IEEE Open Journal of
Intelligent Transportation
Systems

Received 30 May 2024; revised 16 July 2024; accepted 5 October 2024. Date of publication 9 October 2024; date of current version 21 October 2024.

Digital Object Identifier 10.1109/OJITS.2024.3476540

Computationally Efficient Minimum-Time Motion Primitives for Vehicle Trajectory Planning

MATTIA PICCININI [1] (Member, IEEE), SIMON GOTTSCHALK [2], MATTHIAS GERDTS [2], AND FRANCESCO BIRAL [1]

[1] Department of Industrial Engineering, University of Trento, 38123 Trento, Italy
[2] Institute for Applied Mathematics and Scientific Computing, Department of Aerospace Engineering, Universität der Bundeswehr München, 85577 Neubiberg, Germany

CORRESPONDING AUTHOR: M. PICCININI (e-mail: mattia.piccinini@unitn.it)

ABSTRACT In the context of vehicle trajectory planning, motion primitives are trajectories connecting pairs of boundary conditions. In autonomous racing, motion primitives have been used as computationally faster alternatives to model predictive control, for online obstacle avoidance. However, the existing motion primitive formulations are either simplified and suboptimal, or computationally expensive for accurate collision avoidance. This paper introduces new motion primitives for autonomous racing, aiming to accurately approximate the minimum-time vehicle trajectories while ensuring computational efficiency. We present a novel neural network, named PathPoly-NN, whose internal architecture is designed to learn the minimum-time vehicle path. Our motion primitives combine PathPoly-NN with a fast forward-backward method to compute the minimum-time speed profile. Compared to existing neural networks, PathPoly-NN generalizes better with small training sets, and it has better accuracy in approximating the minimum-time path. Additionally, our motion primitives have lower computational burden and higher accuracy than existing methods based on cubic polynomials and G^2 clothoid curves. Finally, the motion primitives of this paper achieve similar maneuver times as minimum-time economic nonlinear model predictive control (E-NMPC), but with significantly lower computational load (two orders of magnitude). The results open promising perspectives of applications in graph-based trajectory planners for autonomous racing.

INDEX TERMS Autonomous driving, autonomous racing, optimal control, motion primitives, minimum-lap-time simulations, neural networks, trajectory planning.

I. INTRODUCTION

AUTONOMOUS vehicle racing has been gaining interest in the research community [1], as a testbed to accelerate the development of autonomous driving technologies. Indeed, circuits are safe environments to validate new trajectory planning and control algorithms for autonomous driving near the vehicle limits. Recently, graph-based algorithms [2], [3], [4], [5], [6], [7] have been developed to deal with dynamic obstacles in autonomous racing. As depicted in the example of Fig. 1, these algorithms explore a graph of *motion primitives*, which are vehicle trajectories to connect pairs of waypoints, within a fixed [5] or dynamically defined [2], [6] mesh. The best maneuver (*i.e.,* set of motion primitives) in the graph is finally selected according to a certain cost function. The optimality and feasibility of the output maneuver depends on the motion primitives, which need to accurately model the minimum-time trajectories (path and speed profile) and have a low computational cost. However, the existing motion primitive formulations are either simplified and highly suboptimal [3], [5], or computationally expensive [2], [6], thus limiting the online trajectory planning capabilities.

This paper proposes novel motion primitives to accurately approximate the minimum-time trajectories, while being computationally efficient. Before discussing the main contributions, let us critically review the related work.

The review of this article was arranged by Associate Editor Johannes Betz.

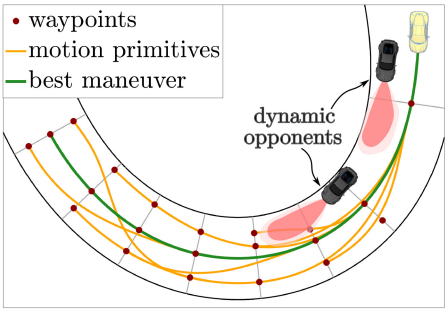


FIGURE 1. Example of use of minimum-time motion primitives, within a graph-based trajectory planner for overtaking moving opponents, like the ones of [2], [3]. Each motion primitive is a vehicle trajectory (path + speed profile) connecting two waypoints.

A. RELATED WORK

The existing techniques to compute motion primitives for autonomous racing can be classified into three main categories: optimization-based, geometric curve-based, and learning-based.

1) OPTIMIZATION-BASED METHODS

Optimal control problems (OCPs) and model predictive control (MPC) are popular tools to compute the minimum-time vehicle trajectory, respectively on a full circuit lap [8] or receding horizons [9], [10], [11], [12]. However, race-like scenarios with dynamic opponents make MPC computationally expensive, due to non-convex constraints. Some authors, like [13], [14], convexified the obstacle constraints. However, their methods were not designed for racing scenarios with multiple dynamic opponents. Computationally efficient motion primitives are therefore needed as a viable alternative to MPC.

In the last decade, approximate methods were developed to speed up the solution of OCPs. The authors of [15], [16] developed two-step algorithms, which iteratively updated the minimum-curvature vehicle path with convex optimization, and then computed the minimum-time speed profile with a forward-backward (*FW-BW*) integration. However, the minimum-curvature path may be far from the minimum-time one [16]. In [17], an iterative approach was used to compute the minimum-time speed profile for a given path. A major limitation of [15], [16], [17] is their high computational times (several seconds): they are not suitable for online motion primitives computation. In [18], a lookup table of precomputed motion primitives was used for online planning of drifting maneuvers. In [19], minimum-time motion primitives were developed with trapezoidal speed profiles and acceleration constraints. Their method was applied for online trajectory planning at low speed, but the computational times were not reported.

Other researchers adopted time-dependent polynomial motion primitives, for autonomous vehicles [20] and drones [21], [22], [23]. However, most of these primitives were derived from minimum-jerk OCPs, which are suboptimal for racing scenarios.

Finally, the authors of [24] presented a fast FW-BW scheme, which computes a semi-analytical minimum-time speed profile for a given path. Their method will be integrated into our motion primitives (Section III).

2) GEOMETRIC CURVE-BASED METHODS

Other authors used geometric curves to define the vehicle path for motion primitives, within graph-based trajectory planners. In [6], G^1 clothoid curves were used to connect pairs of waypoints, by matching the initial and final waypoints' positions and tangent angles. Approximate G^1 clothoids were also used in [25]. In similar planners, [3], [5] used cubic polynomials to connect the waypoints. Such polynomials were built by matching the desired initial and final waypoints' positions and tangents. However, when interconnecting multiple G^1 clothoids or cubic polynomials, curvature jumps may occur, which hinders the execution of the planned paths with real cars. To overcome this issue, the authors of [2] used G^2 clothoids, yet with a higher computational cost. In [26], curvature-continuous Bézier curves were adopted, but only for offline trajectory planning. Nonetheless, clothoid and spline curves may not accurately model the minimum-time vehicle path, since they are geometric curves that neglect the current vehicle speed and dynamics.

3) LEARNING-BASED METHODS

The authors of [27] devised probabilistic motion primitives (ProMPs) to model the variability of the human race drivers' trajectories. However, their ProMPs were used only for global trajectory planning on a full circuit lap, while they employed G^1 clothoids for local planning.

In [28], a deep neural network (DNN) was used to approximate the minimum-jerk OCP solutions. Following [29], the DNN of [28] was trained to satisfy the KKT conditions of the OCP, which provided better generalization than a classical supervised learning. However, their application neglected the longitudinal vehicle dynamics. The authors of [30] used a DNN to approximate the minimum-time vehicle path. However, their DNN needed a huge training dataset with over 6000 racetracks (real or artificial), it was not designed to ensure the path curvature continuity, and it neglected the current vehicle speed.

4) CRITICAL SUMMARY

To the best of our knowledge, the existing approaches to compute motion primitives for real-time autonomous racing are limited by at least one of the following aspects:

- MPC-based methods are limited by their computational load, which hinders the online calculation of many motion primitives in racing scenarios with multiple dynamic opponents.
- Geometric curves are suboptimal, since they neglect the vehicle speed and dynamics. G^1 clothoids and the cubic polynomials of [3], [5] are curvature-discontinuous,

while G^2 clothoids are computationally expensive for accurate collision checking.
- Neural networks with general-purpose architectures require huge training datasets to generalize in unseen scenarios, and they were not designed to return a curvature-continuous path.

B. CONTRIBUTIONS AND PAPER STRUCTURE

The main paper contributions are:

- A novel neural network, named *PathPoly-NN*, whose internal structure is conceived to accurately model the minimum-time vehicle path on a variable prediction horizon, while ensuring the curvature continuity. PathPoly-NN is able to generalize in unseen scenarios, even with small training sets.
- Given the PathPoly-NN path, the minimum-time speed profile is computed with the fast forward-backward (FW-BW) method of [24], which considers the vehicle longitudinal dynamics and a speed-dependent g-g diagram, to constrain the lateral and longitudinal accelerations. The combination of PathPoly-NN and FW-BW yields our minimum-time motion primitives.
- A comparison of PathPoly-NN with general-purpose neural networks, cubic polynomials and G^2 clothoid curves, in terms of accuracy in approximating the minimum-time path and computational burden.
- A comparison of our motion primitives with minimum-time economic nonlinear MPC (E-NMPC), in terms of maneuver times and computational load.

This paper is structured as follows. Section II introduces the PathPoly-NN architecture and training method, while Section III defines the minimum-time motion primitives. Section IV presents the results and the comparison with existing methods. Finally, Section V concludes the paper and discusses the future work.

II. LEARNING THE MINIMUM-TIME VEHICLE PATH

In graph-based trajectory planners (Fig. 1), several motion primitives need to be computed at every planning step. For example, in [2], many motion primitives are evaluated every time a new waypoint is sampled, to find its best parent waypoint and rewire the graph. However, if the motion primitives are not fast to evaluate, the planner will explore a limited number of paths, and a highly suboptimal maneuver may be selected.

In this section, we introduce a novel neural network, named PathPoly-NN, as a computationally efficient approximator of the minimum-time vehicle path. The following subsections describe the original architecture of PathPoly-NN, and the training method.

A. NEURAL NETWORK ARCHITECTURE

The internal architecture of PathPoly-NN is designed to approximate the minimum-time path on a variable prediction horizon, for given boundary conditions on the vehicle states and for a given road curvature profile.

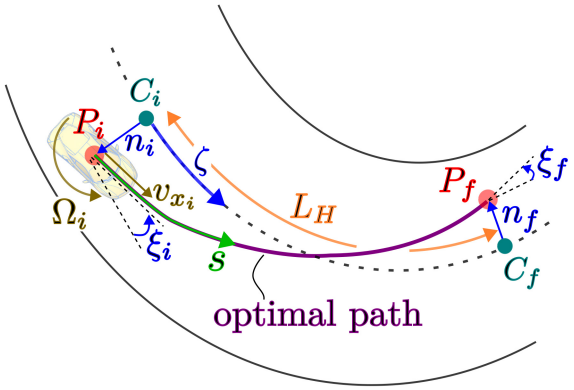


FIGURE 2. PathPoly-NN approximates the minimum-time vehicle path joining the waypoints P_i and P_f. The inputs of PathPoly-NN are the boundary conditions on the vehicle states and the road curvature parameters, given in (3).

1) ROAD MODEL AND PREDICTION HORIZON

PathPoly-NN computes the vehicle path joining the waypoints P_i and P_f (Fig. 2), whose distance along the road center-line is L_H. The horizon length L_H can vary in the continuous range $[L_{H_{\min}}, L_{H_{\max}}]$, where $L_{H_{\min}}$ and $L_{H_{\max}}$ are design parameters (more details in Section II-A7).

The vehicle path is expressed using the curvilinear coordinates $\{\zeta, n, \xi\}$ [31]. Referring to Fig. 2, ζ is the curvilinear abscissa of the vehicle motion along the center-line, n is the lateral coordinate of the vehicle center, and ξ is the relative yaw angle. For a given horizon length L_H, the curvilinear abscissa ζ ranges in $[0, L_H]$, where $\zeta = 0$ corresponds to the initial waypoint C_i, projection of P_i on the center-line (Fig. 2).

To consider the geometry of the road center-line over the prediction horizon, PathPoly-NN uses the road curvature κ_i in the initial center-line point C_i, and the curvature variation $\Delta\kappa$ over the horizon.

2) OUTPUTS OF PATHPOLY-NN

For a given prediction horizon $\zeta \in [0, L_H]$, PathPoly-NN approximates the minimum-time path by returning the following quantities:

- The lateral coordinate profile $n(\zeta)$ of the vehicle path, by combining neural polynomial functions (Section II-A5).
- The derivative $n'(\zeta) = \mathrm{d}n(\zeta)/\mathrm{d}\zeta$, by analytical differentiation of $n(\zeta)$.
- The relative yaw angle $\xi(\zeta)$, using the following curvilinear coordinate equations[1] (see [31]):

$$\begin{cases} \dfrac{\mathrm{d}\zeta(t)}{\mathrm{d}t} = v_\zeta(t) = \dfrac{v_x(t)\cos(\xi(t))}{1 - n(t)\,\kappa(\zeta(t))} & \text{(1a)} \\ \dfrac{\mathrm{d}n(t)}{\mathrm{d}t} = v_x(t)\sin(\xi(t)) & \text{(1b)} \end{cases}$$

[1]Equations (1a)-(1b) assume that the vehicle lateral velocity v_y is negligible. Indeed, in high-speed racing maneuvers, v_y is typically a lot smaller than the forward velocity v_x (drift maneuvers are not considered).

where v_x is the forward speed, and $\kappa(\zeta)$ is the curvature profile of the road center-line. The combination of (1a)-(1b) yields $\xi(\zeta)$ as:

$$\begin{aligned} n'(\zeta) &= \tan(\xi(\zeta))\,(1-\kappa(\zeta)\,n(\zeta)) \\ \Longrightarrow \xi(\zeta) &= \arctan\left(\frac{n'(\zeta)}{1-\kappa(\zeta)\,n(\zeta)}\right) \end{aligned} \tag{2}$$

- The derivative $\xi'(\zeta) = \mathrm{d}\xi(\zeta)/\mathrm{d}\zeta$, by analytical differentiation of (2).
- The curvilinear abscissa $s(\zeta)$ and the curvature profile $\kappa_p(\zeta)$ of the vehicle path (Section II-A6).

3) INPUTS OF PATHPOLY-NN

As depicted in Fig. 2, the inputs of PathPoly-NN are:

- The initial conditions (in the point P_i) for the lateral coordinate n_i, the relative yaw angle ξ_i, the forward speed v_{x_i}, the yaw rate Ω_i, the first and second derivatives $\{n_i', n_i''\}$ with respect to ζ.
- The final conditions (in the point P_f) for the lateral coordinate n_f, the relative yaw angle ξ_f, the first and second derivatives $\{n_f', n_f''\}$ with respect to ζ.
- The road curvature κ_i in the initial center-line point C_i, and the curvature variation $\Delta\kappa$ over the horizon.

Let us collect the inputs in the following vectors $\boldsymbol{p}$ and $\tilde{\boldsymbol{p}}$:

$$\begin{aligned} \boldsymbol{p} &= \begin{bmatrix} n_i & \xi_i & v_{x_i} & \Omega_i & n_f & \xi_f & \kappa_i & \Delta\kappa \end{bmatrix}^T \\ \tilde{\boldsymbol{p}} &= \begin{bmatrix} n_i' & n_i'' & n_f' & n_f'' \end{bmatrix}^T \end{aligned} \tag{3}$$

Ensuring the Path Curvature Continuity: When using PathPoly-NN to connect several contiguous path waypoints, the final conditions of the previous motion primitive are used as the initial conditions of the next one. Also, we want the resulting path to be smooth and curvature-continuous. To this end, when connecting multiple primitives, we impose the continuity of the first and second derivatives of the lateral coordinate n at the interface between two primitives.

Specifically, the boundary conditions n_i' (at $\zeta = 0$) and n_f' (at $\zeta = L_H$) are set to impose the continuity of the relative yaw angles ξ_i and ξ_f at the interface between two primitives, and they are computed using (2):

$$\begin{cases} n_i' = \tan(\xi_i)\cdot(1-\kappa_i\,n_i) & \text{(4a)} \\ n_f' = \tan\big(\xi_f\big)\cdot\big(1-(\kappa_i+\Delta\kappa\,L_H)\,n_f\big) & \text{(4b)} \end{cases}$$

The boundary conditions on the second derivatives n_i'' and n_f'' are instead used to ensure the continuity of the path curvature at the interface between two primitives. Indeed, a continuous curvature is fundamental to avoid infeasible jumps in the planned lateral accelerations. More precisely, n_i'' and n_f'' are computed in such a way to match the desired initial and final derivatives $\{\xi_i', \xi_f'\}$ of the vehicle's relative yaw angle. To obtain a relation among $\{n_i'', n_f''\}$ and $\{\xi_i', \xi_f'\}$, we differentiate the expression of $n'(\zeta)$ in (2) with respect to ζ, and evaluate the result at $\zeta = 0$ and $\zeta = L_H$:

$$\begin{cases} n_i'' = \dfrac{\left(-n_i'^2 - (n_i\kappa_i - 1)^2\right)\xi_i' + \left(n_i\Delta\kappa + \kappa_i n_i'\right)n_i'}{n_i\kappa_i - 1} & \text{(5a)} \\ n_f'' = \dfrac{\left(-n_f'^2 - \left((L_H\Delta\kappa + \kappa_i)n_f - 1\right)^2\right)\xi_f'}{(L_H\Delta\kappa + \kappa_i)n_f - 1} + \\ \quad + \dfrac{n_f'\Big(\Delta\kappa\, n_f + (L_H\Delta\kappa + \kappa_i)n_f'\Big)}{(L_H\Delta\kappa + \kappa_i)n_f - 1} & \text{(5b)} \end{cases}$$

4) NORMALIZATION OF INPUTS AND OUTPUT

To ease the numerical convergence of the PathPoly-NN's training process, we normalize all the input quantities in the vectors $\boldsymbol{p}$ and $\tilde{\boldsymbol{p}}$ and the output n. Also, the independent variable ζ is re-scaled in the range $[0, 1]$, so that the normalized horizon length L_H is 1.

5) INTERNAL STRUCTURE OF PATHPOLY-NN

We devise PathPoly-NN as the combination of two neuralized polynomial functions. The intrinsic smoothness of polynomial functions is suited to approximate the lateral coordinate of the minimum-time vehicle path, which has a smooth behavior on short-medium horizons. The optimal path can even be proved to be of polynomial type, but only for minimum-jerk OCPs with linear lateral or longitudinal vehicle dynamics [20], [32]. In the case of nonlinear minimum-time OCPs with more complex vehicle dynamic models, polynomial-based functions can still be used as suitable approximators.

Given the inputs $\boldsymbol{p}$ and $\tilde{\boldsymbol{p}}$ in (3), and a horizon length L_H, PathPoly-NN computes the lateral coordinate profile $n(\zeta)$ of the vehicle path, as a function of the curvilinear abscissa $\zeta \in [0, L_H]$ along the center-line. $n(\zeta)$ is computed by smoothly combining a pair of polynomial neural networks, as follows:

$$\begin{aligned} n(\zeta) = &\overbrace{\left[\left(\sum_{q=1}^{D} f_{1_q}(\boldsymbol{p},\tilde{\boldsymbol{p}})\cdot\zeta^q\right) + n_i\right]}^{\text{first neural polynomial}}\phi\left(\frac{L_H}{2} - \zeta\right) + \\ &+ \underbrace{\left[\left(\sum_{q=1}^{D} f_{2_q}(\boldsymbol{p},\tilde{\boldsymbol{p}})\cdot(\zeta - L_H)^q\right) + n_f\right]}_{\text{second neural polynomial}}\phi\left(\zeta - \frac{L_H}{2}\right) \end{aligned} \tag{6}$$

where the D is the degree of the polynomials, which is a hyperparameter of PathPoly-NN. The activation function $\phi(\cdot)$ smoothly combines the outputs of the two polynomials in (6), and is designed as:

$$\phi : a \mapsto \frac{1}{2}\cdot\left[\frac{\sin\left(\arctan\left(\frac{a}{r_f}\right)\right)}{\sin\left(\arctan\left(\frac{L_H/2}{r_f}\right)\right)} + 1\right] \tag{7}$$

with r_f being a tunable regularization factor. Fig. 3 plots the functions $\{\phi(\frac{L_H}{2} - \zeta), \phi(\zeta - \frac{L_H}{2})\}$ appearing in (6), for

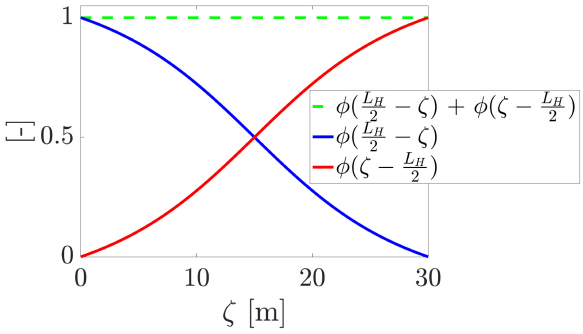


FIGURE 3. Activation functions $\{\phi(\frac{L_H}{2}-\zeta), \phi(\zeta-\frac{L_H}{2})\}$ to combine the two neural polynomials in (6). In this example, $L_H = 30$ m.

FIGURE 4. Internal structure of the neural networks $f_{NN_{j,q}}(p)$, $j \in [1, 2]$, $q \in [1, 2, \ldots, D-2]$, providing the neuralized polynomial terms in (8)-(9). The numbers in green indicate the size of the propagated signals.

$\zeta \in [0, L_H]$ and $L_H = 30$ m. Note that $\phi(\frac{L_H}{2}-\zeta) = 1$ for $\zeta = 0$, which ensures the satisfaction of the imposed initial condition $n(0) = n_i$. Similarly, $\phi(\zeta-\frac{L_H}{2}) = 1$ for $\zeta = L_H$, which ensures that $n(L_H) = n_f$. In the central point of the planning horizon ($\zeta = \frac{L_H}{2}$), we have $\phi(0) = 0.5$, so that the two neural polynomials are weighted equally. Finally, $\phi(\frac{L_H}{2}-\zeta)+\phi(\zeta-\frac{L_H}{2}) = 1$, $\forall \zeta \in [0, L_H]$.

Exact Matching of the Boundary Conditions: The internal architecture (6) of PathPoly-NN is designed to exactly satisfy the imposed boundary conditions on the vehicle's curvilinear coordinates $\{n_i, \xi_i, n_f, \xi_f\}$ and the derivatives $\{n_i', n_i'', n_f', n_f''\}$, which are part of the inputs (3).

The functions $\{f_{1_q}(\cdot), f_{2_q}(\cdot)\}$ in (6) are designed as:

$$f_{1_q}(\boldsymbol{p}, \tilde{\boldsymbol{p}}) = \begin{cases} f_{\mathrm{NN}_{1,q}}(\boldsymbol{p}), & q \in [1, 2, \ldots, D-2] \\ f_{\mathrm{analy}_{1,q}}(\tilde{\boldsymbol{p}}), & q \in [D-1, D] \end{cases} \tag{8}$$

$$f_{2_q}(\boldsymbol{p}, \tilde{\boldsymbol{p}}) = \begin{cases} f_{\mathrm{NN}_{2,q}}(\boldsymbol{p}), & q \in [1, 2, \ldots, D-2] \\ f_{\mathrm{analy}_{2,q}}(\tilde{\boldsymbol{p}}), & q \in [D-1, D] \end{cases} \tag{9}$$

where $\{f_{\mathrm{NN}_{1,q}}(\boldsymbol{p}), f_{\mathrm{NN}_{2,q}}(\boldsymbol{p})\}$, $q \in [1, 2, \ldots, D-2]$, are two-layer feedforward neural networks, whose internal structure is depicted in Fig. 4. These neural networks have two fully connected (FC) layers: the first layer has n_{neur} neurons and a tanh activation function, while the second layer has 1 neuron and a linear activation function. The number of neurons n_{neur} is a hyperparameter of PathPoly-NN.

The functions $\{f_{\mathrm{analy}_{1,q}}(\tilde{\boldsymbol{p}}), f_{\mathrm{analy}_{2,q}}(\tilde{\boldsymbol{p}})\}$ in (8)-(9), $q \in [D-1, D]$, are instead non-trainable and are derived analytically, to satisfy the desired boundary conditions $\{n_i', n_i'', n_f', n_f''\}$. Specifically, the following four constraint equations are solved for $f_{\mathrm{analy}_{1,D-1}}(\tilde{\boldsymbol{p}})$, $f_{\mathrm{analy}_{2,D-1}}(\tilde{\boldsymbol{p}})$, $f_{\mathrm{analy}_{1,D}}(\tilde{\boldsymbol{p}})$, $f_{\mathrm{analy}_{2,D}}(\tilde{\boldsymbol{p}})$:

$$\begin{cases} n_i' = \dfrac{\mathrm{d}n(\zeta)}{\mathrm{d}\zeta}\Big|_{\zeta=0}, \quad n_f' = \dfrac{\mathrm{d}n(\zeta)}{\mathrm{d}\zeta}\Big|_{\zeta=L_H} & \text{(10a)} \\ n_i'' = \dfrac{\mathrm{d}^2n(\zeta)}{\mathrm{d}\zeta^2}\Big|_{\zeta=0}, \quad n_f'' = \dfrac{\mathrm{d}^2n(\zeta)}{\mathrm{d}\zeta^2}\Big|_{\zeta=L_H} & \text{(10b)} \end{cases}$$

where $n(\zeta)$ is given by (6). As will be explained in Section II-A3, we use neural polynomials of degree $D = 4$ in (6), which provides the best trade-off between complexity and accuracy. Remembering that the curvilinear abscissa ζ is normalized in $[0, 1]$, equations (10a)-(10b) can be analytically solved for $f_{\mathrm{analy}_{1,3}}(\tilde{\boldsymbol{p}})$, $f_{\mathrm{analy}_{2,3}}(\tilde{\boldsymbol{p}})$, $f_{\mathrm{analy}_{1,4}}(\tilde{\boldsymbol{p}})$, $f_{\mathrm{analy}_{2,4}}(\tilde{\boldsymbol{p}})$:

$$\begin{cases} f_{\mathrm{analy}_{1,3}}(\tilde{\boldsymbol{p}}) = \frac{5}{8}n_f'' - \frac{17}{4}n_f' - 4n_i + 4n_f - 3f_{\mathrm{NN}_{1,1}}(\boldsymbol{p}) + \\ \quad -2f_{\mathrm{NN}_{1,2}}(\boldsymbol{p}) + \frac{13}{4}f_{\mathrm{NN}_{2,1}}(\boldsymbol{p}) - \frac{5}{4}f_{\mathrm{NN}_{2,2}}(\boldsymbol{p}) \\ f_{\mathrm{analy}_{2,3}}(\tilde{\boldsymbol{p}}) = -\frac{5}{8}n_i'' - \frac{17}{4}n_i' - 4n_i + 4n_f + \frac{13}{4}f_{\mathrm{NN}_{1,1}}(\boldsymbol{p}) + \\ \quad -3f_{\mathrm{NN}_{2,1}}(\boldsymbol{p}) + 2f_{\mathrm{NN}_{2,2}}(\boldsymbol{p}) + \frac{5}{4}f_{\mathrm{NN}_{1,2}}(\boldsymbol{p}) \\ f_{\mathrm{analy}_{1,4}}(\tilde{\boldsymbol{p}}) = -\frac{5}{8}n_f'' + 3n_f' + 3n_i - 3n_f + 2f_{\mathrm{NN}_{1,1}}(\boldsymbol{p}) + \\ \quad +f_{\mathrm{NN}_{1,2}}(\boldsymbol{p}) - 2f_{\mathrm{NN}_{2,1}}(\boldsymbol{p}) + \frac{5}{4}f_{\mathrm{NN}_{2,2}}(\boldsymbol{p}) \\ f_{\mathrm{analy}_{2,4}}(\tilde{\boldsymbol{p}}) = -\frac{5}{8}n_i'' - 3n_i' - 3n_i + 3n_f + 2f_{\mathrm{NN}_{1,1}}(\boldsymbol{p}) + \\ \quad +\frac{5}{4}f_{\mathrm{NN}_{1,2}}(\boldsymbol{p}) - 2f_{\mathrm{NN}_{2,1}}(\boldsymbol{p}) + f_{\mathrm{NN}_{2,2}}(\boldsymbol{p}) \end{cases} \tag{11}$$

6) ADDITIONAL OUTPUTS OF PATHPOLY-NN

As anticipated in Section II-A2, additional quantities are computed by processing the outputs of PathPoly-NN.

In this section, we focus on computing the curvilinear abscissa $s(\zeta)$ and the curvature profile $\kappa_p(\zeta)$ of the vehicle path. Both $s(\zeta)$ and $\kappa_p(\zeta)$ are needed to later compute the minimum-time vehicle speed profile, as will be described in Section III.

Fig. 2 highlights the difference between the variables s and ζ: s is the curvilinear abscissa along the vehicle path, measured from the initial vehicle pose P_i to the final pose P_f. ζ is instead the curvilinear abscissa along the road center-line, measured from C_i to C_f (projections of P_i and P_f on the center-line). Similarly, $\kappa_p(\zeta)$ and $\kappa(\zeta)$ are the curvature profiles of the vehicle path and of the road center-line, respectively.

To compute $s(\zeta)$, let us substitute the definition of the vehicle speed $v_x = \frac{\mathrm{d}s}{\mathrm{d}t}$ into (1a), which yields:

$$\frac{\mathrm{d}s(\zeta)}{\mathrm{d}\zeta} = \frac{1 - n(\zeta)\,\kappa(\zeta)}{\cos(\xi(\zeta))} \Longrightarrow s(\bar{\zeta}) = \int_0^{\bar{\zeta}} \frac{1 - n(\zeta)\,\kappa(\zeta)}{\cos(\xi(\zeta))}\,\mathrm{d}\zeta \tag{12}$$

The integral in (12) is computed numerically, using an array of ζ values at which PathPoly-NN is evaluated.

Finally, the path curvature $\kappa_p(\zeta)$, $\zeta \in [0, L_H]$, can be computed from its definition:

$$\kappa_p = \frac{\mathrm{d}\psi}{\mathrm{d}s} = \frac{\mathrm{d}(\theta+\xi)}{\mathrm{d}s} = \frac{\mathrm{d}(\theta+\xi)}{\mathrm{d}\zeta}\frac{\mathrm{d}\zeta}{\mathrm{d}s} = \left(\kappa + \frac{\mathrm{d}\xi}{\mathrm{d}\zeta}\right)\frac{\mathrm{d}\zeta}{\mathrm{d}s} \tag{13}$$

where ψ is the heading angle of the vehicle path, θ and κ are the heading angle and the curvature of the road center-line. The expression (13) is fully determined: κ is known, $\frac{d\xi}{d\zeta}$ is the analytical derivative of (2), and $\frac{d\zeta}{ds}$ is the inverse of (12).

7) OPERATION WITH VARIABLE HORIZON LENGTHS

One of the PathPoly-NN's possible applications is in graph-based trajectory planners (Fig. 1), where waypoints need to be connected by minimum-time primitives, and the waypoints spacing L_H is variable.

To approximate the time-optimal path for different values of L_H, we train a set of PathPoly-NNs, with each network being trained for a certain value of L_H. More precisely, we train $N_{\text{NN}} = 20$ neural networks, whose corresponding values of L_H range in $[L_{H_{\min}}, L_{H_{\max}}]$. In this work, the upper bound $L_{H_{\max}}$ is set to 45 m, which enables the calculation of the time-optimal vehicle path by combining a pair of neural polynomials (Section II-A5). Longer prediction horizons can be reached by concatenating more motion primitives, or by combining more than two neural polynomials. Considering the use of PathPoly-NN inside the graph-based planner of [2], 45 m are enough to connect single pairs of waypoints on many real-world racetracks. The lower bound $L_{H_{\min}}$ is set to 4 m, to ensure an accurate path calculation in tight corners.

For a given horizon length $L_H{}^*$, we select the PathPoly-NN network trained for the closest value of L_H, and we use it to compute the time-optimal path. However, since $L_H{}^*$ may not exactly match any of the values of L_H used for training, the curvilinear abscissa ζ in (6) is re-scaled in the range $[0, L_H{}^*]$, to obtain the desired horizon length and still satisfy the imposed boundary conditions.

8) NUMBER OF EVALUATION POINTS OF PATHPOLY-NN

Depending on the connection length L_H, we use a different number of ζ points to evaluate PathPoly-NN. The number of evaluation points N_{pts} is proportional to L_H, to ensure a suitably dense discretization of the time-optimal path, which is helpful for two reasons:

- Accurate collision checking with possible obstacles.
- Accurate numerical calculation of the integral (12) for the curvilinear abscissa $s(\zeta)$.

Since N_{pts} affects the computational times of PathPoly-NN, the value of N_{pts} is varied depending on L_H. Specifically, the set of $N_{\text{NN}} = 20$ trained neural networks is subdivided into 5 groups, depending on the L_H value used for training. The first group, trained for small L_H values, is evaluated with $N_{\text{pts}} = 100$ points. The second, third, fourth and fifth groups are evaluated with respectively 200, 300, 400 and 500 points, so that the sampling density is at least 10 points per meter.

B. NEURAL NETWORK TRAINING

PathPoly-NN is trained using supervised learning, to approximate the numerical solutions of a minimum-time economic nonlinear MPC problem (E-NMPC). We solve the E-NMPC problems in a receding horizon fashion, on a set of racetracks and with different prediction horizons.

1) E-NMPC PROBLEM USED FOR TRAINING

The minimum-time E-NMPC problem used to train PathPoly-NN is formulated as follows:

$$\min_{\boldsymbol{u}\in\mathcal{U}} \int_0^{L_H} \frac{w_T}{v_\zeta(\zeta)}\,\mathrm{d}\zeta \tag{14a}$$

$$\text{s.t.}\begin{cases} \dfrac{\mathrm{d}\boldsymbol{x}(\zeta)}{\mathrm{d}\zeta} = \dfrac{\boldsymbol{f}(\boldsymbol{x}(\zeta),\boldsymbol{u}(\zeta))}{v_\zeta(\zeta)}, & \text{(14b)}\\ \boldsymbol{b}(\boldsymbol{x}(0),\boldsymbol{x}(L_H)) = 0, & \text{(14c)}\\ \boldsymbol{c}(\boldsymbol{x}(\zeta),\boldsymbol{u}(\zeta)) \le 0 & \text{(14d)} \end{cases}$$

The cost function (14b) minimizes the maneuver time, where w_T is a weight factor, and $v_\zeta(\zeta)$ is given by (1a). The dynamics (14c) is the kineto-dynamical vehicle model of [33], whose time-domain equations are:

$$\begin{cases} \dot{v}_x(t) = a_x(t) & \text{(15a)}\\ \tau_{a_x}\dot{a}_x(t) + a_x(t) = a_{x_c}(t) & \text{(15b)}\\ \tau_\Omega(v_x(t))\,\dot{\Omega}(t) + \Omega(t) = \Omega_{c_s}(t)\,\dfrac{\overline{a}_y}{v_x(t)} & \text{(15c)}\\ \dot{\xi}(t) = \Omega(t) - \kappa(\zeta(t))\left(\dfrac{v_x(t)\cos(\xi(t))}{1-n(t)\,\kappa(\zeta(t))}\right) & \text{(15d)}\\ \text{(1b)} \end{cases}$$

The model states are $\boldsymbol{x} = [v_x, a_x, \Omega, \xi, n]$, being respectively the longitudinal speed and acceleration, the yaw rate, the relative yaw angle and the lateral coordinate. a_{x_c} is the longitudinal acceleration control, and is subject to a (fast) first-order dynamics (15b), with τ_{a_x} being a small time constant. Similarly, the yaw rate Ω evolves with the first-order model (15c), where the time constant $\tau_\Omega(v_x)$ is an identified second-order polynomial function. Ω_{c_s} in (15c) is the yaw rate control, normalized in the range $[-1, 1]$, while $\overline{a}_y/v_x$ is the maximum achievable yaw rate, with $\overline{a}_y$ being the maximum lateral acceleration. The initial and final states are imposed with (14d), while the inequality constraints (14d) are the following:

$$\begin{cases} \underline{a}_x \le a_x(\zeta) - c_1 v_x(\zeta) - c_2 v_x^2(\zeta) \le \overline{a}_x & \text{(16a)}\\ \left|a_y(\zeta)\right| = \left|\Omega(\zeta)\right| v_x(\zeta) = \left|\kappa_p(s(\zeta))\right| v_x^2(\zeta) \le \overline{a}_y & \text{(16b)}\\ -1 \le \Omega_{c_s}(\zeta) \le 1 & \text{(16c)}\\ n(\zeta) \ge \dfrac{W}{2} - M_R(\zeta), \quad n(\zeta) \le M_L(\zeta) - \dfrac{W}{2} & \text{(16d)} \end{cases}$$

The inequalities (16a)-(16b) impose aspeed-dependent g-g diagram constraint, described by the parameters $\{c_1, c_2\}$ (linear and quadratic drag), the minimum and maximum longitudinal accelerations $\{\underline{a}_x, \overline{a}_x\}$, and the maximum lateral acceleration $\overline{a}_y$. (16c) limits the normalized yaw rate control, and (16d) enforces the road boundaries. In (16d), the functions $M_R(\zeta)$ and $M_L(\zeta)$ provide the distances from the road center-line to the right and left road boundaries, while W is the vehicle width.

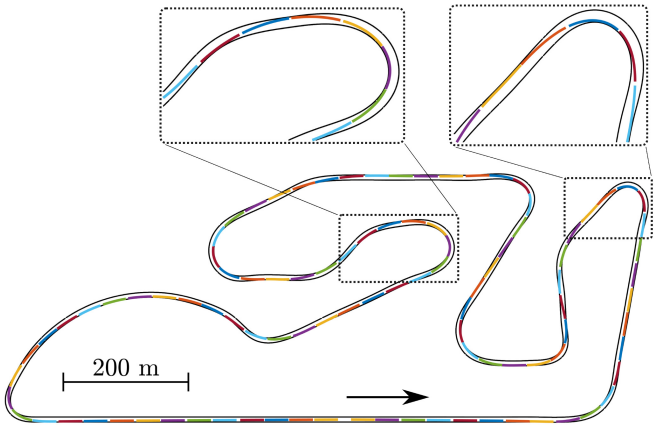


FIGURE 5. Minimum-time E-NMPC solutions (colored trajectories) computed numerically on the Valencia circuit, and used to train PathPoly-NN. In the example of this plot, the horizon length of E-NMPC is $L_H = 38$ m.

The OCP solver Pins [34] is adopted to compute the solutions of the E-NMPC problems.

As discussed in [9], the kineto-dynamical model (15a)-(16a) has a simple yet accurate formulation, and can capture the transient nonlinear vehicle dynamics without relying on complex tire models. Our motion primitives are conceived to be integrated into a high-level trajectory planner, to compute feasible maneuvers in complex racing scenarios. The planned maneuvers can then be tracked by low-level controllers, like the ones in [9], considering more detailed steering dynamic models and tire-road interactions.

We underline that PathPoly-NN could be trained to approximate the solutions of any minimum-time E-NMPC problem for trajectory planning, subject to any vehicle model and constraints. The kineto-dynamical vehicle model used in this paper is just an example of application.

2) TRAINING DATASET AND IMPLEMENTATION DETAILS

The training set consists of E-NMPC solutions on the following circuits: *Valencia* (Spain), *Misano* (Italy), *Imola* (Italy), *Pista Azzurra* (Italy), *Catalunya* (Spain). An independent test dataset contains the E-NMPC solutions on the *Adria* circuit (Italy). Fig. 5 shows some E-NMPC solutions on the Valencia circuit. For each circuit, the training and test datasets are generated for different values of the prediction horizon $L_H \in [L_{H_{\min}}, L_{H_{\max}}]$. For each training example (E-NMPC trajectory), we store in the training set the (normalized) vectors of inputs $\boldsymbol{p}$ and $\tilde{\boldsymbol{p}}$ (3), and the output profile $n(\zeta)$.

The total number of training examples is 18000 (5 circuits), which is significantly smaller than the 2.7 million examples (6000 circuits) used to train the neural network of [30]. Indeed, the specialized architecture of PathPoly-NN yields a better generalization with small training sets, as will be shown in Section IV.

The loss function used to train PathPoly-NN is the root mean square error (RMSE) between the output $n(\zeta)$ of PathPoly-NN and the E-NMPC solutions, both of which are evaluated on $N_o = 20$ points, evenly spaced in the range $\zeta \in [0, L_H]$. PathPoly-NN is formulated with the Tensorflow framework, and is trained with the Adam optimizer [35].

TABLE 1. Hyperparameters of PathPoly-NN, tuned with a grid search.

Hyperparameter	Value
Number of neurons n_{neur}	7
Neural polynomial degree D	4
Regularization factor r_f	1.0

3) HYPERPARAMETERS TUNING

The main hyperparameters of PathPoly-NN are:

1) The number of neurons n_{neur} in the fully connected layers of the neural terms $f_{\text{NN}_{1,q}}(\boldsymbol{p})$ and $f_{\text{NN}_{2,q}}(\boldsymbol{p})$ in (8)-(9), $q \in [1, 2, \ldots, D-2]$;
2) The degree D of the neural polynomials in (6);
3) The regularization factor r_f in (7).

A grid search method is adopted to tune the hyperparameters above. The grid search is performed by evaluating the accuracy (RMSE) of PathPoly-NN on the test dataset. For the sake of brevity, we omit the grid search results, and we report in Table 1 the selected hyperparameters.

The number of trainable parameters for PathPoly-NN is:

$$N_{\text{pars}} = 2 \cdot \left[\sum_{q=1}^{D-2} \left(\overbrace{n_{\text{neur}} \cdot n_{\text{inputs}} + n_{\text{neur}}}^{\text{weights FC layers}} + \overbrace{n_{\text{neur}} + 1}^{\text{biases FC layers}} \right) \right] \tag{17}$$

where $n_{\text{inputs}} = 8$ is the number of inputs in $\boldsymbol{p}$ (3), and FC stands for fully connected layer. For the hyperparameters in Table 1, the number of trainable parameters is:

$N_{\text{pars}} = 2 \cdot 2 \cdot (7 \cdot 8 + 7 + 7 + 1) = 284$.

III. MINIMUM-TIME MOTION PRIMITIVES

Our motion primitives consist of two building blocks:

- A minimum-time path, computed by PathPoly-NN.
- A minimum-time vehicle speed profile, computed by the fast forward-backward (FW-BW) algorithm of [24].

The FW-BW algorithm computes the time-optimal speed profile to traverse a given path, defined by a sequence of curvilinear abscissae s and curvatures κ_p. More precisely, the FW-BW returns a semi-analytical solution of the following OCP:

$$\min_{a_x \in \mathcal{A}} \quad T$$
$$\text{s.t.} \begin{cases} \text{(15a)}, \quad \dot{s}(t) = v_x(t) \\ v_x(0) = v_{x_i}, v_x(T) \le v_{x_f}, \quad s(0) = 0, s(T) = L \\ \text{(16a)}, \text{(16b)} \end{cases}$$

where T is the maneuver time, v_{x_i} is the initial speed, v_{x_f} is an upper bound on the final speed, and L is the path length. The acceleration a_x is the control input, and thespeed-dependent g-g acceleration constraints (16a)-(16b) are the same as in the E-NMPC problem (16a). The authors of [24] proved that their FW-BW method returns the global minimum-time trajectory, for a given path and constraints.

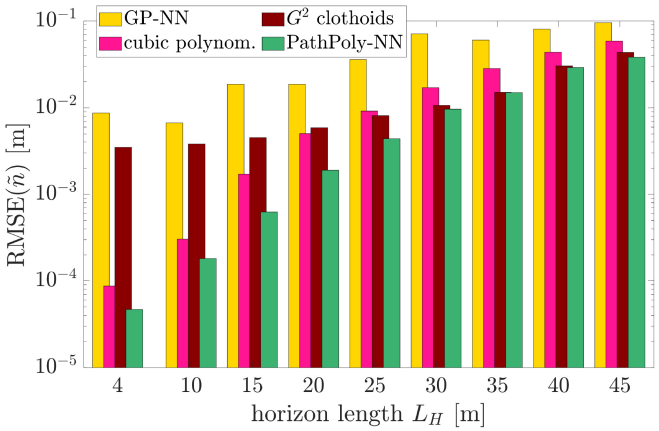


FIGURE 6. Comparing the proposed PathPoly-NN with the benchmarks on the test dataset, for different values of horizon length L_H. The plot shows the RMS of the lateral coordinate prediction error ($\tilde{n}$). Our PathPoly-NN outperforms the benchmarks for all the considered values of L_H.

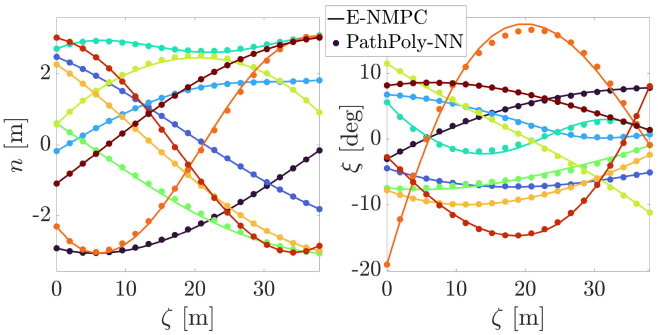


FIGURE 7. Comparing the lateral coordinates n and relative yaw angles ξ computed by PathPoly-NN (dots) and E-NMPC (solid line), for different boundary conditions and road curvatures. The plots show part of the test dataset, for a horizon length $L_H = 38$ m.

IV. RESULTS

In this section, we evaluate our minimum-time motion primitives using the following criteria:

- Approximation accuracy of the vehicle path (lateral coordinate n, relative yaw angle ξ) and speed profile, in comparison with minimum-time E-NMPC solutions.
- Computational efficiency, as a function of the horizon length L_H and the number of evaluation points.

The trajectories returned by our motion primitives are compared with the following methods:

1) Minimum-time E-NMPC (Section IV-A), used to train our PathPoly-NN.
2) Minimum-lap-time optimal control (Section IV-B).
3) A *general-purpose* neural network (Section IV-C), whose internal structure is similar to [30].
4) Cubic polynomials (Section IV-D), used by [3], [5].
5) G^2 clothoid curves (Section IV-E), used by [2].

In this paper, the parameters of the kineto-dynamical model (15a)-(16a) for E-NMPC are identified to fit the dynamics and performance limits of the validated double-track sports car model in [36, Ch. 4]. The identification procedure is the same as in [9].

Our motion primitives and all the benchmark methods are implemented in C++ code, and are executed on a laptop with a 2.6 GHz 6-Core Intel i7 processor.

A. EVALUATION AND COMPARISON WITH E-NMPC

This section evaluates the performance of our motion primitives, and compares them with the minimum-time E-NMPC problem (14c). We remind that the same E-NMPC is also used to generate the training and test datasets for our PathPoly-NN (Section IV-A1).

1) ACCURACY OF PATHPOLY-NN

Table 2 reports the accuracy of PathPoly-NN on the test dataset, for different values of horizon length L_H. The table shows the RMS values of the lateral coordinate ($\tilde{n}$) and relative yaw angle ($\tilde{\xi}$) prediction errors. Fig. 6 plots the results of Table 2, regarding the $\tilde{n}$ errors.

Considering that n ranges in $[-\frac{W_r}{2}, \frac{W_r}{2}]$, with W_r being the local circuit width ($W_r > 8$ m for most racetracks), the RMSE values of our PathPoly-NN in Table 2 are very small for all the considered values of L_H. As shown in Fig. 6, the RMSE values of PathPoly-NN increase for larger values of L_H, but they are still small (3.81 cm) for $L_H = L_{H_{\max}} = 45$ m.

Fig. 7 analyzes part of the test dataset, for a horizon length $L_H = 38$ m. The plots show that PathPoly-NN accurately approximates the lateral coordinate and the relative yaw angle of the minimum-time vehicle path, for different boundary conditions and road curvatures.

2) COMPUTATIONAL EFFICIENCY

Table 3 compares the meanand maximum computational (CPU) times obtained by E-NMPC and by our motion primitives, which use PathPoly-NN to approximate the time-optimal path, and the FW-BW algorithm to optimize the speed profile.

The mean CPU times are computed as the average of 300 planning steps on the Adria circuit, for each horizon length L_H. In the case of E-NMPC, the initial solver guess is the solution of the previous planning step, which represents a very good warm-start. Despite warm-starting, themean CPU times of E-NMPC are in the order of $5-7$ ms,while the maximum CPU times can reach 42 ms. In contrast, our motion primitives yield CPU times around $50-100$ μs (last two columns in Table 3), which are two orders of magnitude lower than E-NMPC. Interestingly, the computational loads of the E-NMPC and FW-BW algorithms are related to the road geometry, and the CPU times typically decrease in straight or constant-curvature road segments. On the contrary, the computational times of our PathPoly-NN are weakly related to the road geometry.

In graph-based trajectory planners, like the one in Fig. 1, many motion primitives need to be evaluated at every planning step. Hence, the computational times of a single

TABLE 2. Comparing the proposed PathPoly-NN with the benchmarks on the test dataset, for different values of horizon length L_H. The table reports the RMS values of the lateral coordinate $\tilde{n}$ and the relative yaw angle $\tilde{\xi}$ prediction errors. Our PathPoly-NN outperforms the benchmarks for all L_H.

Horizon length L_H	RMSE($\tilde{n}$)				RMSE($\tilde{\xi}$)		
	GP-NN (benchmark)	**cubic polynom.** (benchmark)	G^2 **clothoids** (benchmark)	**PathPoly-NN**	**cubic polynom.** (benchmark)	G^2 **clothoids** (benchmark)	**PathPoly-NN**
4 m	0.8697 cm	0.0086 cm	0.3477 cm	**0.0046** cm	0.0040 deg	0.0571 deg	**0.0027** deg
15 m	1.8656 cm	0.1707 cm	0.4527 cm	**0.0625** cm	0.0234 deg	0.0664 deg	**0.0101** deg
25 m	3.6080 cm	0.9173 cm	0.8127 cm	**0.4367** cm	0.0775 deg	0.0830 deg	**0.0423** deg
35 m	6.0332 cm	2.8315 cm	1.5085 cm	**1.4955** cm	0.1734 deg	0.1096 deg	**0.1030** deg
45 m	8.4700 cm	5.8748 cm	4.3383 cm	**3.8183** cm	0.2557 deg	0.1940 deg	**0.1834** deg

TABLE 3. Comparing the CPU times (mean and max values) to compute a single motion primitive, using the proposed method (last two columns) and the benchmarks. The proposed method is two orders of magnitude faster than E-NMPC, and at least one order of magnitude faster than cubic polynomials and G^2 clothoids. These computational times were obtained on a 2.6 GHz 6-Core Intel i7 processor.

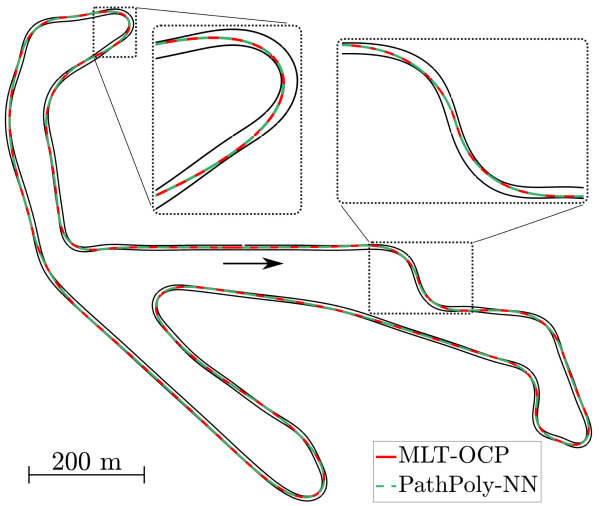
Horizon length L_H	CPU time							
	E-NMPC (benchmark)		**cubic polynomials + FW-BW** (benchmark)		G^2 **clothoids + FW-BW** (benchmark)		**PathPoly-NN + FW-BW**	
	Mean	Max	Mean	Max	Mean	Max	Mean	Max
20 m	4534.6 µs	22509.0 µs	361.1 µs	460.8 µs	303.7 µs	550.6 µs	**46.2** µs	70.1 µs
35 m	5407.2 µs	31229.4 µs	754.7 µs	956.4 µs	592.0 µs	823.3 µs	**48.5** µs	82.8 µs
45 m	7016.0 µs	42163.3 µs	1190.3 µs	1423.0 µs	752.8 µs	968.5 µs	**53.4** µs	103.1 µs

primitive impact the overall re-planning rates and solution quality. The results in Table 3 show that our motion primitives are computationally efficient, and can be used for online trajectory planning.

B. COMPARISON WITH A MINIMUM-LAP-TIME OCP

As previously explained, PathPoly-NN is trained to approximate the minimum-time E-NMPC solutions, computed on short planning horizons. Let us now evaluate how PathPoly-NN performs against a minimum-lap-time optimal control problem (MLT-OCP), solved offline on a full lap. The MLT-OCP has the same structure as the E-NMPC problem (14c), but it is solved on an entire lap. The MLT-OCP is solved using Pins [34]. Other examples of MLT-OCPs can be found in [9], [10], [16].

Fig. 8 compares the vehicle path computed by the MLT-OCP and PathPoly-NN, on the Misano circuit. In the plot, we evaluate PathPoly-NN with a horizon length $L_H = 35$ m, and we concatenate the PathPoly-NN solutions to obtain a full lap. As shown in the plot and the table of Fig. 8, PathPoly-NN closely approximates the MLT-OCP trajectory, with small RMS deviations for the lateral coordinate n and the relative yaw angle ξ.



RMS$(n - n_{\text{MLT}})$	**RMS**$(\xi - \xi_{\text{MLT}})$
1.6126 cm	0.1264 deg

FIGURE 8. Comparing the vehicle path computed by a minimum-lap-time optimal control problem (MLT-OCP) and PathPoly-NN, on the Misano circuit. PathPoly-NN is evaluated with a horizon length $L_H = 35$ m, and the solutions are concatenated to obtain a full lap. The table reports the RMS of the deviations of the lateral coordinate n and the relative yaw angle ξ.

C. COMPARISON WITH A GENERAL-PURPOSE NN

This section compares the novel PathPoly-NN with a general-purpose neural network (GP-NN). The internal structure of GP-NN is depicted in Fig. 9: GP-NN is a feedforward neural network, with three fully connected layers. The first and second layers have respectively N_{n_1} and N_{n_2} neurons, and tanh activation functions. The third layer has N_{o} neurons and a linear activation function. The GP-NN's structure is similar to the one used by [30].

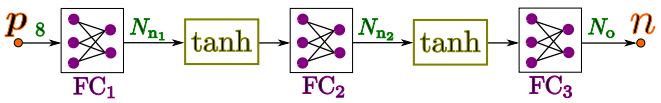


FIGURE 9. Internal structure of the GP-NN neural network, used as a benchmark. The numbers in green indicate the size of the propagated signals.

Like PathPoly-NN, the GP-NN is trained to approximate the lateral coordinate $n(\zeta)$ of the vehicle path, using the same (normalized) inputs $\boldsymbol{p}$ given in (3). More precisely,

TABLE 4. CPU times to evaluate our PathPoly-NN and the benchmarks, with different numbers of evaluation points. The CPU times of PathPoly-NN are almost two orders of magnitude lower than the benchmarks, which makes a huge difference in online trajectory planning.

N. evaluation points	Mean CPU time		
	cubic polynomials (benchmark)	G^2 clothoids (benchmark)	PathPoly-NN
50	149.7 µs	90.1 µs	**1.8** µs
200	492.2 µs	336.4 µs	**5.1** µs
300	713.4 µs	550.2 µs	**7.4** µs

GP-NN outputs $n(\zeta)$ on $N_\text{o} = 20$ points, evenly spaced in the range $\zeta \in [0, L_H]$. The hyperparameters of GP-NN are tuned with a grid search, to obtain the best accuracy on the test dataset. The tuned values of N_{n_1} and N_{n_2} are 64 and 128, respectively. The number of trainable parameters for GP-NN is 11476, which is considerably higher than the number of parameters of PathPoly-NN (284).

GP-NN is formulated in Tensorflow, and is trained with the Adam optimizer [35], using the same loss function and training hyperparameters employed for PathPoly-NN.

Table 2 and Fig. 6 compare the accuracy (RMS of the prediction error $\tilde{n}$) of PathPoly-NN and GP-NN on the test dataset, for different values of horizon length L_H. Despite its lower number of parameters (284 vs 11476), our PathPoly-NN significantly outperforms GP-NN for all the considered values of L_H: the difference is above or around one order of magnitude for $L_H \leq 30$ m. The main reasons of the better accuracy of PathPoly-NN are:

- The internal structure of PathPoly-NN is tailored to the problem of learning the minimum-time vehicle path, for which neural polynomial-like functions are suitable approximators. On the contrary, GP-NN has a general-purpose structure, which shows poor generalization when tested on new data.
- The structure of PathPoly-NN is designed to exactly satisfy the imposed boundary conditions on n, ξ and their derivatives (n_i', n_i'', n_f', n_f''), which improves the accuracy and smoothness of the approximated path.

As shown in [30], the GP-NN would require a very large training dataset to achieve acceptable performance. In contrast, PathPoly-NN is able to learn and generalize the minimum-time path with a much smaller training set, thanks to its tailored internal structure. PathPoly-NN belongs to the class of neural networks with domain-specific architectures, which is an emerging research field [9], [37].

D. COMPARISON WITH CUBIC POLYNOMIALS

Let us now compare our PathPoly-NN with cubic polynomial curves, which were used as motion primitives for autonomous racing in [3], [5]. We adopt the same cubic polynomial interpolation problem of [5, Sec. IV], which matches the Cartesian coordinates of the waypoints to be connected, and their relative yaw angles $\{\xi_i, \xi_f\}$.

As shown in Table 2 and Fig. 6, cubic polynomials are more accurate than the GP-NN, but they are outperformed by our PathPoly-NN, due to the following reasons:

- The cubic polynomials of [3], [5] are only C^1-continuous, and they do not match the desired boundary conditions on n_i'' and n_f'' in (3). In contrast, PathPoly-NN matches these boundary conditions, yielding curvature-continuous paths.
- Cubic polynomials are the result of a geometric optimization problem, which does not consider the minimum-travel-time objective.

Table 3 shows that the mean CPU times of the cubic polynomial primitives are significantly higher than our primitives (754.7 μs versus 48.5 μs when $L_H = 35$ m). The computational burden of cubic polynomials is due to two factors. First, the polynomial parameters need to be computed with an iterative procedure [5]. Second, the evaluation of the curvilinear coordinates $\{\zeta, n, \xi\}$ is computationally expensive, since it involves determining the closest point on the circuit center-line to a given point on the curve.

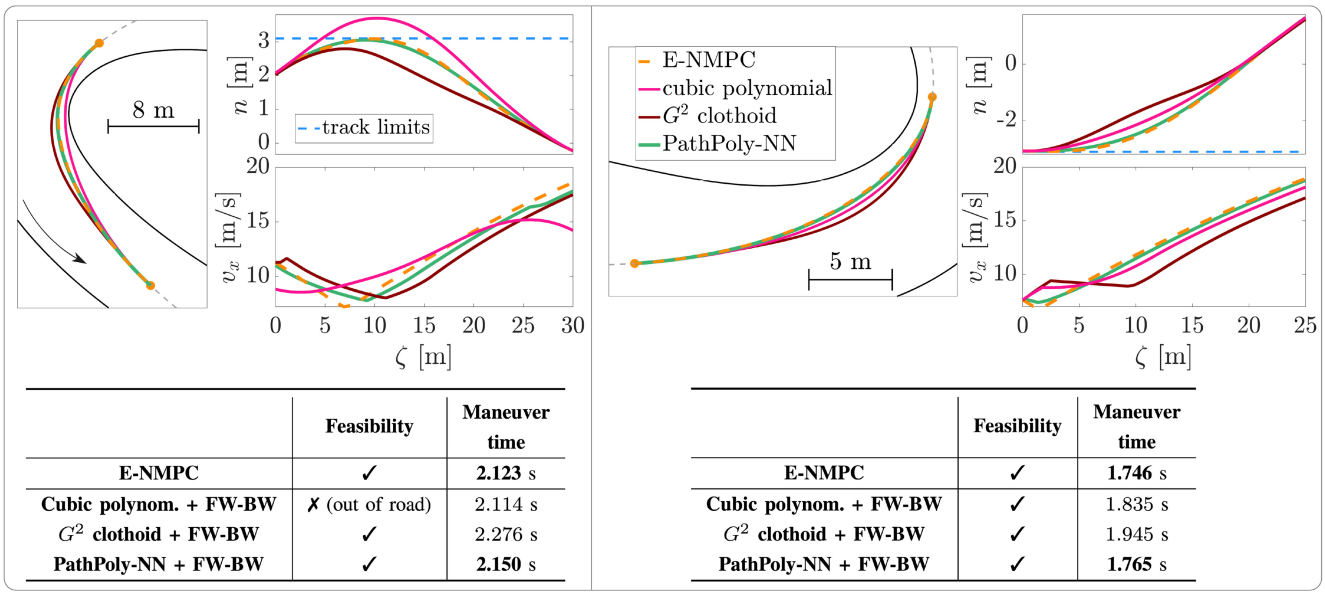
To perform accurate collision checking between a motion primitive and an obstacle trajectory, the number of evaluation points on the primitive must be sufficiently high. As reported in Table 4, the computational times of cubic polynomials increase with the number of evaluation points, and the CPU time difference with PathPoly-NN remains around two orders of magnitude. This limits the use of cubic polynomials for online trajectory planning and accurate obstacle avoidance. In contrast, our PathPoly-NN is computationally efficient even with many evaluation points.

E. COMPARISON WITH CLOTHOID CURVES

This section compares our PathPoly-NN with G^2 clothoid curves, which were used as vehicle motion primitives in [2]. Following [38], the G^2 Hermite interpolation problem involves finding the clothoid parameters that match given initial and final positions, yaw angles and curvatures. In [38], the clothoid parameters are computed with a Newton scheme, which solves a nonlinear system of equations. In our paper, we use the C++ clothoid library of [39], which is an efficient implementation of [38].

As reported in Table 2 and Fig. 6, G^2 clothoids are more accurate than cubic polynomials when the horizon length L_H is larger than 20 m. However, clothoids are outperformed by our PathPoly-NN, for all the considered values of L_H. Indeed, clothoids were proved to approximate the optimal vehicle path only at constant speed [40], which is not the case in racing applications.

Tables 3 and 4 show that the computational times of G^2 clothoids are lower than cubic polynomials, but they are still around one order of magnitude higher than our PathPoly-NN. Indeed, the clothoid parameters are the solution of a nonlinear system, which is solved iteratively. Moreover, like cubic polynomials, the evaluation of the clothoid curvilinear



	Feasibility	Maneuver time
E-NMPC	✓	**2.123** s
Cubic polynom. + FW-BW	✗ (out of road)	2.114 s
G^2 clothoid + FW-BW	✓	2.276 s
PathPoly-NN + FW-BW	✓	**2.150** s

	Feasibility	Maneuver time
E-NMPC	✓	**1.746** s
Cubic polynom. + FW-BW	✓	1.835 s
G^2 clothoid + FW-BW	✓	1.945 s
PathPoly-NN + FW-BW	✓	**1.765** s

FIGURE 10. Trajectories generated by our PathPoly-NN and the benchmarks (cubic polynomials and G^2 clothoids), on two corners of the Pista Azzurra circuit. The plots compare the vehicle paths, the lateral coordinates n, and the vehicle speed v_x. The tables report the maneuver feasibility and the travel times. The trajectories and maneuver times returned by our method are the closest to the minimum-time E-NMPC solutions.

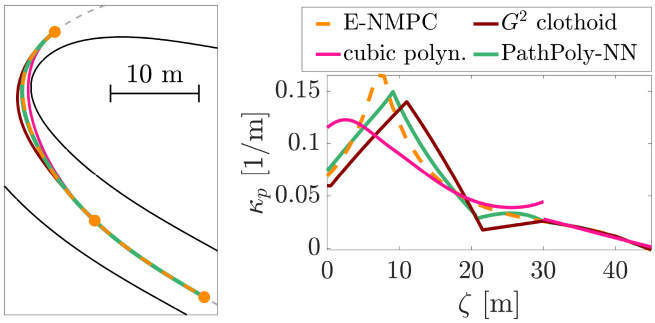


FIGURE 11. Concatenation of two motion primitives, on a corner of the Pista Azzurra circuit. Unlike the cubic polynomials of [3], [5], our PathPoly-NN provides a continuous path curvature κ_p. G^2 clothoids are curvature-continuous as well, but their curvature profile is further from the E-NMPC solution.

coordinates $\{\zeta, n, \xi\}$ is computationally expensive, especially when the number of evaluation points is high (Table 4).

F. ANALYSIS OF THE VEHICLE TRAJECTORIES

Fig. 10 shows the trajectories generated by our PathPoly-NN, and compares them with the benchmark cubic polynomials, G^2 clothoids, and E-NMPC (GP-NN is excluded since its accuracy is lower than the other benchmarks). This analysis focuses on two corners of the Pista Azzurra circuit. In both scenarios of Fig. 10, the paths computed by PathPoly-NN are the closest to the minimum-time E-NMPC solution.The corresponding speed profiles, computed by the FW-BW algorithm, are also close to the E-NMPC ones, yet with local differences due to the different path curvatures. Conversely, cubic polynomials and G^2 clothoids yield suboptimal or infeasible trajectories.

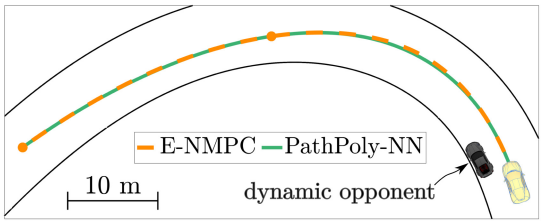
In the scenario of Fig. 10(a), the cubic polynomial's path is not feasible, as the left side of the car would go off the road.In contrast, PathPoly-NN generates a feasible trajectory, which is very close to the E-NMPC one. Interestingly, the speed profile of our primitive locally differs from the E-NMPC one, due to the different path curvatures. More precisely, the E-NMPC is slower up to the corner apex, but is faster in the corner exit. Considering the whole maneuver, E-NMPC is faster than our primitive by only 27 ms. The G^2 clothoid's solution is feasible, but it is 153 ms slower than the E-NMPC. In the scenario of Fig. 10(b), PathPoly-NN yields again the fastest trajectory among the benchmarks, with a maneuver time only 19 ms higher than the E-NMPC.

1) MOTION PRIMITIVES CONCATENATION

Fig. 11 plots the concatenation of two motion primitives. In the transition between two primitives, the cubic polynomials of [3], [5] show a path curvature discontinuity. The G^2 clothoids are curvature-continuous, but their curvature profile is not close to the E-NMPC solution. In contrast, our PathPoly-NN provides a continuous curvature profile, which is closer to the E-NMPC solution.

2) OBSTACLE AVOIDANCE EXAMPLE

Fig. 12 shows an example of use of our motion primitives for dynamic obstacle avoidance. The figure depicts the corner n.4 of the Adria circuit, which was never seen during the training of PathPoly-NN. In this example scenario, the ego vehicle needs to overtake a slower opponent (gray car). To perform dynamic obstacle avoidance with the E-NMPC, an additional time-varying ellipse constraint is added to the E-NMPC problem (14c). To approximate the E-NMPC



	CPU time	Maneuver time
E-NMPC	195205.4 µs	3.632 s
PathPoly-NN + FW-BW	**98.1** µs	**3.647** s

FIGURE 12. Example of dynamic obstacle avoidance at the corner n.4 of the Adria circuit, which was never seen during the training of PathPoly-NN. Our motion primitives closely approximate the E-NMPC maneuver (path and travel times), with a noticeable decrease in computational times.

solution in Fig. 12, which covers a planning horizon of around 70 m, we concatenate two of our motion primitives: the initial point coincides with the current ego vehicle pose, while the intermediate point and the final point are obtained from the E-NMPC solution. The vehicle path computed by PathPoly-NN is very close to the E-NMPC, the maneuver time is only 15 ms higher than the E-NMPC, while the computational times are 98.1 μs against 195.2 ms, which is a huge difference for real-world applications. Moreover, in similar obstacle avoidance scenarios, the E-NMPC solution has numerical convergence issues, especially with more than one dynamic obstacle. Both our primitives and the E-NMPC solution are feasible, as they satisfy the obstacle avoidance and road margins constraints, and the imposed performance limits.

Our motion primitives approximate the E-NMPC solutions for the given initial and final waypoints: future work will use our primitives in a graph-based trajectory planner, to search for the minimum-time trajectory in a mesh of waypoints.

3) TRAJECTORY FEASIBILITY EVALUATION

Evaluating the feasibility of the planned trajectories is crucial for real-world applications. As proved in [24], the speed profile computed by the FW-BW algorithm is feasible and time-optimal for a given path, g-g diagram constraints and boundary conditions. However, we need to make sure that the path itself is feasible. The satisfaction of the road margins constraints (16d) can be easily checked with the lateral coordinate profile $n(\zeta)$ computed by PathPoly-NN in (6). If our motion primitives were used inside a graph-based trajectory planner (Fig. 1), the infeasible waypoints connections could be detected and discarded. We underline that, in all our training and testing scenarios, the PathPoly-NN's path was always feasible.

Other sources of infeasibility are related to obstacle avoidance. Static obstacle collisions can be checked by evaluating the distance between the vehicle path and the obstacle position, using the curvilinear coordinates $n(\zeta), \xi(\zeta)$ returned by PathPoly-NN. Collision-checking with dynamic obstacles is out of the scope of this paper, but it can be performed using $n(\zeta), \xi(\zeta)$ and the travel time $t(\zeta)$ computed by our primitives, and the predicted obstacle motion. Once again, a graph-based trajectory planner can discard the waypoints connections leading to obstacle collisions.

V. CONCLUSION AND FUTURE WORK

This paper presented novel motion primitives for minimum-time trajectory planning with autonomous vehicles. We introduced PathPoly-NN, a neural network that learns the minimum-time vehicle path with neural polynomial-like functions. The internal architecture of PathPoly-NN was tailored to learn the minimum-time path, and to satisfy the desired boundary conditions on the curvilinear coordinates and their derivatives. In our motion primitives, PathPoly-NN was coupled with a fast forward-backward algorithm, which computes the minimum-time speed profile for given acceleration constraints.

When trained with the solutions of a minimum-time E-NMPC problem, on 5 racetracks, PathPoly-NN was able to compute the minimum-time path in new scenarios, with high accuracy and computational efficiency.

In comparison with a general purpose neural network, our PathPoly-NN achieved better generalization and accuracy (around one order of magnitude in RMSE), with a much smaller number of parameters and training dataset size. Compared with geometric primitives like cubic polynomials and G^2 clothoids, our primitives were more accurate and faster in computation by one to two orders of magnitude. The curvilinear coordinates of our primitives could be evaluated with many points, to enable accurate collision checking, while still preserving low computational times. When compared with minimum-time E-NMPC, our motion primitives yielded similar vehicle trajectories and maneuver times, with computational times two orders of magnitude lower.

Future work will apply the proposed motion primitives for real-time dynamic obstacle avoidance, using the graph-based trajectory planner in [2].The full planning and control scheme will include the high-level trajectory planner of [2], and the low-level controllers developed in [9]. Also, we will employ the PathPoly-NN to learn the minimum-jerk path in various autonomous driving applications, such as automated parking [41].The FW-BW algorithm will be extended to deal with generically-shaped g-g-v diagrams [42], which will help driving closer to the vehicle limits. Finally, we will test the motion primitives on our small-scale autonomous vehicle [12], to validate their performance in real-world scenarios.

REFERENCES

[1] J. Betz et al., "Autonomous vehicles on the edge: A survey on autonomous vehicle racing," *IEEE Open J. Intell. Transp. Syst.*, vol. 3, pp. 458–488, 2022.

[2] M. Piazza, M. Piccinini, S. Taddei, and F. Biral, "MPTree: A sampling-based vehicle motion planner for real-time obstacle avoidance," *IFAC-PapersOnLine*, vol. 58, no. 10, pp. 146–153, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896324004166

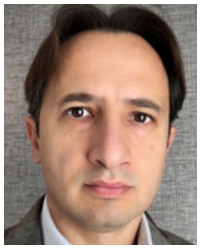
[3] M. Rowold, L. Ögretmen, T. Kerbl, and B. Lohmann, "Efficient spatiotemporal graph search for local trajectory planning on oval race tracks," *Actuators*, vol. 11, no. 11, p. 319, 2022.
[4] L. Ögretmen, M. Rowold, T. Betz, A. Langmann, and B. Lohmann, "A hybrid trajectory planning approach for autonomous rule compliant multi-vehicle oval racing," *SAE Int. J. Connect. Autom. Veh.*, vol. 7, pp. 1–14, Sep. 2023.
[5] T. Stahl, A. Wischnewski, J. Betz, and M. Lienkamp, "Multilayer graph-based trajectory planning for race vehicles in dynamic scenarios," in *Proc. IEEE Intell. Transp. Syst. Conf. (ITSC)*, 2019, pp. 3149–3154.
[6] M. Frego, P. Bevilacqua, E. Bertolazzi, F. Biral, D. Fontanelli, and L. Palopoli, "Trajectory planning for car-like vehicles: A modular approach," in *Proc. IEEE 55th Conf. Decis. Control (CDC)*, 2016, pp. 203–209.
[7] R. Trauth, K. Moller, and J. Betz, "Toward safer autonomous vehicles: Occlusion-aware trajectory planning to minimize risky behavior," *IEEE Open J. Intell. Transp. Syst.*, vol. 4, pp. 929–942, 2023.
[8] M. Massaro and D. J. N. Limebeer, "Minimum-lap-time optimisation and simulation," *Veh. Syst. Dyn.*, vol. 59, no. 7, pp. 1069–1113, 2021.
[9] M. Piccinini, S. Taddei, M. Larcher, M. Piazza, and F. Biral, "A physics-driven artificial agent for online time-optimal vehicle motion planning and control," *IEEE Access*, vol. 11, pp. 46344–46372, 2023.
[10] M. Rowold, L. Ögretmen, U. Kasolowsky, and B. Lohmann, "Online time-optimal trajectory planning on three-dimensional race tracks," in *Proc. IEEE Intell. Veh. Symp. (IV)*, 2023, pp. 1–8.
[11] J. K. Subosits and J. C. Gerdes, "From the racetrack to the road: Real-time trajectory replanning for autonomous driving," *IEEE Trans. Intell. Veh.*, vol. 4, no. 2, pp. 309–320, Jun. 2019.
[12] E. Pagot, M. Piccinini, and F. Biral, "Real-time optimal control of an autonomous RC car with minimum-time maneuvers and a novel kineto-dynamical model," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2020, pp. 2390–2396.
[13] Z. Dong, X. Xu, X. Zhang, X. Zhou, X. Li, and X. Liu, "Real-time motion planning based on MPC with obstacle constraint convexification for autonomous ground vehicles," in *Proc. 3rd Int. Conf. Unmanned Syst. (ICUS)*, 2020, pp. 1035–1041.
[14] J. V. Frasch et al., "An auto-generated nonlinear MPC algorithm for real-time obstacle avoidance of ground vehicles," in *Proc. Eur. Control Conf. (ECC)*, 2013, pp. 4136–4141.
[15] N. R. Kapania, J. Subosits, and J. C. Gerdes, "A sequential two-step algorithm for fast generation of vehicle racing trajectories," *J. Dyn. Syst. Meas. Control*, vol. 138, no. 9, Sep. 2016, Art. no. 91005.
[16] A. Heilmeier, A. Wischnewski, L. Hermansdorfer, J. Betz, M. Lienkamp, and B. Lohmann, "Minimum curvature trajectory planning and control for an autonomous race car," *Veh. Syst. Dyn.*, vol. 58, no. 10, pp. 1497–1527, 2020.
[17] B. Lenzo and V. Rossi, "A simple mono-dimensional approach for lap time optimisation," *Appl. Sci.*, vol. 10, no. 4, p. 1498, 2020.
[18] Y. Lu, B. Yang, J. Li, Y. Zhou, H. Chen, and Y. Mo, "Consecutive inertia drift of autonomous RC car via primitive-based planning and data-driven control," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2023, pp. 4835–4840.
[19] G. Klančar and S. Blažič, "Optimal constant acceleration motion primitives," *IEEE Trans. Veh. Technol.*, vol. 68, no. 9, pp. 8502–8511, Sep. 2019.
[20] M. Da Lio et al., "Artificial co-drivers as a universal enabling technology for future intelligent vehicles and transportation systems," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 1, pp. 244–263, Feb. 2015.
[21] M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient motion primitive for quadrocopter trajectory generation," *IEEE Trans. Robot.*, vol. 31, no. 6, pp. 1294–1310, Dec. 2015.
[22] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Proc. 16th Int. Symp. Robot. Res.*, 2016, pp. 649–666.
[23] Z. Han, Z. Wang, N. Pan, Y. Lin, C. Xu, and F. Gao, "Fast-racing: An open-source strong baseline for SE(3) planning in autonomous drone racing," *IEEE Robot. Autom. Lett.*, vol. 6, no. 4, pp. 8631–8638, Oct. 2021.
[24] M. Frego, E. Bertolazzi, F. Biral, D. Fontanelli, and L. Palopoli, "Semi-analytical minimum time solutions with velocity constraints for trajectory following of vehicles," *Automatica*, vol. 86, pp. 18–28, Dec. 2017.
[25] M. Brezak and I. Petrović, "Real-time approximation of clothoids with bounded error for path planning applications," *IEEE Trans. Robot.*, vol. 30, no. 2, pp. 507–515, Apr. 2014.
[26] J.-W. Choi, R. Curry, and G. Elkaim, *Piecewise Bezier Curves Path Planning with Continuous Curvature Constraint for Autonomous Driving*. Dordrecht, The Netherlands: Springer, 2010, pp. 31–45.
[27] S. Löckel, J. Peters, and P. van Vliet, "A probabilistic framework for imitating human race driver behavior," *IEEE Robot. Autom. Lett.*, vol. 5, no. 2, pp. 2086–2093, Apr. 2020.
[28] S. Gottschalk, M. Gerdts, and M. Piccinini, "Reinforcement learning and optimal control: A hybrid collision avoidance approach," in *Proc. 10th Int. Conf. Veh. Technol. Intell. Transp. Syst. (VEHITS)*, 2024, pp. 76–87.
[29] A. De Marchi, A. Dreves, M. Gerdts, S. Gottschalk, and S. Rogovs, "A function approximation approach for parametric optimization," *J. Optim. Theory Appl.*, vol. 196, no. 1, pp. 56–77, 2023.
[30] S. Garlick and A. Bradley, "Real-time optimal trajectory planning for autonomous vehicles and lap time simulation using machine learning," *Veh. Syst. Dyn.*, vol. 60, no. 12, pp. 4269–4289, 2022.
[31] R. Lot and F. Biral, "A curvilinear abscissa approach for the lap time optimization of racing vehicles," *IFAC Proc. Vol.*, vol. 47, no. 3, pp. 7559–7565, 2014.
[32] M. Werling, J. Ziegler, S. Kammel, and S. Thrun, "Optimal trajectory generation for dynamic street scenarios in a Frenét frame," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2010, pp. 987–993.
[33] M. Piccinini, M. Larcher, E. Pagot, D. Piscini, L. Pasquato, and F. Biral, "A predictive neural hierarchical framework for on-line time-optimal motion planning and control of black-box vehicle models," *Veh. Syst. Dyn.*, vol. 61, no. 1, pp. 83–110, 2023.
[34] F. Biral, E. Bertolazzi, and P. Bosetti, "Notes on numerical methods for solving optimal control problems," *IEEJ J. Ind. Appl.*, vol. 5, pp. 154–166, Mar. 2016.
[35] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Represent.*, 2014, pp. 1–15.
[36] M. Piccinini, "Artificial drivers for online time-optimal vehicle trajectory planning and control," Ph.D. dissertation, Ingegneria industriale, Univ. Trento, Trento, Italy, 2024.
[37] M. D. Lio, M. Piccinini, and F. Biral, "Robust and sample-efficient estimation of vehicle lateral velocity using neural networks with explainable structure informed by kinematic principles," *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 12, pp. 13670–13684, Dec. 2023.
[38] E. Bertolazzi and M. Frego, "On the G2 hermite interpolation problem with clothoids," *J. Comput. Appl. Math.*, vol. 341, pp. 99–116, Oct. 2018.
[39] E. Bertolazzi. "Clothoids: G1 and G2 fitting with clothoids, spline of clothoids, circle arc and biarc." Accessed: May 1, 2024. [Online]. Available: https://github.com/ebertolazzi/Clothoids
[40] T. Fraichard and A. Scheuer, "From reeds and Shepp's to continuous-curvature paths," *IEEE Trans. Robot.*, vol. 20, no. 6, pp. 1025–1035, Dec. 2004.
[41] E. Pagot, M. Piccinini, E. Bertolazzi, and F. Biral, "Fast planning and tracking of complex autonomous parking maneuvers with optimal control and pseudo-neural networks," *IEEE Access*, vol. 11, pp. 124163–124180, 2023.
[42] M. Piccinini, S. Taddei, M. Piazza, and F. Biral, "Impacts of g-g-v constraints formulations on online minimum-time vehicle trajectory planning," *IFAC-PapersOnLine*, vol. 58, no. 10, pp. 87–93, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896324004075

MATTIA PICCININI (Member, IEEE) received the M.Sc. degree (cum laude) in mechatronics engineering and the Ph.D. degree (cum laude) in autonomous systems from the University of Trento, Italy, in 2019 and 2024, respectively.

From March to June 2022, he was a visiting Ph.D. student with the Universität der Bundeswehr, Munich, Germany. He is currently a Postdoctoral Researcher with the University of Trento. From December 2024, he will be a TUM Global Postdoctoral Fellow with the Technical University of Munich. His research focuses on trajectory planning, control, and state estimation methods for urban and racing autonomous vehicles.

SIMON GOTTSCHALK received the M.Sc. degree in industrial mathematics and the Ph.D. degree for the dissertation titled "A Differential Equation-Based Framework for Deep Reinforcement Learning" from TU Kaiserslautern, Germany. He is currently a Postdoctoral Researcher with the Universität der Bundeswehr München. His research interests involve control theory and trajectory planning for autonomous vehicles and satellites using optimal control and reinforcement learning approaches.

MATTHIAS GERDTS received the master's degree in mathematics from the Technical University of Clausthal, Germany, in 1997, and the Ph.D. degree and Habilitation from the University of Bayreuth, Germany, in 2001 and 2006, respectively. Since 2010, he has been a Professor of Engineering Mathematics with the Department of Aerospace Engineering, Universität der Bundeswehr München. His research interests include optimization methods and optimal control techniques.



FRANCESCO BIRAL received the master's degree in mechanical engineering from the University of Padova, Italy, and the Ph.D. degree in mechanism and machine theory from the University of Brescia, Italy, in 2000, for his work on minimum lap time of racing vehicles with the use of optimal control. He is currently an Associate Professor with the Department of Industrial Engineering with the University of Trento. He has 15 years experience in the development and validation of ADAS and AD functions, both for cars and PTWs, gained in several European and industrial funded research projects. His research interests include symbolic and numerical multibody dynamics and optimization and constrained optimal control, mainly in the field of vehicle dynamics with special focus on intelligent vehicles and optimal maneuver for racing vehicles.

Open Access funding provided by 'Università degli Studi di Trento' within the CRUI CARE Agreement