2024 European Control Conference (ECC)
June 25-28, 2024. Stockholm, Sweden

Towards Predictive Path-Following Control using Deep Neural Networks and Path Primitives

Pablo Zometa[1] and Timm Faulwasser[2]

Abstract— **Model Predictive Path Following is an attractive solution for motion control of mobile robots and other autonomous systems. It requires solving a non-convex constrained optimization problem at each sampling period. We focus on a general approach to approximate Model predictive Path-Following Control (MPFC) for a mobile robot using a Deep neural network (DNN) that learns a set of base MPFC representations via path primitives. We show that using simple algebraic operations, any path can be followed with reasonable accuracy, and we illustrate how to apply this approach for paths described by linear segments and parabolic blends that can be generated by a robotic path planning algorithm. Compared to the computational requirements of MPFC, our proposed approach requires significantly less memory and the execution speed is two orders of magnitude faster. This makes our approach suitable for microcontroller implementation, with only a small degradation of the path-following accuracy compared to online MPFC.**

I. Introduction

Model predictive Path-Following Control, i.e., the idea to consider geometric references in Nonlinear Model Predictive Control (NMPC) for motion control problems of mechatronic or robotic systems and also for autonomous vehicles has received widespread research attention, cf. [1]–[6]. In the literature, it also goes under the name model predictive contouring control [7]. However, as with most NMPC schemes, MPFC is based on the online solution of an optimal control problem that involves nonlinear dynamics and thus it usually results in a non-convex optimization problem.

Due to the considerable computational burden of NMPC, the conceptual idea of learning the implicit NMPC feedback law via neural networks is of renewed interest. We refer to [8] for an early reference and, e.g., to [9]–[14] for recent investigations. In the recent conference paper [15], we focus on how to construct an efficient approximation of MPFC, to be deployed on a microcontroller, using neural networks for a *fixed* path. That is, therein we discuss how to learn one Neural Network (NN) approximation for one given path. Additionally, we show how to further reduce the memory requirements of the NN using quantization, and how to improve the path-following performance by adding a simple online controller. Indeed, it appears that the problem of how information about the control reference (setpoints, paths, etc.) can be included in NN approximations of the MPC feedback law has not received widespread attention.

[1]PZ is with the Faculty of Engineering, German International University (Berlin), Germany. pablo.zometa@giu-berlin.de
[2]TF is with the Institute of Energy Systems, Energy Efficiency and Energy Economics, TU Dortmund University, Germany. timm.faulwasser@ieee.org

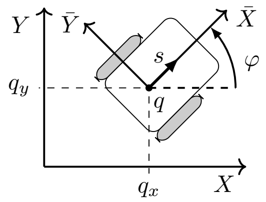

Fig. 1: Differential drive robot and its coordinate systems.

In the present paper, we turn towards the problem of learning an NN approximation of the MPFC feedback which can handle a set of path primitives. These path primitives can be transformed by a translation and rotation in Cartesian space, such that, due to the symmetry properties of the underlying robot model, much more general path references can be followed using the same NN without any re-training. We also argue that with our approach, embedded implementation of MPFC becomes a viable option.

The remainder of the paper is structured as follows: In Section II we recap the problem outline from [15] in compact form, while in Section III present our main ideas. We show how a tailored coordinate change allows us to exploit the symmetry properties of the considered problem. We also discuss the steps necessary to implement our main idea using path primitives based on Linear Segments and Parabolic Blends (LSPB), i.e., piecewise polynomials of first and second degree. Section IV touches briefly upon the training problem. Section V presents the results of a numerical case study and our conclusions are in Section VI.

II. Preliminaries

We recall MPFC for a differential drive robot as presented in [15], which is itself based on [5], [16].

System Description: Figure 1 shows the state and control variables of a differential drive robot. The axes XY define the global frame, whereas the axes $\bar{X}\bar{Y}$ define the local frame attached to the vehicle. The robot's pose is $\xi = [q_x \; q_y \; \varphi]^\intercal$, with $q = [q_x \; q_y]^\intercal$ the robot's Cartesian position vector in frame XY, and φ its orientation. The robot dynamics are given by the rate of change of the pose in terms of the robot's forward speed s, and its angular velocity ω:

$$\dot{\xi} = f(\xi, u) = \begin{bmatrix} s\cos(\varphi) \\ s\sin(\varphi) \\ \omega \end{bmatrix}, \quad \xi(0) = \xi_0. \tag{1}$$

We have $\xi \in \mathcal{X} \subseteq \mathbb{R}^3$, and $u = [s \quad \omega]^\intercal \in \mathcal{PC}(\mathcal{U}) \subset \mathbb{R}^2$, where $\mathcal{PC}(\mathcal{U})$ means that the inputs are piecewise continuous and take values from the compact set $\mathcal{U}$.

State-Space Path Following: The goal of the path-following problem is to make system (1) follow a geometric reference without a priori specifying the timing *when to be where* on the path. The path is given by

$$\mathcal{P} = \{\xi \in \mathbb{R}^3 \mid \exists\, \theta \in \mathbb{R} \mapsto \xi = p(\theta)\}. \tag{2}$$

The variable $\theta(t) \in \mathbb{R}$ is called the path parameter and $p(\theta(t)) \in \mathbb{R}^3$ is an explicit parameterization of $\mathcal{P}$. The crucial idea of path following is that the time evolution $t \mapsto \theta$ is not explicitly given. Put differently, the control inputs $u \in \mathcal{PC}(\mathcal{U})$ and the timing $\theta : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ can be chosen such that the system (1) follows the path accurately.

Problem 1 (State-space path following):

1) Convergence to the path: the robot's state ξ converges to the path $\mathcal{P}$ such that $\lim_{t\to\infty} \|\xi(t) - p(\theta)\| = 0$.
2) Constraint satisfaction: the constraints on the states $\xi \in \mathcal{X}$ and inputs $u \in \mathcal{U}$ are satisfied at all times.
3) Velocity convergence: $\dot{\theta}$ converges to a predefined profile $v_r(t)$ such that $\lim_{t\to\infty} \|\dot{\theta}(t) - v_r(t)\| = 0$.

We note that extensions to path following in output spaces are given in [2]. Subsequently, we restrict our discussion to path parametrizations of the form

$$p(\theta) = [p_x(\theta) \;\; p_y(\theta) \;\; p_\varphi(\theta)]^\intercal, \tag{3}$$

with $p_\varphi(\theta) = \arctan\left(\frac{p'_y}{p'_x}\right)$, $p'_x = \frac{\partial p_x}{\partial \theta}$, $p'_y = \frac{\partial p_y}{\partial \theta}$, and $p_x(\theta)$, $p_y(\theta)$ at least twice continuously differentiable [5].

The conceptual idea of predictive path following is to consider the path parameter θ as a virtual state, which is controlled by the virtual input v. For the sake of simplicity, the dynamics of θ are chosen as

$$\dot{\theta} = v, \quad \theta(0) = \theta_0, \tag{4}$$

where $v \in \mathcal{PC}(\mathcal{V})$, $\mathcal{V} \doteq [0, \bar{v}]$, and $\bar{v} \in \mathbb{R}$. Model predictive path following is formulated using the augmented system

$$\dot{z} = f(z, w) = \begin{bmatrix} \dot{q}_x \\ \dot{q}_y \\ \dot{\varphi} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} s\cos(\varphi) \\ s\sin(\varphi) \\ \omega \\ v \end{bmatrix},$$

with the augmented state vector $z = [\xi^\intercal \; \theta]^\intercal = [q_x \; q_y \; \varphi \; \theta]^\intercal \in \mathcal{Z} = \mathcal{X} \times \mathbb{R}_0^+$ and the augmented input vector $w = [u^\intercal \; v]^\intercal = [s \; \omega \; v]^\intercal \in \mathcal{PC}(\mathcal{U} \times \mathcal{V}) \subset \mathbb{R}^3$.

Moreover, system (1) is *differentially flat*, and $[q_x \; q_y]^\intercal$ is one of its flat outputs [17]. Therefore there exists an input $u_r = [s_r \; \omega_r]^\intercal$ which guarantees that the path (3) is exactly followed by the system.

The vector u_r is used as a reference for the input vectors and can be built by observing that the first two equations of system (1) satisfy $s^2 = \dot{q}_x^2 + \dot{q}_y^2$, and thus $s_r(\theta, v) = v\sqrt{(p'_x)^2 + (p'_y)^2}$. Furthermore, from the last equation of system (1) we have $\omega = \dot{\varphi}$, which yields

$$\omega_r(\theta, v) = v\left((p'_x)^2 + (p'_y)^2\right)^{-1}\left(p'_x p''_y - p'_y p''_x\right), \tag{5}$$

with $p''_x = \frac{\partial^2 p_x}{\partial \theta^2}$, and $p''_y = \frac{\partial^2 p_y}{\partial \theta^2}$. See [5], [18] for details.

Model Predictive Path Following Control (MPFC): In this section we consider paths defined piecewise by a set of coefficient vectors $\varsigma_i \in \mathbb{R}^{n_\varsigma}$, $i = 1, \ldots, N_\varsigma$. For the sake of simplicity, we restrict ourselves to polynomial parametrizations [1]. To highlight the dependence of the path parametrization on the coefficients ς_i we write $p(\theta, \varsigma_i)$.

State-space MPFC considers the sampling period $\delta > 0$ and the prediction horizon $T = N\delta$, $N \in \mathbb{N}$ [16]. The extended state at the current sampling time $t_k = k\delta$ is $z_k = \begin{bmatrix}\xi(t_k) & \theta(t_k)\end{bmatrix}$ and the extended control input is $w = \begin{bmatrix}u^\intercal & v\end{bmatrix}^\intercal$. We have

$$\ell(z, w; \varsigma) = \left\| \begin{matrix} \xi - p(\theta; \varsigma) \\ \theta \end{matrix} \right\|_Q^2 + \left\| \begin{matrix} u - u_r(\theta, v; \varsigma) \\ v - v_r \end{matrix} \right\|_R^2, \tag{6}$$

with $Q = Q^\intercal \succeq 0$ and $R = R^\intercal \succ 0$, i.e., symmetric positive (semi)definite diagonal matrices as the stage cost. Given the extended state z_k and the path coefficients ς_i as parametric data, the Optimal Control Problem (OCP) to be solved repeatedly at each sampling instant t_k reads

$$\begin{aligned} \mathbf{w}^* \in \underset{w \in \mathcal{PC}(\mathcal{W})}{\arg\min} \quad & \int_0^T \ell(z(\tau), w(\tau); \varsigma)\mathrm{d}\tau \\ \text{subject to} \quad & \dot{z}(\tau) = f(z(\tau), w(\tau)), \quad z(0) = z_k, \\ & z(\tau) \in \mathcal{Z}, \; w(\tau) \in \mathcal{W}, \quad \varsigma = \varsigma_i. \end{aligned} \tag{7}$$

Although this OCP is formulated in continuous time, our MPFC *implementation* relies on a direct numerical solution method (e.g., CasADi combined with ipopt [19]) and thus an optimal discrete-time input sequence $\mathbf{w}^* = \{w_0^*, \ldots, w_{N-1}^*\} \in \mathcal{W}^N$ is computed at each time step. Typically, in MPC we only apply the first element $w = w_0^*$ of the sequence $\mathbf{w}^*$ to control the system. Conceptually, the MPFC feedback controller based on (7) can be expressed as the map $\mathbb{M} : \mathbb{R}^4 \times \mathbb{R}^{n_\varsigma} \to \mathcal{W}$

$$w = \begin{bmatrix}u^\intercal & v\end{bmatrix}^\intercal = \mathbb{M}(z_k, \varsigma_i). \tag{8}$$

Note that w entails the robot command u *and* the virtual input v which controls the evolution of the path parameter θ, cf. (4). Hence only u is applied to the robot. Also, note that the path coefficient vector ς is constant inside the OCP. This simplifies the training of the neural network. See [1] for an MPFC example where the path coefficients vary.

Deep Neural Networks: Eventually, we want to approximate the MPFC map (8) and hence we aim to avoid repeatedly solving the OCP online. Neural networks are excellent candidates for the underlying regression tasks due to the universal approximation theorem [20]. In the case of MPC, their online evaluation (i.e. after training) is computationally quite efficient, see, e.g., [9], [15]. We define the mapping $w^D = \mathbb{D}(z, \varsigma; \Theta)$, where $[z^\intercal \; \varsigma^\intercal]^\intercal$ is the input to the network, Θ is a set of N_Θ unknown parameters, which are determined during *training*. The training as such is based on data obtained from solving the OCP (7) for different initial conditions and different path data. Once the neural network $\mathbb{D}(z, \varsigma; \Theta)$ is trained, we use it to *infer* the current inputs

$$w^D = \mathbb{D}(z, \varsigma; \Theta) \approx \mathbb{M}(z, \varsigma)$$

instead of solving (7) at each sampling period.

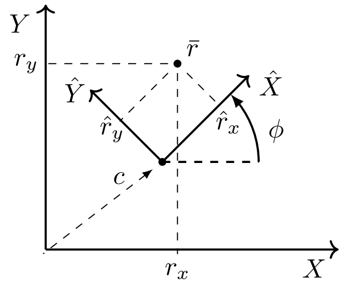

Fig. 2: Frames related through translation by c and rotation by ϕ: point $\bar{r}$ in XY: $[r_x\ r_y]^\intercal$, whereas in $\hat{X}\hat{Y}$: $[\hat{r}_x\ \hat{r}_y]^\intercal$.

III. Generalized DNN-based MPFC Approximation

This section presents our main contribution. Recall that in (8) we have introduced $\mathbb{M}$ which maps the initial condition z and the path coefficients ς to control actions. With slight abuse of notation, let us consider the path parametrization $p(\theta)$ from (2) as the second input argument of $\mathbb{M}$ from (8).

A. Main Idea

Consider two MPFC problems defined for different paths $\mathcal{P}$, parametrized by $p(\theta)$, and $\hat{\mathcal{P}}$, given by $\hat{p}(\theta)$. We denote the corresponding maps $\mathbb{M}(z, p(\theta))$ and $\mathbb{M}(\hat{z}, \hat{p}(\theta))$. Let the the parametrizations $p(\theta)$ and $\hat{p}(\theta)$ satisfy

$$\hat{p} = T_p(p, c, \phi) = \begin{bmatrix} R_\phi^{-1}(p_{xy} - c) \\ p_\varphi - \phi \end{bmatrix}, \tag{9a}$$

with $p_{xy} = [p_x\ p_y]^\intercal$. Put differently, $\hat{\mathcal{P}}$ is given in the $\hat{X}\hat{Y}$ coordinate frame, while $\mathcal{P}$ is given in the XY coordinate frame, cf. Fig. 2. Moreover, due to the kinematic nature of (1), the corresponding state vectors z and $\hat{z}$ satisfy

$$\hat{z} = T_z(z, c, \phi) = \begin{bmatrix} R_\phi^{-1}(q - c) \\ \varphi - \phi \\ \theta \end{bmatrix}. \tag{9b}$$

Note that $\hat{r} = R_\phi^{-1}(r - c)$, relates $r = [r_x\ r_y]^\intercal$, i.e., the Cartesian coordinates of point $\bar{r}$ with respect to the global frame XY, to $\hat{r} = [\hat{r}_x\ \hat{r}_y]^\intercal$, i.e, the coordinates of the same point $\bar{r}$ in frame $\hat{X}\hat{Y}$. The constant vector $c = [c_x\ c_y]^\intercal$ defines the origin of the frame $\hat{X}\hat{Y}$ with respect to XY, and the angle ϕ is the rotation of $\hat{X}\hat{Y}$ with respect to XY, cf. Fig. 2. Moreover, the rotation matrix $R_\phi \in \mathbb{R}^{2\times 2}$ is given by

$$R_\phi = R(\phi) = \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix}. \tag{9c}$$

Proposition 1 (Rotation and translation invariance): Consider two paths $\mathcal{P}$ and $\hat{\mathcal{P}}$, parametrized by $p(\theta)$ and $\hat{p}(\theta)$, respectively. Suppose that the corresponding MPFC problems are feasible with respect to the constraints of OCP (7). Let the corresponding MPFC-input maps be given by $\mathbb{M}(z, p(\theta))$ and $\mathbb{M}(\hat{z}, \hat{p}(\theta))$ and suppose that for all θ the parametrizations $p(\theta)$ and $\hat{p}(\theta)$ satisfy (9a) for some values of c and ϕ. Then under the change of coordinates of the path $p(\theta)$ (9a) and of the state z (9b) we have that $\mathbb{M}(z, p(\theta)) = \mathbb{M}(T_z(z, c, \phi), T_p(p, c, \phi)) = \mathbb{M}(\hat{z}, \hat{p}(\theta))$.

Proof: Due to space constraints, we only sketch the main steps of the proof. For both problems, the coordinate transformation (9) implies that the inputs $w = \mathbb{M}(T_z(z, \bar{p}, \phi))$ and $\hat{w} = \mathbb{M}(\hat{z}, \hat{p}(\theta))$, do not change and thus the input constraints are invariant under the considered transformation. Recall that the states of the kinematic robot model refer to the position and orientation of the robot. Thus, going from $\mathbb{M}(\hat{z}, \hat{p}(\theta))$ to $\mathbb{M}(z, p(\theta))$ the state constraints are changed from the box set $\mathcal{Z} = \{z \in \mathbb{R}^4 \mid \underline{z} \le z \le \overline{z}\}$, to another box-shaped set in rotated coordinates $\hat{z} \in \hat{\mathcal{Z}}$ due to the transformation (9b). Thus the state constraints are equivalent, i.e., $\hat{z} \in \hat{\mathcal{Z}} \iff z \in \mathcal{Z}$. To prove that $u = \hat{u}$, it remains to show that the cost function is also the same for both problems. Note that if the stage cost (6) for both problems is the same, then so are the cost functions. The speed reference v_r is a constant not affected by the transformations. From the rotation and translation, we have that $p_\varphi = \hat{p}_\varphi + \phi$, $p_{xy} = R_\phi \hat{p}_{xy} + c$ and we get $p_x = \hat{p}_x \cos\phi - \hat{p}_y \sin\phi + c_x$, $p_y = \hat{p}_x \sin\phi + \hat{p}_y \cos\phi + c_y$. It follows that $p'_x = \hat{p}'_x \cos\phi - \hat{p}'_y \sin\phi$, $p'_y = \hat{p}'_x \sin\phi + \hat{p}'_y \cos\phi$. Using the identity $1 = \cos^2\phi + \sin^2\phi$, it is easy to show that $(p'_x)^2 + (p'_y)^2 = (\hat{p}'_x)^2 + (\hat{p}'_y)^2$, and thus $s_r = \hat{s}_r$. Similarly, $p''_x = \hat{p}''_x \cos\phi - \hat{p}''_y \sin\phi$, $p''_y = \hat{p}''_x \sin\phi + \hat{p}''_y \cos\phi$. Moreover, we have that $\hat{p}'_x \hat{p}''_y - \hat{p}'_y \hat{p}''_x = p'_x p''_y - p'_y p''_x$, and thus $\hat{\omega}_r = \omega_r$ holds. Note that $Q = \mathrm{diag}(q_1, q_2, q_3, 0)$, i.e., the value of the path parameter θ is not penalized in the optimization problem. It remains to show that $\|\xi - p\|^2_{\bar{Q}} = \|\hat{\xi} - \hat{p}\|^2_{\bar{Q}}$, with $\hat{\xi} = \bar{R}_\phi^{-1}(\xi - \bar{\xi})$, $\bar{Q} = \mathrm{diag}(q_1, q_2, q_3)$, $\bar{R}_\phi = \left[\begin{smallmatrix} R_\phi & \mathbf{0} \\ \mathbf{0} & 1 \end{smallmatrix}\right]$, and $\bar{\xi} = [c_x\ c_y\ \phi]$. Consider $e = \xi - p$ and $\hat{e} = \hat{\xi} - \hat{p} = \bar{R}_\phi^{-1}(\xi - \bar{\xi}) - \hat{p}$, then $\bar{R}_\phi \hat{e} = \xi - (\bar{\xi} + \bar{R}_\phi \hat{p})$. Note that $\bar{\xi} + \bar{R}_\phi \hat{p} = p$. That is, $\bar{R}_\phi \hat{e} = e$. Finally, because a rotation matrix does not change the magnitude of a vector (i.e., $\det(\bar{R}_\phi) = 1$), then $\|\bar{R}_\phi \hat{e}\|^2_{\bar{Q}} = \|\hat{e}\|^2_{\bar{Q}}$. ■

Proposition 1 implies that for any path $p(\theta)$ and state z, we can solve $w = \mathbb{M}(\hat{z}, \hat{p}(\theta))$ instead of $w = \mathbb{M}(z, p(\theta))$, for any combination of ϕ and c, at the expense of applying the coordinate transformation (9) to the system state and the path. From the optimization point of view, this may not be of much advantage. However, we can efficiently learn a DNN approximation of the *base* MPC problem $w = \mathbb{M}(\hat{z}, \hat{p}(\theta))$ for given paths (see Fig. 3) and then apply the coordinate transformation (9) only to adapt to different paths without re-optimization. Specifically, we detail this idea using *path primitives* $\hat{p}(\theta; \hat{\varsigma})$, with $\hat{\varsigma} = \eta \in \mathbb{R}_0^+$, of the form

$$\hat{p}_y = \eta \hat{p}_x^2 = \bar{\eta}\theta^2, \quad \hat{p}_x = g(\eta)\theta, \quad \bar{\eta} = \eta\,(g(\eta))^2, \tag{10}$$

where $g : \mathbb{R}_0^+ \to \mathbb{R}^+$ is a scaling factor dependent on η. The notion of *path primitive* is in reference to the concept of motion primitives [21] as here we learn a controller on path primitives and adapt to specific paths online.

B. Path Primitives and Path Planning

To be able to follow the path accurately, the following conditions need to be met so that the tuning of (7) (i.e., the selection of matrices Q and R) works for all values of η (i.e., different curvatures). First, we consider how to

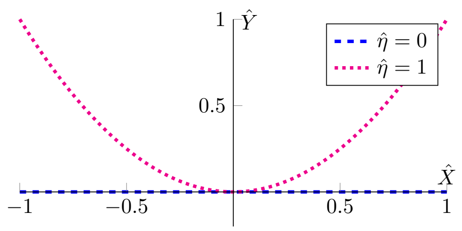

Fig. 3: Two path primitives used to learn the MPFC input by the NN: a line ($\breve{\eta} = 0$), and a quadratic ($\breve{\eta} = 1$).

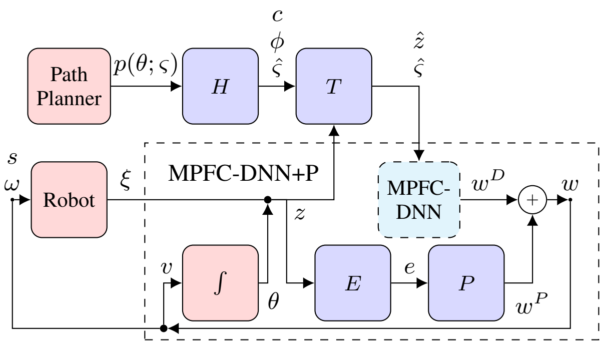

Fig. 4: Block diagram of the proposed controller.

compute the scaling factor $g(\eta) = \Delta\hat{p}_x\Delta\theta^{-1}$ for a given η. Let $\Gamma(\eta)$ denote the length of a path segment for a unit change in the $\hat{X}$ axis, i.e., $\Delta\hat{p}_x = 1$. For $\eta \geq 0$, the length of $\hat{p}_y = \eta\hat{p}_x^2$ from 0 to $\Delta\hat{p}_x$ is given by $\Gamma(\eta) = \int_0^1 \sqrt{1 + (2\eta x)^2}\mathrm{d}x$. For a line $\hat{p}_y = 0$, we get that $\Gamma(0) = \Delta\hat{p}_x = 1$. For $\eta > 0$, integrating the above expression we get: $\Gamma(\eta) = \frac{1}{2}\left(\sqrt{1 + (2\eta)^2} + \frac{\operatorname{arcsinh}(2\eta)}{2\eta}\right)$. The OCP depends on the parameter v_r. We aim at having the robot follow the path at its maximum forward speed $s_{\max}$ whenever the path error remains small. In such cases, $v = v_r$, $\Delta\theta = v_r\Delta t$, and $\Gamma(\eta) = s_{\max}\Delta t$. With $v_r = 1$, the two previous expressions yield $\Gamma(\eta) = s_{\max}\Delta\theta$, and $g(\eta) = \frac{\Delta x}{\Delta\theta} = \frac{s_{\max}}{\Gamma(\eta)}$.

For example, for a line, we have $\Gamma(0) = 1$ and thus $g(0) = s_{\max}$. The path parameterization is then $\hat{p}_y = 0$; $\hat{p}_x = s_{\max}\theta$.

In applications, a planning algorithm provides a path consisting of n_ς polynomials of the form

$$p_{x,i}(\theta) = a_{x,i}\theta^2 + b_{x,i}\theta + c_{x,i}, \tag{11a}$$
$$p_{y,i}(\theta) = a_{y,i}\theta^2 + b_{y,i}\theta + c_{y,i}. \tag{11b}$$

For each segment i, we have $\underline{\theta}_i \leq \theta \leq \overline{\theta}_i$. To avoid differentiability issues at the intersection of paths, two adjacent polynomials satisfy

$$p_{x,i}(\overline{\theta}_i) = p_{x,i+1}(\underline{\theta}_{i+1}), \quad p_{y,i}(\overline{\theta}_i) = p_{y,i+1}(\underline{\theta}_{i+1}),$$
$$p'_{x,i}(\overline{\theta}_i) = p'_{x,i+1}(\underline{\theta}_{i+1}), \quad p'_{y,i}(\overline{\theta}_i) = p'_{y,i+1}(\underline{\theta}_{i+1}),$$

i.e., two polynomials connect at the same coordinates, and the tangents are equal.

The key parameters to determine from (11) are η_i, c_i, and ϕ_i, which are given by

$$c_i = [c_{x,i} \quad c_{y,i}], \quad \phi_i = \arctan_2(b_{y,i}, b_{x,i}), \tag{12a}$$
$$\hat{\varsigma}_i = \eta_i = a_{y,i}b_{x,i}^{-2}\cos\phi_i. \tag{12b}$$

Remark 1 (Negative curvature): Note that we train our network using only positive curvatures $\breve{\eta}_k \geq 0$. Thus, if a parabolic segment of the path has $\eta_i \geq 0$, w^D does not need to be transformed and can be used directly in the control input u applied to the robot. That is, using (9) and (12), we can compute $\hat{z} = T_z(z, c_i, \phi_i)$, and consequently $w^D = \mathbb{D}(\hat{z}, \eta_i; \Theta)$.

However, if $\eta_i < 0$, we need to further transform the network input and the inferred output. If $[\hat{z}^\intercal, \eta_i]^\intercal = [\hat{q}_x \ \hat{q}_y \ \hat{\varphi} \ \theta \ \eta_i]^\intercal$ is the input to the network resulting from the transformation for $\eta_i < 0$, the input to the DNN trained using only $\breve{\eta}_k \geq 0$ is $[\hat{q}_x \ -\hat{q}_y \ -\hat{\phi} \ \theta \ -\eta_i]^\intercal$. From the DNN output $w^D = [s \ \omega \ v]^\intercal$, we compute the control input applied to the robot as $u = [s \ -\omega]^\intercal$.

The proposed control approach is summarized in Fig. 4. A planner generates a piecewise path $p(\theta;\varsigma)$ in the global frame XY. The rotation ϕ, translation c, and path description $\hat{\varsigma}$ in frame $\hat{X}\hat{Y}$ of the current path primitive are computed using (12). The transformation T maps z to $\hat{z}$ via (9). Inside the dashed block (MPFC-DNN) is the MPFC controller approximated by a DNN. This block takes the path primitive information $\hat{z}, \hat{\varsigma}$ as input, and outputs the augmented control input w^D. Block E computes the path deviation e used by the P controllers, cf. [15]. The vector w^P is added to w^D to compute the input w. The forward velocity s and the angular velocity ω are applied to the robot, whereas the path velocity v is integrated to compute θ.

IV. Training Implementation

To generate a training set $\mathcal{T}$ that is not too large, we use a *corridor*, as presented in [15]. The idea is as follows: for different values of $\breve{\eta}_k \geq 0$, $k = 1, \ldots, N_\eta$, and at specific points $\theta_{k,i}$, $i = 1, \ldots, N_k$ in path $\hat{p}(\theta, \breve{\eta}_k)$, we select different poses $\xi_{k,i,j}$, $j = 1, \ldots, N_c$ that are inside an orthogonal cuboid centered at $\theta_{k,i}$. Fig. 5 illustrates a corridor for $\breve{\eta} = 1$: At specific base points on the path $p(\theta_{k,i}, \breve{\eta}_k)$, several points $q_{k,i,j}$ (representing the robot's relative Cartesian position to the path) are computed inside a box (a cuboid, when the robot's orientation φ is also considered) whose axes are tangential and normal to $p(\theta_{k,i}, \breve{\eta}_k)$. In Fig. 5, each axis is divided into three, given 9 different poses inside each box.

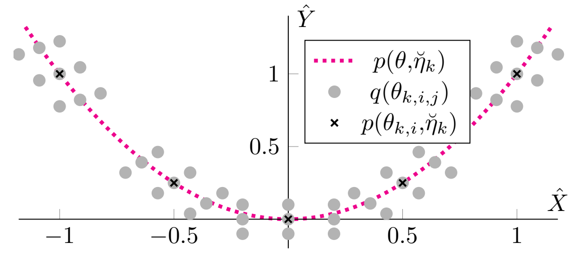

Fig. 5: Simplified 2-dimensional view of a corridor used to build the training set for $\breve{\eta}_k = 1$.

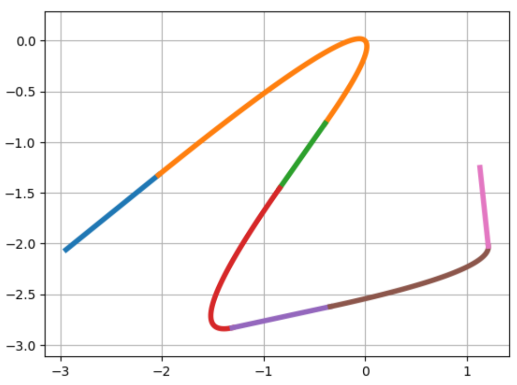

Fig. 6: Reference path using LSPB. It consist of 4 linear and 3 parabolic segments; start left side, end right side.

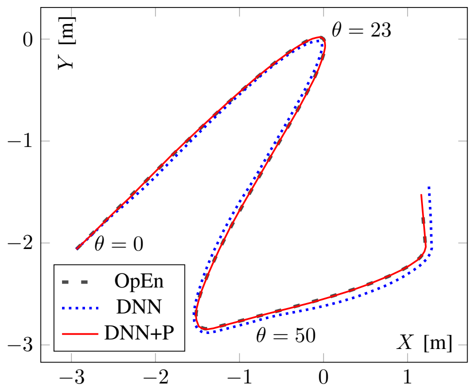

Fig. 7: Path followed by a simulated robot using different MPFC implementations.

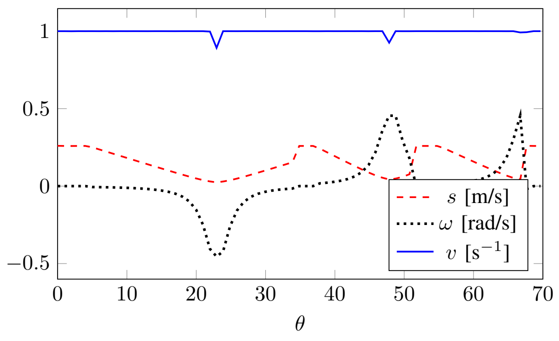

Fig. 8: Control inputs applied to the robot computed by MPFC-OpEn while following the reference path in Fig. 7.

For our example, we build $N_\eta = 11$ paths, with $\breve{\eta}_0 = 0, \breve{\eta}_1 = 1, \dots, \breve{\eta}_{10} = 10$. We split each path into equidistant segments between $-g(\breve{\eta}_k)\underline{d}_k \leq \theta \leq g(\breve{\eta}_k)\overline{d}_k$, i.e., the number of points N_k depends on the length of the path. The path parameter limit factors $\underline{d}_k, \overline{d}_k$ are application-dependent. Here we use $\underline{d}_k = \overline{d}_k = 1$ for all values of k. Each axis of a cuboid is split into 10, 10, and 10 equidistant points ($N_c = 1000$). The set $\mathcal{Z}_k$ contains all augmented states $\hat{z} = [\xi_{k,i,j}^\top \;\; \theta_{k,i}]^\top$ related to a specific path $\hat{p}(\theta, \breve{\eta}_k)$.

To solve the OCP (7), and consequently find $w = \mathbb{M}(\hat{z}, \breve{\eta}_k)$, $\forall\ \hat{z} \in \mathcal{Z}_k$, $k = 1, \dots, N_\eta$, according to (8), we use the Optimization Engine (OpEn) [22]. The training set $\mathcal{T}$ consists of $N_c N_k N_\eta$ vectors $[\hat{z}^\top \;\; \mathbb{M}(\hat{z}, \hat{\eta}_k)^\top]^\top$ for all $z \in \mathcal{Z}_k$, $k = 1, \dots, N_\eta$. The discretization interval and the considered horizon length in (7) are $\delta = 0.01$ s and $T = 0.6$ s, respectively. We observe that for the considered LSPB setup, networks of around 10.000 parameters perform well. The training is done using Keras/TensorFlow [23], [24].

V. Results

Our reference path is shown in Fig. 6. It has 7 segments, described by (11). The path is constructed to highlight some characteristics of the presented approach. From (12), the linear segments all have $\eta_1 = \eta_3 = \eta_5 = \eta_7 = 0$, the values η_i for the parabolic blends are $\eta_2 = -9.5, \eta_4 = 5.5, \eta_6 = 6.2$. Note that curvature values of the parabolic blends are not used to build the training set (i.e., $\eta_i \neq \breve{\eta}_k$; $i = 2, 4, 6$; $k = 1, \dots, N_\eta$), meaning that the network generalizes well. Also, note that $\eta_2 = -9.5$ has a negative curvature, which requires a sign inversion, cf. Remark 1.

We utilize OpEn, the same solver employed during training, as our reference implementation, denoted MPFC-OpEn, to solve (7).

A. Discussion

In Fig. 7 a comparison of the path followed by the simulated robot in the Cartesian XY plane using different implementations: : OpEn (visually indistinguishable from the reference path $p(\theta; \varsigma)$), the approximation using a deep neural network using base paths $\hat{p}(\theta; \eta)$ (DNN, 32-bit float), and the same network plus a P controller (DNN+P) In Fig. 8 show the inputs computed by MPFC-OpEn. Recall that the path speed reference is $v_r = 1$, and notice in Fig. 8 how the path speed equals $v = 1$ most of the time, except near tight curvatures (e.g., $\theta \approx 23$). Also, $s = s_{\max} = 0.26$ mostly around the linear segments (e.g., $\theta \approx 35$), except around the curves. At other parts of the paths, the MPFC algorithm reduces the speed s to decrease path deviation.

The absolute Cartesian position error is shown in Fig. 9. The OpEn implementation is the most accurate, DNN is two orders of magnitude worse than the OpEn implementation on average. To improve the performance of the DNN, we add an online compensation based on two proportional (P) controllers, as presented in [15]. The input to the P-controllers are the Cartesian position error of the robot in the directions normal and tangential to the current path reference $p(\theta, \eta_i)$. The implementation of a DNN with a P-control compensation is denoted as DNN+P, and reduces the DNN worst-case error by an order of magnitude. Finally, Table I compares the computation time of MPFC based on OpEn against MPFC based on DNN+P. Note that the proposed DNN approach is two orders of magnitude faster on a PC.

B. Application on Low-Cost Embedded Hardware?

The inference of a relatively small network like the one used here (around 10.000 parameters), is executed much faster than solving OCP (7). Also, the resulting network of the approach presented here is only twice as large as

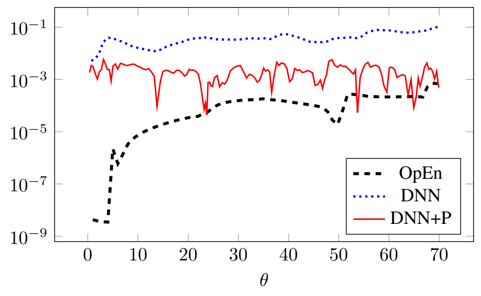

Fig. 9: Cartesian position error with respect to the reference path for all implementations.

Implementation	Mean [s]	Std. [s]	Worst [s]
OpEn (PC)	1.6E-3	3.6E-4	5.6E-3
DNN+P (PC)	3.7E-5	6.4E-6	1.5E-4

TABLE I: Mean, standard deviation (Std.), worst-case execution (Worst) time in seconds for 10000 steps close to the path. The table is split into MPFC implementations on a personal computer.

the network learned for a single path in [15]. In this previous work, we deployed a quantized neural network on a MicroController Unit (MCU) to perform hardware-in-the-loop simulations of the path-following problem, with inference time of around 400μs. Thus, as this approach relies on a similar neural network, and at each sampling period performs only the additional operations (9) and (12), it is likely feasible to deploy the DNN proposed here on an MCU using fast sampling rates (i.e., sampling period $\delta \leq 10$ ms), in particular if the network is quantized as in [15].

VI. Conclusions

This paper presented a novel approach to approximate a model predictive path-following control problem using neural networks. Specifically, we have shown that the problem structure can be exploited to learn one neural network approximation based on path primitives. The key idea is to exploit the underlying symmetry properties of the robot. Under these conditions, a neural network must only learn the MPFC mapping for a small set of path primitives, to be able to follow a path derived from e.g., a path planning algorithm. We illustrated our approach using a piecewise polynomial path constructed using linear segments and parabolic blends.

Based on simulations on a laptop computer, we show that this approach requires only a fraction of the memory and runs on average two orders of magnitude faster than an MPFC implementation based on online optimization. This opens a new avenue towards the feasibility of real-time implementation of path-following control on microcontrollers and other embedded platforms.

References

[1] T. Faulwasser, T. Weber, P. Zometa, and R. Findeisen, “Implementation of nonlinear model predictive path-following control for an industrial robot,” *IEEE Transactions on Control Systems Technology*, vol. 25, no. 4, pp. 1505–1511, 2016.

[2] T. Faulwasser and R. Findeisen, “Nonlinear model predictive control for constrained output path following,” *IEEE Transactions on Automatic Control*, vol. 61, no. 4, pp. 1026–1039, 2015.

[3] A. Völz and K. Graichen, “A predictive path-following controller for continuous replanning with dynamic roadmaps,” *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3963–3970, 2019.

[4] M. Böck and A. Kugi, “Real-time nonlinear model predictive path-following control of a laboratory tower crane,” *IEEE Transactions on Control Systems Technology*, vol. 22, no. 4, pp. 1461–1473, 2013.

[5] M. W. Mehrez, K. Worthmann, G. K. Mann, R. G. Gosine, and T. Faulwasser, “Predictive path following of mobile robots without terminal stabilizing constraints,” *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 9852–9857, 2017.

[6] D. Reinhardt, S. Gros, and T. A. Johansen, “Fixed-wing uav path-following control via nmpc on the lowest level,” in *2023 IEEE Conference on Control Technology and Applications (CCTA)*. IEEE, 2023, pp. 451–458.

[7] D. Lam, C. Manzie, and M. C. Good, “Model predictive contouring control for biaxial systems,” *IEEE Transactions on Control Systems Technology*, vol. 21, no. 2, pp. 552–559, 2012.

[8] T. Parisini and R. Zoppoli, “A receding-horizon regulator for nonlinear systems and a neural approximation,” *Automatica*, vol. 31, no. 10, pp. 1443–1451, 1995.

[9] S. Lucia and B. Karg, “A deep learning-based approach to robust nonlinear model predictive control,” *IFAC-PapersOnLine*, vol. 51, no. 20, pp. 511–516, 2018, 6th IFAC Conference on Nonlinear Model Predictive Control NMPC 2018.

[10] S. S. P. Kumar, A. Tulsyan, B. Gopaluni, and P. Loewen, “A deep learning architecture for predictive control,” *IFAC-PapersOnLine*, vol. 51, no. 18, pp. 512–517, 2018.

[11] N. Lanzetti, Y. Z. Lian, A. Cortinovis, L. Dominguez, M. Mercangöz, and C. Jones, “Recurrent neural network based mpc for process industries,” in *2019 18th European Control Conference (ECC)*. IEEE, 2019, pp. 1005–1010.

[12] E. T. Maddalena, C. d. S. Moraes, G. Waltrich, and C. N. Jones, “A neural network architecture to learn explicit mpc controllers from data,” *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 11 362–11 367, 2020.

[13] G. Bai, Y. Meng, L. Liu, Q. Gu, J. Huang, G. Liang, G. Wang, L. Liu, X. Chang, and X. Gan, “Path tracking for car-like robots based on neural networks with nmpc as learning samples,” *Electronics*, vol. 11, no. 24, p. 4232, 2022.

[14] P. Saha, L. Guerrero-Bonilla, M. Egerstedt, and S. Mukhopadhyay, “Learning deep neural network controller for path following of unicycle robots,” *IEEE Robotics and Automation Letters*, vol. 8, no. 1, pp. 248–255, 2022.

[15] P. Zometa and T. Faulwasser, “Quantized deep path-following control on a microcontroller,” in *2023 European Control Conference (ECC)*, 2023, pp. 1–6.

[16] T. Faulwasser and R. Findeisen, “Nonlinear model predictive path-following control,” in *Nonlinear model predictive control*. Springer, 2009, pp. 335–343.

[17] P. Martin, R. Murray, and P. Rouchon, “Flat systems,” in *Proc. of the 4th European Control Conf*, 1997, pp. 211–264.

[18] T. Faulwasser, V. Hagenmeyer, and R. Findeisen, “Optimal exact path-following for constrained differentially flat systems,” in *Proc. of 18th IFAC World Congress, Milano, Italy*, 2011, pp. 9875–9880.

[19] J. Andersson, J. Gillis, G. Horn, J. 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] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, 1989.

[21] E. Frazzoli, M. A. Dahleh, and E. Feron, “Maneuver-based motion planning for nonlinear systems with symmetries,” *IEEE transactions on robotics*, vol. 21, no. 6, pp. 1077–1091, 2005.

[22] P. Sopasakis, E. Fresk, and P. Patrinos, “OpEn: Code generation for embedded nonconvex optimization,” in *IFAC World Congress*, Berlin, Germany, 2020.

[23] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, “TensorFlow: a system for Large-Scale machine learning,” in *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, 2016, pp. 265–283.

[24] F. Chollet *et al.*, “Keras,” https://keras.io, 2015.