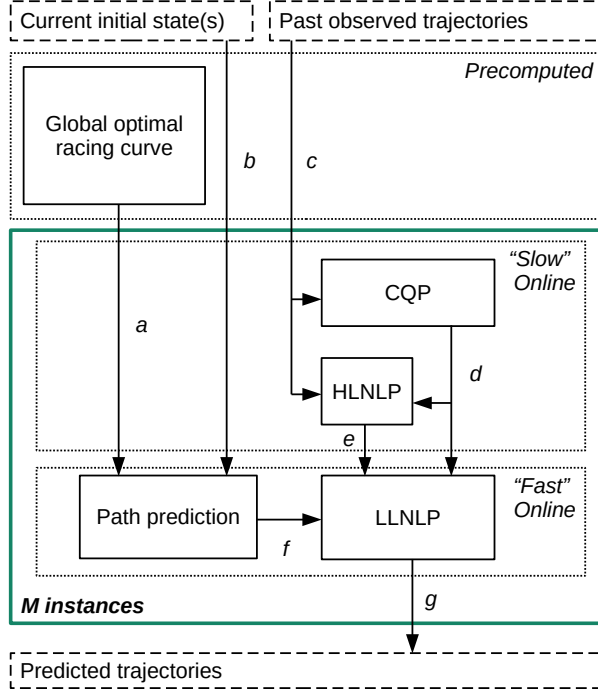


Fig. 2. Algorithm architecture. (a: global racing path, b: initial state \bar{x}_0 , c: trajectory data samples, d: constraints a_{\max} , e: weights w , f: Cartesian coordinates and curvature parameters of blended path segment $\bar{\kappa}$, g: predicted trajectory)

B. Low-level program for the trajectory prediction (LLNLP)

The path predictor predicts the curve that is described by its path length s and the associated curvature $\kappa(s) = \frac{d\phi}{ds}$. The curvature is described by a piece-wise *linear* polynomial and parameterized to interpolate N_κ precomputed values κ_i for the curvature along the path segment. The values are summarized as $\bar{\kappa} = [\kappa_i \dots \kappa_{N_\kappa-1}]$. Details on the computation can be found in [14]. Note that the *linear* interpolation leads to discontinuous derivatives in the inequality constraints and violates the condition of twice continuously differentiable functions required for second order NLP algorithms. Nevertheless, we empirically found a speedup of a factor of 100 to 1000 compared to *bsplines* as interpolating polynomials, with practically no convergence problems.

The LLNLP predicts the estimated velocity along this curve by solving an optimal control problem which consists of a linear discrete model $F(x_k, u_k, \Delta t)$, acceleration constraints $h_a(x_k, \bar{\kappa}, a_{\max})$ and state constraints \underline{x} and \bar{x} . Since the path is given, the predicted motion along the curve is described by means of three chained integrators, where the input u is the jerk. The state vector consequently consists of the path progress s , the velocity v and the acceleration a with $x = [s \ v \ a]^\top \in \mathbb{R}^3$. Since the integrator chain is a linear system, the discretization (zero-order-hold controls) of the dynamics can be computed exactly by matrix exponentials

and leads to the affine function $F(x_k, u_k, \Delta t) = A(\Delta t)x_k + B(\Delta t)u_k$. The only constraint captured in the box constraints $\underline{x} \preceq x_k \preceq \bar{x}$ is the limitation of the speed v to v_{\max} and to positive values. The optimal control problem is discretized in $N-1$ intervals using discrete multiple shooting and solved by sequential quadratic programming using the real-time NMPC solver *acados* [16]. To account for the progress maximizing requirement for the resulting trajectory, a linear negative cost $q_n = [-1 \ 0 \ 0]^\top$ for the last discrete position is used. The matrix $W = \text{diag}([0 \ 0 \ w_{\text{acc}}])$ and the scalar $R = w_{\text{jerk}}$ are the weights that describe the motion of the predicted opponent vehicle in the presented structure, if no constraints are active. Finding the values of w_{acc} and w_{jerk} is the objective of the HLNLP component. Slack variables $s_{\text{LL}} = [s_{\text{LL},0}, \dots, s_{\text{LL},N}] \in \mathbb{R}^{8 \times N}$ with weights α_1, α_2 are added for the online forward implementation to account for the robustness of the SQP algorithm. We can then state the lower-level problem $P_{\text{LL}}(w, \bar{x}_0, \bar{\kappa}, a_{\max})$ as

$$\begin{aligned}
 & \underset{\substack{x_0, \dots, x_N, \\ u_0, \dots, u_{N-1}, \\ s_0, \dots, s_N}}{\text{minimize}} & \sum_{k=0}^{N-1} \|x_k\|_{2,W}^2 + \|u_k\|_{2,R}^2 + q_N^\top x_N \\
 & + \sum_{k=0}^N \alpha_1 \mathbf{1}^\top s_{\text{LL},k} + \alpha_2 \|s_{\text{LL},k}\|_2^2 \\
 & \text{subject to} & x_0 = \bar{x}_0, \\
 & & x_{k+1} = F(x_k, u_k, \Delta t), \quad k = 0, \dots, N-1, \\
 & & \underline{x} \preceq x_k \preceq \bar{x}, \\
 & & 0 \preceq h_a(x_k, \bar{\kappa}, a_{\max}) + s_{\text{LL},k}, \\
 & & 0 \preceq s_{\text{LL},k}, \quad k = 0, \dots, N,
 \end{aligned} \tag{2}$$

where $\mathbf{1}$ is a vector of all 1's of appropriate size. The acceleration constraints $h_a(x_k, \bar{\kappa}, a_{\max})$ approximate the friction, throttle and breaking boundaries of the vehicle by means of a polytope in the space of the two-dimensional acceleration vector $a(x_k, \bar{\kappa}) = [a_{\text{lat}}(x_k, \bar{\kappa}) \ a_{\text{lon}}(x_k)]$ which are often related to the "Kamm's circle". The polytope is typically symmetric to the longitudinal axis. It is chosen such that it consists of box-constraints along the axes and diagonal constraints that are parallel to the lines described by the connection of the axis aligned maximum values. Consequently, the diagonal constraints depend on values of the axis aligned constraints. The presented approach computes the axis aligned constraints first and uses those values as inputs to the diagonal constraints. An example of the fitted acceleration constraints can be seen in Fig. 3. Therefore, 8 linear constraints arise, where 6 of them are pair-wise symmetric. The only non convexity in (2) emerges from the dependency of $a_{\text{lat}}(x_k) = -v_k^2 \kappa(s_k, \bar{\kappa})$. The acceleration constraints $h_a(x_k, \bar{\kappa}, a_{\max})$ can be stated as

$$h_a(x_k, \bar{\kappa}, a_{\max}) = a_{\max} - \text{diag}(d_{\text{len}}) D a(x_k, \bar{\kappa}) \tag{3a}$$

$$\bar{a} = \sqrt{a_{\text{lat},\max}^2 + a_{\text{lon},\max}^2} \tag{3b}$$

$$d_{\text{len}}^\top = [1 \ 1 \ 1 \ 1 \ \bar{a} \ \bar{a} \ \bar{a} \ \bar{a}] \tag{3c}$$

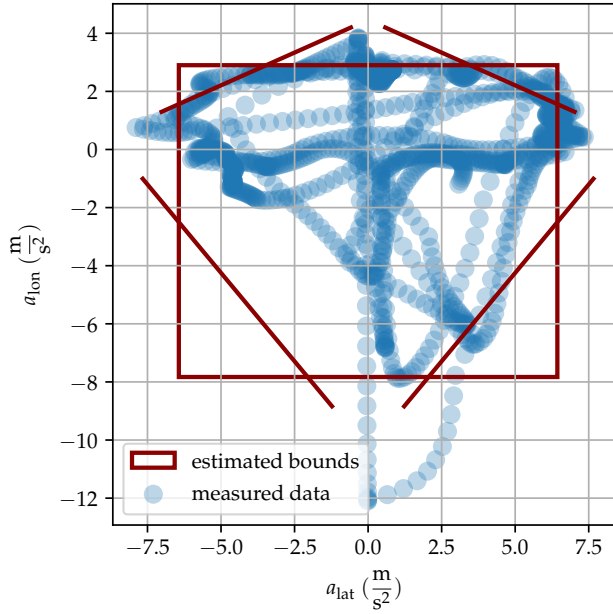


Fig. 3. Acceleration constraint estimation. In total 8 constraints are fitted as a convex polytope to measurement data.

$$D = \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & -1 \\ 0 & -1 \\ a_{lon,max} & a_{lat,max} \\ -a_{lon,max} & a_{lat,max} \\ a_{lon,min} & -a_{lat,max} \\ -a_{lon,min} & -a_{lat,max} \end{bmatrix}, \quad a_{max} = \begin{bmatrix} a_{lat,max} \\ a_{lat,max} \\ a_{lon,max} \\ a_{lon,min} \\ a_{q,north} \\ a_{q,north} \\ a_{q,south} \\ a_{q,south} \end{bmatrix} \quad (3d)$$

The term $\text{diag}(d_{len})D$ collects the row vectors with unit length that represent direction vectors that are used to project the acceleration vector and to constrain to the consecutive projected scalar value. Obviously, the second part of the vector d_{len} and the lower part of the matrix D shows the dependency on the axis aligned constraints. The maximum acceleration values are collected in the vector a_{max} , the initial observed state \bar{x}_0 , together with the weights $w^\top = [w_{jerk} \ w_{acc}]$ are the input parameters of the LLNLP. The parameters that are not subject to be changed by the estimation within the HLNLP are summarized as $p^\top = [\bar{x}_0 \ a_{max}^\top \ \bar{\kappa}]$.

C. Quadratic program for constraint estimation (CQP)

In order to remove computational complexity from the HLNLP, the constraint estimation was separated. Even with fixed constraints, the structure of the HLNLP is highly non-convex and challenging to solve. By means of a Kalman-filter-based vehicle state estimation, the observed accelerations a_i are computed and stored as a data set of N_a data samples in \mathbb{R}^2 . Those acceleration data are used to fit linear constraints. The projection $e_k^\top Da_i$ for the estimation of the linear constraint c_k is performed for the 8 constraints. The vector e_k represents the k -th unit vector in \mathbb{R}^8 . Since the measured data is noisy, it requires a robust estimation of the

constraints which accounts for outliers. This is achieved by the quadratic program (4), where the estimated constraint violation is penalized linearly and realized by means of a hinge loss $h(x) = \max(0, x)$. This function adds no costs if the measured value is lower than the constraint, and penalizes linearly otherwise. The deflection of the constraint is penalized quadratically with a weight ω and a minimum at the prior estimated value \hat{c}_k . Since the prior estimated value acts as a lower bound and would not decrease during iterations, the value is lowered by a factor $r < 1$ in each iteration for the axis aligned constraints with $\hat{c}_k \leftarrow r\hat{c}_k$. For the diagonal constraints, \hat{c}_k is chosen as the distance of the diagonal line of the origin. The problem formulation for estimating constraint k in the bounding polytope with 8 linear constraints is stated as

$$\begin{aligned} \underset{c_k \in \mathbb{R}}{\text{minimize}} \quad & \frac{1}{N_a} \sum_{i=0}^{N_a-1} \max(0, e_k^\top Da_i - c_k) \\ & + \omega(c_k - \hat{c}_k)^2. \end{aligned} \quad (4)$$

The problem can be formulated into a smooth quadratic program using slack variables ζ for implementing the hinge function which leads to the formulation

$$\begin{aligned} \underset{\substack{c_k \in \mathbb{R}, \\ \zeta \in \mathbb{R}^{N_a}}}{\text{minimize}} \quad & \omega(c_k - \hat{c}_k)^2 + \frac{1}{N_a} \sum_{i=0}^{N_a-1} \zeta_i \\ \text{subject to} \quad & 0 \leq \zeta_i \\ & e_k^\top Da_i - c_k \leq \zeta_i \quad i = 0, \dots, N_a - 1. \end{aligned} \quad (5)$$

The solutions of the CQP are directly used as acceleration constraints a_{max} in (3d).

D. Bi-level program for the LLNLP parameter estimation (HLNLP)

The HLNLP fits the LLNLP with the parameters derived from the CQP to observed measurement data \bar{x} with a least-squares error measure. The observed trajectory might differ at the last points from the predicted trajectory, even if the true parameters were used, since the controller of the observed vehicle most likely had adapted to even further distant constraints like a sharp curve. To account for this structural uncertainty, the weight matrix Q_k is linearly reduced to zero for the final N_R points. Problem (6) shows the basic structure of the problem. The optimization variables are the estimated trajectory x of the LLNLP and the weighting parameters w . To account for the iterative estimation of the parameter w , the previously estimated parameter \hat{w} together with the associated weight matrix P is used as an arrival cost, as shown with MHE in [17]. To simplify the algorithm, the weight matrix is set constant. The basic structure of the problem can be written as

$$\begin{aligned} \underset{x, u, w}{\text{minimize}} \quad & \sum_{k=1}^{N_T-1} \|x_k - \bar{x}_k\|_{2, Q_k}^2 + \|w - \hat{w}\|_{2, P}^2 \\ \text{subject to} \quad & x, u \in \text{argmin} P_{LL}(w, \bar{x}_0, \bar{\kappa}, a_{max}), \\ & w \succcurlyeq 0, \end{aligned} \quad (6)$$

where $x \in \mathbb{R}^{N_x \times N_T}$, $u \in \mathbb{R}^{N_u \times (N_T-1)}$ and $w \in \mathbb{R}^2$. The optimization variables are the estimated trajectory x of the LLNLP and the weighting parameters w .

The low-level program $P_{LL}(w, \bar{x}_0, \bar{\kappa}, a_{\max})$ in (2) can be written as

$$\begin{aligned} & \underset{z \in \mathbb{R}^{N_z}}{\text{minimize}} && f_{LL}(z, w) \\ & \text{subject to} && g_{LL}(z) = 0, \\ & && h_{LL}(z, \bar{x}_0, \bar{\kappa}, a_{\max}) \succcurlyeq 0, \end{aligned} \quad (7)$$

with $z = [\text{vec}(x)^\top \text{vec}(u)^\top]^\top$ and $N_z = N_x N_T + N_u(N_T - 1)$. The domains and co-domains of the functions are $f_{LL} : \mathbb{R}^{N_z \times N_w} \rightarrow \mathbb{R}$, $g_{LL} : \mathbb{R}^{N_z} \rightarrow \mathbb{R}^{N_T N_x}$ and $h_{LL} : \mathbb{R}^{N_z} \rightarrow \mathbb{R}^{N_T N_{h,LL}}$, where $N_{h,LL} = 10$ in this case, with two bounds on the velocity state and 8 acceleration constraints. The number of weights corresponding to the smoothness is $N_w = 2$. The constraints a_{\max} are parameters and updated by means of the estimation of the CQP.

To solve the problem, the bi-level problem can be formulated as an NLP which is summarized as

$$\begin{aligned} & \underset{x, u, w, \tau, \lambda, \mu, s}{\text{minimize}} && \sum_{k=0}^{N_T-1} \|x_k - \bar{x}_k\|_{2,Q_k}^2 + q_\tau \tau \end{aligned} \quad (8a)$$

$$+ \beta_1 \mathbf{1}^\top s + \beta_2 \|s\|_2^2 + \|w - \hat{w}\|_{2,P-1}^2 \quad (8b)$$

$$\begin{aligned} & \text{subject to} && 0 = \nabla_z f(z, w) - \nabla_z g_{LL}(z) \lambda \\ & && - \nabla_z h_{LL}(z, p) \mu, \end{aligned} \quad (8c)$$

$$0 \preceq w, \quad (8d)$$

$$0 = g_{LL}(z), \quad (8e)$$

$$0 \leq \tau, \quad (8f)$$

$$0 \preceq \mu, \quad (8g)$$

$$0 \preceq h_{LL}(z, p) + s, \quad (8h)$$

$$\tau \geq \mu_i h_{LL,i}(z, p), \quad i = 0, \dots, N_{h,LL} - 1, \quad (8i)$$

$$s \succeq 0, \quad (8j)$$

where $x \in \mathbb{R}^{N_x \times N_T}$, $u \in \mathbb{R}^{N_u \times (N_T-1)}$, $w \in \mathbb{R}^2$, $s \in \mathbb{R}^{N_T N_{h,LL}}$, $\tau \in \mathbb{R}$, $\lambda \in \mathbb{R}^{N_T N_x}$, and $\mu \in \mathbb{R}^{N_T N_{h,LL}}$.

We enforce a stationary point in the LLNLP as constraint in the high-level problem by enforcing the KKT conditions by means of constraints which are stated in (8c-8i). For this aim, additional optimization variables arise that are the dual variables λ and μ . A major challenge here is to account for the highly non-convex complementarity conditions arising from the inequalities of the LLNLP. Therefore, a relaxed problem is stated within the constraints, which is lower bounded by the actual complementarity condition and upper bounded by its relaxed version related to the interior point approach as seen in (8f-8i). If the complementarity is relaxed too much, the estimation of the weight parameters can become wrong. Consequently, the relaxing parameter $\tau \in \mathbb{R}$ is also integrated as an optimization variable into the HLNLP and initialized with a "high" value (e.g. 1.0). A high value for q_τ together with the linear penalty of τ is used to achieve the exact complementarity constraints. Slack variables s account for infeasibilities.

The number of primal variables in the high-level program, which are $N_{\text{var,HL}} = 2N_x N_T + N_u(N_x - 1) + 2N_h N_T$ rises notably compared to the low-level program, which are $N_{\text{var,LL}} = N_x N_T + N_u(N_x - 1)$, but is of the same complexity w.r.t. N_x, N_T and N_h . This program is solved using the interior point solver *IPOPT* [18] formulated in CasADi [19], which again uses a relaxation of the problem in order to account for the inequality constraints. By using the presented formulation, we can explicitly account for the accuracy of the complementarity constraint in the stationary point of the low-level program. Note that the Hessian of the HLNLP actually contains third-order derivatives of the original LLNLP, thus posing the condition of three times differentiable smooth functions in the LLNLP.

E. Algorithm

Algorithm (1) describes the sequential interaction of the components with respect to the architecture in Fig. 2. The solvers *CQP* and *HLNLP* are executed as threads that update the estimation values in a lower frequency than the main predicting solver *LLNLP*, together with the path prediction *PP*.

```

input : Initial weights and constraints  $\hat{w}$ ,  $c_k$ ,
        Observed state measurements  $\bar{x}_0$ 
output: Predicted trajectory  $x_{\text{pred}}$ 
1 HLNLPsolved  $\leftarrow$  True;
2 CQPsolved  $\leftarrow$  True;
3 while True do
4   if CQPsolved then
5     CQPsolved  $\leftarrow$  False;
6      $\bar{x} \leftarrow$  last  $N_a$  state samples;
7      $\bar{\kappa} \leftarrow \text{curv}(\bar{x})$ ;
8      $\hat{c}_k \leftarrow r c_k \quad k = 0, \dots, 3$ ;
9      $\hat{c}_k \leftarrow \text{dist}(\hat{c}) \quad k = 4, \dots, 7$ ;
10    Set CQP parameters  $\hat{c}_k$ ,  $\omega$ ,  $a(\bar{x}, \bar{\kappa})$ ;
11    Start CQP solver (Updates: CQPsolved,  $c_k$ );
12  end
13  if HLNLPsolved then
14    HLNLPsolved  $\leftarrow$  False;
15     $\bar{x} \leftarrow$  last  $N_T$  state samples;
16     $\bar{\kappa} \leftarrow \text{curv}(\bar{x})$ ;
17     $\hat{w} \leftarrow w$ ;
18     $a_{\text{lat},k} \leftarrow c_k \quad k = 0, \dots, 7$ ;
19    Set HLNLP parameters  $a_{\text{lat}}$ ,  $\bar{x}$ ,  $\bar{\kappa}$ ,  $\hat{w}$ ;
20    Start HLNLP solver (Updates: HLNLPsolved,
         $w$ );
21  end
22   $\bar{x}_0 \leftarrow$  State measurement input;
23   $a_{\text{lat},k} \leftarrow c_k \quad k = 0, \dots, 7$ ;
24   $\bar{\kappa} \leftarrow PP(\bar{x}_0)$ ;
25   $x_{\text{pred}} \leftarrow \text{solve LLNLP}(\bar{x}_0, \bar{\kappa}, w, a_{\text{lat}})$ ;
26 end

```

Algorithm 1: IOC Prediction

TABLE I
PARAMETER SETTINGS

Parameter	Value	Parameter	Value
$c_{k,0} \dots c_{k,3}$	5, 5, 2.5, -5 m/s	s_f	300m
w_0^T	[0.5 0.2]	N	111
ω	12.5	ΔT (LL)	0.1s
N_a	10^3	ΔT (HL)	1s
N_κ	700	α_1, α_2	$10^4, 10^8$
N_T	25s	β_1, β_2	$10^5, 10^6$
P	$\text{diag}([2 \cdot 10^{-7} \ 9 \cdot 10^7])$	q_τ	10^7

TABLE II
COMPONENT SETTINGS

Component	Samples/Nodes	Notes	Runs
PP	150		1e3
CQP	500	Eval. per constraint (1/5)	1e2
HLNLP	35	$\Delta t=1s$	50
LLNLP	60	$\Delta t=0.1s$	1e3

IV. RESULTS

The algorithm was tested with recorded data. Qualitatively, these tests fully describe the performance of the algorithm. Nevertheless, the embedded performance, especially the real-time performance of the LLNLP was proven in several real racing events. This shows that the proposed algorithm can work in embedded real-world systems.

A. Hardware and Software Setup

The proposed LLNLP was tested on a race car hardware (Section IV-C) including the NVIDIA DrivePX 2 in a Docker environment with Ubuntu 20.04. This electronic control unit (ECU) provides two CPUs (4x ARM Denver, 8x ARM Cortex A57) and two GPUs (2x Tegra X2, 2x Pascal GPU). The open-source *OSQP* solver [20] was used in a mixed Python/C++ ROS-framework for solving problem (III-C) using CasADi [19] as an interface. CasADi was also used as an interface together with *IPOPT* [18] to solve the HLNLP of Section III-D. The time critical real-time estimation related to the LLNLP (Section III-B) was performed using *acados* [16] as an NLP solver. For each opponent car, a separate solver was created which was executed as a thread, updating a data-structure that contained the most recent prediction. The full algorithm was tested offline (Section IV-B) in simulations with an Alienware m-15 Notebook and an Intel Core i7-8550 CPU (1.8 GHz). The parameters used for the evaluation are shown in Table I. In Table III, the time statistics of the different optimization parts are shown and in Table II the relevant settings are given. Notably, the LLNLP was failing in 2 out of 1000 randomly parameterized test runs, which was due to the linear interpolation of the curvature as described in Section III-B. This failure rate is out-weighted in practice by the enormous speed gain of a linear interpolation.

B. Performance analysis with recorded data

1) *Validation of the CQP*: Fig. 3 shows the estimation of constraints related to 1000 recorded acceleration data samples. The acceleration was computed out of the observed

TABLE III
SOLVER TIMING STATISTICS

Component	Solver	t_{max} (ms)	t_{ave} (ms)	fail rate (%)
PP	none	< 1	< 1	0
CQP	OSQP	15.5	8.1	0
HLNLP	IPOPT	6237	520	5
LLNLP	acados hpipm(QP)	2748	91	0.2

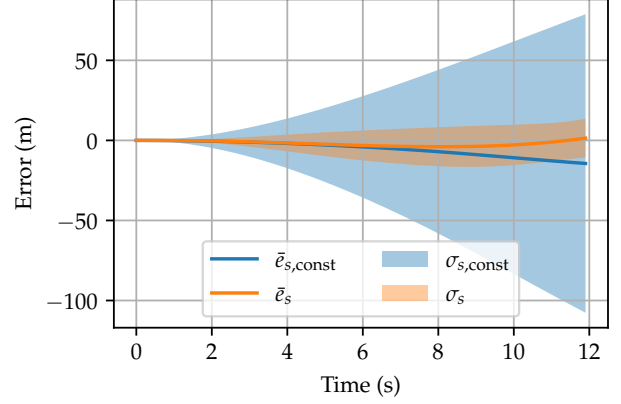


Fig. 4. Mean and standard deviation of the position estimation errors for the constant velocity estimator $\bar{e}_{s,const}/\sigma_{s,const}$ and the presented algorithm \bar{e}_s/σ_s along the path obtained from the PP component evaluated on recorded data.

and estimated trajectory state. Obviously, the constraints of any observed race car acceleration data could be of any shape, but the representational capacity of the constraint assumptions have to be traded off for a fast and reliable real-time execution in the LLNLP. According to our experience, the approximation with either 4 (box only) or 8 (adding diagonals) constraints achieved the best performance.

2) *Validation of the velocity profile estimation*: Using the same recorded real-world trajectory as in IV-B.1 and also its estimated constraints a_{max} as seen in Fig. 3, we use the HLNLP to estimate the parameters w . All estimated parameters together are then forwarded to the LLNLP, which predicts the velocity and the progress along the given curve by solving the nonlinear program. The results are compared to the standard constant velocity predictor, which is often used in robotic applications [21] and that assumes a vehicle progression with the measured constant velocity. In Fig. 4, the mean position error of the presented algorithm \bar{e}_s is compared to the mean position error of the constant velocity predictor $\bar{e}_{s,const}$. Furthermore, the standard deviations σ_s and $\sigma_{s,const}$ are compared respectively. In Fig. 5 the prediction velocity error \bar{e}_v and its standard deviation σ_v of the presented algorithm are compared to the mean error and standard deviation of the velocity of the constant velocity estimator, that are $\bar{e}_{v,const}$ and $\sigma_{v,const}$. The presented algorithm outperforms the constant velocity predictor significantly, although in a short prediction horizon the errors are similar.

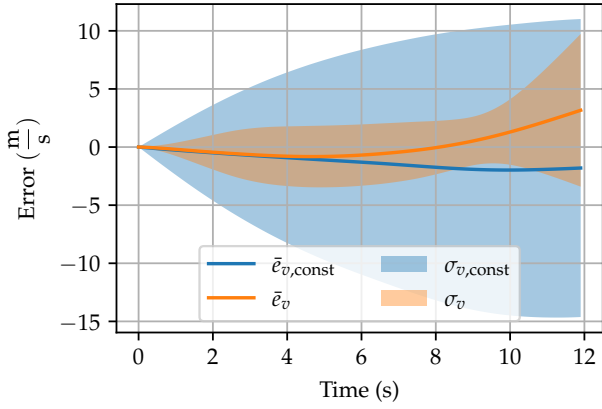


Fig. 5. Mean and standard deviation of the velocity estimation errors for the constant velocity estimator $\bar{e}_{v,\text{const}}/\sigma_{v,\text{const}}$ and the presented algorithm \bar{e}_v/σ_v along the path obtained from the PP component evaluated on recorded data.

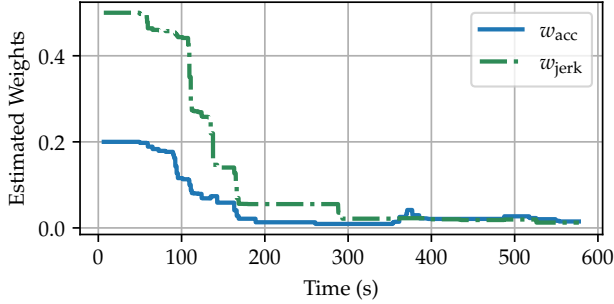


Fig. 6. Weight estimates for the jerk and acceleration weighting obtained by the HLNLP component. Low weights correspond to aggressive driving that is only limited by the velocity and acceleration constraints.

C. Validation of the full algorithm

The algorithm was evaluated with two opponent vehicles in a simulation environment as shown in Fig. 1. The two opponent race cars follow a racing line that was computed by a semi-analytic velocity profile computation as shown in [22] together with differently parameterized racing paths according to [14]. Therefore, the resulting trajectories are not in the solution space of the LLNLP and consequently can not be approximated exactly, which is similar to real observations. The HLNLP estimates the weight parameters and keeps converging to a semi-stationary solution after approximately 200 seconds as shown in Fig. 6. The convergence behavior depends heavily on the choice of hyperparameters, particularly on the arrival weight P in (8). After the weights converged, the predictions of all active components (*all components*) were compared to other estimation algorithms. First, the initial parameter setting was simulated, where the weights and constraints were kept constant and only the LLNLP was active (*LLNLP*). Secondly, a constant velocity estimation was used, where the path was computed by means of the PP component, but the velocity was set constant to the observed velocity (*constant velocity*). Fig. 7 shows the comparison of the three settings by evaluating the Euclidean

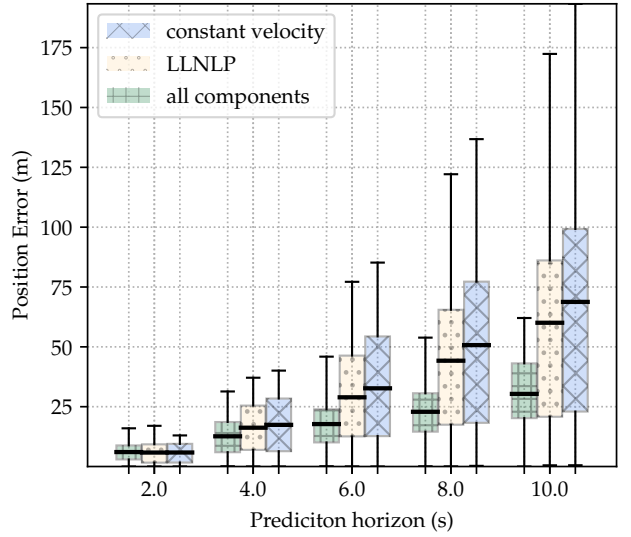


Fig. 7. Box plot statistics (mean, standard deviation, maximum and minimum values) of the Euclidean position error of the prediction compared to the ground truth for different prediction algorithms and at particular prediction horizons

position error after certain prediction horizons. It can be seen that for increasing prediction horizons, the differences in the error measures becomes large, due to the acceleration constraints related to curves and the integrating errors. For short prediction horizons, the constant velocity estimator is performing similarly in our test cases, which was also observed in [21]. The prediction performance with respect to the Euclidean position error was further compared by deactivating either the CQP or the HLNLP part. Fig. 8 shows the comparison with either components active (*CQP active* or *HLNLP active*), with all parameters fixed (*fixed parameters*) or with the full algorithm (*all active*) at a prediction horizon of 6 and 8 seconds. The results looked similar for all observed race cars. In our simulation the biggest improvement originated from the HLNLP part which can be seen in Fig. 8 and which is due to the rather aggressive driving behavior of the observed vehicles and the moderate initialization of the corresponding weight parameters of the LLNLP.

V. CONCLUSIONS

The paper presents a novel approach for predicting race car trajectories in real time and with sparse observation data. It is shown that the algorithm works in an embedded setting and yields satisfying predictions. The key advantage of using an optimization problem as a predictor is the natural integration of constraints. Nevertheless, the quality of the solution is restricted by the assumptions related to the low-level problem, e.g., which norms are used as penalties and what quantities are supposed to be penalized. The expressiveness of the low-level problem is limited due to its KKT conditions arising in a high-level optimization problem, which poses a non-smooth optimization problem with no guaranteed solution. Yet, in practice, the problem as stated is

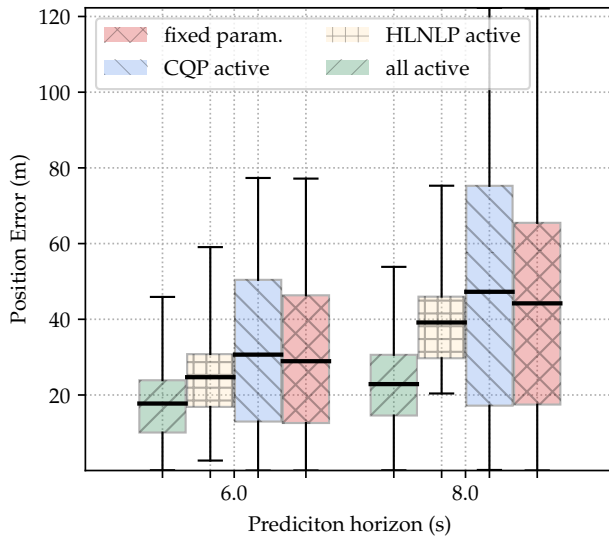


Fig. 8. Box plot statistics (mean, standard deviation, maximum and minimum values) of the Euclidean position error of the prediction compared to the ground truth for different active components at a prediction horizon of 6 and 8 seconds.

posed well enough to be solvable by means of a robust solver like *IPOPT*. Future investigations might include an algorithm that also estimates an uncertainty measure and updates the arrival cost correspondingly and investigating rich function approximators in various parts of the algorithm to achieve a vanishing error, as the number of samples increases.

ACKNOWLEDGMENT

This research was supported by DFG via Research Unit FOR 2401 and project 424107692 and by the EU via ELO-X 953348. The authors thank Jonathan Frey, Katrin Baumgärtner, Jasper Hoffmann, Martin Kirchengast and the other members of the *Autonomous Racing Graz* team for their valuable input to this work.

REFERENCES

- [1] F. Leon and M. Gavrilescu, "A review of tracking and trajectory prediction methods for autonomous driving," *Mathematics*, vol. 9, no. 6, 2021. [Online]. Available: <https://www.mdpi.com/2227-7390/9/6/660>
- [2] S. Capobianco, L. M. Millefiori, N. Forti, P. Braca, and P. Willett, "Deep learning methods for vessel trajectory prediction based on recurrent neural networks," *CoRR*, vol. abs/2101.02486, 2021. [Online]. Available: <https://arxiv.org/abs/2101.02486>
- [3] J. Zhang, H. Liu, Q. Chang, L. Wang, and R. X. Gao, "Recurrent neural network for motion trajectory prediction in human-robot collaborative assembly," *CIRP Annals*, vol. 69, no. 1, pp. 9–12, 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0007850620300998>
- [4] A. Ip, L. Irio, and R. Oliveira, "Vehicle trajectory prediction based on lstm recurrent neural networks," in *2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring)*, 2021, pp. 1–5.
- [5] N. Nikhil and B. T. Morris, "Convolutional neural network for trajectory prediction," 2018.
- [6] S. Gros and M. Zanon, "Data-driven economic nmpe using reinforcement learning," *IEEE Transactions on Automatic Control*, vol. 65, no. 2, p. 636–648, Feb 2020. [Online]. Available: <http://dx.doi.org/10.1109/TAC.2019.2913768>

- [7] A. Agrawal, B. Amos, S. T. Barratt, S. P. Boyd, S. Diamond, and J. Z. Kolter, "Differentiable convex optimization layers," *CoRR*, vol. abs/1910.12430, 2019. [Online]. Available: <http://arxiv.org/abs/1910.12430>
- [8] S. L. Cleac'h, M. Schwager, and Z. Manchester, "Lucidgames: Online unscented inverse dynamic games for adaptive trajectory prediction and planning," *CoRR*, vol. abs/2011.08152, 2020. [Online]. Available: <https://arxiv.org/abs/2011.08152>
- [9] S. Le Cleac'h, M. Schwager, and Z. Manchester, "Algames: A fast solver for constrained dynamic games," *Robotics: Science and Systems XVI*, Jul 2020. [Online]. Available: <http://dx.doi.org/10.15607/RSS.2020.XVI.091>
- [10] M. Menner, P. Worsnop, and M. N. Zeilinger, "Constrained inverse optimal control with application to a human manipulation task," *IEEE Transactions on Control Systems Technology*, vol. 29, no. 2, pp. 826–834, 2021.
- [11] K. Mombaur, A. Truong, and J.-P. Laumond, "From human to humanoid locomotion—an inverse optimal control approach," *Auton. Robots*, vol. 28, pp. 369–383, 04 2010.
- [12] M. Menner and M. N. Zeilinger, "Convex formulations and algebraic solutions for linear quadratic inverse optimal control problems," *2018 European Control Conference (ECC)*, pp. 2107–2112, 2018.
- [13] A. Sinha, P. Malo, and K. Deb, "A review on bilevel optimization: From classical to evolutionary approaches and applications," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 2, pp. 276–295, 2018.
- [14] R. Reiter and M. Diehl, "Parameterization approach of the frenet transformation for model predictive control of autonomous vehicles," *Proceedings of the European Control Conference (ECC)*, 2021.
- [15] R. Reiter, M. Kirchengast, D. Watenig, and M. Diehl, "Mixed-integer optimization-based planning for autonomous racing with obstacles and rewards," *Proceedings of the IFAC Conference on Nonlinear Model Predictive Control (NMPC)*, 2021.
- [16] R. Verschuere, G. Frison, D. Kouzoupis, J. Frey, N. van Duijkeren, A. Zanelli, B. Novoselnik, J. Frey, T. Albin, R. Quirynen, and M. Diehl, "acados: a modular open-source framework for fast embedded optimal control," *arXiv preprint; accepted at: Mathematical Programming Computation*, 2019. [Online]. Available: <https://arxiv.org/abs/1910.13753>
- [17] J. B. Rawlings, D. Q. Mayne, and M. M. Diehl, *Model Predictive Control: Theory, Computation, and Design*, 2nd ed. Nob Hill, 2017.
- [18] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2006.
- [19] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – a software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.
- [20] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: an operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020. [Online]. Available: <https://doi.org/10.1007/s12532-020-00179-2>
- [21] C. Schöller, V. Aravantis, F. Lay, and A. C. Knoll, "The simpler the better: Constant velocity for pedestrian motion prediction," *CoRR*, vol. abs/1903.07933, 2019. [Online]. Available: <http://arxiv.org/abs/1903.07933>
- [22] E. Velenis and P. Tsiotras, "Optimal velocity profile generation for given acceleration limits: Theoretical analysis," 07 2005, pp. 1478 – 1483 vol. 2.